MPS-060602

高性能双通道 IEPE（ICP）传感器

信号采集卡使用说明

Ver. 1.0

第一章 产品概述

一、 产品简介

MPS-060602 是一款基于 USB 总线的高性能 16 位 IEPE（ICP）传感器专用信号采集卡。IEPE（也称 ICP）传感器是指一种自带电量放大器或电压放大器的特殊传感器，具有优秀的抗噪声性和易封装性，在加速度检测、振动检测或声音检测中被广泛应用。MPS-060602 是专为 IEPE 传感器设计的高性能信号采集卡，其内部集成有恒流源和隔直电路，可无需外部电路直接驱动 IEPE 传感器。通过高速高分辨率的数据采集单元，MPS-060602 可以对各种 IEPE 传感器信号进行全面而精确的分析。

MPS-060602 采用 USB2.0 高速总线接口，采用 USB 自供电，无需外部电源。其内部包含两路同步的高性能 16 位 ADC，采样率高达每通道 450K（每秒 45 万个样点）。MPS-060602 内部包含两路 4mA 输出的恒流源，可为 IEPE 传感器提供恒流激励，并且内置隔直电路，可消除传感器直流偏移电压分量的影响。MPS-060602 还内置了可编程增益放大器，可用来对信号进行放大或衰减，从而获得最佳的采集效果。MPS-060602 采用全金属外壳，具有完整的电磁屏蔽层，抗干扰性强。

MPS-060602 采用跨平台的动态链接库提供驱动函数接口，可工作在 Win9X/Me、Win2000/XP、Windows 7 等常用操作系统下，支持 VB, VC, C++Builder, Dephi，LabVIEW，Matlab 等绝大多数编程语言。此外，MPS-060602 还附送了相应的配套应用软件，可实现高速信号触发采集、滤波处理和数据记录等多项高级功能，一些基本应用可以无限编程直接实现，为用户测试板卡性能提供了便利。

二、 性能指标

2.1、USB 总线

- USB2.0 高速总线传输
- 支持热插拔和即插即用

2.2、IEPE 驱动单元

- 24V 驱动电压
- 4mA 恒流输出
- 10μF 隔直电容

2.3、采样通道

- SMA 插头输入
- 双通道同步采样

- 100KHz 低通滤波
- 低零点偏移误差

2.4、分辨率

- 16 位（65536）

2.5、采样率

- 每通道 1K - 450K
- 可软件编程设置

2.6、程控放大器（PGA）

- PGA = 1、2、5、10
- 所有通道 PGA 相同

2.7、量程

- 量程 = ±10V/PGA

2.8、工作温度

- 0℃ - 70℃

三、 *应用领域*

IEPE 传感器信号采集
便携式仪表和测试设备
振动信号分析
音频信号采集与处理
教学仪器等

四、 软件资源

Windows95/98/NT/2000/XP/WIN7/WIN8 下的驱动程序（支持 32 位和 64 位系统）；通用 DLL 动态链接库；LabVIEW、VB、VC 环境下的编程参考代码；配套应用软件等。其他资源参见网站信息。

五、 配件清单

[1] MPS-060602 信号采集卡一套；
[2] 高屏蔽 USB 延长线一根；
[3] 保修卡一张；
[4] SMA 传感器信号线（选配）；

六、 售后服务

一周退换，一年保修。

第二章 设备安装

一、 MPS-060602 信号采集卡硬件接口

USB: 接至计算机的 USB 总线插口

IN1: 传感器输入通道一，接专用 SMA 信号线

IN2: 传感器输入通道二，接专用 SMA 信号线

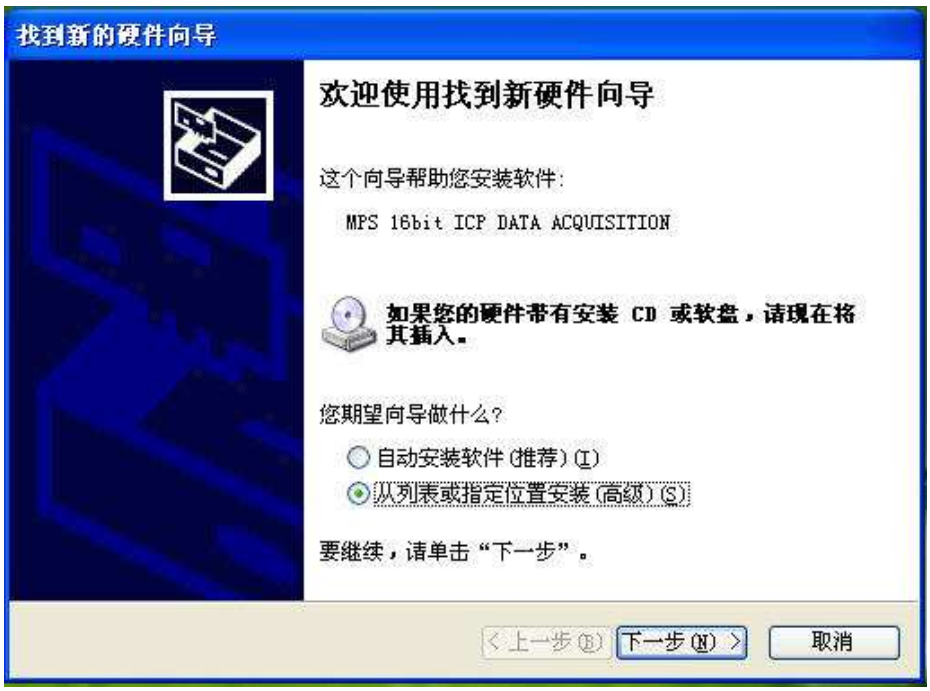
IND: 工作状态指示灯

二、 接口说明

- 使用设备时请通过 USB 数据线将设备与计算机连接，设备正常连接后，可在 WINDOWS 设备管理器中找到设备。
- 进行信号采集时，须先将传感器连接到设备，再将设备连接到计算机。
- 设备的指示灯 LED (IND)在设备上电时会闪烁一次以指示设备是否成功自检，此后 LED 将作为当前数据采集的状态指示，当设备正常工作时，LED 亮表示正在进行数据采集，LED 灭表示采集停止或数据传输中断。

三、 驱动安装

1. 首次使用 MPS-060602 时，计算机将提示“发现新硬件”，如下图所示。选择“从列表或指定位置安装（高级)”，并点击“下一步”。

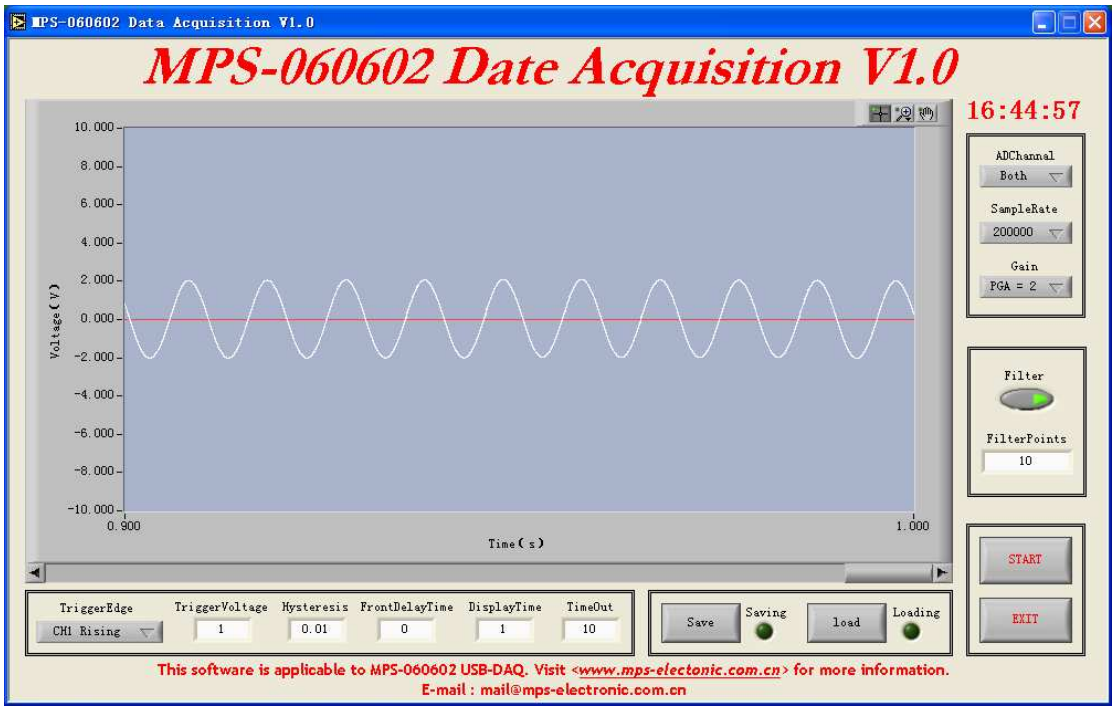


2. 选择“在搜索中包括这个位置”，并点击“浏览”选择 MPS-060602 驱动文件所在的文件夹，点击“下一步”。

3. 开始安装驱动。

4. 驱动安装完成。

四、 功能测试



MPS-060602 信号采集软件 V1.0

1. 将 IEPE 传感器连接至 MPS-060602 输入口。
2. 将 MPS-060602 数据采集卡与计算机 USB 接口连接。
3. 按第三小节所描述的过程安装驱动程序。
4. 下载并解压安装 MPS 应用软件运行库（MPSSoftwareRuntimeLib.rar)。
5. 打开测试程序“MPS-060602 Data Acquisition V1.0.exe”。
6. 出现如上图所示的界面。

7. 右侧“ADChannel”选项用以设置连接传感器的输入通道，可以设置为 IN1、IN2 或双通道；“SampleRate”选项用以设置采样率；“Gain”用以设置硬件增益放大器的放大倍数；“Filter”开关用以使能软件内置的平滑数字滤波器；“FilterPoints”用来设置滤波器的滤波参数，值越大滤波效果越好。

8. 底部“TriggerEdge”用以设置对信号进行触发操作的边沿模式，可设置为无触发、IN1 上升沿触发、IN1 下降沿触发、IN2 上升沿触发或 IN2 下降沿触发；“TriggerVoltage”用以设置触发电平；“Hysteresis”用以设置触发的回滞值；“FrontDelayTime”用以设置触发前的预采集时长；“DisplayTime”用以设置波形图中显示的时长；“TimeOut”用以设置触发超时时限。

9. 点击“START”开始采集。如未启用触发，波形图中立刻开始显示采集到的信号波形；如启用了触发，则在触发信号点出现时开始显示信号波形。

10. 一次采集周期结束后，波形显示停止更新。可通过修改波形图 Y 轴的边界值来修改 Y 轴的显示区间，通过修改 X 轴边界值或拖动 X 轴底部的滑块来选择 X 轴显示的区间。也可通过波形图右上角的显示设置工具来对进行调整，获得最佳的显示效果。

11. 点击“Save”可以将当前显示的信号波形保存到计算机文件，按钮旁边的绿色指示灯亮起时表明记录正在进行，指示灯熄灭表示记录完成；点击“Load”可从已记录的文件中读取数据进行回放显示。

12. 点击“Exit”退出程序。

13. 软件运行时，如果检测到硬件连接异常会提示“DeviceError!”并自动退出，出现异常报警后，用户需检查是否正确连接了板卡并安装了驱动程序。

14. 如上示意图中所示为单路加速度传感器采集到的激振器产生的震荡(1000Hz)信号波形。软件目录下有相应的数据记录文件，可通过回放以还原信号，来对系统的性能进行评估。

第三章 用户编程

一、 *动态链接库（DLL）*

MPS-060602 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-060602 提供的驱动 DLL 文件名为 MPS-060602.dll，软件程序员只需调用该 DLL 内的驱动函数即可实现对板卡的控制，编程与打包发布时该 DLL 必须被包含在工程内。

MPS-060602.dll 内部的驱动函数如下：

- HANDLE MPS_OpenDevice (int DeviceNumber)

HANDLE MPS_OpenDevice 函数执行打开设备的功能。如设备打开成功将返回设备的句柄；打开失败则返回-1。

int DeviceNumber 当前打开设备的序号。当有多套 MPS-060602 设备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从 0 到 9 依次分配序号，打开特定序号的设备时将返回设备的句柄，该句柄将用后续操作。最多支持同时连接 10 个设备。只有一块卡连接时，默认设备号为 0。

- int MPS_Configure (int ADChannel, int ADSampleRate, int Gain, HANDLE DeviceHandle)

int MPS_Configure 函数执行设定设备参数的功能。若函数执行成功，返回 1；执行失败返回 0。

int ADChannel：输入通道选通模式。若 ADChannel = 0，IN1 与 IN2 均未被选通，信号采集功能被禁止，此时不能调用“MPS_DataIn”函数；若 ADChannel = 1，IN1 输入被选通，MPS_DataIn 函数采集 IN1 端口所输入的信号；若 ADCEnable = 2，IN2 端口被选通，MPS_DataIn 函数采集 IN2 端口所输入的信号；若 ADCEnable = 3，IN1 与 IN2 同时被选通,MPS_DataIn 函数同时采集 IN1 端口与 IN2 端口所输入的信号;若 ADChannel = 4，IN1 与 IN2 同时被选通，并且工作在差分模式下，MPS_DataIn 函数采集到的是 IN1 端口与 IN2 端口输入信号的差值；当 ADChannel 取其他值时，ADChannel 自动配置为 0。

int ADSampleRate：信号采集采样率。即每秒内每个信号输入通道所采集的样点个数。取值范围为 1000 到 450000 之间，且应为 1000 的整倍数，当设定其他值时，函数会将设定值自动校正为小于并且最接近该值的有效值。

int Gain：板载硬件可编程增益放大器（PGA）放大倍数索引。Gain 取值为 0-3，分别对应从低到高的四种 PGA 放大倍数。PGA 不同对应的量程也不同。Gain = 0，PGA =

1，量程为正负 10V；Gain = 1，PGA = 2，量程为正负 5V；Gain = 2，PGA = 5，量程为正负 2V；Gain = 3，PGA = 10，量程为正负 1V；Gain 取其他值时自动配置为 Gain = 0。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_Start (HANDLE DeviceHandle)

int MPS_Start 函数执行启动采集的功能。若函数执行成功，返回 1；执行失败返回 0。板卡在待机状态时不能调用 MPS_DataIn 函数，只有在调用 MPS_Start 后（并且已经通过 MPS_Configure 函数选通了某个输入通道），才能通过调用 MPS_DataIn 函数获取数据，否则会导致 MPS_DataIn 无法正常执行。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_DataIn(unsigned short * DataBuffer, int SampleNumber , HANDLE DeviceHandle)

int MPS_DataIn 函数执行获取采集到的数据的功能。若函数执行成功，返回 1；执行失败返回 0。当板卡处于待机状态或 ADChannel = 0 时，不能调用该函数，否则函数可能会无响应。

unsigned short * DataBuffer：保存所采集到数据的缓存区的首地址。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。

DataBuffer 为一个一维数组，其每个元素对应一个采样点的数据，每个采样点数据是一个 unsigned short 型的 16 位整型数据。当 ADChannel = 1 时，DataBuffer 的每个样点依次代表 IN1 的一个样点数据；当 ADChannel = 2 时，DataBuffer 的每个样点依次代表 IN2 的一个样点数据；当 ADChannel = 3 时， DataBuffer 的第奇数个样点依次代表 IN1 的样点数据，第偶数个样点依次代表 IN2 的样点数据，如 DataBuffer[0]代表 IN1 的第一个样点数据，DataBuffer[1]代表 IN2 的第一个样点数据，DataBuffer[2]代表 IN1 的第二个样点数据，DataBuffer[3]代表 IN2 的第二个样点数据……依次类推；当 ADChannel = 4 时，DataBuffer 的每个样点依次代表一个 IN1 与 IN2 的差值数据。当函数成功执行后，DataBuffer 的前 SampleNumber 个数据将被更新为所采集到的数据。

采集到的样点值与真实电压值的对应关系为

Voltage1[0] = 量程 - ((double)DataBuffer[0] / 65536) * 量程 * 2

int SampleNumber：执行一次 MPS_DataIn 函数所获取的样点个数。该参数决定函数执行一次数据数组中所更新的数据个数，若 ADChannel = 1、2 或 4，函数从所设定的输入通道中读到 SampleNumber 个数据点后将成功返回；若 ADChannel = 3 时，函数从每个的输入通道中读到（SampleNumber/2）个数据点后将成功返回。该参数的最小值为 256，最大值为 32768，设定值必须为 256 的整倍数，其他设定值将导致函数执行失败。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_Stop (HANDLE DeviceHandle)

int MPS_Stop 函数执行停止板卡工作的功能。若函数执行成功，返回 1；执行失败返回 0。该函数执行后，在下一次执行 MPS_Start 函数之前，板卡处于待机状态，待机状态下不能执行 MPS_DataIn，否则无法响应。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_CloseDevice (HANDLE DeviceHandle)

int MPS_Close 函数执行关闭设备的功能。若函数执行成功，返回 1；执行失败返回 0。设备打开并执行完所有操作后，必须对设备进行关闭。若没有经过关闭而意外退出，必须关闭所有与设备操作函数相关的程序线程，再重新连接设备硬件。

HANDLE DeviceHandle：操作所针对的设备句柄。

三、 编程范例

一个采集过程的编程流程：

OpenDevice —> Configure —> Start —> 循环 DataIn—> Stop —> CloseDevice

```
#define NOInput 0           // No input channel
#define EnableIN1 1         // IN1 is enabled as input channel
#define EnableIN2 2         // IN2 is enabled as input channel
#define EnableIN1andIN2 3   // IN1 and IN2 are enabled as two input channels
#define Differential 4      // IN1 and IN2 are enabled as a couple of differential input channel

#define DEF_450Ksps     450000
#define DEF_250Ksps     250000
#define DEF_100Ksps     100000
#define DEF_50Ksps      50000
#define DEF_25Ksps      25000
#define DEF_10Ksps      10000
#define DEF_5Ksps       5000
#define DEF_2Ksps       2000
#define DEF_1Ksps       1000

#define DEF_Gain_1      0
#define DEF_Gain_2      1
#define DEF_Gain_5      2
#define DEF_Gain_10     3

#define DEF_Range_Gain1     10
#define DEF_Range_Gain2     5
#define DEF_Range_Gain5     2
#define DEF_Range_Gain10    1

int TEST()
{
    //Declare function in DLL.
```

```
HINSTANCE hDll;                                          //open DLL.
hDll=LoadLibrary("MPS-060602.dll");
if(NULL==hDll)
{
     AfxMessageBox("Can't find DLL");
     return 0;
}

typedef HANDLE(*lpOpenDevice)(int DeviceNumber);    //function "OpenDevice".
lpOpenDevice OpenDevice=(lpOpenDevice)GetProcAddress(hDll,"MPS_OpenDevice");
if(NULL==OpenDevice)
{
     AfxMessageBox("Can't find <MPS_OpenDevice> function");
}

typedef int(*lpCloseDevice)(HANDLE DeviceHandle);   //function "CloseDevice".
lpCloseDevice CloseDevice=(lpCloseDevice)GetProcAddress(hDll,"MPS_CloseDevice");
if(NULL==CloseDevice)
{
     AfxMessageBox("Can't find <MPS_CloseDevice> function");
}

typedef int(*lpStart)(HANDLE DeviceHandle);         //function "Start".
lpStart Start=(lpStart)GetProcAddress(hDll,"MPS_Start");
if(NULL==Start)
{
     AfxMessageBox("Can't find <MPS_Start> function");
}

typedef int(*lpStop)(HANDLE DeviceHandle);          //function "Stop".
lpStop Stop=(lpStop)GetProcAddress(hDll,"MPS_Stop");
if(NULL==Stop)
{
     AfxMessageBox("Can't find <MPS_Stop> function");
}

typedef int(*lpConfigure)(int ADChannel, int ADSampleRate, int Gain, HANDLE DeviceHandle);
                                                         //function "Configure".
lpConfigure Configure=(lpConfigure)GetProcAddress(hDll,"MPS_Configure");
if(NULL==Configure)
{
     AfxMessageBox("Can't find <MPS_Configure> function");
}
```

```
typedef int(*lpDataIn)(unsigned short *DataArray,int SampleNumber, HANDLE DeviceHandle);
                                                    //function "DataIn".
lpDataIn DataIn=(lpDataIn)GetProcAddress(hDll,"MPS_DataIn");
if(NULL==DataIn)
{
     AfxMessageBox("Can't find <MPS_DataIn> function");
}

//main program.
HANDLE DeviceHandle;                                // Hhandle of device.
unsigned short DataInBuffer[1024] = {0};            // Temporary array to hold data returned
from "DataIn" function.
double VoltageArrayIN1[512] = {0};                  // Array of voltage reciving from IN1.
double VoltageArrayIN2[512] = {0};                  // Array of voltage reciving from IN2.
int flag = 0;                                       // Flag to indicate whether function is
successful.

DeviceHandle = OpenDevice(0);                       // Open device .
if(DeviceHandle == (HANDLE)-1)                      // If open faild, send a message and return.
{
     AfxMessageBox("OpenDeviceError");
     return 0;
}

Configure(EnableIN1andIN2, DEF_100Ksps, DEF_Gain_1, DeviceHandle);// Configure the device
- enable all input, set input samplerate as 100K, set gain as 1.

Start(DeviceHandle);                                // Start input.


for(int i = 0 ; i < 3; i++)                         // Input several times.
{
     flag = DataIn(DataInBuffer,1024,DeviceHandle); // Input.

     if(flag == 0)AfxMessageBox("Error!");          // If input faild, send a message.

     else                                           // If successed, convert 16bit ADC data to
voltage.
     {
          for(int j = 0; j < 512; j++)
          {
               VoltageArrayIN1[j] = (double) DataInBuffer[j*2]*DEF_Range_Gain1*2/65536
- DEF_Range_Gain1;      // Data of IN1 channel - odd elements in DataInBuffer.
```

```
                    VoltageArrayIN2[j] = (double) DataInBuffer[j * 2 + 1] * DEF_Range_Gain1 * 2
/ 65536 - DEF_Range_Gain1;  // Data of IN2 channel - even elements in DataInBuffer.
                }
                AfxMessageBox("Success!");            // Send a message.
            }
        }

        Stop(DeviceHandle);                           // Stop input and output.
        CloseDevice(DeviceHandle);                    // Close device to end operation cycle.
        return flag;                                  // Return the flag.
    }
```

第四章 注意事项

- 拔插设备接线端口请用力适度，以免损害接口。
- 信号输入端允许接入 IEPE 传感器，驱动电压为 24V，驱动电流为 4mA，非 IEPE 传感器或参数不同的传感器请谨慎接入，以免损坏设备或传感器。
- 将采集卡拔离计算机后，请间隔一定时间再重新插入。拔插过快有可能造成计算机系统配置驱动异常，此时请重新拔插采集卡。
- 本设备属于精密电子仪器，请注意妥善保管，注意防尘防潮与防静电。
- 禁止用户自行打开设备外壳，一经打开即失去保修资格，出现的故障与其他后果由用户自行承担。
- MPS-060602 自出厂之日起一年内，凡用户遵守贮存、运输和使用要求，由于产品质量导致的故障，凭保修卡免费维修。因违反操作规定和使用要求而造成损坏的，需交纳器件维修费。
- MPS 系列信号采集卡由 Morpheus Electronic 提供，更多产品和相关信息请浏览：www.mps-electronic.com.cn,咨询邮箱 mail@mps-electronic.com.cn。