固高科技
GOOGOL TECHNOLOGY

扩展功能

GSN 系列运动控制器编程手册

DMA 功能

R1.1

2019 年 04 月

版权申明

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

联系我们

固高科技（深圳）有限公司

地　　址：深圳市高新技术产业园南区深港产学研基地西座二楼 W211 室

电　　话：0755-26970817　26737236　26970824

传　　真：0755-26970821

电子邮件：support@googoltech.com

网　　址：http://www.googoltech.com.cn

固高科技（香港）有限公司

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

電　　話：+(852) 2358-1033

傳　　真：+(852) 2719-8399

電子郵件：info@googoltech.com

網　　址：http://www.googoltech.com

臺灣固高科技股份有限公司

地　　址：台中室西屯區台中港路三段 97 號 7 樓之 3

電　　話：+886-4-23588245

傳　　真：+886-4-23586495

電子郵件：googoltw@googoltech.com

文档版本

版本号	修订日期
1.0	2018 年 10 月 16 日
1.1	2019 年 04 月 30 日

前言

感谢选用固高运动控制器

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

固高产品的更多信息

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

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

技术支持和售后服务

您可以通过以下途径获得我们的技术支持和售后服务：

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

编程手册的用途

用户通过阅读本手册，能够了解运动控制器的功能，掌握函数的用法，熟悉编程实现。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

本手册由四章内容组成，详细介绍了运动控制器的 Gantry 功能及编程实现。

相关文件

关于控制器的调试和安装，请参见随产品配套的运动控制器用户手册。
关于控制器基本功能使用，请参见随产品配套的《GSN 系列运动控制器编程手册之基本功能》
关于更复杂的控制器功能，请参见随产品配套的《GSN 系列运动控制器编程手册之高级功能》
关于扩展模块的使用，请参见随产品配套的扩展模块编程手册。

相关手册及控制器适用文档列表见于光盘的 manual 目录下。
亦可通过固高科技公司网站下载如驱动程序、dll 文件、例程、Demo 等相关文件，网址为：www.googoltech.com.cn/pro_view-53.html

目录

版权申明 **1**

联系我们 **1**

文档版本 **2**

前言 **3**

目录 **4**

索引 **5**

1. 表格索引 5
2. 例程索引 5
3. 指令索引 5

一、 指令列表 **6**

二、重点说明 **6**

三、例程 **6**

四、指令详细说明 **12**

索引

1. 表格索引

表 1 DMA 功能指令列表 6

2. 例程索引

例程 1 未使用前瞻的插补运动 DMA 6

例程 2 使用新前瞻的插补运动 DMA 8

例程 3 振镜运动 DMA 10

3. 指令索引

指令 1 GTN_CrdHsOn 12

指令 2 GTN_CrdHsOff 12

指令 3 GTN_ScanHsOn 12

指令 4 GTN_ScanHsOff 13

一、 指令列表

本章表格中右侧的数字为“页码”，其中指令右侧的为“**四、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 GTN_CrdHsOn）均带有超级链接，点击可跳转至指令说明。

表 1 DMA 功能指令列表

指令	说明	页码
GTN_CrdHsOn	打开插补 DMA	12
GTN_CrdHsOff	关闭插补 DMA	12
GTN_ScanHsOn	打开振镜 DMA（运控卡包含激光振镜功能时才可以使用）	12
GTN_ScanHsOff	关闭振镜 DMA	13

二、重点说明

1. DMA 功能为数据段批次压入功能，每次压入的数据段的数量由用户设定，压完数据段之后，再调用 GTN_CrdData（振镜 DMA 功能需要调用 GTN_CrdDataEnd）指令将剩余数据段压入缓冲区。
2. 开启 DMA 功能后，GTN_CrdSpace 和 GTN_ScanCrdSpace 将查询的为 PC 端的剩余数据段空间（插补共有 4096 段空间，振镜共有 1000 段空间）。
3. 插补和振镜同时开启 DMA 功能时，GTN_CrdHsOn 和 GTN_ScanCrdHsOn 中的 link 参数应设为不同值。

三、例程

例程 1 未使用前瞻的插补运动 DMA

```
short commandhandle(char * command,short returnValue)
{
    if(0!=returnValue)
    {
        printf("%s=%d\n",command,returnValue);
    }
    return 0;
}
int main(int argc,char *argv[])
{
    short rtn,coreTemp,crd,fifo;
    coreTemp = 1;crd = 1;fifo = 1;
```

```
short axisNumber;
short seg = 0;
long space;
bool startflag = true;
bool dataFlag = true;
long xPos,yPos;
double synVel,synAcc,velEnd;
xPos = 0;yPos = 0;synVel = 500;synAcc = 1;velEnd = 0;
short link = 1;
rtn = GTN_ZeroPos(coreTemp,1,2);
commandhandle("GTN_ZeroPos",rtn);
for(axisNumber=1;axisNumber<=8;axisNumber++)
{
    rtn = GTN_AlarmOff(coreTemp,axisNumber);
    commandhandle("GTN_AlarmOff",rtn);
    rtn = GTN_LmtsOff(coreTemp,axisNumber);
    commandhandle("GTN_LmtsOff",rtn);
    rtn = GTN_ClrSts(coreTemp,axisNumber);
    commandhandle("GTN_ClrSts",rtn);
    rtn = GTN_AxisOn(coreTemp,axisNumber);
    commandhandle("GTN_AxisOn",rtn);
}
TCrdPrm crdPrm;
memset(&crdPrm, 0, sizeof(crdPrm));
crdPrm.dimension = 2;
crdPrm.synVelMax = 3000;
crdPrm.synAccMax = 40;
crdPrm.evenTime = 50;
crdPrm.profile[0] = 1;
crdPrm.profile[1] = 2;
crdPrm.setOriginFlag = 1;
crdPrm.originPos[0] = 0;
crdPrm.originPos[1] = 0;
rtn = GTN_SetCrdPrm(coreTemp,crd,&crdPrm);
commandhandle("GTN_SetCrdPrm",rtn);
rtn = GTN_CrdClear(coreTemp,crd,fifo);
commandhandle("GTN_CrdClear",rtn);
rtn = GTN_CrdHsOn(coreTemp,crd,fifo,link,100,0);
commandhandle("GTN_CrdHsOn",rtn);
while(1)
{
    rtn = GTN_CrdSpace(coreTemp,crd,&space,fifo);
    commandhandle("GTN_CrdSpace",rtn);
    if((0 != space)&&(dataFlag == true))
    {
        seg = seg+1;
```

```
                xPos = xPos + 2;
                rtn = GTN_LnXY(coreTemp,crd,xPos,yPos,synVel,synAcc,velEnd,fifo);
                commandhandle("GTN_CrdStart",rtn);
                SetDlgItemInt(AfxGetApp()->m_pMainWnd->m_hWnd,IDC_EDIT12,seg,false);
                if( 10000 == seg)
                {
                    do
                    {
                        rtn = GTN_CrdData(coreTemp,crd,NULL,fifo);
                    } while (rtn);
                    dataFlag = false;
                }
            }
            long segment;
            rtn = GTN_GetRemainderSegNum(coreTemp,crd,&segment,fifo);
            commandhandle("GTN_GetPrfPos",rtn);
            if((true == startflag)&&(0 != segment))
            {
                rtn = GTN_CrdStart(coreTemp,1,1);
                commandhandle("GTN_CrdStart",rtn);
                startflag = false;
            }
            double prfPos[2];
            rtn = GTN_GetPrfPos(coreTemp,1,&prfPos,2,0);
            commandhandle("GTN_GetPrfPos",rtn);
            printf("X_Pos = %d      Y_Pos = %d\r\n", prfPos [0], prfPos [1]);
        }
        return 0;
}
```

例程 2 使用新前瞻的插补运动 DMA

```
// 新前瞻初始化
void InitialNewLookAhead(short core,short crd,short fifo,int lookAheadNum)
{
    short rtn;
    EMachineMode machineMode;                   // 机床类型
    EVelSettingDef velDefineMode;               // 速度定义模式
    int axisLimitMode[8];                       // 轴限制模式
    EWorkLimitMode workLimitMode;               // 工件坐标系限制模式
    int axisFollowMode[8];                      // 轴跟随模式
    TLookAheadParameter lookAheadPara;          // 前瞻参数

    machineMode = NORMAL_THREE_AXIS;    // 标准三轴机床
    velDefineMode = NORMAL_DEF_VEL;     // 输入速度为三轴合成速度
    workLimitMode = WORK_LIMIT_VALID;   // 工件坐标系限制生效
```

```
    for (int i=0;i<8;++i)
    {
        axisLimitMode[i] = AXIS_LIMIT_NONE; // 轴限制不生效
        axisFollowMode[i] = 0;              // 非跟随轴
    }
    // 坐标系第4轴为跟随轴并限制轴运动能力
    axisLimitMode[3]=AXIS_LIMIT_MAX_VEL|AXIS_LIMIT_MAX_DV;// 轴最大速度和速度
                                                          最大跳变生效
    axisFollowMode[3]=1;                // 跟随轴

    lookAheadPara.lookAheadNum = lookAheadNum;
    lookAheadPara.time = 1;             // 时间常数
    lookAheadPara.radiusRatio = 50;     // 曲率参数
    for (int i=0;i<8;++i)
    {
        lookAheadPara.vMax[i] = 5000;          // 轴最大速度限制
        lookAheadPara.aMax[i] = 100;           // 轴最大加速度限制
        lookAheadPara.DVMax[i] = 500;          // 轴跳变速度限制
        lookAheadPara.axisRelation[i] = i+1;   // 坐标系轴与前瞻轴一一映射
        lookAheadPara.scale[i] = 1000;         // 脉冲当量
    }

    rtn = GTN_SetupLookAheadCrd(core ,crd,machineMode);   // 设置机床模式
    rtn = GTN_SetAxisLimitModeLa(core ,crd,axisLimitMode);// 设置轴限制模式
    rtn = GTN_SetAxisVelValidModeLa(core ,crd,0xF);       // 设置轴速度有效，按位设置，0xF
                                                          表示前4个轴
    rtn = GTN_InitLookAheadEx(core ,crd,&lookAheadPara,fifo,0); // 设置前瞻参数（需要放在最
                                                                后设置）
}

int _tmain(int argc, _TCHAR* argv[])
{
    short i;
    short rtn;
    short core = 1;
    rtn = GTN_Open();
    rtn = GTN_Reset(core);
    for (i=1;i<5;i++)
    {
        rtn = GTN_AlarmOff(core ,i);
        rtn = GTN_LmtsOff(core ,i);
    }
    rtn = GTN_ClrSts(core ,1,4);
    rtn = GTN_ZeroPos(core ,1,4);
    TCrdPrm crdPrm;
    rtn = GTN_GetCrdPrm(core ,1,&crdPrm);
```

```
    crdPrm.dimension=2;   // 坐标系为二维坐标系
    crdPrm.synVelMax=500; // 最大合成速度：500pulse/ms
    crdPrm.synAccMax=1;   // 最大加速度：1pulse/ms^2
    crdPrm.evenTime = 50;   // 最小匀速时间：50ms
    crdPrm.profile[0] = 1;    // 规划器1对应到X轴
    crdPrm.profile[1] = 2;    // 规划器2对应到Y轴

    rtn = GTN_SetCrdPrm(core ,1, &crdPrm);// 建立1号坐标系，设置坐标系参数
    rtn = GTN_CrdClear(core ,1, 0);             // 清除此缓存区

    rtn = GT_CrdHsOn(1,0,1,200,0);

    double synVel=100.0*(1000/1000),synAcc=10*(1000000/1000);// 单位换算，此处分别为mm/s
                                                              和mm/s^2
    // 前瞻初始化
    InitialNewLookAhead(core ,crd,0,300);

    int count = 1000;
    double x = 0;
    double y = 0;
    for(i=0;i<count;i++)
    {
        x += 0.01;
        y += 0;
        rtn += GTN_LnXYZEx(core ,1,x,y,0,synVel,synAcc,0,0,0);
    }
    // 压入数据并启动运动
    while(1)
    {
        rtn=GTN_CrdDataEx(core ,1,NULL,0);
        if (rtn==0)
        {
            break;
        }
    }
    rtn=GTN_CrdStart(core ,1,0);
    return 0;
}
```

例程 3 振镜运动 DMA

```
short commandhandle(char * command,short returnValue)
{
    if(0!=returnValue)
    {
        printf("%s=%d\n",command,returnValue);
```

```
    }
    return 0;
}
int main(int argc,char *argv[])
{
    short rtn,crd,coreTemp;
    crd = 1;coreTemp=1;
    double jumpAcc,markAcc;
    jumpAcc = 0;markAcc = 0;
    short link = 1;
    bool scanStart = true;
    bool dataFlag = true;
    short space;
    TScanMap map;
    rtn = GTN_GetScanMap(coreTemp,crd,&map);
    commandhandle("GTN_GetScanMap",rtn);
    rtn = GTN_ScanInit(coreTemp,0,jumpAcc,markAcc,crd);
    commandhandle("GTN_ScanInit",rtn);
    rtn = GTN_ScanCrdClear(coreTemp,crd);
    commandhandle("GTN_ScanCrdClear",rtn);
    rtn = GTN_SetScanMode(coreTemp,FIFO_MODE_DYNAMIC,crd);
    commandhandle("GTN_SetScanMode",rtn);
    rtn = GTN_ScanHsOn(coreTemp,crd,link,100);
    commandhandle("GTN_ScanHsOn",rtn);
    while(1)
        {
            rtn = GTN_ScanCrdSpace(coreTemp,&space,crd);
            commandhandle("GTN_ScanCrdSpace",rtn);
            if((0 != space)&&(dataFlag == true))
            {
                segment = segment + 1;
                x = x + 2;
                y = y + 2;
                rtn = GTN_ScanJump(coreTemp,x,y,motionVel,crd);
                commandhandle("GTN_ScanJump",rtn);
                if(10000 == segment)
                {
                    dataFlag = false;
                    do
                    {
                        rtn = GTN_ScanCrdDataEnd(coreTemp,crd);
                    } while (rtn);
                }
            }
        if((scanStart == true)&&(10 == segment))
        {
```

```
                rtn = GTN_ScanCrdStart(coreTemp,crd); //  启动振镜运动
                commandhandle("GTN_ScanCrdStart",rtn);
                scanStart = false;
            }
            short pos[2];
            rtn = GTN_ScanGetCrdPos (coreTemp,&pos[0],crd);
            commandhandle("GTN_ScanGetCrdPos",rtn);
            printf("scan_X_Pos = %d      scan_Y_Pos = %d\r\n",pos[0],pos[1]);
        }
        return 0;
    }
```

四、指令详细说明

指令 1　GTN_CrdHsOn

指令原型	short GTN_CrdHsOn(short core,short crd,short fifo,short link=1,unsigned short threshold=200, short lookAheadInMc=0)		
指令说明	打开插补 DMA		
指令类型	立即指令，调用后立即生效。	**章节页码**	6
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
core	核号，正整数，取值范围[1,2]		
crd	坐标系号，取值范围[1,2]		
fifo	FIFO号，取值范围[0,1]		
link	默认取 1，取值范围[1,4]，如果同时使用振镜 DMA，则振镜 DMA 指令对应的 link 应该为不同于插补 DMA 指令的 link		
threshold	阈值，PC 机指令达到该阈值时自动启动 DMA 发送数据		
lookAheadInMc	默认取 0		

指令 2　GTN_CrdHsOff

指令原型	short GTN_CrdHsOff(short core,short crd,short fifo);		
指令说明	关闭插补 DMA		
指令类型	立即指令，调用后立即生效。	**章节页码**	6
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	核号，正整数，取值范围[1,2]		
crd	坐标系号，取值范围[1,2]		
fifo	FIFO号，取值范围[0,1]		

指令 3　GTN_ScanHsOn

指令原型	short GTN_ScanHsOn(short core,short scan=1,short link=1,unsigned short threshold=200);
指令说明	打开振镜DMA

指令类型	立即指令，调用后立即生效。	**章节页码**	6
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	核号，正整数，取值范围[1,2]		
scan	振镜坐标系号，取值范围[1,6]		
link	默认取 1，取值范围[1,4]，如果同时使用插补 DMA，则插补 DMA 指令对应的 link 应该为不同于振镜 DMA 指令的 link		
threashold	DMA通道每次传输的数据段		

指令 4　GTN_ScanHsOff

指令原型	short GTN_ScanHsOff(short core,short scan);		
指令说明	关闭振镜DMA		
指令类型	立即指令，调用后立即生效。	**章节页码**	6
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	核号，正整数，取值范围[1,2]		
scan	振镜坐标系号，取值范围[1,6]		