

USB 扩展模块用户手册



2024

Version 1.01

目录

目录	2
版权声明.....	3
联系我们.....	3
文档版本.....	3
前言	5
1 产品概述.....	6
1.1 IO64M01(数字量).....	6
1.1.1 尺寸图.....	6
1.1.2 电气规格.....	6
1.2 IO32M01(数字量).....	7
1.2.1 尺寸图.....	7
1.2.2 电气规格.....	8
1.3 IO24M01(模拟量).....	9
1.3.1 尺寸图.....	9
1.3.2 电气规格.....	9
2 配线指导.....	12
2.1 挂接 GCN 控制器.....	12
2.2 独立使用.....	12
3 模块测试.....	14
3.1 挂接 GCN 控制器.....	14
3.2 独立使用.....	14
4 软件编程.....	16
4.1 挂接 GCN 控制器指令.....	16
4.2 独立使用指令.....	20
4.3 应用案例.....	25
4.3.1 挂接 GCN 控制器例程.....	25
4.3.2 独立使用例程(C#).....	26

版权声明

本手册版权归深圳市高川自动化技术有限公司所有, 未经本公司书面许可, 任何人不得翻印、翻译和抄袭本手册中的任何内容。

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

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 1 月 29 日	-
V1.01	2024 年 8 月 20 日	增加部分指令说明，图和其他

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 USB 扩展模块硬件使用上的讲解，包括控制器的产品概述，配线指导，模块测试和软件编程，方便用户更好的使用我们的产品。

1 产品概述

USBIO64M01, USBIO32M01 和 USBIO24M01 模块提供一种低成本、简单易用、功能可靠的 IO 扩展方式，通过 RS485 或 USB2.0 接口与外部连接，可搭配我司的 GCN 系列控制器或者独立使用，无需安装驱动。

1.1 IO64M01 (数字量)

1.1.1 尺寸图

如图 1.1.1 为 IO64M01 模块的外观及尺寸：

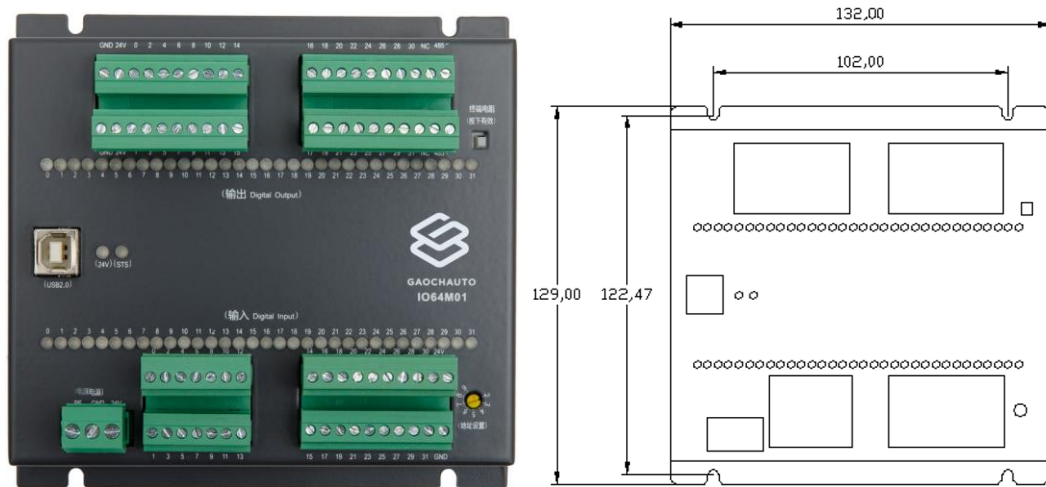


图 1.1.1 IO64M01 外观及尺寸

1.1.2 电气规格

数字量输入	
通道	32
输入类型	NPN 型/低电平有效
输入阻抗	5K Ω
过压保护	70 VDC
ESD	2000 VDC
输入电流	3.5mA@0 VDC
输入电压	Logic 0: 4Vmax. Logic 1: 5V min(50V max)
数字量输出	
通道	32

输出类型	Sink (NPN), 短路保护
输出电压	Logic 0: 0.5Vmax. Logic 1: 开路(50V max)
Sink 电流	100mA
通讯	
通讯接口	USB (独立使用时通讯)、RS485 (只与 GCN 控制器通讯)
常规规格	
尺寸	130.5X125mm
系统供电	24VDC +/- 20%
湿度	5 ~ 95% RH, non-condensing (IEC 68-2-3)
工作温度	0 ~ 60° C (32 ~ 140° F)
存储温度	-20 ~ 85° C (-4 ~ 185° F)
软件支持	
操作系统	Windows XP/7/8/10/11 Linux WinCE
软件兼容性	VB/VC++/BCB/C#
演示软件	GCS.exe

1.2 I032M01 (数字量)

1.2.1 尺寸图

如图 1.2.1 为 I032M01 模块的外观及尺寸:

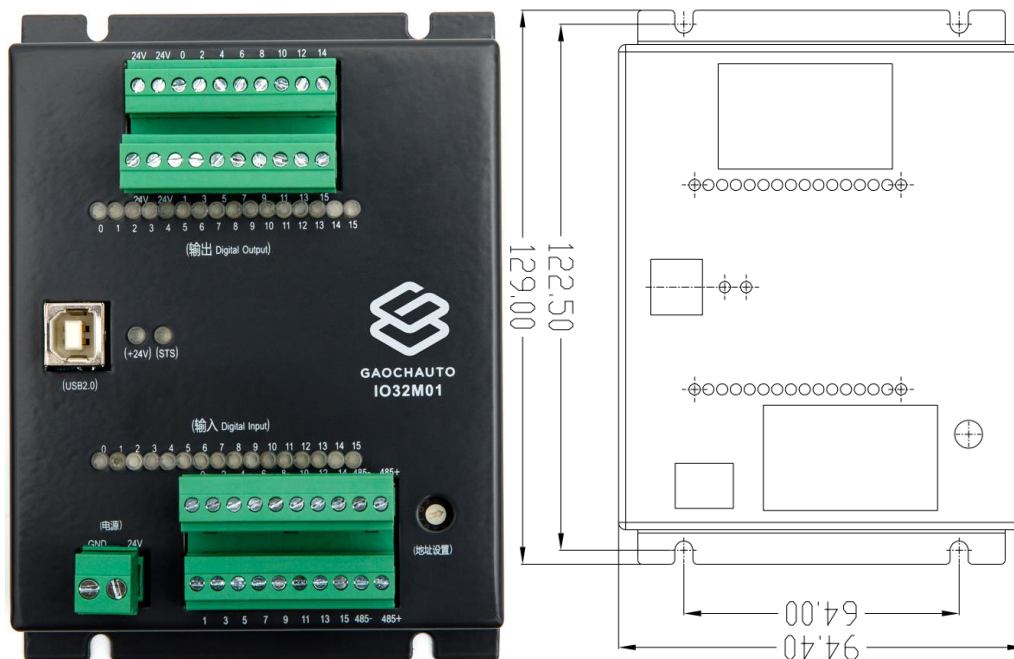


图 1.2.1 IO32M01 外观及尺寸

1.2.2 电气规格

数字量输入	
通道	16
输入类型	NPN 型/低电平有效
输入阻抗	5K Ω
过压保护	70 VDC
ESD	2000 VDC
输入电流	3.5mA@0 VDC
输入电压	Logic 0: 4Vmax. Logic 1: 5V min(50V max)
数字量输出	
通道	16
输出类型	Sink (NPN), 短路保护
输出电压	Logic 0: 0.5Vmax. Logic 1: 开路(50V max)
Sink 电流	100mA
通讯	
通讯接口	USB (独立使用时通讯)、RS485 (只与 GCN 控制器通讯)

常规规格	
尺寸	94X123mm
系统供电	24VDC +/- 20%
湿度	5 ~ 95% RH, non-condensing (IEC 68-2-3)
工作温度	0 ~ 60° C (32 ~ 140° F)
存储温度	-20 ~ 85° C (-4 ~ 185° F)
软件支持	
操作系统	Windows XP/7/8/10/11 Linux WinCE
软件兼容性	VB/VC++/BCB/C#
演示软件	GCS. EXE

1.3 IO24M01 (模拟量)

1.3.1 尺寸图

如图 1.3.1 为 IO24M01 模块的外观及尺寸：

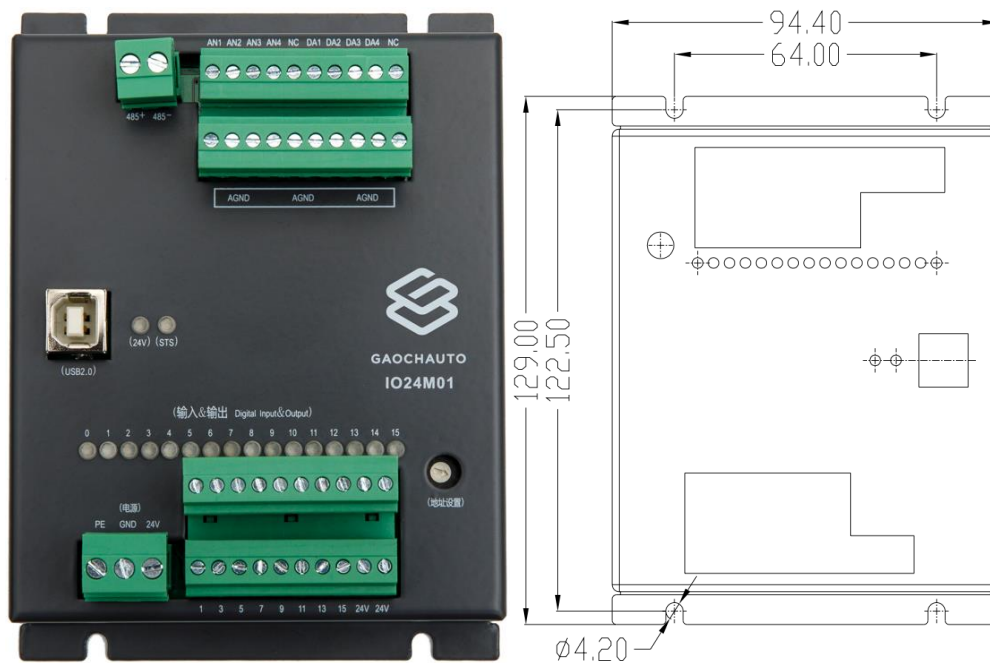


图 1.3.1 IO24M01 外观及尺寸

1.3.2 电气规格

数字量输入

通道	16 路(输入&输出) 共用
输入类型	NPN 型/低电平有效
输入阻抗	5K Ω
过压保护	70 VDC
ESD	2000 VDC
输入电流	3.5mA@0 VDC
输入电压	Logic 0: 4Vmax. Logic 1: 5V min(50V max)
数字量输出	
通道	16 路(输入&输出) 共用
输出类型	Sink (NPN), 短路保护
输出电压	Logic 0: 0.5Vmax. Logic 1: 开路(50V max)
Sink 电流	100mA
模拟量输入	
通道	4
输入阻抗	5K Ω
过压保护	70 VDC
ESD	2000 VDC
输入电流	3.5mA@0 VDC
输入电压范围	-10V ~ +10V
精度	12 位精度
模拟量输出	
通道	4
输出电压范围	0 - 10V
精度	12 位精度
通讯	
通讯接口	USB (独立使用时通讯)、RS485 (只与 GCN 控制器通讯)
常规规格	
尺寸	130.5X125mm

系统供电	24VDC +/- 20%
湿度	5 ~ 95% RH, non-condensing (IEC 68-2-3)
工作温度	0 ~ 60° C (32 ~ 140° F)
存储温度	-20 ~ 85° C (-4 ~ 185° F)
软件支持	
操作系统	Windows XP/7/8/10/11 Linux WinCE
软件兼容性	VB/VC++/BCB/C#
演示软件	GCS. EXE

2 配线指导

I064M01, I032M01和I024M01在配线和测试使用中是相同的，下面对模块作统一说明。

2.1 挂接GCN控制器

接线示意图如图 2.1.1，黄色连接线为 RS485 通讯线：



图 2.1.1 IO 模块接线示意图

输入和输出部分请参考图 2.1.2 接线；

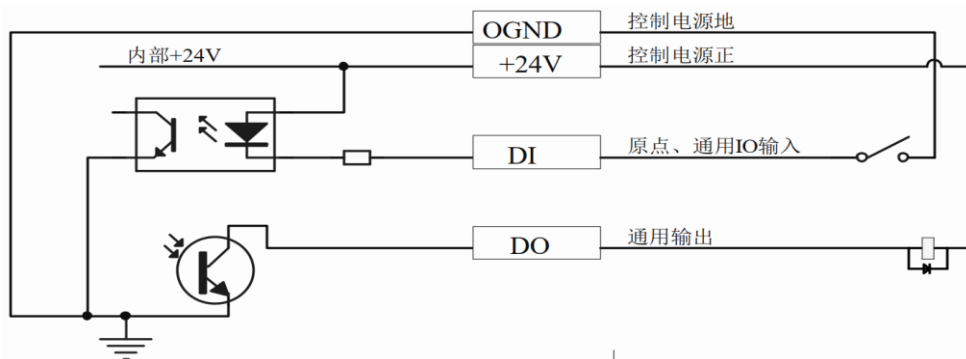


图 2.1.2 输入和输出配线示意图



- (1) 当扩展模块与控制器配合使用时，模块上的拨码地址从 2 开始，一个控制器最多挂接 8 个扩展模块；
- (2) 当模块独立使用时，拨码地址无效，一根 USB 线与模块独立通信，模块之间用 RS485 连接无效；

2.2 独立使用

接线示意图如图 2.2.1，黄色连接线为 USB 通讯线：



图 2.2.1 IO 模块独立使用连接图

3 模块测试

3.1 挂接GCN控制器

- (1) 根据图 2.1.1 正确连接后上电；
- (2) 打开 GCS 工具，在菜单栏“功能”->“IO 相关”->通用 IO 扩展模块，如图 3.1.1；
- (3) 点击“使能”，保持常绿时，表示通讯成功；
- (4) 模块类型匹配型号：I024M01->I024_*；I032M01->I032_*；I064M01->I064_*；
- (5) 扩展 IO 地址：匹配模块上的旋转拨码，范围 2~9；

旋转拨码	IO 起始位	IO 结束位	说明
2	64	95	注：每增加一个模块，则多增加 32； 如拨码为 2 的模块输入/输出第 1 位索引为 64； 软件编写请移步到 4.1；
3	96	127	
4	128	159	
5	160	191	
.....	



图 3.1.1 扩展模块测试界面

3.2 独立使用

- (1) 根据图 2.2.1 正确连接上电，**无需安装驱动**；
- (2) 拨码地址无效；
- (3) 从提供的资料包中，打开“独立使用(测试工具)”工具；测试界面如图 3.2.1；

NIO_MODULE_test

×

输出

	端口:	数量:	值:		位置:	值:	
	<input type="text" value="0"/>	<input type="text" value="1"/>	<input type="text"/>	<input type="button" value="读"/>	<input type="text" value="0"/>	<input type="text"/>	<input type="button" value="读"/>
<input type="checkbox"/> ALL	<input type="text" value="0"/>	<input type="text" value="1"/>	<input type="text" value="FFFF"/>	<input type="button" value="写"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="button" value="写"/>

0C 01 02 03 04 0E 0E 07 0E 09 1C 11 12 13 14 1E

输入

	端口:	数量:	值:		位置:	值:	
	<input type="text" value="0"/>	<input type="text" value="1"/>	<input type="text"/>	<input type="button" value="读"/>	<input type="text" value="0"/>	<input type="text"/>	<input type="button" value="读"/>

0C 01 02 03 04 0E 0E 07 0E 09 1C 11 12 13 14 1E

AD: V V V V

DA: V V V V

日期/时间

年: 月: 日: 时: 分: 秒: 星期:

设备ID

产品唯一ID

用户数据 (字符串):

IO逻辑 设备ID:

图 3.2.1 IO 独立使用测试界面

4 软件编程

4.1 挂接GCN控制器指令

函数原形	函数说明
NMC_SetDOGroup	设置扩展 I/O 输出 (按通道, 支持超过 32 位), 带默认 group
NMC_GetDOGroup	读取扩展 I/O 输出 (按通道, 支持超过 32 位), 带默认 group
NMC_SetDOBit	按位设置扩展 I/O 输出
NMC_GetDOBit	按位读取扩展 I/O 输出
NMC_GetDIGroup	读扩展 I/O 输入 (按通道, 支持超过 32 位), 带默认 group
NMC_GetDIBit	按位读取扩展 I/O 输入
NMC_SetDIBitRevs	按位设置扩展 I/O 输入信号取反 (不会改变灯的状态)
NMC_GetDIBitRevs	按位读取扩展 I/O 输入信号取反的设置值
NMC_SetDOBitRevs	按位设置扩展 I/O 输出信号取反 (会改变灯的状态)
NMC_GetDOBitRevs	按位读取扩展 I/O 输出信号取反的设置值
NMC_GetIOModuleSts	读取扩展 I/O 模块的状态
NMC_IOModuleSetEn	设置扩展 I/O 模块有效 (带模块类型)
NMC_IOModuleGetType	读取扩展 I/O 模块类型

(1) 设置扩展 I/O 输出 (按通道, 支持超过 32 位), 与 NMC_SetDO 功能相同

`NMC_SetDOGroup(HAND devHandle, long value, short groupID);`

参数	输入/输出	描述
<code>devHandle</code>	输入	控制器句柄
<code>groupID</code>	输入	D0 组, 取值范围 [0, n], 0: 本地 D00~D031, 1: 本地 D032~D063。其他指扩展 I/O 模块, 无拨码: 第 1 个默认 groupID=2, 以此类推; 有拨码: 根据拨码设置; (2 ≤ groupID ≤ 9) 第 1 个模块开始位为 64, 每增加 1 个模块, 则多增加 32 位;
<code>value</code>	输入	默认情况下, 1 表示高电平, 0 表示低电平

(2) 按位设置扩展 I/O 输出

`NMC_SetDOBit(HAND devHandle, short bitIndex, unsigned char value);`

参数	输入/输出	描述
<code>devHandle</code>	输入	控制器句柄
<code>bitIndex</code>	输入	取值范围[0,n],位序号,前 64 位为本地通用输出。其他指扩展 IO 模块,第 1 个模块开始位为 64,每多 1 个模块,则多增加 32 位;
<code>value</code>	输入	默认情况下,1 表示高电平,0 表示低电平

(3) 按位读取扩展 IO 输出

`NMC_GetDOBit(HAND devHandle, short bitIndex, short *bitValue);`

名称	输入/输出	描述
<code>devHandle</code>	输入	控制器句柄
<code>bitIndex</code>	输入	取值范围[0,n],位序号,前 64 位为本地通用输出。其他指扩展 IO 模块,第 1 个模块开始位为 64,每多 1 个模块,则多增加 32 位;
<code>bitValue</code>	输出	返回输出电平,默认情况下,1 表示高电平,0 表示低电平;

(4) 读取扩展 IO 输出(按通道,支持超过 32 位),此函数与 NMC_GetDO 功能相同

`NMC_GetDOGroup(HAND devHandle, long *pDoValue, short groupID);`

参数	输入/输出	描述
<code>devHandle</code>	输入	控制器句柄
<code>groupID</code>	输入	DO 组,取值范围 [0,n],0:本地 D00~D031, 1:本地 D032~D063。其他指扩展 IO 模块,无拨码:第 1 个默认 groupID=2,以此类推;有拨码:根据拨码设置;(2≤groupID≤9)第 1 个模块开始位为 64,每增加 1 个模块,则多增加 32 位;
<code>value</code>	输出	按组返回输出电平的状态 默认情况下 1 表示高电平,0 表示低电平

(5) 读取扩展 IO 输入(按通道,支持超过 32 位),此函数与 NMC_GetDI 功能相同

`NMC_GetDIGroup(HAND devHandle, long *pInValue, short groupID);`

参数	输入/输出	描述
<code>devHandle</code>	输入	控制器句柄

groupID	输入	DI 组，取值范围 [0, n], 0: 本地 DOI~DI31, 1: 本地 DI32~DI63。其他指扩展 IO 模块，无拨码：第 1 个默认 groupID=2，以此类推；有拨码：根据拨码设置：(2 ≤ groupID ≤ 9) 第 1 个模块开始位为 64，每增加 1 个模块，则多增加 32 位；
pInValue	输出	按位返回输入电平的状态 1: 表示高电平/开路 0: 表示低电平/闭合

(6) 按位读取扩展 IO 输入

NMC_GetDIBit(HAND devHandle, short bitIndex, short *bitValue);

参数	输入/输出	描述
devHandle	输入	控制器句柄
bitIndex	输入	取值范围 [0, n], 位序号, 前 64 位为本地通用输入。其他指扩展 IO 模块，第 1 个模块开始位为 64，每多 1 个模块，则多增加 32 位；
bitValue	输出	按位返回电平输入的状态 1: 表示高电平/开路 0: 表示低电平/闭合

(7) 按位设置扩展 IO 输入信号取反（不会改变灯的状态）

NMC_SetDIBitRevs (HAND devHandle, short bitIndex, short revs);

参数	输入/输出	描述
devHandle	输入	控制器句柄
bitIndex	输入	取值范围 [0, n], 位序号, 前 64 位为本地通用输入。其他指扩展 IO 模块，第 1 个模块开始位为 64，每多 1 个模块，则多增加 32 位；
revs	输入	是否取反 1: 取反, 0: 不取反

(8) 按位读取扩展 IO 输入信号取反的设置值

NMC_GetDIBitRevs (HAND devHandle, short bitIndex, short *pRevs);

参数	输入/输出	描述
devHandle	输入	控制器句柄
bitIndex	输入	取值范围 [0, n], 位序号, 前 64 位为本地通用输入。其他指扩展 IO 模块，第 1 个模块开始位为 64，每多 1 个模块，则多增加 32 位；
pRevs	输出	是否取反 1: 取反, 0: 不取反

(9) 按位设置扩展 I0 输出信号取反 (会改变灯的状态)

```
NMC_SetDOBitRevs( HAND devHandle, short bitIndex, short revs);
```

参数	输入/输出	描述
devHandle	输入	控制器句柄
bitIndex	输入	取值范围[0, n], 位序号, 前 64 位为本地通用输出。其他指扩展 I0 模块, 第 1 个模块开始位为 64, 每多 1 个模块, 则多增加 32 位;
revs	输入	是否取反 1: 取反, 0: 不取反

(10) 按位读取扩展 I0 输出信号取反的设置值

```
NMC_GetDOBitRevs( HAND devHandle, short bitIndex, short *pRevs);
```

参数	输入/输出	描述
devHandle	输入	控制器句柄
bitIndex	输入	取值范围[0, n]位序号, 前 64 位为本地的 I0 输出, 大于 64 为扩展 I0 输出
pRevs	输出	是否取反 1: 取反, 0: 不取反

(11) 读取扩展 I0 模块的状态

```
NMC_GetIOModuleSts(HAND devHandle, unsigned long *sts);
```

参数	输入/输出	描述
devHandle	输入	控制器句柄
sts	输出	在线: 比特位为 1; 离线: 比特位为 0(最大支持 32 位比特位)

(12) 设置扩展 I0 模块有效(带模块类型)

```
NMC_IOModuleSetEn( HAND devHandle, unsigned char chDevId, short chDevType);
```

参数	输入/输出	描述
devHandle	输入	控制器句柄
chDevId	输入	设备 ID
chDevType	输入	模块类型, 见宏定义 // 扩展模块类型定义 #define IOMODULE_TYPE_I064 1// 32DI32DO 模块 (包括 16DI16DO 模块) #define IOMODULE_TYPE_I032_DA 2// 4AD4DA 模块

(13) 读取扩展 IO 模块类型

```
NMC_IOModuleGetType( HAND devHandle, unsigned char chDevId, short *pChDevType);
```

参数	输入/输出	描述
devHandle	输入	控制器句柄
chDevId	输入	设备 ID
pChDevType	输出	返回的模块类型，见宏定义 // 扩展模块类型定义 #define IOMODULE_TYPE_IO64 1// 32DI32DO 模块（包括 16DI16DO 模块） #define IOMODULE_TYPE_IO32_DA 2 // 4AD4DA 模块

4.2 独立使用指令

当需要独立使用扩展模块时，请联系我司技术人员提供对应的库文件和例程，下面罗列出基本使用函数。

函数原形	函数说明
NIO_Search	模块搜寻
NIO_OpenByID	模块搜寻
NIO_Close	模块关闭
NIO_GetDO	获取设备点位输出状态
NIO_SetDO	设置设备点位输出状态
NIO_GetDOBit	获取设备单个点位输入状态
NIO_SetDOBit	设置设备单个点位输出状态
NIO_GetDI	获取设备点位输入状态
NIO_GetDIBit	获取设备单个点位输入状态
NIO_GetADC	获取设备模拟量输入
NIO_SetDAC	设置设备模拟量输出
NIO_WriteID	修改 SIO 设备在系统中名称
NIO_ReadID	读取 SIO 设备名称
NIO_WriteDevID	修改 SIO 设备 ID 序号（用于 Modbus 通讯的设备 ID）
NIO_ReadDevID	读取 SIO 设备 ID 序号（用于 Modbus 通讯的设备 ID）

NIO_GetIOLogic	获取设备 IO 口逻辑
NIO_SetIOLogic	设置设备 IO 口逻辑
NIO_GetError	SC 设备操作过程中，最后出现的错误，以字符串的形式提供

(1) 模块搜寻

[NIO_Search\(unsigned short *pDevNo, TDevInfo *pInfoList\);](#)

参数	输入/输出	描述
pDevNo	输出	搜索到的模块数量
pInfoList	输出	模块信息，结构体请参考头文件

(2) 模块打开

[NIO_OpenByID\(const char *idStr, PHAND pDevHandle\);](#)

参数	输入/输出	描述
idStr	输入	模块 ID 字符串，通过搜索模块函数获得的结构体 TDevInfo 中获得
pDevHandle	输出	模块句柄

(3) 模块关闭

[NIO_Close\(HAND devHandle\);](#)

参数	输入/输出	描述
devHandle	输入	模块句柄

(4) 获取设备点位输出状态

[NIO_GetDO\(HAND devHandle, unsigned short portStart, unsigned short portCount, unsigned short *value\);](#)

参数	输入/输出	描述
devHandle	输入	模块句柄
portStart	输入	起始端口号，取值范围(0-1)
portCount	输入	要获取的端口数量，取值范围(1-2)
value	输出	用于存储端口状态值缓冲(每个端口 16 个点位)，每个点位:1 代表触发; 0 代表未触发

(5) 设置设备点位输出状态

[NIO_SetDO\(HAND devHandle, unsigned short portStart, unsigned short portCount, unsigned short](#)

*value);

参数	输入/输出	描述
devHandle	输入	模块句柄
portStart	输入	起始端口号, 取值范围(0-1)
portCount	输入	要设置的端口数量, 取值范围(1-2)
value	输出	要设置的端口状态值缓冲(每个端口 16 个点位), 每个点位:1 代表触发; 0 代表不触发

(6) 获取设备单个点位输入状态

NIO_GetDOBit(HAND devHandle, unsigned short channel, unsigned short *value);

参数	输入/输出	描述
devHandle	输入	模块句柄
channel	输入	点位位置, 取值范围: 0-31
value	输出	用于存储获取的点位状态, 1 代表触发; 0 代表未触发

(7) 设置设备单个点位输出状态

NIO_SetDOBit(HAND devHandle, unsigned short channel, unsigned short value);

参数	输入/输出	描述
devHandle	输入	模块句柄
channel	输入	点位位置, 取值范围: 0-31
value	输出	用于存储获取的点位状态, 1 代表触发; 0 代表未触发

(8) 获取设备点位输入状态

NIO_GetDI(HAND devHandle, unsigned short portStart, unsigned short portCount, unsigned short *value);

参数	输入/输出	描述
devHandle	输入	模块句柄
portStart	输入	起始端口号, 取值范围(0-1)
portCount	输入	要获取的端口数量, 取值范围(1-2)
value	输出	用于存储端口状态值缓冲(每个端口 16 个点位), 每个点位:1 代表触发; 0 代表未触发

(9) 获取设备单个点位输入状态

NIO_GetDIBit(HAND devHandle, unsigned short channel, unsigned short *value);

参数	输入/输出	描述
devHandle	输入	模块句柄
channel	输入	点位位置, 取值范围: 0-31
value	输出	用于存储获取的点位状态, 1 代表触发; 0 代表未触发

(10) 获取设备模拟量输入

NIO_GetADC(HAND devHandle, unsigned short adcStart, unsigned short adcCount, unsigned short *value);

参数	输入/输出	描述
devHandle	输入	模块句柄
adcStart	输入	起始通道号, 取值范围(0-3)
adcCount	输出	要获取的通道数量, 取值范围(1-4)
value	输入	用于存储模拟量输入值缓冲

(11) 设置设备模拟量输出

NIO_SetDAC(HAND devHandle, unsigned short dacStart, unsigned short dacCount, const unsigned short *value);

参数	输入/输出	描述
devHandle	输入	模块句柄
dacStart	输入	起始通道号, 取值范围(0-3)
dacCount	输出	要获取的通道数量, 取值范围(1-4)
value	输入	要设置的端口模拟量输出值缓冲

(12) 修改 SIO 设备在系统中名称

NIO_WriteID(HAND devHandle, const char *idStr);

参数	输入/输出	描述
devHandle	输入	模块句柄
idStr	输入	新的设备名称, 最长 16 字节包括末尾'\0'字符

(13) 读取 SIO 设备名称

NIO_ReadID(HAND devHandle, char *productId);

参数	输入/输出	描述
devHandle	输入	模块句柄
productId	输入	用于存放设备名称缓冲区, 最长 16 字节

(14) 修改 SIO 设备 ID 序号（用于 Modbus 通讯的设备 ID）

`NIO_WriteDevID(HAND devHandle, const unsigned char devId);`

参数	输入/输出	描述
<code>devHandle</code>	输入	模块句柄
<code>devId</code>	输入	新的设备序号

(15) 读取 SIO 设备 ID 序号（用于 Modbus 通讯的设备 ID）

`NIO_ReadDevID(HAND devHandle, unsigned char *devId);`

参数	输入/输出	描述
<code>devHandle</code>	输入	模块句柄
<code>devId</code>	输出	用于存放设备 ID 序号

(16) 获取设备 IO 口逻辑

`NIO_GetIOLogic(HAND devHandle, unsigned short *value);`

参数	输入/输出	描述
<code>devHandle</code>	输入	模块句柄
<code>value</code>	输出	用于存储 IO 逻辑状态

(17) 设置设备 IO 口逻辑

`NIO_SetIOLogic(HAND devHandle, unsigned short value);`

参数	输入/输出	描述
<code>devHandle</code>	输入	模块句柄
<code>value</code>	输入	要设置的 IO 逻辑状态

(18) SC 设备操作过程中，最后出现的错误，以字符串的形式提供

`NIO_GetError(HAND devHandle, char *str, unsigned short len);`

参数	输入/输出	描述
<code>devHandle</code>	输入	模块句柄
<code>str</code>	输入	用于存储错误信息的缓冲
<code>len</code>	输出	缓冲区大小

4.3 应用案例

4.3.1 挂接 GCN 控制器例程

```

/*****此处省略控制器初始化部分*****/
HAND devHandle = 0;
short rtn = 0;
long divaule = 0;
long divalueex = 0;
long dovalue = 0;
long dovalueex = 0;
short di0 = 0;
short di64 = 0;
//读取扩展模块，设备 ID 从 2 开始，拨码开关也要拨码成对应 ID
rtn = NMC_IOModuleSetEn(devHandle ,2,1);
//读取一组扩展模块 IO 的数字量输入
rtn = NMC_GetDIGroup(devHandle ,&divalueex,2);
//读取一组本地 IO 数字量输入
rtn = NMC_GetDIGroup(devHandle ,&divaule,0);
//按位读取（单个读取），这里示范读取第 0 个输入
rtn = NMC_GetDIBit(devHandle ,0,&di0);
//按位读取第一个扩展模块第一个 IO 输入值
rtn = NMC_GetDIBit(devHandle ,64,&di64);
//读取一组扩展模块 IO 的数字量输出（地址为 2 的扩展模块 DO）
rtn = NMC_GetDOGroup(devHandle ,&dovalueex,2);
//读取本地数字量输出
rtn = NMC_GetDOGroup(devHandle ,&dovalue,0);
//设置本地数字量输出
rtn = NMC_SetDOGroup(devHandle ,0,0);
//设置本地第 1 个输出
rtn = NMC_SetDOGroup(devHandle ,0xFFFF,0);
//设置本地第 0 个输出,按位输出
rtn = NMC_SetDOBit(devHandle ,0,0);
return rtn;
    
```

4.3.2 独立使用例程(C#)

```
/****** using 引用部分自行引入, rtn 返回值自行判断 *****/
short rtn = 0;
ushort devNum = 0;
private IntPtr devHandle;
TDevInfo devInfo=new TDevInfo();
// 搜索扩展模块
rtn = NIO_Search(ref devNum, ref devInfo);
// 打开扩展模块, 默认设备名称: CARD1
rtn = NIO_OpenByID("CARD1", ref devHandle);
ushort input = 0;
// 获取输入状态, 0代表有输入
rtn = NIO_GetDI(devHandle, 0, 1, ref input);
ushort diVal = 0;
//按位获取输入
rtn = NIO_GetDIBit(devHandle, 2, diVal);
ushort output = 0;
//获取输出状态, 0 代表有输出
rtn = NIO_GetDO(devHandle, 0, 1, ref output);
// 按位设置输出 关闭其他输出
ushort output = 65534;
rtn = NIO_SetDO(devHandle, 0, 1, ref output);
// 按位设置输出
rtn = NIO_SetDOBit(devHandle, 1, 0);
ushort pv = 0;
// 按位获取输出
rtn = NIO_GetIOLogic(devHandle, ref pv);
```