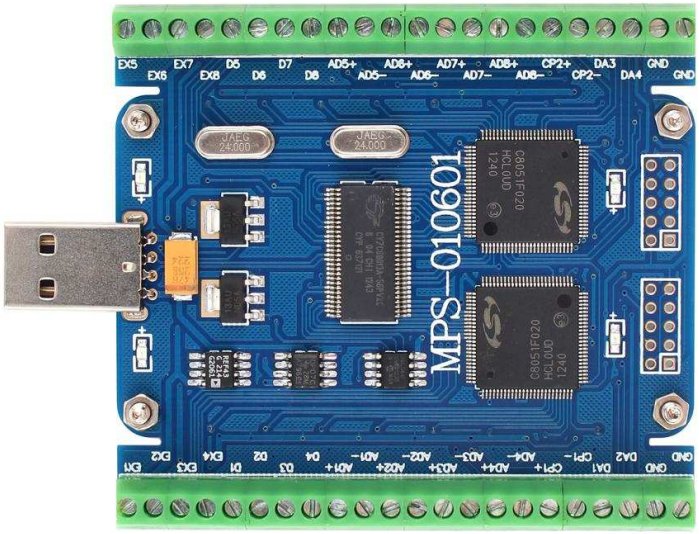
MPS-010601

多功能 USB 信号采集卡

使用说明

Ver 1.2

第一章 产品概述



MPS-010601 多功能 USB 信号采集卡

一、 *产品简介*

MPS-010601 信号采集卡是一款基于 USB 总线的多功能信号采集卡，具有 8 路差分模拟信号采集、4 路模拟信号输出、8 路数字信号输入/输出、2 路比较器、2 路计数器及 2 路 PWM 输出。可用于传感器信号数据采集与分析、工业现场监测与控制、高等院校科研与教学等多种领域。使用 MPS-010601 可以将传感器和控制器与计算机结合在一起，利用计算机强大的数据处理能力和灵活的软件编程方式，对信号进行分析、处理、显示与记录，从而用低廉的成本取代多种价格昂贵的专用仪器，并且能通过编程来获得免费的功能升级。先进的设计理念、丰富的硬件功能与简洁的编程方式使 MPS-010601 成为工业企业和科研机构必备的强大设计工具。

MPS-010601 采用 USB2.0 高速总线接口，支持即插即用和热插拔，是便携式系统用户的最佳选择，可以取代传统仪器与 PCI 等接口板卡。

MPS-010601 可工作在 Win9X/Me、Win2000/XP、WIN7/WIN8/WIN10 等常用操作系统中，并提供可供 VB, VC, C++Builder, Dephi，LabVIEW，Matlab 等常用编程语言调用的动态链接库，编程函数接口简单易用，易于编写应用程序。

二、 *性能指标*

2.1、USB 总线性能

- USB2.0 高速总线传输
- 支持热插拔和即插即用

2.2、模拟信号输入

- 模拟输入通道: 2 路差分（同步）、4 路差分（多路扫描）、8 路差分（多路扫描）
- 输入端口耐压： -5V—5V
- 差分输入量程： ±7.5V
- 模拟输入阻抗: 30K 欧姆
- 分辨率： 12Bit(4096)

· 最大总误差： < 0.5%
· 可编程增益： 1、2、4、8、16
· 采样时钟： 5Ksps-80Ksps 内部时钟或外部时钟

2.3、模拟信号输出

· 模拟输出通道: 4 路单端（同步）
· 模拟输出范围： 0-2.5V
· 模拟输出电流: 300 微安
· 分辨率： 12Bit(4096)
· 非线性误差： ±2LSB
· 刷新时钟： 5Ksps-80Ksps 内部时钟或外部时钟

2.4、数字信号输入/输出

· 输入/输出通道： 8 路
· 输入/输出模式： 全输入/全输出/半输入半输出
· 输入电平： 兼容 TTL 或 CMOS
· 输出电平： CMOS
· 输入/输出时钟： 5Ksps-80Ksps 内部时钟或外部时钟

2.5、比较器

· 比较器个数： 2
· 输入电压： 0-3.3V
· 响应时间： ≤10 微秒
▪ 回差电压： 正向与反向各 2mV
▪ 比较器输出： CMOS 电平

2.6、计数器

▪ 计数器个数： 2
▪ 输入电平： TTL 或 CMOS
▪ 计数位： 16 位（最大 65535）
▪ 工作时钟： 5Ksps-80Ksps 内部时钟或外部时钟

2.7、PWM 输出

▪ PWM 输出通道： 2
▪ PWM 输出电平： CMOS
▪ PWM 输出脉宽： 8bit 或 16bit
▪ PWM 时基： 2M 或 24M
▪ PWM 状态显示： LED

2.8、FIFO 存储器

▪ FIFO 个数： 4
▪ 存储深度： 1K

2.9、工作温度

▪ 0℃ - 70℃

三、 *应用领域*

便携式仪表和测试设备
传感器信号采集与分析
工业控制

四、 软件支持

提供 Windows95/98/NT/2000/XP/WIN7/WIN8/WIN10 下的驱动程序（支持 64 位系统）及

DLL 文件，并提供 LabVIEW 编写的应用软件范例程序。

五、 销售清单

[1] MPS-010601 多功能 USB 信号采集卡一张；

[2] 高屏蔽 USB 数据传输电缆一根；

[3] 保修卡一张；

六、 售后服务

一年内免费维修，终身维修。

第二章 设备安装

一、 *MPS-010601 信号采集卡硬件接口说明*

GND： 采集卡地线端口

DAx： 模拟信号输出端口

CPx+: 比较器正输入端口

CPx-: 比较器负输入端口

ADx+: 模拟信号差分输入正端口

ADx-: 模拟信号差分输入负端口

Dx: 数字信号输入/输出端口

EXx: 扩展端口，其中：

EX1：内部工作时钟输出

EX2: 计数器 1 输入

EX3: PWM1 输出

EX4: CP1 状态输出，CP1+ > CP1- 时 EX4 为高电平

EX5：使用外部时钟时为外部时钟输入；使用内部时钟时为内部时钟输出

EX6: 计数器 2 输入

EX7: PWM2 输出

EX8: CP2 状态输出，CP2+ > CP2- 时 EX8 为高电平

二、 *MPS-010601 信号采集卡指示灯状态说明*

- 绿色 LED：系统自检指示。LED 亮，系统正常工作。
- 红色 LED：采集状态指示。LED 亮，正在进行采集；采集卡灭，采集中断或停止。
- 蓝色 LED：PWM 输出状态指示。LED 亮度指示 PWM 占空比。

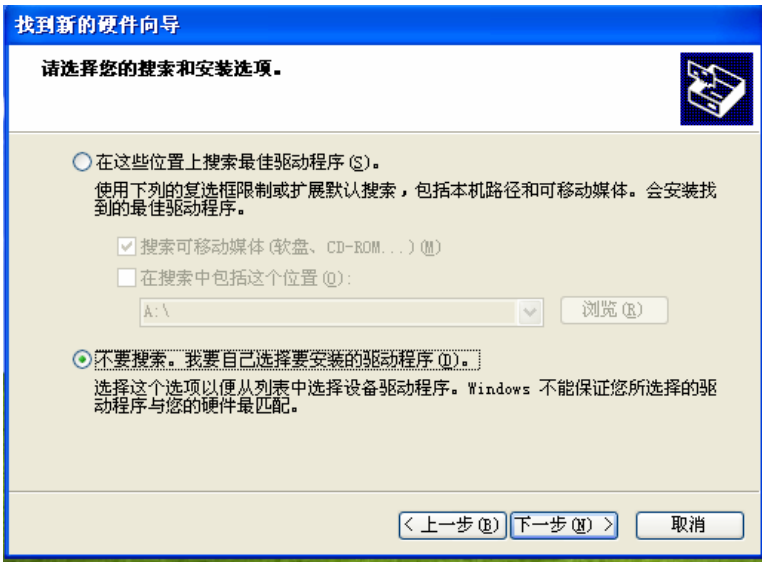
三、 *驱动安装*

注：驱动安装说明以 XP 系统为例。在 WIN7、WIN8、WIN10 等操作系统下，若插入板卡后操作系统未自动弹出驱动安装向导，可到“设备管理器”中手动调出向导。参考步骤为：在“我的电脑”上点鼠标右键，选择“属性”-“硬件”-“设备管理器”，来打开设备管理器。一般未安装驱动时，板卡设备会出现在管理器中“未知设备”、“其他设备”或“通用串行总线控制器”列表中，找到板卡设备后，在上面点击鼠标右键，选择“更新驱动程序”，即可调出驱动安装向导。调出驱动安装向导后，后续步骤与 XP 系统一致，按如下步骤说明进行安装即可。

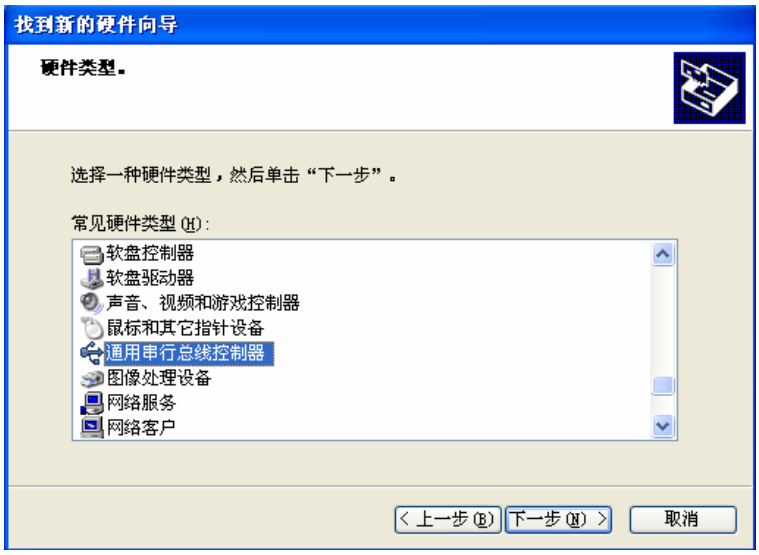
1. 首次使用本卡时，计算机将提示“发现新硬件”，如下图所示。选择“否，暂时不”，并点击“下一步”。

2. 选择“从列表或指定位置安装（高级）”，点击“下一步”。

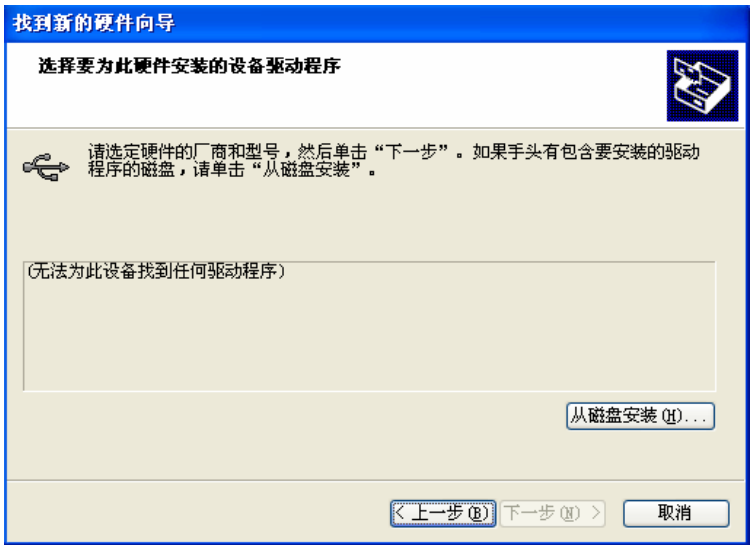
3. 选择“不要搜索。我要自己选择安装的驱动程序”，点击下一步。



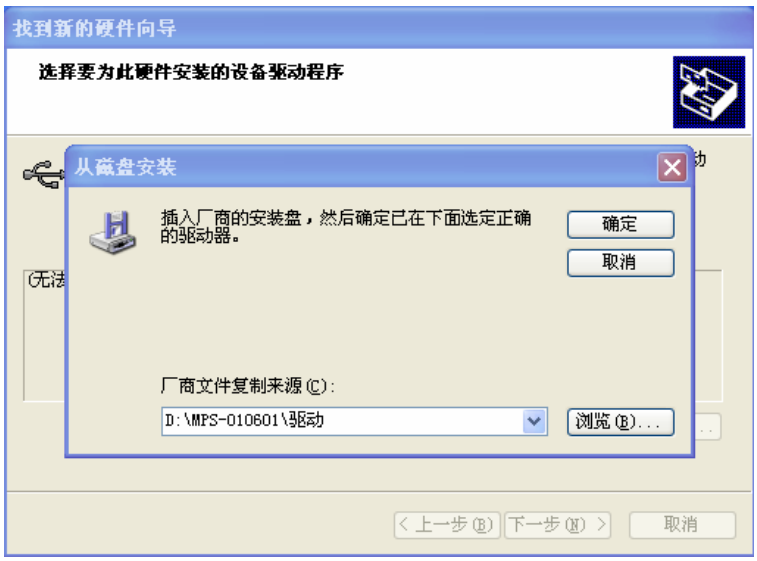
4. 选择“通用串行总线控制器”，点击下一步。



5. 在弹出的对话框中选择“从磁盘安装”。

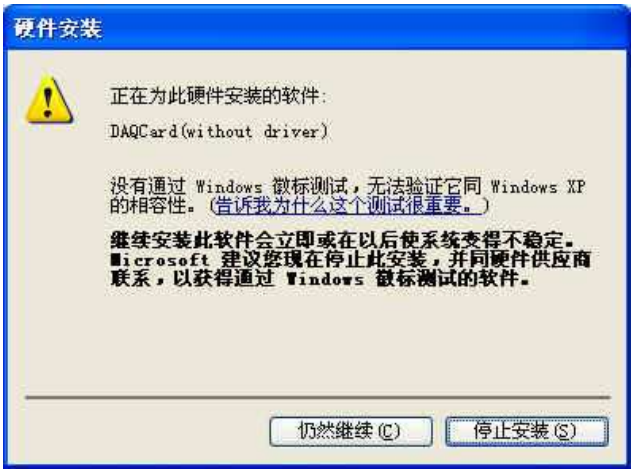


6. 选择 MPS-010601 采集卡驱动所在的目录，选中文件“MPS-010601.inf”。点击确定。



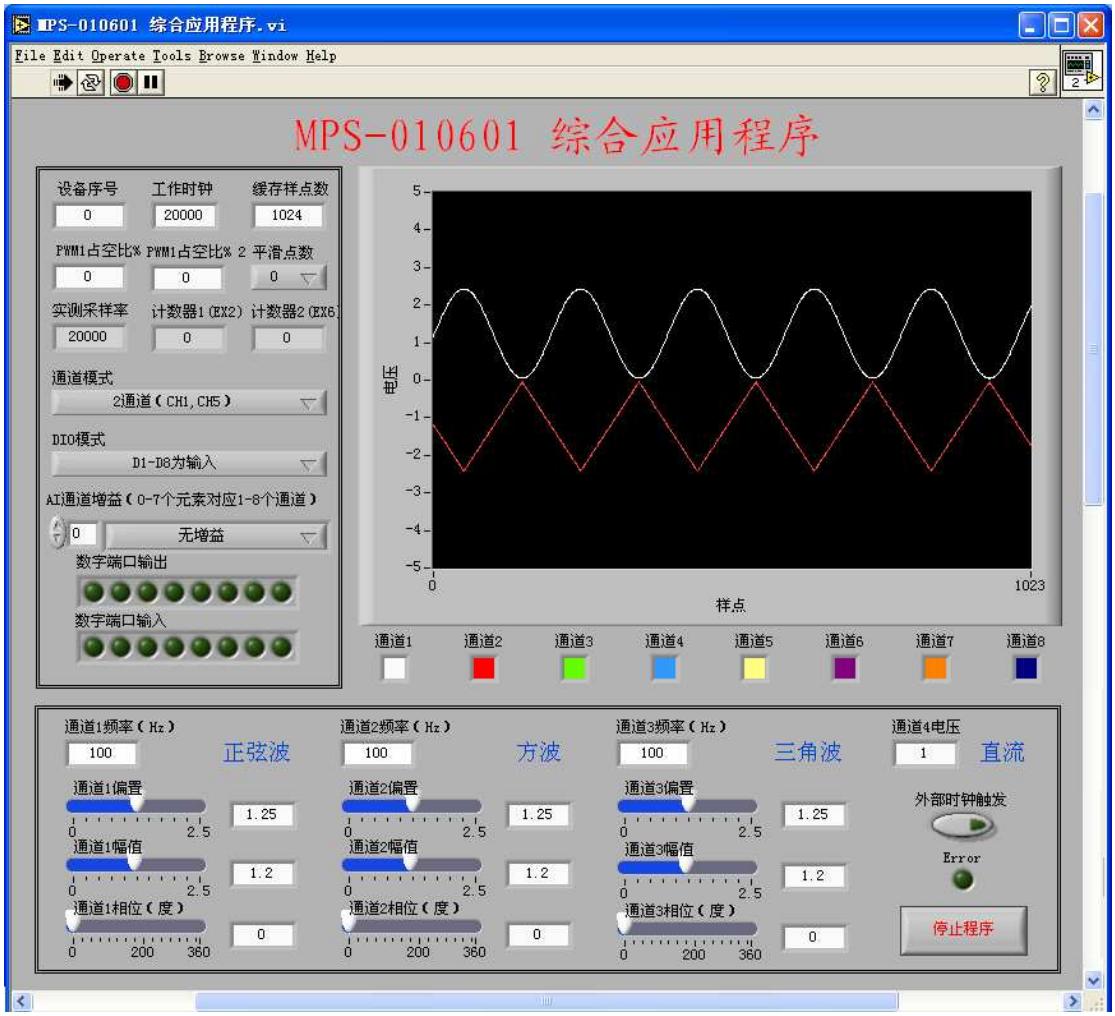
7. 选中“MPS-010601 USBDAQCard”，点击下一步。

8. 系统开始安装驱动，若弹出如下对话框，选择“仍然继续”。



9. 驱动安装完成。

四、 基本功能测试



MPS-010601 各项基本功能测试：

1. 将 MPS-010601 信号采集卡与计算机的 USB 接口连接。
2. 按第三小节所描述的过程安装驱动程序。
3. 打开范例程序中的“MPS-010601 综合应用程序”（需先安装 LabVIEW 软件）。
4. 点击左上角的执行按钮（白色箭头），运行程序。板卡上的红色指示灯亮起。
5. 将 DA1 至 DA4 分别连接 AD1+，AD1-接至 GND，则从波形图上可以依次看到正弦波、方波、三角波和直流电平。修改软件下部的参数可以改变 DA 输出的信号特征。
6. 点击“数字端口输出”相应的状态指示灯，可从“数字端口输入”看到对应的状态改变。
7. 修改 PWM 输出值（0%-100%），可看到采集卡相应 PWM 指示灯的亮度发生变化。
8. 可通过修改其他参数来进行更多测试。
9. 按“Ctrl”键可暂停波形显示。
10. 若采集异常，软件会报错，并自动停止程序。
11. 点击“停止程序”退出程序。
12. 更多测试程序请参考所提供的范例程序包。

第三章 用户编程

一、 *动态链接库（DLL）*

MPS-010601 采用 DLL（Dynamic Linkable Library，动态链接库)的方式来进行编程驱动。DLL 的编制与具体的编程语言及编译器无关，只要遵循约定的 DLL 接口规范和调用方式，用各种语言编写的 DLL 都可以相互调用。

DLL 可以方便的在 VC、VB、LabVIEW 等语言下被调用，具体方式分别为：

- VC 下调用 DLL

```
typedef void ( * FUNC )(void);                  //定义一个函数指针
FUNC Func;                                      //定义一个函数指针变量
HINSTANCE hDLL=LoadLibrary("DllTest.dll");      //加载 dll
Func=(FUNC)GetProcAddress(hDLL,"FuncInDLL");    //找到 dll 中的函数
Func();                                         //调用 dll 里的函数
```

- VB 下调用 DLL

[Public | Private] Declare Function name Lib " labname " [Alias " aliasname "] [([arglist])] [As type] "

Public（可选）用于声明在所有模块中的所有过程都可以使用的函数；Private（可选）用于声明只能在包含该声明的模块中使用的函数。

Name（必选）任何合法的函数名。动态链接库的入口处（entry points）区分大小写。

Libname（必选）包含所声明的函数动态链接库名或代码资源名。

Alias（可选）表示将被调用的函数在动态链接库（DLL）中还有另外的名称。当外部函数名与某个函数重名时，就可以使用这个参数。当动态链接库的函数与同一范围内的公用变量、常数或任何其它过程的名称相同时，也可以使用 Alias。如果该动态链接库函数中的某个字符不符合动态链接库的命名约定时，也可以使用 Alias。

Aliasname（可选）动态链接库。如果首字符不是数字符号（#），则 aliasname 是动态链接库中该函数入口处的名称。如果首字符是（#），则随后的字符必须指定该函数入口处的顺序号。

Arglist（可选）代表调用该函数时需要传递参数的变量表。

Type（可选）Function 返回值的数据类型；可以是 Byte、Boolean、Integer、Long、Currency、Single、Double、Decimal（目前尚不支持）、Date、String（只支持变长）或 Variant，用户定义类型，或对象类型。

arglist 参数的语法如下：

[Optional] [ByVal | ByRef] [ParaArray] varname [()] [As type]

Optional（可选）表示参数不是必需的。如果使用该选项，则 arglist 中的后续参数都必需是可选的，而且必须都使用 Optional 关键字声明。如果使用了 ParamArray，则任何参数都不能使用 Optional。

ByVal（可选）表示该参数按值传递。

ByRef（可选）表示该参数按地址传递。

- LabVIEW 下调用 DLL

在 LabVIEW 中，调用 DLL 是通过 CLF 节点来完成的。所谓 CLF 节点（Call Library Function，调用函数库节点），是指可以在 LabVIEW 调用其他语言封装的 DLL，CLF 节点位于位于 LabVIEW 功能模板中的 Advanced 子模板中，其配置过程如下：

- 在 CLF 节点的右键菜单中选择“Configure”，弹出 CLF 节点配置对话框；
- 点击“Browse”按钮，在随后弹出的选择 DLL 文件对话框中找到你需要用的 DLL 文件，此时，LabVIEW 就会自动装载选定的 DLL 文件，并检测 DLL 文件中所包含函数。但是函数中的参数和参数的数据类型需要用户根据函数的输入、输出参数手动设置。因而在调用 DLL 文件时，要求用户对 DLL 文件有较为详细的了解。
- 在 FunctionName 下拉列表框中选定动态连接库中所包含的所需要 API 函数；
- 在 Calling Convention 下拉菜单中选择 StdCall(WINAPI)和 C 两个选项，若用户选定的是 Windows API 函数，则选用 StdCall(WINAPI)选项;若用户选用的 DLL 中的函数是非 Windows API 函数，则选用 C 选项；
- 设置函数的返回参数。函数参数的类型要与 DLL 中函数本身所定义的函数参数类型相对应，如果不对应，函数就会出现数据错误和强制类型转换；
- 根据所选函数的函数原型，设置函数的输入参数及数据类型。点击 Add a Parameter 按钮，即可以添加一个新的输入参数。

二、 *编程函数及参数*

MPS-010601 提供的驱动 DLL 文件名为 MPS-010601.dll，内部共有四个驱动函数，分别为：

- extern "C" int SetPara(int SampleRate, int ADChannelNumber, int *ADPGAofChannels, int DIOModal, unsigned short PWM1, unsigned short PWM2, int ExtTrigger,int DeviceNumber)

int SetPara 函数执行配置采集卡参数的功能。若函数执行成功，返回 1；执行失败返回 0。

int SampleRate：采样率、刷新率等工作时钟频率。此参数为内部时钟频率设定。参数取值范围为 5000-80000，小于 5000 将被设置为 5000，大于 80000 将被设置为 80000。若 ExtTrigger = 0， EX1 和 EX5 对外输出该频率的时钟脉冲；若 ExtTrigger = 1，只有 EX1 输出时钟脉冲。对于 AD 而言，SampleRate 的值为总采样率值，实际分配到每个通道上的采样率为 SampleRate / (ADChannelNumber / 2)。对于 DA 和 DIO 而言，SampleRate 就是每个通道的工作频率值。

int ADChannelNumber:模拟输入通道数。ADChannelNumber = 2， AD1 与 AD5 分别被配置为两路模拟信号输入，并且为同步采集，其余 ADx 口无效；ADChannelNumber = 4，AD1、AD2、AD5、AD6 被配置为四路模拟信号输入，AD1 与 AD5 同步，AD2 与 AD6 同步，相邻通道为切换扫描模式，其余 ADx 口无效；ADChannelNumber = 8，所有通道被配置为八路模拟信号输入，AD1 与 AD5 同步，AD2 与 AD6 同步，AD3 与 AD7 同步，AD4 与 AD8 同步，相邻通道为切换扫描模式。若给出的 ADChannelNumber 参数值小于 4，则自动配置为 2；若给出的 ADChannelNumber 大于 4 而小于 8，则自动配置为 4；若给出的 ADChannelNumber 大于 8，则自动配置为 8。

int * ADPGAofChannels：模拟输入增益设置。ADPGAofChannels 为一维 8 元素数组，数组元素依次代表模拟输入 1-8 通道的增益系数。每个元素的取值为：ADPGAofChannels[i] = 1，无增益；ADPGAofChannels[i] = 2，2 倍增益；ADPGAofChannels[i] = 4，4 倍增益；ADPGAofChannels[i] = 8，8 倍增益；ADPGAofChannels[i] = 16，16 倍增益；ADPGAofChannels[i] = 255，0.5 倍增益。若 ADPGAofChannels[i]为其他值，则自动设置 ADPGAofChannels[i] = 1。

int DIOModal：数字输入/输出端口模式设置。DIOModal = 0，D1-D8 全部为输入模式；DIOModal = 1，D1-D8 全部为输出模式；DIOModal = 2，D1-D4 为输出模式，D5-D8

为输入模式；DIOModal = 3，D1-D4 为输入模式，D5-D8 为输出模式。若 DIOModal 为其他值自动配置 DIOModal = 0。

unsigned short PWM1：PWM1 输出占空比设置。PWM1 取值范围为 0-65535，其值越大占空比越高。注：若 SampleRate 大于等于 50000，PWM1 时基为 24M，PWM1 输出为 16bit 循环模式；若 SampleRate 小于 50000，PWM1 时基为 2M，PWM1 输出为 8bit 循环模式（PWM1 的高 8 位有效）。一般情况下建议使用 16 位循环模式。

unsigned short PWM2：PWM2 输出占空比设置。PWM2 取值范围为 0-65535，其值越大占空比越高。注：若 SampleRate 大于等于 50000，PWM2 时基为 24M，PWM2 输出为 16bit 循环模式；若 SampleRate 小于 50000，PWM2 时基为 2M，PWM2 输出为 8bit 循环模式（PWM2 的高 8 位有效）。一般情况下建议使用 16 位循环模式。

int ExtTrigger：外部时钟触发使能。ExtTrigger = 0，使用内部时钟触发采集和输出；ExtTrigger 为其他值时使用外部时钟触发。一般情况下建议使用内部时钟。若使用内部时钟，则内部时钟将从 EX1 与 EX5 输出；若使用外部时钟，则外部时钟从 EX5 输入，同时内部时钟从 EX1 输出。可通过该功能同步多块采集卡进行同步采集来扩展通道数。

int DeviceNumber：操作所针对的设备号。当有多块采集卡同时连接到计算机时，将按照设备连接到计算机的先后顺序依次分配序号 0、1、2、……9，该序号将用于对卡进行标识。只有一块卡连接时，默认设备号为 0。最多支持同时连接 10 个设备。

- extern "C" int DataIn(float *VoltageIn1,float *VoltageIn2,unsigned char *DI,int SampleNumber ,int DeviceNumber)

int DataIn 函数执行模拟信号和数字信号的采集功能。若函数执行成功，返回 1；执行失败返回 0。

float * VoltageIn1：第一组模拟信号输入（AD1-AD4）的数据。VoltageIn1 为一个一维数组，其每个元素代表一个采样点的电压值。如 VoltageIn1[i] = 1.245，则表示第 i 个样点对应的电压为 1.245V。若采集卡工作在 2 通道模拟输入模式下，则 VoltageIn1 中的元素全部代表 AD1 采集到的数据；若采集卡工作在 4 通道模拟输入模式下，则 VoltageIn1 的第一个元素代表 AD1 采集到的数据，第二个元素代表 AD2 采集到的数据，第三个元素代表 AD1 采集到的数据，第四个元素代表 AD2 采集到的数据……以此类推；若采集卡工作在 8 通道模拟输入模式下，则 VoltageIn1 的第一个元素代表 AD1 采集到的数据，第二个元素代表 AD2 采集到的数据，第三个元素代表 AD3 采集到的数据，第四个元素代表 AD4 采集到的数据，第五个元素代表 AD1 采集到的数据，……以此类推。若函数执行成功，该数组内数据被自动更新为最新采集到的数据（更新的元素个数由 SampleNumber 决定）；若函数执行失败，该数组内数据无效。VoltageIn1 所指向的数组大小应大于 SampleNumber 的大小。

float * VoltageIn2：第二组模拟信号输入（AD5-AD8）的数据。VoltageIn2 为一个一维数组，其每个元素代表一个采样点的电压值。如 VoltageIn2[i] = 1.245，则表示第 i 个样点对应的电压为 1.245V。若采集卡工作在 2 通道模拟输入模式下，则 VoltageIn2 的元素全部代表 AD5 采集到的数据；若采集卡工作在 4 通道模拟输入模式下，则 VoltageIn2 的第一个元素代表 AD5 采集到的数据，第二个元素代表 AD6 采集到的数据，第三个元素代表 AD5 采集到的数据，第四个元素代表 AD6 采集到的数据……以此类推；若采集卡工作在 8 通道模拟输入模式下，则 VoltageIn2 的第一个元素代表 AD5 采集到的数据，第二个元素代表 AD6 采集到的数据，第三个元素代表 AD7 采集到的数据，第四个元素代表 AD8 采集到的数据，第五个元素代表 AD5 采集到的数据，……以此类推。

若函数执行成功，该数组内数据被自动更新为最新采集到的数据（更新的元素个数由 SampleNumber 决定）；若函数执行失败，该数组内数据无效。VoltageIn2 指向的数组大小应大于 SampleNumber 的大小。

unsigned char * DI：数字信号采集得到的数据。DI 为一个一维数组，其每个元素为 8 位 unsigned char 型数据，8 个数据位分别代表同一时刻采样得到的 8 路数字端口电平状态。如：DI[i] = 17，即 DI = 0b00010001，表示 D1 和 D5 为高电平，其余 6 个端口为低电平。无论数字输入/输出端口工作在输入模式还是输出模式，都可以获得当前 Dx 端口的电平状态。若函数执行成功，该数组内数据被自动更新为最新采集到的数据（更新的元素个数由 SampleNumber 决定）；若函数执行失败，该数组内数据无效。DI 指向的数组大小应大于 SampleNumber 的大小。

int SampleNumber：一次采集的样点个数。该参数决定函数执行一次数据数组中所更新的数据个数，当从采集卡中读到 SampleNumber 个数据点后函数成功返回。该参数的最小值为 128，且必须为 128 的倍数，否则根据向下就近原则自动配置为 128 的倍数。该参数无最大限制，但建议一次采集样点不要过大，以免等待时间过长影响程序执行效率。若 SampleNumber 较大或两次执行采集程序间隔较长，导致出现采集到的数据前段出现杂乱，可以将杂乱部分（一般为 256 个样点）从有效数据中剔除，或在采集有效数据之前先执行一个 256 样点的读数操作以清空 FIFO 的数据缓存。

int DeviceNumber：操作所针对的设备号。

- extern "C" int DataOut(float *VoltageOut1,float *VoltageOut2,float *VoltageOut3,float *VoltageOut4,unsigned char *DO,int SampleNumber,int DeviceNumber)

int DataOut 函数执行模拟信号和数字信号的输出功能。若函数执行成功，返回 1；执行失败返回 0。

float * VoltageOut1：模拟输出通道 DA1 将输出的数据。VoltageOut1 为一个一维数组，其每个元素代表 DA1 输出的一个样点的电压值，元素的取值范围为 0-2.5V。如 VoltageOut1[i] = 1.29，表示 DA1 即将输出的第 i 个样点电压为 1.29V。若元素值超出取值范围，则会出现数据溢出导致输出失真。若函数执行成功，该数组内 SampleNumber 个数据将会被输出；若函数执行失败，该数组内数据不被输出。VoltageOut1 指向的数组大小应大于 SampleNumber 的大小。

float * VoltageOut2：模拟输出通道 DA2 将输出的数据。其规定类同 VoltageOut1。

float * VoltageOut3：模拟输出通道 DA3 将输出的数据。其规定类同 VoltageOut1。

float * VoltageOut4：模拟输出通道 DA4 将输出的数据。其规定类同 VoltageOut1。

unsigned char * DO：数字输入/输出通道 D1-D8 将输出的数据。DO 为一个一维数组，其每个元素为一个 unsigned char 型数据，8 个数据，分别代表同一时刻输出的 8 路数字端口电平状态。如：DO[i] = 17，即 DI = 0b00010001，表示 D1 和 D5 输出为高电平，其余 6 个端口为低电平。值得注意的是：当对应的端口 Dx 端口被设置为输入模式时，向 Dx 输出低电平表示将 Dx 拉低，Dx 将不随输入变化而变化；向 Dx 输出高电平表示将 Dx 配置为输入状态，此时才可以从 Dx 中读到外界输入的电平状态。因此**在配置 Dx 为输入模式的同时，应对 Dx 输出高电平。**当 Dx 被配置为输出模式时，其输出电平状态将由 DO 中的输出值决定。若函数执行成功，该数组内 SampleNumber 个数据将会被输出；若函数执行失败，该数组内数据不被输出。DO 指向的数组大小应大于 SampleNumber 的大小。

int SampleNumber：一次输出的样点个数。该参数决定函数执行一次数据数组中所输出的数据个数，当向采集卡中输出 SampleNumber 个数据点后函数成功返回。该参数的最小值为 128，且必须为 128 的倍数，否则根据向下就近原则自动配置为 128 的倍数。该参数无最大限制，但建议一次输出样点不要过大，以免等待时间过长影响程序执行效率。

int DeviceNumber：操作所针对的设备号。

- extern "C" int Counter(int * Counter1, int * Counter2, int DeviceNumber)

int Counter 函数执行读取计数器计数值的功能。若函数执行成功，返回 1；执行失败返回 0。

int * Counter1：指向保存计数器 1 计数值的变量的指针。指针所指向的对象为整型数，函数成果执行后该整型数将被更新为最新的计数器计数值。计数值范围为 0-65535，超出后将从新从 0 计数。执行 SetPara 函数后，计数器的值将被清零。

int * Counter2：指向保存计数器 2 计数值的变量的指针。指针所指向的对象为整型数，函数成果执行后该整型数将被更新为最新的计数器计数值。计数值范围为 0-65535，超出后将从新从 0 计数。执行 SetPara 函数后，计数器的值将被清零。

int DeviceNumber：操作所针对的设备号。

三、 *VC 程序范例*

```
void MPS_TEST()
{
    HINSTANCE hDll = LoadLibrary("MPS-010601.dll");            //加载DLL
    if(NULL==hDll)
    {
        AfxMessageBox("Can’t find DLL");
    }

    //函数声明
    //SetPara函数
    typedef int(* Type_SetPara)(int SampleRate, int ADChannelNumber, int *ADPGAofChannels, int
DIOModal, unsigned short PWM1, unsigned short PWM2, int ExtTrigger,int DeviceNumber);
    Type_SetPara MPS_SetPara=(Type_SetPara)GetProcAddress(hDll,"SetPara");
    if(MPS_SetPara == NULL)
    {
        AfxMessageBox("Can’t find <SetPara> function");
    }

    //DataIn函数
    typedef int(* Type_DataIn)(float *VoltageIn1,float *VoltageIn2,unsigned char *DI,int
SampleNumber ,int DeviceNumber);
    Type_DataIn MPS_DataIn=(Type_DataIn)GetProcAddress(hDll,"DataIn");
    if(MPS_DataIn == NULL)
    {
        AfxMessageBox("Can’t find <DataIn> function");
```

```
    }

    //DataOut函数
    typedef int(* Type_DataOut)(float *VoltageOut1, float *VoltageOut2, float *VoltageOut3, float
*VoltageOut4, unsigned char *DO, int SampleNumber, int DeviceNumber);
    Type_DataOut MPS_DataOut=(Type_DataOut)GetProcAddress(hDll, "DataOut");
    if(MPS_DataOut == NULL)
    {
        AfxMessageBox("Can' t find <DataOut> function");
    }

    //Counter函数
    typedef int(* Type_Counter)(int * Counter1, int * Counter2, int DeviceNumber);
    Type_Counter MPS_Counter=(Type_Counter)GetProcAddress(hDll, "DataOut");
    if(MPS_Counter == NULL)
    {
        AfxMessageBox("Can' t find <Counter> function");
    }

    //变量定义及初始化
    int PGAArray[8] = {1, 1, 1, 1, 1, 1, 1, 1,};                    //PGA数组
    #define SampleNumber 128
    float VoltageIn1[SampleNumber];
    float VoltageIn2[SampleNumber];
    float VoltageOut1[SampleNumber];
    float VoltageOut2[SampleNumber];
    float VoltageOut3[SampleNumber];
    float VoltageOut4[SampleNumber];
    unsigned char DI[SampleNumber];
    unsigned char DO [SampleNumber];
    int MPS_flag = 0;
    int i = SampleNumber;
    while(i--)                                                      //数据数组初始化
    {
        VoltageIn1[i] = 0;
        VoltageIn2[i] = 0;
        VoltageOut1[i] = (float)i/SampleNumber;                     //初始化为锯齿波
        VoltageOut2[i] = (float) ( 2 * (i >= SampleNumber/2));      //初始化为方波
        if(i < SampleNumber/2)                                      //初始化为三角波
        VoltageOut3[i] = ((float)i / SampleNumber ) * 4;
        else VoltageOut3[i] = ((float)(SampleNumber - i) / SampleNumber ) * 4;
        VoltageOut4[i] = 1;                                         //初始化为直流电平
        DI[i] = 0;
        DO[i] = 0x11;                                               //D1和D5为高电平
```

```
    }

    //采集与输出
    MPS_flag = MPS_SetPara( 50000, 2, PGAArray, 1, 0, 0, 0, 0);        //初始化参数：采样率为
K；2通道采集；无增益；Dx全部为输出模式；无PWM输出；内部时钟；设备0
    if(MPS_flag == 0)
    {
        AfxMessageBox("DAQ Error!Please check hardware!");            //报错
    }
    else
    {
        for(i = 0; i < 10; i++)                                       //循环执行
    {
            MPS_flag = MPS_DataIn(VoltageIn1, VoltageIn2, DI, SampleNumber, 0);   //采集
            if(MPS_flag == 0)
            {
                AfxMessageBox("DAQ Error!Please check hardware!");
                break;
            }

            MPS_flag = MPS_DataOut(VoltageOut1, VoltageOut2, VoltageOut3, VoltageOut4, DO,
SampleNumber,0);   //输出
            if(MPS_flag == 0)
            {
                AfxMessageBox("DAQ Error!Please check hardware!");
                break;
            }
            //此处可添加对采集到的数据的处理代码
        }
    }
    return;
}
```

第四章 注意事项

- 拔插采集卡请用力适度，以免损害 USB 接口。
- 模拟信号输入端口输入电压范围为-5V 至+5V，数字信号输入端口电压不得高于 5V 或低于 0V，允许范围之外的电压有可能对采集卡硬件造成损害。
- 用户须注意电源的开关顺序，使用时要求先将采集卡连至计算机，而后开启信号源电源；关闭时先关信号源电源，后将采集卡与计算机断开。
- 将采集卡拔离计算机后，请间隔 5 秒以上再将卡插入。拔插过快有可能造成采集卡初始化异常，此时请重新拔插采集卡。
- 使用过程中不要用手接触电路和芯片，避免人体静电对硬件造成损害。采集卡保存时请妥善保管，注意防尘防潮。
- MPS-010601 自出厂之日起，一年内凡用户遵守贮存，运输和使用要求，而产品质量低于技术指标的，凭保修卡免费维修。因违反操作规定和要求而造成损坏的，需交纳器件维修费。
- MPS 系列信号采集卡由北京启创莫非电子科技有限公司设计生产，公司网址：www.mps-electronic.com.cn,咨询邮箱 mail@mps-electronic.com.cn。