



运动控制器 编程手册

GUS Controller 系列

R1.3



2020.10

www.googoltech.com.cn

© 2020 固高科技 版权所有

版权申明

固高科技（深圳）有限公司

保留所有权力

固高科技（深圳）有限公司（以下简称固高科技）保留在不事先通知的情况下，修改本手册中的产品和产品规格等文件的权力。

固高科技不承担由于使用本手册或本产品不当，所造成直接的、间接的、特殊的、附带的或相应产生的损失或责任。

固高科技具有本产品及其软件的专利权、版权和其它知识产权。未经授权，不得直接或间接地复制、制造、加工、使用本产品及其相关部分。



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：googol@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

电话：+(852) 2358-1033

传真：+(852) 2719-8399

电子邮件：sales@googoltech.com

info@googoltech.com

网址：<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

电话：+886-4-23588245

传真：+886-4-23586495

电子邮件：twinfo@googoltech.com

文档版本

版本号	修订日期
1.0	2014年07月31日
1.1	2015年11月04日
1.2	2015年12月31日
1.3	2020年10月28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 GUS 系列运动控制器的控制功能，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 GUS 系列运动控制器的控制功能及编程实现。

相关文件

关于 GUS 系列运动控制器调试和安装，请参见随产品配套的《GUS-400-TG02 系列控制器用户手册》、《GUS-600-TG02 系列控制器用户手册》《GUS 激光控制说明书》。



注意

相关手册及控制器适用文档列表见于光盘的 manual 目录下。

目录

版权申明	1
联系我们	1
文档版本	2
前言	3
目录	4
第 1 章 指令列表	7
第 2 章 运动控制器函数库的使用	13
2.1 Windows 系统下动态链接库的使用	13
2.1.1 Visual C++ 6.0 中的使用	13
2.1.2 Visual Basic 6.0 中的使用	13
2.1.3 Delphi 中的使用	13
2.2 WinCE 下自启动功能的使用	14
第 3 章 命令返回值及其意义	15
3.1 指令返回值	15
第 4 章 系统配置	16
4.1 系统配置基本概念	16
4.1.1 硬件资源	16
4.1.2 软件资源	16
4.1.3 资源组合	16
4.2 系统配置工具	18
4.2.1 配置 axis	19
4.2.2 配置 step	21
4.2.3 配置 dac	22
4.2.4 配置 encoder	23
4.2.5 配置 control	24
4.2.6 配置 profile	25
4.2.7 配置 di	26
4.2.8 配置 do	27
4.3 配置文件生成和下载	27
4.4 配置信息修改指令	28
4.4.1 指令列表	29
4.5 重点说明	29
第 5 章 运动状态检测	31
5.1 指令列表	31
5.2 重点说明	31
5.2.1 轴状态定义	31
5.2.2 轴(axis)状态读取的相关指令	32
第 6 章 运动模式	33
6.1 点位运动	33
6.1.1 指令列表	33
6.1.2 重点说明	33
6.1.3 例程	34
6.2 Jog 模式	36

6.2.1 指令列表	36
6.2.2 重点说明	36
6.2.3 例程	37
6.3 PT 模式.....	40
6.3.1 指令列表	40
6.3.2 重点说明	40
6.3.3 例程	42
6.4 电子齿轮模式	48
6.4.1 指令列表	48
6.4.2 重点说明	48
6.4.3 例程	49
6.5 Follow 模式	51
6.5.1 指令列表	51
6.5.2 重点说明	51
6.5.3 例程	53
6.6 插补运动模式	63
6.6.1 指令列表	63
6.6.2 重点说明	64
6.7 PVT 模式.....	79
6.7.1 指令列表	79
6.7.2 重点说明	80
6.7.3 例程	87
第 7 章 访问硬件资源.....	102
7.1 访问数字 IO	102
7.1.1 指令列表	102
7.1.2 重点说明	102
7.1.3 例程	102
7.2 访问编码器	103
7.2.1 指令列表	103
7.2.2 例程	103
7.3 访问 DAC.....	104
7.4 访问模拟量输入	104
第 8 章 高速硬件捕获.....	105
8.1 Home/Index 硬件捕获	105
8.1.1 指令列表	105
8.1.2 重点说明	105
8.1.3 例程	105
8.2 Home 回原点	107
8.2.1 重点说明	107
8.2.2 例程	108
8.3 Home+Index 回原点	111
8.3.1 重点说明	111
8.3.2 例程	112
第 9 章 安全机制.....	116
9.1 限位	116
9.1.1 指令列表	116

9.1.2 重点说明	116
9.1.3 例程	117
9.2 报警	118
9.3 平滑停止和急停	118
9.4 跟随误差极限	119
第 10 章 运动程序	120
10.1 简介	120
10.2 编写运动程序	120
10.2.1 指令列表	120
10.2.2 重点说明	121
10.2.3 例程	121
10.3 语言元素	135
10.3.1 数据类型	135
10.3.2 常量	136
10.3.3 变量	136
10.3.4 数组	136
10.3.5 函数	136
10.3.6 数据类型转换	136
10.4 运算指令	136
10.4.1 算数运算	136
10.4.2 逻辑运算	136
10.4.3 关系运算	136
10.4.4 位运算	137
10.5 流程控制	137
第 11 章 其它指令	138
11.1 打开/关闭运动控制器	138
11.2 读取固件版本号	138
11.3 读取系统时钟	139
11.4 打开/关闭电机使能信号	139
11.5 维护位置值	139
11.6 电机到位检测	140
11.7 设置 PID 参数	145
11.8 反向间隙补偿	145
11.9 自动回原点功能	146
11.9.1 指令列表	146
11.9.2 重点说明	146
11.10 控制器断电保护	148
11.10.1 应用原理	148
11.10.2 断电保护程序	148
第 12 章 指令详细说明	151
第 13 章 索引	231
13.1 指令索引	231
13.2 例程索引	235
13.3 图片索引	235
13.4 表格索引	237

第1章 指令列表



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“第12章 指令详细说明”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

表 1-1 GUS 系列控制器指令列表

第4章 系统配置		页码
4.3 配置文件生成和下载		27
GT_LoadConfig	加载运动控制器系统配置文件	199
4.4 配置信息修改指令		28
GT_AlarmOff	控制轴驱动报警信号无效	151
GT_AlarmOn	控制轴驱动报警信号有效	151
GT_LmtsOn	控制轴限位信号有效	195
GT_LmtsOff	控制轴限位信号无效	195
GT_ProfileScale	设置控制轴的规划器当量变换值	202
GT_EncScale	设置控制轴的编码器当量变换值	165
GT_StepDir	将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”	227
GT_StepPulse	将脉冲输出通道的脉冲输出模式设置为“CW/CCW”	228
GT_SetMtrBias	设置模拟量输出通道的零漂电压补偿值	220
GT_GetMtrBias	读取模拟量输出通道的零漂电压补偿值	183
GT_SetMtrLmt	设置模拟量输出通道的输出电压饱和极限值	220
GT_GetMtrLmt	读取模拟量输出通道的输出电压饱和极限值	184
GT_EncSns	设置编码器的计数方向	165
GT_EncOn	设置为“外部编码器”计数方式	165
GT_EncOff	设置为“脉冲计数器”计数方式	164
GT_SetPosErr	设置跟随误差极限值	222
GT_GetPosErr	读取跟随误差极限值	185
GT_SetStopDec	设置平滑停止减速度和急停减速度	224
GT_GetStopDec	读取平滑停止减速度和急停减速度	189
GT_LmtSns	设置运动控制器各轴限位开关触发电平	194
GT_CtrlMode	设置控制轴为模拟量输出或脉冲输出	164
GT_SetStopIo	设置平滑停止和紧急停止数字量输入的信息	225
GT_GpiSns	设置运动控制器数字量输入的电平逻辑	192
GT_SetAdcFilter	设置模拟量输入的滤波器时间参数	210
第5章 运动状态检测		31
GT_GetSts	读取轴状态	189
GT_ClrSts	清除驱动器报警标志、跟随误差超限标志、限位触发标志 <ul style="list-style-type: none"> ✓ 只有当驱动器没有报警时才能清除轴状态字的报警标志 ✓ 只有当跟随误差正常以后，才能清除跟随误差超限标志 ✓ 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 	161

第 1 章 指令列表

GT_GetPrfMode	读取轴运动模式	185
GT_GetPrfPos	读取规划位置	186
GT_GetPrfVel	读取规划速度	186
GT_GetPrfAcc	读取规划加速度	185
GT_GetAxisPrfPos	读取轴(axis)的规划位置值	172
GT_GetAxisPrfVel	读取轴(axis)的规划速度值	173
GT_GetAxisPrfAcc	读取轴(axis)的规划加速度值	172
GT_GetAxisEncPos	读取轴(axis)的编码器位置值	170
GT_GetAxisEncVel	读取轴(axis)的编码器速度值	171
GT_GetAxisEncAcc	读取轴(axis)的编码器加速度值	171
GT_GetAxisError	读取轴(axis)的规划位置值和编码器位置值的差值	171
GT_Stop	停止一个或多个轴的规划运动，停止坐标系运动	228
第 6 章 运动模式		33
6.1 点位运动		33
GT_PrftTrap	设置指定轴为点位运动模式	202
GT_SetTrapPrm	设置点位模式运动参数	226
GT_GetTrapPrm	读取点位模式运动参数	190
GT_SetPos	设置目标位置	222
GT_GetPos	读取目标位置	184
GT_SetVel	设置目标速度，单位是“脉冲/毫秒”	227
GT_GetVel	读取目标速度，单位是“脉冲/毫秒”	191
GT_Update	启动点位运动	230
6.2 Jog 模式		36
GT_PrftJog	设置指定轴为 Jog 模式	201
GT_SetJogPrm	设置 Jog 运动参数	219
GT_GetJogPrm	读取 Jog 运动参数	183
GT_SetVel	设置目标速度，单位是“脉冲/毫秒”	227
GT_GetVel	读取目标速度，单位是“脉冲/毫秒”	191
GT_Update	启动 Jog 模式运动	230
6.3 PT 模式		40
GT_PrftPt	设置指定轴为 PT 模式	201
GT_PtSpace	查询 PT 指定 FIFO 的剩余空间	203
GT_PtData	向 PT 指定 FIFO 增加数据	203
GT_PtClear	清除 PT 指定 FIFO 中的数据 运动状态下该指令无效 动态模式下该指令无效	202
GT_SetPtLoop	设置 PT 模式循环执行的次数 动态模式下该指令无效	223
GT_GetPtLoop	查询 PT 模式循环执行的次数 动态模式下该指令无效	187
GT_PtStart	启动 PT 模式运动	204
GT_SetPtMemory	设置 PT 运动模式的缓存区大小	223
GT_GetPtMemory	读取 PT 运动模式的缓存区大小	187
6.4 电子齿轮模式		48

第 1 章 指令列表

GT_PrffGear	设置指定轴为电子齿轮模式	200
GT_SetGearMaster	设置跟随主轴	218
GT_GetGearMaster	读取跟随主轴	182
GT_SetGearRatio	设置电子齿轮比	219
GT_GetGearRatio	读取电子齿轮比	182
GT_GearStart	启动电子齿轮	168
6.5 Follow 模式		51
GT_PrffFollow	设置指定轴为 Follow 模式	200
GT_SetFollowMaster	设置跟随主轴	217
GT_GetFollowMaster	读取跟随主轴	181
GT_SetFollowLoop	设置循环次数	217
GT_GetFollowLoop	读取循环次数	180
GT_SetFollowEvent	设置 Follow 模式启动跟随条件	216
GT_GetFollowEvent	读取 Follow 模式启动跟随条件	180
GT_FollowSpace	查询 Follow 指定 FIFO 的剩余空间	167
GT_FollowData	向 Follow 指定 FIFO 增加数据	166
GT_FollowClear	清除 Follow 指定 FIFO 中的数据 运动状态下该指令无效	166
GT_FollowStart	启动 Follow 模式运动	167
GT_FollowSwitch	切换 Follow 所使用的 FIFO	168
GT_SetFollowMemory	设置 Follow 运动模式的缓存区大小	218
GT_GetFollowMemory	读取 Follow 运动模式的缓存区大小	181
6.6 插补运动模式		63
GT_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系	212
GT_GetCrdPrm	查询坐标系参数	176
GT_CrdData	向插补缓存区增加插补数据	162
GT_LnXY	缓存区指令，两维直线插补	196
GT_LnXYZ	缓存区指令，三维直线插补	197
GT_LnXYZA	缓存区指令，四维直线插补	197
SGT_LnXYG0	缓存区指令，两维直线插补(终点速度始终为 0)	196
GT_LnXYZG0	缓存区指令，三维直线插补(终点速度始终为 0)	198
GT_LnXYZAG0	缓存区指令，四维直线插补(终点速度始终为 0)	198
GT_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)	152
GT_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)	151
GT_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)	153
GT_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)	153
GT_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)	155
GT_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)	154
GT_BufIO	缓存区指令，缓存区内数字量 IO 输出设置指令	158
GT_BufDelay	缓存区指令，缓存区内延时设置指令	157
GT_BufDA	缓存区指令，缓存区内输出 DA 值	156
GT_BufLmtsOn	缓存区指令，缓存区内有效限位开关	159
GT_BufLmtsOff	缓存区指令，缓存区内无效限位开关	158
GT_BufSetStopIo	缓存区指令，缓存区内设置 axis 的停止 IO 信息	160

第 1 章 指令列表

GT_BufMove	缓存区指令，实现刀向跟随功能，启动某个轴点位运动	159
GT_BufGear	缓存区指令，实现刀向跟随功能，启动某个轴跟随运动	157
GT_CrdSpace	查询插补缓存区剩余空间	162
GT_CrdClear	清除插补缓存区内的插补数据	162
GT_CrdStart	启动插补运动	163
GT_CrdStatus	查询插补运动坐标系状态	163
GT_SetUserSegNum	缓存区指令，设置自定义插补段段号	226
GT_GetUserSegNum	读取自定义插补段段号	190
GT_GetRemainderSegNum	读取未完成的插补段段数	188
GT_SetOverride	设置插补运动目标合成速度倍率	221
GT_SetCrdStopDec	设置插补运动平滑停止、急停合成加速度	213
GT_GetCrdStopDec	查询插补运动平滑停止、急停合成加速度	176
GT_GetCrdPos	查询该坐标系的当前坐标位置值	175
GT_GetCrdVel	查询该坐标系的合成速度值	176
GT_InitLookAhead	初始化插补前瞻缓存区	194
6.7 PVT 模式		79
GT_PrnPvt	设置指定轴为 PVT 模式	201
GT_SetPvtLoop	设置循环次数	224
GT_GetPvtLoop	查询循环次数	188
GT_PvtTable	向指定数据表传送数据，采用 PVT 描述方式	206
GT_PvtTableComplete	向指定数据表传送数据，采用 Complete 描述方式	207
GT_PvtTablePercent	向指定数据表传送数据，采用 Percent 描述方式	208
GT_PvtPercentCalculate	计算 Percent 描述方式下各数据点的速度	205
GT_PvtTableContinuous	向指定数据表传送数据，采用 Continuous 描述方式	207
GT_PvtContinuousCalculate	计算 Continuous 描述方式下各数据点的时间	205
GT_PvtTableSelect	选择数据表	209
GT_PvtStart	启动运动	205
GT_PvtStatus	读取状态	206
第 7 章 访问硬件资源		102
GT_GetDi	读取数字 IO 输入状态	177
GT_GetDiRaw	读取数字 IO 输入状态的原始值	177
GT_GetDiReverseCount	读取数字量输入信号的变化次数	178
GT_SetDiReverseCount	设置数字量输入信号的变化次数的初值	214
GT_SetDo	设置数字 IO 输出状态	214
GT_SetDoBit	按位设置数字 IO 输出状态	215
GT_SetDoBitReverse	使数字量输出信号输出定时脉冲信号	215
GT_GetDo	读取数字 IO 输出状态	178
GT_GetEncPos	读取编码器位置	179
GT_GetEncVel	读取编码器速度	179
GT_SetEncPos	修改编码器位置	216
GT_SetDac	设置 dac 输出电压	213
GT_GetDac	读取 dac 输出电压	177
GT_GetAdc	读取模拟量输入的电压值	169
GT_GetAdcValue	读取模拟量输入的数字转换值	169

第 8 章 高速硬件捕获		105
GT_SetCaptureMode	设置编码器捕获方式, 并启动捕获	211
GT_GetCaptureMode	读取编码器捕获方式	173
GT_GetCaptureStatus	读取编码器捕获状态	174
GT_SetCaptureSense	设置捕获电平	211
GT_ClearCaptureStatus	清除捕获状态	161
第 9 章 安全机制		116
GT_SetSoftLimit	设置轴正向软限位和负向软限位	224
GT_GetSoftLimit	读取轴正向软限位和负向软限位	188
第 10 章 运动程序		120
GT_Download	下载运动程序到运动控制器	164
GT_GetFunId	读取运动程序中函数的标识	182
GT_GetVarId	读取运动程序中变量的标识	191
GT_Bind	绑定线程、函数、数据页	156
GT_RunThread	启动线程	209
GT_StopThread	停止正在运行的线程	229
GT_PauseThread	暂停正在运行的线程	200
GT_GetThreadSts	读取线程的状态	189
GT_SetVarValue	设置运动程序中变量的值	227
GT_GetVarValue	读取运动程序中变量的值	191
第 11 章 其它指令		138
11.1 打开/关闭运动控制器		138
GT_Open	打开运动控制器	199
GT_Close	关闭运动控制器	161
GT_SetCardNo	切换当前运动控制器卡号	212
GT_GetCardNo	读取当前运动控制器卡号	174
GT_Reset	复位运动控制器	209
11.2 读取固件版本号		138
GT_GetVersion	读取运动控制器固件的版本号	192
11.3 读取系统时钟		139
GT_GetClock	读取运动控制器系统时钟	174
GT_GetClockHighPrecision	读取运动控制器系统高精度时钟	175
11.4 打开/关闭电机使能信号		139
GT_AxisOn	打开驱动器使能	156
GT_AxisOff	关闭驱动器使能	151
11.5 维护位置值		139
GT_SetPrfPos	修改指定轴的规划位置	222
GT_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	229
GT_ZeroPos	清零规划位置和实际位置, 并进行零漂补偿	230
11.6 电机到位检测		140
GT_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, 置起到位标志	210

GT_GetAxisBand	读取轴到位误差带	170
11.7 设置 PID 参数		145
GT_SetControlFilter	设定 PID 索引，支持 3 组 PID 参数	212
GT_GetControlFilter	读取当前 PID 索引	175
GT_SetPid	设置 PID 参数	221
GT_GetPid	读取 PID 参数	184
11.8 反向间隙补偿		145
GT_SetBacklash	设置反向间隙补偿的相关参数	210
GT_GetBacklash	读取反向间隙补偿的相关参数	173
11.9 自动回原点功能		146
GT_HomeInit	初始化自动回原点功能	193
GT_Home	启动自动回原点功能	192
GT_Index	设置自动回原点功能为 home+index 模式	194
GT_HomeStop	启动原点停止功能	193
GT_HomeSts	查询自动回原点的运行状态	193

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。

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、头文件和 lib 文件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。

2.2 WinCE 下自启动功能的使用

如何开机自启动应用程序：

1. 进入应用程序所在目录，在可执行文件上点击鼠标右键，从右键菜单中选择‘Copy’。
2. 进入GUS控制器的路径 ‘\Hard Disk\Startup’文件夹，在空白处点击鼠标右键，选择‘Paste Shortcut’，即在Startup文件夹中生成一个应用程序的快捷方式，如图 2-1：

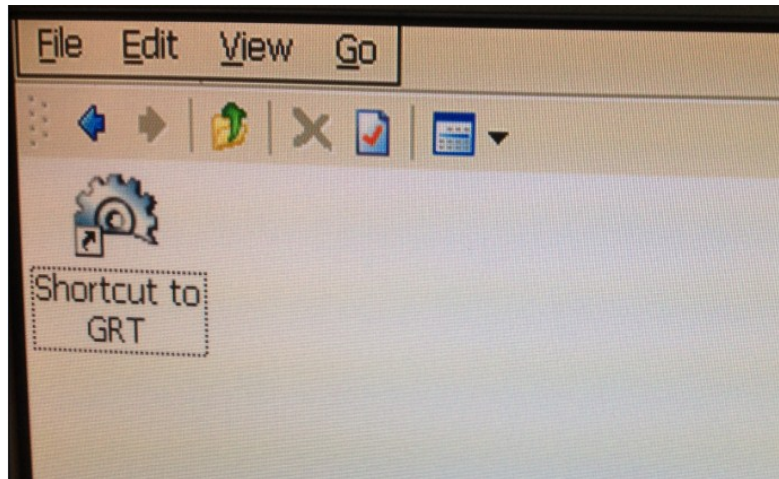


图 2-1 GRT 快捷方式

3. 重新启动GUS控制器，则该应用程序会在WinCE进入后，自动启动。

第3章 命令返回值及其意义



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

3.1 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。

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	
1	指令执行错误	检查当前指令的执行条件是否满足
2	license 不支持	如果需要此功能，请与生产厂商联系。
7	指令参数错误	检查当前指令输入参数的取值
-1	主机和运动控制器通讯失败	<ol style="list-style-type: none"> 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
-6	打开控制器失败	<ol style="list-style-type: none"> 1. 是否正确安装运动控制器驱动程序 2. 是否调用了 2 次 GT_Open 指令 3. 其他程序是否已经打开运动控制器
-7	运动控制器没有响应	更换运动控制器



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



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“第 12 章 指令详细说明”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

在使用运动控制器进行各种操作之前，需要对运动控制器进行配置，使运动控制器的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 Motion Controller Toolkit 2008 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。

4.1 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。

4.1.1 硬件资源

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

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

编码器计数资源(encoder): 用来对外部编码器的脉冲输出进行计数。

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

电压输出资源(dac): 电压输出通道，输出-10V~+10V 的控制电压。

4.1.2 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。

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

轴资源(axis): 将软件资源、硬件资源进行组合，作为整体进行操作。其中包括驱动报警信号、正限位信号、负限位信号、平滑停止信号、紧急停止信号的管理；规划器输出的规划位置的当量变换；编码器计数位置的当量变换等功能。

4.1.3 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。

步进控制方式的基本配置如图 4-1 所示。

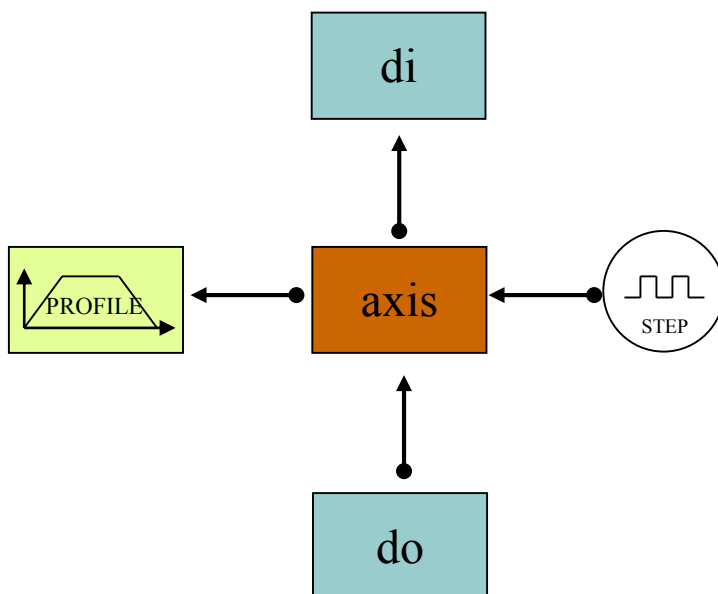


图 4-1 步进控制

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息给数字量输出，来使电机使能。

伺服控制方式的基本配置如图 4-2 所示。

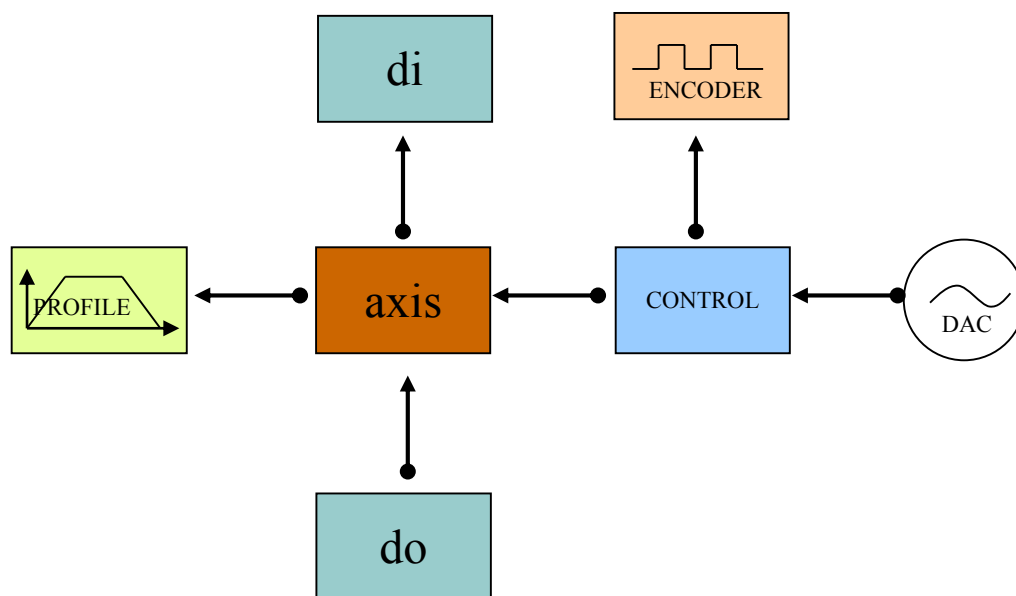


图 4-2 伺服控制

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息给数字量输出，来使电机使能。

4.2 系统配置工具

使用固高科技提供的 Motion Controller Toolkit 2008 运动控制器管理软件能够方便地对系统进行配置，启动以后显示如下界面。



图 4-3 MCT2008 运动控制器管理软件

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。

4.2.1 配置 axis

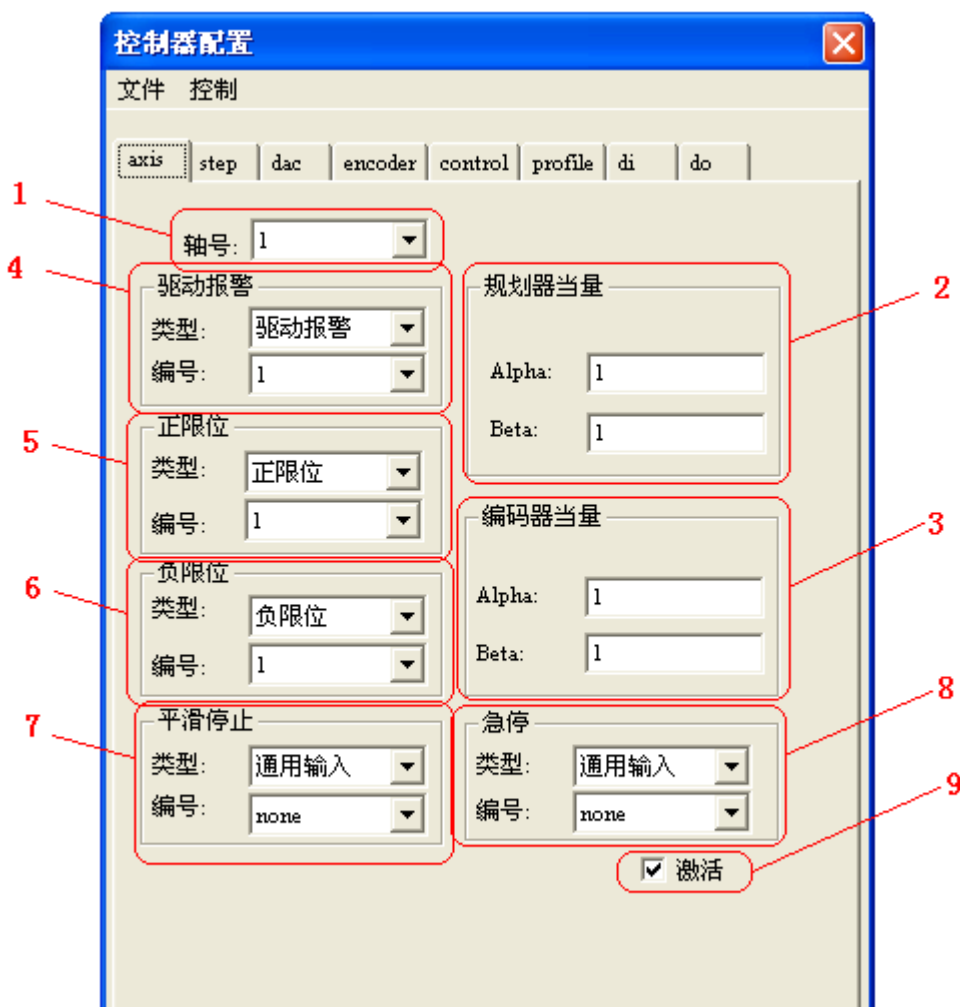


图 4-4 axis 配置界面

1. axis编号选择：选择需要进行配置的axis的编号。
2. 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，所以，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 [GT_ProfileScale](#) 来设置。

3. 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, 所以, 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 [GT_EncScale](#) 来设置。

4. 驱动报警信号数字量输入选择：选择驱动报警信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为驱动报警信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为选择驱动报警数字量输入；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“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 [GT_AlarmOff](#)设置，驱动报警有效可以通过指令 [GT_AlarmOn](#)设置。
5. 正限位信号数字量输入选择：选择正限位信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为正限位信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为选择正限位数字量输入；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“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 [GT_LmtsOff](#)设置，限位开关有效可以通过指令 [GT_LmtsOn](#)设置。
6. 负限位信号数字量输入选择：选择负限位信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为负限位信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为选择负限位数字量输入；第二个下拉列表选择数字量输入的编号，在第二个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 [GT_LmtsOff](#)设置，限位开关有效可以通过指令 [GT_LmtsOn](#)设置。
7. 平滑停止信号数字量输入选择：选择平滑停止信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为平滑停止信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为没有平滑停止信号；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该axis没有平滑停止信号。平滑停止信号数字量输入选择可以通过指令 [GT_SetStopIo](#)设置。
8. 紧急停止信号数字量输入选择：选择紧急停止信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为紧急停止信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为没有紧急停止信号；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该axis没有紧急停止信号。紧急停止信号数字量输入选择可以通过指令 [GT_SetStopIo](#)设置。
9. Axis激活选项：如果axis不被激活，则与该axis相关的所有计算和管理任务将会无效。默认axis都是激活的。但是如果没用用到某个axis的相关功能，则可以不把该axis激活，这样可以节约运动控制器的处理资源。

4.2.2 配置 step



图 4-5 step 配置界面

1. Step编号选择：选择需要进行配置的step的编号。
2. Step输出脉冲信号模式选择：可以选择step脉冲输出通道的脉冲输出模式，可以为“脉冲+方向”或者“CW/CCW”，默认为“脉冲+方向”。设置为“脉冲+方向”模式，可以调用指令 [GT_StepDir](#)来实现；设置为“CW/CCW”模式，可以调用指令 [GT_StepPulse](#)来实现。
3. Step激活选项：如果step不被激活，则该脉冲输出通道将不可用，不会输出脉冲。默认step都是激活的。但是如果没用用到某个step，则可以不把该step激活，这样可以节约运动控制器的处理资源。

4.2.3 配置 dac

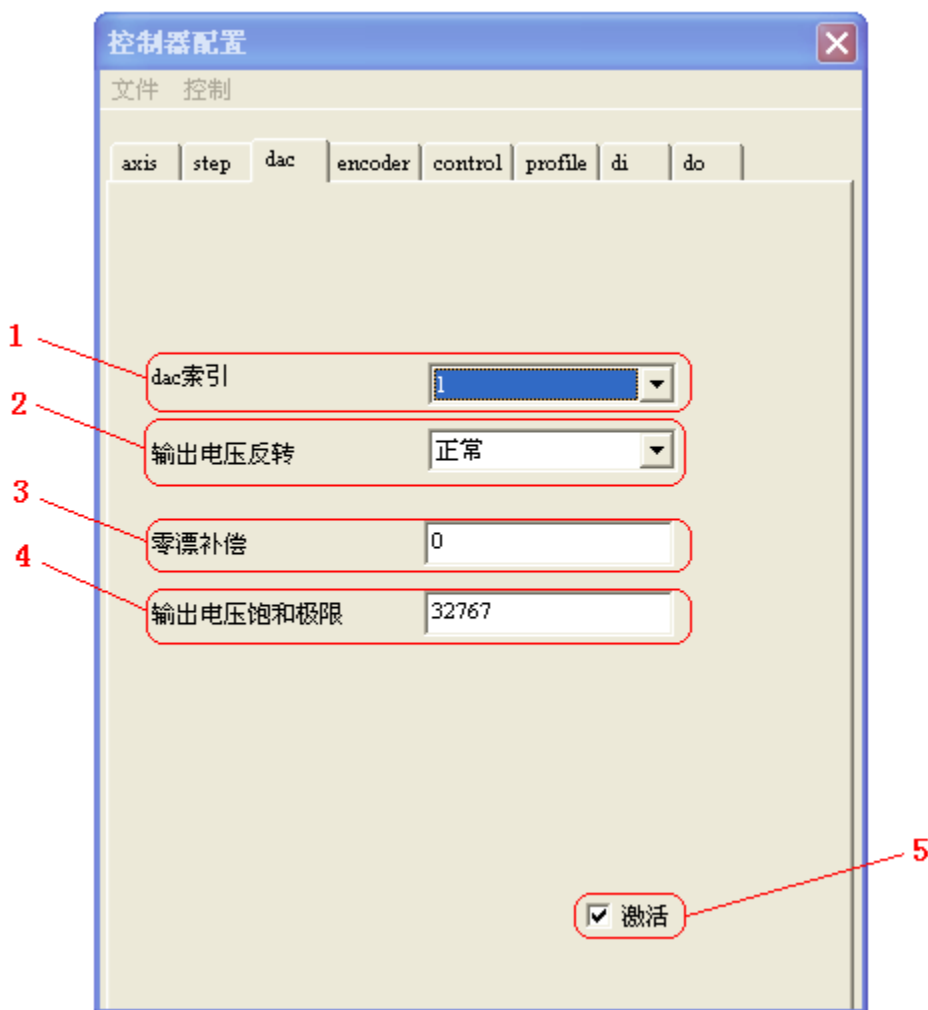


图 4-6 dac 配置界面

1. Dac编号选择：选择需要进行配置的dac的编号。
2. Dac输出电压反转：选择是否需要将dac的输出电压取反，如果为“正常”，则向dac中写入正值时，dac输出正电压，向dac中写入负值时，dac输出负电压；如果为“取反”，则反之。
3. Dac的零漂补偿值：如果需要对dac进行零漂补偿时，在这里设置具体的零漂补偿值。该项可以通过指令 `GT_SetMtrBias`来设置。
4. Dac输出电压饱和极限：该项设置dac能够输出的最大电压绝对值。如果设置为32767，则允许输出的电压范围为：-10V~+10V，设置为16384，则允许输出的电压范围为：-5V~+5V。如果control输出的控制量绝对值，或者用户使用 `GT_SetDac`指令设置的电压值的绝对值超过设定值时，将会按照该项设置的参数被限制在指定电压范围之内。该项可以通过指令 `GT_SetMtrLmt`来设置。
5. Dac激活选项：如果dac不被激活，则该电压输出通道将不可用，不会输出电压值。默认dac都是激活的。但是如果没用用到某个dac，则可以不把该dac激活，这样可以节约运动控制器的处理资源。

4.2.4 配置 encoder

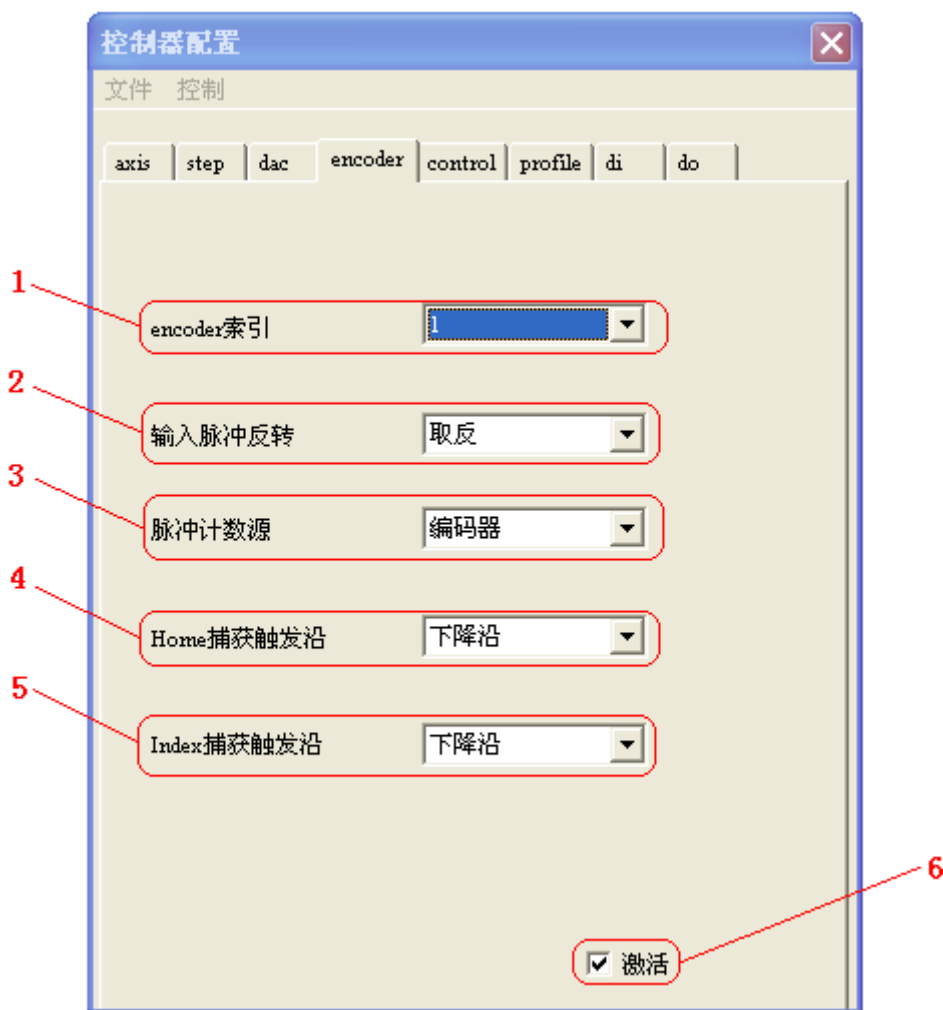


图 4-7 encoder 配置界面

1. Encoder编号选择：选择需要进行配置的encoder的编号。
2. Encoder输入脉冲反转：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下表所示，该项可以通过指令 `GT_EncSns`来修改。

表 4-1 反馈脉冲方向与编码器技术方向关系图

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

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

4. Home捕获触发沿：用来设置Home捕获的触发沿，默认为下降沿触发。如果选择了常闭开关，可以将捕获沿设置为上升沿触发。该项可以通过指令 [GT_SetCaptureSense](#)来修改。
5. Index捕获触发沿：用来设置Index捕获的触发沿，默认为下降沿触发。该项可以通过指令 [GT_SetCaptureSense](#)来修改。
6. Encoder激活选项：如果encoder不被激活，则将不会对输入脉冲进行计数。默认encoder都是激活的。但是如果没用用到某个encoder，则可以不把该encoder激活，这样可以节约运动控制器的处理资源。

4.2.5 配置 control

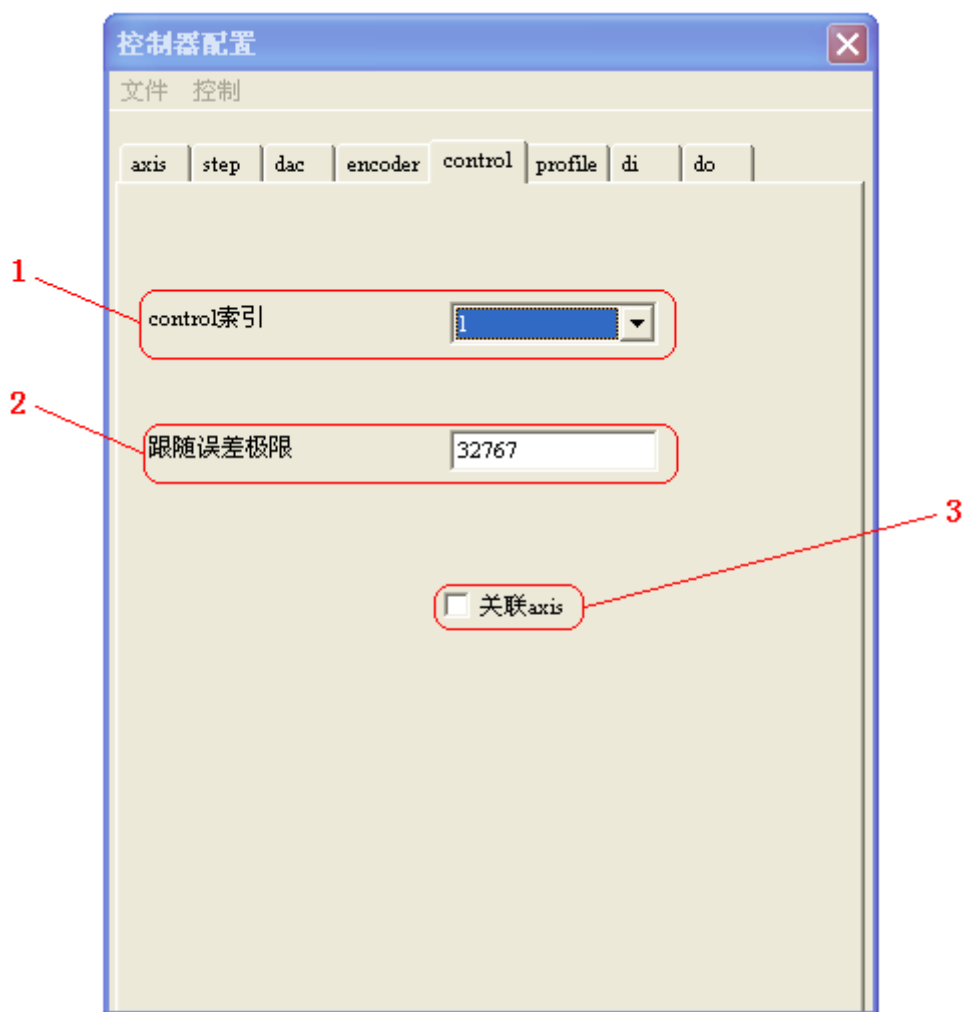


图 4-8 control 配置界面

1. Control编号选择：选择需要进行配置的control的编号。
2. 跟随误差极限：表示当规划位置和实际位置的误差的极限。当跟随误差超过设定的极限时，自动关闭axis的驱动器使能信号。默认为：32767，单位：pulse。该项可以通过指令 [GT_SetPosErr](#)来设置。
3. Control关联选项：如果需要伺服闭环控制，应该使control与axis关联。默认为：不关联，即开环脉冲控制方式。关联之后，运动控制器会将相应编号的encoder、dac、axis、control关联在一起，如图 4-2所示，此时，dac的值将不能独立设置，如果调用 [GT_SetDac](#)指令将会无效。切换开环方式和闭环方式可以通过指令 [GT_CtrlMode](#)来实现。

4.2.6 配置 profile

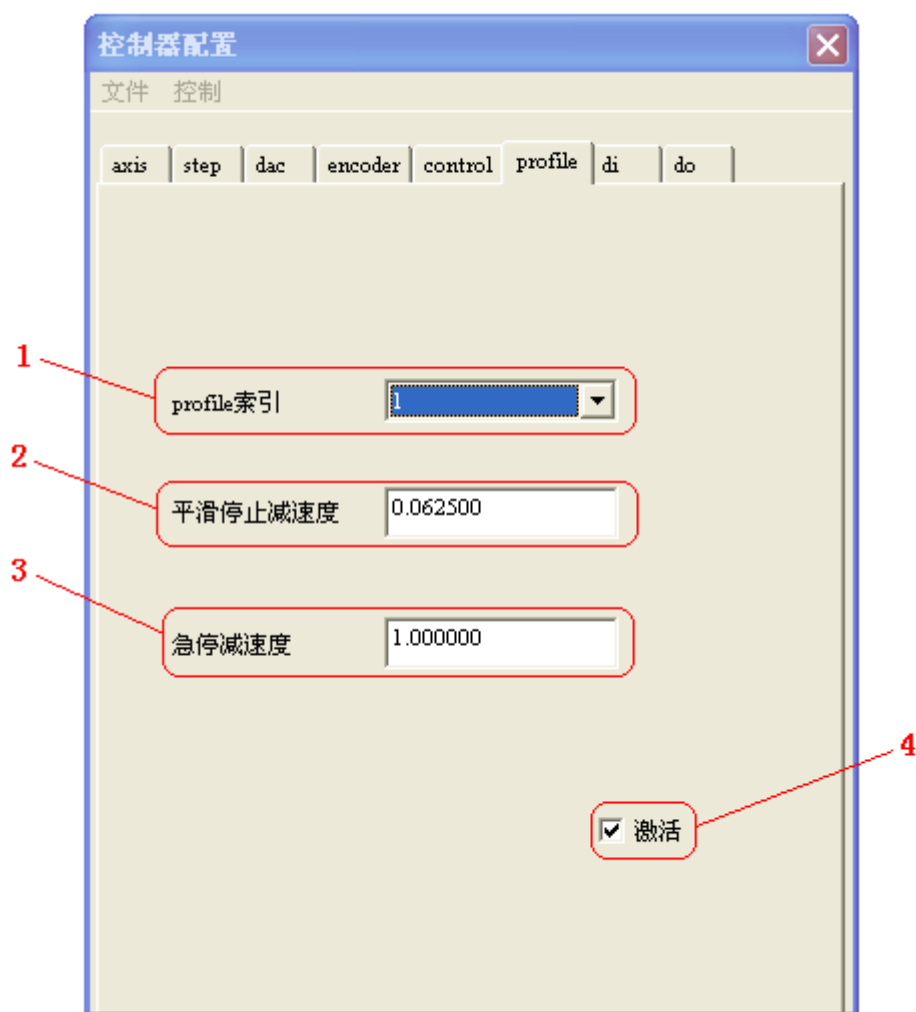


图 4-9 profile 配置界面

1. Profile编号选择：选择需要进行配置的profile的编号。
2. 平滑停止减速度：表示调用 `GT_Stop`指令平滑停止时所使用的减速度，默认为 $0.0625 \text{ pulse/ms}^2$ 。该值可以通过调用指令 `GT_SetStopDec`来修改。
3. 紧急停止减速度：表示调用 `GT_Stop`指令急停时所使用的减速度，默认为 1 脉冲/毫秒^2 。该值可以通过调用指令 `GT_SetStopDec`来修改。
4. Profile激活选项：如果profile不被激活，则将不会进行运动规划。默认profile都是激活的。但是如果如果没有用到某个profile，则可以不把该profile激活，这样可以节约运动控制器的处理资源。

4.2.7 配置 di



图 4-10 di 配置界面

1. Di类型选择：选择需要配置的di的类型，包括：驱动报警、正限位、负限位、原点、通用输入。
2. Di编号选择：选择需要配置的di的编号。
3. 输入反转：表示di的输入逻辑取反。默认情况下，0表示输入低电平，1表示输入高电平；输入反转后，0表示输入高电平，1表示输入低电平。该项可以通过指令 `GT_GpiSns`设置。
4. 滤波时间：表示di输入信号维持设定时间才有效，默认滤波时间为3，单位为：250微秒。
5. Di激活选项：如果di不被激活，则该数字量输入将不起作用。默认di都是激活的。但是如果不用到某个di，则可以不把该di激活，这样可以节约运动控制器的处理资源。

4.2.8 配置 do

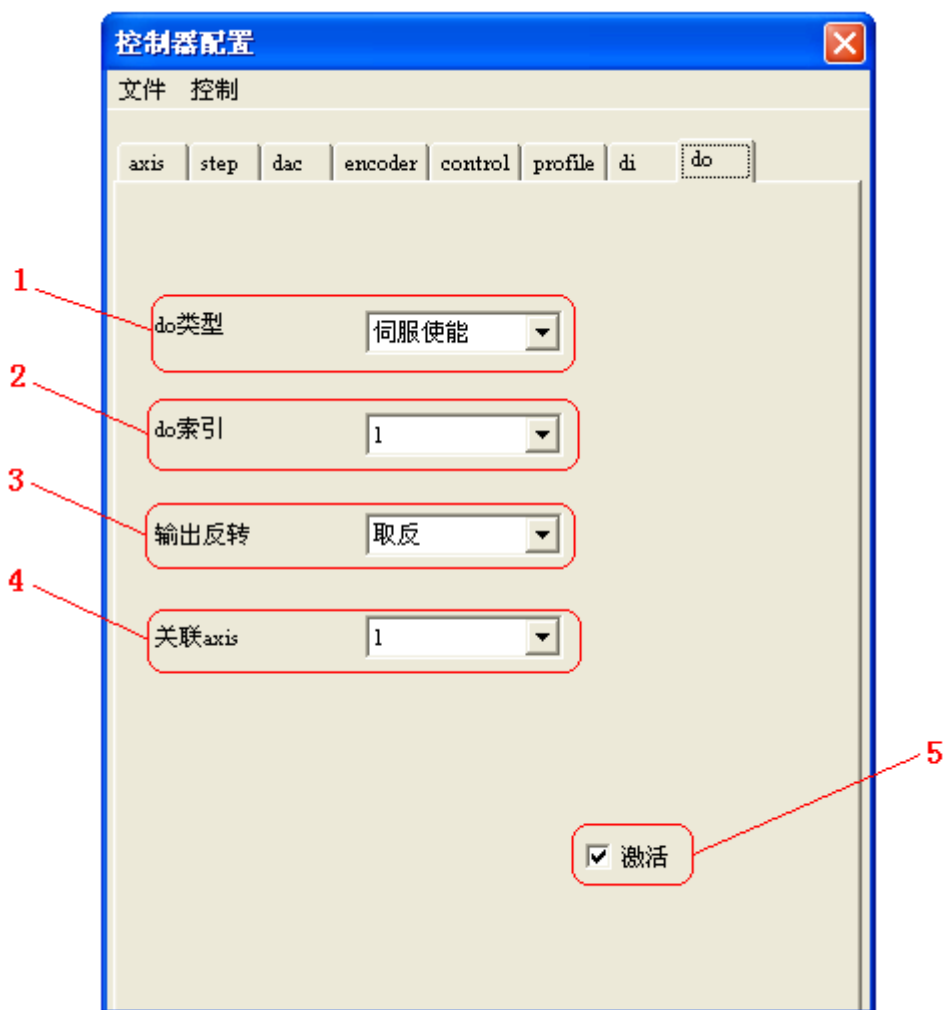


图 4-11 do 配置界面

1. Do类型选择：选择需要配置的do的类型，包括：伺服使能、清除报警、通用输出。
2. Do编号选择：选择需要配置的do的编号。
3. 输出反转：表示do的输出逻辑取反。默认为：正常，0表示输出低电平，1表示输出高电平。输出反转后，0表示输出高电平，1表示输出低电平。
4. 关联axis：表示该do关联到指定axis的驱动器使能。默认情况下，各轴都具有独立的驱动器使能do作为输出，当调用 `GT_AxisOn` 指令时，该do置1。如果驱动器是低电平使能，必须把“输出反转”设置为取反。Do一旦和axis关联，就不能调用 `GT_SetDo`或 `GT_SetDoBit`指令直接设置其输出电平。如果不需要驱动器使能信号，可以取消和驱动器使能do的关联。取消关联以后，驱动器使能do就可以作为普通do来使用，可以调用 `GT_SetDo`或 `GT_SetDoBit`直接设置其输出电平。
5. Do激活选项：如果do不被激活，则该数字量输出将不起作用。默认do都是激活的。但是如果没用用到某个do，则可以不把该do激活，这样可以节约运动控制器的处理资源。

4.3 配置文件生成和下载

表 4-2 下载配置文件指令

指令	说明	页码
GT_LoadConfig	下载配置信息到运动控制器	199

按照 4.2 中所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”界面的“文件”菜单，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。

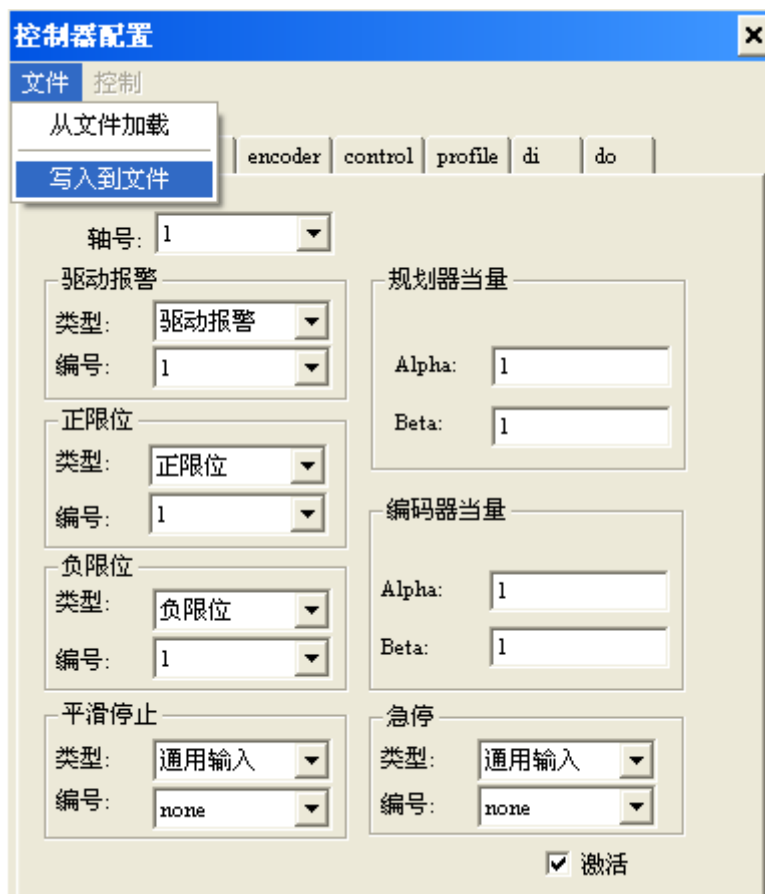


图 4-12 生成配置文件界面

用户可以调用 `GT_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 `GT_LoadConfig` 指令时，参数需要包含配置文件的绝对路径。

4.4 配置信息修改指令



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“**第 12 章 指令详细说明**”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

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。

4.4.1 指令列表

表 4-3 配置信息指令列表

指令	说明	页码
GT_AlarmOff	控制轴驱动报警信号无效	151
GT_AlarmOn	控制轴驱动报警信号有效	151
GT_LmtsOn	控制轴限位信号有效	195
GT_LmtsOff	控制轴限位信号无效	195
GT_ProfileScale	设置控制轴的规划器当量变换值	202
GT_EncScale	设置控制轴的编码器当量变换值	165
GT_StepDir	将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”	227
GT_StepPulse	将脉冲输出通道的脉冲输出模式设置为“CW/CCW”	228
GT_SetMtrBias	设置模拟量输出通道的零漂电压补偿值	220
GT_GetMtrBias	读取模拟量输出通道的零漂电压补偿值	183
GT_SetMtrLmt	设置模拟量输出通道的输出电压饱和极限值	220
GT_GetMtrLmt	读取模拟量输出通道的输出电压饱和极限值	184
GT_EncSns	设置编码器的计数方向	165
GT_EncOn	设置为“外部编码器”计数方式	165
GT_EncOff	设置为“脉冲计数器”计数方式	164
GT_SetPosErr	设置跟随误差极限值	222
GT_GetPosErr	读取跟随误差极限值	185
GT_SetStopDec	设置平滑停止减速度和急停减速度	224
GT_GetStopDec	读取平滑停止减速度和急停减速度	189
GT_LmtSns	设置运动控制器各轴限位开关触发电平	194
GT_CtrlMode	设置控制轴为模拟量输出或脉冲输出	164
GT_SetStopIo	设置平滑停止和紧急停止数字量输入的信息	225
GT_GpiSns	设置运动控制器数字量输入的电平逻辑	192
GT_SetAdcFilter	设置模拟量输入的滤波器时间参数	210

4.5 重点说明

(1) 编码器计数方向设置

指令 `GT_EncSns` 可以修改运动控制器各编码器的计数方向，当指令参数的某个状态位为 1 时，将所对应的控制轴的编码器计数方向取反，指令参数的状态位定义如表 4-4 所示：

表 4-4 编码器计数方向设置

状态位	8	7	6	5	4	3	2	1	0
编码器	辅助编码器	Enc8	Enc7	Enc6	Enc5	Enc4	Enc3	Enc2	Enc1

(2) 设置限位开关触发电平

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

指令 `GT_LmtSns` 的参数设置各轴正负限位开关的触发电平，当该参数的某个状态位为 0 时，表示将对应的限位开关设置为高电平触发，当某个状态位为 1 时，表示将对应的限位开关设置为低电平触发。指令参数和各轴限位的对应关系如下所示：

表 4-5 设置限位开发触发电平

状态位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
限位开关	轴 8		轴 7		轴 6		轴 5		轴 4		轴 3		轴 2		轴 1	
	—	+	—	+	—	+	—	+	—	+	—	+	—	+	—	+

第5章 运动状态检测



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“第 12 章 指令详细说明”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

5.1 指令列表

表 5-1 运动状态检测指令列表

指令	说明	页码
GT_GetSts	读取轴状态	189
GT_ClrSts	清除驱动器报警标志、跟随误差超限标志、限位触发标志 ✓ 只有当驱动器没有报警时才能清除轴状态字的报警标志 ✓ 只有当跟随误差正常以后，才能清除跟随误差超限标志 ✓ 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	161
GT_GetPrfMode	读取轴运动模式	185
GT_GetPrfPos	读取规划位置	186
GT_GetPrfVel	读取规划速度	186
GT_GetPrfAcc	读取规划加速度	185
GT_GetAxisPrfPos	读取轴(axis)的规划位置值	172
GT_GetAxisPrfVel	读取轴(axis)的规划速度值	173
GT_GetAxisPrfAcc	读取轴(axis)的规划加速度值	172
GT_GetAxisEncPos	读取轴(axis)的编码器位置值	170
GT_GetAxisEncVel	读取轴(axis)的编码器速度值	171
GT_GetAxisEncAcc	读取轴(axis)的编码器加速度值	171
GT_GetAxisError	读取轴(axis)的规划位置值和编码器位置值的差值	171
GT_Stop	停止一个或多个轴的规划运动，停止坐标系运动	228

5.2 重点说明

5.2.1 轴状态定义

当调用 [GT_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下：

表 5-2 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	跟随误差超限标志 控制轴规划位置 and 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 and 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
12~31	保留

驱动器报警标志、限位触发标志、IO 停止、跟随误差超限标志触发以后, 不会自动清 0。只有当产生异常的原因已经消除以后, 调用 `GT_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。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。

5.2.2 轴(axis)状态读取的相关指令

在 4.2.1 中关于轴(axis)的配置部分曾经提到, 规划器(profile)和编码器(encoder)的输出值可以通过 axis 进行当量变换之后, 再输出。上述与 axis 相关的状态读取指令主要用来读取 profile 和 encoder 的输出值经过当量变换之后的值。

其中, `GT_GetAxisPrfPos`、`GT_GetAxisPrfVel` 和 `GT_GetAxisPrfAcc` 用来读取 profile 输出值经过当量变换之后的规划位置、规划速度和规划加速度。`GT_GetAxisEncPos`、`GT_GetAxisEncVel` 和 `GT_GetAxisEncAcc` 用来读取 encoder 输出值经过当量变换之后的编码器位置值。`GT_GetAxisError` 指令用来读取 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。

第6章 运动模式



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“第12章 指令详细说明”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

GUS 系列运动控制器每个轴都可以独立工作在点位、Jog、PT、PVT、电子齿轮或 Follow 运动模式(电子凸轮)下。

表 6-1 设置运动模式指令列表

指令	说明	页码
GT_PrffTrap	设置指定轴为点位运动模式	202
GT_PrffJog	设置指定轴为 Jog 模式	201
GT_PrffPt	设置指定轴为 PT 模式	201
GT_PrffGear	设置指定轴为电子齿轮模式	200
GT_PrffFollow	设置指定轴为 Follow 模式	200
GT_PrffPvt	设置指定轴为 PVT 模式	201
GT_GetPrffMode	读取轴运动模式	185

只能在规划静止状态下切换轴运动模式。调用 [GT_Stop](#) 指令可以停止一个或多个轴的运动。

6.1 点位运动

6.1.1 指令列表

表 6-2 设置点位运动指令列表

指令	说明	页码
GT_PrffTrap	设置指定轴为点位运动模式	202
GT_SetTrapPrm	设置点位模式运动参数	226
GT_GetTrapPrm	读取点位模式运动参数	190
GT_SetPos	设置目标位置	222
GT_GetPos	读取目标位置	184
GT_SetVel	设置目标速度，单位是“脉冲/毫秒”	227
GT_GetVel	读取目标速度，单位是“脉冲/毫秒”	191
GT_Update	启动点位运动	230

6.1.2 重点说明

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 [GT_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標速度。

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 6-1 所示。

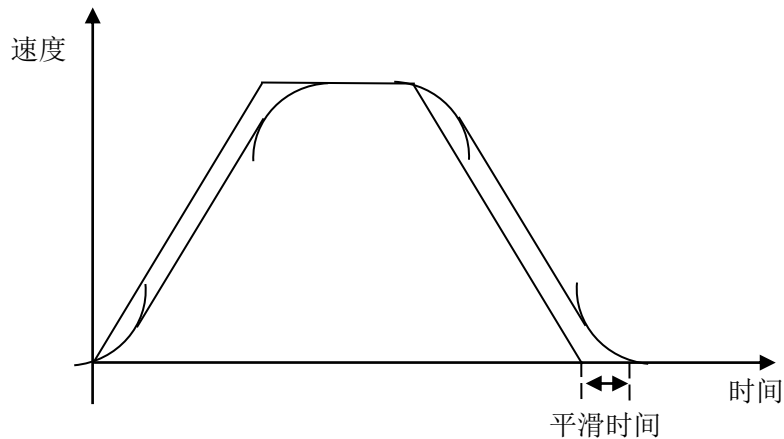


图 6-1 点位运动速度曲线

平滑时间是指加速度的变化时间，单位是毫秒，取值范围是[0,50]。

6.1.3 例程

例程 6-1 点位运动

该例程生成一段梯形曲线速度规划，如图 6-2 所示：

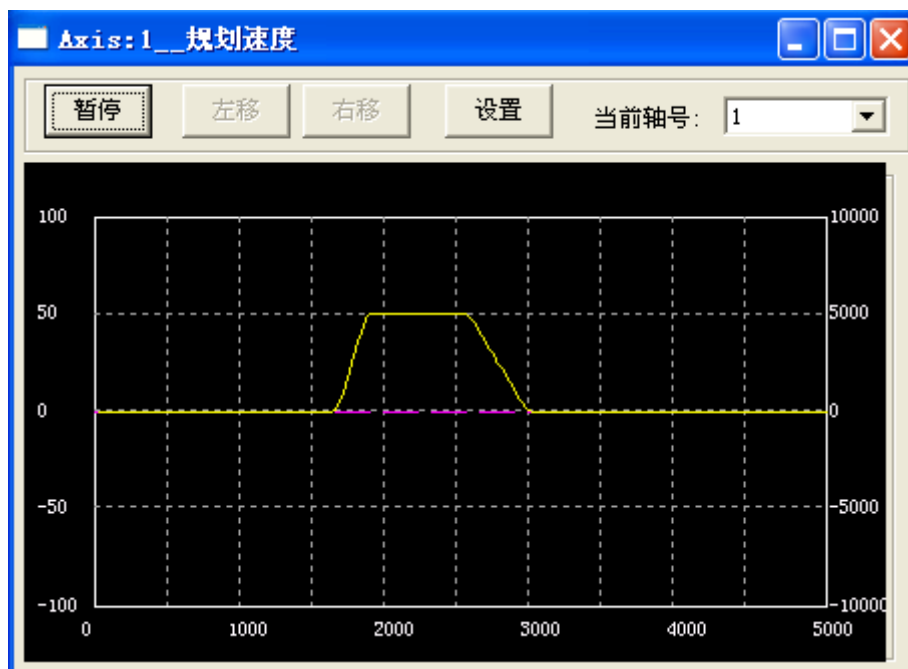


图 6-2 点位运动速度规划

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

```
#define AXIS 1
```

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

```
short rtn;
TTrapPrm trap;
long sts;
double prfPos;

// 打开运动控制器
rtn = GT_Open ();
printf(" GT_Open ()=%d\n",rtn);

// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
rtn = GT_LoadConfig ("test.cfg");
printf(" GT_LoadConfig()=%d\n",rtn);

// 清除各轴的报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts()=%d\n",rtn);

// AXIS轴规划位置清零
rtn = GT_SetPrfPos (AXIS,0);
printf(" GT_SetPrfPos ()=%d\n",rtn);

// 将AXIS轴设为点位模式
rtn = GT_PrTrap(AXIS);
printf(" GT_PrTrap()=%d\n",rtn);

// 读取点位运动参数
rtn = GT_GetTrapPrm(AXIS,&trap);
printf(" GT_GetTrapPrm()=%d\n",rtn);
trap.acc = 0.25;
trap.dec = 0.125;
trap.smoothTime = 25;
// 设置点位运动参数
rtn = GT_SetTrapPrm(AXIS,&trap);
printf(" GT_SetTrapPrm()=%d\n",rtn);

// 设置AXIS轴的目标位置
rtn = GT_SetPos(AXIS,50000L);
printf(" GT_SetPos()=%d\n",rtn);

// 设置AXIS轴的目标速度
rtn = GT_SetVel(AXIS,50);
printf(" GT_SetVel()=%d\n",rtn);

// 启动AXIS轴的运动
rtn = GT_Update(1<<(AXIS-1));
```

```

printf("  GT_Update()=%d\n",rtn);

do
{
    // 读取AXIS轴的状态
    rtn =  GT_GetSts(AXIS,&sts);

    // 读取AXIS轴的规划位置
    rtn =  GT_GetPrfPos(AXIS,&prfPos);

    printf("sts=0x%-10lprfPos=%-10.11f\r",sts,prfPos);
}while(sts&0x400);// 等待AXIS轴规划停止

getch();

return 0;
}

```

6.2 Jog 模式

6.2.1 指令列表

表 6-3 设置 Jog 模式指令列表

指令	说明	页码
GT_PrjJog	设置指定轴为 Jog 模式	201
GT_SetJogPrm	设置 Jog 运动参数	219
GT_GetJogPrm	读取 Jog 运动参数	183
GT_SetVel	设置目标速度，单位是“脉冲/毫秒”	227
GT_GetVel	读取目标速度，单位是“脉冲/毫秒”	191
GT_Update	启动 Jog 模式运动	230

6.2.2 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 `GT_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 6-3 所示：

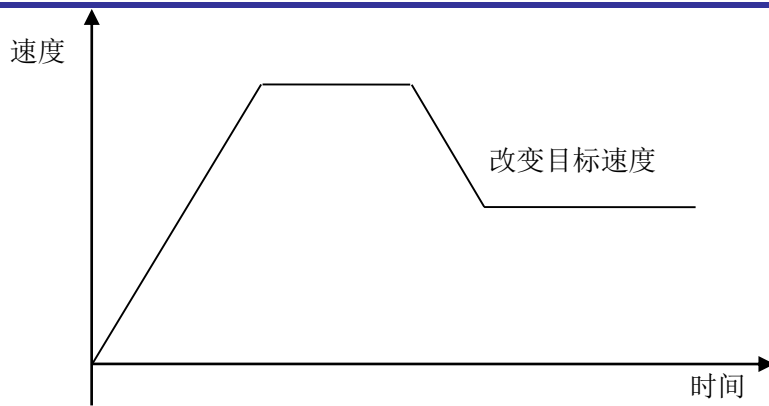


图 6-3 Jog 模式速度曲线

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。

6.2.3 例程

例程 6-2 Jog 运动

该例程在 jog 模式下动态改变目标速度，如图所示：

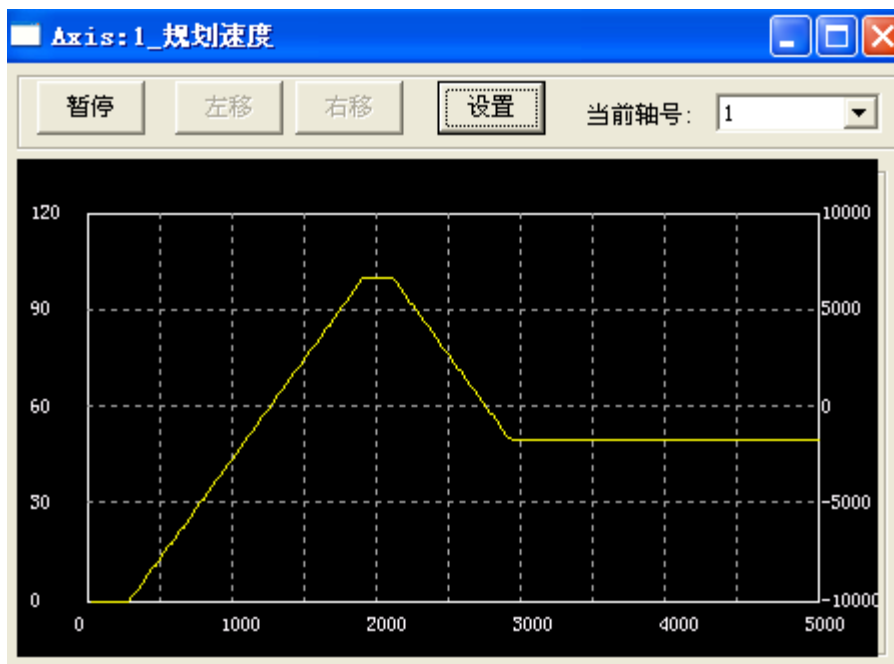


图 6-4 Jog 模式动态改变目标速度

```
#include "stdafx.h"
#include "conio.h"
#include "gts.h"
#define AXIS 1
int main(int argc, char* argv[])
{
    short rtn;
    TJogPrm jog;
    long sts;
```

```
double prfPos,prfVel;

// 打开运动控制器
rtn = GT_Open ();
printf(" GT_Open ()=%d\n",rtn);

// 复位运动控制器
rtn = GT_Reset ();
printf(" GT_Reset ()=%d\n",rtn);

// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
rtn = GT_LoadConfig("test.cfg");
printf(" GT_LoadConfig()=%d\n",rtn);

// 清除各轴的报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts()=%d\n",rtn);

// 将AXIS轴设为Jog模式
rtn = GT_PrJog(AXIS);
printf(" GT_PrJog()=%d\n",rtn);
// 读取Jog运动参数
rtn = GT_GetJogPrm(AXIS,&jog);
printf(" GT_GetJogPrm()=%d\n",rtn);
jog.acc = 0.0625;
jog.dec = 0.0625;
// 设置Jog运动参数
rtn = GT_SetJogPrm(AXIS,&jog);
printf(" GT_SetJogPrm()=%d\n",rtn);

// 设置AXIS轴的目标速度
rtn = GT_SetVel(AXIS,100);
printf(" GT_SetVel()=%d\n",rtn);

// 启动AXIS轴的运动
rtn = GT_Update(1<<(AXIS-1));
printf(" GT_Update()=%d\n",rtn);

while(1)
{
    // 读取AXIS轴的状态
    rtn = GT_GetSts(AXIS,&sts);

    // 读取AXIS轴的规划位置
    rtn = GT_GetPrfPos(AXIS,&prfPos);
```

```
// 读取AXIS轴的规划速度
rtn = GT_GetPrfVel(Axis,&prfVel);

printf("sts=0x%-10lprfVel=%-10.2lfprfPos=%-10.1lf\r",sts,prfVel,prfPos);

if( prfPos >= 100000 )
{
    // 设置AXIS轴新的目标速度
    rtn = GT_SetVel(Axis,50);
    printf("\n GT_SetVel()=%d",rtn);

    // AXIS轴新的目标速度生效
    rtn = GT_Update(1<<(Axis-1));
    printf("\n GT_Update()=%d\n",rtn);

    break;
}
}

while(!kbhit())
{
    // 读取AXIS轴的状态
    rtn = GT_GetSts(Axis,&sts);

    // 读取AXIS轴的规划位置
    rtn = GT_GetPrfPos(Axis,&prfPos);

    // 读取AXIS轴的规划速度
    rtn = GT_GetPrfVel(Axis,&prfVel);

    printf("sts=0x%-10lprfVel=%-10.2lfprfPos=%-10.1lf\r",sts,prfVel,prfPos);
}

getch();

return 0;
}
```


6.3 PT 模式

6.3.1 指令列表

表 6-4 设置 PT 模式指令列表

指令	说明	页码
GT_PrftPt	设置指定轴为 PT 模式	201
GT_PtSpace	查询 PT 指定 FIFO 的剩余空间	203
GT_PtData	向 PT 指定 FIFO 增加数据	203
GT_PtClear	清除 PT 指定 FIFO 中的数据 运动状态下该指令无效 动态模式下该指令无效	202
GT_SetPtLoop	设置 PT 模式循环执行的次数 动态模式下该指令无效	223
GT_GetPtLoop	查询 PT 模式循环执行的次数 动态模式下该指令无效	187
GT_PtStart	启动 PT 模式运动	204
GT_SetPtMemory	设置 PT 运动模式的缓存区大小	223
GT_GetPtMemory	读取 PT 运动模式的缓存区大小	187

6.3.2 重点说明

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

PT 模式使用一系列“位置、时间”数据点描述速度规划，用户需要将速度曲线分割成若干段，如图 6-5 所示。

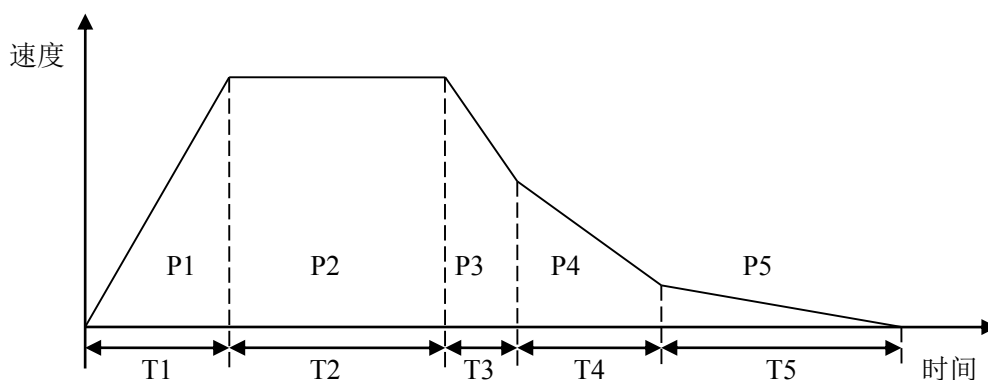


图 6-5 PT 运动速度曲线

整个速度曲线被分割成 5 段，第 1 段起点速度为 0，经过时间 T_1 运动位移 P_1 ，因此第 1 段的终点速度为 $v_1 = \frac{2P_1}{T_1}$ ；第 2 段起点速度为 v_1 ，经过时间 T_2 运动位移 P_2 ，因此第 2 段的终点速度为 $v_2 = \frac{2P_2}{T_2} - v_1$ ；第 3、4、5 段依此类推。

用户只需要给出每段所需时间和位移，运动控制器会计算段内各点的速度和位置，生成一条连续的速度曲线。为了得到光滑的速度曲线，可以增加速度曲线的分割段数。

在描述一次完整的 PT 运动时，第 1 段的起点位置和时间被假定为 0，各段的终点位置和时间都是相对于第 1 段的起点的绝对值。位置的单位是脉冲，时间单位是毫秒。

(1) 数据段类型

PT 模式的数据段有 3 种类型。

PT_SEGMENT_NORMAL 表示普通段，FIFO 中第 1 段的起点速度为 0，从第 2 段起每段的起点速度等于上一段的终点速度。

PT_SEGMENT_EVEN 表示匀速段，FIFO 中各段的段内速度保持不变，段内速度=段内位移/段内时间。如图 6-6 所示：

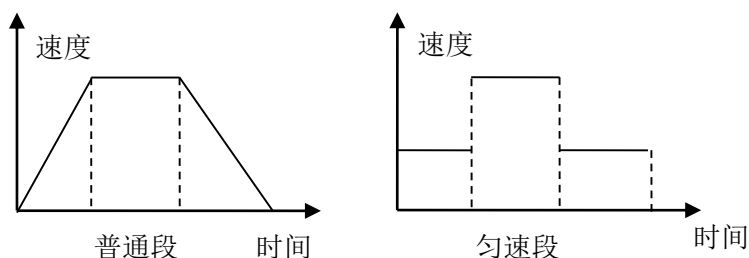


图 6-6 PT 模式匀速段类型

PT_SEGMENT_STOP 表示停止段，该段的终点速度为 0，起点速度根据段内位移和段内时间计算得到，和上一段的终点速度无关。如图 6-7 所示：

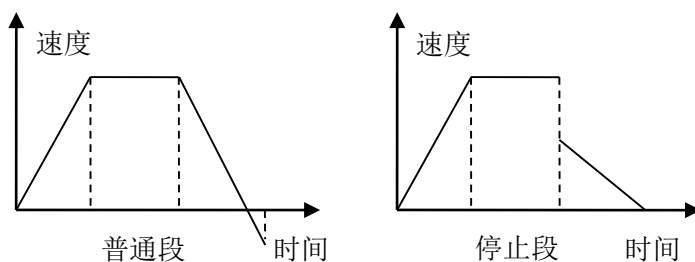


图 6-7 PT 模式停止段类型

(2) PT 运动模式

PT 具有 2 种 FIFO 使用模式：静态模式和动态模式。PT 具有 2 个 FIFO 用来存放数据。

静态模式下，可以选择启动其中一个 FIFO，运动完成以后规划停止。控制器不会清除 FIFO 中的数据，用户可以重复使用 FIFO 中的数据。静止状态下调用 `GT_PtClear` 指令可以清空指定 FIFO。在运动状态下不能清空正在使用的 FIFO，但可以清除没有在使用的 FIFO。

动态模式下，当一个 FIFO 中的数据用完以后会自动清空，同时切换到另一个 FIFO，此时可以向控制器发送新的 PT 数据。当 2 个 FIFO 中的数据都用完以后规划停止。为了避免异常停止，必须在 2 个 FIFO 中的数据都用完之前及时发送新的数据。调用 `GT_PtSpace` 指令可以查询剩余多少数据空间。

在切换到 PT 模式的同时设置 FIFO 为“静态模式”或“动态模式”。进入 PT 模式以后就不能修改 FIFO 的使用模式。

6.3.3 例程

(1) PT 静态 FIFO

例程 6-3 PT 静态 FIFO

该例程生成一段梯形曲线速度规划，如图 6-8 所示：

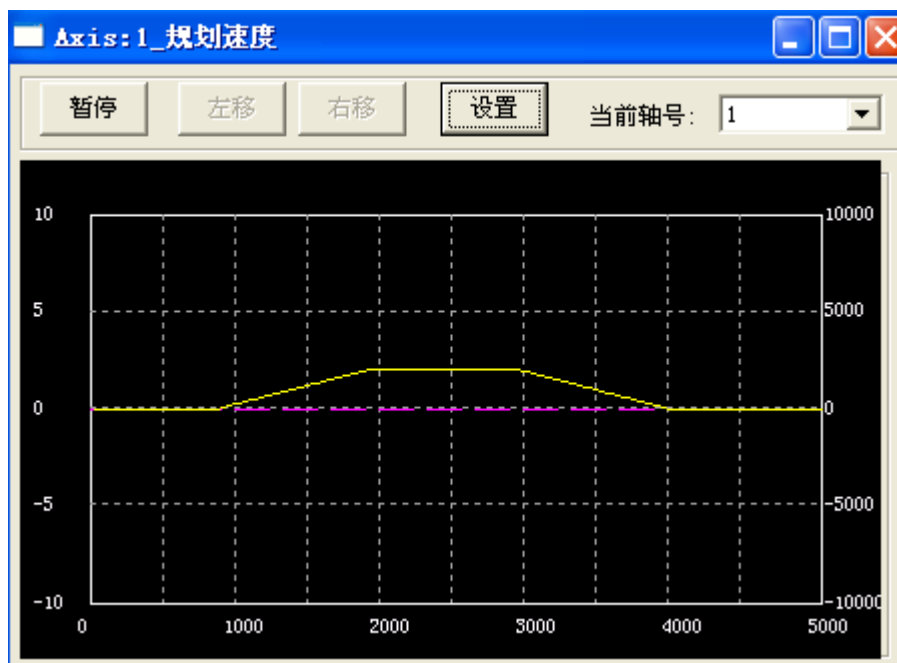


图 6-8 PT 模式梯形曲线速度规划

```

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

#define AXIS      1

int main(int argc, char* argv[])
{
    short rtn,space;
    double pos;
    long time;
    long sts;
    double prfPos,prfVel;

    // 打开运动控制器
    rtn =  GT_Open ();
    printf("  GT_Open ()=%d\n",rtn);

    // 复位控制器
    rtn =  GT_Reset ();
    printf("  GT_Reset ()=%d\n",rtn);

```

```
// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
rtn = GT_LoadConfig("test.cfg");
printf(" GT_LoadConfig()=%d\n",rtn);

// 清除各轴的报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts()=%d\n",rtn);

// 将AXIS轴设为PT模式
rtn = GT_PrPt(AXIS);
printf(" GT_PrPt()=%d\n",rtn);

// 清除AXIS轴的FIFO
rtn = GT_PtClear(AXIS);
printf(" GT_PtClear()=%d\n",rtn);

// 查询PT模式FIFO的剩余空间
rtn = GT_PtSpace(AXIS,&space);
printf(" GT_PtSpace()=%d\tspace=%d\n",rtn,space);

// 向FIFO中增加运动数据
pos = 1024;
time = 1024;
rtn = GT_PtData(AXIS,pos,time);
printf(" GT_PtData()=%d\n",rtn);

// 查询PT模式FIFO的剩余空间
rtn = GT_PtSpace(AXIS,&space);
printf(" GT_PtSpace()=%d\tspace=%d\n",rtn,space);

// 向FIFO中增加运动数据
pos += 2048;
time += 1024;
rtn = GT_PtData(AXIS,pos,time);
printf(" GT_PtData()=%d\n",rtn);

// 查询PT模式FIFO的剩余空间
rtn = GT_PtSpace(AXIS,&space);
printf(" GT_PtSpace()=%d\tspace=%d\n",rtn,space);

// 向FIFO中增加运动数据
pos += 1024;
time += 1024;
rtn = GT_PtData(AXIS,pos,time);
```

```

printf(" GT_PtData()=%d\n",rtn);

// 启动PT运动
rtn = GT_PtStart(1<<(AXIS-1));
printf(" GT_PtStart()=%d\n",rtn);

while(!kbhit())
{
    // 读取AXIS轴的状态
    rtn = GT_GetSts(AXIS,&sts);

    // 读取AXIS轴的规划位置
    rtn = GT_GetPrfPos(AXIS,&prfPos);

    // 读取AXIS轴的规划速度
    rtn = GT_GetPrfVel(AXIS,&prfVel);

    printf("sts=0x%-10l\prfVel=%-10.2lfprfPos=%-10.1lf\r",sts,prfVel,prfPos);
}

return 0;
}

```

(2) PT 动态 FIFO

例程 6-4 PT 动态 FIFO

该例程生成一段正弦曲线速度规划，如图 6-9 所示：

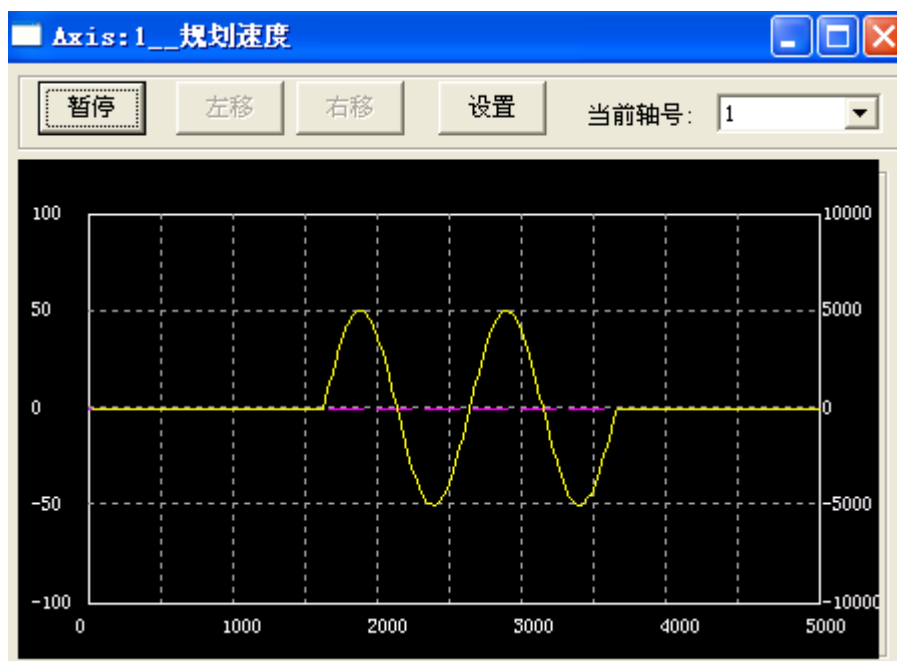


图 6-9 PT 模式正弦曲线速度规划

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

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

#define AXIS      1

#define A        50           // 幅值
#define T        1           // 周期
#define DELTA    0.016       // 时间分段
#define LOOP     2           // 循环次数

#define PI        3.1415926

int main(int argc, char* argv[])
{
    short rtn,space;
    double pos,vel,velPre,time;
    long sts;
    double prfPos,prfVel;
    short start,loop;

    // 打开运动控制器
    rtn =  GT_Open ();
    printf("  GT_Open ()=%d\n",rtn);

    // 配置运动控制器
    // 注意：配置文件取消了各轴的报警和限位
    rtn =  GT_LoadConfig("test.cfg");
    printf("  GT_LoadConfig()=%d\n",rtn);

    // 清除各轴的报警和限位
    rtn =  GT_ClrSts(1,8);
    printf("  GT_ClrSts()=%d\n",rtn);

    // 将AXIS轴设为PT模式
    rtn =  GT_PrftPt(AXIS,PT_MODE_DYNAMIC);
    printf("  GT_PrftPt()=%d\n",rtn);

    // 清空AXIS轴的FIFO
    rtn =  GT_PtClear(AXIS);
    printf("  GT_PtClear()=%d\n",rtn);

    pos = 0;
    vel = velPre = 0;
    time = 0;
    start = 0;
```

```
loop = 1;

while(1)
{
    // 查询PT模式FIFO的剩余空间
    rtn = GT_PtSpace(Axis,&space);
    if( space > 0 )
    {
        time += DELTA;

        // 计算段末速度
        vel = A*sin((2*PI)/T*time);

        // 计算段内位移
        pos += 1000*(vel+velPre)*DELTA/2;

        velPre = vel;

        if(time < loop*T)
        {
            // 发送新数据
            rtn = GT_PtData(Axis,pos,(long)(time*1000));
            if( 0 != rtn )
            {
                printf("\n GT_PtData()=%d\n",rtn);
                getch();
                return 1;
            }
        }
        else
        {
            // 发送终点数据
            rtn = GT_PtData(Axis,0,loop*T*1000,PT_SEGMENT_STOP);
            if( 0 != rtn )
            {
                printf("\n GT_PtData()=%d\n",rtn);
                getch();
                return 1;
            }
        }

        pos = 0;
        time = loop*T;
        velPre = 0;

        ++loop;
    }
}
```

```
        if( loop > LOOP )
        {
            break;
        }
    }
}
else if( 0 == start )
{
    // 启动PT运动
    rtn = GT_PtStart(1<<(AXIS-1));
    printf("\n GT_PtStart(=%d\n",rtn);

    start = 1;
}

// 读取AXIS轴的状态
rtn = GT_GetSts(AXIS,&sts);

// 读取AXIS轴的规划位置
rtn = GT_GetPrfPos(AXIS,&prfPos);

// 读取AXIS轴的规划速度
rtn = GT_GetPrfVel(AXIS,&prfVel);

printf("sts=0x%-10lxprfVel=%-10.2lfprfPos=%-10.1lf\r",sts,prfVel,prfPos);
}

while(!kbhit())
{
    // 读取AXIS轴的状态
    rtn = GT_GetSts(AXIS,&sts);

    // 读取AXIS轴的规划位置
    rtn = GT_GetPrfPos(AXIS,&prfPos);

    // 读取AXIS轴的规划速度
    rtn = GT_GetPrfVel(AXIS,&prfVel);

    printf("sts=0x%-10lxprfVel=%-10.2lfprfPos=%-10.1lf\r",sts,prfVel,prfPos);
}

getch();

return 0;
}
```


6.4 电子齿轮模式

6.4.1 指令列表

表 6-5 设置电子齿轮指令列表

指令	说明	页码
GT_PrflGear	设置指定轴为电子齿轮模式	200
GT_SetGearMaster	设置跟随主轴	218
GT_GetGearMaster	读取跟随主轴	182
GT_SetGearRatio	设置电子齿轮比	219
GT_GetGearRatio	读取电子齿轮比	182
GT_GearStart	启动电子齿轮	168

6.4.2 重点说明

电子齿轮模式能够将 2 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。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。

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。

为了减少跟随滞后，从轴的轴号应当大于主轴的轴号。

电子齿轮模式能够在线修改传动比，当改变传动比时，可以设置离合区间，实现平滑变速，如图 6-10 所示。

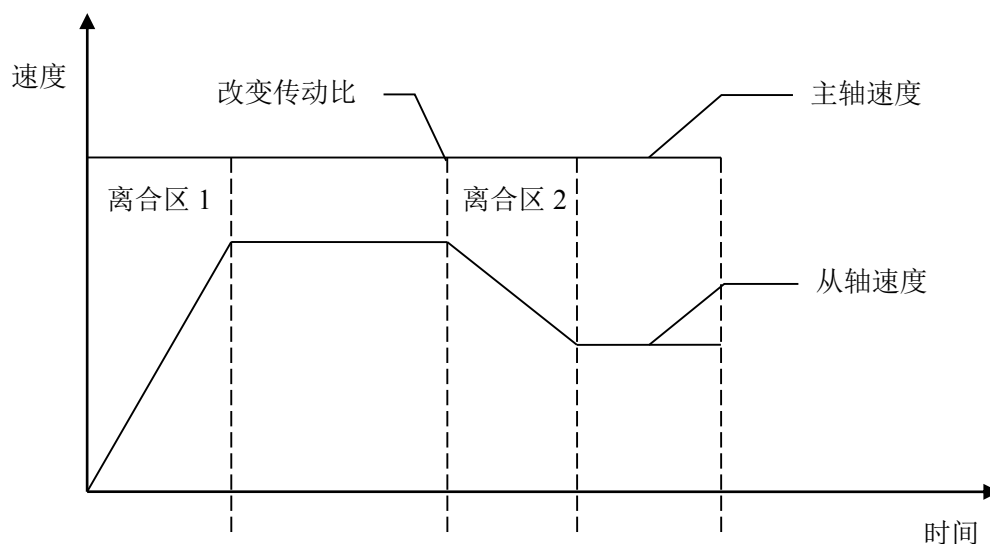


图 6-10 电子齿轮模式速度曲线

主轴匀速运动，从轴为电子齿轮模式，在离合区 1 从轴的传动比从 0 逐渐增大到设定传动比。当改变传动比时，在离合区 2 从轴的传动比逐渐变化到新的传动比。离合区越大，从轴传动比的变化过程越平稳。

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：

```
GT_SetGearRatio(slave, alpha, beta, slope);
```

6.4.3 例程

例程 6-5 电子齿轮跟随

该例程主轴为 Jog 模式，从轴为电子齿轮模式，传动比为 2: 1，主轴运动离合区位移后，从轴达到设定的传动比，如图 6-11 所示：

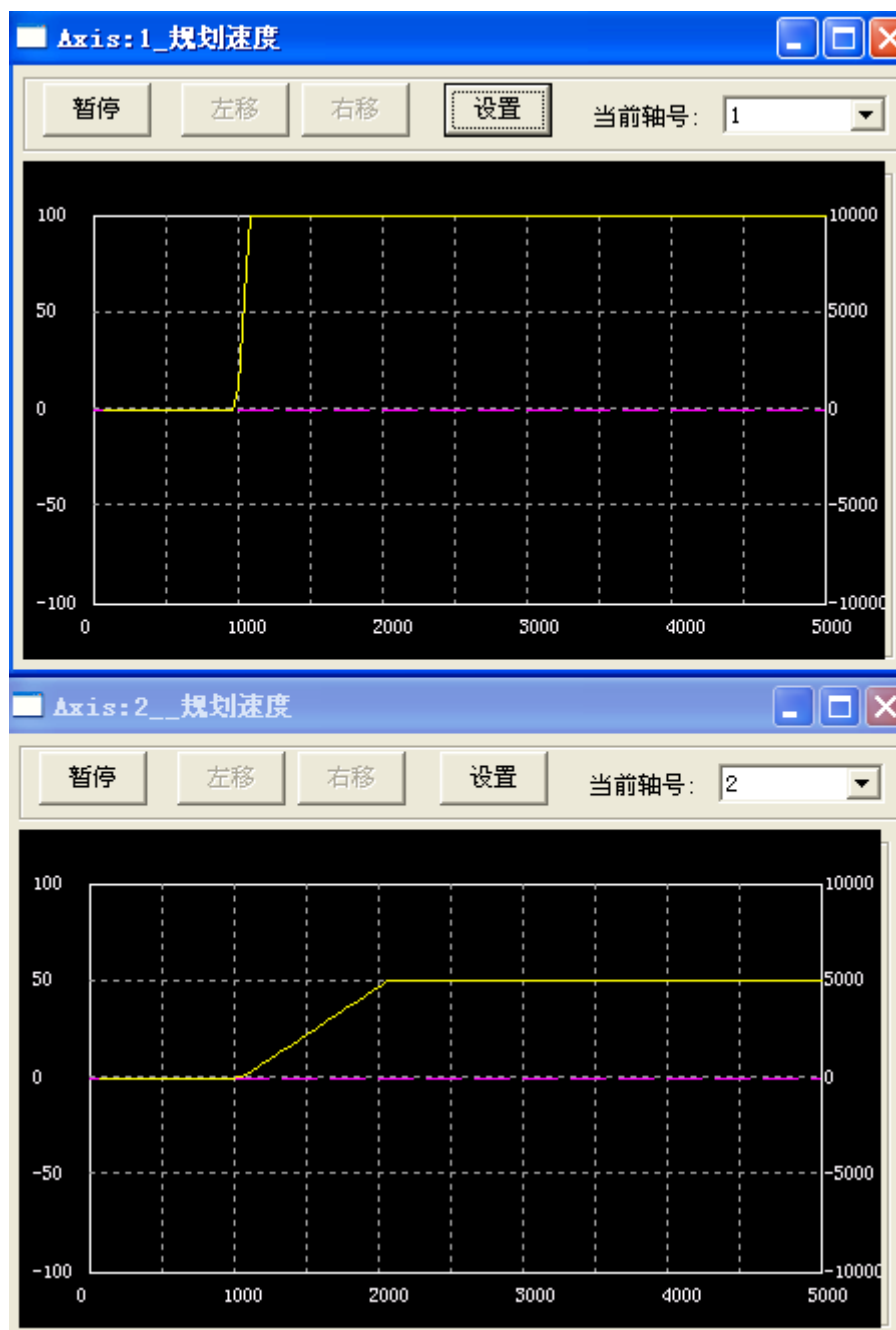


图 6-11 电子齿轮速度规划

```
#include "stdafx.h"
#include "conio.h"
#include "gts.h"
#define MASTER      1
#define SLAVE       2
int main(int argc, char* argv[])
```

```
{
    short rtn;
    double prfVel[8];
    TJogPrm jog;

    // 打开运动控制器
    rtn = GT_Open ();
    printf(" GT_Open ()=%d\n",rtn);

    // 复位运动控制器
    rtn = GT_Reset ();
    printf(" GT_Reset ()=%d\n",rtn);

    // 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
    rtn = GT_LoadConfig("test.cfg");
    printf(" GT_LoadConfig()=%d\n",rtn);

    // 清除各轴的报警和限位
    rtn = GT_ClrSts(1,8);
    printf(" GT_ClrSts()=%d\n",rtn);

    // 将主轴设为Jog模式
    rtn = GT_PrkJog(MASTER);
    printf(" GT_PrkJog()=%d\n",rtn);

    // 设置主轴运动参数
    rtn = GT_GetJogPrm(MASTER,&jog);
    printf(" GT_GetJogPrm()=%d\n",rtn);
    jog.acc = 1;
    rtn = GT_SetJogPrm(MASTER,&jog);
    printf(" GT_SetJogPrm()=%d\n",rtn);

    rtn = GT_SetVel(MASTER,100);
    printf(" GT_SetVel()=%d\n",rtn);

    // 启动主轴
    rtn = GT_Update(1<<(MASTER-1));
    printf(" GT_Update()=%d\n",rtn);

    // 将从轴设为Gear模式
    rtn = GT_PrjGear(SLAVE);
    printf(" GT_PrjGear()=%d\n",rtn);

    // 设置主轴，默认跟随主轴规划位置
    rtn = GT_SetGearMaster(SLAVE,MASTER);
```

```

printf(" GT_SetGearMaster(=%d\n",rtn);

// 设置从轴的传动比和离合区
rtn = GT_SetGearRatio(SLAVE,2,1,100000);
printf(" GT_SetGearRatio(=%d\n",rtn);

// 启动从轴
rtn = GT_GearStart(1<<(SLAVE-1));
printf(" GT_GearStart(=%d\n",rtn);

while(!kbhit())
{
    // 查询各轴的规划速度
    rtn = GT_GetPrfVel(1,prfVel,8);
    printf("master vel=%-10.2lf\tslave vel=%-10.2lf\n",prfVel[MASTER-1],prfVel[SLAVE-1]);
}
return 0;
}

```

6.5 Follow 模式

6.5.1 指令列表

表 6-6 设置 Follow 指令列表

指令	说明	页码
GT_PrFFollow	设置指定轴为 Follow 模式	200
GT_SetFollowMaster	设置跟随主轴	217
GT_GetFollowMaster	读取跟随主轴	181
GT_SetFollowLoop	设置循环次数	217
GT_GetFollowLoop	读取循环次数	180
GT_SetFollowEvent	设置 Follow 模式启动跟随条件	216
GT_GetFollowEvent	读取 Follow 模式启动跟随条件	180
GT_FollowSpace	查询 Follow 指定 FIFO 的剩余空间	167
GT_FollowData	向 Follow 指定 FIFO 增加数据	166
GT_FollowClear	清除 Follow 指定 FIFO 中的数据 运动状态下该指令无效	166
GT_FollowStart	启动 Follow 模式运动	167
GT_FollowSwitch	切换 Follow 所使用的 FIFO	168
GT_SetFollowMemory	设置 Follow 运动模式的缓存区大小	218
GT_GetFollowMemory	读取 Follow 运动模式的缓存区大小	181

6.5.2 重点说明

在很多应用中，2 轴或多轴之间需要保证位置同步和速度同步。如图 6-12 所示：

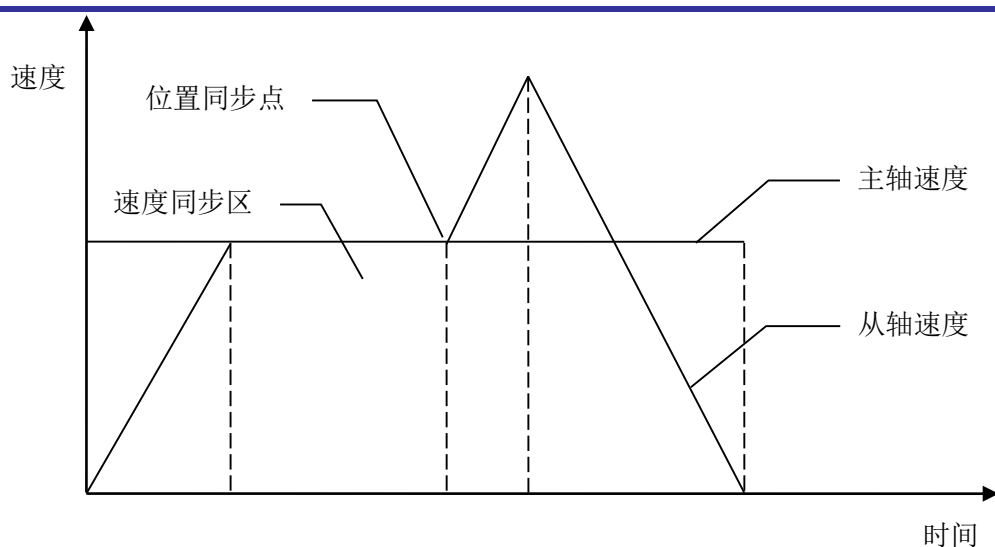


图 6-12 Follow 模式速度曲线

位置同步点表示主轴和从轴必须同时到达各自指定位置。

速度同步区表示主轴和从轴之间必须保持准确的速度比。

第 1 段是加速区，从轴逐渐加速，直至到达同步速度。

第 2 段是速度同步区，从轴和主轴保持设定的速度比，速度同步区结束时，主轴和从轴同时到达位置同步点。

第 3 段是加速区，从轴穿越位置同步点以后迅速加速，脱离速度同步区。

第 4 段是减速区，从轴逐渐减速到 0。

为了减少跟随滞后，从轴的轴号应当大于主轴的轴号。

(1) 数据段类型

Follow 模式的数据段有 4 种类型。

FOLLOW_SEGMENT_NORMAL 表示普通段，FIFO 中第 1 段的起点速度为 0，从第 2 段起每段的起点速度等于上一段的终点速度。

FOLLOW_SEGMENT_EVEN 表示匀速段，FIFO 中各段的段内速度保持不变。

FOLLOW_SEGMENT_STOP 表示停止段，该段的终点速度为 0，起点速度根据段内位移和段内时间计算得到，和上一段的终点速度无关。

FOLLOW_SEGMENT_CONTINUE 表示连续段，FIFO 中第一段的起点速度等于上个 FIFO 的终点速度，从第 2 段起每段的起点速度等于上一段的终点速度。

(2) 切换 FIFO

Follow 模式下有 2 个独立的 FIFO 用来保存数据。2 个 FIFO 之间可以在运动状态下进行切换。

如图 5-5-2 所示，从轴的运动规律需要从“速度曲线 1”变化到“速度曲线 3”，为了实现从轴速度的平滑过渡，增加了一个“速度曲线 2”的过渡状态。“速度曲线 2”的起始速度和“速度曲线 1”相等，“速度曲线 2”的终点速度和“速度曲线 3”相等。

为了实现 2 个 FIFO 之间的速度连续，在调用 `GT_FollowData` 指令时应当将数据类型设置为

FOLLOW_SEGMENT_CONTINUE。

首先将“速度曲线 2”传递到 FIFO2 中，然后调用 `GT_FollowSwitch` 指令切换工作 FIFO。

当 FIFO1 中的数据全部执行完毕以后才会切换到 FIFO2，并自动清空 FIFO1 中的数据。

当切换到 FIFO2 以后，立即将“速度曲线 3”传递到 FIFO1，然后调用 `GT_FollowSwitch` 指令切换工作 FIFO。

当 FIFO2 中的数据全部执行完毕以后才会切换到 FIFO1，并自动清空 FIFO2 中的数据。

至此从轴的运动规律从“速度曲线 1”经“速度曲线 2”过渡到“速度曲线 3”。

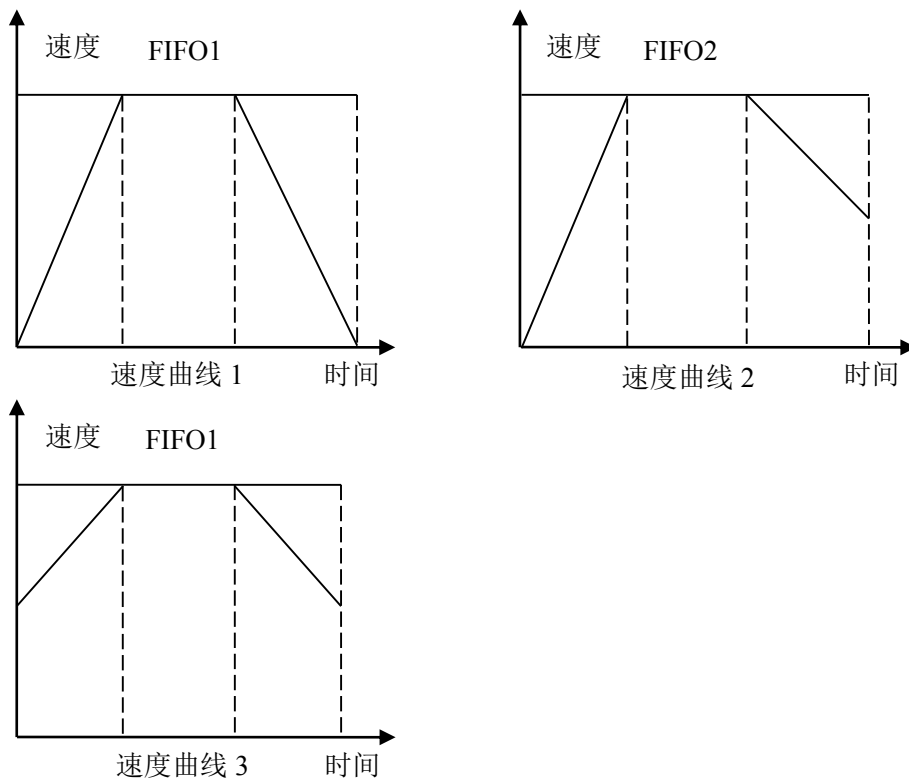


图 6-13 Follow 模式切换 FIFO

6.5.3 例程

(1) Follow 单 FIFO

例程 6-6 Follow 单 FIFO

该例程主轴为 Jog 模式，从轴为 Follow 模式。从轴的运动规律由 3 段组成，加速段跟随，匀速跟随，减速跟随。如图 6-14 所示：

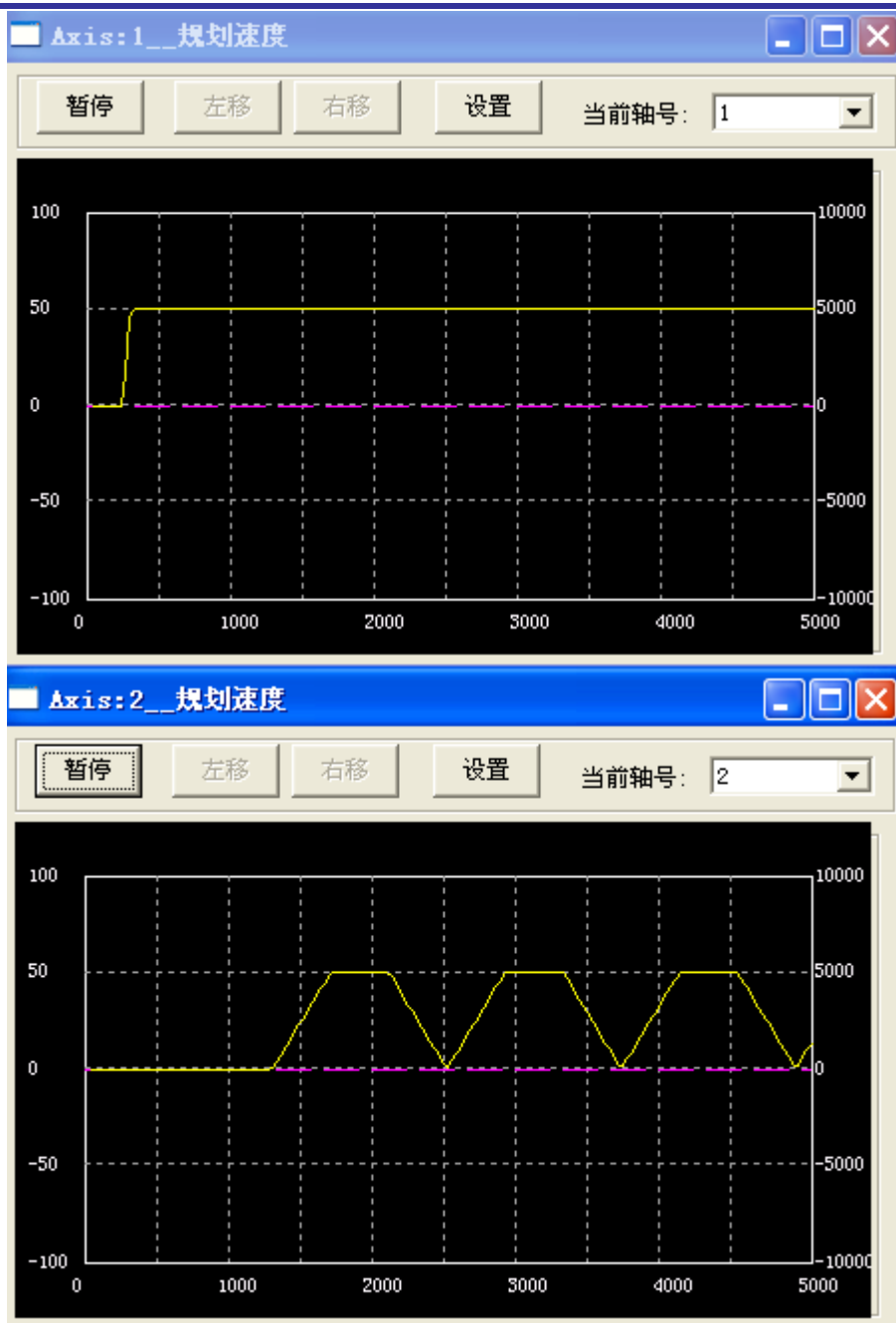


图 6-14 Follow 速度规划

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

```
#define MASTER      1
#define SLAVE       2
```

```
int main(int argc, char* argv[])
{
    short rtn;
    double prfVel[8];
    TJogPrm jog;
```

```
short space;
long masterPos;
double slavePos;
long loop;

// 打开运动控制器
rtn = GT_Open ();
printf(" GT_Open ()=%d\n",rtn);

// 复位运动控制器
rtn = GT_Reset ();
printf(" GT_Reset ()=%d\n",rtn);

// 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
rtn = GT_LoadConfig("test.cfg");
printf(" GT_LoadConfig()=%d\n",rtn);

// 清除各轴的报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts()=%d\n",rtn);

// 将主轴设为Jog模式
rtn = GT_PrkJog(MASTER);
printf(" GT_PrkJog()=%d\n",rtn);

// 设置主轴运动参数
rtn = GT_GetJogPrm(MASTER,&jog);
printf(" GT_GetJogPrm()=%d\n",rtn);
jog.acc = 1;
rtn = GT_SetJogPrm(MASTER,&jog);
printf(" GT_SetJogPrm()=%d\n",rtn);

rtn = GT_SetVel(MASTER,50);
printf(" GT_SetVel()=%d\n",rtn);

// 启动主轴
rtn = GT_Update(1<<(MASTER-1));
printf(" GT_Update()=%d\n",rtn);

// 将从轴设为Follow模式
rtn = GT_PrFfollow(SLAVE);
printf(" GT_PrFfollow()=%d\n",rtn);

// 清空从轴FIFO
rtn = GT_FollowClear(SLAVE);
```



```
printf(" GT_FollowClear()=%d\n",rtn);

// 设置主轴，默认跟随主轴规划位置
rtn = GT_SetFollowMaster(SLAVE,MASTER);
printf(" GT_SetFollowMaster()=%d\n",rtn);

// 查询Follow模式的剩余空间
rtn = GT_FollowSpace(SLAVE,&space);
printf(" GT_FollowSpace()=%d space=%d\n",rtn,space);

// 向FIFO中增加运动数据
masterPos = 20000;
slavePos = 10000;
rtn = GT_FollowData(SLAVE,masterPos,slavePos);
printf(" GT_FollowData()=%d\n",rtn);

// 查询Follow模式的剩余空间
rtn = GT_FollowSpace(SLAVE,&space);
printf(" GT_FollowSpace()=%d space=%d\n",rtn,space);

// 向FIFO中增加运动数据
masterPos += 20000;
slavePos += 20000;
rtn = GT_FollowData(SLAVE,masterPos,slavePos);
printf(" GT_FollowData()=%d\n",rtn);

// 查询Follow模式的剩余空间
rtn = GT_FollowSpace(SLAVE,&space);
printf(" GT_FollowSpace()=%d space=%d\n",rtn,space);

// 向FIFO中增加运动数据
masterPos += 20000;
slavePos += 10000;
rtn = GT_FollowData(SLAVE,masterPos,slavePos);
printf(" GT_FollowData()=%d\n",rtn);

// 设置循环次数为无限循环
rtn = GT_SetFollowLoop(SLAVE,0);
printf(" GT_SetFollowLoop()=%d\n",rtn);

// 设置启动跟随条件
rtn = GT_SetFollowEvent(SLAVE,FOLLOW_EVENT_PASS,1,50000);
printf(" GT_SetFollowEvent()=%d\n",rtn);

// 启动从轴Follow运动
rtn = GT_FollowStart(1<<(SLAVE-1));
```

```
printf(" GT_FollowStart(=%d\n",rtn);

while(!kbhit())
{
    // 查询各轴的规划速度
    rtn = GT_GetPrfVel(1,prfVel,8);
    // 查询循环次数
    rtn = GT_GetFollowLoop(SLAVE,&loop);

printf("master=%-10.2lf\tslave=%-10.2lf\tloop=%d\r",prfVel[MASTER-1],prfVel[SLAVE-1],loop);
}
return 0;
}
```

(2) Follow 双 FIFO 切换

例程 6-7 Follow 双 FIFO

该例程主轴为 Jog 模式，从轴为 Follow 模式，从轴在运动时更换跟随策略。如图 6-15 所示：

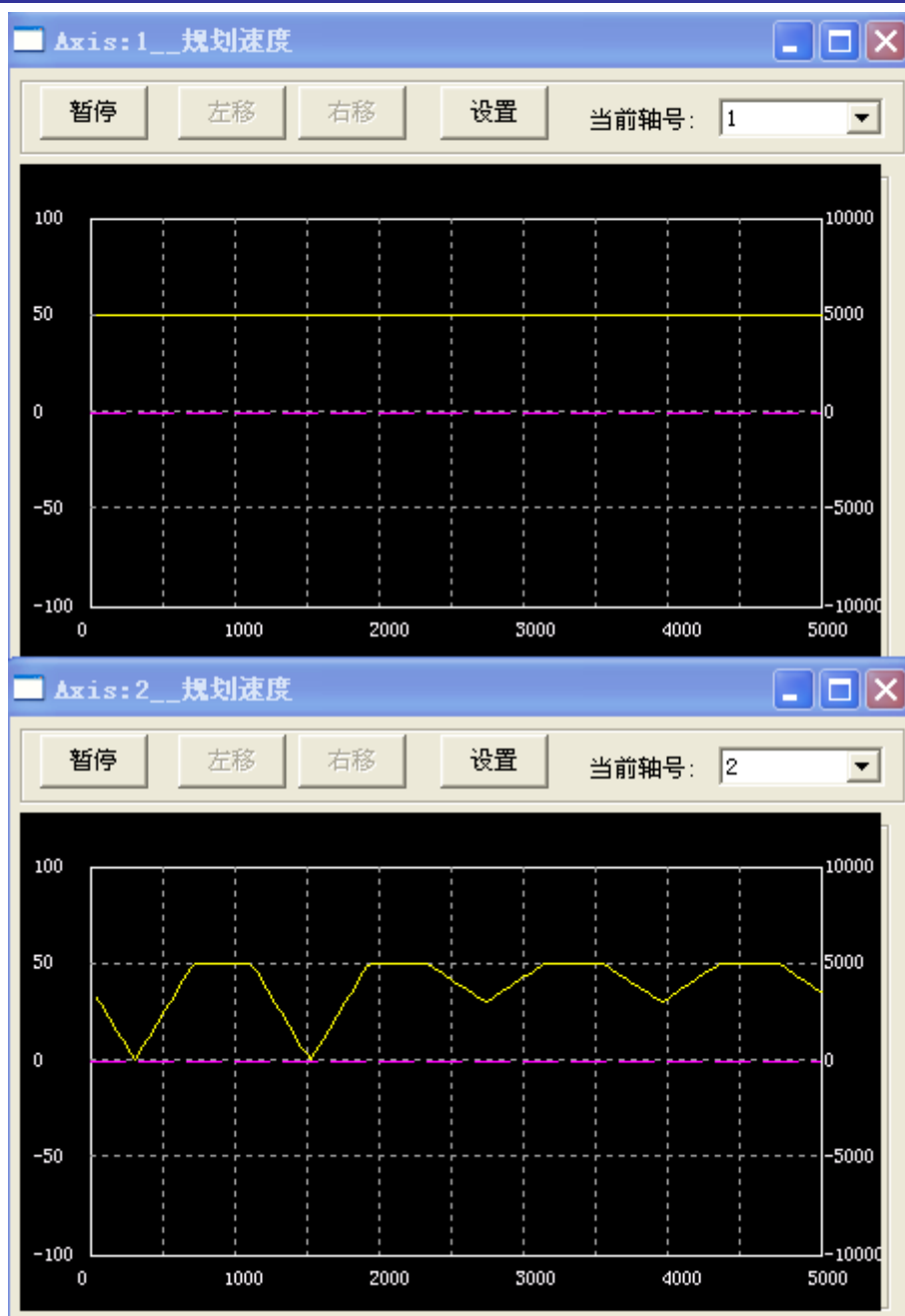


图 6-15 Follow 切换 FIFO

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

```
#define MASTER          1
#define SLAVE           2
```

```
#define STAGE_FIFO1     1
#define STAGE_TO_FIFO2 2
#define STAGE_TO_FIFO1 3
#define STAGE_END       4
```

```
int main(int argc, char* argv[])
{
    short rtn;
    double prfVel[8];
    TJogPrm jog;
    short space;
    long masterPos;
    double slavePos;
    short stage,pressKey;

    // 打开运动控制器
    rtn = GT_Open ();
    printf(" GT_Open ()=%d\n",rtn);

    // 复位运动控制器
    rtn = GT_Reset ();
    printf(" GT_Reset ()=%d\n",rtn);

    // 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
    rtn = GT_LoadConfig("test.cfg");
    printf(" GT_LoadConfig()=%d\n",rtn);

    // 清除各轴的报警和限位
    rtn = GT_ClrSts(1,8);
    printf(" GT_ClrSts()=%d\n",rtn);

    // 将主轴设为Jog模式
    rtn = GT_PrkJog(MASTER);
    printf(" GT_PrkJog()=%d\n",rtn);

    // 设置主轴运动参数
    rtn = GT_GetJogPrm(MASTER,&jog);
    printf(" GT_GetJogPrm()=%d\n",rtn);
    jog.acc = 1;
    rtn = GT_SetJogPrm(MASTER,&jog);
    printf(" GT_SetJogPrm()=%d\n",rtn);

    rtn = GT_SetVel(MASTER,50);
    printf(" GT_SetVel()=%d\n",rtn);

    // 启动主轴
    rtn = GT_Update(1<<(MASTER-1));
    printf(" GT_Update()=%d\n",rtn);

    // 将从轴设为Follow模式
```

```
rtn = GT_PrFFollow(SLAVE);
printf(" GT_PrFFollow()=%d\n",rtn);

// 清空从轴FIFO1
rtn = GT_FollowClear(SLAVE,0);
printf(" GT_FollowClear()=%d\n",rtn);

// 清空从轴FIFO2
rtn = GT_FollowClear(SLAVE,1);
printf(" GT_FollowClear()=%d\n",rtn);

// 设置主轴，默认跟随主轴规划位置
rtn = GT_SetFollowMaster(SLAVE,MASTER);
printf(" GT_SetFollowMaster()=%d\n",rtn);

// 查询Follow模式FIFO1的剩余空间
rtn = GT_FollowSpace(SLAVE,&space,0);
printf(" GT_FollowSpace()=%d space=%d\n",rtn,space);

// 向FIFO1中增加运动数据
masterPos = 20000;
slavePos = 10000;
rtn = GT_FollowData(SLAVE,masterPos,slavePos,FOLLOW_SEGMENT_NORMAL,0);
printf(" GT_FollowData()=%d\n",rtn);

// 查询Follow模式FIFO1的剩余空间
rtn = GT_FollowSpace(SLAVE,&space,0);
printf(" GT_FollowSpace()=%d space=%d\n",rtn,space);

// 向FIFO1中增加运动数据
masterPos += 20000;
slavePos += 20000;
rtn = GT_FollowData(SLAVE,masterPos,slavePos,FOLLOW_SEGMENT_NORMAL,0);
printf(" GT_FollowData()=%d\n",rtn);

// 查询Follow模式FIFO1的剩余空间
rtn = GT_FollowSpace(SLAVE,&space,0);
printf(" GT_FollowSpace()=%d space=%d\n",rtn,space);

// 向FIFO1中增加运动数据
masterPos += 20000;
slavePos += 10000;
rtn = GT_FollowData(SLAVE,masterPos,slavePos,FOLLOW_SEGMENT_NORMAL,0);
printf(" GT_FollowData()=%d\n",rtn);

// 设置从轴循环次数为无限循环
```

```
rtn = GT_SetFollowLoop(SLAVE,0);
printf(" GT_SetFollowLoop()=%d\n",rtn);

// 设置启动跟随条件
rtn = GT_SetFollowEvent(SLAVE,FOLLOW_EVENT_START,1);
printf(" GT_SetFollowEvent()=%d\n",rtn);

// 启动从轴Follow运动
rtn = GT_FollowStart(1<<(SLAVE-1));
printf(" GT_FollowStart()=%d\n",rtn);

stage = STAGE_FIFO1;
pressKey = 0;

while(1)
{
    if(kbhit())
    {
        getch();
        pressKey = 1;
    }

    if( STAGE_FIFO1 == stage )
    {
        if( 1 == pressKey )
        {
            pressKey = 0;
            stage = STAGE_TO_FIFO2;

            // 向FIFO2中发送过渡数据
            rtn = GT_FollowClear(SLAVE,1);

            masterPos = 20000;
            slavePos = 10000;
            rtn = GT_FollowData(SLAVE,masterPos,slavePos,FOLLOW_SEGMENT_CONTINUE,1);

            masterPos += 20000;
            slavePos += 20000;
            rtn = GT_FollowData(SLAVE,masterPos,slavePos,FOLLOW_SEGMENT_NORMAL,1);

            masterPos += 20000;
            slavePos += 16000;
            rtn = GT_FollowData(SLAVE,masterPos,slavePos,FOLLOW_SEGMENT_NORMAL,1);
```

```

        // 切换FIFO
        // 当前工作FIFO遍历完以后才会切换FIFO
        rtn = GT_FollowSwitch (1<<(SLAVE-1));
    }
}

if( STAGE_TO_FIFO2 == stage )
{
    // 查询FIFO1的剩余空间
    GT_FollowSpace(SLAVE,&space,0);

    // 如果FIFO1被清空, 说明已经切换到FIFO2
    if( 16 == space )
    {
        stage = STAGE_TO_FIFO1;

        masterPos = 20000;
        slavePos = 16000;
        rtn = GT_FollowData(SLAVE,masterPos,slavePos,FOLLOW_SEGMENT_CONTINUE,0);

        masterPos += 20000;
        slavePos += 20000;
        rtn = GT_FollowData(SLAVE,masterPos,slavePos,FOLLOW_SEGMENT_NORMAL,0);

        masterPos += 20000;
        slavePos += 16000;
        rtn = GT_FollowData(SLAVE,masterPos,slavePos,FOLLOW_SEGMENT_NORMAL,0);

        // 切换FIFO
        // 当前工作FIFO遍历完以后才会切换FIFO
        rtn = GT_FollowSwitch (1<<(SLAVE-1));
    }
}

if( STAGE_TO_FIFO1 == stage )
{
    // 查询FIFO2的剩余空间
    GT_FollowSpace(SLAVE,&space,1);

    // 如果FIFO2被清空, 说明已经切换到FIFO1
    if( 16 == space )
    {
        stage = STAGE_END;
    }
}

```

```

    }
}

// 查询各轴的规划速度
rtn = GT_GetPrfVel(1,prfVel,8);

printf("master=%-10.2lf\tslave=%-10.2lfr",prfVel[MASTER-1],prfVel[SLAVE-1]);

if( STAGE_END == stage )
{
    if( 1 == pressKey )
    {
        pressKey = 0;
        break;
    }
}
return 0;
}
}

```

6.6 插补运动模式

插补运动模式可以实现多轴的协调运动，从而完成一定的运动轨迹。该插补运动模式具有以下一些功能，可以实现直线插补和圆弧插补；可以同时有两个坐标系进行插补运动；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；具有缓存区延时和缓存区数字量输出的功能；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。

6.6.1 指令列表

表 6-7 设置插补运动指令列表

指令	说明	页码
GT_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系	212
GT_GetCrdPrm	查询坐标系参数	176
GT_CrdData	向插补缓存区增加插补数据	162
GT_LnXY	缓存区指令，两维直线插补	196
GT_LnXYZ	缓存区指令，三维直线插补	197
GT_LnXYZA	缓存区指令，四维直线插补	197
SGT_LnXYG0	缓存区指令，两维直线插补(终点速度始终为 0)	196
GT_LnXYZG0	缓存区指令，三维直线插补(终点速度始终为 0)	198
GT_LnXYZAG0	缓存区指令，四维直线插补(终点速度始终为 0)	198
GT_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)	152
GT_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)	151
GT_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)	153
GT_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)	153
GT_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)	155

GT_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)	154
GT_BufIO	缓存区指令, 缓存区内数字量 IO 输出设置指令	158
GT_BufDelay	缓存区指令, 缓存区内延时设置指令	157
GT_BufDA	缓存区指令, 缓存区内输出 DA 值	156
GT_BufLmtsOn	缓存区指令, 缓存区内有效限位开关	159
GT_BufLmtsOff	缓存区指令, 缓存区内无效限位开关	158
GT_BufSetStoplo	缓存区指令, 缓存区内设置 axis 的停止 IO 信息	160
GT_BufMove	缓存区指令, 实现刀向跟随功能, 启动某个轴点位运动	159
GT_BufGear	缓存区指令, 实现刀向跟随功能, 启动某个轴跟随运动	157
GT_CrdSpace	查询插补缓存区剩余空间	162
GT_CrdClear	清除插补缓存区内的插补数据	162
GT_CrdStart	启动插补运动	163
GT_CrdStatus	查询插补运动坐标系状态	163
GT_SetUserSegNum	缓存区指令, 设置自定义插补段段号	226
GT_GetUserSegNum	读取自定义插补段段号	190
GT_GetRemainderSegNum	读取未完成的插补段段数	188
GT_SetOverride	设置插补运动目标合成速度倍率	221
GT_SetCrdStopDec	设置插补运动平滑停止、急停合成加速度	213
GT_GetCrdStopDec	查询插补运动平滑停止、急停合成加速度	176
GT_GetCrdPos	查询该坐标系的当前坐标位置值	175
GT_GetCrdVel	查询该坐标系的合成速度值	176
GT_InitLookAhead	初始化插补前瞻缓存区	194

6.6.2 重点说明

(1) 建立坐标系

运动控制器初始状态下, 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, 当需要进行插补运动时, 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。

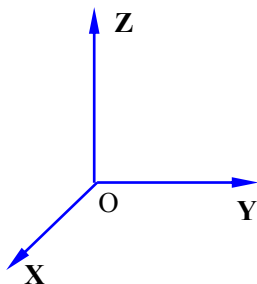


图 6-16 右手坐标系

用户通过调用 `GT_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 `GT_SetCrdPrm` 指令时, 所映射的各规划轴必须处于静止状态。

建立坐标系例程如下:

例程 6-8 建立坐标系

```

.....
short rtn;
TCrdPrm crdPrm;           // 定义坐标系结构体变量
memset(&crdPrm,0,sizeof(crdPrm)); // 将变量初始化为全
crdPrm.dimension=2;       // 坐标系为二维坐标系
crdPrm.synVelMax=500;     // 最大合成速度: pulse/ms
crdPrm.synAccMax=1;      // 最大加速度: pulse/ms^2
crdPrm.evenTime = 50;    // 平滑时间: ms
crdPrm.profile[0] = 1;   // 规划器对应到X轴
crdPrm.profile[1] = 2;   // 规划器对应到Y轴
crdPrm.setOriginFlag = 1; // 需要明确指定坐标系原点的规划位置
crdPrm.originPos[0] = 100;
crdPrm.originPos[1] = 100;
rtn = GT_SetCrdPrm(1,&crdPrm); // 建立号坐标系, 设置坐标系参数
.....

```

例程说明:

dimension: 表示所建立的坐标系的维数, 取值范围为[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 6-17 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 6-18 所示, 减小了加速度突变的冲击。

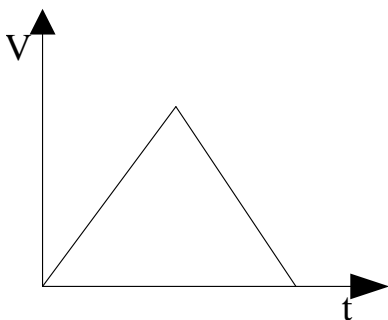


图 6-17 evenTime=0

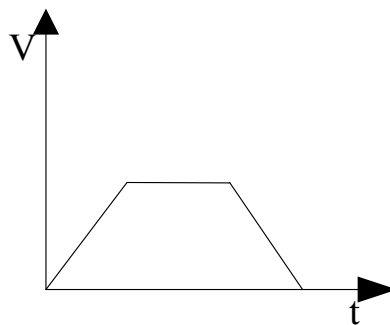


图 6-18 evenTime>0

profile[x]: 规划轴与坐标轴之间的对应关系。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。

setOriginFlag: 表示是否需要指定坐标系原点的规划位置值, 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。如果该参数为 0, 则加工坐标系的原点在当前的规划位置上, 如果该参数为 1, 则加工坐标系的原点在用户指定的规划位置上, 通过 **originPos** 来指定。

originPos[x]: 加工坐标系原点的规划位置值, 即相对于机床坐标系的偏移量。建立的加工坐标系如图 6-19 所示。

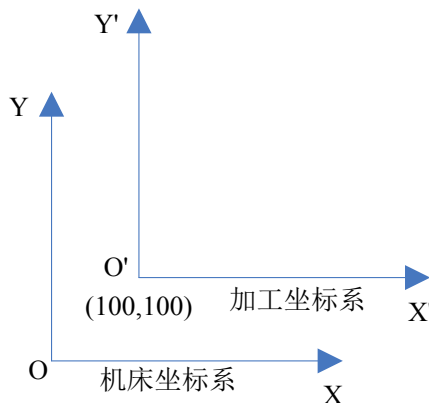


图 6-19 加工坐标系偏移量示意图

(2) 坐标系运动

坐标系运动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。

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。FIFO 支持动态管理的方式, 即插补数据运动完成之后, 其所占用的缓存区空间将会被释放, 用户可以继续传递新的插补数据, 通过这种方式, 就可以支持大于 4096 段的用户插补数据。

坐标系运动的直线插补例程如下:

例程 6-9 直线插补

```
.....
short rtn;
short run; // 定义坐标系运动状态查询变量
long segment; // 定义坐标系运动完成段查询变量
rtn = GT_CrdClear(1,0); // 清除坐标系的FIFO0中的数据
// 第一段插补数据
rtn = GT_LnXY(1,200000,0,100,0.1,0,0); // 向坐标系的FIFO0传递直线插补数据
// 该插补段的终点坐标(200000,0)
// 该插补段的目标速度: pulse/ms
// 插补段的加速度: .1pulse/ms^2
// 终点速度为
// 第二段插补数据
rtn = GT_LnXY(1,100000,173205,100,0.1,0,0);
```

```

// 缓存区数字量输出
rtn = GT_BufIO (1,MC_GPO,0xffff,0x55,0); // 数字量输出类型: 通用输出
// bit0~bit15全部都输出
// 输出的数值为x55
// 第三段插补数据
rtn = GT_LnXY (1,-100000,173205,100,0.1,0,0);
// 缓存区数字量输出
rtn = GT_BufIO (1,MC_GPO,0xffff,0xaa,0);
// 第四段插补数据
rtn = GT_LnXY (1,-200000,0,100,0.1,0,0);
// 缓存区延时指令
rtn = GT_BufDelay (1,400,0); // 该处延时ms
// 第五段插补数据
rtn = GT_LnXY (1,-100000,-173205,100,0.1,0,0);
// 缓存区数字量输出
rtn = GT_BufIO (1,MC_GPO,0xffff,0x55,0);
// 缓存区延时指令
rtn = GT_BufDelay (1,100,0);
// 第六段插补数据
rtn = GT_LnXY (1,100000,-173205,100,0.1,0,0);
// 第七段插补数据
rtn = GT_LnXY (1,200000,0,100,0.1,0,0);
rtn = GT_CrdSpace(1,&space,0); // 查询坐标系的FIFO0所剩余的空间
rtn = GT_CrdStart(1,0); // 启动坐标系的FIFO0的插补运动
// 等待运动完成
rtn = GT_CrdStatus(1,&run,&segment,0); // 查询坐标系的FIFO的插补运动状态
do
{
    rtn = GT_CrdStatus(1,&run,&segment,0);
}while(run == 1); // 坐标系在运动,查询到的run的值为
.....

```

例程说明:

通过 `GT_LnXY` 指令向插补缓存区传递数据, 在该指令中包含了终点坐标、加速度、目标速度, 还可以调用 `GT_BufIO` 指令在缓存区中进行数字量输出, 调用 `GT_BufDelay` 指令在缓存区中进行延时操作。该例程共向插补缓存区传递了 7 段运动数据, 例程的运行结果如图 6-20 所示, 是一个六边形。其中, 在运动过程中进行了缓存区延时和数字量输出的操作。

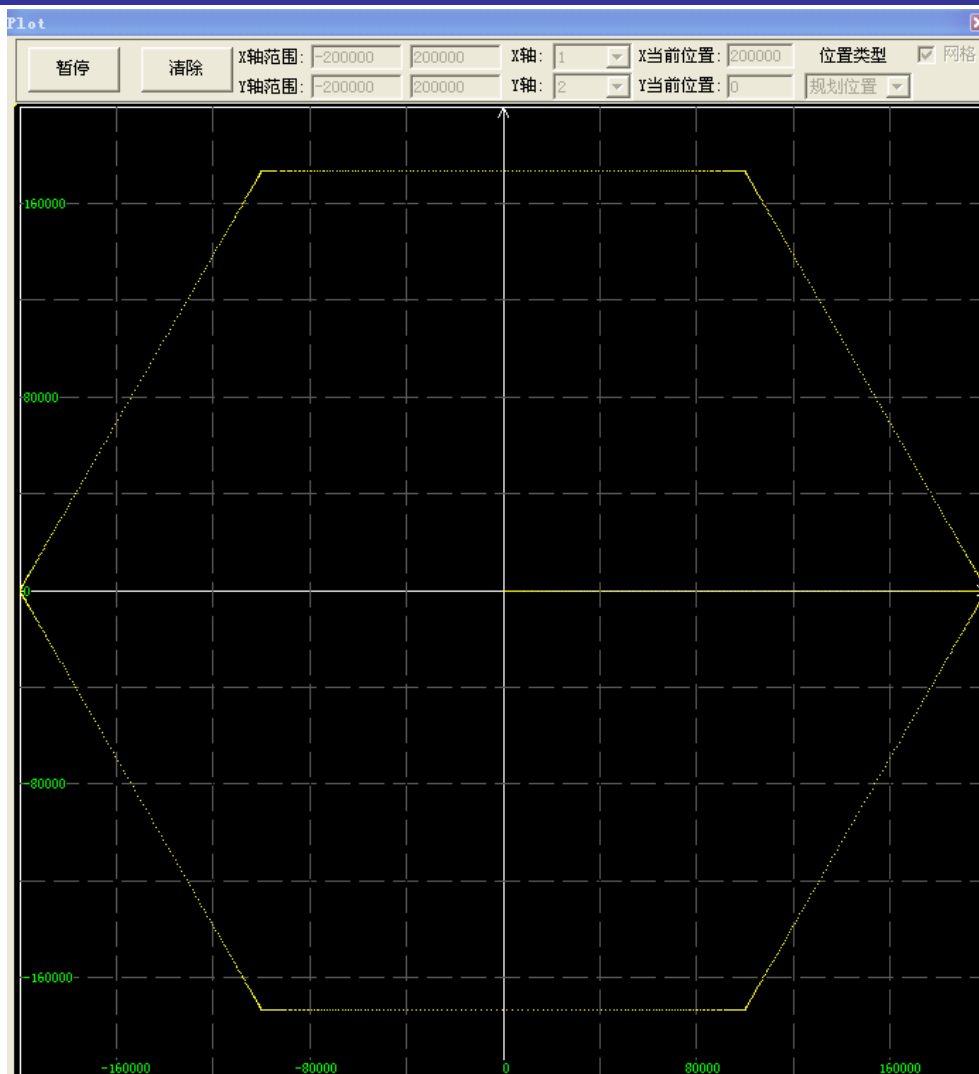


图 6-20 直线插补例程运动结果

`GT_CrdStatus` 指令可以用来查询坐标系指定 FIFO 的运动状态(运动或静止),以及当前已经完成的插补运动的段数,该段数从建立坐标系之后的第一个 `GT_CrdStart` 的调用之后开始累加,直到销毁坐标系或者调用 `GT_CrdClear` 指令时才被清零。

坐标系运动的圆弧插补例程如下:

例程 6-10 圆弧插补

```

.....
short rtn;
short run; // 定义坐标系运动状态查询变量
long segment; // 定义坐标系运动完成段查询变量
rtn = GT_CrdClear(1,0); // 清除坐标系的FIFO0中的数据

// 直线插补数据
rtn = GT_LnXY (1,200000,0,100,0.1,0,0);

// 圆弧插补数据
rtn = GT_ArcXYC(1,200000,0,-100000,0,0,100,0.1, 0,0);
// 使用圆心描述方法描述一个整圆

```

```

// 圆心坐标(100000,0)
// 终点坐标与起点坐标重合(200000,0)
// 顺时针圆弧
// 该插补段的目标速度: pulse/ms
// 插补段的加速度: .1pulse/ms^2
// 终点速度为
// 圆弧插补数据
rtn = GT_ArcXYR (1,0,200000,200000,1,100,0.1,0,0);
// 使用半径描述方法描述一个/4圆弧
// 终点坐标为: (0,200000)
// 半径:
// 逆时针圆弧
rtn = GT_LnXY (1,0,0,100,0.1,0,0);           // 回到原点位置
rtn = GT_CrdSpace(1,&space,0);                // 查询坐标系的FIFO0所剩余的空间
rtn = GT_CrdStart(1,0);                       // 启动坐标系的FIFO0的插补运动
// 等待运动完成
rtn = GT_CrdStatus(1,&run,&segment,0);        // 查询坐标系的FIFO的插补运动状态
do
{
    rtn = GT_CrdStatus(1,&run,&segment,0);
}while(run == 1);                             // 坐标系在运动,查询到的 run 的值为
.....

```

圆弧插补例程的运行结果如图 6-21 所示。

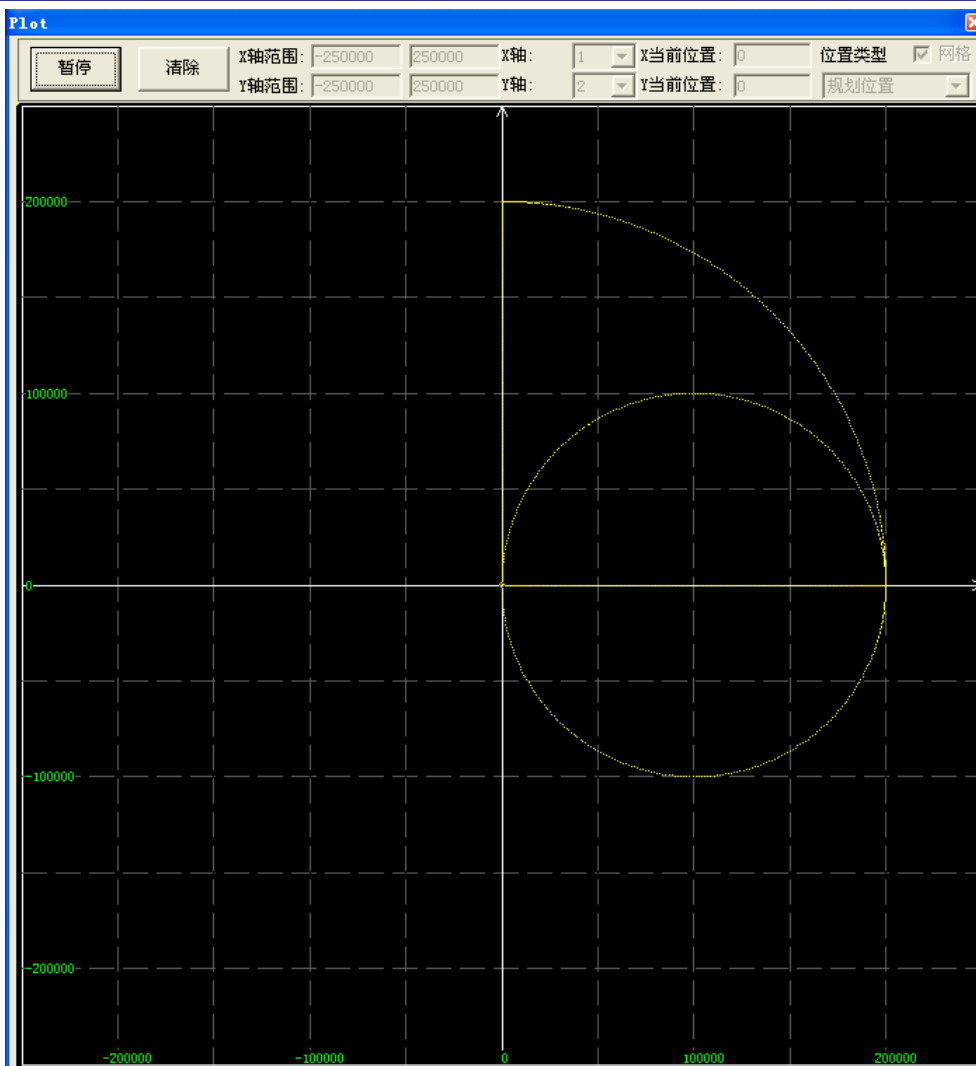


图 6-21 圆弧插补例程运动结果

该控制器支持在 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。

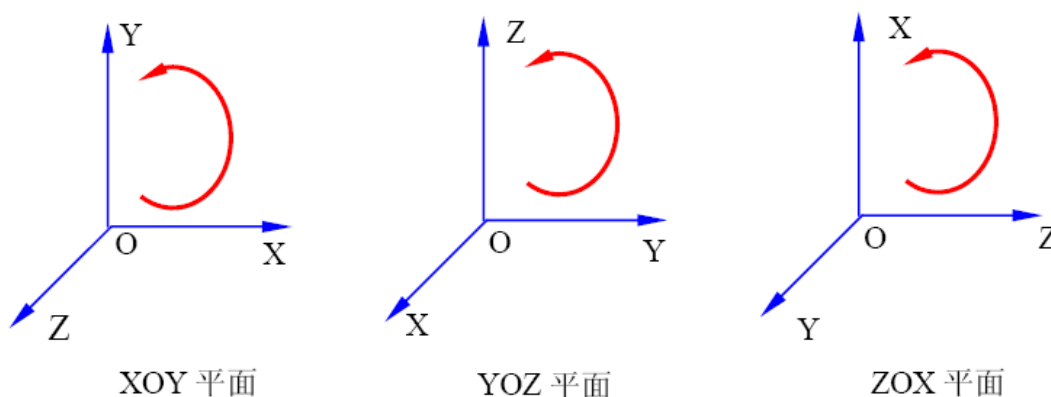


图 6-22 圆弧插补逆时针方向

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

半径描述方法，即调用指令 `GT_ArcXYR`、`GT_ArcYZR`、`GT_ArcZXR` 对圆弧进行描述，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 $\leq 180^\circ$ ，负值表示圆弧的旋转角度 $> 180^\circ$ ，如图 6-23 所示，半径描述方法无法描述 360° 的整圆。

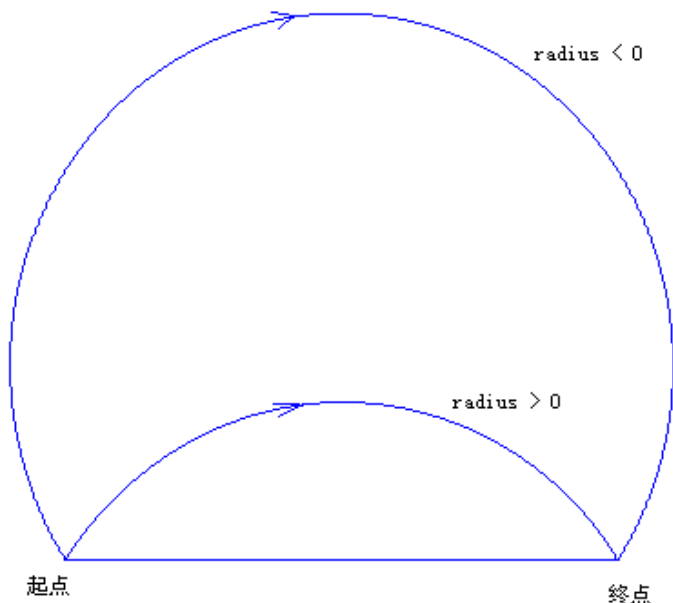


图 6-23 半径取正值/负值圆弧插补示意图

b) 圆心描述方法

圆心描述方法，即调用指令 `GT_ArcXYC`、`GT_ArcYZC`、`GT_ArcZXC` 对圆弧进行描述，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 6-24 所示：

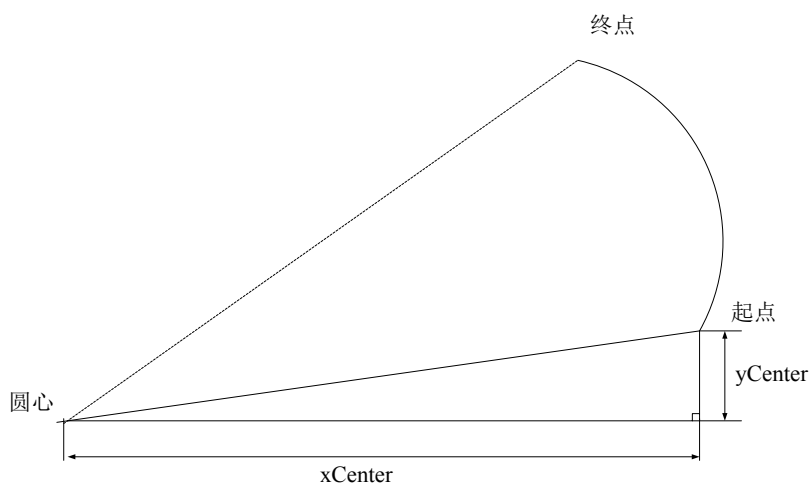


图 6-24 圆心表示方法示意图

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 (x_{Start}, y_{Start}) ，用户设置的圆心参数为 (x_{Center}, y_{Center}) ，则圆心坐标值为 $(x_{Start} + x_{Center}, y_{Start} + y_{Center})$ 。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。

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)。

(3) 前瞻预处理

小线段插补加工的特点：为保证刀具与加工工件接触面的光顺，尽量保持轨迹运动过程中切向速度的恒定，同时又必须保证一定的轨迹加工精度。

观察图 6-25，可以了解该图形中每条线段终点处都是拐点(轨迹特征发生明显改变的点)，且必须降速，但是否应该降到 0，需要根据线段长度、速度、加速度以及拐点速度变化限值等与加工工艺相关的参数来计算出各段的终点速度。

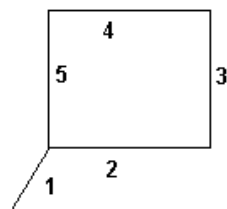


图 6-25 X-Y 平面多段轨迹图形

观察图 6-26，可以了解对于以小线段拟合曲线轨迹的线段组合，应在加工过程中尽量保持切向速度的恒定，但又必须保证在拐点(第 8 点)处将速度降到一个合理的值(合理的终点速度)，以保证加工执行机构(机械本体和电机)能够承受由于拐点处轨迹特征发生变化而带来的速度变化量。

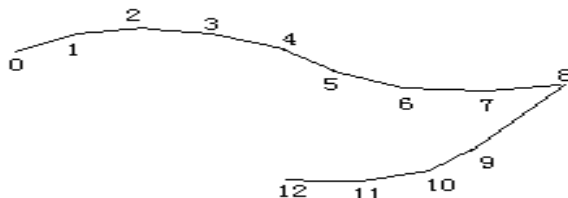


图 6-26 X-Y 平面小线段轨迹图形

为了解决高速和高精度这对矛盾，运动控制器对运动过程中的速度规划采用基于前瞻预处理的处理方式。

用户可根据本身机床的工艺特征参数(脉冲当量、目标速度、最大加速度、允许拐弯时间等)，调用运动控制器提供的前瞻预处理模块，给出每段的终点速度，运动控制器则严格按照每段终点速度进行加减速控制。运动控制器将这套实现速度规划预处理功能的指令称为前瞻预处理(又称 LookAhead)指令。

从图 6-27 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：

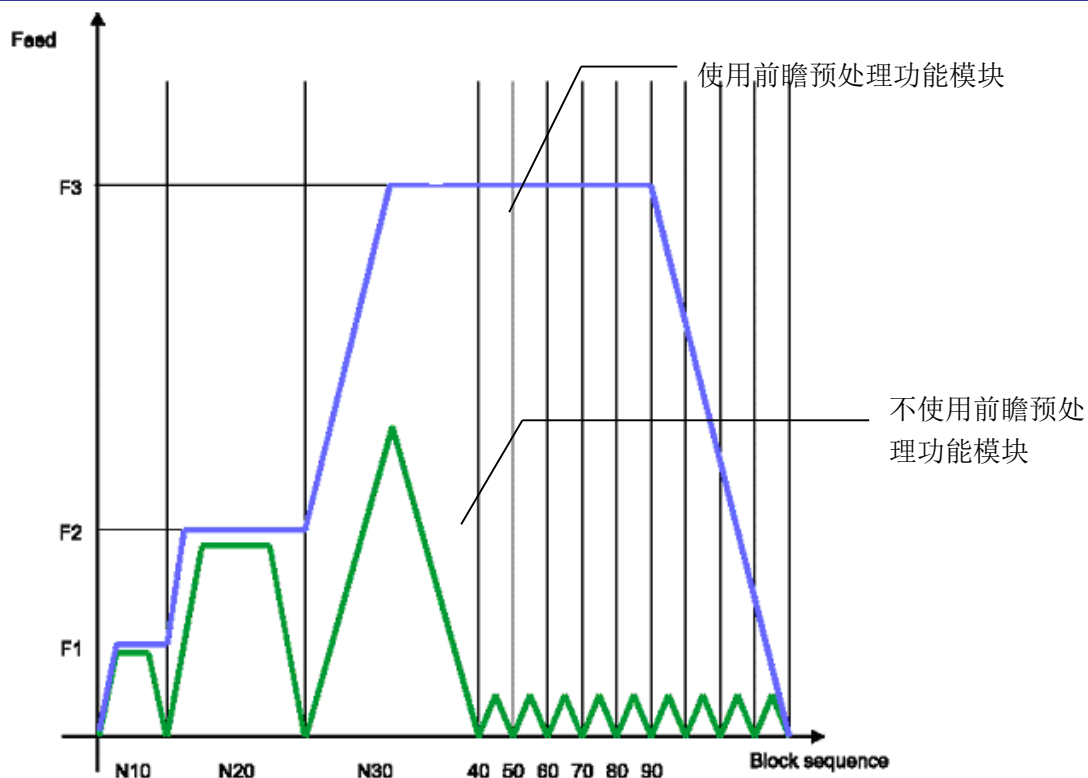


图 6-27 使用和不使用前瞻预处理功能模块的速度曲线对比图

使用前瞻预处理的例程如下：

例程 6-11 前瞻预处理

.....

```

short rtn;
int i;

TCrdData crdDataSend;
TCrdData crdData[200];           // 定义前瞻缓存区内存区
long posTest[2];
rtn = GT_InitLookAhead(1,0,5,1,200,crdData); // 初始化坐标系的FIFO0的前瞻模块
// 压插补数据：小线段加工
posTest[0] = 0;
posTest[1] = 0;
for(i=0;i<300;++i)
{
    rtn = GT_LnXY (1,8000+posTest[0],9000+posTest[1],100,0.8,0,0);
    posTest[0] += 1600;
    posTest[1] += 1852;
}
rtn = GT_CrdData(1,NULL,0);      // 将前瞻缓存区中的数据压入控制器
rtn = GT_CrdStart(1,0);
.....

```

例程说明:

拐弯时间(T): `GT_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): `GT_InitLookAhead` 指令的第四个参数, 单位: pulse/(ms*ms)。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。

前瞻缓存区大小和前瞻缓存区内存区指针: 该前瞻模块采用用户提供前瞻缓存区内存区的方式, 因此, 用户可以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, 前瞻缓存区越大, 占用的内存区就越大。用户需要先定义一个插补数据数组变量, 申请一定的内存区, 然后通过 `GT_InitLookAhead` 指令把内存区的指针传递给运动控制器的前瞻模块, 在进行前瞻预处理的过程中, 用户不能再对该内存区进行任何操作, 否则将会破坏前瞻缓存区中的数据, 造成数据的错误。

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 `GT_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 `GT_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。

如果不使用前瞻预处理功能, 则运动控制器不会对插补段的终点速度和目标速度进行优化, 运动控制器将严格按照用户指定的目标速度和终点速度进行速度规划。如果用户调用 `SGT_LnXYG0`、`GT_LnXYZG0` 和 `GT_LnXYZAG0` 指令, 则该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。如果调用其他插补运动指令(包括直线插补和圆弧插补指令), 则用户可以指定插补段的目标速度和终点速度, 运动控制器会按照用户指定的目标速度和终点速度进行速度规划。

使用前瞻预处理功能之后, 控制器会根据用户设置的参数将每段的终点速度设置为一个合理的值, 该值不一定为 0, 如果用户的工艺要求某段插补数据的终点速度必须为 0, 则需要调用 `SGT_LnXYG0`、`GT_LnXYZG0` 或者 `GT_LnXYZAG0`, 该指令会将该直线插补段的终点速度设置为 0。如果调用其他插补运动指令(包括直线插补和圆弧插补指令), 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。另外, 如果某段插补运动数据与下段插补运动数据之间存在缓存区延时, 则该段插补运动的终点速度会被设置为 0。

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)。

如果没有进行前瞻预处理的合成速度曲线如图 6-28 所示, 合成速度在不停的变化。进行前瞻预处理的合成速度曲线如图 6-29 所示, 由于例程中的插补运动都是在同一条直线上, 所以速度可以一直维持在目标速度, 大大提高了加工效率。

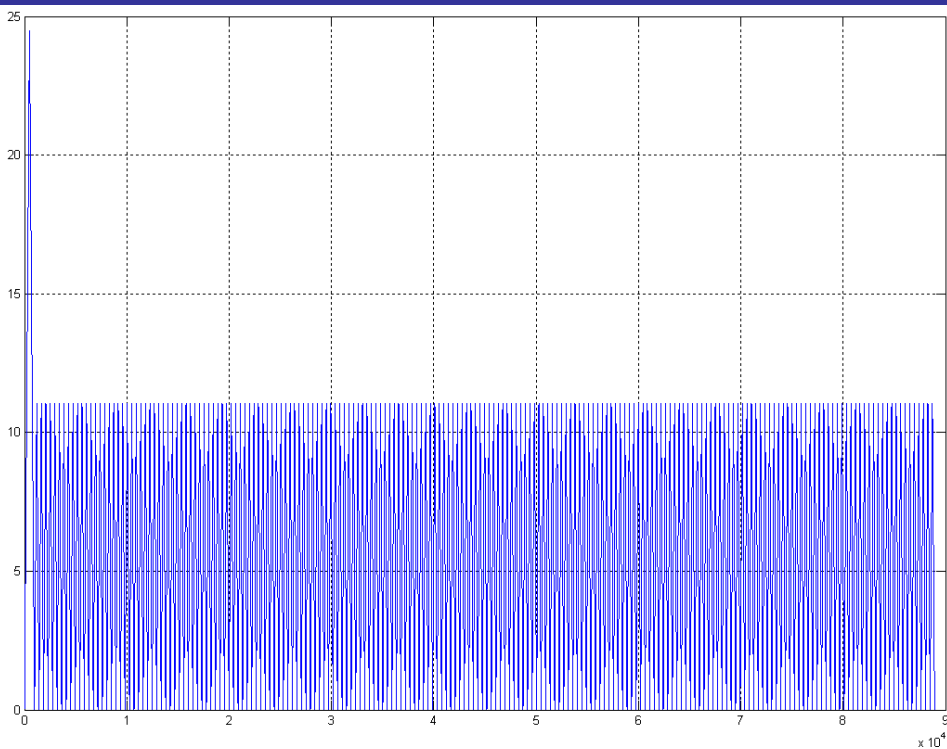


图 6-28 没有进行前瞻预处理的合成速度曲线

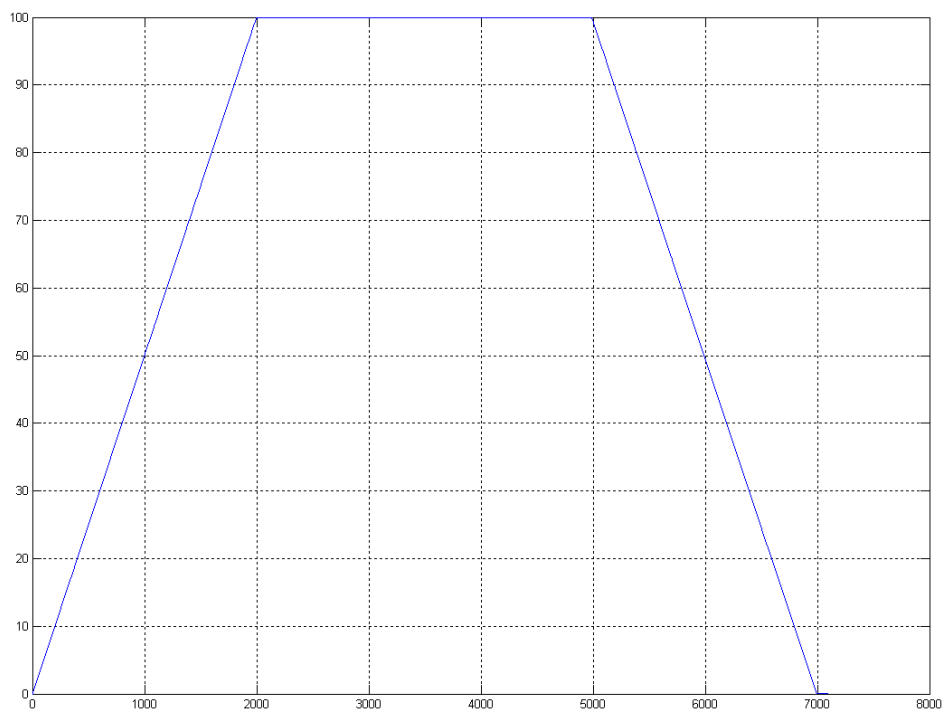


图 6-29 进行了前瞻预处理后的合成速度曲线

(4) 缓存区暂停、恢复

在缓存区插补的过程中，用户可能会需要暂停加工，查看加工效果，或者在暂停之后进行其他操作，如换刀等，该运动控制器支持上述操作过程。为了实现上述的操作过程，每个坐标系提供了两个插补缓存区：FIFO0 和 FIFO1。两个缓存区都有 4096 段插补缓存区，并且可以独立设置各自的前瞻预处理缓存区。

其中，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，中断后可以辅助 FIFO1 的插补运动，辅助 FIFO1 的插补运动完成后，FIFO0 可从断点处继续恢复原来的运动。

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，在暂停、恢复之间不可以进行 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。

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 `GT_CrdStart` 指令启动 FIFO0 的运动，将会提示错误。

(5) 刀向跟随功能

刀向跟随，就是在插补运动的过程中，部分轴会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：`GT_BufMove` 和 `GT_BufGear`。`GT_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；`GT_BufGear` 可以在插补过程中实现其他轴跟随插补合成位移的运动。

1) 插补过程中的点位运动

插补过程中的点位运动通过在缓存区中压入 `GT_BufMove` 指令来实现，该指令的第二个参数是需要进行点位运动的轴号，这里需要注意的是，需要进行点位运动的轴不能是坐标系中的轴；当该轴的运动模式不是点位运动模式而且正在运动时，该指令将不能正常执行。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，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。使用的具体例程如下：

例程 6-12 点位运动中的刀向跟随

```

.....
short rtn;
short run; // 定义坐标系运动状态查询变量
long segment; // 定义坐标系运动完成段查询变量
rtn = GT_CrdClear(1,0); // 清除坐标系的FIFO0中的数据

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

rtn = GT_BufMove(1,4,50000,30,0.1,0,0); // 缓存区内的点位运动指令
// 点位运动的轴号：第轴
// 点位运动的目标位置：pulse
// 点位运动的目标速度：pulse/ms
// 点位运动的目标加速度：.1 pulse/(ms*ms)
// 该点位运动是非模态指令
rtn = GT_LnXY (1,200000,0,100,0.1,0,0); // 直线插补指令

rtn = GT_BufMove(1,4,100000,30,0.1,1,0); // 缓存区内的点位运动指令
// 点位运动的轴号：第轴
// 点位运动的目标位置：pulse
// 点位运动的目标速度：pulse/ms

```

```

// 点位运动的目标加速度: .1 pulse/(ms*ms)
// 该点位运动是模态指令

rtn = GT_ArcXYC(1,-200000,0,-200000,0,0,100,0.1,0,0); // 圆弧插补指令
do
{
    rtn = GT_CrdStatus(1,&run,&segment,0);
}while(run == 1); // 坐标系在运动,查询到的run的值为1
.....

```

例程的运行结果如图 6-30 所示:



图 6-30 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，等点位运动结束之后，再进行插补运动。

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：

- a. 点位运动的目标位置是相对于机床原点的绝对位置。
- b. 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。
- c. 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 `GT_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。
- d. 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) 插补过程中的跟随运动

插补过程中的跟随运动通过在缓存区中压入 `GT_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，这里需要注意的是，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 `GT_BufGear` 时该轴正在运动时，该指令将不能正常执行。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：

例程 6-13 跟随运动中的刀向跟随

```

.....
short rtn;
short run;                // 定义坐标系运动状态查询变量
long segment;            // 定义坐标系运动完成段查询变量
rtn = GT_CrdClear(1,0);  // 清除坐标系的FIFO中的数据

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

rtn = GT_BufGear(1,4,50000, 0);           // 缓存区内的跟随运动指令
// 跟随运动的轴号：第轴
// 跟随运动的位移量：pulse

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

rtn = GT_BufGear(1,4,50000,0);           // 缓存区内的跟随运动指令
// 跟随运动的轴号：第轴
// 跟随运动的位移量：pulse

rtn = GT_ArcXYC(1,-200000,0,-200000,0,0,100,0.1,0,0); // 圆弧插补指令
do
{
    rtn = GT_CrdStatus(1,&run,&segment,0);
}while(run == 1);           // 坐标系在运动,查询到的run的值为1
.....

```

例程的运行结果如图 6-31 所示：

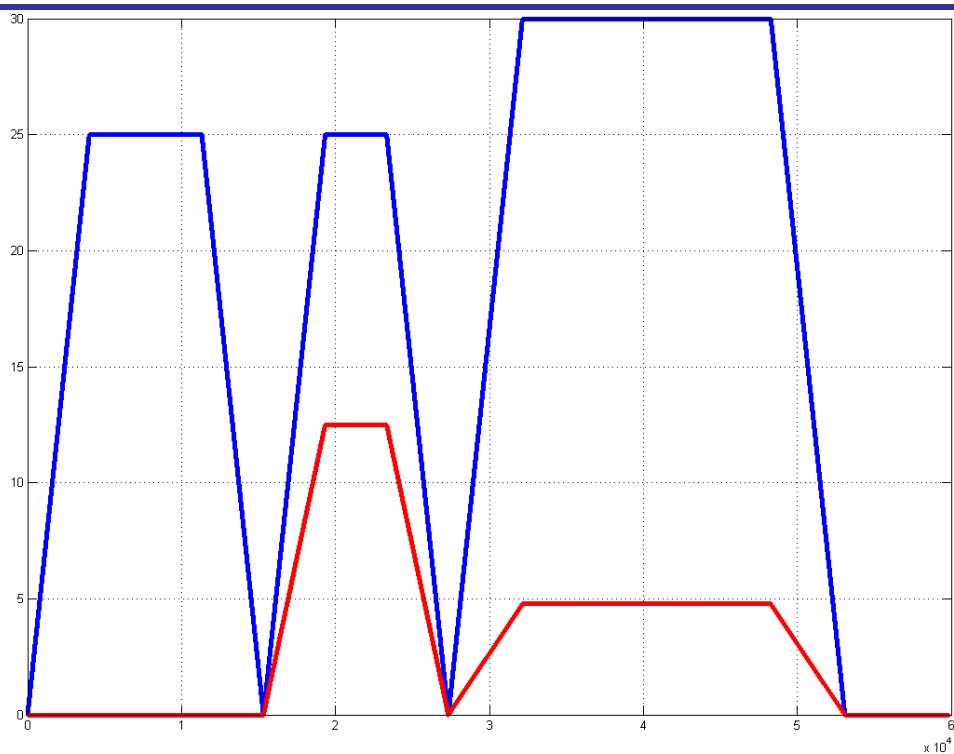


图 6-31 插补缓存区内的跟随运动速度图

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：

- `GT_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 `GT_BufGear` 指令来实现多个轴跟随插补运动。
- 当暂停坐标系运动时，当插补的合成速度减速到 0 时，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 `GT_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。

6.7 PVT 模式

6.7.1 指令列表

表 6-8 PVT 指令列表

指令	说明	页码
<code>GT_PrPvt</code>	设置指定轴为 PVT 模式	201
<code>GT_SetPvtLoop</code>	设置循环次数	224
<code>GT_GetPvtLoop</code>	查询循环次数	188
<code>GT_PvtTable</code>	向指定数据表传送数据，采用 PVT 描述方式	206
<code>GT_PvtTableComplete</code>	向指定数据表传送数据，采用 Complete 描述方式	207
<code>GT_PvtTablePercent</code>	向指定数据表传送数据，采用 Percent 描述方式	208
<code>GT_PvtPercentCalculate</code>	计算 Percent 描述方式下各数据点的速度	205
<code>GT_PvtTableContinuous</code>	向指定数据表传送数据，采用 Continuous 描述方式	207
<code>GT_PvtContinuousCalculate</code>	计算 Continuous 描述方式下各数据点的时间	205

GT_PvtTableSelect	选择数据表	209
GT_PvtStart	启动运动	205
GT_PvtStatus	读取状态	206

6.7.2 重点说明

PVT 模式使用一系列数据点的“位置、速度、时间”参数来描述运动规律。

位置、速度和时间满足如下函数关系：

$$p = at^3 + bt^2 + ct + d$$

$$v = \frac{dp}{dt} = 3at^2 + 2bt + c$$

如果给定相邻 2 个数据点的“位置、速度、时间”参数，可以得到如下方程组：

$$\begin{cases} at_1^3 + bt_1^2 + ct_1 + d = p_1 \\ 3at_1^2 + 2bt_1 + c = v_1 \\ at_2^3 + bt_2^2 + ct_2 + d = p_2 \\ 3at_2^2 + 2bt_2 + c = v_2 \end{cases}$$

求解该方程组，可以得到 a、b、c、d，因此相邻 2 个数据点的运动规律就可以确定下来。

运动控制器提供 32 个数据表存储数据点。每个数据表具有 1024 个存储空间。数据表和轴之间相互独立，一个数据表可以供多个轴使用。

调用 GT_PvtTable、GT_PvtTableComplete、GT_PvtTablePercent 或 GT_PvtTableContinuous 指令向数据表中传递数据。这些指令会删除数据表中原先的数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

调用 GT_PvtTableSelect 指令选择数据表。可以在运动状态下切换数据表，但是不会立即切换。只有当前数据表执行完毕以后，才会切换到新的数据表。

调用 GT_PvtStart 启动运动。启动以后，各轴时间清 0。如果第一个数据点的时间为 0 则立即启动，否则会延时启动，延时时间等于第一个数据点的时间。

数据表可以循环执行，调用 GT_SetPvtLoop 设置循环次数，循环次数为 0 表示无限循环。当遍历完数据表以后，时间初始化为第一个数据点的时间，而不是 0。

假设有如下 4 个数据点，采用 PVT 方式进行描述。

表 6-9 PVT 案例示范数据

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	1,000	0	0
P2	2,000	5,000	10
P3	3,000	15,000	10
P4	4,000	20,000	0

1. 调用 GT_PrfPvt 将轴切换到 PVT 模式；
2. 调用 GT_PvtTable 将 4 个数据点传递到数据表；

3. 调用 `GT_SetPvtLoop` 设置为循环执行；
4. 调用 `GT_PvtStart` 启动运动。

由于 P1 的时间为 1000 毫秒，因此调用 `GT_PvtStart` 以后延时 1000 毫秒启动。由于是循环执行，到达 P4 以后返回到 P1，速度曲线如图 6-32 所示。

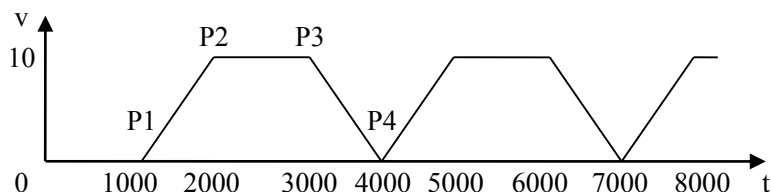


图 6-32 循环执行数据表

PVT 模式有 4 种方式描述运动规律，PVT、Complete、Percent 和 Continuous，下面对此进行详细说明。

(1) PVT 描述方式

PVT 描述方式直接定义各数据点的“位置、速度、时间”。相邻 2 个数据点之间，运动控制器使用 3 次多项式对位置进行插值，使用 2 次多项式对速度进行插值。因此当给出各数据点“位置、速度、时间”参数以后，相应的运动规律也就确定下来。

例如下面 4 组数据点，采用 PVT 描述方式。

表 6-10 合理的 PVT 数据点参数

数据点	时间 (毫秒)	位置 (脉冲)	速度 (脉冲/毫秒)
P1	0	0	0
P2	1,000	5,000	10
P3	2,000	15,000	10
P4	3,000	20,000	0
数据点	时间 (毫秒)	位置 (脉冲)	速度 (脉冲/毫秒)
P1	0	0	0
P2	1,000	5,000	9
P3	2,000	15,000	9
P4	3,000	20,000	0
数据点	时间 (毫秒)	位置 (脉冲)	速度 (脉冲/毫秒)
P1	0	0	0
P2	1,000	5,000	7.5
P3	2,333	15,000	7.5
P4	3,333	20,000	0
数据点	时间 (毫秒)	位置 (脉冲)	速度 (脉冲/毫秒)
P1	0	0	0
P2	750	1,667	6.6669
P3	2,250	18,333	6.6669
P4	3,000	20,000	0

这 4 组数据点对应的运动规律如图 6-33 所示。

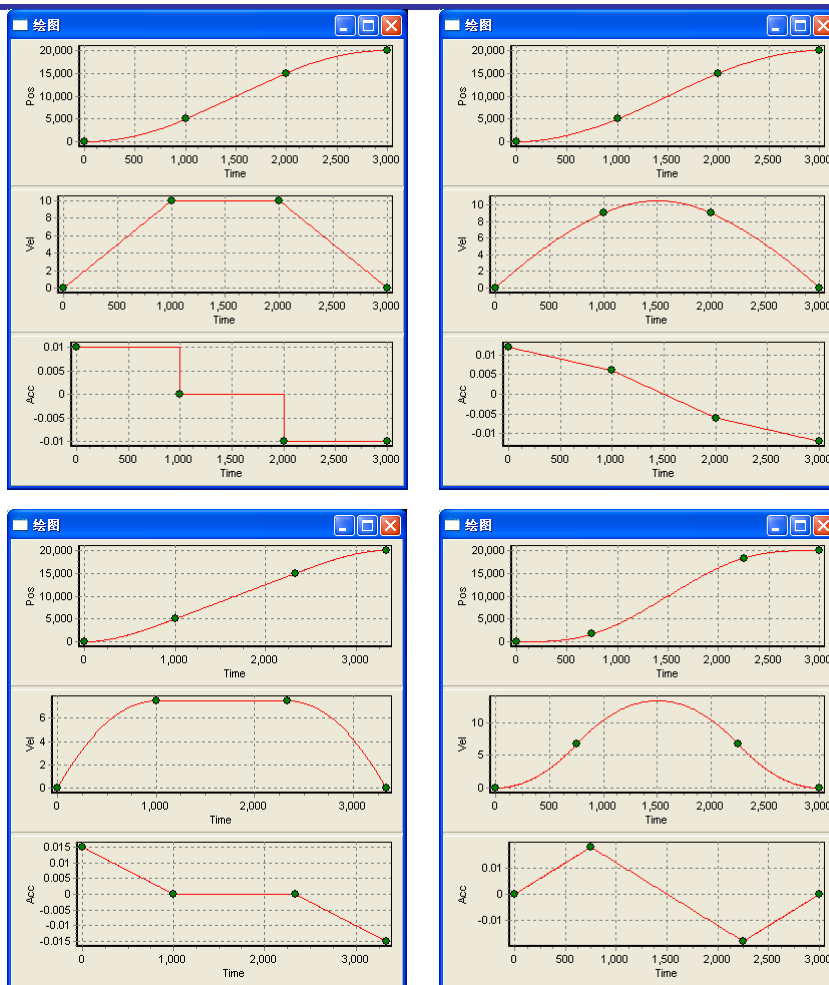


图 6-33 合理的 PVT 数据点参数

可以看出，PVT 描述方式非常灵活。给定数据点的“位置、速度、时间”参数，就能够得到相应的运动规律。

需要注意的是，数据点参数需要仔细设计，否则难以得到理想的运动规律。

例如下面 2 组数据点参数不合理，得到的速度曲线不够平滑。

表 6-11 不合理的 PVT 数据点

数据点	时间 (毫秒)	位置 (脉冲)	速度 (脉冲/毫秒)
P1	0	0	0
P2	1,000	5,000	15
P3	2,000	15,000	15
P4	3,000	20,000	0
数据点	时间 (毫秒)	位置 (脉冲)	速度 (脉冲/毫秒)
P1	0	0	0
P2	1,000	5,000	5
P3	2,000	15,000	5
P4	3,000	20,000	0

这 2 组数据点对应的运动规律如图 6-34 所示。

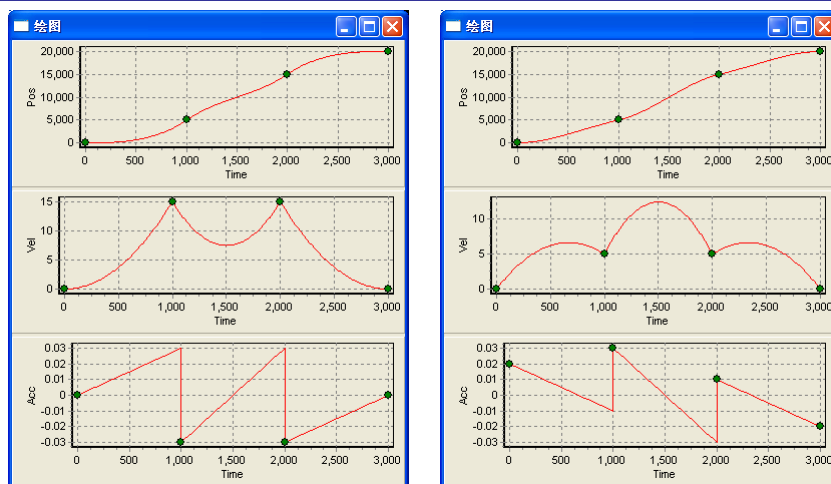


图 6-34 不合理的 PVT 数据点参数

(2) Complete 描述方式

Complete 描述方式定义各数据点的“位置、时间”，以及起点速度和终点速度。

Complete 方式只定义了起点速度和终点速度。运动控制器根据各数据点的“位置、时间”参数计算中间各点的速度，确保各数据点速度连续和加速度连续。

例如下面这组数据点，采用 Complete 描述方式，可以轻松得到光滑的速度曲线。

表 6-12 Complete 描述方式示范数据点

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	1,000	5,000	不指定
P3	2,000	15,000	不指定
P4	3,000	20,000	0

这组数据点对应的运动规律如图 6-35 所示。

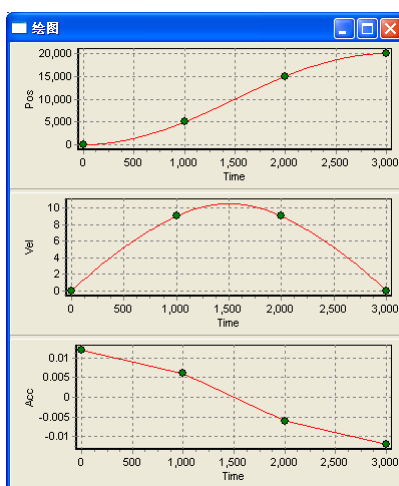


图 6-35 Complete 描述方式

Complete 适合描述光滑的速度曲线，例如三角函数等。

假设位置和时间之间的关系由函数 $P=50000\sin^2(\pi/2000*t)$ 确定。在一个函数周期 $[0,2000]$ 内取 5 个时间点计算相应的位置，如下表所示。

表 6-13 Complete 方式描述三角函数示范数据点

数据点	时间 (毫秒)	位置 (脉冲)	速度 (脉冲/毫秒)
P1	0	0	0
P2	500	25,000	不指定
P3	1,000	50,000	不指定
P4	1,500	25,000	不指定
P5	2,000	0	0

这组数据点对应的运动规律如图 6-36 所示。

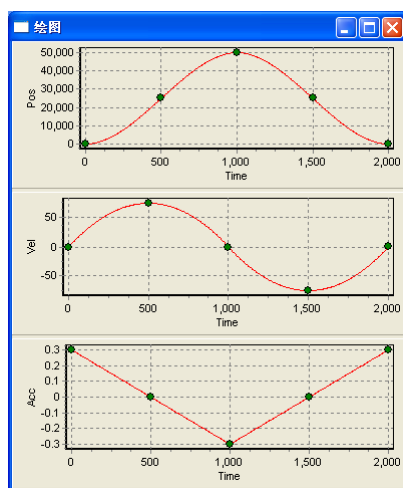


图 6-36 Complete 方式描述三角函数

增加数据点可以减小与函数 $P=50000\sin^2(\pi/2000*t)$ 的逼近误差。下图给出了数据点数为 5、10、50 时的位置误差。

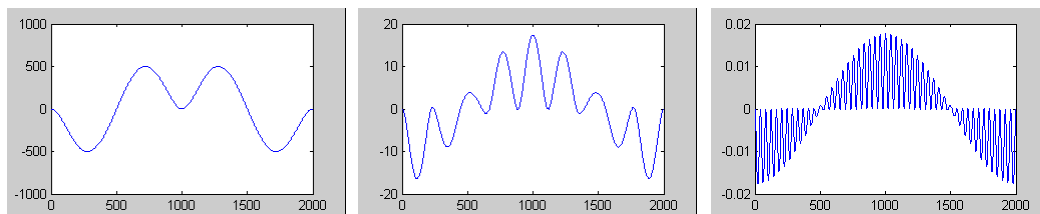


图 6-37 数据点数分别为 5、10、50 时的位置误差

(3) Percent 描述方式

Percent 描述方式定义各数据点的“位置、时间、百分比”，以及起点速度。

Percent 描述方式能够精确定义加速段、匀速段、减速段的位移、速度和时间。

Percent 描述方式假设相邻 2 个数据点之间速度为线性变化，利用起点速度以及各数据点的“位置、时间”参数，通过如下递推公式可以计算出各数据点的速度。

$$v_{i+1} = \frac{2(p_{i+1} - p_i)}{t_{i+1} - t_i} - v_i$$

因此指定了各数据点的“位置、时间”参数以后，各数据点的速度实际上也就已经确定下来。

通过“百分比”参数可以调整速度曲线的光滑性。数据点的百分比参数是指“相邻 2 个数据点之间加速度的变化时间占速度变化时间的百分比”。以图 6-38 为例来进行说明。

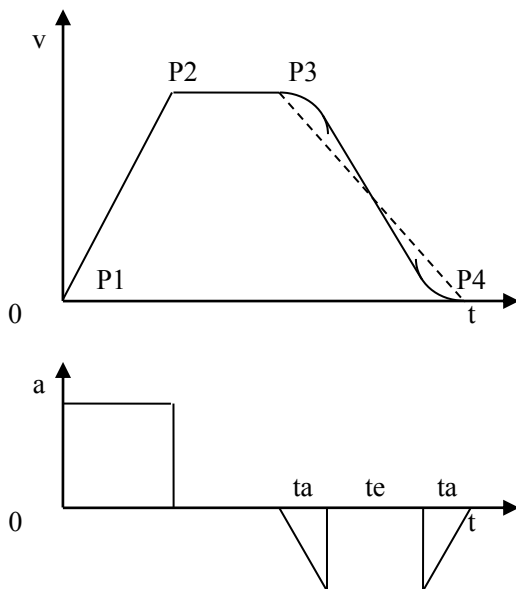


图 6-38 百分比的定义

数据点 P1 和 P2 之间加速度不变，因此数据点 P1 的百分比为 0。

数据点 P2 和 P3 之间加速度不变，因此数据点 P2 的百分比为 0。

数据点 P3 和 P4 之间加速度变化时间为 $2ta$ ，运动时间为 $2ta+te$ ，因此数据点 P3 的百分比为 $2ta/(2ta+te)*100\%$ 。

调整百分比参数，不会影响数据点的“位置、时间参数”。以上图 6-38 为例，当数据点 P3 的百分比为 0 时，数据点 P3 和 P4 之间的速度曲线为虚线；当数据点 P3 的百分比不为 0 时，数据点 P3 和 P4 之间的速度曲线为实线。

例如下面这组数据点，采用 Percent 描述方式。

表 6-14 Percent 描述方式示范数据点

数据点	时间 (毫秒)	位置 (脉冲)	百分比	速度 (脉冲/毫秒)
P1	0	0	60	0
P2	1,000	5,000	0	不指定
P3	2,000	15,000	100	不指定
P4	3,000	20,000	0	不指定

这组数据点对应的运动规律如图所示。

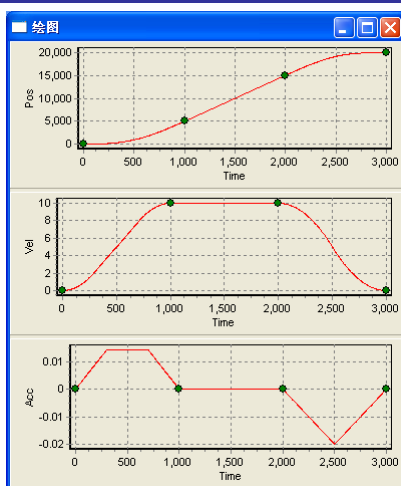


图 6-39 Percent 描述方式

(4) Continuous 描述方式

Continuous 描述方式定义各数据点的“位置、速度、最大速度、加速度、减速度、百分比”。不用指定数据点的时间。运动控制器根据数据点参数，自动将相邻 2 个数据点之间拆分为加速段、匀速段和减速度段。

数据点 P_i 的最大速度是指从数据点 P_i 到数据点 P_{i+1} 之间的速度上限。

数据点 P_i 的加速度是指从数据点 P_i 到数据点 P_{i+1} 之间的加速段所使用的加速度。

数据点 P_i 的减速度是指从数据点 P_i 到数据点 P_{i+1} 之间的减速段所使用的减速度。

数据点 P_i 的百分比是指从数据点 P_i 到数据点 P_{i+1} 之间的加减速度段中，加速度变化时间占速度变化时间的百分比。

相邻 2 个数据点之间能够拆分出来的段数和这 2 个数据点的参数有关，图 6-40 示例了一些可能的分段情况。

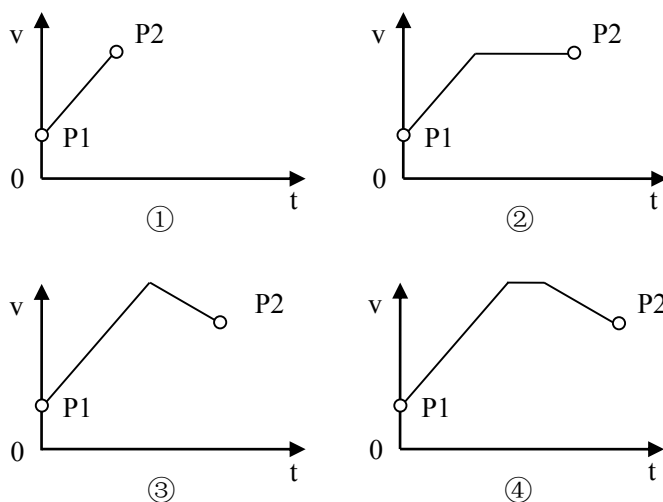


图 6-40 Continuous 描述方式

例如下面这 2 组数据点，采用 Continuous 描述方式。

表 6-15 Continuous 描述方式示范数据点

数据点	位置	速度	最大速度	加速度	减速度	百分比
P1	0	0	10	0.01	0.01	60
P2	20,000	0	10	0.01	0.01	0

数据点	位置	速度	最大速度	加速度	减速度	百分比
P1	0	0	10	0.01	0.01	60
P2	19,800	2	2	0.02	0.02	0
P3	21,800	0	2	0.02	0.02	0

这 2 组数据点对应的运动规律如图 6-41 所示。

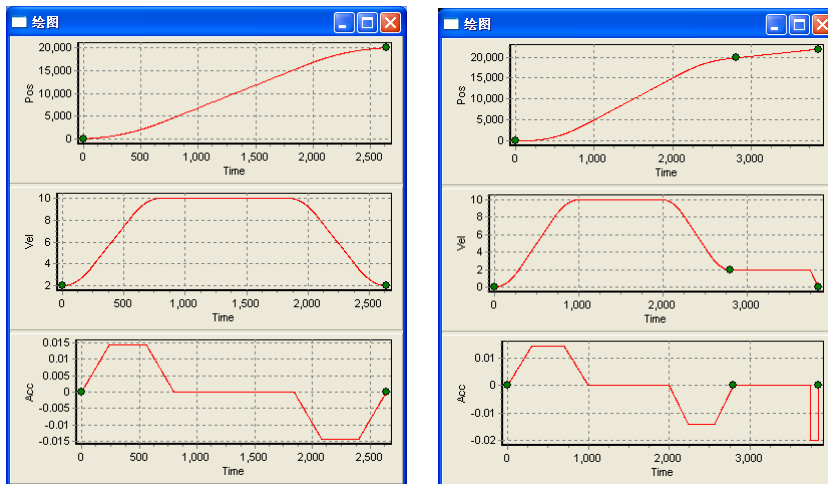


图 6-41 Continuous 描述方式

6.7.3 例程

(1) PVT 描述方式

整个速度曲线由 5 段组成，并且带有起跳速度。

第一段速度增大，加速度保持不变。

第二段速度增大，加速度减小。

第三段速度不变，加速度为 0。

第四段速度减小，加速度增大。

第五段速度减小，加速度不变。

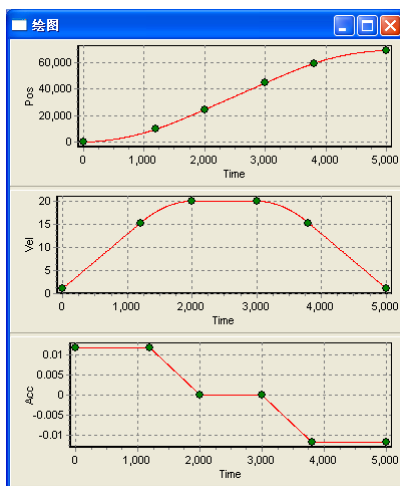


图 6-42 PVT 例程速度曲线

可以满足上述要求的一组数据表如下。

表 6-16 PVT 例程速度曲线数据点

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	1
P2	1,200	9,750	15.25
P3	2,000	24,483	20
P4	3,000	44,483	20
P5	3,800	59,216	15.25
P6	5,000	68,966	1

例程 6-14 PVT 描述方式

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

#define AXIS 1
#define TABLE 1

int main(int argc, char* argv[])
{
    short rtn;
    long mask;

    // X轴的数据点参数
    double time[6]={0,1200,2000,3000,3800,5000};
    double pos[6]={0,9750,24483,44483,59216,68966};
    double vel[6]={1,15.25,20,20,15.25,1};

    double prfVel,prfPos,t;
    short tableId;

    // 打开运动控制器
    rtn = GT_Open ();
    printf(" GT_Open=%d\n",rtn);

    // 复位运动控制器
    rtn = GT_Reset ();
    printf(" GT_Reset=%d\n",rtn);

    // 配置运动控制器
    // 注意：配置文件取消了各轴的报警和限位
    rtn = GT_LoadConfig ("test.cfg");
    printf(" GT_LoadConfig=%d\n",rtn);
```

```

// 清除各轴报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts=%d\n",rtn);

rtn = GT_AxisOn (AXIS);
printf(" GT_AxisOn=%d\n",rtn);

// 等待伺服使能就绪
Sleep(1000);

// 设置为PVT模式
rtn = GT_PrFpvt(AXIS);
printf(" GT_PrFpvt=%d\n",rtn);

// 发送数据
rtn = GT_PvtTable (TABLE,6,&time[0],&pos[0],&vel[0]);
printf(" GT_PvtTable=%d\n",rtn);

// 选择数据表
rtn = GT_PvtTableSelect (AXIS,TABLE);
printf(" GT_PvtTableSelect=%d\n",rtn);

mask = 1<<(AXIS-1);
rtn = GT_PvtStart (mask);
printf(" GT_PvtStart=%d\n",rtn);

while(!kbhit())
{
    // 读取数据表和运动时间
    rtn = GT_PvtStatus (AXIS,&tableId,&t);

    // 读取规划速度
    rtn = GT_GetPrfVel(AXIS,&prfVel);

    // 读取规划位置
    rtn = GT_GetPrfPos(AXIS,&prfPos);

    printf("%2d %10.0lf %10.2lf %10.1lf\r",tableId,t,prfVel,prfPos);
}
return 0;
}

```

(2) Complete 描述方式

例程 6-15 Complete 描述方式

假设位置和时间之间的关系由函数 $P=40000\sin^2(\pi/2000*t)$ 确定。要求启动以后能够循环运动，按 A 键幅值增大 50%，按 B 键幅值减小 50%。

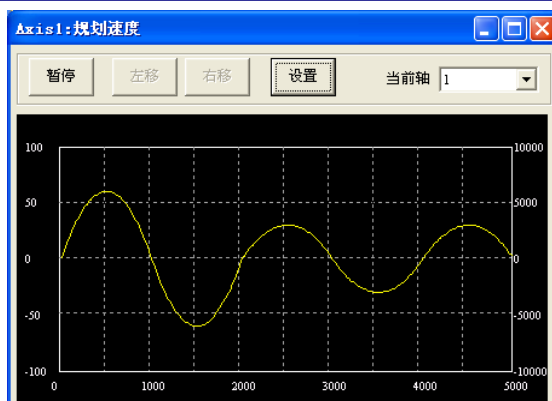


图 6-43 速度曲线

```

#include "stdafx.h"
#include "conio.h"
#include "windows.h"
#include "math.h"
#include "stdlib.h"
#include "gts.h"

#define AXIS          1
#define TABLE1      1
#define TABLE2      2
#define PI            3.1415926

void Calculate(double amplitude,long n,double *pTime,double *pPos)
{
    long i;

    for(i=0;i<n;++i)
    {
        pPos[i] = amplitude*sin(PI/2000*pTime[i])*sin(PI/2000*pTime[i]);
    }
}

int main(int argc, char* argv[])
{
    short rtn;
    long mask;

    // X轴的数据点参数
    double time[5]={0,500,1000,1500,2000};

    double pos[5];
    double a[5],b[5],c[5];

    double prfVel,prfPos,t;
    short tableId;

```

```
double amplitude = 40000;
short table = TABLE1;
char key;

// 打开运动控制器
rtn = GT_Open ();
printf(" GT_Open=%d\n",rtn);

// 复位运动控制器
rtn = GT_Reset ();
printf(" GT_Reset=%d\n",rtn);

// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
rtn = GT_LoadConfig ("test.cfg");
printf(" GT_LoadConfig=%d\n",rtn);

// 清除各轴报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts=%d\n",rtn);

rtn = GT_AxisOn (AXIS);
printf(" GT_AxisOn=%d\n",rtn);

// 等待伺服使能就绪
Sleep(1000);

// 设置为PVT模式
rtn = GT_PrPvt(AXIS);
printf(" GT_PrPvt=%d\n",rtn);

Calculate(amplitude,5,&time[0],&pos[0]);

// 发送数据
rtn = GT_PvtTableComplete (table,5,&time[0],&pos[0],&a[0],&b[0],&c[0],0,0);
printf(" GT_PvtTableComplete=%d\n",rtn);

// 选择数据表
rtn = GT_PvtTableSelect (AXIS,table);
printf(" GT_PvtTableSelect=%d\n",rtn);

// 设置为循环执行
rtn = GT_SetPvtLoop (AXIS,0);
printf(" GT_SetPvtLoop=%d\n",rtn);
```

```
mask = 1<<(AXIS-1);
rtn = GT_PvtStart(mask);
printf(" GT_PvtStart=%d\n",rtn);

while(1)
{
    // 读取数据表和运动时间
    rtn = GT_PvtStatus (AXIS,&tableId,&t);

    // 读取规划速度
    rtn = GT_GetPrfVel(AXIS,&prfVel);

    // 读取规划位置
    rtn = GT_GetPrfPos(AXIS,&prfPos);

    if( kbhit() )
    {
        key = getch();

        if( 'A' == toupper(key) )
        {
            amplitude *= 1.5;
        }

        if( 'B' == toupper(key) )
        {
            amplitude *= 0.5;
        }

        if( ( 'A' == toupper(key) ) || ( 'B' == toupper(key) ) )
        {
            Calculate(amplitude,5,&time[0],&pos[0]);

            table = TABLE1 + TABLE2 - tableId;

            // 发送数据
            rtn = GT_PvtTableComplete (table, 5, &time[0], &pos[0], &a[0], &b[0], &c[0], 0,
                                      0);

            if( 0 != rtn )
            {
                printf("\n GT_PvtTableComplete=%d\n",rtn);
                exit(0);
            }

            // 选择数据表
            rtn = GT_PvtTableSelect (AXIS,table);
```

```

if( 0 != rtn )
{
    printf("\n GT_PvtTableSelect=%d\n",rtn);
    exit(0);
}
}

if( 'Q' == toupper(key) )
{
    mask = 1 << (AXIS-1);
    GT_Stop(mask,0);
    exit(0);
}
}
printf("%2d %10.0lf %10.2lf %10.1lf r",tableId,t,prfVel,prfPos);
}
return 0;
}
    
```

(3) Percent 描述方式

例程 6-16 Percent 描述方式

X 轴往复运动，Y 轴正向进给。X 轴加减速时 Y 轴开始进给，X 轴匀速运动时，Y 轴保持静止。

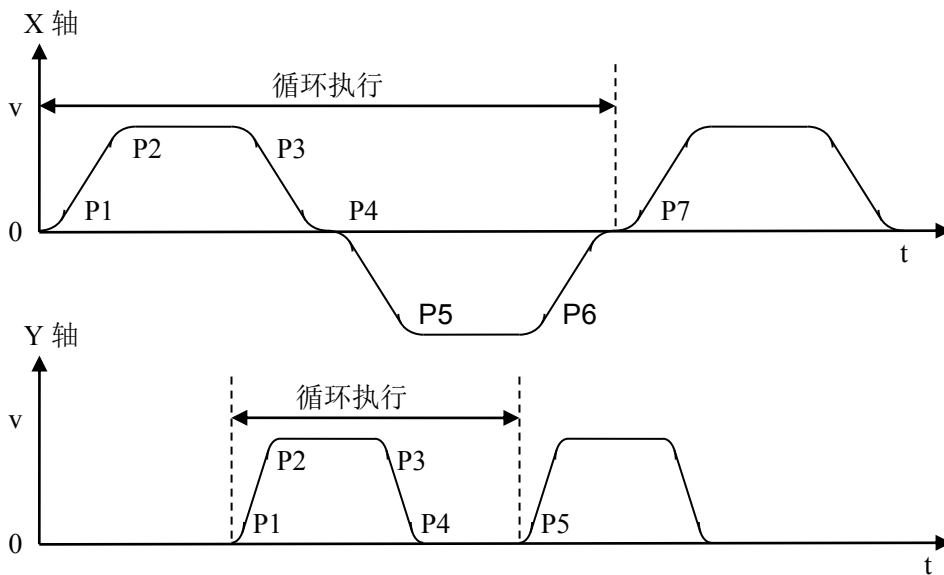


图 6-44 X 轴和 Y 轴的运动规律

X 轴取 7 个数据点，设置为循环模式。数据点参数如下：

表 6-17 Percent 描述方式 X 轴数据点

数据点	时间 (毫秒)	位置 (脉冲)	百分比	速度 (脉冲/毫秒)
P1	0	0	60	0
P2	1,000	5,000	0	不指定
P3	2,000	15,000	60	不指定

第 6 章 运动模式

P4	3,000	20,000	60	不指定
P5	4,000	15,000	0	不指定
P6	5,000	5,000	60	不指定
P7	6,000	0	0	不指定

根据 X 轴数据点参数，可以计算出各数据点的速度，百分比参数对数据点的速度计算没有影响。

表 6-18 Percent 描述方式 X 轴数据点计算结果

数据点	时间 (毫秒)	位置 (脉冲)	速度 (脉冲/毫秒)
P1	0	0	0
P2	1,000	5,000	$2(5000-0)/(1000-0)-0=10$
P3	2,000	15,000	$2(15000-5000)/(2000-1000)-10=10$
P4	3,000	20,000	$2(20000-15000)/(3000-2000)-10=0$
P5	4,000	15,000	$2(15000-20000)/(4000-3000)-0=-10$
P6	5,000	5,000	$2(5000-15000)/(5000-4000)-(-10)=-10$
P7	6,000	0	$2(0-5000)/(6000-5000)-(-10)=0$

Y 轴取 5 个数据点，设置为循环模式。X 轴启动以后到达数据点 P3 时 Y 轴才启动，因此第 1 个数据点的时间设置为 2000 毫秒。当 Y 轴到达 P5 以后，返回到 P1 循环执行。数据点参数如下：

表 6-19 Percent 描述方式 Y 轴数据点

数据点	时间 (毫秒)	位置 (脉冲)	百分比	速度 (脉冲/毫秒)
P1	2,000	0	60	0
P2	2,500	2,500	0	不指定
P3	3,500	12,500	60	不指定
P4	4,000	15,000	0	不指定
P5	5,000	15,000	0	不指定

根据 Y 轴数据点参数，可以计算出各数据点的速度，百分比参数对数据点的速度计算没有影响。

表 6-20 Percent 描述方式 Y 轴数据点计算结果

数据点	时间 (毫秒)	位置 (脉冲)	速度 (脉冲/毫秒)
P1	2,000	0	0
P2	2,500	2,500	$2(2500-0)/(2500-2000)-0=10$
P3	3,500	12,500	$2(12500-2500)/(3500-2500)-10=10$
P4	4,000	15,000	$2(15000-12500)/(4000-3500)-10=0$
P5	5,000	15,000	$2(15000-15000)/(5000-4000)-0=0$

X 轴循环 n 次，Y 轴需要循环 2n-1 次。下图是当 X 轴的循环次数为 2，Y 轴循环次数为 3 时的 XY 位置图。横轴是 X 轴的位置，纵轴是 Y 轴的位置。黄线是实际的运动轨迹。

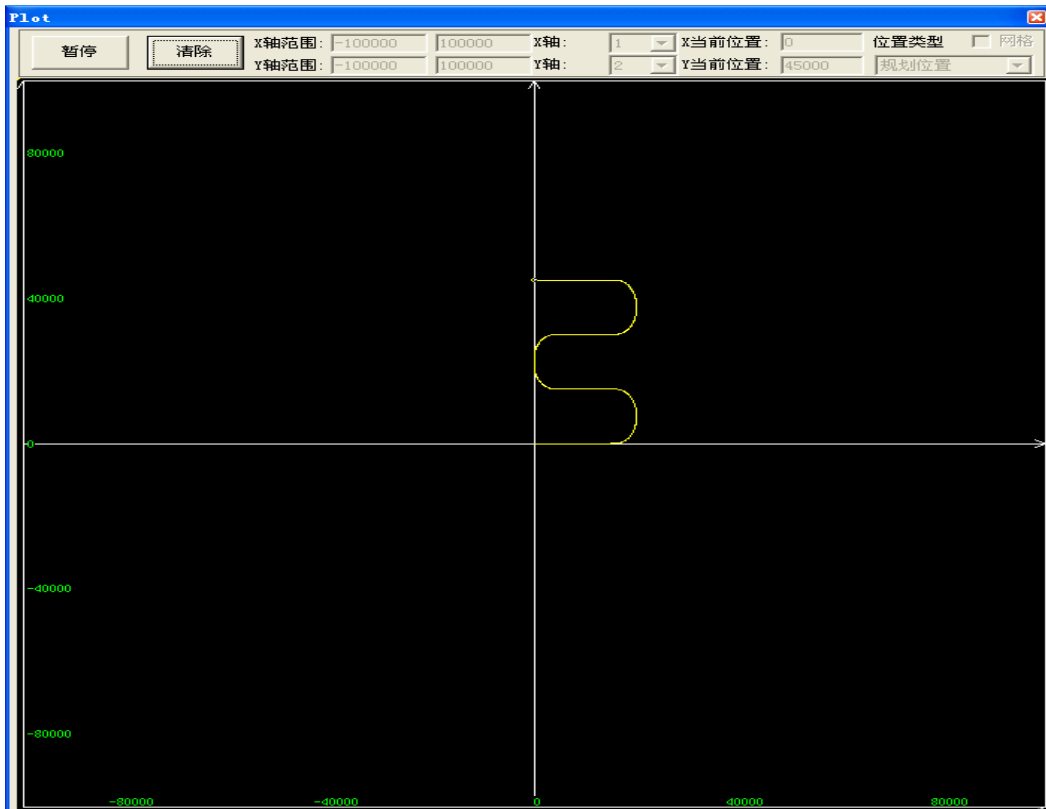


图 6-45 X-Y 位置图

```

#include "stdafx.h"
#include "conio.h"
#include "windows.h"
#include "gts.h"
#define AXIS_X      1
#define AXIS_Y      2
#define TABLE_X    1
#define TABLE_Y    2
#define LOOP_COUNT 2

int main(int argc, char* argv[])
{
    short rtn;
    long mask;

    // X轴的数据点参数
    double time_x[7] = {0,1000,2000,3000,4000,5000,6000};
    double pos_x[7] = {0,5000,15000,20000,15000,5000,0};
    double percent_x[7] = {60,0,60,60,0,60,0};

    // Y轴的数据点参数
    double time_y[5] = {2000,2500,3500,4000,5000};
    double pos_y[5] = {0,2500,12500,15000,15000};
    double percent_y[5] = {60,0,60,0,0};

```



```
double prfVel[2],prfPos[2],time[2];
short tableId[2];

// 打开运动控制器
rtn = GT_Open ();
printf(" GT_Open=%d\n",rtn);

// 复位运动控制器
rtn = GT_Reset ();
printf(" GT_Reset=%d\n",rtn);

// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
rtn = GT_LoadConfig ("test.cfg");
printf(" GT_LoadConfig=%d\n",rtn);

// 清除各轴报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts=%d\n",rtn);

rtn = GT_AxisOn (AXIS_X);
printf(" GT_AxisOn=%d\n",rtn);

rtn = GT_AxisOn (AXIS_Y);
printf(" GT_AxisOn=%d\n",rtn);

// 等待伺服使能就绪
Sleep(1000);

// 将X轴设置为PVT模式
rtn = GT_PrPvt(AXIS_X);
printf(" GT_PrPvt=%d\n",rtn);

// 将Y轴设置为PVT模式
rtn = GT_PrPvt(AXIS_Y);
printf(" GT_PrPvt=%d\n",rtn);

// 向X轴的数据表发送数据
rtn = GT_PvtTablePercent (TABLE_X,7,&time_x[0],&pos_x[0],&percent_x[0],0);
printf(" GT_PvtTablePercent=%d\n",rtn);

// 向Y轴的数据表发送数据
rtn = GT_PvtTablePercent (TABLE_Y,5,&time_y[0],&pos_y[0],&percent_y[0],0);
printf(" GT_PvtTablePercent=%d\n",rtn);

// X轴选择数据表TABLE_X
```

```

rtn = GT_PvtTableSelect (AXIS_X, TABLE_X);
printf(" GT_PvtTableSelect=%d\n", rtn);

// Y轴选择数据表TABLE_Y
rtn = GT_PvtTableSelect (AXIS_Y, TABLE_Y);
printf(" GT_PvtTableSelect=%d\n", rtn);

// 设置循环次数
rtn = GT_SetPvtLoop (AXIS_X, LOOP_COUNT);
printf(" GT_SetPvtLoop=%d\n", rtn);

// 设置循环次数
rtn = GT_SetPvtLoop (AXIS_Y, 2*LOOP_COUNT-1);
printf(" GT_SetPvtLoop=%d\n", rtn);

// 同时启动X轴和Y轴
// 由于Y轴的第二个数据点时间为ms
// 因此X轴启动ms以后，Y轴才开始运动
mask = 1<<(AXIS_X-1);
mask |= 1<<(AXIS_Y-1);
rtn = GT_PvtStart (mask);
printf(" GT_PvtStart=%d\n", rtn);

while(!kbhit())
{
    // 读取数据表和运动时间
    rtn = GT_PvtStatus (AXIS_X, &tableId[0], &time[0]);
    rtn = GT_PvtStatus (AXIS_Y, &tableId[1], &time[1]);

    // 读取规划速度
    rtn = GT_GetPrfVel(AXIS_X, &prfVel[0]);
    rtn = GT_GetPrfVel(AXIS_Y, &prfVel[1]);

    // 读取规划位置
    rtn = GT_GetPrfPos(AXIS_X, &prfPos[0]);
    rtn = GT_GetPrfPos(AXIS_Y, &prfPos[1]);

    printf("x:%2d %10.0lf %10.2lf %10.1lf y:%2d %10.0lf %10.2lf %10.1lf\r",
        tableId[0], time[0], prfVel[0], prfPos[0], tableId[1], time[1], prfVel[1], prfPos[1]);
}
return 0;
}

```

(4) Continuous 描述方式

例程 6-17 Continuous 描述方式

X 轴从 A 点运动到 B 点，Y 轴从 C 点运动到 D 点。要求 X 轴到达 B 点时，Y 轴同时到达 D 点。

首先调用 `GT_PvtContinuousCalculate` 指令计算 X 轴的运动时间 t_x 和 Y 轴的运动时间 t_y 。该指令不会把数据点发送到运动控制器。如果 $t_x > t_y$ ，Y 轴延时 $t_x - t_y$ 启动；如果 $t_x < t_y$ ，X 轴延时 $t_y - t_x$ 启动。

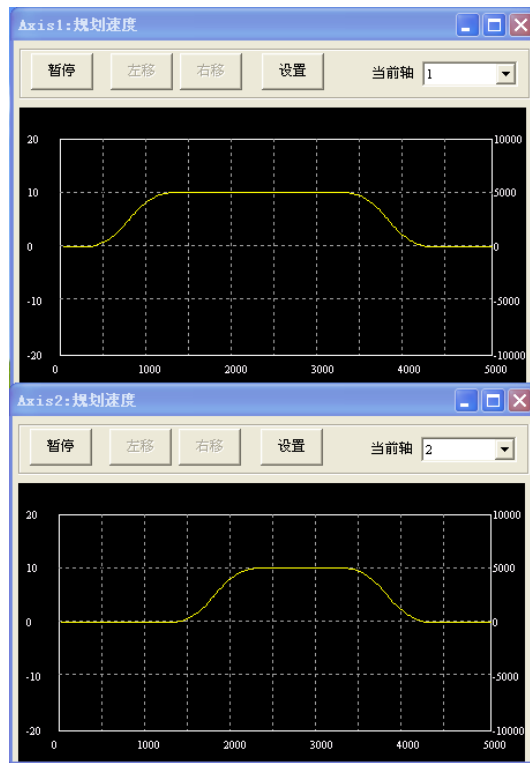


图 6-46 X 轴和 Y 轴的速度曲线

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

#define AXIS_X      1
#define AXIS_Y      2
#define TABLE_X    1
#define TABLE_Y    2

int main(int argc, char* argv[])
{
    short rtn;
    long mask;

    // X轴的数据点参数
    double pos_x[2] = {0,30000};
    double vel_x[2] = {0,0};
    double percent_x[2] = {100,100};
    double velMax_x[2] = {10,10};
    double acc_x[2] = {0.01,0.01};
    double dec_x[2] = {0.01,0.01};
    double time_x[2];
```

```
double timeBegin_x;

// Y轴的数据点参数
double pos_y[2] = {0,20000};
double vel_y[2] = {0,0};
double percent_y[2] = {100,100};
double velMax_y[2] = {10,10};
double acc_y[2] = {0.01,0.01};
double dec_y[2] = {0.01,0.01};
double time_y[2];
double timeBegin_y;

double prfVel[2],prfPos[2],time[2];
short tableId[2];

// 打开运动控制器
rtn = GT_Open (1,3);
printf(" GT_Open=%d\n",rtn);

// 复位运动控制器
rtn = GT_Reset ();
printf(" GT_Reset=%d\n",rtn);

// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
rtn = GT_LoadConfig ("test.cfg");
printf(" GT_LoadConfig=%d\n",rtn);

// 清除各轴报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts=%d\n",rtn);

rtn = GT_AxisOn (AXIS_X);
printf(" GT_AxisOn=%d\n",rtn);

rtn = GT_AxisOn (AXIS_Y);
printf(" GT_AxisOn=%d\n",rtn);

// 等待伺服使能就绪
Sleep(1000);

// 将X轴设置为PVT模式
rtn = GT_PrFpvt(AXIS_X);
printf(" GT_PrFpvt=%d\n",rtn);

// 将Y轴设置为PVT模式
```

```
rtn = GT_PrFpvt (AXIS_Y);
printf(" GT_PrFpvt=%d\n",rtn);

// 计算X轴运动时间
rtn = GT_PvtContinuousCalculate (2,&pos_x[0],&vel_x[0],&percent_x[0],&velMax_x[0],
                                &acc_x[0], &dec_x[0], &time_x[0]);
printf(" GT_PvtContinuousCalculate=%d\n",rtn);

// 计算Y轴运动时间
rtn = GT_PvtContinuousCalculate (2,&pos_y[0],&vel_y[0],&percent_y[0],&velMax_y[0],
                                &acc_y[0], &dec_y[0],&time_y[0]);
printf(" GT_PvtContinuousCalculate=%d\n",rtn);

// 计算启动延时
if( time_x[1] < time_y[1] )
{
    timeBegin_x = time_y[1] - time_x[1];
    timeBegin_y = 0;
}
else
{
    timeBegin_x = 0;
    timeBegin_y = time_x[1] - time_y[1];
}

// 发送X轴数据点
rtn = GT_PvtTableContinuous (TABLE_X,2,&pos_x[0],&vel_x[0],&percent_x[0], &velMax_x[0],
                              &acc_x[0],&dec_x[0],timeBegin_x);
printf(" GT_PvtTableContinuous=%d\n",rtn);

// 发送Y轴数据点
rtn = GT_PvtTableContinuous (TABLE_Y,2,&pos_y[0],&vel_y[0],&percent_y[0], &velMax_y[0],
                              &acc_y[0],&dec_y[0],timeBegin_y);
printf(" GT_PvtTableContinuous=%d\n",rtn);

// X轴选择数据表TABLE_X
rtn = GT_PvtTableSelect (AXIS_X,TABLE_X);
printf(" GT_PvtTableSelect=%d\n",rtn);

// Y轴选择数据表TABLE_Y
rtn = GT_PvtTableSelect (AXIS_Y,TABLE_Y);
printf(" GT_PvtTableSelect=%d\n",rtn);

// 同时启动X轴和Y轴
// 由于Y轴的第二个数据点时间为ms
// 因此X轴启动ms以后，Y轴才开始运动
```

```
mask = 1<<(AXIS_X-1);
mask |= 1<<(AXIS_Y-1);
rtn = GT_PvtStart(mask);
printf(" GT_PvtStart=%d\n",rtn);

while(!kbhit())
{
    // 读取数据表和运动时间
    rtn = GT_PvtStatus (AXIS_X,&tableId[0],&time[0]);
    rtn = GT_PvtStatus (AXIS_Y,&tableId[1],&time[1]);

    // 读取规划速度
    rtn = GT_GetPrfVel(AXIS_X,&prfVel[0]);
    rtn = GT_GetPrfVel(AXIS_Y,&prfVel[1]);

    // 读取规划位置
    rtn = GT_GetPrfPos(AXIS_X,&prfPos[0]);
    rtn = GT_GetPrfPos(AXIS_Y,&prfPos[1]);

    printf("x:%2d %10.0lf %10.2lf %10.1lf y:%2d %10.0lf %10.2lf %10.1lf",
        tableId[0],time[0],prfVel[0],prfPos[0],tableId[1],time[1],prfVel[1],prfPos[1]);
}

return 0;
}
```

第7章 访问硬件资源



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“第12章 指令详细说明”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

7.1 访问数字 IO

7.1.1 指令列表

表 7-1 访问数字 IO 指令列表

指令	说明	页码
GT_GetDi	读取数字 IO 输入状态	177
GT_GetDiRaw	读取数字 IO 输入状态的原始值	177
GT_GetDiReverseCount	读取数字量输入信号的变化次数	178
GT_SetDiReverseCount	设置数字量输入信号的变化次数的初值	214
GT_SetDo	设置数字 IO 输出状态	214
GT_SetDoBit	按位设置数字 IO 输出状态	215
GT_SetDoBitReverse	使数字量输出信号输出定时脉冲信号	215
GT_GetDo	读取数字 IO 输出状态	178

7.1.2 重点说明

[GT_GetDi](#) 指令可以读取限位、驱动报警、原点、通用输入的输入电平状态。

[GT_SetDo](#) 指令可以设置驱动器使能、报警清除、通用输出的输出电平状态。默认情况下由于驱动器使能和轴的关联，不能直接设置驱动器使能的输出电平状态。关于如何设置或取消 do 和轴的关联，请参见“第三章 系统配置”。

[GT_GetDiReverseCount](#) 指令用来读取数字量输入的变化次数，当数字量输入由 0 变为 1，或者由 1 变为 0，该次数就会增加一次。[GT_SetDiReverseCount](#) 指令用来设置数字量变化次数计数器的初值。

[GT_SetDoBitReverse](#) 指令用来使数字量输出信号输出一个定时的脉冲，例如，假设当前通用数字量输出信号 1 是高电平，当调用指令 [GT_SetDoBitReverse](#) (MC_GPO,1,0,100);则该数字量信号将会发出一个 25ms 时间宽度的负脉冲。

7.1.3 例程

例程 7-1 访问数字 IO

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

```
int main(int argc, char* argv[])
```

```
{
```

```

short rtn;
long alarm,home,gpi,limitPositive,limitNegative;
rtn = GT_Open ();
printf(" GT_Open () = %d\n",rtn);
printf("alarm limit+ limit- home gpi\n");
while(1)
{
    // 读取驱动器报警电平
    GT_GetDi (MC_ALARM,&alarm);
    // 读取正限位开关电平
    GT_GetDi (MC_LIMIT_POSITIVE,&limitPositive);
    // 读取负限位开关电平
    GT_GetDi (MC_LIMIT_NEGATIVE,&limitNegative);
    // 读取原点开关电平
    GT_GetDi (MC_HOME,&home);
    // 读取通用输入电平
    GT_GetDi (MC_GPI,&gpi);
    // 读取辅助编码器通用输入电平
    GT_GetDi (MC_MPG,&mpg);
    printf("%-6lx%-7lx%-7lx%-5lx%-5lx%-5lx\r",alarm,limitPositive,
        limitNegative, home, gpi, mpg);
}
}

```

7.2 访问编码器

7.2.1 指令列表

表 7-2 访问编码器指令列表

指令	说明	页码
GT_GetEncPos	读取编码器位置	179
GT_GetEncVel	读取编码器速度	179
GT_SetEncPos	修改编码器位置	216

7.2.2 例程

例程 7-2 访问编码器

```

#include "gts.h"

int main(int argc, char* argv[])
{
    short rtn,i;
    double enc[8];

    rtn = GT_Open ();

```



```

printf("  GT_Open () = %d\n",rtn);
while(1)
{
    // 读取个编码器轴的位置
    GT_GetEncPos (1,&enc[0],8);
    for(i=0;i<8;++i)
    {
        printf("%10.0lf",enc[i]);
    }
    printf("\r");
}
}

```

7.3 访问 DAC

表 7-3 问 DAC 指令列表

指令	说明	页码
GT_SetDac	设置 dac 输出电压	213
GT_GetDac	读取 dac 输出电压	177

7.4 访问模拟量输入

表 7-4 问模拟量输入指令列表

指令	说明	页码
GT_GetAdc	读取模拟量输入的电压值	169
GT_GetAdcValue	读取模拟量输入的数字转换值	169

第8章 高速硬件捕获



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“第 12 章 指令详细说明”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

8.1 Home/Index 硬件捕获

8.1.1 指令列表

表 8-1 Home/Index 硬件捕获指令列表

指令	说明	页码
GT_SetCaptureMode	设置编码器捕获方式，并启动捕获	211
GT_GetCaptureMode	读取编码器捕获方式	173
GT_GetCaptureStatus	读取编码器捕获状态	174
GT_SetCaptureSense	设置捕获电平	211
GT_ClearCaptureStatus	清除捕获状态	161

8.1.2 重点说明

Home 捕获和 Index 捕获默认都是下降沿触发。

Home 捕获模式下，当出现 Home 下降沿时，FPGA 立刻锁存 Home 开关所对应的编码器位置，同时将该编码器轴的捕获状态标志位置 1，然后退出 Home 捕获模式。

Index 捕获模式下，当出现 Index（编码器 C 相）下降沿时，FPGA 立刻锁存该编码器位置，同时将该编码器轴的捕获触发标志位置 1，然后退出 Index 捕获模式。

当 Home 或 Index 捕获触发以后，重新启动 Home 或 Index 捕获时，会自动清除对应轴的捕获触发标志位。

8.1.3 例程

例程 8-1 Home/Index 硬件捕获

```
#include "gts.h"

#define ENCODER          1

void CaptureIndex(void)
{
    short rtn,status;
    long pos;
    double encPos;
```

```
// 启动Index捕获
rtn = GT_SetCaptureMode(ENCODER,CAPTURE_INDEX);
printf("\nGT_SetCaptureMode() = %d\n",rtn);

do
{
    // 查询捕获状态
    GT_GetCaptureStatus(ENCODER,&status,&pos);

    // 读取编码器位置
    // 该指令和捕获无关，仅用于显示编码器位置
    GT_GetEncPos (ENCODER,&encPos);
    // 显示捕获状态和编码器位置
    printf("status = %d enc = %-8.0lf\r",status,encPos);
    // 当指定轴捕获触发时退出循环
}while( 0 == status );

// 显示捕获位置
printf("\ncapture = %-8.0ld\n",pos);
}

void CaptureHome(void)
{
    short rtn,status;
    long pos;
    double encPos;

    // 启动Home捕获
    rtn = GT_SetCaptureMode(ENCODER,CAPTURE_HOME);
    printf("\nGT_SetCaptureMode() = %d\n",rtn);

    do
    {
        // 查询捕获状态
        GT_GetCaptureStatus(ENCODER,&status,&pos);

        // 读取编码器位置
        // 该指令和捕获无关，仅用于显示编码器位置
        GT_GetEncPos (ENCODER,&encPos);
        // 显示捕获状态和编码器位置
        printf("status = %d enc = %-8.0lf\r",status,encPos);
        // 当指定轴捕获触发时退出循环
    }while( 0 == status );
}
```

```

// 显示捕获位置
printf("\ncapture = %-8.0ld\n",pos);
}

int main(int argc, char* argv[])
{
    char ch;
    short rtn;

    rtn = GT_Open ();
    printf(" GT_Open () = %d\n",rtn);

    while(1)
    {
        // 选择捕获模式, H: Home捕获, I: Index捕获, Q: 退出程序
        printf("\n[H:Home|I:Index|Q:Quit]:");
        ch=toupper(getche());
        switch(ch)
        {
            case 'H': CaptureHome();break;
            case 'I': CaptureIndex(); break;
            case 'Q': return 0;
        }
    }
}

```

8.2 Home 回原点

8.2.1 重点说明

1. 工作台向原点（Home）开关方向运动，启动 Home 捕获。

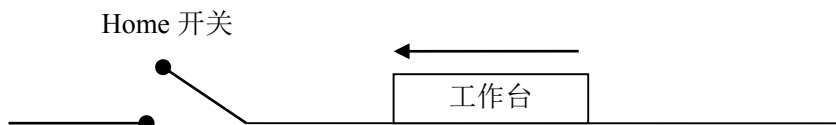


图 8-1 Home 回原点示意图 1

2. 当 Home 信号产生时，读取 Home 信号触发时工作台的实际位置，并将目标位置修改为“Home 捕获位置+偏移量”。

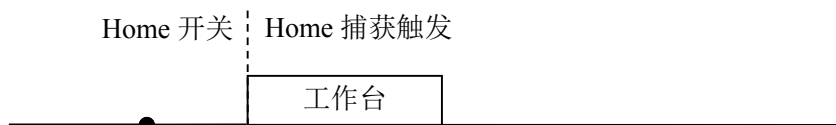


图 8-2 Home 回原点示意图 2

3. 等工作台停稳以后，调用 `GT_ZeroPos` 设置机械原点。

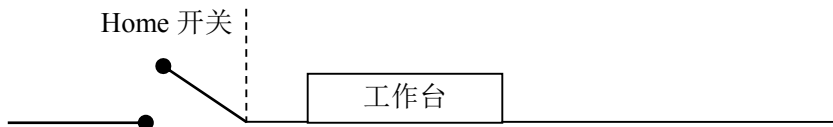


图 8-3 Home 回原点示意图 3

8.2.2 例程

例程 8-2 Home 回原点

```

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

#define AXIS          1

#define SEARCH_HOME    -200000
#define HOME_OFFSET    -2000

int main(int argc, char* argv[])
{
    short rtn, capture;
    TTrapPrm trapPrm;
    long status, pos;
    double prfPos, encPos, axisPrfPos, axisEncPos;
    // 打开运动控制器
    rtn =  GT_Open ();
    printf("  GT_Open ()=%d\n", rtn);
    // 复位运动控制器
    rtn =  GT_Reset ();
    printf("  GT_Reset ()=%d\n", rtn);
    // 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
    rtn =  GT_LoadConfig("test.cfg");
    printf("  GT_LoadConfig()=%d\n", rtn);
    // 清除指定轴的报警和限位
    rtn =  GT_ClrSts(AXIS);
    printf("  GT_ClrSts()=%d\n", rtn);
    // 驱动器使能
    rtn =  GT_AxisOn (AXIS);
    printf("  GT_AxisOn ()=%d\n", rtn);
    // 启动Home捕获
    rtn = GT_SetCaptureMode(AXIS, CAPTURE_HOME);
    printf("GT_SetCaptureMode()=%d\n", rtn);
    // 切换到点位运动模式

```

```

rtn = GT_PrftTrap(Axis);
printf(" GT_PrftTrap()=%d\n",rtn);
// 读取点位模式运动参数
rtn = GT_GetTrapPrm(Axis,&trapPrm);
printf(" GT_GetTrapPrm()=%d\n",rtn);
trapPrm.acc = 0.25;
trapPrm.dec = 0.25;
// 设置点位模式运动参数
rtn = GT_SetTrapPrm(Axis,&trapPrm);
printf(" GT_SetTrapPrm()=%d\n",rtn);
// 设置点位模式目标速度，即回原点速度
rtn = GT_SetVel(Axis,10);
printf(" GT_SetVel()=%d\n",rtn);
// 设置点位模式目标位置，即原点搜索距离
rtn = GT_SetPos(Axis,SEARCH_HOME);
printf(" GT_SetPos()=%d\n",rtn);
// 启动运动
rtn = GT_Update(1<<(Axis-1));
printf(" GT_Update()=%d\n",rtn);

printf("\nWaiting for home signal...\n");

do
{
    // 读取轴状态
    rtn = GT_GetSts(Axis,&status);
    // 读取捕获状态
    rtn = GT_GetCaptureStatus(Axis,&capture,&pos);
    // 读取规划位置
    rtn = GT_GetPrfPos(Axis,&prfPos);
    // 读取编码器位置
    rtn = GT_GetEncPos (Axis,&encPos);

    printf("capture=%d prfPos=%lf encPos=%lf\n", capture, prfPos, encPos);
    // 如果运动停止，返回出错信息
    if( 0 == ( status & 0x400 ) )
    {
        printf("\nno home found\n");
        getch();
        return 1;
    }
    // 等待捕获触发
}while( 0 == capture );
// 显示捕获位置
printf("\ncapture pos = %ld\n",pos);
// 运动到“捕获位置+偏移量”

```

```

rtn =  GT_SetPos(Axis, pos + HOME_OFFSET);
printf("  GT_SetPos()=%d\n",rtn);
// 在运动状态下更新目标位置
rtn =  GT_Update(1<<(Axis-1));
printf("  GT_Update()=%d\n",rtn);

do
{
    // 读取轴状态
    rtn =  GT_GetSts(Axis,&status);
    // 读取规划位置
    rtn =  GT_GetPrfPos(Axis,&prfPos);
    // 读取编码器位置
    rtn =  GT_GetEncPos (Axis,&encPos);

    printf("status=0x%-8lx prfPos=%-10.1lf encPos=%-10.1lf\n",status,prfPos,encPos);
    // 等待运动停止
}while( status & 0x400 );
// 检查是否到达“Home捕获位置+偏移量”
if( prfPos != pos+HOME_OFFSET )
{
    printf("\nmove to home pos error\n");
    getch();
    return 2;
}

printf("\nHome finish\n");
// 延时一段时间，等待电机停稳
Sleep(200);

printf("\nPress any key to set pos as 0...\n");
getch();

// 位置清零
rtn =  GT_ZeroPos (Axis);
printf("  GT_ZeroPos ()=%d\n",rtn);

// 读取规划位置
rtn =  GT_GetPrfPos(Axis,&prfPos);
// 读取编码器位置
rtn =  GT_GetEncPos (Axis,&encPos);
// 读取axis规划位置
rtn =  GT_GetAxisPrfPos(Axis,&axisPrfPos);
// 读取axis编码器位置
rtn =  GT_GetAxisEncPos(Axis,&axisEncPos);
printf("\nprfPos=%-10.0lf encPos=%-10.0lf axisPrfPos=%-10.0lf axisEncPos=%-10.0lf", prfPos,

```

```
encPos, axisPrfPos, axisEncPos);
```

```
    getch();
    return 0;
}
```

8.3 Home+Index 回原点

8.3.1 重点说明

1. 工作台向原点（Home）开关方向运动，启动 Home 捕获。

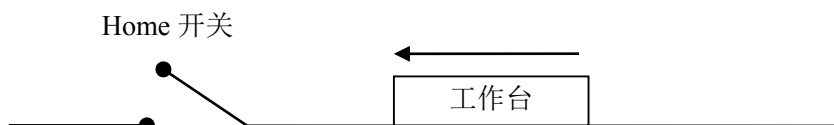


图 8-4 Home+Index 回原点示意图 1

2. 当 Home 信号产生时，读取 Home 信号触发时工作台的实际位置，并将目标位置修改为“Home 捕获位置+偏移量”。

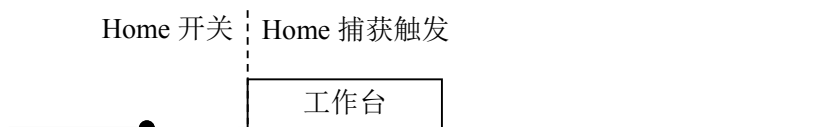


图 8-5 Home+Index 回原点示意图 2

3. 等工作台停稳以后，启动 Index 捕获。

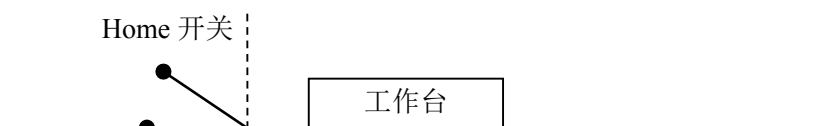


图 8-6 Home+Index 回原点示意图 3

4. 工作台继续运动，寻找 Index 信号。当 Index 信号产生时，读取 Index 信号触发时工作台的实际位置，并将目标位置修改为“Index 捕获位置+偏移量”。



图 8-7 Home+Index 回原点示意图 4

等工作台停稳以后，调用 `GT_ZeroPos` 设置机械原点。

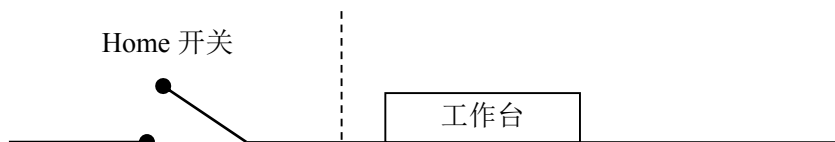


图 8-8 Home+Index 回原点示意图 5

8.3.2 例程

例程 8-3 Home+Index 回原点

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

#define AXIS          1

#define SEARCH_HOME   -200000
#define HOME_OFFSET   -2000
#define SEARCH_INDEX   15000
#define INDEX_OFFSET   1000

int main(int argc, char* argv[])
{
    short rtn,capture;
    TTrapPrm trapPrm;
    long status, pos;
    double prfPos,encPos,axisPrfPos,axisEncPos;

    rtn =  GT_Open();
    printf("  GT_Open()=%d\n",rtn);

    rtn =  GT_Reset();
    printf("  GT_Reset()=%d\n",rtn);

    rtn =  GT_LoadConfig("test.cfg");
    printf("  GT_LoadConfig()=%d\n",rtn);

    rtn =  GT_ClrSts(AXIS);
    printf("  GT_ClrSts()=%d\n",rtn);

    rtn =  GT_AxisOn(AXIS);
    printf("  GT_AxisOn()=%d\n",rtn);

    rtn = GT_SetCaptureMode(AXIS,CAPTURE_HOME);
    printf("GT_SetCaptureMode()=%d\n",rtn);

    rtn =  GT_PrftTrap(AXIS);
    printf("  GT_PrftTrap()=%d\n",rtn);

    rtn =  GT_GetTrapPrm(AXIS,&trapPrm);
    printf("  GT_GetTrapPrm()=%d\n",rtn);
```

```
trapPrm.acc = 0.25;
trapPrm.dec = 0.25;

rtn = GT_SetTrapPrm(Axis,&trapPrm);
printf(" GT_SetTrapPrm()=%d\n",rtn);

rtn = GT_SetVel(Axis,10);
printf(" GT_SetVel()=%d\n",rtn);

rtn = GT_SetPos(Axis,SEARCH_HOME);
printf(" GT_SetPos()=%d\n",rtn);

rtn = GT_Update(1<<(Axis-1));
printf(" GT_Update()=%d\n",rtn);

printf("\nWaiting for home signal...\n");

do
{
    rtn = GT_GetSts(Axis,&status);
    rtn = GT_GetCaptureStatus(Axis,&capture,&pos);
    rtn = GT_GetPrfPos(Axis,&prfPos);
    rtn = GT_GetEncPos(Axis,&encPos);

    printf("capture=%d prfPos=%-10.11f encPos=%-10.11fr", capture,prfPos,encPos);

    if( 0 == ( status & 0x400 ) )
    {
        printf("\nno home found");
        return 1;
    }
}while( 0 == capture );

printf("\ncapture pos = %ld\n",pos);

rtn = GT_SetPos(Axis, pos + HOME_OFFSET);
printf(" GT_SetPos()=%d\n",rtn);

rtn = GT_Update(1<<(Axis-1));
printf(" GT_Update()=%d\n",rtn);

do
{
    rtn = GT_GetSts(Axis,&status);
    rtn = GT_GetPrfPos(Axis,&prfPos);
```

```

rtn = GT_GetEncPos (AXIS,&encPos);

printf("status=0x%-8lx prfPos=%-10.1lf encPos=%-10.1lf\r",status,prfPos,encPos);
}while( status & 0x400 );

if( prfPos != pos + HOME_OFFSET)
{
printf("\nmove to home pos error");
return 2;
}

printf("\nHome finish\n");

Sleep(200);

rtn = GT_SetCaptureMode(AXIS,CAPTURE_INDEX);
printf("GT_SetCaptureMode()=%d\n",rtn);

rtn = GT_SetPos(AXIS,(long)(prfPos + SEARCH_INDEX));
printf(" GT_SetPos()=%d\n",rtn);

rtn = GT_Update(1<<(AXIS-1));
printf(" GT_Update()=%d\n",rtn);

printf("\nWaiting for index signal...\n");

do
{
rtn = GT_GetSts(AXIS,&status);
rtn = GT_GetCaptureStatus(AXIS,&capture,&pos);
rtn = GT_GetPrfPos(AXIS,&prfPos);
rtn = GT_GetEncPos (AXIS,&encPos);
printf("capture=%d prfPos=%-10.1lf encPos=%-10.1lf\r", capture,prfPos,encPos);

if( 0 == ( status & 0x400 ) )
{
printf("\nno home found\n");
getch();
return 1;
}
}while( 0 == capture );

printf("\ncapture pos = %ld\n",pos);

rtn = GT_SetPos(AXIS, pos+ INDEX_OFFSET);
printf(" GT_SetPos()=%d\n",rtn);

```

```
rtn = GT_Update(1<<(AXIS-1));
printf(" GT_Update()=%d\n",rtn);

do
{
    rtn = GT_GetSts(AXIS,&status);
    rtn = GT_GetPrfPos(AXIS,&prfPos);
    rtn = GT_GetEncPos (AXIS,&encPos);

    printf("status=0x%-8lx prfPos=%-10.1lf encPos=%-10.1lf",status,prfPos,encPos);
}while( status & 0x400 );

if( prfPos != pos+ INDEX_OFFSET)
{
    printf("\nmove to index pos error\n");
    getch();
    return 2;
}

printf("\nHome+Index finish\n");

printf("\nPress any key to set pos as 0...\n");
getch();

Sleep(200);

rtn = GT_ZeroPos (AXIS);
printf(" GT_ZeroPos ()=%d\n",rtn);

rtn = GT_GetPrfPos(AXIS,&prfPos);
rtn = GT_GetEncPos (AXIS,&encPos);
rtn = GT_GetAxisPrfPos(AXIS,&axisPrfPos);
rtn = GT_GetAxisEncPos(AXIS,&axisEncPos);
printf("\nprfPos=%-10.0lf encPos=%-10.0lf axisPrfPos=%-10.0lf axisEncPos=%-10.0lf", prfPos,
encPos,axisPrfPos,axisEncPos);

getch();
return 0;
}
```

第9章 安全机制



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“第12章 指令详细说明”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

9.1 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：

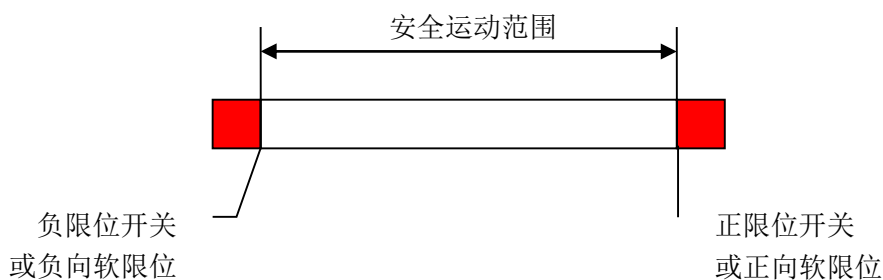


图 9-1 轴运动范围

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 [GT_ClrSts](#) 清除限位触发状态，才能使控制轴回到正常运动状态。

9.1.1 指令列表

表 9-1 软限位指令列表

指令	说明	页码
GT_SetSoftLimit	设置轴正向软限位和负向软限位	224
GT_GetSoftLimit	读取轴正向软限位和负向软限位	188

9.1.2 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 1 脉冲/毫秒²，如何设置急停加速度请参见 [4.2.6 配置 profile](#)。

9.1.3 例程

例程 9-1 限位

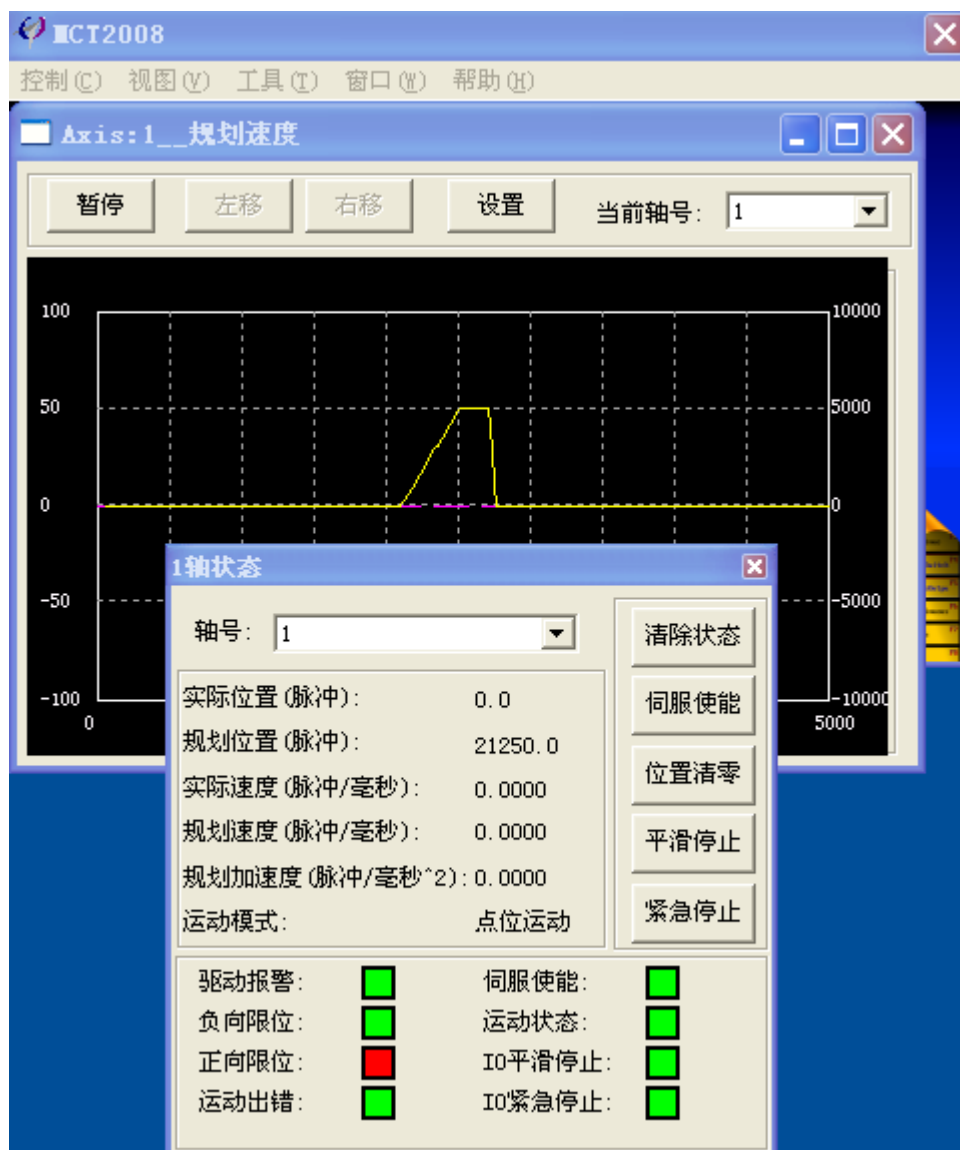


图 9-2 软限位触发

```

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

#define AXIS 1

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

```

```

rtn =  GT_Open ();
rtn =  GT_Reset ();
rtn =  GT_LoadConfig("test.cfg");
rtn =  GT_ClrSts(1,8);
rtn =  GT_SetSoftLimit(Axis,20000,-20000);
rtn =  GT_PrflTrap(Axis);

rtn =  GT_GetTrapPrm(Axis,&trap);
trap.acc = 0.125;
trap.dec = 0.125;
rtn =  GT_SetTrapPrm(Axis,&trap);

rtn =  GT_SetVel(Axis,50);
rtn =  GT_SetPos(Axis,1000000L);
rtn =  GT_Update(1<<(Axis-1));

while(!kbhit())
{
    rtn =  GT_GetSts(Axis,&sts);
    rtn =  GT_GetPrfPos(Axis,&prfPos);

    printf("sts=0x%-8lx prfPos=%-10.2lf\r",sts,prfPos);
}

return 0;
}

```

9.2 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 `GT_ClrSts`清除报警，重新回机床原点

9.3 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 `GT_ClrSts` 指令清除停止标志位 (bit7 和 bit8)，才能继续运动。

9.4 跟随误差极限

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

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

第10章 运动程序



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“第 12 章 指令详细说明”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

10.1 简介

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”。

C 语言编写的运动程序编译以后能够下载到运动控制器中执行。运动程序能够脱离主机在运动控制器上独立执行，从而将主机从繁琐的运动逻辑管理中解放出来。一方面主机能够将 CPU 资源分配给其它任务，另一方面运动程序能够直接访问运动控制器，不需要进行频繁通讯，从而具有更高的实时性。

当然，如果需要，主机仍然可以在任何时候向控制器发送指令，即使运动控制器上的运动程序正在执行。注意：当主机指令和运动控制器上的运动程序控制相同的轴时，需要仔细设计运动逻辑，以免造成混乱。

运动控制器允许多达 32 个运动程序在运动控制器上同时执行。

运动控制器内建的线程调度机制保证多线程环境下运动程序所有指令的执行都是完整的。

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。

10.2 编写运动程序

10.2.1 指令列表

表 10-1 运动程序指令列表

指令	说明	页码
GT_Download	下载运动程序到运动控制器	164
GT_GetFunId	读取运动程序中函数的标识	182
GT_GetVarId	读取运动程序中变量的标识	191
GT_Bind	绑定线程、函数、数据页	156
GT_RunThread	启动线程	209
GT_StopThread	停止正在运行的线程	229
GT_PauseThread	暂停正在运行的线程	200
GT_GetThreadSts	读取线程的状态	189
GT_SetVarValue	设置运动程序中变量的值	227
GT_GetVarValue	读取运动程序中变量的值	191

10.2.2 重点说明

编写运动程序的方法如下：

- 1) 使用文本编辑器编写运动程序的 C 语言源程序。
- 2) 使用 MCT2008 编译运动程序，生成目标程序文件 (*.bin) 和符号文件 (*.ini)。
- 3) 调用 `GT_Download` 指令将运动程序目标程序下载到运动控制器中。
- 4) 调用 `GT_GetFunId` 指令获取函数 ID
- 5) 调用 `GT_GetVarId` 指令获取变量 ID
- 6) 调用 `GT_Bind` 指令绑定线程、函数和数据页。
- 7) 调用 `GT_SetVarValue` 指令初始化局部变量和全局变量。
- 8) 调用 `GT_RunThread` 指令，启动线程。
- 9) 调用 `GT_GetThreadSts` 指令查询线程状态，或者调用 `GT_GetVarValue` 指令查询变量。

调用 `GT_Download` 指令可以将运动程序下载到运动控制器的 SDRAM 中。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。在发布应用程序时，应同时发布目标文件和符号文件。否则运动程序无法正确下载执行。

运动程序下载到运动控制器以后还不能立即执行，必须调用 `GT_Bind` 指令绑定线程、函数和数据页，然后调用 `GT_RunThread` 指令启动线程。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。

应用程序可以随时调用 `GT_GetThreadSts` 指令查询线程的执行状态。

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

10.2.3 例程

(1) 单线程累加求和

例程 10-1 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;
    TCompileInfo compile;
    short funId;
    TVarInfo sum,begin,end;
    double value;
    TThreadSts thread;

    // 打开运动控制器
    rtn = GT_Open ();
    printf(" GT_Open ()=%d\n",rtn);

    // 复位运动控制器
    rtn = GT_Reset ();
    printf(" GT_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 = GT_Compile("sum.c",&compile);
```

```
printf("GT_Compile(=%d\n",rtn);

// 下载运动程序sum.bin
rtn = GT_Download ("sum.bin");
printf(" GT_Download (=%d\n",rtn);

// 获取函数ID
rtn = GT_GetFunId ("add",&funId);
printf(" GT_GetFunId (=%d\n",rtn);

// 获取全局变量sum的ID
rtn = GT_GetVarId (NULL,"sum",&sum);
printf(" GT_GetVarId (=%d\n",rtn);

// 获取局部变量begin的ID
rtn = GT_GetVarId ("add","begin",&begin);
printf(" GT_GetVarId (=%d\n",rtn);

// 获取局部变量end的ID
rtn = GT_GetVarId ("add","end",&end);
printf(" GT_GetVarId (=%d\n",rtn);

// 绑定线程, 函数, 数据页
rtn = GT_Bind (0,funId,0);
printf(" GT_Bind (=%d\n",rtn);

value = 0;
// 初始化运动程序的全局变量sum
rtn = GT_SetVarValue (-1,&sum,&value);
printf(" GT_SetVarValue (=%d\n",rtn);

value = 1;
// 初始化运动程序的局部变量begin
rtn = GT_SetVarValue (0,&begin,&value);
printf(" GT_SetVarValue (=%d\n",rtn);

value = 100;
// 初始化运动程序的局部变量end
rtn = GT_SetVarValue (0,&end,&value);
printf(" GT_SetVarValue (=%d\n",rtn);

// 启动线程
rtn = GT_RunThread (0);
printf(" GT_RunThread (=%d\n",rtn);

do
```

```

{
    // 查询线程状态
    rtn = GT_GetThreadSts (0,&thread);

    // 查询全局变量sum的值
    rtn = GT_GetVarValue (-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 多线程累加求和

运动程序代码和例程 10-1 单线程累加求和相同。

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

```

#include "stdafx.h"
#include "conio.h"
#include "gts.h"
int main(int argc, char* argv[])
{
    short rtn;
    TCompileInfo compile;
    short funId;
    TVarInfo sum,begin,end;
    double value;
    TThreadSts thread;

    // 打开运动控制器
    rtn = GT_Open ();
    printf(" GT_Open ()=%d\n",rtn);

    // 复位运动控制器
    rtn = GT_Reset ();
    printf(" GT_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 = GT_Compile("sum.c",&compile);
    printf("GT_Compile()=%d\n",rtn);
}

```

```
// 下载运动程序sum.bin
rtn = GT_Download ("sum.bin");
printf(" GT_Download ()=%d\n",rtn);

// 获取函数ID
rtn = GT_GetFunId ("add",&funId);
printf(" GT_GetFunId ()=%d\n",rtn);

// 获取全局变量sum的ID
rtn = GT_GetVarId (NULL,"sum",&sum);
printf(" GT_GetVarId ()=%d\n",rtn);

// 获取局部变量begin的ID
rtn = GT_GetVarId ("add","begin",&begin);
printf(" GT_GetVarId ()=%d\n",rtn);

// 获取局部变量end的ID
rtn = GT_GetVarId ("add","end",&end);
printf(" GT_GetVarId ()=%d\n",rtn);

// 绑定线程，函数，数据页
rtn = GT_Bind (0,funId,0);
printf(" GT_Bind ()=%d\n",rtn);

// 绑定线程，函数，数据页
rtn = GT_Bind (1,funId,1);
printf(" GT_Bind ()=%d\n",rtn);

value = 0;
// 初始化运动程序的全局变量sum
rtn = GT_SetVarValue (-1,&sum,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);

value = 1;
// 初始化运动程序的局部变量begin
rtn = GT_SetVarValue (0,&begin,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);

value = 50;
// 初始化运动程序的局部变量end
rtn = GT_SetVarValue (0,&end,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);

value = 51;
// 初始化运动程序的局部变量begin
```

```

rtn = GT_SetVarValue (1,&begin,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);

value = 100;
// 初始化运动程序的局部变量end
rtn = GT_SetVarValue (1,&end,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);

// 启动线程
rtn = GT_RunThread (0);
printf(" GT_RunThread ()=%d\n",rtn);

// 启动线程
rtn = GT_RunThread (1);
printf(" GT_RunThread ()=%d\n",rtn);

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

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

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

getch();
return 0;
}

```

(3) 组合运动

例程 10-3 组合运动

该例程实现 3 个运动轴协调运动，其应用场景为半导体加工设备，定义 3 个运动轴分别为摆臂轴、送料轴和点胶轴。其中各个轴的运动关系如下所述。

摆臂轴：点胶轴离开工作区域时，启动摆臂轴向工作区域运动，完成工作后无需等待任何信号，即可撤离工作区域。

送料轴：摆臂轴离开工作区域时，即可启动送料轴运动。

点胶轴：摆臂轴离开工作区域之后，启动点胶轴向工作区域运动；当送料轴运动到位时，点胶轴完成

点胶工作，然后离开工作区域。

运动程序将三个轴的运动分别用三个函数来实现，ArmMotion 控制摆臂轴，GlueMotion 控制点胶轴，PieceMotion 控制物料轴，通过四个全局同步变量，来实现各个运动之间的互斥和关联。

pieceArrival: 标志送料轴已经到达，可以启动点胶轴的运动离开工作区域。

pieceStart: 启动送料轴标志位，标志送料轴可以启动。

glueStart: 启动点胶轴标志位，标志点胶轴可以开始运动到工作区域。

armStart: 启动摆臂轴标志位，标志摆臂轴可以开始向工作区域运动。

ArmMotion 函数与线程 0 和数据页 0 绑定；GlueMotion 函数与线程 1 和数据页 1 绑定；PieceMotion 函数与线程 2 和数据页 2 绑定。每个线程独立控制一个轴的运动。运动程序的变量初始值设置如下：

全局变量 pieceArrival 值：0；(物料轴未到达)

全局变量 armStart 值：0；(初始状态下摆臂轴运动条件不满足)

全局变量 pieceStart 值：1；(初始状态下物料轴运动条件满足)

全局变量 glueStart 值：1；(初始状态下点胶轴运动条件满足)

图 10-1 中 1 轴是摆臂轴规划速度，2 轴是点胶轴规划速度，3 轴是物料轴规划速度。

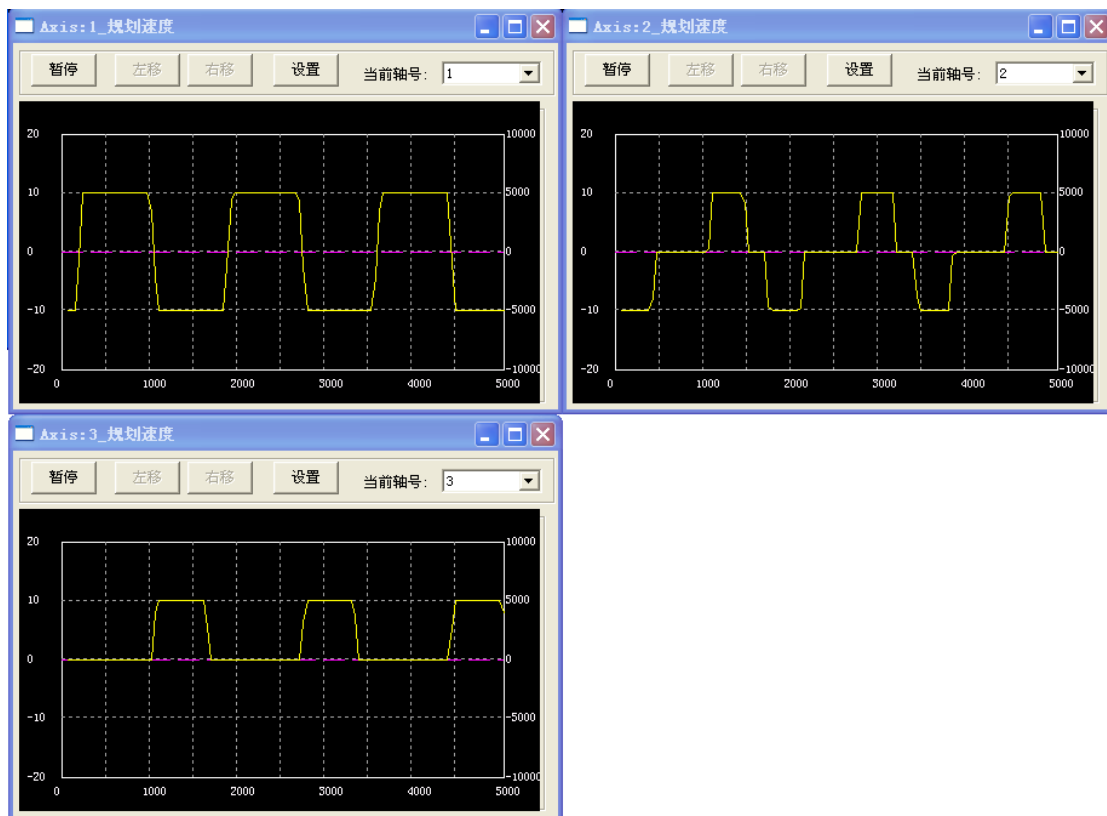


图 10-1 组合运动时序图

运动程序代码如下：

```
//-----
// 组合运动
//-----
```



```
int pieceArrival;    // 物料轴到达标志
int pieceStart;     // 物料轴启动标志
int glueStart;      // 点胶轴启动标志
int armStart;       // 摆臂轴启动标志
```

```
// 摆臂轴运动
```

```
void ArmMotion(short arm)
```

```
{
```

```
    long armStep;
```

```
    short cc;
```

```
    long clock;
```

```
    long sts;
```

```
    short axis;
```

```
lbl_loop:
```

```
    axis=arm-1;
```

```
    axis=1<<axis;
```

```
    // 等待摆臂启动标志
```

```
lbl_wait_for_arm_start:
```

```
    cc = !armStart;
```

```
    if(cc) goto lbl_wait_for_arm_start;
```

```
    // 清除摆臂启动标志
```

```
    armStart = 0;
```

```
    // 摆臂轴运动
```

```
        GT_SetPos(arm,armStep);
```

```
        GT_Update(axis);
```

```
    // 等待摆臂轴运动停止
```

```
lbl_wait_for_arm_stop:
```

```
        GT_GetSts(arm,&sts,1,&clock);
```

```
    cc = sts & 0x400;
```

```
    if(cc) goto lbl_wait_for_arm_stop;
```

```
    // 摆臂轴返回
```

```
        GT_SetPos(arm,0);
```

```
        GT_Update(axis);
```

```
    // 置起点胶轴启动标志
```

```
    glueStart = 1;
```

```
    // 置起物料轴启动标志
```

```
    pieceStart = 1;
```

```
// 等待摆臂轴运动停止
lbl_wait_for_arm_return:
    GT_GetSts(arm,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_arm_return;

    goto lbl_loop;
}

// 点胶轴运动
void GlueMotion(short glue)
{
    long glueStep;
    short cc;
    long clock;
    long sts;
    short axis;

lbl_loop:
    axis=glue-1;
    axis=1<<axis;

    // 等待点胶轴启动标志
lbl_wait_for_glue_start:
    cc = !glueStart;
    if(cc) goto lbl_wait_for_glue_start;

    // 清除点胶轴启动标志
    glueStart = 0;

    // 点胶轴运动
    GT_SetPos(glue,glueStep);
    GT_Update(axis);

    // 等待点胶轴运动停止
lbl_wait_for_glue_stop:
    GT_GetSts(glue,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_glue_stop;

    // 等待物料轴到达
lbl_wait_for_piece_arrival:
    cc = !pieceArrival;
    if(cc) goto lbl_wait_for_piece_arrival;

    // 点胶轴返回
```

```
GT_SetPos(glue,0);
GT_Update(axis);

// 置起摆臂轴启动标志
armStart = 1;

// 等待运动停止
lbl_wait_for_glue_return:
    GT_GetSts(glue,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_glue_return;

    goto lbl_loop;
}

// 物料轴运动
void PieceMotion(short piece)
{
    long pieceStep;
    short cc;
    long clock;
    long sts;
    long pos;
    short axis;
    pos = 0;

lbl_loop:
    axis=piece-1;
    axis=1<<axis;

    // 等待启动物料标志
lbl_wait_for_piece_start:
    cc = !pieceStart;
    if(cc) goto lbl_wait_for_piece_start;

    // 清除物料轴启动标志
    pieceStart = 0;

    // 清除物料轴到达标志
    pieceArrival = 0;

    // 启动物料轴运动
    pos = pos + pieceStep;
    GT_SetPos(piece,pos);
    GT_Update(axis);
```

```

// 等待物料轴运动停止
lbl_wait_for_piece_stop:
    GT_GetSts(piece,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_piece_stop;

// 置起物料轴到达标志
pieceArrival = 1;

goto lbl_loop;
}

```

应用程序负责编译、下载、初始化、启动运动程序。代码如下：

```

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

#define ARM            1
#define GLUE          2
#define PIECE         3

#define ARM_STEP      8000
#define GLUE_STEP     4000
#define PIECE_STEP    6000

#define ARM_VEL       10
#define GLUE_VEL      10
#define PIECE_VEL     10

int main(int argc, char* argv[])
{
    short rtn;
    TTrapPrm trap;
    TCompileInfo compile;
    short armMotion,glueMotion,pieceMotion;
    TVarInfo pieceArrival,pieceStart,glueStart,armStart;
    TVarInfo arm,armStep,glue,glueStep,piece,pieceStep;
    double value;
    double prfPos[8];

// 打开运动控制器
    rtn = GT_Open ();
    printf(" GT_Open ()=%d\n",rtn);

// 复位运动控制器
    rtn = GT_Reset ();

```

```
printf(" GT_Reset ()=%d\n",rtn);

// 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
rtn = GT_LoadConfig("test.cfg");
printf(" GT_LoadConfig()=%d\n",rtn);

// 清除各轴的报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts()=%d\n",rtn);

// 设置ARM运动参数
rtn = GT_PrflTrap(ARM);
printf(" GT_PrflTrap()=%d\n",rtn);
rtn = GT_GetTrapPrm(ARM,&trap);
printf(" GT_GetTrapPrm()=%d\n",rtn);
trap.acc = 0.25;
trap.dec = 0.25;
rtn = GT_SetTrapPrm(ARM,&trap);
printf(" GT_SetTrapPrm()=%d\n",rtn);
rtn = GT_SetVel(ARM,ARM_VEL);
printf(" GT_SetVel()=%d\n",rtn);
rtn = GT_Update(1<<(ARM-1));
printf(" GT_Update()=%d\n",rtn);

// 设置GLUE运动参数
rtn = GT_PrflTrap(GLUE);
printf(" GT_PrflTrap()=%d\n",rtn);
rtn = GT_GetTrapPrm(GLUE,&trap);
printf(" GT_GetTrapPrm()=%d\n",rtn);
trap.acc = 0.25;
trap.dec = 0.25;
rtn = GT_SetTrapPrm(GLUE,&trap);
printf(" GT_SetTrapPrm()=%d\n",rtn);
rtn = GT_SetVel(GLUE,GLUE_VEL);
printf(" GT_SetVel()=%d\n",rtn);
rtn = GT_Update(1<<(GLUE-1));
printf(" GT_Update()=%d\n",rtn);

// 设置PIECE运动参数
rtn = GT_PrflTrap(PIECE);
printf(" GT_PrflTrap()=%d\n",rtn);
rtn = GT_GetTrapPrm(PIECE,&trap);
printf(" GT_GetTrapPrm()=%d\n",rtn);
trap.acc = 0.25;
trap.dec = 0.25;
```

```
rtn = GT_SetTrapPrm(PIECE,&trap);
printf(" GT_SetTrapPrm()=%d\n",rtn);
rtn = GT_SetVel(PIECE,PIECE_VEL);
printf(" GT_SetVel()=%d\n",rtn);
rtn = GT_Update(1<<(PIECE-1));
printf(" GT_Update()=%d\n",rtn);

// 编译运动程序
// 编译成功以后生成扩展名为bin和ini的文件
// 必须保证error.ini文件位于工程文件夹中
rtn = GT_Compile("led.c",&compile);
printf("GT_Compile()=%d\n",rtn);

// 下载运动程序
rtn = GT_Download ("led.bin");
printf(" GT_Download ()=%d\n",rtn);

// 获取函数ID
rtn = GT_GetFunId ("ArmMotion",&armMotion);
printf(" GT_GetFunId ()=%d\n",rtn);

rtn = GT_GetFunId ("GlueMotion",&glueMotion);
printf(" GT_GetFunId ()=%d\n",rtn);

rtn = GT_GetFunId ("PieceMotion",&pieceMotion);
printf(" GT_GetFunId ()=%d\n",rtn);

// 获取全局变量ID
rtn = GT_GetVarId (NULL,"pieceArrival",&pieceArrival);
printf(" GT_GetVarId ()=%d\n",rtn);

rtn = GT_GetVarId (NULL,"pieceStart",&pieceStart);
printf(" GT_GetVarId ()=%d\n",rtn);

rtn = GT_GetVarId (NULL,"glueStart",&glueStart);
printf(" GT_GetVarId ()=%d\n",rtn);

rtn = GT_GetVarId (NULL,"armStart",&armStart);
printf(" GT_GetVarId ()=%d\n",rtn);

// 获取局部变量ID
rtn = GT_GetVarId ("ArmMotion","arm",&arm);
printf(" GT_GetVarId ()=%d\n",rtn);

rtn = GT_GetVarId ("ArmMotion","armStep",&armStep);
printf(" GT_GetVarId ()=%d\n",rtn);
```

```
rtn = GT_GetVarId ("GlueMotion","glue",&glue);
printf(" GT_GetVarId ()=%d\n",rtn);

rtn = GT_GetVarId ("GlueMotion","glueStep",&glueStep);
printf(" GT_GetVarId ()=%d\n",rtn);

rtn = GT_GetVarId ("PieceMotion","piece",&piece);
printf(" GT_GetVarId ()=%d\n",rtn);

rtn = GT_GetVarId ("PieceMotion","pieceStep",&pieceStep);
printf(" GT_GetVarId ()=%d\n",rtn);

// 绑定线程, 函数, 数据页
rtn = GT_Bind (0,armMotion,0);
printf(" GT_Bind ()=%d\n",rtn);

rtn = GT_Bind (1,glueMotion,1);
printf(" GT_Bind ()=%d\n",rtn);

rtn = GT_Bind (2,pieceMotion,2);
printf(" GT_Bind ()=%d\n",rtn);

// 初始化运动程序的全局变量
value = 0;
rtn = GT_SetVarValue (-1,&pieceArrival,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);

value = 1;
rtn = GT_SetVarValue (-1,&pieceStart,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);

value = 1;
rtn = GT_SetVarValue (-1,&glueStart,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);

value = 0;
rtn = GT_SetVarValue (-1,&armStart,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);

// 初始化运动程序的局部变量
value = ARM;
rtn = GT_SetVarValue (0,&arm,&value);
printf(" GT_SetVarValue ()=%d\n",rtn);
value = ARM_STEP;
rtn = GT_SetVarValue (0,&armStep,&value);
```

```

printf("  GT_SetVarValue (0)=%d\n",rtn);

value = GLUE;
rtn =  GT_SetVarValue (1,&glue,&value);
printf("  GT_SetVarValue (0)=%d\n",rtn);
value = GLUE_STEP;
rtn =  GT_SetVarValue (1,&glueStep,&value);
printf("  GT_SetVarValue (0)=%d\n",rtn);

value = PIECE;
rtn =  GT_SetVarValue (2,&piece,&value);
printf("  GT_SetVarValue (0)=%d\n",rtn);
value = PIECE_STEP;
rtn =  GT_SetVarValue (2,&pieceStep,&value);
printf("  GT_SetVarValue (0)=%d\n",rtn);

// 启动线程
rtn =  GT_RunThread (0);
printf("  GT_RunThread (0)=%d\n",rtn);

rtn =  GT_RunThread (1);
printf("  GT_RunThread (0)=%d\n",rtn);

rtn =  GT_RunThread (2);
printf("  GT_RunThread (0)=%d\n",rtn);

while(!kbhit())
{
    // 读取各轴规划位置
    GT_GetPrfPos(1,prfPos,8);

    printf("ARM=%-10.0lf GLUE=%-10.0lf PIECE=%-10.0lf", prfPos[ARM-1],  prfPos[GLUE-1],
           prfPos[PIECE-1]);
}
return 0;
}

```

10.3 语言元素

10.3.1 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 $(1/2)^{16}=0.0000152587890625$ 。

10.3.2 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。

10.3.3 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。

10.3.4 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。

10.3.5 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。

10.3.6 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。

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

10.4 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。

10.4.1 算数运算

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

10.4.2 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。

10.4.3 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

(!=)六种。参与比较的参数类型必须一致。

10.4.4 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。

10.5 流程控制

支持条件跳转、条件返回，语法规则和 C 语言相同。

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。

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。

第11章 其它指令



注意

本章表格中右侧的数字为“页码”，其中指令右侧的为“第 12 章 指令详细说明”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

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

11.1 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	页码
GT_Open	打开运动控制器	199
GT_Close	关闭运动控制器	161
GT_SetCardNo	切换当前运动控制器卡号	212
GT_GetCardNo	读取当前运动控制器卡号	174
GT_Reset	复位运动控制器	209

在使用运动控制器之前，首先需要使用 [GT_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 [GT_Close](#) 指令关闭运动控制器。

调用 [GT_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。

[GT_SetCardNo](#) 用于切换当前运动控制器卡号，一台计算机上使用多个运动控制器时，该指令用于指定当前运动控制器。当指令执行成功后，之后的所有指令只操作当前运动控制器。在多运动控制器系统中，每个运动控制器在操作系统启动时被分配一个卡号(0~15)，用于区别不同控制卡。卡号分配原则遵循 PNP 规则，第一个被系统识别的运动控制器卡号为 0，所以在硬件配置没有改变的情况下，系统每次分配的卡号是相同的。

11.2 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	页码
GT_GetVersion	读取运动控制器固件的版本号	192

为了方便用户核对运动控制器固件版本，提供 [GT_GetVersion](#) 指令来读取运动控制器固件版本号，版本号是一个含有 18 个字符的字符串：aaa bbbbbb ccc dddddd。具体的定义如表 11-3：

表 11-3 运动控制器固件版本号定义

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

例程 11-1 读取固件版本

调用 `GT_GetVersion` 指令的例程:

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

rtn = GT_Open ();

rtn = GT_GetVersion (&pVersion); // 读取版本号

printf(“%s\n”,pVersion);

rtn = GT_Close ();
```

11.3 读取系统时钟

表 11-4 读取系统时钟指令列表

指令	说明	页码
<code>GT_GetClock</code>	读取运动控制器系统时钟	174
<code>GT_GetClockHighPrecision</code>	读取运动控制器系统高精度时钟	175

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 `GT_GetClock` 指令可以读取该计数时钟的值。`GT_GetClockHighPrecision` 读取的时钟每 125 微秒增加 1，调用 `GT_Reset` 指令将会使计数时钟值清零。

11.4 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	页码
<code>GT_AxisOn</code>	打开驱动器使能	156
<code>GT_AxisOff</code>	关闭驱动器使能	151

调用 `GT_AxisOn` 指令将打开指定控制轴所连电机的伺服使能信号，使指定控制轴进入控制状态。如果在系统配置时，没有数字量输出与该 axis 关联，则该指令将会无效(参见 4.2.8 配置 do)。如果运动控制器被配置成了伺服控制方式(参见第 4 章系统配置)，那么必须首先设置指定轴的位置环的 PID 参数。

11.5 维护位置值

表 11-6 维护位置值函数列表

指令	说明	页码
<code>GT_SetPrfPos</code>	修改指定轴的规划位置	222
<code>GT_SynchAxisPos</code>	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	229
<code>GT_ZeroPos</code>	清零规划位置和实际位置，并进行零漂补偿	230

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 `GT_SetPrfPos` 指令或者 `GT_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 `GT_SynchAxisPos` 指令。

11.6 电机到位检测

表 11-7 电机到位检测函数列表

指令	说明	页码
GT_SetAxisBand	设置轴到位误差带 规划器静止，规划位置 and 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	210
GT_GetAxisBand	读取轴到位误差带	170

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 `GT_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 5.2.1)，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 5.2.1)。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。

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

调用 `GT_ZeroPos` 进行对位置进行清零，同时进行自动零漂补偿。

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。

例程 11-2 电机到位检测

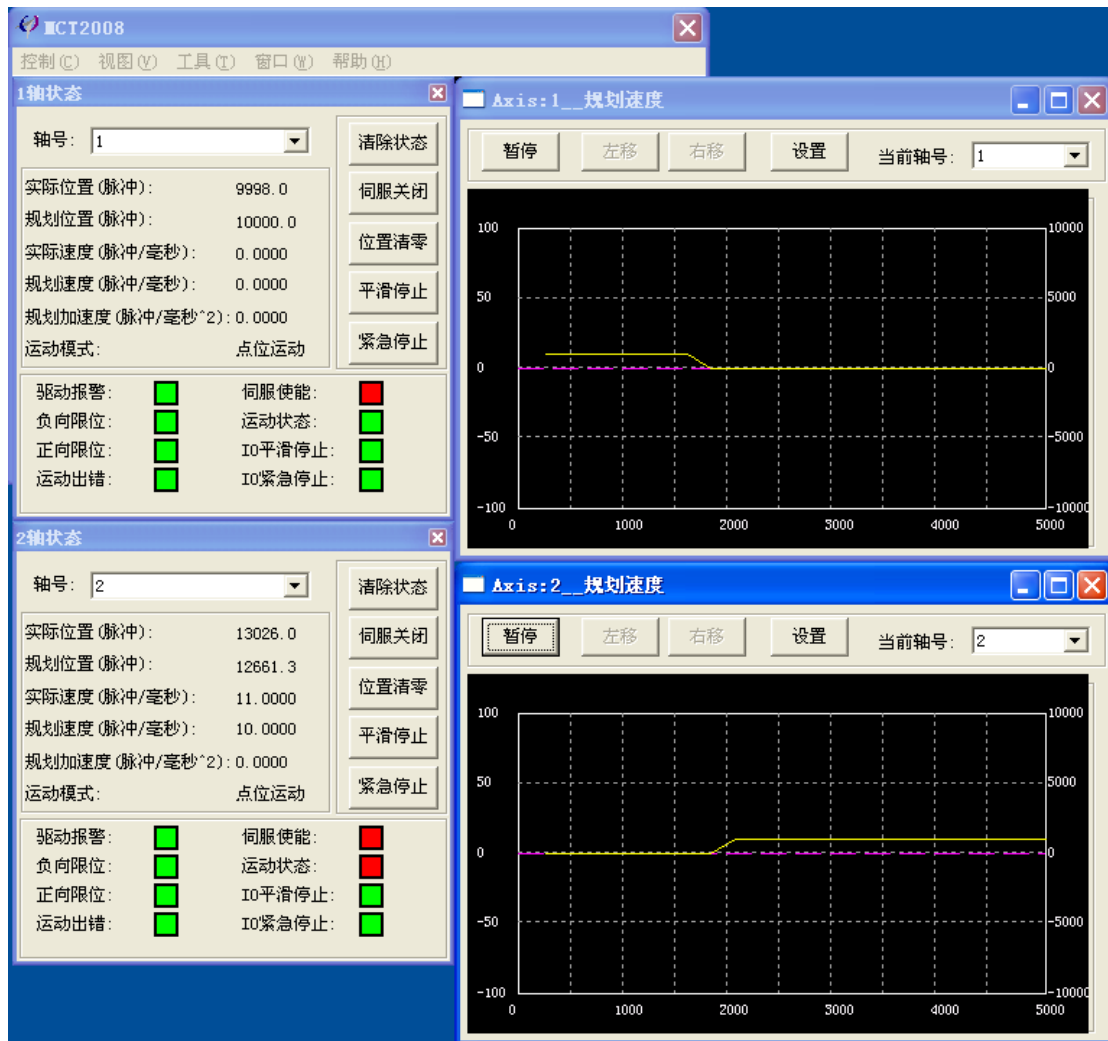


图 11-1 运动状态检测

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

#define AXIS_X      1
#define AXIS_Y      2

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

    // 打开运动控制器
    rtn = GT_Open();
```

```
printf(" GT_Open()=%d\n",rtn);

// 复位控制器
rtn = GT_Reset();
printf(" GT_Reset()=%d\n",rtn);

// 配置运动控制器为伺服模式
rtn = GT_LoadConfig("servo.cfg");
printf(" GT_LoadConfig()=%d\n",rtn);

// 延时一段时间
Sleep(100);

// 清除各轴的报警和限位
rtn = GT_ClrSts(1,8);
printf(" GT_ClrSts()=%d\n",rtn);

// 读取X轴PID参数
rtn = GT_GetPid(Axis_X,1,&pid);
printf(" GT_GetPid()=%d\n",rtn);
pid.kp = 10;
// 更新X轴PID参数
rtn = GT_SetPid(Axis_X,1,&pid);
printf(" GT_SetPid()=%d\n",rtn);

// 读取Y轴PID参数
rtn = GT_GetPid(Axis_Y,1,&pid);
printf(" GT_GetPid()=%d\n",rtn);
pid.kp = 10;
// 更新Y轴PID参数
rtn = GT_SetPid(Axis_Y,1,&pid);
printf(" GT_SetPid()=%d\n",rtn);

// X轴伺服使能
rtn = GT_AxisOn(Axis_X);
printf(" GT_AxisOn()=%d\n",rtn);

// Y轴伺服使能
rtn = GT_AxisOn(Axis_Y);
printf(" GT_AxisOn()=%d\n",rtn);

// 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
rtn = GT_ZeroPos(Axis_X);
```

```
printf(" GT_ZeroPos()=%d\n",rtn);

// 设置X轴到位误差带
rtn = GT_SetAxisBand(Axis_X,20,5);
printf(" GT_SetAxisBand()=%d\n",rtn);

// 位置清零,并进行自动零漂补偿
rtn= GT_ZeroPos(Axis_Y);
printf(" GT_ZeroPos()=%d\n",rtn);

// 设置Y轴到位误差带
rtn = GT_SetAxisBand(Axis_Y,20,5);
printf(" GT_SetAxisBand()=%d\n",rtn);

// X轴设为点位模式
rtn = GT_PrftTrap(Axis_X);
printf(" GT_PrftTrap()=%d\n",rtn);

// 读取X轴点位运动参数
rtn = GT_GetTrapPrm(Axis_X,&trap);
printf(" GT_GetTrapPrm()=%d\n",rtn);
trap.acc = 1;
trap.dec = 0.5;
// 设置X轴点位运动参数
rtn = GT_SetTrapPrm(Axis_X,&trap);
printf(" GT_SetTrapPrm()=%d\n",rtn);

// 设置X轴的目标速度
rtn = GT_SetVel(Axis_X,10);
printf(" GT_SetVel()=%d\n",rtn);

// Y轴设为点位模式
rtn = GT_PrftTrap(Axis_Y);
printf(" GT_PrftTrap()=%d\n",rtn);

// 读取Y轴点位运动参数
rtn = GT_GetTrapPrm(Axis_Y,&trap);
printf(" GT_GetTrapPrm()=%d\n",rtn);
trap.acc = 1;
trap.dec = 0.5;
// 设置Y轴点位运动参数
rtn = GT_SetTrapPrm(Axis_Y,&trap);
printf(" GT_SetTrapPrm()=%d\n",rtn);

// 设置Y轴的目标速度
rtn = GT_SetVel(Axis_Y,10);
```



```
printf("  GT_SetVel()=%d\n",rtn);
posX = 10000;
posY = 20000;

while(!kbhit())
{
    // 设置X轴目标位置
    rtn =  GT_SetPos(Axis_X,posX);
    printf("  GT_SetPos()=%d\n",rtn);

    // 启动X轴的运动
    rtn =  GT_Update(1<<(Axis_X-1));
    printf("  GT_Update()=%d\n",rtn);

    posX = - posX;

    // 等待X轴进入误差带
    do
    {
        GT_GetSts(Axis_X,&sts);
        GT_GetPrfPos(Axis_X,&prfPos);
        GT_GetPrfVel(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
    rtn =  GT_SetPos(Axis_Y,posY);
    printf("  GT_SetPos()=%d\n",rtn);

    // 启动Y轴的运动
    rtn =  GT_Update(1<<(Axis_Y-1));
    printf("  GT_Update()=%d\n",rtn);

    posY = - posY;

    // 等待Y轴进入误差带
    do
    {
        GT_GetSts(Axis_Y,&sts);
        GT_GetPrfPos(Axis_Y,&prfPos);
        GT_GetPrfVel(Axis_Y,&prfVel);
        printf("y pos=%-10.2lf vel=%-6.2lf\r",prfPos,prfVel);
    }while( 0x800 != ( sts & 0x800 ) );

    printf("\n");
}
```

```

}
return 0;
}

```

11.7 设置 PID 参数

表 11-8 设置 PID 参数函数列表

指令	说明	页码
GT_SetControlFilter	设定 PID 索引, 支持 3 组 PID 参数	212
GT_GetControlFilter	读取当前 PID 索引	175
GT_SetPid	设置 PID 参数	221
GT_GetPid	读取 PID 参数	184

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

11.8 反向间隙补偿

表 11-9 反向间隙补偿函数列表

指令	说明	页码
GT_SetBacklash	设置反向间隙补偿的相关参数	210
GT_GetBacklash	读取反向间隙补偿的相关参数	173

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 GT_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 GT_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 GT_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 GT_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 GT_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 自动回原点功能

11.9.1 指令列表

表 11-10 自动回原点功能函数列表

指令	说明	页码
GT_HomeInit	初始化自动回原点功能	193
GT_Home	启动自动回原点功能	192
GT_Index	设置自动回原点功能为 home+index 模式	194
GT_HomeStop	启动原点停止功能	193
GT_HomeSts	查询自动回原点的运行状态	193

11.9.2 重点说明

(1) 使用前的注意事项

- 1) 在实际应用中，使用者在进行多轴同时回零操作时，需确保不会因为同时回零而造成多轴之间干涉。
- 2) 在使用本指令时，请确保没有同时使用 GUS 中运动程序的功能。自动回原点功能与控制器的运动程序功能共用了运动控制器内的一些资源，所以这两个功能不能同时使用，如果在运动程序的使用过程中使用了自动回原点功能，则再次使用运动程序时，需要重新下载运动程序代码以及相关的初始化操作。如果在使用自动回原点功能的使用过程中使用了运动程序功能，则再次使用自动回原点功能前需要再次调用 `GT_HomeInit` 来进行自动回原点功能的初始化。
- 3) 在使用过程中，需保证电机规划位置与编码器位置同向，即当规划位置增大时，编码器位置也在增大。
- 4) 在自动回原点过程执行完毕后，进行零点清除前，需要设置一个延时操作，防止由于电机未到位而产生误差，延时操作一般要大于 100ms。

(2) 使用方法

自动回原点指令函数共有五个函数，`GT_HomeInit`，`GT_Home`，`GT_Index`，`GT_HomeStop`，`GT_HomeSts`。其作用分别是 Home 回零模式初始化，Home 模式回零，Home+Index 模式回零，原点急停指令，以及查询回原点状态指令。

- 1) 自动回原点功能的初始化

// 在进行所有轴的回原点操作之前，都需要进行自动回原点功能的初始化

```
rtm = GT_Open();
```

```
rtm = GT_Reset();
```

```
rtm = GT_HomeInit()
```

// 自动回原点功能初始化操作在每次打开卡时只需要调用一次，最好不要频繁调用

- 2) Home 信号回原点使用方法

```

rtn = GT_AxisOn(axis);

rtn = GT_Home(axis,pos,vel,acc,offset);

```

3) Home+Index 信号回原点使用方法

```

rtn = GT_AxisOn(axis);

rtn = GT_Index(axis,pos,offset);

rtn = GT_Home(axis,pos,vel,acc,offset);

```

4) HomeStop 急停操作

```

rtn = GT_AxisOn(axis);

rtn = GT_HomeStop(axis,pos,vel,acc);

```

在指令执行过程中，可以随时调用 `GT_HomeSts` 来查询当前清零操作的执行结果。当状态值为 0 时表示当前正在运行状态，当状态值为 1 时表示操作完成。

(3) 例程**例程 11-3 自动回原点**

```

#include "stdafx.h"
#include "gts.h"

int main(int argc, char* argv[])
{
    short rtn;
    unsigned short sts1,sts2;
    rtn = GT_Open (); // 打开运动控制器
    rtn = GT_Reset (); // 复位运动控制器
    rtn = GT_LoadConfig("test.cfg"); // 下载配置文件
    rtn = GT_ClrSts(1,8); // 清楚状态
    rtn = GT_HomeInit (); // 初始化自动回原点功能
    rtn = GT_AxisOn (1); // 使能轴
    rtn = GT_AxisOn (2); // 使能轴
    rtn = GT_Index (1,20000,2000); // 轴为Home+Index回零模式
    rtn = GT_Home (1,200000,50,0.5,2000);
    rtn = GT_Home (2,200000,50,0.5,3000); // 轴为Home回零模式
    while (!kbhit())
    {
        rtn = GT_HomeSts (1,&sts1); // 查询返回状态
        rtn = GT_HomeSts (2,&sts2);
        printf("%d %d %d\r",sts1,sts2,rtn);
    }
    getch();
    return 0;
}

```

11.10 控制器断电保护

11.10.1 应用原理

现场设备应用少不了对重要数据的采集，而采集来的数据有时要写入文件系统中，用于设备运行状况的分析或是系统调度的参考依据。但有时写入数据的时候突然断电，这时如果没有断电保护可能会引起数据丢失，严重会导致控制器系统出现故障；为了防止突然断电引起故障，固高提供了一个断电保持设备，保持设备在没有断电的情况下提供了一个 IO 输出信号到控制器用于实时监测，只有在监测到这个信号有效时（高电平有效）才可以去写文件，而当控制器监测断电保持设备的输出信号无效时就要停止写文件，同时关闭打开了的文件。保持设备可供选择型号：GPSM02。

11.10.2 断电保护程序

(1) 重要说明

控制器实时监测断电保持设备的输出信号为IO信号，此信号接入控制器的任意通用输入信号端。例如接入控制器的输入DI0，则调用函数：`GT_GetDi (MC_GPI,ADR(GpiDi))`进行实时监测，此时GpiDi.0即是监测的信号，此信号正常时是0,可以对文件进行写操作，而当此信号为1时不可以对文件进行写操作(当出现低电平的第一个周期不予保护，第二个周期出现才保护，即对信号进行滤波，如20ms或200ms滤波)，同时关闭已经打开的文件；注：此控制方法也适用于C，C++编写的控制程序；

(2) 例程

ST 编程例程

```
PROGRAM PLC_PRG
VAR
```

```
  hFile: DWORD;
  WriteBuffer: ARRAY[1..100000] OF BYTE := 100000(100);
  dwWritten: DWORD;
  GpiDi: DINT;
  bClose_F: F_TRIG;
  bWriteFile: BOOL;
  bFileOpenTag: BOOL;
  bCloseFile: BOOL;
```

```
END_VAR
```

(*捕获断电保持设备输出信号，10ms 以上滤波处理，打开文件写入数据*)

```
  bOpen_Ton(IN:=NOT GpiDi.0 , PT:= T#10ms, Q=> , ET=> );
  IF bOpen_Ton.Q AND bWriteFile THEN
  IF NOT bFileOpenTag THEN (*文件是否打开标记*)
    hFile := SysFileOpen('\Hard Disk\Test.txt', 'w');(*打开文件*)
    bFileOpenTag:=TRUE;
  END_IF
  dwWritten:=SysFileWrite(hFile, ADR(WriteBuffer), SIZEOF(WriteBuffer));(*写入数据*)
  END_IF
```

(*捕获断电保持设备输出信号为低电平，则 10ms 以上滤波处理*)

```

bClose_Ton(IN:=GpiDi.0 AND bFileOpenTag, PT:= T#10ms, Q=>, ET=>);
(*关闭已经打开的文件*)
IF (bClose_Ton.Q OR bCloseFile) AND bFileOpenTag THEN
    SysFileClose(hFile);      (*关闭打开的文件*)
    bFileOpenTag:=FALSE;     (*清除文件打开标记位*)
END_IF

```

C 编程例程

```

#include "stdafx.h"
#include <windows.h>
#include "gts.h"
int _tmain(int argc, _TCHAR* argv[])
{
    short rtn=0;
    short FileOpenTag=0;
    long GpiDi=0;
    char buffer[]="File content";
    FILE *hFile;
    rtn= GT_Open ();
    printf(" GT_Open ()=%d\n",rtn);
    while(1)
    {
        // 读取数字量输入
        rtn= GT_GetDi (MC_GPI,&GpiDi);
        // 保持设备的输出接入到数字量输入第一通道，实时监测其状态，状态为低电平时允许写文件，否则不可以;
        if(GpiDi & 0x1==0)
        {
            if (FileOpenTag==0)
            {
                // 打开文件
                hFile=fopen("\\Hard Disk\\Test.txt", "w+");
                FileOpenTag=1;
                printf("打开文件");
            }

            // 写数据到文件
            fwrite(buffer, sizeof(buffer),1,hFile);
            printf("写入数据");

        }
        else
        {
            printf("进入掉电处理");
            // 保持设备的输出信号出现高电平，则延时ms（即滤波）后再次判断是否是低电平
            Sleep(100);
        }
    }
}

```

```

rtn=  GT_GetDi (MC_GPI,&GpiDi);
if(FileOpenTag==1 && (GpiDi & 0x1==1))
{
    fclose(hFile);
    FileOpenTag=0;
    printf("关闭文件");
}
}
}
return 0;
}

```

(3) 写数据耗时记录

经过测试每次写入最大数据为 2M (Byte)，如果控制器在未写入时 CPU 负载大于 74%时要减小每次写入的数据大小：

注：一次写入文件的最大数据不能超过 2M (Byte)；每次写入数据的大小以及 CPU 负载不同所需要的时间不同，测试如下：

每次写入数据大小 (Byte)	每次操作过程	最小用时(ms)	最大用时(ms)	CPU 负载
1K	打开文件，写入，关闭文件	18	26	74%
30K	打开文件，写入，关闭文件	36	45	74%
50K	打开文件，写入，关闭文件	52	70	74%
100K	打开文件，写入，关闭文件	81	205	74%
300K	打开文件，写入，关闭文件	210	350	74%
500K	打开文件，写入，关闭文件	350	600	74%
1M	打开文件，写入，关闭文件	720	850	74%
2M	打开文件，写入，关闭文件	1300	1800	74%
1k	打开文件，写入，关闭文件	6.2	13	4%
30K	打开文件，写入，关闭文件	8	18	4%
50K	打开文件，写入，关闭文件	8	21	4%
100k	打开文件，写入，关闭文件	10	21	4%
300k	打开文件，写入，关闭文件	10	22	4%
500K	打开文件，写入，关闭文件	110	320	4%
1M	打开文件，写入，关闭文件	220	450	4%
2M	打开文件，写入，关闭文件	410	560	4%

第12章 指令详细说明



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。

指令 1 GT_AlarmOff

指令原型	short GT_AlarmOff(short axis)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	29
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
	控制轴号。		
axis	正整数。对于 GUS-400-TG02 系列的控制器，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制器，取值范围：[1, 6]。		
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	GT_AlarmOn		
指令示例	无。		

指令 2 GT_AlarmOn

指令原型	short GT_AlarmOn(short axis)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	29
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
	控制轴号。		
axis	正整数。对于 GUS-400-TG02 系列的控制器，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制器，取值范围：[1, 6]。		
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	GT_AlarmOff		
指令示例	无。		

指令 3 GT_ArcXYC

指令原型	short GT_ArcXYC(short crd, long x, long y, double xCenter, double yCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		

x	圆弧插补 x 轴的终点坐标值。取值范围: [-1073741823, 1073741823], 单位: pulse。
y	圆弧插补 y 轴的终点坐标值。取值范围: [-1073741823, 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 6-10 圆弧插补

指令 4 GT_ArcXYR

指令原型	short GT_ArcXYR (short crd, long x, long y, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 9 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		
x	圆弧插补 x 轴的终点坐标值。取值范围: [-1073741823, 1073741823], 单位: pulse。		
y	圆弧插补 y 轴的终点坐标值。取值范围: [-1073741823, 1073741823], 单位: pulse。		
radius	圆弧插补的圆弧半径值。取值范围: [-1073741823, 1073741823], 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:		

	<p>(1) 检查当前坐标系是否映射了相关轴。</p> <p>(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。</p> <p>(3) 检查相应的 fifo 是否已满。</p> <p>其他返回值：请参照指令返回值列表。</p>
相关指令	无。
指令示例	例程 6-10 圆弧插补

指令 5 GT_ArcYZC

指令原型	short GT_ArcYZC (short crd, long y, long z, double yCenter, double zCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 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	<p>若返回值为 1：</p> <p>(1) 检查当前坐标系是否映射了相关轴。</p> <p>(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。</p> <p>(3) 检查相应的 fifo 是否已满。</p> <p>其他返回值：请参照指令返回值列表。</p>		
相关指令	无。		
指令示例	无。		

指令 6 GT_ArcYZR

指令原型	short GT_ArcYZR (short crd, long y, long z, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 9 个参数，参数的详细信息如下。		

crd	坐标系号。正整数，取值范围：[1, 2]。
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
radius	圆弧插补的圆弧半径值。取值范围：[-1073741823, 1073741823]，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	无。

指令 7 GT_ArcZXC

指令原型	short GT_ArcZXC (short crd, long z, long x, double zCenter, double xCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741823, 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	<p>若返回值为 1:</p> <ol style="list-style-type: none"> (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。 <p>其他返回值: 请参照指令返回值列表。</p>
相关指令	无。
指令示例	无。

指令 8 GT_ArcZXR

指令原型	short GT_ArcZXR (short crd, long z, long x, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 9 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		
z	圆弧插补 z 轴的终点坐标值。取值范围: [-1073741823, 1073741823], 单位: pulse。		
x	圆弧插补 x 轴的终点坐标值。取值范围: [-1073741823, 1073741823], 单位: pulse。		
radius	<p>圆弧插补的圆弧半径值。取值范围: [-1073741823, 1073741823], 单位: pulse。</p> <p>半径为正时, 表示圆弧为小于等于 180°圆弧。</p> <p>半径为负时, 表示圆弧为大于 180°圆弧。</p> <p>半径描述方式不能用来描述整圆。</p>		
circleDir	<p>圆弧的旋转方向。</p> <p>0: 顺时针圆弧。</p> <p>1: 逆时针圆弧。</p>		
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	<p>若返回值为 1:</p> <ol style="list-style-type: none"> (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。 <p>其他返回值: 请参照指令返回值列表。</p>		
相关指令	无。		
指令示例	无。		

指令 9 GT_AxisOff

指令原型	short GT_AxisOff(short axis)
适用板卡	GUS-400-TG02,GUS-600-TG02
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	139
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
axis	关闭伺服使能的轴的编号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_AxisOn		
指令示例	例程 6-1 点位运动		

指令 10 GT_AxisOn

指令原型	short GT_AxisOn(short axis)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	139
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
axis	打开伺服使能的轴的编号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
指令返回值	<p>若返回值为 1：</p> <p>(1) 若在配置文件中报警有效，请检查驱动器是否有报警。</p> <p>若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。</p> <p>(2) 在闭环控制模式下，采用 MCT2008 对控制器配置，请检查配置 control 一项中是否选择了关联，若没有，选择关联。若已选择，请检查配置 encoder 一项是否选择激活，若没有，选择激活。若已选择，则请检查 PID 参数中的 Kp 是否设置为 0，Kp 必须大于 0，调试阶段可设为一个较小值 5。</p> <p>其他返回值：请参照指令返回值列表。</p>		
相关指令	GT_AxisOff		
指令示例	例程 6-14 PVT 描述方式		

指令 11 GT_Bind

指令原型	short GT_Bind(short thread, short funId, short page)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	120
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
thread	线程编号，取值范围：[0, 31]。		
funId	函数标识，可以调用 GT_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 单线程累加求和		

指令 12 GT_BufDA

指令原型	short GT_BufDA (short crd, short chn, short daValue, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
chn	模拟量输出的通道号。取值范围：[1, 6]。		
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： <ol style="list-style-type: none"> (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 GT_BufDelay

指令原型	short GT_BufDelay (short crd, unsigned short delayTime, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
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： <ol style="list-style-type: none"> (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 6-9 直线插补		

指令 14 GT_BufGear

指令原型	short GT_BufGear (short crd, short gearAxis, long pos, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
gearAxis	需要进行跟随运动的轴号，取值范围：[1, 6]。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	<p>若返回值为 1：</p> <ol style="list-style-type: none"> (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。 <p>其他返回值：请参照指令返回值列表。</p>
相关指令	无。
指令示例	例程 6-13 跟随运动中的刀向跟随

指令 15 GT_BufIO

指令原型	short GT_BufIO (short crd, unsigned short doType, unsigned short doMask, unsigned short doValue, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
doType	<p>数字量输出的类型。</p> <p>MC_ENABLE(该宏定义为 10)：输出驱动器使能。</p> <p>MC_CLEAR(该宏定义为 11)：输出驱动器报警清除。</p> <p>MC_GPO(该宏定义为 12)：输出通用输出。</p>		
doMask	<p>从 bit0~bit15 按位表示指定的数字量输出是否有操作。</p> <p>0：该路数字量输出无操作。1：该路数字量输出有操作。</p>		
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	<p>若返回值为 1：</p> <ol style="list-style-type: none"> (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。 <p>其他返回值：请参照指令返回值列表。</p>		
相关指令	无。		
指令示例	例程 6-9 直线插补		

指令 16 GT_BufLmtsOff

指令原型	short GT_BufLmtsOff (short crd, short axis, short limitType, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
axis	需要将限位无效的轴的编号。取值范围：[1, 6]。		

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	GT_BufLmtsOn
指令示例	无。

指令 17 GT_BufLmtsOn

指令原型	short GT_BufLmtsOn(short crd, short axis, short limitType, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 4 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		
axis	需要将限位有效的轴的编号, 取值范围: [1, 6]。		
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	GT_BufLmtsOff		
指令示例	无。		

指令 18 GT_BufMove

指令原型	short GT_BufMove (short crd, short moveAxis, long pos, double vel, double acc, short modal, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 7 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		

moveAxis	需要进行点位运动的轴号，取值范围：[1, 6]。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 6-12 点位运动中的刀向跟随

指令 19 GT_BufSetStopIo

指令原型	short GT_BufSetStopIo (short crd, short axis, short stopType, short inputType, short inputIndex, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
axis	需要设置停止 IO 信息的轴的编号。取值范围：[1, 6]。		
stopType	需要设置停止 IO 信息的停止类型。 0：紧急停止类型。 1：平滑停止类型。		
inputType	设置的数字量输入的类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。		
inputIndex	设置的数字量输入的索引号，取值范围根据 inputType 的取值而定。 当 inputType= MC_LIMIT_POSITIVE 时，取值范围：[1, 6]。 当 inputType= MC_LIMIT_NEGATIVE 时，取值范围：[1, 6]。 当 inputType= MC_ALARM 时，取值范围：[1, 6]。 当 inputType= MC_HOME 时，取值范围：[1, 6]。 当 inputType= MC_GPI 时，取值范围：[1, 16]。 当 inputType= MC_ARRIVE 时，取值范围：[1, 4]。		
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 20 GT_ClearCaptureStatus

指令原型	short GT_ClearCaptureStatus(short encoder)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	清除捕获状态。		
指令类型	立即指令,调用后立即生效。	章节页码	105
指令参数	该指令共有 1 个参数,参数的详细信息如下。		
	需要被清除捕获状态的编码器轴号。		
encoder	正整数。对于 GUS-400-TG02 系列的控制卡,取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡,取值范围: [1, 6]。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 21 GT_Close

指令原型	short GT_Close()		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	138
指令参数	无。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_Open		
指令示例	无。		

指令 22 GT_ClrSts

指令原型	short GT_ClrSts(short axis, short count=1)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	清除驱动器报警标志、跟随误差超限标志、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,才能清除跟随误差超限标志; 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	31
指令参数	该指令共有 2 个参数,参数的详细信息如下。		
	起始轴号。		
axis	正整数。对于 GUS-400-TG02 系列的控制卡,取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡,取值范围: [1, 6]。		

count	读取的轴数，默认为 1。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
指令返回值	请参照指令返回值列表。
相关指令	GT_GetSts
指令示例	例程 6-1 点位运动

指令 23 GT_CrdClear

指令原型	short GT_CrdClear(short crd, short fifo)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
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 6-9 直线插补		

指令 24 GT_CrdData

指令原型	short GT_CrdData (short crd, TCrdData *pCrdData, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
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 GT_CrdSpace 检查）。当检查运动缓存区有空间时，再次调用 GT_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 6-11 前瞻预处理		

指令 25 GT_CrdSpace

指令原型	short GT_CrdSpace (short crd, long *pSpace, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
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 6-9 直线插补		

指令 26 GT_CrdStart

指令原型	short GT_CrdStart (short mask, short option)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
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 6-9 直线插补		

指令 27 GT_CrdStatus

指令原型	short GT_CrdStatus (short crd, short *pRun, long *pSegment, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
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 GT_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 指令示例
无。
例程 6-9 直线插补

指令 28 GT_CtrlMode

指令原型	short GT_CtrlMode(short axis, short mode)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置控制轴为模拟量输出或脉冲输出。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	控制轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
mode	切换的模式。 0：将指定轴切换为闭环控制模式(即电压控制方式)。 1：将指定轴切换为开环控制模式(即脉冲控制方式)。		
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 29 GT_Download

指令原型	short GT_Download(char *pFileName)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	120
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
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 单线程累加求和		

指令 30 GT_EncOff

指令原型	short GT_EncOff(short encoder)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置为“脉冲计数器”计数方式。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 1 个参数，参数的详细信息如下。		

encoder	编码器通道号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
指令返回值	请参照指令返回值列表
相关指令	GT_EncOn
指令示例	无。

指令 31 GT_EncOn

指令原型	short GT_EncOn(short encoder)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置为“外部编码器”计数方式。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 1 个参数，参数的详细信息如下： 编码器通道号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
encoder	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
指令返回值	请参照指令返回值列表		
相关指令	GT_EncOff		
指令示例	无。		

指令 32 GT_EncScale

指令原型	short GT_EncScale(short axis, short alpha, short beta)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	29
指令参数	该指令共有 3 个参数，参数的详细信息如下。 控制轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。 规划器当量的alpha值，取值范围：(-32767, 0)和(0, 32767)。 规划器当量的beta值，取值范围：(-32767, 0)和(0, 32767)。		
axis	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
alpha	规划器当量的alpha值，取值范围：(-32767, 0)和(0, 32767)。		
beta	规划器当量的beta值，取值范围：(-32767, 0)和(0, 32767)。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_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	GT_ProfileScale		
指令示例	无。		

指令 33 GT_EncSns

指令原型	short GT_EncSns(unsigned short sense)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置编码器的计数方向。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 1 个参数，参数的详细信息如下。 按位标识编码器的计数方向。 对于GUS-600-TG02系列的控制卡：bit0~bit5依次对应编码器1~6。		
sense	按位标识编码器的计数方向。 对于GUS-600-TG02系列的控制卡：bit0~bit5依次对应编码器1~6。		

	对于GUS-400-TG02系列的控制卡：bit0~bit3依次对应编码器1~4。 0：该编码器计数方向不取反。 1：该编码器计数方向取反。 系统复位后，默认为编码器计数方向取反。
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 34 GT_FollowClear

指令原型	short GT_FollowClear (short profile, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	清除 Follow 运动模式指定 FIFO 中的数据。 运动状态下该指令无效。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
fifo	指定需要清除的 FIFO，取值范围：0、1 两个值。默认为 0。		
指令返回值	若返回值为 1： 请检查当前轴是否为 Follow 模式，若不是，请先调用 GT_PrFFollow 将当前轴设置为 Follow 模式。 (1) 请检查要清除的 FIFO 是否正在使用，运动是否结束。 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 6-6 Follow 单 FIFO，例程 6-7 Follow 双 FIFO		

指令 35 GT_FollowData

指令原型	short GT_FollowData (short profile, long masterSegment, double slaveSegment, short type=FOLLOW_SEGMENT_NORMAL, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	向 Follow 运动模式指定 FIFO 增加数据。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
masterSegment	主轴位移。单位：pulse。		
slaveSegment	从轴位移。单位：pulse。		
type	数据段类型。 FOLLOW_SEGMENT_NORMAL（该宏定义为 0）普通段。默认为该类型。 FOLLOW_SEGMENT_EVEN（该宏定义为 1）匀速段。 FOLLOW_SEGMENT_STOP（该宏定义为 2）减速到 0 段。 FOLLOW_SEGMENT_CONTINUE（该宏定义为 3）保持 FIFO 之间速度连续。		

fifo	指定存放数据的 FIFO，取值范围：0、1 两个值。默认为 0。
指令返回值	若返回值为 1： 请检查当前轴是否为 Follow 模式，若不是，请先调用 <code>GT_PrFFollow</code> 将当前轴设置为 Follow 模式。 (1) 请检查是否有足够的空间放新的数据。 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 6-6 Follow 单 FIFO，例程 6-7 Follow 双 FIFO

指令 36 GT_FollowSpace

指令原型	short GT_FollowSpace (short profile, short *pSpace, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	查询 Follow 运动模式指定 FIFO 的剩余空间。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pSpace	读取 FIFO 的剩余空间。 说明此空间的含义。		
fifo	指定所要查询的 FIFO，取值范围：0、1 两个值。默认为 0。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 <code>GT_PrFFollow</code> 将当前轴设置为 Follow 模式。 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 6-6 Follow 单 FIFO，例程 6-7 Follow 双 FIFO		

指令 37 GT_FollowStart

指令原型	short GT_FollowStart (long mask, long option)						
适用板卡	GUS-400-TG02,GUS-600-TG02						
指令说明	启动 Follow 运动。						
指令类型	立即指令，调用后立即生效。	章节页码	51				
指令参数	该指令共有 2 个参数，参数的详细信息如下。						
mask	按位指示需要启动 Follow 运动的轴号。当 bit 位为 1 时表示启动对应的轴。 对于 GUS-400-TG02 系列的控制卡：						
	Bit	3	2	1	0		
	对应轴	4 轴	3 轴	2 轴	1 轴		
	对于 GUS-600-TG02 系列的控制卡：						
	Bit	5	4	3	2	1	0
	对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
option	按位指示所使用的 FIFO，默认为 0。当 bit 位为 0 时表示对应的轴使用 FIFO1。当 bit 位为 1 时表示对应的轴使用 FIFO2。 对于 GUS-400-TG02 系列的控制卡：						
	Bit	3	2	1	0		

	对应轴	4 轴	3 轴	2 轴	1 轴	
	对于 GUS-600-TG02 系列的控制卡:					
	Bit	5	4	3	2	1 0
	对应轴	6 轴	5 轴	4 轴	3 轴	2 轴 1 轴
指令返回值	若返回值为 1: 请检查当前轴是否为 Follow 模式, 若不是, 请先调用 GT_PrFFollow 将当前轴设置为 Follow 模式。 (1) 检查运动是否结束, 运动进行时, 指令调用会失败; (2) 检查相应轴是否设置了跟随主轴; (3) 检查 FIFO 是否有数据; (4) 检查 mask 参数是否设置了启动相应的轴。 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 6-6 Follow 单 FIFO, 例程 6-7 Follow 双 FIFO					

指令 38 GT_FollowSwitch

指令原型	short GT_FollowSwitch(long mask)						
适用板卡	GUS-400-TG02,GUS-600-TG02						
指令说明	切换 Follow 运动模式所使用的 FIFO。						
指令类型	立即指令, 调用后立即生效。					章节页码	51
指令参数	该指令共有 1 个参数, 参数的详细信息如下。 按位指示需要切换 Follow 工作 FIFO 的轴号。当 bit 位为 1 时表示切换对应的轴的 FIFO。 对于 GUS-400-TG02 系列的控制卡:						
mask	Bit	3	2	1	0		
	对应轴	4 轴	3 轴	2 轴	1 轴		
mask	对于 GUS-600-TG02 系列的控制卡:						
	Bit	5	4	3	2	1 0	
mask	对应轴	6 轴	5 轴	4 轴	3 轴	2 轴 1 轴	
	指令返回值	若返回值为 1: 请检查当前轴是否为 Follow 模式, 若不是, 请先调用 GT_PrFFollow 将当前轴设置为 Follow 模式。 (1) 检查运动是否进行, 只有运动中才能切换。 (2) 检查目标 FIFO 是否为空。 (3) 检查 mask 参数是否设置了启动相应的轴。 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 6-6 Follow 单 FIFO, 例程 6-7 Follow 双 FIFO						

指令 39 GT_GearStart

指令原型	short GT_GearStart(long mask)						
适用板卡	GUS-400-TG02,GUS-600-TG02						
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	48
指令参数	该指令共有 1 个参数, 参数的详细信息如下。						

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。						
	对于 GUS-400-TG02 系列的控制卡：						
	Bit	3	2	1	0		
	对应轴	4 轴	3 轴	2 轴	1 轴		
指令返回值	对于 GUS-600-TG02 系列的控制卡：						
	Bit	5	4	3	2	1	0
	对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
	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 GT_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 6-5 电子齿轮跟随						

指令 40 GT_GetAdc

指令原型	short GT_GetAdc (short adc, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取模拟量输入的电压值。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
adc	adc 起始通道号，取值范围：[1, 6]。		
pValue	读取的输入电压值。单位：伏特。		
count	读取的通道数，默认为 1。 1 次最多可以读取 8 路 adc 输入电压值。		
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	GT_GetAdcValue		
指令示例	无。		

指令 41 GT_GetAdcValue

指令原型	short GT_GetAdcValue (short adc, short *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取模拟量输入的数字转换值。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
adc	adc 起始通道号，取值范围：[1, 6]。		
pValue	读取的输入电压数值。单位：bit，取值范围：[-26214, 26214]，对应的电压值为[-10, 10]伏特。		
count	读取的通道数。默认为 1。 1 次最多可以读取 8 路 adc 输入电压值。		

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	GT_GetAdc
指令示例	无。

指令 42 GT_GetAxisBand

指令原型	short GT_GetAxisBand(short axis, long *pBand, long *pTime)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	140
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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	GT_SetAxisBand		
指令示例	例程 11-2 电机到位检测		

指令 43 GT_GetAxisEncAcc

指令原型	short GT_GetAxisEncAcc(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pValue	轴的编码器加速度。单位：pulse/ms ² 。		
count	读取的轴数，默认为 1。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 GT_GetAxisEncPos

指令原型	short GT_GetAxisEncPos(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pValue	轴的编码器位置。单位：pulse。		
count	读取的轴数，默认为 1。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 8-2 Home 回原点		

指令 45 GT_GetAxisEncVel

指令原型	short GT_GetAxisEncVel(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 GT_GetAxisError

指令原型	short GT_GetAxisError(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		

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。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
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 47 GT_GetAxisPrfAcc

指令原型	short GT_GetAxisPrfAcc(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 48 GT_GetAxisPrfPos

指令原型	short GT_GetAxisPrfPos(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pValue	轴的规划位置。单位：pulse。		
count	读取的轴数，默认为 1。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 8-2 Home 回原点

指令 49 GT_GetAxisPrfVel

指令原型	short GT_GetAxisPrfVel(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
	起始轴号。		
axis	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pValue	轴的规划速度。单位：pulse/ms。		
	读取的轴数，默认为 1。		
count	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 50 GT_GetBacklash

指令原型	short GT_GetBacklash (short axis, long *pCompValue, double *pCompChangeValue, long *pCompDir)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	145
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
	查询的轴号。		
axis	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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	GT_SetBacklash		
指令示例	无。		

指令 51 GT_GetCaptureMode

指令原型	short GT_GetCaptureMode(short encoder, short *pMode, short count=1)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取编码器捕获方式。		
指令类型	立即指令，调用后立即生效。	章节页码	105

指令参数	该指令共有 2 个参数，参数的详细信息如下。
encoder	编码器起始轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
pMode	编码器捕获模式。
count	读取的轴数。默认为 1。 1 次最多可以读取 8 个编码器轴。
指令返回值	请参照指令返回值列表。
相关指令	GT_SetCaptureMode
指令示例	无。

指令 52 GT_GetCaptureStatus

指令原型	short GT_GetCaptureStatus (short encoder, short *pStatus, long *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取编码器捕获状态。		
指令类型	立即指令，调用后立即生效。	章节页码	105
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
encoder	编码器起始轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pStatus	读取编码器捕获状态。 为 1 时表示对应轴捕获触发。		
pValue	读取编码器捕获值。 当捕获触发时，编码器捕获值会自动更新。		
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	GT_GetCaptureMode		
指令示例	无。		

指令 53 GT_GetCardNo

指令原型	short GT_GetCardNo(short *pIndex)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取当前运动控制器卡号。		
指令类型	立即指令，调用后立即生效。	章节页码	138
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
pIndex	读取的当前运动控制器的卡号。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetCardNo		
指令示例	无。		

指令 54 GT_GetClock

指令原型	short GT_GetClock(unsigned long *pClock, unsigned long *pLoop=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	139
指令参数	该指令共有 2 个参数，参数的详细信息如下：		
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 55 GT_GetClockHighPrecision

指令原型	short GT_GetClockHighPrecision(unsigned long *pClock)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	139
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
pClock	读取的运动控制器的时钟，单位：125us。		
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 56 GT_GetControlFilter

指令原型	short GT_GetControlFilter(short control, short *pIndex)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取当前 PID 索引。		
指令类型	立即指令，调用后立即生效。	章节页码	145
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
control	伺服控制器编号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pIndex	读取的伺服控制参数的索引号。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetControlFilter		
指令示例	无。		

指令 57 GT_GetCrdPos

指令原型	short GT_GetCrdPos (short crd, double *pPos)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 2 个参数，参数的详细信息如下。		

crd	坐标系号。正整数，取值范围：[1, 2]。
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	无。

指令 58 GT_GetCrdPrm

指令原型	short GT_GetCrdPrm (short crd, TCrdPrm *pCrdPrm)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pCrdPrm	读取坐标系的相关参数 结构体的成员含义参照 GT_SetCrdPrm 指令说明。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetCrdPrm		
指令示例	无。		

指令 59 GT_GetCrdStopDec

指令原型	short GT_GetCrdStopDec (short crd, double *pDecSmoothStop, double *pDecAbruptStop)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
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 60 GT_GetCrdVel

指令原型	short GT_GetCrdVel(short crd, double *pSynVel)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
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 61 GT_GetDac

指令原型	short GT_GetDac(short dac, short *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取 dac 输出电压。		
指令类型	立即指令，调用后立即生效。	章节页码	103
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
dac	dac 起始轴号。		
pValue	输出电压。		
count	读取的通道数。默认为 1。 1 次最多可以读取 8 个 dac 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	GT_SetDac		
指令示例	无。		

指令 62 GT_GetDi

指令原型	short GT_GetDi(short diType, long *pValue)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取数字 IO 输入状态。		
指令类型	立即指令，调用后立即生效。	章节页码	102
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE(该宏定义为 0): 正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1): 负限位。 MC_ALARM(该宏定义为 2): 驱动报警。 MC_HOME(该宏定义为 3): 原点开关。 MC_GPI(该宏定义为 4): 通用输入。 MC_ARRIVE(该宏定义为 5): 电机到位信号。 MC_MPG(该宏定义为 6): 手轮 MPG 轴选和倍率信号 (5V 电平输入)。		
pValue	数字 IO 输入状态，按位指示 IO 输入电平(根据配置工具 di 的 reverse 值不同而不同)。 当 reverse=0 时，1 表示高电平，0 表示低电平。 当 reverse=1 时，1 表示低电平，0 表示高电平。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 7-1 访问数字 IO		

指令 63 GT_GetDiRaw

指令原型	short GT_GetDiRaw(short diType, long *pValue)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取数字 IO 输入状态的原始值。		
指令类型	立即指令，调用后立即生效。	章节页码	102

指令参数	该指令共有 2 个参数，参数的详细信息如下。
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。
pValue	数字 IO 输入状态的原始值，按位指示 IO 输入电平。 1 表示高电平，0 表示低电平。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 64 GT_GetDiReverseCount

指令原型	short GT_GetDiReverseCount (short diType, short diIndex, unsigned long *pReverseCount, short count)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	取数字量输入信号的变化次数。		
指令类型	立即指令，调用后立即生效。	章节页码	102
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。		
diIndex	数字量输入的索引。 取值范围： diType= MC_LIMIT_POSITIVE 时：[1, 8]。 diType= MC_LIMIT_NEGATIVE 时：[1, 8]。 diType= MC_ALARM 时：[1, 8]。 diType= MC_HOME 时：[1, 8]。 diType= MC_GPI 时：[1, 16]。 diType= MC_ARRIVE 时：[1, 4]。		
pReverseCount	读取的数字量输入的变化次数。		
count	读取变化次数的数字量输入的个数，默认为 1。 1 次最多可以读取 4 个数字量输入的变化次数。		
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 65 GT_GetDo

指令原型	short GT_GetDo (short doType, long *pValue)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取数字 IO 输出状态		
指令类型	立即指令，调用后立即生效。	章节页码	102
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
doType	指定数字 IO 类型。 MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。		
pValue	数字 IO 输出状态，按位指示 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetDo		
指令示例	无。		

指令 66 GT_GetEncPos

指令原型	short GT_GetEncPos (short encoder, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	103
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
encoder	编码器起始轴号。正整数。 对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]，第 11 轴为手轮接口专用编码器。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]，第 11 轴为手轮接口专用编码器。		
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	GT_SetEncPos		
指令示例	例程 7-2 访问编码器		

指令 67 GT_GetEncVel

指令原型	short GT_GetEncVel (short encoder, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	103
指令参数	该指令共有 4 个参数，参数的详细信息如下：		
encoder	编码器起始轴号。正整数。 对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]，第 11 轴为手轮接口专用编码器。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]，第 11 轴为手轮接口专用编码器。		
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	无。

指令 68 GT_GetFollowEvent

指令原型	short GT_GetFollowEvent (short profile, short *pEvent, short *pMasterDir, long *pPos)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取 Follow 运动模式启动跟随条件。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pEvent	读取启动跟随条件。 FOLLOW_EVENT_START（该宏定义为 1）表示调用 GT_FollowStart 以后立即启动。 FOLLOW_EVENT_PASS（该宏定义为 2）表示主轴穿越设定位置以后启动跟随。		
pMasterDir	读取主轴运动方向。 1 主轴正向运动，-1 主轴负向运动。		
pPos	读取穿越位置，单位：pulse。 当 event 为 FOLLOW_EVENT_PASS 时有效。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GT_PrFFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetFollowEvent		
指令示例	无。		

指令 69 GT_GetFollowLoop

指令原型	short GT_GetFollowLoop(short profile, long *pLoop)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取 Follow 运动模式循环已经执行完成的次数。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pLoop	读取 Follow 模式循环已经执行完成的次数。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GT_PrFFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetFollowLoop		
指令示例	无。		

指令 70 GT_GetFollowMaster

指令原型	short GT_GetFollowMaster (short profile, short *pMasterIndex, short *pMasterType, short *pMasterItem)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取 Follow 运动模式跟随主轴。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
	读取主轴索引。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。 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。		
pMasterIndex	读取主轴类型。 FOLLOW_MASTER_PROFILE（该宏定义为 2）表示跟随规划轴(profile)的输出值，默认 为此类型。 FOLLOW_MASTER_ENCODER（该宏定义为 1）表示跟随编码器(encoder)的输出值。 FOLLOW_MASTER_AXIS（该宏定义为 3）表示跟随轴(axis)的输出值。		
	合成轴类型，当 masterType= FOLLOW_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。		
pMasterType	合成轴类型，当 masterType= FOLLOW_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。		
pMasterItem	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GT_PrFFollow 将 当前轴设置为 Follow 模式。 其他返回值：请参照指令返回值列表。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GT_PrFFollow 将 当前轴设置为 Follow 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetFollowMaster		
指令示例	无。		

指令 71 GT_GetFollowMemory

指令原型	short GT_GetFollowMemory (short profile, short *pMemory)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取 Follow 运动模式的缓存区大小。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
	读取 Follow 运动缓存区大小标志。 0：每个 Follow 运动缓存区有 16 段空间。 1：每个 Follow 运动缓存区有 512 段空间。		
pMemory	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GT_PrFFollow 将 当前轴设置为 Follow 模式。 其他返回值：请参照指令返回值列表。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GT_PrFFollow 将 当前轴设置为 Follow 模式。 其他返回值：请参照指令返回值列表。		

相关指令	GT_SetFollowMemory
指令示例	无。

指令 72 GT_GetFunId

指令原型	short GT_GetFunId (char *pFunName, short *pFunId)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	120
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
pFunName	运动程序函数名称。		
pFunId	根据运动程序函数名称查询函数标识。		
指令返回值	<p>若返回值为 1，2007 或者 2008： 请检查重新检查 GT_Download 是否调用成功。</p> <p>若失败，请根据 GT_Download 返回值提示操作，直至成功。</p> <p>其他返回值：请参照指令返回值列表。</p>		
相关指令	无。		
指令示例	例程 10-1 单线程累加求和		

指令 73 GT_GetGearMaster

指令原型	short GT_GetGearMaster (short profile, short *pMasterIndex, short *pMasterType, short *pMasterItem)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	48
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	<p>规划轴号。</p> <p>正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。</p>		
pMasterIndex	<p>读取主轴索引。</p> <p>正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。</p>		
pMasterType	<p>读取主轴类型。</p> <p>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。</p>		
pMasterItem	<p>读取输出位置类型。当 masterType=GEAR_MASTER_AXIS 时起作用。</p> <p>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。</p>		
指令返回值	<p>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 GT_Prfgear 将当前轴设置为电子齿轮模式。</p> <p>其他返回值：请参照指令返回值列表。</p>		
相关指令	GT_SetGearMaster		
指令示例	无。		

指令 74 GT_GetGearRatio

指令原型	short GT_GetGearRatio(short profile, long *pMasterEven, long *pSlaveEven, long *pMasterSlope)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	48
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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	读取主轴离合器区位移。单位：pulse。		
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 GT_PrflGear 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	GT_SetGearRatio		
指令示例	无。		

指令 75 GT_GetJogPrm

指令原型	short GT_GetJogPrm(short profile, TJogPrm *pPrm)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	36
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 GT_SetJogPrm 指令说明。		
指令返回值	若返回值为 1： 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 请检查当前轴是否为 Jog 模式，若不是，请先调用 GT_PrflJog 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	GT_SetJogPrm		
指令示例	例程 6-2 Jog 运动		

指令 76 GT_GetMtrBias

指令原型	short GT_GetMtrBias(short dac, short *pBias)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取模拟量输出通道的零漂电压补偿值。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
	模拟量输出通道号。		
dac	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。		

pBias 指令返回值 相关指令 指令示例	对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。		
	读取的零漂补偿值。		
	请参照指令返回值列表。		
	GT_SetMtrBias		
	无。		

指令 77 GT_GetMtrLmt

指令原型	short GT_GetMtrLmt(short dac, short *pLimit)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取模拟量输出通道的输出电压饱和和极限值。		
指令类型	立即指令, 调用后立即生效。	章节页码	29
指令参数	该指令共有 2 个参数, 参数的详细信息如下。		
dac	模拟量输出通道号。		
	正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。		
pLimit	读取的输出电压饱和和极限值。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetMtrLmt		
指令示例	无。		

指令 78 GT_GetPid

指令原型	short GT_GetPid(short control, short index, TPid *pPid)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取 PID 参数。		
指令类型	立即指令, 调用后立即生效。	章节页码	145
指令参数	该指令共有 3 个参数, 参数的详细信息如下。		
control	伺服控制器编号。		
	正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。		
index	伺服控制参数的索引号。取值范围: [1, 3]。		
pPid	读取 PID 参数。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetPid		
指令示例	例程 11-2 电机到位检测		

指令 79 GT_GetPos

指令原型	short GT_GetPos(short profile, long *pPos)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	33
指令参数	该指令共有 2 个参数, 参数的详细信息如下:		
profile	规划轴号。		
	正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。		

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 GT_PrTrp 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	GT_SetPos
指令示例	无。

指令 80 GT_GetPosErr

指令原型	short GT_GetPosErr(short control, long *pError)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取跟随误差极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
control	伺服控制器编号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pError	读取的跟随误差极限值。单位：pulse。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetPosErr		
指令示例	无。		

指令 81 GT_GetPrfAcc

指令原型	short GT_GetPrfAcc(short profile, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	起始规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pValue	规划加速度。单位：pulse/ms ² 。		
count	读取的轴数，默认为 1。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 82 GT_GetPrfMode

指令原型	short GT_GetPrfMode(short profile, long *pValue, short count=1, unsigned long *pClock=NULL)
适用板卡	GUS-400-TG02,GUS-600-TG02

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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	起始规划轴号。		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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；		
count	5：插补模式；		
	6：Pvt 模式。		
pClock	读取的轴数，默认为 1。		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
指令返回值	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
相关指令	请参照指令返回值列表。		
指令示例	无。		

指令 83 GT_GetPrfPos

指令原型	short GT_GetPrfPos(short profile, double *pValue, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	起始规划轴号。		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pValue	规划位置。单位：pulse。		
count	读取的轴数，默认为 1。		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 点位运动		

指令 84 GT_GetPrfVel

指令原型	short GT_GetPrfVel(short profile, double *pValue, short count=1, unsigned long *pClock=NULL)		
------	--	--	--

适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
	起始规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pValue	规划速度。单位：pulse/ms。		
	读取的轴数，默认为 1。		
count	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 6-2 Jog 运动		

指令 85 GT_GetPtLoop

指令原型	short GT_GetPtLoop(short profile, long *pLoop)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	查询 PT 运动模式循环执行的次数。动态模式下该指令无效。		
指令类型	立即指令，调用后立即生效。	章节页码	40
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pLoop	查询 PT 模式循环已经执行完成的次数。动态模式下该参数无效。		
指令返回值	若返回值为 1：请检查当前轴是否为 PT 模式，若不是，请先调用 GT_PrFpt 将当前轴设置为 PT 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetPtLoop		
指令示例	无。		

指令 86 GT_GetPtMemory

指令原型	short GT_GetPtMemory(short profile, short *pMemory)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取 PT 运动模式的缓存区大小。		
指令类型	立即指令，调用后立即生效。	章节页码	40
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pMemory	读取 PT 运动缓存区大小标志。		
指令返回值	若返回值为 1：请检查当前轴是否为 PT 模式，若不是，请先调用 GT_PrFpt 将当前轴设置为 PT 模式。		

其他返回值：请参照指令返回值列表。	
相关指令 指令示例	GT_SetPtMemory 无。

指令 87 GT_GetPvtLoop

指令原型	short GT_GetPvtLoop(short profile, long *pLoopCount, long *pLoop)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	查询 PVT 运动模式循环次数。		
指令类型	立即指令，调用后立即生效。	章节页码	79
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pLoopCount	查询已经循环的次数。		
pLoop	查询循环执行的总次数。		
指令返回值	若返回值为 1：请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照指令返回值列表。		
相关指令 指令示例	GT_SetPvtLoop 无。		

指令 88 GT_GetRemainderSegNum

指令原型	short GT_GetRemainderSegNum(short crd, long *pSegment, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
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 指令示例	无。 无。		

指令 89 GT_GetSoftLimit

指令原型	short GT_GetSoftLimit (short axis, long *pPositive, long *pNegative)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	116
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		

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	GT_SetSoftLimit
指令示例	无。

指令 90 GT_GetStopDec

指令原型	short GT_GetStopDec(short profile, double *pDecSmoothStop, double *pDecAbruptStop)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	29
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划器的编号		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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	GT_SetStopDec		
指令示例	无。		

指令 91 GT_GetSts

指令原型	short GT_GetSts(short axis, long *pSts, short count=1, unsigned long *pClock=NULL)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	31
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pSts	32 位轴状态字，详细定义参见表 5-2。		
count	读取的轴数，默认为 1。		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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	GT_ClrSts		
指令示例	例程 6-1 点位运动		

指令 92 GT_GetThreadSts

指令原型	short GT_GetThreadSts(short thread, TThreadSts *pThreadSts)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取线程的状态。		
指令类型	立即指令，调用后立即生效。	章节页码	120

指令参数	该指令共有 2 个参数，参数的详细信息如下。
thread	线程编号。取值范围：[0, 31]。
pThreadSts	读取线程状态 <pre>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;</pre>
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 单线程累加求和

指令 93 GT_GetTrapPrm

指令原型	short GT_GetTrapPrm(short profile, TTrapPrm *pPrm)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	33
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 GT_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 GT_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	GT_SetTrapPrm		
指令示例	例程 6-1 点位运动		

指令 94 GT_GetUserSegNum

指令原型	short GT_GetUserSegNum (short crd, long *pSegment, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
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	GT_SetUserSegNum		
指令示例	无。		

指令 95 GT_GetVarId

指令原型	short GT_GetVarId(char *pFunName, char *pVarName, TVarInfo *pVarInfo)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	120
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 GT_Download 是否调用成功。 若失败, 请根据 GT_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 单线程累加求和		

指令 96 GT_GetVarValue

指令原型	short GT_GetVarValue(short page, TVarInfo *pVarInfo, double *pValue, short count=1)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	120
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
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, 6]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetVarValue		
指令示例	例程 10-1 单线程累加求和		

指令 97 GT_GetVel

指令原型	short GT_GetVel(short profile, double *pVel)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	33,36
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围: [1, 6]。		
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 GT_PrflTrap 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	GT_SetVel
指令示例	无。

指令 98 GT_GetVersion

指令原型	short GT_GetVersion(char **pVersion)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	138
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
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 读取固件版本		

指令 99 GT_GpiSns

指令原型	short GT_GpiSns(unsigned short sense)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置运动控制器数字量输入的电平逻辑。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 1 个参数，参数的详细信息如下： 按位表示各数量输入的电平逻辑，从 bit0~bit15，分别对应数字量输入 1 到 16。影响 GT_GetDi 指令读取电平的结果。 0：输入电平不取反，通过 GT_GetDi 指令读取到 0 表示输入低电平，通过 GT_GetDi 指令读取到 1 表示输入高电平； 1：输入电平取反，通过 GT_GetDi 指令读取到 0 表示输入高电平，通过 GT_GetDi 指令读取到 1 表示输入低电平；		
sense			
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 100 GT_Home

指令原型	short GT_Home (short axis, long pos, double vel, double acc, long offset)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	启动自动回原点功能。不能在运动程序运行中调用。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 5 个参数，参数的详细信息如下。 需要进行自动回原点操作的轴号。 axis 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。 pos 搜索距离。以当前位置为起点，搜索距离为正时向正方向搜索，搜索距离为负时向负方		

	向搜索。单位: pulse。
vel	搜索速度。单位: pulse/ms。
acc	搜索加速度。单位: pulse/ms ² 。
offset	原点偏移量。当原点信号触发时, 将当前轴目标位置自动更新为“原点位置+原点偏移”。如果原点偏移量为 0, 当原点信号触发时, 首先平滑停止减速到 0, 然后返回原点。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	例程 11-3 自动回原点

指令 101 GT_HomeInit

指令原型	short GT_HomeInit()		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	初始化自动回原点功能。不能在运动程序运行中调用。		
指令类型	立即指令, 调用后立即生效。	章节页码	146
指令参数	该指令无输入参数。		
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 11-3 自动回原点		

指令 102 GT_HomeStop

指令原型	short GT_HomeStop(short axis, long pos, double vel, double acc)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	启动原点停止功能。不能在运动程序运行中调用。		
指令类型	立即指令, 调用后立即生效。	章节页码	146
指令参数	该指令共有 4 个参数, 参数的详细信息如下。		
axis	需要进行原点急停操作的轴号。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。		
pos	搜索距离。以当前位置为起点, 搜索距离为正时向正方向搜索, 搜索距离为负时向负方向搜索, 单位: pulse。		
vel	搜索速度。单位: pulse/ms。		
acc	搜索加速度。单位: pulse/ms ² 。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 103 GT_HomeSts

指令原型	short GT_HomeSts(short axis, unsigned short *pStatus)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	查询自动回原点的运行状态。不能在运动程序运行中调用。		
指令类型	立即指令, 调用后立即生效。	章节页码	146
指令参数	该指令共有 2 个参数, 参数的详细信息如下。		
axis	需要查询自动回原点状态的轴号。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。		

	对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。
pStatus	查询到的状态值。 0: 自动回原点操作正在执行。 1: 自动回原点操作成功执行完毕。 2: 执行完毕, 未触发。
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 自动回原点

指令 104 GT_Index

指令原型	short GT_Index(short axis, long pos, long offset)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置自动回原点功能为 home+index 模式。不能在运动程序运行中调用。		
指令类型	立即指令, 调用后立即生效。	章节页码	146
指令参数	该指令共有 3 个参数, 参数的详细信息如下。		
axis	需要进行原点急停操作的轴号。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。		
pos	搜索距离。以当前位置为起点, 搜索距离为正时向正方向搜索, 搜索距离为负时向负方向搜索。单位: pulse。		
offset	原点偏移量, 单位: pulse。		
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 自动回原点		

指令 105 GT_InitLookAhead

指令原型	short GT_InitLookAhead (short crd, short fifo, double T, double accMax, short n, TCrdData *pLookAheadBuf)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 6 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		
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 6-11 前瞻预处理		

指令 106 GT_LmtSns

指令原型	short GT_LmtSns(unsigned short sense)
------	---------------------------------------

适用板卡	GUS-400-TG02,GUS-600-TG02																																							
指令说明	设置运动控制器各轴限位开关触发电平。																																							
指令类型	立即指令，调用后立即生效。 章节页码 29																																							
指令参数	该指令共有 1 个参数，当该参数的某个状态位为 0 时，表示将对应的限位开关设置为高电平触发，当某个状态位为 1 时，表示将对应的限位开关设置为低电平触发。参数的详细信息如下。																																							
sense	按位标识轴的限位的触发电平状态。下表中“+”代表正限位，“-”代表负限位。 对于 GUS-400-TG02 系列的控制卡，状态位与限位开关对应关系如下：																																							
	<table border="1"> <tr> <td>状态位</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>限 位</td> <td colspan="2">轴 4</td> <td colspan="2">轴 3</td> <td colspan="2">轴 2</td> <td colspan="2">轴 1</td> </tr> <tr> <td>开 关</td> <td>-</td> <td>+</td> <td>-</td> <td>+</td> <td>-</td> <td>+</td> <td>-</td> <td>+</td> </tr> </table>	状态位	7	6	5	4	3	2	1	0	限 位	轴 4		轴 3		轴 2		轴 1		开 关	-	+	-	+	-	+	-	+												
	状态位	7	6	5	4	3	2	1	0																															
限 位	轴 4		轴 3		轴 2		轴 1																																	
开 关	-	+	-	+	-	+	-	+																																
对于 GUS-600-TG02 系列的控制卡，状态位与限位开关对应关系如下：																																								
	<table border="1"> <tr> <td>状态位</td> <td>11</td> <td>10</td> <td>9</td> <td>8</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>限 位</td> <td colspan="2">轴 6</td> <td colspan="2">轴 5</td> <td colspan="2">轴 4</td> <td colspan="2">轴 3</td> <td colspan="2">轴 2</td> <td colspan="2">轴 1</td> </tr> <tr> <td>开 关</td> <td>-</td> <td>+</td> <td>-</td> <td>+</td> <td>-</td> <td>+</td> <td>-</td> <td>+</td> <td>-</td> <td>+</td> <td>-</td> <td>+</td> </tr> </table>	状态位	11	10	9	8	7	6	5	4	3	2	1	0	限 位	轴 6		轴 5		轴 4		轴 3		轴 2		轴 1		开 关	-	+	-	+	-	+	-	+	-	+	-	+
状态位	11	10	9	8	7	6	5	4	3	2	1	0																												
限 位	轴 6		轴 5		轴 4		轴 3		轴 2		轴 1																													
开 关	-	+	-	+	-	+	-	+	-	+	-	+																												
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 107 GT_LmtsOff

指令原型	short GT_LmtsOff(short axis, short limitType=-1)
适用板卡	GUS-400-TG02,GUS-600-TG02
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 29
指令参数	该指令共有 2 个参数，参数的详细信息如下。
axis	控制轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
	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：将该轴的正限位和负限位都无效，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	GT_LmtsOn
指令示例	无。

指令 108 GT_LmtsOn

指令原型	short GT_LmtsOn(short axis, short limitType=-1)
适用板卡	GUS-400-TG02,GUS-600-TG02
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 29
指令参数	该指令共有 2 个参数，参数的详细信息如下。
axis	控制轴号。

	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
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	GT_LmtsOff
指令示例	无。

指令 109 GT_LnXY

指令原型	short GT_LnXY (short crd, long x, long y, double synVel, double synAcc, double velEnd=0, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 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 6-9 直线插补		

指令 110 SGT_LnXYG0

指令原型	short GT_LnXYG0 (short crd, long x, long y, double synVel, double synAcc, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 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 GT_LnXYZ

指令原型	short GT_LnXYZ (short crd, long x, long y, long z, double synVel, double synAcc, double velEnd=0, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	三维直线插补。		
指令类型	缓存区指令。	章节页码	63
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 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 112 GT_LnXYZA

指令原型	short GT_LnXYZA (short crd, long x, long y, long z, long a, double synVel, double synAcc, double velEnd=0, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	四维直线插补。		
指令类型	缓存区指令。	章节页码	63
指令参数	该指令共有 9 个参数，参数的详细信息如下。		

crd	坐标系号。正整数，取值范围：[1, 2]。
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
a	插补段 a 轴终点坐标值。取值范围：[-1073741823, 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 113 GT_LnXYZAG0

指令原型	short GT_LnXYZAG0 (short crd, long x, long y, long z, long a, double synVel, double synAcc, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
a	插补段 a 轴终点坐标值。取值范围：[-1073741823, 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 114 GT_LnXYZG0

指令原型	short GT_LnXYZG0 (short crd, long x, long y, long z, double synVel, double synAcc, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 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： <ol style="list-style-type: none"> (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 115 GT_LoadConfig

指令原型	short GT_LoadConfig(char *pFile)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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 GT_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	27
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
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 MCT2008 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 6-1 点位运动		

指令 116 GT_Open

指令原型	short GT_Open(short channel=0, short param=1)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	打开运动控制器。		
指令类型	立即指令，调用后立即生效。	章节页码	138
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。默认值为：0。 0：正常打开运动控制器。 1：内部调试方式，用户不能使用。		

param	当 channel=1 时，该参数有效。
指令返回值	请参照指令返回值列表。
相关指令	GT_Close
指令示例	例程 6-1 点位运动

指令 117 GT_PauseThread

指令原型	short GT_PauseThread(short thread)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	120
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
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	GT_RunThread ; GT_StopThread		
指令示例	无。		

指令 118 GT_PrFFollow

指令原型	short GT_PrFFollow (short profile, short dir)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置指定轴为 Follow 运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	33,51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
dir	设置跟随方式。 0 表示双向跟随，1 表示正向跟随，-1 表示负向跟随。		
指令返回值	若返回值为 1： 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 (1) 当前已经是 Follow 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 6-6 Follow 单 FIFO，例程 6-7 Follow 双 FIFO		

指令 119 GT_PrFGear

指令原型	short GT_PrFGear (short profile, short dir)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	33,48
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。		

dir	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
	设置跟随方式。 0 表示双向跟随，1 表示正向跟随，-1 表示负向跟随。
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GT_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 6-5 电子齿轮跟随

指令 120 GT_PrJog

指令原型	short GT_PrJog(short profile)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	33,36
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
profile	规划轴号。		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_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 6-2 Jog 运动		

指令 121 GT_PrPt

指令原型	short GT_PrPt(short profile, short mode=PT_MODE_STATIC)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置指定轴为 PT 运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	33,40
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
mode	指定 FIFO 使用模式。		
	PT_MODE_STATIC（该宏定义为 0）静态模式。默认为该模式。 PT_MODE_DYNAMIC（该宏定义为 1）动态模式。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_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 6-3 PT 静态 FIFO		

指令 122 GT_PrPvt

指令原型	short GT_PrPvt(short profile)
------	-------------------------------

适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置指定轴为 PVT 运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	33,79
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_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 6-14 PVT 描述方式		

指令 123 GT_PrfrTrap

指令原型	short GT_PrfrTrap(short profile)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	33
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_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 6-1 点位运动		

指令 124 GT_ProfileScale

指令原型	short GT_ProfileScale(short axis, short alpha, short beta)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置控制轴的规划器当量变换值。注意：alpha 的绝对值要大于 beta 的绝对值。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
	控制轴号。		
axis	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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 GT_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	GT_EncScale		
指令示例	无。		

指令 125 GT_PtClear

指令原型	short GT_PtClear(short profile, short fifo)		
------	---	--	--

适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	清除 PT 运动模式指定 FIFO 中的数据。 运动状态下该指令无效。 动态模式下该指令无效。		
指令类型	立即指令，调用后立即生效。	章节页码	40
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
fifo	指定所要查询的 FIFO，取值范围：0、1 两个值。默认为 0。 动态模式下该参数无效。		
指令返回值	若返回值为 1： (1) 静态模式下，检查要清除的 FIFO 是否正在使用，在运动； (2) 动态模式下，不能在运动时清 FIFO； 请检查当前轴是否为 PT 模式，若不是，请先调用 <code>GT_PrPpt</code> 将当前轴设置为 PT 模式。 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 6-3 PT 静态 FIFO		

指令 126 GT_PtData

指令原型	short GT_PtData(short profile, double pos, long time, short type, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	向 PT 运动模式指定 FIFO 增加数据。		
指令类型	立即指令，调用后立即生效。	章节页码	40
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pos	段末位置。单位：pulse。		
time	段末时间。单位：ms。		
type	数据段类型。 PT_SEGMENT_NORMAL（该宏定义为 0）普通段。默认为该类型。 PT_SEGMENT_EVEN（该宏定义为 1）匀加速段。 PT_SEGMENT_STOP（该宏定义为 2）减速到 0 段。		
fifo	指定所要查询的 FIFO，取值范围：0、1 两个值。默认为 0。 动态模式下该参数无效。		
指令返回值	若返回值为 1： (1) 请检查 Space 是否小于 0，若是，则等待 Space 大于 0； 请检查当前轴是否为 PT 模式，若不是，请先调用 <code>GT_PrPpt</code> 将当前轴设置为 PT 模式。 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 6-3 PT 静态 FIFO		

指令 127 GT_PtSpace

指令原型	short GT_PtSpace(short profile, short *pSpace, short fifo=0)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	查询 PT 运动模式指定 FIFO 的剩余空间。		
指令类型	立即指令，调用后立即生效。	章节页码	40
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pSpace	读取 PT 指定 FIFO 的剩余空间。		
fifo	指定所要查询的 FIFO，取值范围：0、1 两个值。默认为 0。 动态模式下该参数无效。		
指令返回值	若返回值为 1：请检查当前轴是否为 PT 模式，若不是，请先调用 GT_PrPpt 将当前轴设置为 PT 模式。 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 6-3 PT 静态 FIFO		

指令 128 GT_PtStart

指令原型	short GT_PtStart(long mask, long option)																														
适用板卡	GUS-400-TG02,GUS-600-TG02																														
指令说明	启动 PT 运动。																														
指令类型	立即指令，调用后立即生效。	章节页码	40																												
指令参数	该指令共有 2 个参数，参数的详细信息如下。																														
mask	按位指示需要启动 PT 运动的轴号。当 bit 位为 1 时表示启动对应的轴。 对于 GUS-400-TG02 系列的控制卡： <table border="1" style="margin-left: 20px;"> <tr> <td>Bit</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>对应轴</td> <td>4 轴</td> <td>3 轴</td> <td>2 轴</td> <td>1 轴</td> </tr> </table> 对于 GUS-600-TG02 系列的控制卡： <table border="1" style="margin-left: 20px;"> <tr> <td>Bit</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>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>							Bit	3	2	1	0	对应轴	4 轴	3 轴	2 轴	1 轴	Bit	5	4	3	2	1	0	对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
Bit	3	2	1	0																											
对应轴	4 轴	3 轴	2 轴	1 轴																											
Bit	5	4	3	2	1	0																									
对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴																									
option	按位指示所使用的 FIFO，默认为 0。 对于 GUS-400-TG02 系列的控制卡： <table border="1" style="margin-left: 20px;"> <tr> <td>Bit</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>对应轴</td> <td>4 轴</td> <td>3 轴</td> <td>2 轴</td> <td>1 轴</td> </tr> </table> 对于 GUS-600-TG02 系列的控制卡： <table border="1" style="margin-left: 20px;"> <tr> <td>Bit</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>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 bit 位为 0 时表示对应的轴使用 FIFO1（即 fifo=0） 当 bit 位为 1 时表示对应的轴使用 FIFO2（即 fifo=1） 动态模式下该参数无效。							Bit	3	2	1	0	对应轴	4 轴	3 轴	2 轴	1 轴	Bit	5	4	3	2	1	0	对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
Bit	3	2	1	0																											
对应轴	4 轴	3 轴	2 轴	1 轴																											
Bit	5	4	3	2	1	0																									
对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴																									
指令返回值	若返回值为 1：请检查相应轴的 FIFO 是否有数据，若没有，请先压入数据； 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 6-3 PT 静态 FIFO																														

指令 129 GT_PvtContinuousCalculate

指令原型	short GT_PvtContinuousCalculate(long count, double *pPos, double *pVel, double *pPercent, double *pVelMax, double *pAcc, double *pDec, double *pTime)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	计算 PVT 运动模式 Continuous 描述方式下各数据点的时间。		
指令类型	立即指令，调用后立即生效。	章节页码	79
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
count	数据点个数。该指令用来计算各数据点时间，不会将数据点下载到运动控制器。		
pPos	数据点位置数组。单位：pulse，数组长度为 count。		
pVel	数据点速度数组。单位：pulse/ms，数组长度为 count。		
pPercent	数据点百分比数组。数组长度为 count。 百分比的取值范围：[0, 100]。		
pVelMax	数据点最大速度数组。单位：pulse/ms，数组长度为 count。		
pAcc	数据点加速度数组。单位是“pulse/ms ² ”，数组长度为 count。		
pDec	数据点减速度数组。单位是“pulse/ms ² ”，数组长度为 count。		
pTime	返回各数据点的时间。单位：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	例程 6-17 Continuous 描述方式		

指令 130 GT_PvtPercentCalculate

指令原型	short GT_PvtPercentCalculate (long count, double *pTime, double *pPos, double *pPercent, double velBegin, double *pVel)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	计算 PVT 运动模式 Percent 描述方式下各数据点的速度。		
指令类型	立即指令，调用后立即生效。	章节页码	79
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
count	数据点个数。该指令用来计算各数据点的速度，不会将数据点下载到运动控制器。		
pTime	数据点时间数组。单位：ms，数组长度为 count。		
pPos	数据点位置数组。单位：pulse，数组长度为 count。		
pPercent	数据点百分比数组。数组长度为 count。 百分比的取值范围：[0, 100]。		
velBegin	起点速度。单位：pulse/ms。		
pVel	返回各数据点的速度。单位：pulse/ms。		
指令返回值	若返回值为 1： 请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrflPvt 将当前轴设置为 PVT 模式。 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 (1) 请检查传递的数据点是否大于 1024 个。 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 GT_PvtStart

指令原型	short GT_PvtStart(long mask)						
适用板卡	GUS-400-TG02,GUS-600-TG02						
指令说明	启动 PVT 运动。						
指令类型	立即指令，调用后立即生效。	章节页码	79				
指令参数	该指令共有 1 个参数，参数的详细信息如下。						
mask	按位指示需要启动的轴号。当 bit 位为 1 时表示启动对应的轴。						
	对于 GUS-400-TG02 系列的控制卡：						
	Bit	3	2	1	0		
	对应轴	4 轴	3 轴	2 轴	1 轴		
mask	对于 GUS-600-TG02 系列的控制卡：						
	Bit	5	4	3	2	1	0
	对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
	在启动运动之前，可以调用 GT_PvtTableSelect 选择数据表。如果没有选择数据表，默认使用数据表 1。如果数据表为空，则启动失败。						
指令返回值	若返回值为 1：						
	(1) 目标数据表不应为空。请检查目标数据表是否空。						
指令返回值	请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrftPvt 将当前轴设置为 PVT 模式。						
	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 6-14 PVT 描述方式						

指令 132 GT_PvtStatus

指令原型	short GT_PvtStatus(short profile, short *pTableId, double *pTime, short count)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	读取 PVT 运动状态。		
指令类型	立即指令，调用后立即生效。	章节页码	79
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	规划轴号。		
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pTableId	当前正在使用的数据表 ID。		
pTime	当前轴已经运动的时间，单位：ms。		
count	读取的轴数。		
指令返回值	若返回值为 1：请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrftPvt 将当前轴设置为 PVT 模式。		
	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 6-14 PVT 描述方式		

指令 133 GT_PvtTable

指令原型	short GT_PvtTable (short tableId, long count, double *pTime, double *pPos, double *pVel)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	向 PVT 运动模式指定数据表传送数据，采用 PVT 描述方式。		

指令类型	立即指令，调用后立即生效。	章节页码	79
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
tableId	指定数据表。取值范围：[1, 32]。		
count	数据点个数。每个数据表具有 1024 个存储空间。 每个数据点占用 1 个存储空间		
pTime	数据点时间数组，单位：ms，数组长度为 count。		
pPos	数据点位置数组，单位：pulse，数组长度为 count。		
pVel	数据点速度数组，单位：pulse/ms，数组长度为 count。		
指令返回值	<p>若返回值为 1： 请检查当前轴是否为 PVT 模式，若不是，请先调用 <code>GT_PrflPvt</code> 将当前轴设置为 PVT 模式。</p> <p>若当前轴在规划运动，请调用 <code>GT_Stop</code> 停止运动再调用该指令。</p> <p>(1) 请检查传递的数据点是否大于 1024 个。</p> <p>其他返回值：请参照指令返回值列表。</p>		
相关指令	无。		
指令示例	例程 6-14 PVT 描述方式		

指令 134 GT_PvtTableComplete

指令原型	short GT_PvtTableComplete (short tableId, long count, double *pTime, double *pPos, double *pA, double *pB, double *pC, double velBegin, double velEnd)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	向 PVT 运动模式指定数据表传送数据，采用 Complete 描述方式。		
指令类型	立即指令，调用后立即生效。	章节页码	79
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
tableId	指定数据表。取值范围：[1, 32]。		
count	数据点个数。每个数据表具有 1024 个存储空间。 每个数据点占用 1 个存储空间。		
pTime	数据点时间数组。单位：ms，数组长度为 count。		
pPos	数据点位置数组。单位：pulse，数组长度为 count。		
pA、pB、pC	工作数组，内部使用，数组长度为 count。 该数组用户不必赋值。		
velBegin	起点速度。单位：pulse/ms。		
velEnd	终点速度。单位：pulse/ms。		
指令返回值	<p>若返回值为 1： 请检查当前轴是否为 PVT 模式，若不是，请先调用 <code>GT_PrflPvt</code> 将当前轴设置为 PVT 模式。</p> <p>若当前轴在规划运动，请调用 <code>GT_Stop</code> 停止运动再调用该指令。</p> <p>(1) 请检查传递的数据点是否大于 1024 个。</p> <p>其他返回值：请参照指令返回值列表。</p>		
相关指令	无。		
指令示例	例程 6-15 Complete 描述方式		

指令 135 GT_PvtTableContinuous

指令原型	short GT_PvtTableContinuous (short tableId, long count, double *pPos, double *pVel, double
------	--

适用板卡	*pPercent, double *pVelMax, double *pAcc, double *pDec, double timeBegin)		
指令说明	GUS-400-TG02,GUS-600-TG02		
指令类型	向 PVT 运动模式指定数据表传送数据, 采用 Continuous 描述方式。		
指令参数	立即指令, 调用后立即生效。	章节页码	79
tableId	该指令共有 9 个参数, 参数的详细信息如下。		
count	指定数据表。取值范围: [1, 32]。		
pPos	数据点位置数组。单位: pulse, 数组长度为 count。		
pVel	数据点速度数组。单位: pulse/ms, 数组长度为 count。		
pPercent	数据点百分比数组。数组长度为 count。 百分比的取值范围: [0, 100]。		
pVelMax	数据点最大速度数组。单位: pulse/ms。数组长度为 count。		
pAcc	数据点加速度数组。单位是“pulse/ms ² ”。数组长度为 count。		
pDec	数据点减速度数组。单位是“pulse/ms ² ”。数组长度为 count。		
timeBegin	起点时间。单位: ms。		
指令返回值	若返回值为 1: 请检查当前轴是否为 PVT 模式, 若不是, 请先调用 GT_PrflPvt 将当前轴设置为 PVT 模式。 若当前轴在规划运动, 请调用 GT_Stop 停止运动再调用该指令。 (1) 请检查传递的数据点是否大于 1024 个。 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 6-17 Continuous 描述方式		

指令 136 GT_PvtTablePercent

指令原型	short GT_PvtTablePercent (short tableId, long count, double *pTime, double *pPos, double *pPercent, double velBegin)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	向 PVT 运动模式指定数据表传送数据, 采用 Percent 描述方式。		
指令类型	立即指令, 调用后立即生效。	章节页码	79
指令参数	该指令共有 6 个参数, 参数的详细信息如下。		
tableId	指定数据表。取值范围: [1, 32]。		
count	数据点个数。每个数据表具有 1024 个存储空间。 每个数据点占用 1~3 个存储空间。		
pTime	数据点时间数组。单位: ms, 数组长度为 count。		
pPos	数据点位置数组。单位: pulse, 数组长度为 count。		
pPercent	数据点百分比数组。数组长度为 count。 百分比的取值范围: [0, 100]。		
velBegin	起点速度。单位: pulse/ms。		
指令返回值	若返回值为 1: 请检查当前轴是否为 PVT 模式, 若不是, 请先调用 GT_PrflPvt 将当前轴设置为 PVT 模式。 若当前轴在规划运动, 请调用 GT_Stop 停止运动再调用该指令。		

	(1) 请检查传递的数据点是否大于 1024 个。 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 6-16 Percent 描述方式

指令 137 GT_PvtTableSelect

指令原型	short GT_PvtTableSelect(short profile, short tableId)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	选择 PVT 运动模式数据表。		
指令类型	立即指令，调用后立即生效。	章节页码	79
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
tableId	指定数据表。 PVT 模式提供 32 个数据表，取值范围：[1, 32]。		
指令返回值	若返回值为 1：目标数据表不应为空。请检查目标数据表是否空。 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 6-14 PVT 描述方式		

指令 138 GT_Reset

指令原型	short GT_Reset()		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	138
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	例程 6-2 Jog 运动		

指令 139 GT_RunThread

指令原型	short GT_RunThread(short thread)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	120
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
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	GT_PauseThread ; GT_StopThread		
指令示例	例程 10-1 单线程累加求和		

指令 140 GT_SetAdcFilter

指令原型	short GT_SetAdcFilter(short adc, short filterTime)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置模拟量输入的滤波器时间参数。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
adc	模拟量输入通道的编号，取值范围：[1, 6]。		
filterTime	模拟量输入信号的滤波器时间参数，取值范围：[0, 50]。无单位。		
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 141 GT_SetAxisBand

指令原型	short GT_SetAxisBand(short axis, long band, long time)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置轴到位误差带。 规划器静止，规划位置 and 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	140
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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	GT_GetAxisBand		
指令示例	例程 11-2 电机到位检测		

指令 142 GT_SetBacklash

指令原型	short GT_SetBacklash (short axis, long compValue, double compChangeValue, long compDir)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	145
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	需要进行反向间隙补偿的轴的编号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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, 1073741824]，单位：脉冲。		
compChangeValue	反向间隙补偿的变化量。取值范围：[0, 1073741824]，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 GT_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	GT_GetBacklash
指令示例	无。

指令 143 [GT_SetCaptureMode](#)

指令原型	short GT_SetCaptureMode (short encoder, short mode)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置编码器捕获方式，并启动捕获。		
指令类型	立即指令，调用后立即生效。	章节页码	105
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
encoder	编码器轴号 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围: [1, 6]。		
mode	编码器捕获模式 CAPTURE_HOME(该宏定义为 1) Home 捕获 CAPTURE_INDEX(该宏定义为 2) Index 捕获 CAPTURE_PROBE(该宏定义为 3) 探针捕获 CAPTURE_HSIO0(该宏定义为 6) HSIO0 口捕获 CAPTURE_HSIO1(该宏定义为 7) HSIO1 口捕获		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetCaptureMode		
指令示例	无。		

指令 144 [GT_SetCaptureSense](#)

指令原型	short GT_SetCaptureSense (short encoder, short mode, short sense)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置捕获电平。		
指令类型	立即指令，调用后立即生效。	章节页码	105
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
encoder	编码器轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围: [1, 6]。		
mode	编码器捕获模式 CAPTURE_HOME (该宏定义为 1): Home 捕获。 CAPTURE_INDEX (该宏定义为 2): Index 捕获。 CAPTURE_PROBE (该宏定义为 3): 探针捕获。 HSIO 捕获不能设置捕获电平。		

sense	捕获电平，可设置 0 或者 1。 0：下降沿触发。 1：上升沿触发。
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 145 GT_SetCardNo

指令原型	short GT_SetCardNo(short index)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	切换当前运动控制器卡号。		
指令类型	立即指令，调用后立即生效。	章节页码	138
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
index	将被设置为当前运动控制器的卡号。取值范围：[0, 15]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetCardNo		
指令示例	无。		

指令 146 GT_SetControlFilter

指令原型	short GT_SetControlFilter(short control, short index)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设定 PID 索引，支持 3 组 PID 参数。		
指令类型	立即指令，调用后立即生效。	章节页码	145
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
control	伺服控制器编号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
index	伺服控制参数的索引号。取值范围：[1, 3]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetControlFilter		
指令示例	无。		

指令 147 GT_SetCrdPrm

指令原型	short GT_SetCrdPrm(short crd, TCrdPrm *pCrdPrm)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号，取值范围：[1, 2]。		
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/ms²。

evenTime: 每个插补段的最小匀速段时间。取值范围: [0, 32767)。单位: ms。

setOriginFlag: 表示是否需要指定坐标系的原点坐标的规划位置, 该参数可以方便用户建立区别于机床坐标系的加工坐标系。0: 不需要指定原点坐标值, 则坐标系的原点在当前规划位置上。1: 需要指定原点坐标值, 坐标系的原点在 originPos 指定的规划位置上。

originPos[8]: 指定的坐标系原点的规划位置值。

若返回值为 1:

若坐标系下各轴在规划运动, 请调用 `GT_Stop` 停止运动再调用该指令。

(1) 请检查映射到 Profile 有没被激活, 若无, 则返回错误。

(2) 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。

指令返回值

相关指令

`GT_GetCrdPrm`

指令示例

例程 6-8 建立坐标系

指令 148 `GT_SetCrdStopDec`

指令原型	short <code>GT_SetCrdStopDec</code> (short crd, double decSmoothStop, double decAbruptStop)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	63
指令参数	该指令共有 3 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		
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	<code>GT_GetCrdStopDec</code>		
指令示例	无。		

指令 149 `GT_SetDac`

指令原型	short <code>GT_SetDac</code> (short dac, short *pValue, short count)
------	--

适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置 dac 输出电压。当闭环模式下，da 输出通道与轴挂接时，用户不能调用该指令直接输出电压。		
指令类型	立即指令，调用后立即生效。	章节页码	103
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
dac	dac 起始轴号。		
pValue	输出电压。8 路轴控接口 -32768 对应-10V，32767 对应+10V。 4 路非轴接口 0 对应 0V，32767 对应+10V。		
count	设置的通道数。默认为 1。 1 次最多可以设置 8 路 dac 输出。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetDac		
指令示例	无。		

指令 150 GT_SetDiReverseCount

指令原型	short GT_SetDiReverseCount(short diType, short diIndex, unsigned long *pReverseCount, short count)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置数字量输入信号的变化次数的初值。		
指令类型	立即指令，调用后立即生效。	章节页码	102
指令参数	该指令共有 4 个参数，参数的详细信息如下：		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。		
diIndex	数字量输入的索引。 取值范围： diType= MC_LIMIT_POSITIVE 时：[1, 8]。 diType= MC_LIMIT_NEGATIVE 时：[1, 8]。 diType= MC_ALARM 时：[1, 8]。 diType= MC_HOME 时：[1, 8]。 diType= MC_GPI 时：[1, 16]。 diType= MC_ARRIVE 时：[1, 8]。		
pReverseCount	设置的数字量输入的变化次数的初值。		
count	设置变化次数初值的数字量输入的个数。默认为 1。 1 次最多可以设置 4 个数字量输入的变化次数的初值。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetDiReverseCount		
指令示例	无。		

指令 151 GT_SetDo

指令原型	short GT_SetDo(short doType, long value)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置数字 IO 输出状态。若 do 有挂接轴，则对应的不能直接输出。默认驱动器使能与轴挂接，所以用户不能调用该指令设置驱动器使能输出的电平。		
指令类型	立即指令，调用后立即生效。	章节页码	102
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
doType	指定数字 IO 类型。 MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。		
value	按位指示数字 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetDo		
指令示例	无。		

指令 152 GT_SetDoBit

指令原型	short GT_SetDoBit(short doType, short doIndex, short value)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	按位设置数字 IO 输出状态		
指令类型	立即指令，调用后立即生效。	章节页码	102
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
doType	指定数字 IO 类型 MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。		
doIndex	输出 IO 的索引。 取值范围： doType=MC_ENABLE 时：[1, 8]。 doType=MC_CLEAR 时：[1, 8]。 doType=MC_GPO 时：[1, 16]。		
value	设置数字 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。		
指令返回值	若返回值为 1：检查设置输出的 bit 是否挂接轴，若挂接，则不能直接输出。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 153 GT_SetDoBitReverse

指令原型	short GT_SetDoBitReverse (short doType, short doIndex, short value, short reverseTime)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	使数字量输出信号输出定时脉冲信号。		
指令类型	立即指令，调用后立即生效。	章节页码	102
指令参数	该指令共有 4 个参数，参数的详细信息如下。		

doType	指定数字 IO 类型。 MC_ENABLE(该宏定义为 10): 驱动器使能。 MC_CLEAR(该宏定义为 11): 报警清除。 MC_GPO(该宏定义为 12): 通用输出。
doIndex	输出 IO 的索引。 取值范围: doType=MC_ENABLE 时: [1, 8]。 doType=MC_CLEAR 时: [1, 8]。 doType=MC_GPO 时: [1, 16]。
value	设置数字 IO 输出电平。 默认情况下, 1 表示高电平, 0 表示低电平。
reverseTime	维持 value 所设置电平的时间, 取值范围: [0, 32767], 单位: 250μs。
指令返回值	若返回值为 1: 检查设置输出的 bit 是否挂接轴, 若挂接, 则不能直接输出。 其他返回值: 请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 154 GT_SetEncPos

指令原型	short GT_SetEncPos (short encoder, long encPos)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
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 103
指令参数	该指令共有 2 个参数, 参数的详细信息如下。	
encoder	编码器起始轴号。正整数。 对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4], 第 11 轴为手轮接口专用编码器。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6], 第 11 轴为手轮接口专用编码器。	
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	GT_GetEncPos	
指令示例	无。	

指令 155 GT_SetFollowEvent

指令原型	short GT_SetFollowEvent(short profile, short event, short masterDir, long pos)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
指令说明	设置 Follow 运动模式启动跟随条件。	
指令类型	立即指令, 调用后立即生效。	章节页码 51
指令参数	该指令共有 4 个参数, 参数的详细信息如下。	
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。	
event	启动跟随条件。 FOLLOW_EVENT_START (该宏定义为 1) 表示调用 GT_FollowStart 以后立即启动。 FOLLOW_EVENT_PASS (该宏定义为 2) 表示主轴穿越设定位置以后启动跟随。	
masterDir	穿越启动时, 主轴的运动方向。	

	1 主轴正向运动, -1 主轴负向运动。
pos	穿越位置, 单位: pulse。 当 event 为 FOLLOW_EVENT_PASS 时有效。
指令返回值	若返回值为 1: 请检查当前轴是否为 Follow 模式, 若不是, 请先调用 GT_PrFFollow 将当前轴设置为 Follow 模式。 其他返回值: 请参照指令返回值列表。
相关指令	立即指令, 调用后立即生效。
指令示例	例程 6-7 Follow 双 FIFO

指令 156 GT_SetFollowLoop

指令原型	short GT_SetFollowLoop(short profile, short loop)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
指令说明	设置 Follow 运动模式下的循环次数。	
指令类型	立即指令, 调用后立即生效。	章节页码 51
指令参数	该指令共有 2 个参数, 参数的详细信息如下。	
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。	
loop	指定 Follow 模式循环执行的次数。 取值范围: [-32768, 32767]。注: loop 小于 1 表示无限次循环。	
指令返回值	若返回值为 1: 请检查当前轴是否为 Follow 模式, 若不是, 请先调用 GT_PrFFollow 将当前轴设置为 Follow 模式。 其他返回值: 请参照指令返回值列表。	
相关指令	立即指令, 调用后立即生效。	
指令示例	例程 6-7 Follow 双 FIFO	

指令 157 GT_SetFollowMaster

指令原型	short GT_SetFollowMaster (short profile, short masterIndex, short masterType = FOLLOW_MASTER_PROFILE, short masterItem)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
指令说明	设置 Follow 运动模式下的跟随主轴。	
指令类型	立即指令, 调用后立即生效。	章节页码 51
指令参数	该指令共有 4 个参数, 参数的详细信息如下。	
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。	
masterIndex	主轴索引。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。 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。 FOLLOW_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。	

masterItem	FOLLOW_MASTER_ENCODER (该宏定义为 1) 表示跟随编码器(encoder)的输出值。 FOLLOW_MASTER_AXIS (该宏定义为 3) 表示跟随轴(axis)的输出值。
	合成轴类型, 当 masterType= FOLLOW_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: 若当前轴在规划运动, 请调用 GT_Stop 停止运动再调用该指令。 请检查当前轴是否为 Follow 模式, 若不是, 请先调用 GT_PrFFollow 将当前轴设置为 Follow 模式。 其他返回值: 请参照指令返回值列表
相关指令	GT_GetFollowMaster
指令示例	例程 6-6 Follow 单 FIFO, 例程 6-7 Follow 双 FIFO

指令 158 GT_SetFollowMemory

指令原型	short GT_SetFollowMemory (short profile, short memory)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
指令说明	设置 Follow 运动模式的缓存区大小。	
指令类型	立即指令, 调用后立即生效。	章节页码 51
指令参数	该指令共有 2 个参数, 参数的详细信息如下。	
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。	
	Follow 运动缓存区大小标志。 0: 每个 Follow 运动缓存区有 16 段空间。 1: 每个 Follow 运动缓存区有 512 段空间。	
memory	Follow 运动缓存区大小标志。 0: 每个 Follow 运动缓存区有 16 段空间。 1: 每个 Follow 运动缓存区有 512 段空间。	
指令返回值	若返回值为 1: 若当前轴在规划运动, 请调用 GT_Stop 停止运动再调用该指令。 请检查当前轴是否为 Follow 模式, 若不是, 请先调用 GT_PrFFollow 将当前轴设置为 Follow 模式。 其他返回值: 请参照指令返回值列表。	
相关指令	GT_GetFollowMemory	
指令示例	无。	

指令 159 GT_SetGearMaster

指令原型	short GT_SetGearMaster(short profile, short masterIndex, short masterType, short masterItem)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
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 48
指令参数	该指令共有 4 个参数, 参数的详细信息如下。	
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。	
	主轴索引。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。	
masterIndex	主轴索引。 正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。	

	对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。 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。
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： 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 请检查当前轴是否为电子齿轮模式，若不是，请先调用 GT_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	GT_GetGearMaster
指令示例	例程 6-5 电子齿轮跟随

指令 160 GT_SetGearRatio

指令原型	short GT_SetGearRatio(short profile, long masterEven, long slaveEven, long masterSlope)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	48
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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	主轴离合器位移。 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 GT_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	GT_GetGearRatio		
指令示例	例程 6-5 电子齿轮跟随		

指令 161 GT_SetJogPrm

指令原型	short GT_SetJogPrm(short profile, TJogPrm *pPrm)
适用板卡	GUS-400-TG02,GUS-600-TG02
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	36
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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： <pre>typedef struct JogPrm { double acc; double dec; double smooth; } TJogPrm;</pre> 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： 若当前轴在规划运动，请调用 <code>GT_Stop</code> 停止运动再调用该指令。 请检查当前轴是否为 Jog 模式，若不是，请先调用 <code>GT_PrjJog</code> 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	GT_GetJogPrm		
指令示例	例程 6-2 Jog 运动		

指令 162 GT_SetMtrBias

指令原型	short GT_SetMtrBias(short dac, short bias)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置模拟量输出通道的零漂电压补偿值。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
dac	模拟量输出通道号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
bias	零漂补偿值，取值范围：[-32768, 32767]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetMtrBias		
指令示例	无。		

指令 163 GT_SetMtrLmt

指令原型	short GT_SetMtrLmt(short dac, short limit)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置模拟量输出通道的输出电压饱和极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 2 个参数，参数的详细信息如下。		

dac	模拟量输出通道号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
limit	输出电压饱和极限值，取值范围：(0, 32767]。若设置为32767，则限制允许输出的电压范围为：-10V~+10V；若设置为 16384，则限制允许输出的电压范围为：-5V~+5V。
指令返回值	请参照指令返回值列表。
相关指令	GT_GetMtrLmt
指令示例	无。

指令 164 GT_SetOverride

指令原型	short GT_SetOverride (short crd, double synVelRatio)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
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 63
指令参数	该指令共有 2 个参数，参数的详细信息如下。	
crd	坐标系号。正整数，取值范围：[1, 2]。	
synVelRatio	设置的插补目标速度倍率，取值范围：(0, 1]，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 165 GT_SetPid

指令原型	short GT_SetPid(short control, short index, TPid *pPid)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
指令说明	设置 PID 参数。	
指令类型	立即指令，调用后立即生效。	章节页码 145
指令参数	该指令共有 3 个参数，参数的详细信息如下。	
control	伺服控制器编号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。	
index	伺服控制参数的索引号，取值范围：[1, 3]。	
pPid	设置 PID 参数 typedef struct Pid { double kp; double ki; double kd; double kvff; double kaff; long integralLimit; long derivativeLimit; short limit; }TPid;	

	kp : 比例增益; ki : 积分增益; kd : 微分增益; kvff : 速度前馈系数; kaff : 加速度前馈系数; integralLimit : 积分饱和极限; derivativeLimit : 微分饱和极限; limit : 控制量输出饱和极限;
指令返回值	请参照指令返回值列表。
相关指令	GT_GetPid
指令示例	例程 11-2 电机到位检测

指令 166 GT_SetPos

指令原型	short GT_SetPos(short profile, long pos)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	33
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
pos	设置目标位置，单位：pulse。取值范围：[-1073741823, 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 GT_PrTrp 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	GT_GetPos		
指令示例	例程 6-1 点位运动		

指令 167 GT_SetPosErr

指令原型	short GT_SetPosErr(short control, long error)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置跟随误差极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
	伺服控制器编号。		
control	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
error	跟随误差极限值，取值范围：(0, 2147483648]。单位：pulse。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetPosErr		
指令示例	无。		

指令 168 GT_SetPrfPos

指令原型	short GT_SetPrfPos(short profile, long prfPos)
------	--

适用板卡	GUS-400-TG02,GUS-600-TG02		
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	139
指令参数	该指令共有 2 个参数，参数的详细信息如下：		
	规划轴编号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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	GT_GetPrfPos		
指令示例	例程 6-1 点位运动		

指令 169 GT_SetPtLoop

指令原型	short GT_SetPtLoop(short profile, long loop)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置 PT 运动模式循环执行的次数。动态模式下该指令无效。		
指令类型	立即指令，调用后立即生效。	章节页码	40
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
	指定 PT 模式循环执行的次数。		
loop	取值范围：非负整数。如果需要无限循环，设置为 0。 动态模式下该参数无效。		
指令返回值	若返回值为 1：请检查当前轴是否为 PT 模式，若不是，请先调用 GT_PrftPt 将当前轴设置为 PT 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_GetPtLoop		
指令示例	无。		

指令 170 GT_SetPtMemory

指令原型	short GT_SetPtMemory(short profile, short memory)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置 PT 运动模式的缓存区(FIFO)大小。		
指令类型	立即指令，调用后立即生效。	章节页码	40
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
	PT 运动缓存区大小标志：		
memory	0：每个 PT 运动缓存区有 32 段空间。 1：每个 PT 运动缓存区有 1024 段空间。		
指令返回值	若返回值为 1： 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。		

相关指令	请检查当前轴是否为 PT 模式，若不是，请先调用 GT_PrflPt 将当前轴设置为 PT 模式。 其他返回值：请参照指令返回值列表。
指令示例	GT_GetPtMemory 无。

指令 171 GT_SetPvtLoop

指令原型	short GT_SetPvtLoop(short profile, long loop)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	设置 PVT 运动模式循环次数。		
指令类型	立即指令，调用后立即生效。	章节页码	79
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
loop	指定循环执行的次数。 0 表示无限循环。		
指令返回值	若返回值为 1：请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrflPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_GetPvtLoop		
指令示例	例程 6-15 Complete 描述方式		

指令 172 GT_SetSoftLimit

指令原型	short GT_SetSoftLimit (short axis, long positive, long negative)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	116
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
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：0x7ffffff，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	GT_GetSoftLimit		
指令示例	无。		

指令 173 GT_SetStopDec

指令原型	short GT_SetStopDec(short profile, double decSmoothStop, double decAbruptStop)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	29

指令类型	立即指令，调用后立即生效。
指令参数	该指令共有 3 个参数，参数的详细信息如下。
profile	规划器的编号 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
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	GT_GetStopDec
指令示例	无。

指令 174 GT_SetStopIo

指令原型	short GT_SetStopIo(short axis, short stopType, short inputType, short inputIndex)
适用板卡	GUS-400-TG02,GUS-600-TG02
指令说明	设置平滑停止和紧急停止数字量输入的信息。
指令类型	立即指令，调用后立即生效。
指令参数	该指令共有 2 个参数，参数的详细信息如下。
axis	需要设置停止IO信息的轴的编号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。
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。 对于 GUS-600-TG02 系列的控制卡： 当 inputType= MC_LIMIT_POSITIVE 时，取值范围：[1, 6]； 当 inputType= MC_LIMIT_NEGATIVE 时，取值范围：[1, 6]； 当 inputType= MC_ALARM 时，取值范围：[1, 6]； 当 inputType= MC_HOME 时，取值范围：[1, 6]； 当 inputType= MC_GPI 时，取值范围：[1, 16]。 对于 GUS-400-TG02 系列的控制卡： 当 inputType= MC_LIMIT_POSITIVE 时，取值范围：[1, 4]； 当 inputType= MC_LIMIT_NEGATIVE 时，取值范围：[1, 4]； 当 inputType= MC_ALARM 时，取值范围：[1, 4]； 当 inputType= MC_HOME 时，取值范围：[1, 4]； 当 inputType= MC_GPI 时，取值范围：[1, 16]； 当 inputType= MC_ARRIVE 时，取值范围：[1, 4]。
指令返回值	请参照指令返回值列表。

章节页码

29

相关指令	无。
指令示例	无。

指令 175 GT_SetTrapPrm

指令原型	short GT_SetTrapPrm(short profile, TTrapPrm *pPrm)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
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 33
指令参数	该指令共有 4 个参数，参数的详细信息如下。	
profile	规划轴号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。	
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： <pre>typedef struct TrapPrm { double acc; double dec; double velStart; short smoothTime; }TTrapPrm;</pre> 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： 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 请检查当前轴是否为 Trap 模式，若不是，请先调用 GT_PrflTrap 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	GT_GetTrapPrm	
指令示例	例程 9-1 限位	

指令 176 GT_SetUserSegNum

指令原型	short GT_SetUserSegNum(short crd, long segNum, short fifo=0)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
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 63
指令参数	该指令共有 3 个参数，参数的详细信息如下。	
crd	坐标系号。正整数，取值范围：[1, 2]。	
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	<p>若返回值为 1:</p> <ol style="list-style-type: none"> (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。 <p>其他返回值: 请参照指令返回值列表。</p>
相关指令	GT_GetUserSegNum
指令示例	无。

指令 177 GT_SetVarValue

指令原型	short GT_SetVarValue (short page, TVarInfo *pVarInfo, double *pValue, short count=1)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	120
指令参数	该指令共有 4 个参数, 参数的详细信息如下。		
	数据页编号。		
page	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, 6]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetVarValue		
指令示例	例程 10-1 单线程累加求和		

指令 178 GT_SetVel

指令原型	short GT_SetVel(short profile, double vel)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	33,36
指令参数	该指令共有 2 个参数, 参数的详细信息如下。		
	规划轴号。		
profile	正整数。对于 GUS-400-TG02 系列的控制卡, 取值范围: [1, 4]。 对于 GUS-600-TG02 系列的控制卡, 取值范围: [1, 6]。		
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 GT_PrTrp 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	GT_GetVel		
指令示例	例程 6-1 点位运动		

指令 179 GT_StepDir

指令原型	short GT_StepDir(short step)
适用板卡	GUS-400-TG02,GUS-600-TG02

指令说明	将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
step	脉冲输出通道号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
指令返回值	若返回值为 1： (1)若当前轴在规划运动，请调用 <code>GT_Stop</code> 停止运动再调用该指令；。 (2)若当前轴伺服使能，请调用 <code>GT_AxisOff</code> 停止使能再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 180 `GT_StepPulse`

指令原型	<code>short GT_StepPulse(short step)</code>		
适用板卡	GUS-400-TG02,GUS-600-TG02		
指令说明	将脉冲输出通道的脉冲输出模式设置为“CCW/CW”。		
指令类型	立即指令，调用后立即生效。	章节页码	29
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
step	脉冲输出通道号。 正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。		
指令返回值	若返回值为 1： (1)若当前轴在规划运动，请调用 <code>GT_Stop</code> 停止运动再调用该指令。 (2)若当前轴伺服使能，请调用 <code>GT_AxisOff</code> 停止使能再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 181 `GT_Stop`

指令原型	<code>short GT_Stop(long mask, long option)</code>																																														
适用板卡	GUS-400-TG02,GUS-600-TG02																																														
指令说明	停止一个或多个轴的规划运动，停止坐标系运动。																																														
指令类型	立即指令，调用后立即生效。	章节页码	31																																												
指令参数	该指令共有 2 个参数，参数的详细信息如下。																																														
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。 对于 GUS-400-TG02 系列的控制卡： <table border="1" style="width: 100%; text-align: center;"> <tr> <td>Bit</td> <td>9</td> <td>8</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>对应轴或坐标系</td> <td>坐标系 2</td> <td>坐标系 1</td> <td>/</td> <td>/</td> <td>/</td> <td>/</td> <td>4 轴</td> <td>3 轴</td> <td>2 轴</td> <td>1 轴</td> </tr> </table> 对于 GUS-600-TG02 系列的控制卡： <table border="1" style="width: 100%; text-align: center;"> <tr> <td>Bit</td> <td>9</td> <td>8</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>对应轴或坐标系</td> <td>坐标系 2</td> <td>坐标系 1</td> <td>/</td> <td>/</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>			Bit	9	8	7	6	5	4	3	2	1	0	对应轴或坐标系	坐标系 2	坐标系 1	/	/	/	/	4 轴	3 轴	2 轴	1 轴	Bit	9	8	7	6	5	4	3	2	1	0	对应轴或坐标系	坐标系 2	坐标系 1	/	/	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
Bit	9	8	7	6	5	4	3	2	1	0																																					
对应轴或坐标系	坐标系 2	坐标系 1	/	/	/	/	4 轴	3 轴	2 轴	1 轴																																					
Bit	9	8	7	6	5	4	3	2	1	0																																					
对应轴或坐标系	坐标系 2	坐标系 1	/	/	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴																																					
option	按位指示停止方式。当 bit 位为 0 时表示平滑停止对应的轴或坐标系，当 bit 位为 1 时表																																														

指令返回值 相关指令 指令示例	示急停对应的轴或坐标系。										
	对于 GUS-400-TG02 系列的控制卡：										
	Bit	9	8	7	6	5	4	3	2	1	0
	对应轴或坐标系	坐标系 2	坐标系 1	/	/	/	/	4 轴	3 轴	2 轴	1 轴
	对于 GUS-600-TG02 系列的控制卡：										
Bit	9	8	7	6	5	4	3	2	1	0	
对应轴或坐标系	坐标系 2	坐标系 1	/	/	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴	
	请参照指令返回值列表。										
	无。										
	无。										

指令 182 GT_StopThread

指令原型	short GT_StopThread(short thread)		
适用板卡	GUS-400-TG02,GUS-600-TG02		
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	120
指令类型	立即指令，调用后立即生效。		
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
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	GT_RunThread; GT_PauseThread		
指令示例	无。		

指令 183 GT_SynchAxisPos

指令原型	short GT_SynchAxisPos(long mask)						
适用板卡	GUS-400-TG02,GUS-600-TG02						
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					139
指令参数	该指令共有 1 个参数，参数的详细信息如下。						
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。						
	对于 GUS-400-TG02 系列的控制卡：						
	Bit	3	2	1	0		
	对应轴	4 轴	3 轴	2 轴	1 轴		
	对于 GUS-600-TG02 系列的控制卡：						
Bit	5	4	3	2	1	0	
对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴	
指令返回值	若返回值为 1：请检查参数 mask 是否设置为 0。 其他返回值：请参照指令返回值列表。						
相关指令	无。						

指令示例	无。
------	----

指令 184 GT_Update

指令原型	short GT_Update(long mask)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
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 33,36
指令参数	该指令共有 1 个参数，参数的详细信息如下。	
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。	
	对于 GUS-400-TG02 系列的控制卡：	
	Bit	3 2 1 0
	对应轴	4 轴 3 轴 2 轴 1 轴
mask	对于 GUS-600-TG02 系列的控制卡：	
	Bit	5 4 3 2 1 0
	对应轴	6 轴 5 轴 4 轴 3 轴 2 轴 1 轴
	指令返回值	
相关指令	请参照指令返回值列表。	
指令示例	无。	
指令示例	例程 6-1 点位运动	

指令 185 GT_ZeroPos

指令原型	short GT_ZeroPos(short axis, short count)	
适用板卡	GUS-400-TG02,GUS-600-TG02	
指令说明	清零规划位置 and 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 139
指令参数	该指令共有 2 个参数，参数的详细信息如下。	
axis	需要位置清零的起始轴号。	
	正整数。对于 GUS-400-TG02 系列的控制卡，取值范围：[1, 4]。 对于 GUS-600-TG02 系列的控制卡，取值范围：[1, 6]。	
count	需要位置清零的轴数。	
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_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 8-2 Home 回原点	

第13章 索引

13.1 指令索引

指令 1	GT_AlarmOff	151
指令 2	GT_AlarmOn	151
指令 3	GT_ArcXYC	151
指令 4	GT_ArcXYR	152
指令 5	GT_ArcYZC	153
指令 6	GT_ArcYZR	153
指令 7	GT_ArcZXC	154
指令 8	GT_ArcZXR	155
指令 9	GT_AxisOff	155
指令 10	GT_AxisOn	156
指令 11	GT_Bind	156
指令 12	GT_BufDA	156
指令 13	GT_BufDelay	157
指令 14	GT_BufGear	157
指令 15	GT_BufIO	158
指令 16	GT_BufLmtsOff	158
指令 17	GT_BufLmtsOn	159
指令 18	GT_BufMove	159
指令 19	GT_BufSetStopIo	160
指令 20	GT_ClearCaptureStatus	161
指令 21	GT_Close	161
指令 22	GT_ClrSts	161
指令 23	GT_CrdClear	162
指令 24	GT_CrdData	162
指令 25	GT_CrdSpace	162
指令 26	GT_CrdStart	163
指令 27	GT_CrdStatus	163
指令 28	GT_CtrlMode	164
指令 29	GT_Download	164
指令 30	GT_EncOff	164
指令 31	GT_EncOn	165
指令 32	GT_EncScale	165
指令 33	GT_EncSns	165
指令 34	GT_FollowClear	166
指令 35	GT_FollowData	166
指令 36	GT_FollowSpace	167
指令 37	GT_FollowStart	167
指令 38	GT_FollowSwitch	168
指令 39	GT_GearStart	168
指令 40	GT_GetAdc	169

指令 41	GT_GetAdcValue	169
指令 42	GT_GetAxisBand	170
指令 43	GT_GetAxisEncAcc	170
指令 44	GT_GetAxisEncPos	170
指令 45	GT_GetAxisEncVel	171
指令 46	GT_GetAxisError	171
指令 47	GT_GetAxisPrfAcc	172
指令 48	GT_GetAxisPrfPos	172
指令 49	GT_GetAxisPrfVel	173
指令 50	GT_GetBacklash	173
指令 51	GT_GetCaptureMode	173
指令 52	GT_GetCaptureStatus	174
指令 53	GT_GetCardNo	174
指令 54	GT_GetClock	174
指令 55	GT_GetClockHighPrecision	175
指令 56	GT_GetControlFilter	175
指令 57	GT_GetCrdPos	175
指令 58	GT_GetCrdPrm	176
指令 59	GT_GetCrdStopDec	176
指令 60	GT_GetCrdVel	176
指令 61	GT_GetDac	177
指令 62	GT_GetDi	177
指令 63	GT_GetDiRaw	177
指令 64	GT_GetDiReverseCount	178
指令 65	GT_GetDo	178
指令 66	GT_GetEncPos	179
指令 67	GT_GetEncVel	179
指令 68	GT_GetFollowEvent	180
指令 69	GT_GetFollowLoop	180
指令 70	GT_GetFollowMaster	181
指令 71	GT_GetFollowMemory	181
指令 72	GT_GetFunId	182
指令 73	GT_GetGearMaster	182
指令 74	GT_GetGearRatio	182
指令 75	GT_GetJogPrm	183
指令 76	GT_GetMtrBias	183
指令 77	GT_GetMtrLmt	184
指令 78	GT_GetPid	184
指令 79	GT_GetPos	184
指令 80	GT_GetPosErr	185
指令 81	GT_GetPrfAcc	185
指令 82	GT_GetPrfMode	185
指令 83	GT_GetPrfPos	186
指令 84	GT_GetPrfVel	186
指令 85	GT_GetPtLoop	187
指令 86	GT_GetPtMemory	187

指令 87	GT_GetPvtLoop	188
指令 88	GT_GetRemainderSegNum.....	188
指令 89	GT_GetSoftLimit.....	188
指令 90	GT_GetStopDec	189
指令 91	GT_GetSts	189
指令 92	GT_GetThreadSts.....	189
指令 93	GT_GetTrapPrm	190
指令 94	GT_GetUserSegNum.....	190
指令 95	GT_GetVarId	191
指令 96	GT_GetVarValue.....	191
指令 97	GT_GetVel.....	191
指令 98	GT_GetVersion.....	192
指令 99	GT_GpiSns	192
指令 100	GT_Home	192
指令 101	GT_HomeInit.....	193
指令 102	GT_HomeStop.....	193
指令 103	GT_HomeSts	193
指令 104	GT_Index.....	194
指令 105	GT_InitLookAhead.....	194
指令 106	GT_LmtSns.....	194
指令 107	GT_LmtsOff	195
指令 108	GT_LmtsOn.....	195
指令 109	GT_LnXY.....	196
指令 110	SGT_LnXYG0.....	196
指令 111	GT_LnXYZ.....	197
指令 112	GT_LnXYZA.....	197
指令 113	GT_LnXYZAG0.....	198
指令 114	GT_LnXYZG0.....	198
指令 115	GT_LoadConfig.....	199
指令 116	GT_Open	199
指令 117	GT_PauseThread	200
指令 118	GT_PrfFollow	200
指令 119	GT_PrfGear	200
指令 120	GT_PrfJog.....	201
指令 121	GT_PrfPt.....	201
指令 122	GT_PrfPvt.....	201
指令 123	GT_PrfTrap.....	202
指令 124	GT_ProfileScale	202
指令 125	GT_PtClear	202
指令 126	GT_PtData	203
指令 127	GT_PtSpace	203
指令 128	GT_PtStart	204
指令 129	GT_PvtContinuousCalculate	205
指令 130	GT_PvtPercentCalculate.....	205
指令 131	GT_PvtStart	205
指令 132	GT_PvtStatus	206

指令 133	GT_PvtTable	206
指令 134	GT_PvtTableComplete	207
指令 135	GT_PvtTableContinuous	207
指令 136	GT_PvtTablePercent	208
指令 137	GT_PvtTableSelect	209
指令 138	GT_Reset	209
指令 139	GT_RunThread	209
指令 140	GT_SetAdcFilter	210
指令 141	GT_SetAxisBand	210
指令 142	GT_SetBacklash	210
指令 143	GT_SetCaptureMode	211
指令 144	GT_SetCaptureSense	211
指令 145	GT_SetCardNo	212
指令 146	GT_SetControlFilter	212
指令 147	GT_SetCrdPrm	212
指令 148	GT_SetCrdStopDec	213
指令 149	GT_SetDac	213
指令 150	GT_SetDiReverseCount	214
指令 151	GT_SetDo	214
指令 152	GT_SetDoBit	215
指令 153	GT_SetDoBitReverse	215
指令 154	GT_SetEncPos	216
指令 155	GT_SetFollowEvent	216
指令 156	GT_SetFollowLoop	217
指令 157	GT_SetFollowMaster	217
指令 158	GT_SetFollowMemory	218
指令 159	GT_SetGearMaster	218
指令 160	GT_SetGearRatio	219
指令 161	GT_SetJogPrm	219
指令 162	GT_SetMtrBias	220
指令 163	GT_SetMtrLmt	220
指令 164	GT_SetOverride	221
指令 165	GT_SetPid	221
指令 166	GT_SetPos	222
指令 167	GT_SetPosErr	222
指令 168	GT_SetPrfPos	222
指令 169	GT_SetPtLoop	223
指令 170	GT_SetPtMemory	223
指令 171	GT_SetPvtLoop	224
指令 172	GT_SetSoftLimit	224
指令 173	GT_SetStopDec	224
指令 174	GT_SetStopIo	225
指令 175	GT_SetTrapPrm	226
指令 176	GT_SetUserSegNum	226
指令 177	GT_SetVarValue	227
指令 178	GT_SetVel	227

指令 179	GT_StepDir.....	227
指令 180	GT_StepPulse.....	228
指令 181	GT_Stop.....	228
指令 182	GT_StopThread.....	229
指令 183	GT_SynchAxisPos.....	229
指令 184	GT_Update.....	230
指令 185	GT_ZeroPos.....	230

13.2 例程索引

例程 6-1	点位运动.....	34
例程 6-2	Jog 运动.....	37
例程 6-3	PT 静态 FIFO.....	42
例程 6-4	PT 动态 FIFO.....	44
例程 6-5	电子齿轮跟随.....	49
例程 6-6	Follow 单 FIFO.....	53
例程 6-7	Follow 双 FIFO.....	57
例程 6-8	建立坐标系.....	65
例程 6-9	直线插补.....	66
例程 6-10	圆弧插补.....	68
例程 6-11	前瞻预处理.....	73
例程 6-12	点位运动中的刀向跟随.....	76
例程 6-13	跟随运动中的刀向跟随.....	78
例程 6-14	PVT 描述方式.....	88
例程 6-15	Complete 描述方式.....	89
例程 6-16	Percent 描述方式.....	93
例程 6-17	Continuous 描述方式.....	97
例程 7-1	访问数字 IO.....	102
例程 7-2	访问编码器.....	103
例程 8-1	Home/Index 硬件捕获.....	105
例程 8-2	Home 回原点.....	108
例程 8-3	Home+Index 回原点.....	112
例程 9-1	限位.....	117
例程 10-1	单线程累加求和.....	121
例程 10-2	多线程累加求和.....	124
例程 10-3	组合运动.....	126
例程 11-1	读取固件版本.....	139
例程 11-2	电机到位检测.....	140
例程 11-3	自动回原点.....	147

13.3 图片索引

图 2-1	GRT 快捷方式.....	14
图 4-1	步进控制.....	17
图 4-2	伺服控制.....	17
图 4-3	MCT2008 运动控制器管理软件.....	18

图 4-4	axis 配置界面.....	19
图 4-5	step 配置界面.....	21
图 4-6	dac 配置界面.....	22
图 4-7	encoder 配置界面.....	23
图 4-8	control 配置界面.....	24
图 4-9	profile 配置界面.....	25
图 4-10	di 配置界面.....	26
图 4-11	do 配置界面.....	27
图 4-12	生成配置文件界面.....	28
图 6-1	点位运动速度曲线.....	34
图 6-2	点位运动速度规划.....	34
图 6-3	Jog 模式速度曲线.....	37
图 6-4	Jog 模式动态改变目标速度.....	37
图 6-5	PT 运动速度曲线.....	40
图 6-6	PT 模式匀速段类型.....	41
图 6-7	PT 模式停止段类型.....	41
图 6-8	PT 模式梯形曲线速度规划.....	42
图 6-9	PT 模式正弦曲线速度规划.....	44
图 6-10	电子齿轮模式速度曲线.....	48
图 6-11	电子齿轮速度规划.....	49
图 6-12	Follow 模式速度曲线.....	52
图 6-13	Follow 模式切换 FIFO.....	53
图 6-14	Follow 速度规划.....	54
图 6-15	Follow 切换 FIFO.....	58
图 6-16	右手坐标系.....	64
图 6-17	evenTime=0.....	65
图 6-18	evenTime>0.....	65
图 6-19	加工坐标系偏移量示意图.....	66
图 6-20	直线插补例程运动结果.....	68
图 6-21	圆弧插补例程运动结果.....	70
图 6-22	圆弧插补逆时针方向.....	70
图 6-23	半径取正值/负值圆弧插补示意图.....	71
图 6-24	圆心表示方法示意图.....	71
图 6-25	X-Y 平面多段轨迹图形.....	72
图 6-26	X-Y 平面小线段轨迹图形.....	72
图 6-27	使用和不使用前瞻预处理功能模块的速度曲线对比图.....	73
图 6-28	没有进行前瞻预处理的合成速度曲线.....	75
图 6-29	进行了前瞻预处理后的合成速度曲线.....	75
图 6-30	插补缓存区内的点位运动速度图.....	77
图 6-31	插补缓存区内的跟随运动速度图.....	79
图 6-32	循环执行数据表.....	81
图 6-33	合理的 PVT 数据点参数.....	82
图 6-34	不合理的 PVT 数据点参数.....	83
图 6-35	Complete 描述方式.....	83
图 6-36	Complete 方式描述三角函数.....	84
图 6-37	数据点数分别为 5、10、50 时的位置误差.....	84

图 6-38	百分比的定义	85
图 6-39	Percent 描述方式	86
图 6-40	Continuous 描述方式	86
图 6-41	Continuous 描述方式	87
图 6-42	PVT 例程速度曲线	87
图 6-43	速度曲线	90
图 6-44	X 轴和 Y 轴的运动规律	93
图 6-45	X-Y 位置图	95
图 6-46	X 轴和 Y 轴的速度曲线	98
图 8-1	Home 回原点示意图 1	107
图 8-2	Home 回原点示意图 2	107
图 8-3	Home 回原点示意图 3	108
图 8-4	Home+Index 回原点示意图 1	111
图 8-5	Home+Index 回原点示意图 2	111
图 8-6	Home+Index 回原点示意图 3	111
图 8-7	Home+Index 回原点示意图 4	111
图 8-8	Home+Index 回原点示意图 5	111
图 9-1	轴运动范围	116
图 9-2	软限位触发	117
图 10-1	组合运动时序图	127
图 11-1	运动状态检测	141

13.4 表格索引

表 1-1	GUS 系列控制器指令列表	7
表 3-1	运动控制器指令返回值定义	15
表 4-1	反馈脉冲方向与编码器技术方向关系图	23
表 4-2	下载配置文件指令	28
表 4-3	配置信息指令列表	29
表 4-4	编码器计数方向设置	29
表 4-5	设置限位开发触发电平	30
表 5-1	运动状态检测指令列表	31
表 5-2	轴状态定义	31
表 6-1	设置运动模式指令列表	33
表 6-2	设置点位运动指令列表	33
表 6-3	设置 Jog 模式指令列表	36
表 6-4	设置 PT 模式指令列表	40
表 6-5	设置电子齿轮指令列表	48
表 6-6	设置 Follow 指令列表	51
表 6-7	设置插补运动指令列表	63
表 6-8	PVT 指令列表	79
表 6-9	PVT 案例示范数据	80
表 6-10	合理的 PVT 数据点参数	81
表 6-11	不合理的 PVT 数据点	82
表 6-12	Complete 描述方式示范数据点	83
表 6-13	Complete 方式描述三角函数示范数据点	84

表 6-14	Percent 描述方式示范数据点	85
表 6-15	Continuous 描述方式示范数据点	86
表 6-16	PVT 例程速度曲线数据点	88
表 6-17	Percent 描述方式 X 轴数据点	93
表 6-18	Percent 描述方式 X 轴数据点计算结果	94
表 6-19	Percent 描述方式 Y 轴数据点	94
表 6-20	Percent 描述方式 Y 轴数据点计算结果	94
表 7-1	访问数字 IO 指令列表	102
表 7-2	访问编码器指令列表	103
表 7-3	问 DAC 指令列表	104
表 7-4	问模拟量输入指令列表	104
表 8-1	Home/Index 硬件捕获指令列表	105
表 9-1	软限位指令列表	116
表 10-1	运动程序指令列表	120
表 11-1	打开/关闭运动控制器指令列表	138
表 11-2	读取固件版本号指令列表	138
表 11-3	运动控制器固件版本号定义	138
表 11-4	读取系统时钟指令列表	139
表 11-5	打开/关闭电机使能信号指令列表	139
表 11-6	维护位置值函数列表	139
表 11-7	电机到位检测函数列表	140
表 11-8	设置 PID 参数函数列表	145
表 11-9	反向间隙补偿函数列表	145
表 11-10	自动回原点功能函数列表	146