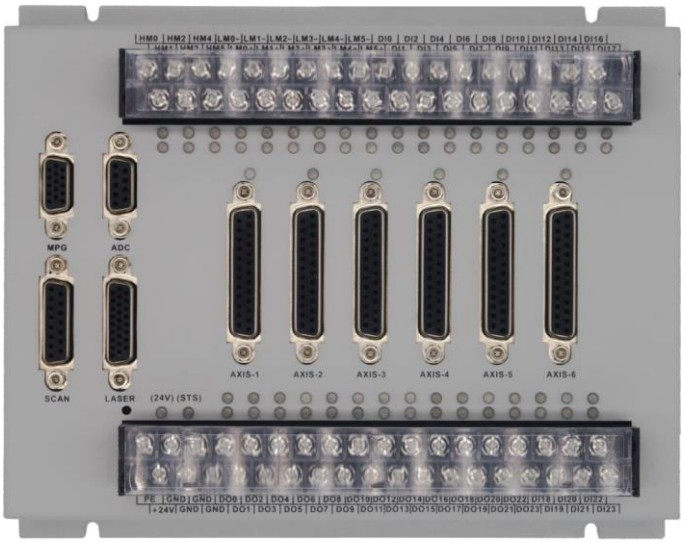
GCN600A-SCAN 振镜编程手册



2023

Version 1.0

目录

版权声明

本手册中的信息资料仅供参考。由于改进设计和功能等原因，高川自动化保留对本资料的最终解释权，内容如有更改，不另行通知。

调试、运动中的机器有危险！用户有责任在机器中设计有效的出错处理和安全保护机制，高川自动化没有义务和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

深圳市高川自动化技术有限公司
电话：0755-23502680
邮箱：sales@gcauto.com.cn
网址：www.gcauto.com.cn

Shenzhen Gaochuan Industrial Automation Co., Ltd.
Tel: 0755-23502680
Email: sales@gcauto.com.cn
Website: www.gcauto.com.cn

文档版本

版本号	修订日期	内容
V1.0	2023 年 10 月 20 日	-

前言

为了给用户提供更快捷，更方便的服务，提高用户的工作效率，本手册主要针对 GCN600A-SCAN 控制器振镜功能的常用指令进行讲解，其他运动指令结合《GC 编程手册》一起使用。

1 振镜控制功能

1.1 功能介绍

激光振镜是用于激光加工领域的特殊的运动器件。它通过控制两个振镜反射激光，在 XY 平面上实现雕刻、切割等工艺。激光振镜由于负载非常小(只有两个小的反射镜片)，系统的响应非常快，能实现非常高速的加工。其工作原理如下图 1.1 所示:

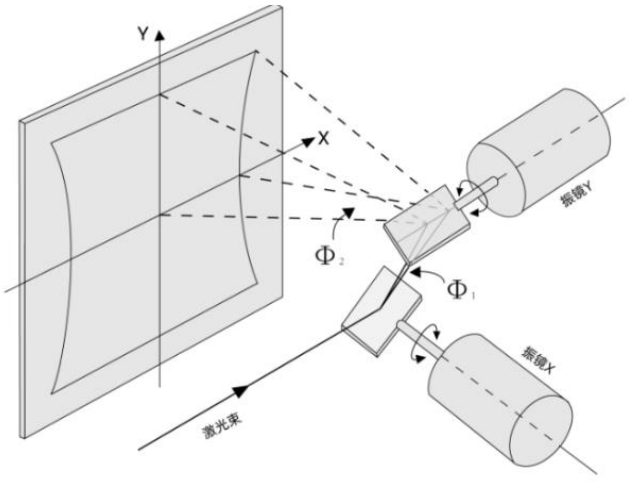


图 1.1 振镜工作原理示意图

高川振镜控制器，通过 XY2-100 数字协议实现对振镜的控制，支持 XYZ 三个方向的控制，支持连续的缓存区高速执行，结合运动控制和激光控制功能，适用于常见的激光振镜控制应用场合。

1.2 指令汇总

函数原形	函数说明
OEM_ScanEn	使能振镜控制功能
OEM_ScanBufClr	清除振镜执行缓存区
OEM_ScanBufMoveAbs	振镜缓存区运动控制指令 XY
OEM_ScanBufMoveAbsEx	振镜缓存区运动控制指令 XYZ
OEM_ScanBufSetSegNo	设置缓存区段号
OEM_ScanBufDelay	设置缓存区延时
OEM_ScanBufLaserOnOff	设置缓存区激光开关指令
OEM_ScanBufLaserSetOnOffDelay	设置缓存区激光开关延时
OEM_ScanBufLaserSetPowerPwm	设置缓存区 PWM 输出能量

OEM_ScanBufSetDOBit	设置缓存区按位 DO
OEM_ScanBufSetDO	设置缓存区 DO
OEM_ScanBufWaitDI	缓存区等待 DI
OEM_ScanEndList	结束缓存区指令压入
OEM_ScanOnOff	启动和停止缓存区指令执行
OEM_ScanGetSts	查询缓存区指令的执行状态
OEM_ScanMoveAbs	振镜运动控制(立即指令)
OEM_ScanLaserOnOff	激光开关 1
OEM_ScanLaserPowerPwm	激光开关 2
OEM_ScanLaserPowerDac	DAC 控制开关

1.3 指令说明

(1)使能振镜控制功能

OEM_ScanEn(HAND devHandle, short onOff,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
OnOff	short	输入	功能开启关闭控制，0：关闭，1：开启
chn	short	输入	振镜控制通道号,取值范围[0,n]

注意：控制器默认振镜功能是关闭的，NMC_DevReset 不会改变振镜的开关状态；

使用振镜功能前，必须先开启该功能；

(2)清除振镜执行缓存区

OEM_ScanBufClr(HAND devHandle, short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chn	short	输入	振镜控制通道号,取值范围[0,n]

注意：在执行振镜缓存区时，必须先调用该指令初始化振镜缓存区；

(3)振镜缓存区运动控制指令 XY（振镜缓存区控制指令支持振镜控制、延时、激光开关等）

OEM_ScanBufMoveAbs(HAND devHandle, short *pXPosArray,short *pYPosArray,short cnt,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pXPosArray	short*	输入	X 方向位置列表
pYPosArray	short*	输入	Y 方向位置列表
cnt	short	输入	位置点数量
chn	short	输入	振镜控制通道号,取值范围[0,n]

(4)振镜缓存区运动控制指令 XYZ

OEM_ScanBufMoveAbsEx(HAND devHandle, short *pXPosArray,short *pYPosArray,short *pZPosArray,short cnt,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pXPosArray	short*	输入	X 方向位置列表
pYPosArray	short*	输入	Y 方向位置列表
pZPosArray	short*	输入	Z 方向位置列表
cnt	short	输入	位置点数量
chn	short	输入	振镜控制通道号,取值范围[0,n]

(5)设置缓存区段号（设置段号后，后续的指令执行段号将自动增加）

OEM_ScanBufSetSegNo(HAND devHandle, long segNo,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
segNo	Int32	输入	段号[1,n]（用户自定义）
chn	short	输入	振镜控制通道号,取值范围[0,n]

(6)设置缓存区延时

OEM_ScanBufDelay(HAND devHandle, long delayUs,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
delayUs	Int32	输入	延时时间，单位微秒
chn	short	输入	振镜控制通道号,取值范围[0,n]

(7)设置缓存区激光开关指令

OEM_ScanBufLaserOnOff(HAND devHandle, short laserOnOff,short pwmChn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
laserOnOff	short	输入	激光开关，0:关闭，1:打开
pwmChn	short	输入	PWM 通道号,取值范围[0,n]

(8)设置缓存区激光开关延时

OEM_ScanBufLaserSetOnOffDelay(HAND devHandle, double onDelay,double offDelay,short pwmChn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
onDelay	short	输入	开光延时，单位 us,取值范围[0,65535]
offDelay	short	输入	关光延时，单位 us,取值范围[0,65535]
pwmChn	short	输入	PWM 通道号,取值范围[0,n]

(9)设置缓存区 PWM 输出能量

OEM_ScanBufLaserSetPowerPwm(HAND devHandle,double pulseWid,double onTime,short pwmChn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pulseWid	short	输入	PWM 宽度，单位 us,取值范围[0.25,16000]
onTime	short	输入	on 时间，单位 us,,取值范围[0.25,16000]
pwmChn	short	输入	PWM 通道号,取值范围[0,n]

(10)设置缓存区模拟量输出

OEM_ScanBufLaserSetPowerDac(HAND devHandle,short dac,short dacChn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
dac	short	输入	模拟量输出值，取值范围[-32768,32767],对应 DAC 输出范围
dacChn	short	输入	扩展模拟量通道,取值范围[0,1]

(11)设置缓存区按位 DO

OEM_ScanBufSetDOBit(HAND devHandle,short gpoIdx,short outSns,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
gpoIdx	short	输入	gpo 的序号，取值范围[0,31]
outSns	short	输入	输出电平 0:低电平 1:高电平
chn	short	输入	振镜控制通道号,取值范围[0,1]

(12)设置缓存区 DO

OEM_ScanBufSetDO(HAND devHandle,long gpoValue,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
gpoValue	short	输入	gpo 输出值
chn	short	输入	振镜控制通道号,取值范围[0,1]

(13)缓存区等待 DI

OEM_ScanBufWaitDI(HAND devHandle,short diType,short diGroup,short diIdxMask,short diValueMask,unsigned short timeOutMs,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
diType	short	输入	保留，请输入 DI_TYPE_GPI
diGroup	short	输入	保留，请输入 0

diIdxMask	short	输入	输入序号的掩码
diValueMask	short	输入	输入电平的掩码
timeOutMs	short	输入	超时时间，单位:ms，输入 0 表示无限等待
chn	short	输入	振镜控制通道号，取值范围[0,1]

(13)结束缓存区指令压入

OEM_ScanEndList(HAND devHandle,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chn	short	输入	振镜控制通道号，取值范围[0,1]

(14)启动和停止缓存区指令执行

OEM_ScanOnOff(HAND devHandle,short onOff,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
onOff	short	输入	振镜缓存区执行的启动及停止，0：停止，1:启
chn	short	输入	振镜控制通道号，取值范围[0,1]

(15)查询缓存区指令的执行状态

OEM_ScanGetSts(HAND devHandle,TOemScanSts *pSts,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pSts	TOemScanSts*	输出	振镜缓存区执行状态，定义如下面表格
chn	short	输入	振镜控制通道号，取值范围[0,1]

```
//状态位定义
#define BIT_SCAN_EN                (0x00000001)    // 使能标志位
#define BIT_SCAN_RUN               (0x00000002)    // 运动:1 ,静止 0
#define BIT_SCAN_END               (0x00000004)    // 是否End标志位
#define BIT_SCAN_BUF_FULL          (0x00000008)    // 数据满
```

```
#define BIT_SCAN_BUF_EMPTY          (0x00000010)    // 数据空
#define BIT_SCAN_ERR_PUSH           (0x00001000)    // 压数据错误
#define BIT_SCAN_ERR_POP            (0x00002000)    // 取数据错误
#define BIT_SCAN_ERR_OTHER          (0x00004000)    // 其他错误，参考errCode
typedef struct
{
    long sts;                   // 运行状态，按位表示，如上
    long userSeg;               // 用户段号
    long leftSpace;             // 剩余空间
    long usedSpace;             // 使用空间
    unsigned long cmdAllCnt;    // 全部指令的计数
    short xPos;                 // x方向当前位置
    short yPos;                 // y方向当前位置
    short errCode;              // 错误码
    short reserved[3];          // 保留
}TOemScanSts;
```

(16)振镜运动控制(立即指令)

OEM_ScanMoveAbs(HAND devHandle,short x,short y,short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
x	short	输入	X 目标位置
y	short	输入	Y 目标位置
chn	short	输入	振镜控制通道号,取值范围[0,1]

(17)激光开关 1

OEM_ScanLaserOnOff(HAND devHandle,short onOff,short pwmChn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄

onOff	short	输入	开关
chn	short	输入	振镜控制通道号,取值范围[0,1]

(18)激光开关 2

OEM_ScanLaserPowerPwm(HAND devHandle,double pulseWid,double onTime,short pwmChn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pulseWid	double	输入	PWM 宽度，单位 us,取值范围[0.5,16000]
onTime	double	输入	on 时间，单位 us
chn	short	输入	PWM 通道,取值范围[0,1]

(19)DAC 控制开关

OEM_ScanLaserPowerDac(HAND devHandle,short dac,short dacChn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
dac	short	输入	模拟量输出值
chn	short	输入	扩展模拟量通道,取值范围[0,1]

1.4 综合示例

```
//振镜控制例子
short ScanTest_Simple(HAND hDev)
{
    short rtn;
    short chn = 0;
    short xPos[1024];
    short yPos[1024];
    // 1.BufClr
    printf("\n 简单测试开始.... \n");
    rtn = OEM_ScanBufClr(hDev, chn);
```

```
if (rtn != 0) { printf("OEM_ScanBufClr error = %d", rtn); return 1; }
// 2.压入指令
rtn = OEM_ScanBufLaserOnOff(hDev, 0, chn);
if (rtn != 0) { printf("OEM_ScanBufLaserOnOff error = %d", rtn); return 1; }
rtn = OEM_ScanBufDelay(hDev, 100, chn);
if (rtn != 0) { printf("OEM_ScanBufDelay error = %d", rtn); return 1; }
rtn = OEM_ScanBufLaserSetOnOffDelay(hDev, 0, 0, chn);
if (rtn != 0) { printf("OEM_ScanBufLaserSetOnOffDelay error = %d", rtn);
return 1; }
rtn = OEM_ScanBufLaserSetPowerPwm(hDev, 2000, 1000, chn);
if (rtn != 0) { printf("OEM_ScanBufLaserSetPowerPwm error = %d", rtn);
return 1; }
rtn = OEM_ScanBufLaserOnOff(hDev, 1, chn);
if (rtn != 0) { printf("OEM_ScanBufLaserOnOff error = %d", rtn); return 1; }
for (int k = 0; k<20; k++)
{
    for (int i = 0; i<1024; i++)
    {
        xPos[i] = (i + 1) * 2;
        yPos[i] = (i + 1) * 3;
    }
    rtn = OEM_ScanBufMoveAbs(hDev, xPos, yPos, 1024, 0);
    if (rtn != 0) { printf("OEM_ScanBufMoveAbs error = %d", rtn);
    return 1; }

    for (int i = 0; i<1024; i++)
    {
        xPos[i] = (1023 - i + 1) * 2;
        yPos[i] = (1023 - i + 1) * 3;
```

```
        }
        rtn = OEM_ScanBufMoveAbs(hDev, xPos, yPos, 1024, 0);
        if (rtn != 0) { printf("OEM_ScanBufMoveAbs error = %d", rtn);
        return 1; }
    }
    rtn = OEM_ScanBufLaserOnOff(hDev, 0, chn);
    if (rtn != 0) { printf("OEM_ScanBufLaserOnOff error = %d", rtn); return 1; }
    // 3.结束指令压入
    rtn = OEM_ScanEndList(hDev, chn);
    if (rtn != 0) { printf("OEM_ScanEndList error = %d", rtn); return 1; }
    // 4.启动 Scan 缓冲指令
    rtn = OEM_ScanOnOff(hDev, 1, chn);
    if (rtn != 0) { printf("OEM_ScanOnOff error = %d", rtn); return 1; }
    // 5.状态检查
    TOemScanSts sts;
    while (1)
    {
        OEM_ScanGetSts(hDev, &sts, chn);
        if ((sts.sts & BIT_SCAN_RUN) == 0)
        {
            break;
        }
        printf("当前位置 x=%d,y=%d \r", sts.xPos, sts.yPos);
    }
    printf("\n 简单测试完成.... \n\n");
    return 0;
}
```