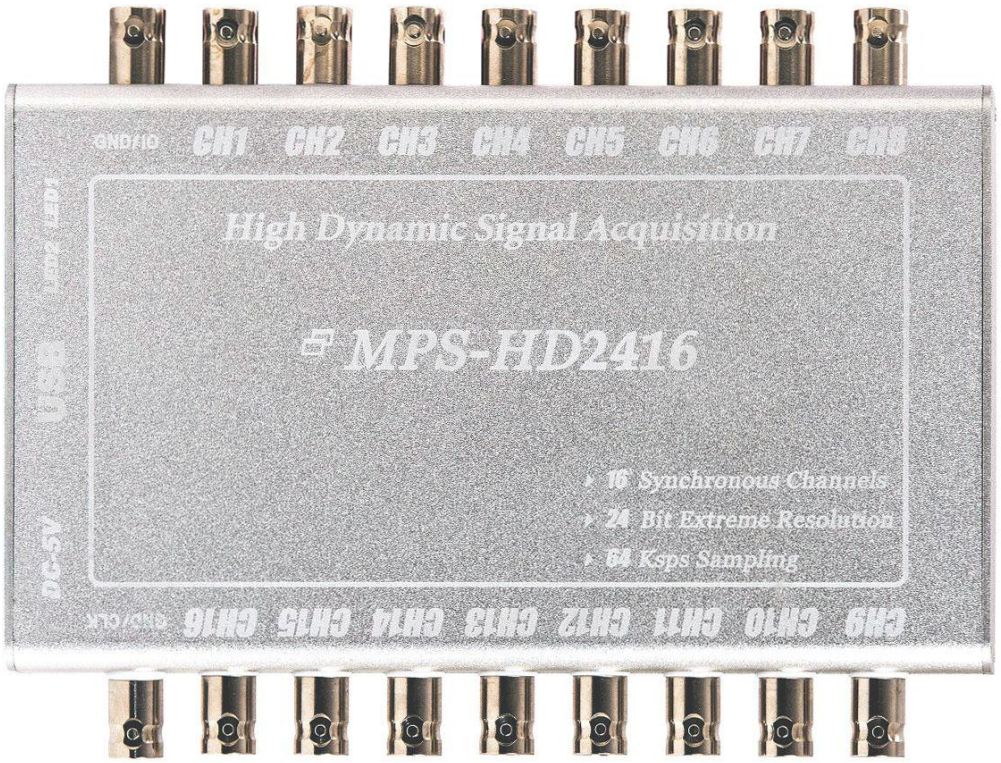
MPS-HD2416

USB 十六通道 24 位高动态范围

信号采集卡使用说明

Ver 1.1.4

第一章 产品概述



MPS-HD2416 十六通道 24 位高动态范围信号采集卡

一、 *产品简介*

MPS-HD2416 是一款 USB 总线的十六通道 24 位高动态范围信号采集卡，可用于高精度信号测量、便携型数据采集、振动与音频信号分析等领域。

MPS-HD2416 采集卡有十六路±5V 量程的同步电压采集通道，采样率从 1K 至 64K 七档可调，可在高速采样下实现不间断的连续实时采集和数据传输。MPS-HD2416 采集卡具有低采样噪声的优点，在 64K 采样率时底噪仅为 66uV*rms*（±5V *满量程*），在 1K 采样率时更可低至 10uV*rms*，有效分辨率达 20.1bit。同时，MPS-HD2416 有六种可选的通道输入模式，支持对包括差分与单端、高输入阻抗与低输入阻抗、直流耦合与交流耦合、是否支持 IEPE（ICP）恒流供电等在内的多种特性进行组合，不同的输入模式可以适配不同的应用场景，每个通道都可独立进行配置，对多种信号的混合测量更加友好。

MPS-HD2416 采集卡采用 USB2.0(High Speed)总线协议与计算机进行连接，支持全系 Windows 操作系统，支持即插即用和热插拔。MPS-HD2416 提供通用型 DLL 驱动，支持在大部分的编程语言下直接调用，并提供有 LabVIEW 等语言下的示例作为编程参考。此外，MPS-HD2416 采集卡还免费附送一套多功能测试软件，测试软件能够实现信号的实时采集与显示、数据记录与回放、频谱图等常用基本功能，也能实现软件滤波、边沿触发、变采样率、计算幅值频率与均值极值等、自动定时采集、信号转音频播放等众多高级功能，从而使不希望自主编程的用户也能利用这套软件来完成应用，大幅降低了设备的使用门槛。

二、 *性能指标*

2.1、通信总线

- USB2.0 高速总线（兼容 3.0 及更高版本）
- USB 总线供电或外接电源供电
- 支持即插即用与热插拔

2.2、输入通道

- 接口类型：BNC 母头
- CH1-CH16 通道：十六路信号输入口
- GND/IO 通道的负极：共地线接地口
- GND/IO 通道的正极：多功能复用 IO 口
- GND/CLK 通道的负极：共地线接地口
- GND/CLK 通道的正极：时钟信号输入/输出口

2.3、输入模式

MPS-HD2416 采集卡支持六种信号输入模式，用户可在订购产品时指定每个通道的输入模式。六种输入模式分别为：

S1 模式（直流单端低输入阻抗模式）：
1、 耦合方式：直流耦合
2、 输入类型：单端输入
3、 输入阻抗：输入正极与地线间 50KΩ【1】；输入负极与地线连通
4、 输入量程：±5V【2】
5、 输入耐压：±25V
6、 恒流供电：无

S2 模式（直流单端高输入阻抗模式）：
1、 耦合方式：直流耦合
2、 输入类型：单端输入
3、 输入阻抗：输入正极与地线间 1MΩ【3】；输入负极与地线连通
4、 输入量程：±5V
5、 输入耐压：±25V
6、 恒流供电：无

S3 模式（直流差分模式）：
1、 耦合方式：直流耦合
2、 输入类型：差分输入
3、 输入阻抗：输入正极与地线间 1MΩ；输入负极与地线间 1MΩ
4、 输入量程：±5V
5、 输入耐压：±25V
6、 恒流供电：无

S4 模式（交流单端模式）：

1、 耦合方式：交流耦合
2、 输入类型：单端输入
3、 输入阻抗：输入正极与地线间 10uF||50KΩ；输入负极与地线连通
4、 输入量程：±5V
5、 输入耐压：±25V
6、 恒流激励：无

S5 模式（恒流供电模式）：
1、 耦合方式：直流耦合
2、 输入类型：单端输入
3、 输入阻抗：输入正极与地线间 1MΩ；输入负极与地线连通
4、 输入量程：±5V
5、 输入耐压：±25V
6、 恒流激励：恒定 4mA[4]；开路驱动电压 24V

S6 模式（IEPE 模式）：
1、 耦合方式：交流耦合
2、 输入类型：单端输入
3、 输入阻抗：输入正极与地线间 10uF||50KΩ；输入负极与地线连通
4、 输入量程：±5V
5、 输入耐压：±25V
6、 恒流激励：恒定 4mA；开路驱动电压 24V

【1】*输入内阻的精确额定值为 49.7KΩ，误差 0.1%。*
【2】*量程的精确额定值为 5.16V，误差 0.1%。*
【3】*输入内阻的精确额定值为 1MΩ，误差 0.1%。*
【4】*恒流激励的精确额定值为 4.15mA，误差 1%。*

2.4、采样率

- 64K、32K、16K、8K、4K、2K、1K 七档可通过软件设置。

2.5、分辨率

- 64K 采样率下，有效分辨率 17.3bit；采样噪声峰峰值 V_{pp} < 435 微伏，有效值 V_{RMS} < 66 微伏；
- 32K 采样率下，有效分辨率 17.7bit；采样噪声峰峰值 V_{pp} < 317 微伏，有效值 V_{RMS} < 48 微伏；
- 16K 采样率下，有效分辨率 18.2bit；采样噪声峰峰值 V_{pp} < 224 微伏，有效值 V_{RMS} < 34 微伏；
- 8K 采样率下，有效分辨率 18.7bit；采样噪声峰峰值 V_{pp} <158 微伏，有效值 V_{RMS} < 24 微伏；
- 4K 采样率下，有效分辨率 19.2bit；采样噪声峰峰值 V_{pp} < 112 微伏，有效值 V_{RMS} < 17 微伏；
- 2K 采样率下，有效分辨率 19.6bit；采样噪声峰峰值 V_{pp} < 86 微伏，有效值 V_{RMS} < 13 微伏；

- 1K 采样率下，有效分辨率 20.1bit；采样噪声峰峰值 V_{pp} < 66 微伏，有效值 V_{RMS} < 10 微伏；

2.6、带宽

- 1K – 64K 采样率下，内置 32KHz 抗混叠滤波，信号通带高截止频率为 31.2KHz；
- S4 或 S6 模式下，内置隔直高通滤波器，信号通带低截止频率为 0.3Hz；
- S1、S2、S3 或 S5 模式下，直流耦合，信号通带低截止频率 0Hz；

2.7、板载缓存

- DAQ Buffer：192K Bytes
- USB FIFO ：1K Bytes

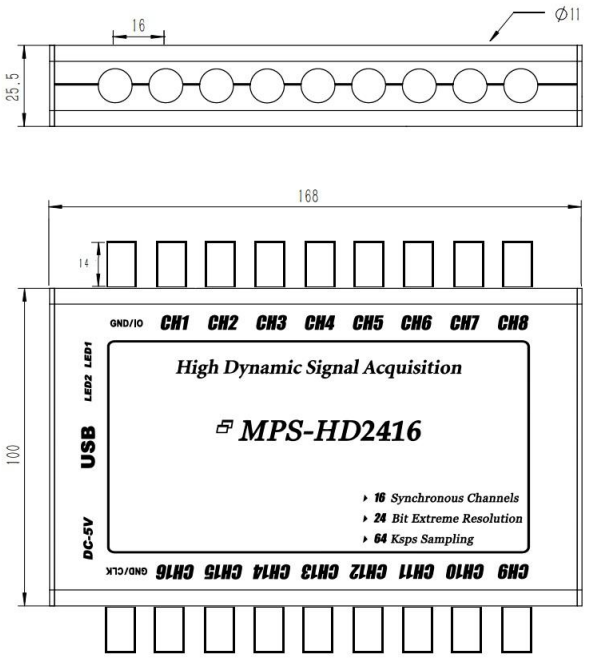
2.8、工作温度

- -40℃ – 85℃

2.9、工作功率

- 待机功率：电流 < 350mA，功率 < 1.75W；
- 运行功率：电流 < 450mA，功率 < 2.25W；
- IEPE 功率：每接入一只 IEPE 传感器，电流增加 25mA，功率增加 0.125W；
- 最大功率：电流 < 800mA，功率< 4W；
- DIO 5V 输出：输出电流 < 50mA；

三、 *外观尺寸*



外观尺寸：168mm * 100mm * 25.5mm

四、 *应用领域*

高精度信号采集与记录
工业生产在线监测
便携式信号测量
声音与振动分析

五、 软件资源

提供支持 Windows 全系操作系统的驱动程序;提供跨编程语言平台通用的 DLL 动态链接库；提供 LabVIEW 语言下的编程参考例程；免费附送一套多功能测试软件。

六、 配件清单

[1] MPS-HD2416 信号采集卡一张；
[2] 高屏蔽 USB 数据传输线一根；
[3] 保修卡一张；
[4] 合格证一张；
[5] 外部电源适配器；

七、 售后服务

免费保修三年。

第二章 设备安装

一、 *接口说明*

USB： USB 数据总线接口。通过使用配套提供的 USB Type-B(方口 USB)数据线将该口连接到计算机主机 USB。USB 连接后，可在计算机 WINDOWS 系统的设备管理器中识别到本设备。设备使用 USB 总线供电，不需其他电源，如果计算机 USB 供电能力欠缺，可使用带外接电源的 USB 集线器来提高供电。

LED1： 设备自检状态指示灯，亮起时呈现蓝色。LED1 常亮，表示设备自检通过，状态正常； LED1 快速闪烁，表示设备自检异常，无法使用，请与售后联系；LED1 熄灭，表示设备供电异常或出现故障，可尝试更换计算机主机和 USB 数据线，若仍无法解决，请与售后联系。

LED2： 采集与数据传输状态的指示灯，亮起时呈现白色。LED2 常亮，表示系统处于连续采集状态并正在进行数据传输；LED2 熄灭，表示设备处于待机状态，或者设备虽然处于采集状态但数据传输已停止；LED2 闪烁，表示设备处于采集状态，但数据传输不连续。当设备在执行连续不间断的采集应用时，LED2 指示灯应表现为常亮，如果发现 LED2 出现闪烁，通常表示软件运行产生了比较严重的卡顿，或者读数速度低于采集速度，导致板载缓存写满，出现了数据覆盖与丢失，此时可尝试改善软件程序的运行效率，或改用使用较低的采样率重新采集。

DC: 外部电源接口。用于连接设备配套提供的外部电源适配器（5V-2A）。***MPS-HD2416 的工作电流可能超过 500mA，若计算机的 USB 口为 2.0 接口（常为黑色 USB），可能存在 USB 自供电不足的情况，需连接外部电源来提供供电；若计算机 USB 为 3.0 及以上接口（常为蓝色 USB 或 TYPEC），通常 USB 自供电充足，可不使用外部电源。***当接入外部电源时，设备会自动断开 USB 供电，仅使用外部电源供电；当拔除外部电源时，设备会自动切换回 USB 总线供电。在接入外部电源前，请提前断开 USB 总线，先接入外部电源，待设备白色指示灯闪烁一次后，再连接 USB 总线。设备断电时，请先断开 USB 连接，然后再断开外部电源。***请务必注意在操作外部电源前，先断开 USB 总线连接，外部电源接口不支持在线热拔插。***

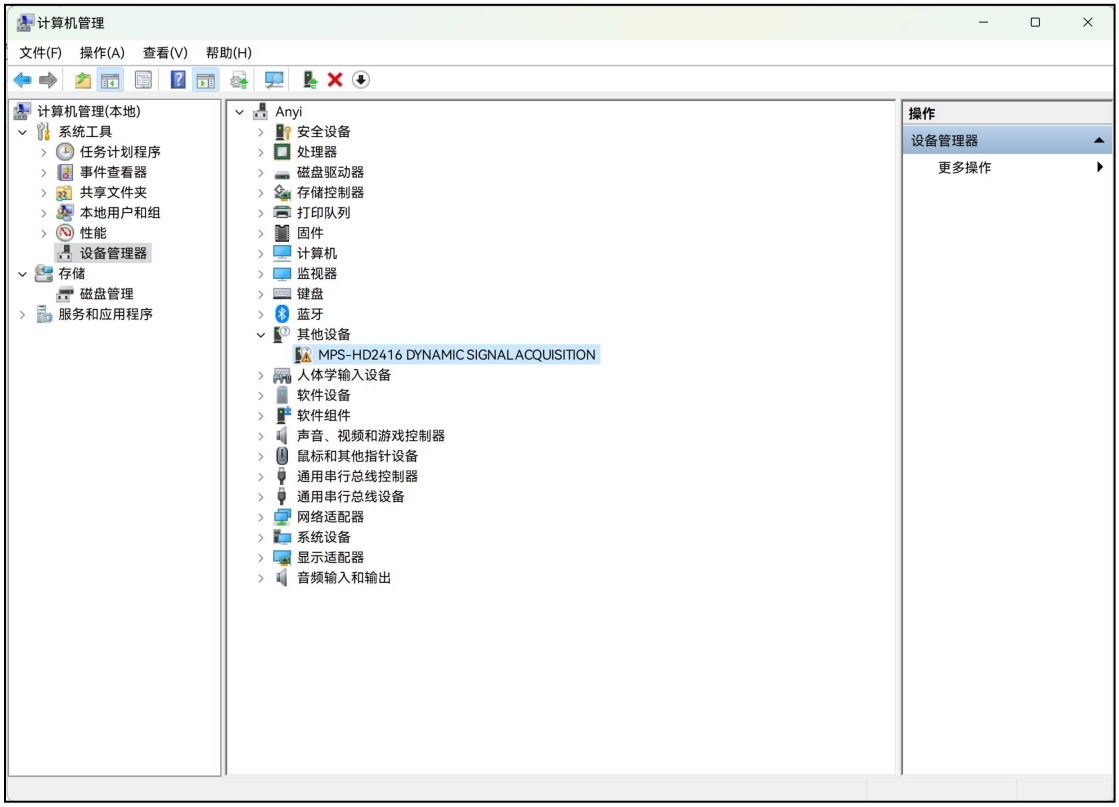
CHx： CH1-CH16 对应通道 1 至通道 16 的信号输入端口。每个输入端口为 BNC 母接头，每个接头的外圈为负极，内芯为正极。不同输入模式下的接口特性可参看第一章 2.3 小节。

GND/IO：设备信号地与复用 IO 口。端口为 BNC 母接头，接头的外圈为设备信号地，内芯为复用 IO 口。设备的信号地用于与外部传感器或信号源进行共地，复用 IO 口可用于对外输出高、低电平，其中高电平也能作为 5V 供电使用（最大供电输出电流 50mA）。同时复用 IO 口还可以作为外部启动触发口来使用，此时通过对该口输入一个下降沿信号，可使设备启动采集。

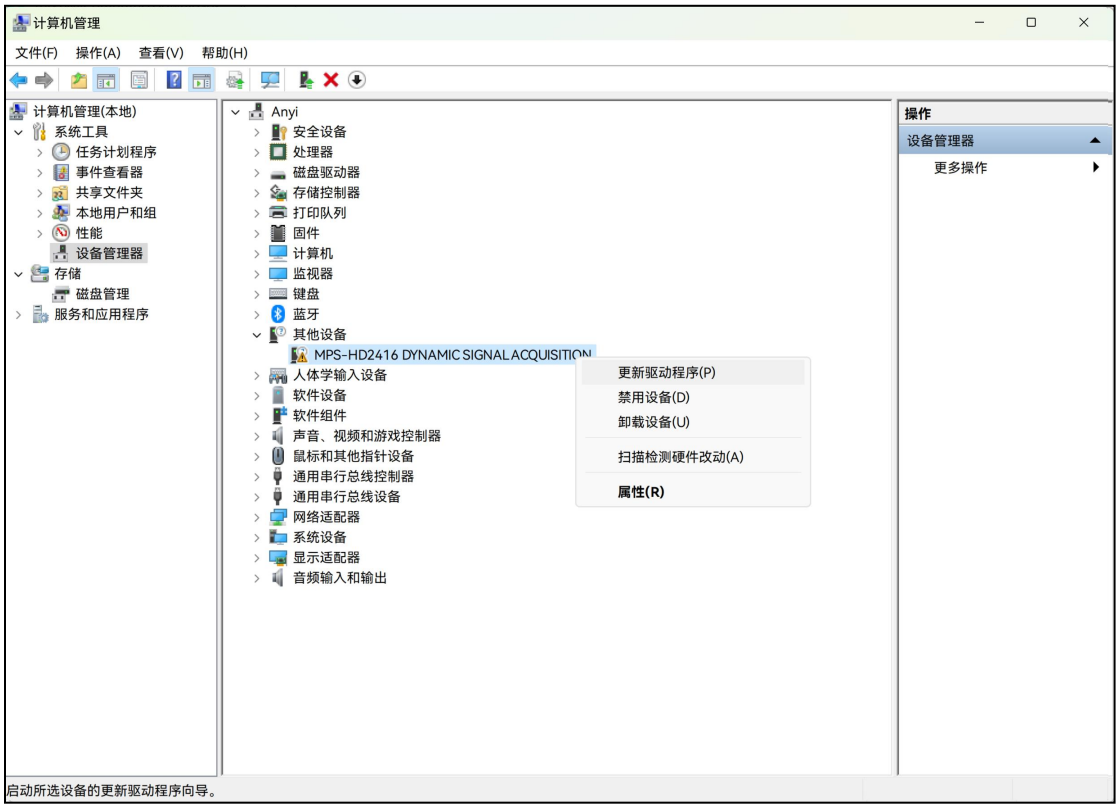
GND/CLK：设备信号地与 CLK 输入/输出口。端口为 BNC 母接头，接头的外圈为设备信号地，内芯为 CLK 输入/输出口。设备的信号地用于与外部传感器或信号源进行共地，CLK 输入/输出口用于将设备的采样时钟 CLK 进行输出或输入。当 CLK 设置为输出时，可将其内部的采样时钟对外输出；当 CLK 设置为输入时，可用于接收另一台 MPS-HD2416 所输出的时钟，并将其作为自身的采样时钟使用。

二、 *驱动安装*

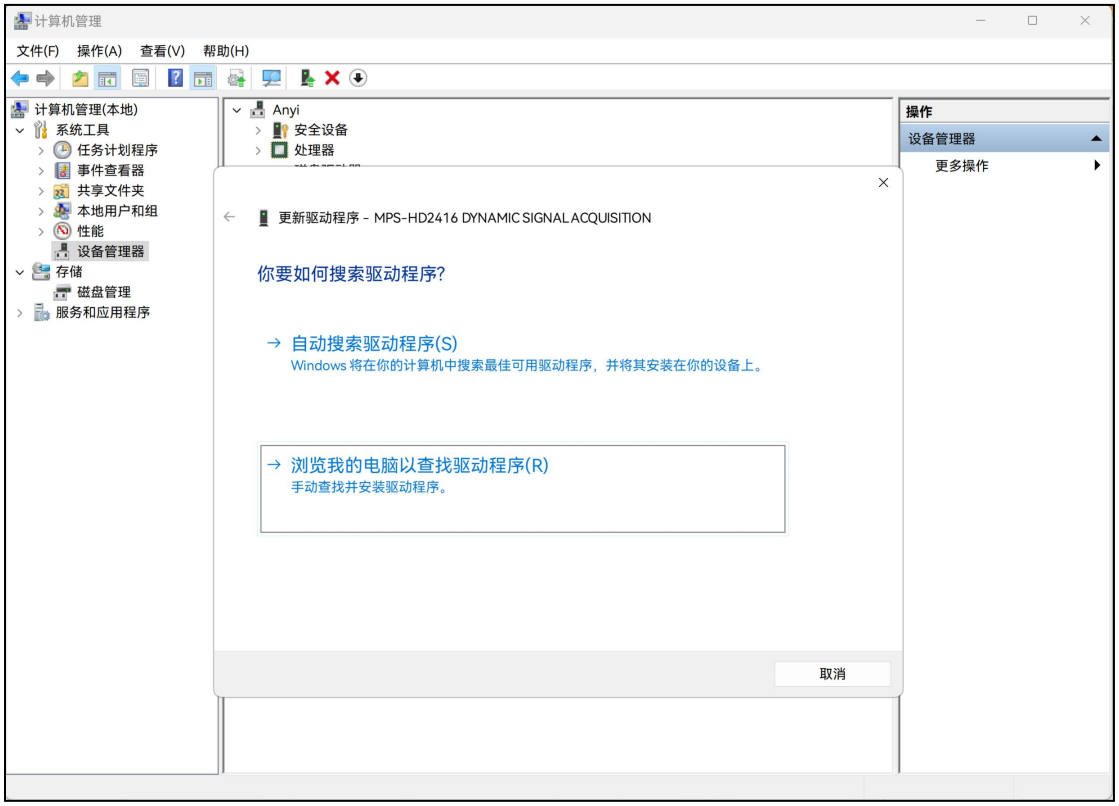
1. 设备接入计算机 USB 后，在“此电脑”上点右键，选择“设备管理器”或者点击“属性”-“管理”-“设备管理器”来打开 WINDOWS 设备管理器，并在“其他设备”下找到本设备。



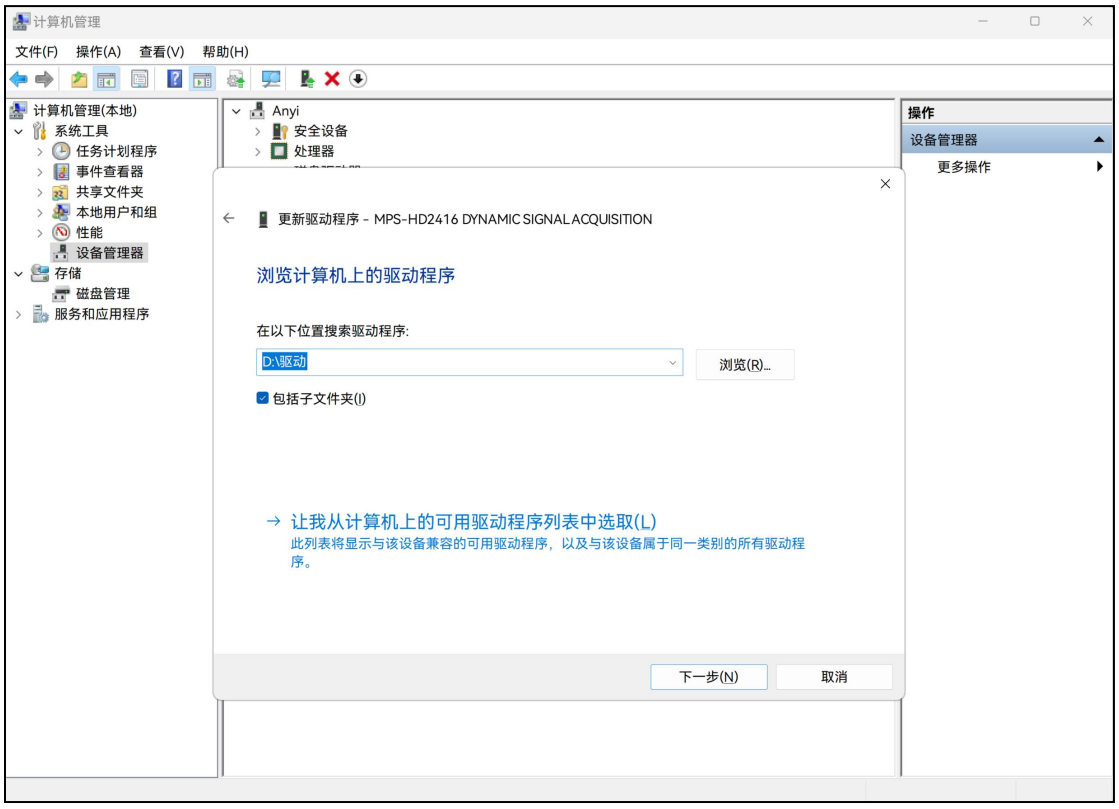
2. 在设备名称上点右键，选择“更新驱动程序”。



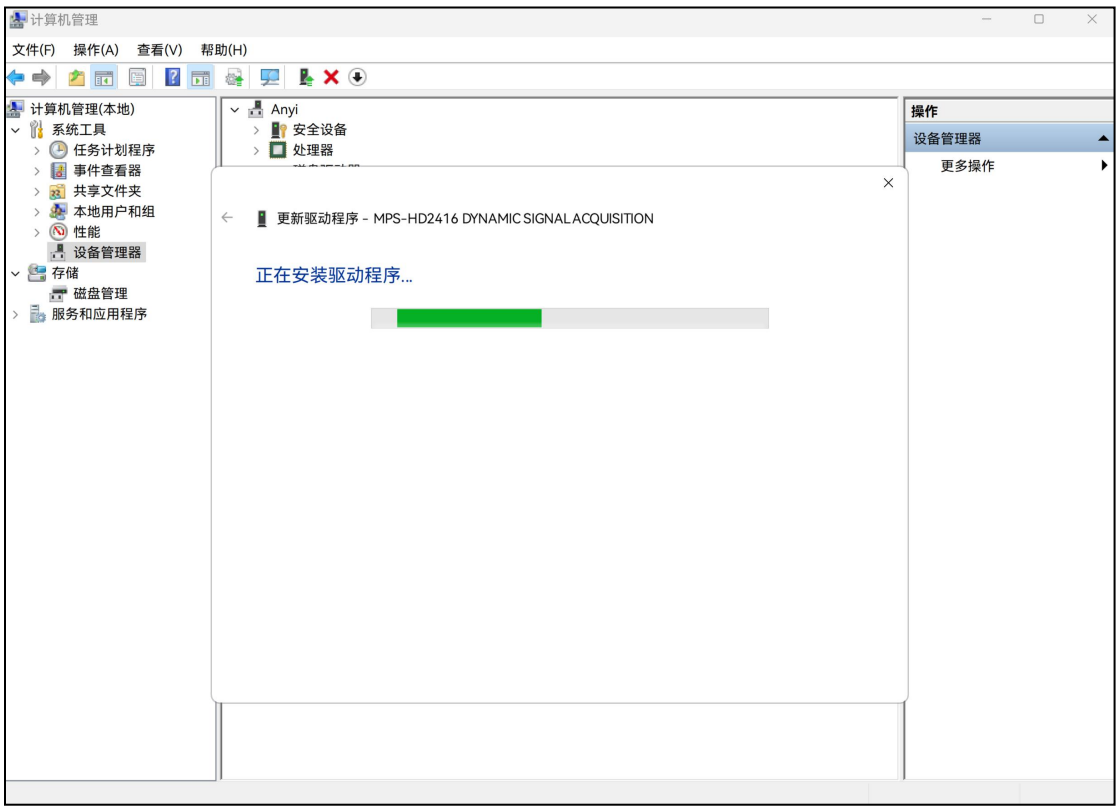
3. 在弹出的驱动安装向导界面中，选择“浏览我的电脑以查找驱动程序”。



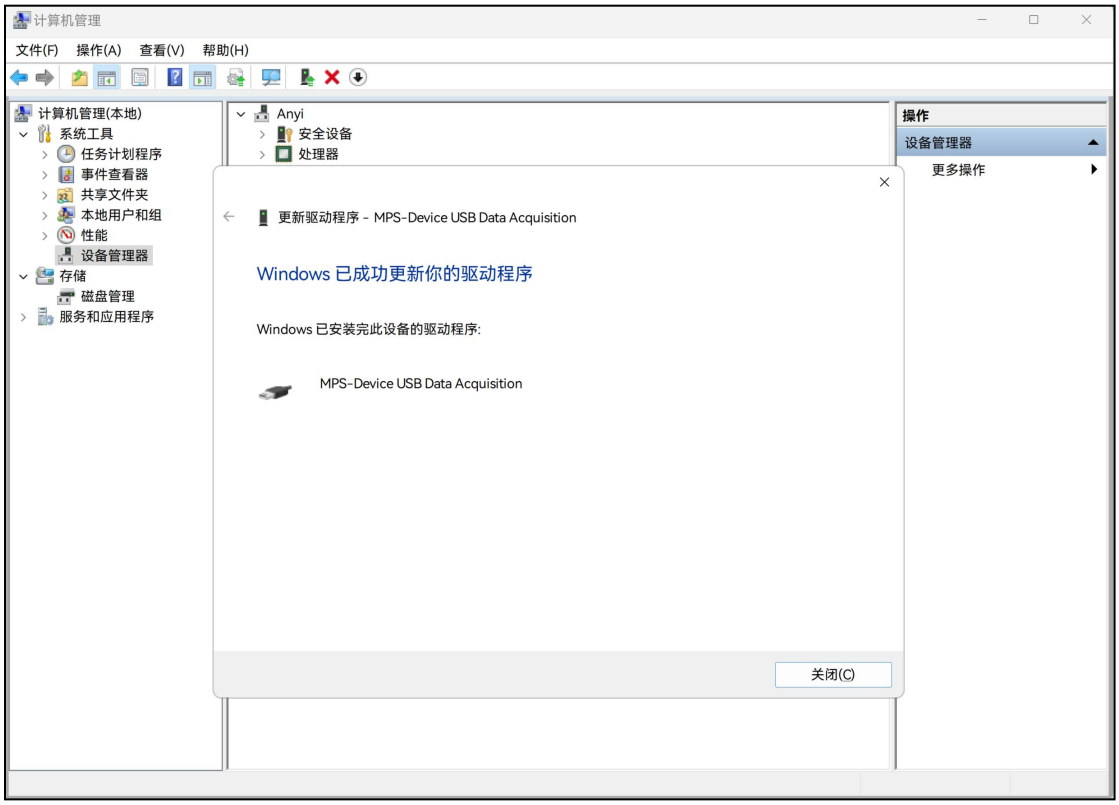
4. 点击“浏览”按钮，选择采集卡驱动所在的文件夹。如果驱动为压缩包，需要先解压缩后再进行操作。



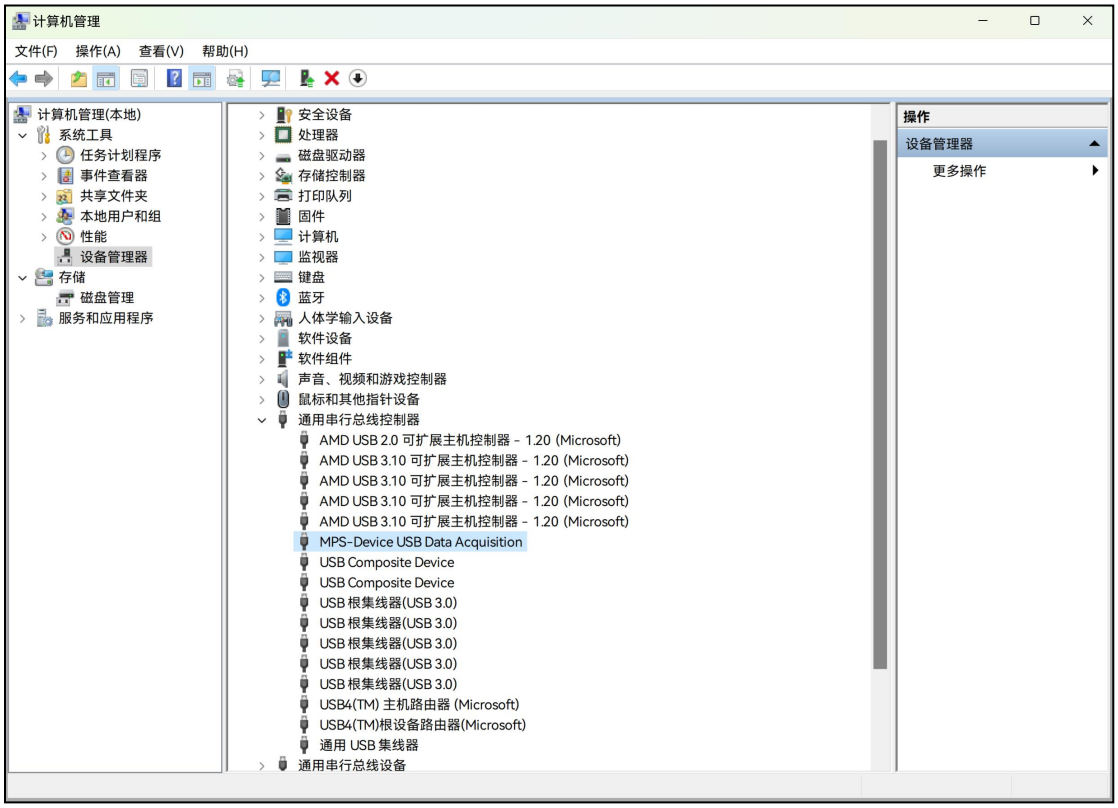
5. 点击下一步，出现如下驱动安装界面。过程中如果出现信任驱动的安全提示，勾选信任选项并点击确认。



6. 驱动安装完成，出现如下提示框。



7. 驱动安装后，可在设备管理器 “通用串行总线控制器”类目下看到设备，名为“MPS-Device USB Data Acquisition”，驱动成功安装。



三、 *信号接入*

MPS-HD2416 有十六个模拟信号输入口 CH1-CH16，分别对应通道 1 至通道 16。输入信号接头为 BNC 母头，可与使用 BNC 公头的信号线进行对接。

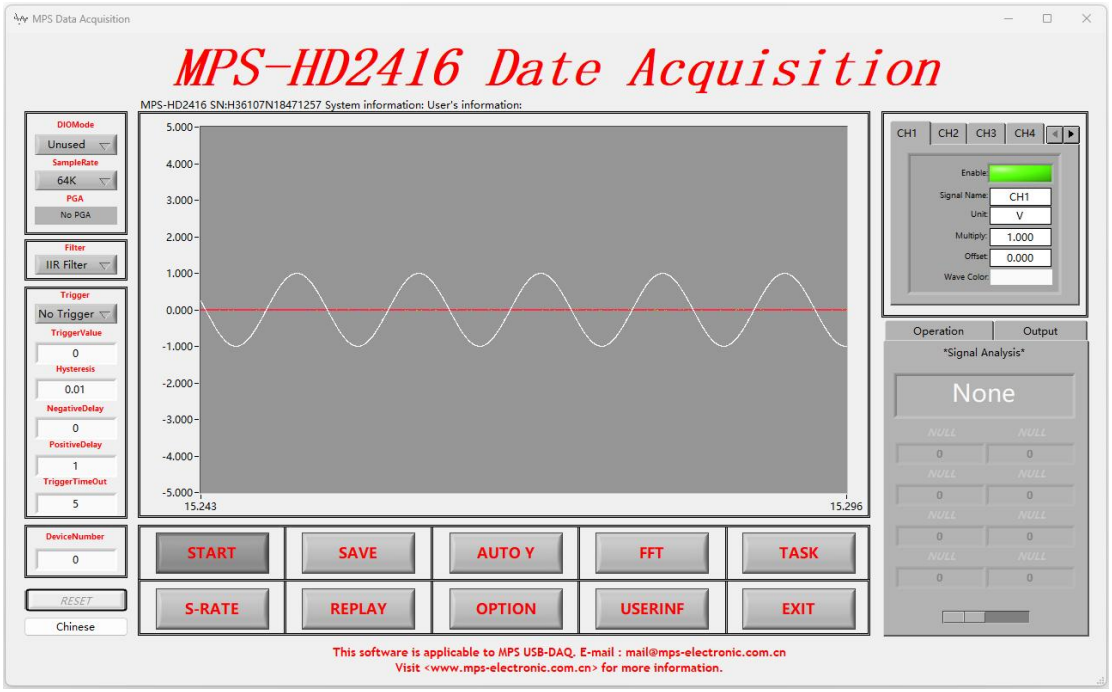
MPS-HD2416 在交付前可对每个通道进行输入模式配置，共六种模式可选（见第一章 2.3 小节），不同的输入模式可匹配不同类型的信号源。交付后如果需要更改模式，可与售后联系进行咨询。

若 MPS-HD2416 的某个通道为 S1、S2 或 S4 模式，该通道为单端输入，输入接头的正极是信号输入，负极是信号共地线。该通道可与外部的单端信号源连接，接头的正极对接单端信号源的信号线，接头的负极对接单端信号源的信号地。

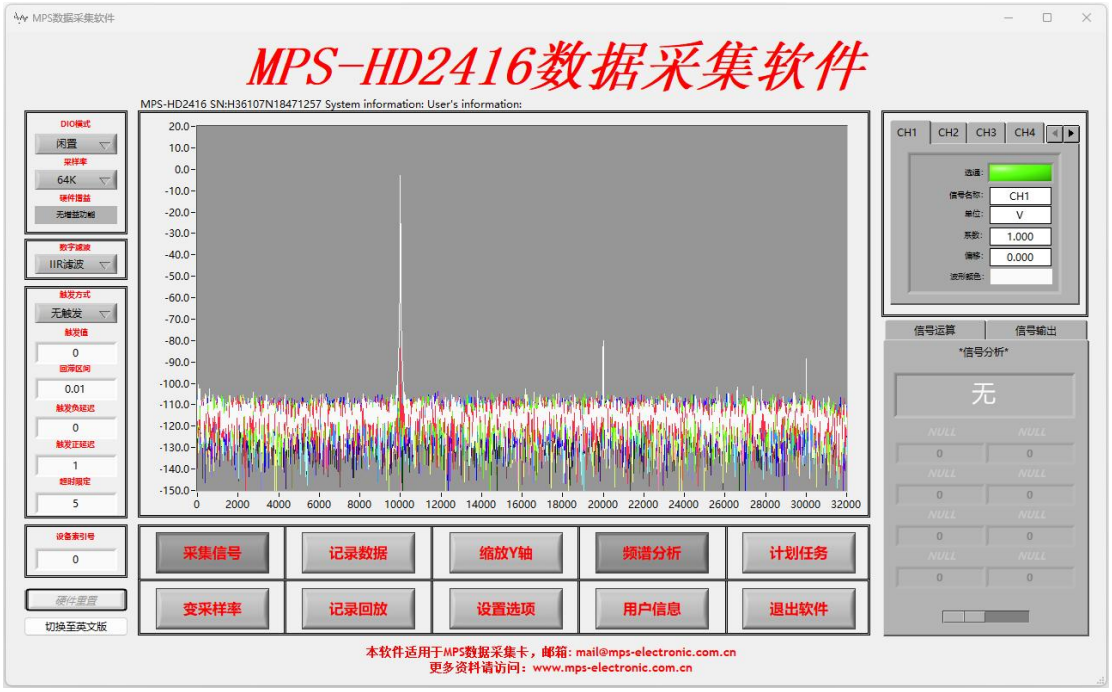
若 MPS-HD2416 的某个通道为 S3 模式，该通道为差分输入，需要与设备的 GND/IO 口负极（信号共地线）联合使用，既可以连接外部的差分信号源，也可以连接单端信号源。连接差分信号源时，差分通道输入接头的正、负极分别对接差分信号源的正、负极，同时 GND/IO 口的负极与差分信号源的独立地线对接进行共地；连接单端信号源时，通道输入正极与单端信号源的信号线连接，通道输入负极与单端信号源的地线连接，同时 GND/IO 口的负极也要与信号源的地线连接以进行共地。

若 MPS-HD2416 的某个通道为 S5 或 S6 模式，该通道为单端输入，输入接头的正极是信号输入，同时也会提供 4mA 的对外恒流输出；接头的负极是信号共地线。该通道可与外部的 IEPE 传感器（S6 模式）、阻值 1250 欧姆以内的电阻型传感器（S5 模式）和某些特定的传感器进行连接。其他类型的传感器一般不能接入，强行接入可能导致传感器无法正常工作，并有损坏传感器的风险。当使用 S6 模式时，通道可以直接对接 IEPE 型传感器，中间不需要再加入额外的信号调理器等辅助设备。

四、 功能测试



MPS 测试软件英文界面（100Hz 正弦波输入状态下波形图）



MPS 测试软件中文界面（10KHz 正弦波输入状态下频谱图）

1. 下载并解压 MPS Data Acquisition Installer.zip，执行“Setup.exe”安装测试软件。
2. 将 MPS-HD2416 信号采集卡与计算机通过 USB 接口连接，首次接入需要按第 3 小节的步骤来安装硬件驱动。
3. 打开 Windows 的开始菜单，打开名为“MPS Data Acquisition”（ 图标为）的软件，软件界面如上图所示。

4. 左上角“SampleRate”处可设置采样率，从 1K 到 64K 七档可选。
5. 点击“START”，可从波形图中看到所采集的信号波形曲线，未连接信号源时，曲线一般呈现为一条接近于 0 的直线。点击“START”后如果出现报错提示，请检查硬件是否接入，以及驱动是否正确安装。
6. 将传感器或信号源接入采集卡后，可观察到随信号源变化的波形曲线。其中白色曲线对应 CH1，红色曲线对应 CH2，绿色曲线对应 CH3，浅蓝色曲线对应 CH4，黄色曲线对应 CH5，浅紫色曲线对应 CH6，粉色曲线对应 CH7，深蓝色曲线对应 CH8，深紫色曲线对应 CH9，蓝白色曲线对应 CH10，浅黄色曲线对应 CH11，棕色曲线对应 CH12，黄绿色曲线对应 CH13，蓝紫色曲线对应 CH14，深绿色曲线对应 CH15，灰黑色曲线对应 CH16。
7. 点击“AUTO Y”可使 Y 轴显示范围自动与当前曲线匹配，直接修改 Y 轴坐标边界值也可以改变显示范围。
8. 软件中数字滤波、软件触发、信号记录、频谱分析、计划任务、变采样率、记录回放、信号分析、运算、输出等附加功能都可供选用。点击左下角“Chinese”可以切换为中文软件界面。
9. 功能测试结束，点击“EXIT”退出软件。
10. 断开信号源，并拔出连接计算机的 USB 插头，测试完成。

第三章 用户编程

一、 *动态链接库（DLL）*

MPS-HD2416 采用 DLL（Dynamic Linkable Library，动态链接库)的方式来进行编程驱动。DLL 与具体的编程语言及编译器无强制关联，只要遵循约定的 DLL 接口规范和调用方式，用各种语言都可以调用 DLL。

以 VC、VB、LabVIEW 等语言下调用 DLL 为例，具体调用方式分别为：

- 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 中的后续参数都必须是可选的，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 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链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、 *驱动函数及参数*

MPS-HD2416 提供文件名为 MPS Driver.dll 的驱动文件，内含十一个驱动函数，分别为：

#define HANDLE int
注：如果未对 HANDL 数据类型进行预定义，可对其赋予 int 类型定义。

- HANDLE MPS_OpenDevice (int DeviceNumber)

HANDLE MPS_OpenDevice 函数执行打开设备并获得设备控制句柄的功能。函数运行后，打开设备成功返回设备的控制句柄，打开失败则返回-1。

int DeviceNumber 打开设备的序号，取值范围为 0-9。调用 MPS_OpenDevice 函数时使用不同的序号作为参数，可获得不同设备的句柄，句柄将作为调用其他函数时的必要参数。

MPS-Device 驱动支持多台设备同时连接到一台计算机，当多个 MPS-Device 设备同时连接时，WINDOWS 将按照设备连接的先后顺序为设备分配 DeviceNumber 序号，第一个接入的设备序号为 0，第二个接入的设备序号为 1……以此类推，最大为 9。当只连接一个设备时，DeviceNumber 应设置为 0。

- int MPS_Configure(int SampleRate, int DIOMode, HANDLE DeviceHandle)

int MPS_Configure 函数执行设置设备参数的功能。函数执行成功返回 1，执行失败返回 0。MPS_Configure 函数应在设备处于待机状态时调用，调用成功后，将设置设备的采样率和复用 IO 口的状态。

int SampleRate：MPS-HD2416 的采样率设置参数，取值规则如下：

该值为 64000 或大于 64000 时，采样率设为每秒 64000 点；
该值为 32000 或小于 64000 且大于 32000 时，采样率设为每秒 32000 点；
该值为 16000 或小于 32000 且大于 16000 时，采样率设为每秒 16000 点；
该值为 8000 或小于 16000 且大于 8000 时，采样率设为每秒 8000 点；

该值为 4000 或小于 8000 且大于 4000 时，采样率设为每秒 4000 点；
该值为 2000 或小于 4000 且大于 2000 时，采样率设为每秒 2000 点；
该值为 1000 或小于 2000 时，采样率设为每秒 1000 点；

int DIOMode：MPS-HD2416 的复用 IO 口与 CLK 口的设置参数，取值可为 0、1、2、22 或 32。

当 DIOMode 设置为 0、1、或 2 时，用于单独设置复用 IO 口。复用 IO 口可设置为三种模式：DIOMode 为 0 时，复用 IO 口设置为闲置模式，呈现高内阻的低电平状态（约 0.5V）；DIOMode 为 1 时，复用 IO 口设置为高电平输出模式，对外输出 5V 电压，且具备 50mA 以内的电流输出能力，可作为数字高电平输出或 5V 供电输出使用；DIOMode 为 2 时，复用 IO 口设置为外部启动模式，呈现弱上拉的电平输入状态，该模式可用来实现外部下降沿触发采集，或者实现多个设备同步启动的功能。三种模式中，闲置模式和高电平输出模式设置后持续有效，外启动模式设置后仅在当前的待机状态中有效，一经启动采集，复用 IO 口自动恢复为闲置模式。

关于外启动模式的补充说明：

外部启动模式下，当将复用 IO 口与 GND 短接，或者将该口的输入从高电平（2V-5V）转为低电平（0V-1.5V）时，复用 IO 口会收到一个电平下降沿，此时设备将自动启动采集，从空闲状态进入采集状态。

外部启动模式下，下降沿触发的启动采集，与上位机调用 MPS_Start 函数启动采集是等效的。即：在外部启动模式下，有两种方式可以启动采集，一是向复用 IO 口输入电平下降沿，二是在软件中直接调用 MPS_Start 函数，任一种的出现都会使设备启动采集。

在程序设计中，如果使用外部触发方式来启动采集，软件程序应在外部低电平下降沿到来之前就开始读取数据，也就是需要先调用 DataIn 函数，然后进行外部触发，以确保数据的完整性。

外部启动模式下，设备处于待机状态时，如果接收到外部下降沿或 MPS_Start 函数命令，会使设备从待机状态变为采集状态，同时使复用 IO 口从外部启动模式变为闲置模式；如果接收到 MPS_Stop 函数命令，设备的待机状态不改变，但复用 IO 口会自动从外部启动模式变为闲置模式；如果接收到新的 MPS_Configure 命令来改变复用 IO 口的模式，也会使设备退出外部启动模式。设备退出外部启动模式后，将不再响应外部下降沿的触发。

假如将两个或多个 MPS-HD2416 设备的复用 IO 口连接导通，并在软件程序中将其都设置为外部启动模式，则可使用一个外接的低电平下降沿来触发多个设备同步启动采集。此外，此状态下如果软件对其中一个设备发送 MPS_Start 函数命令，此设备在启动采集的同时，会使它的复用 IO 口变为低电平(闲置状态)，从而拉低与之相连的其他复用 IO 口，进而触发其他设备也启动采集，此方式可实现通过软件来触发多个设备同步启动采集。

当 DIOMode 设置为 22 或 32 时，用于同时设置复用 IO 口与 CLK 口，使设备处于多卡同步的工作模式。DIOMode 为 22 时，会将该卡设置为主卡模式，此时复用 IO 口被设置为外部启动模式，CLK 口被设置为对外输出时钟模式。DIOMode 为 32 时，会将该卡设置为从卡模式，此时复用 IO 口被设置为外部启动模式，CLK 口被设置为接收时钟模式。通过使用线材分别将主卡的 IO 与从卡的 IO 对接、主卡的 CLK 与从卡的 CLK 对接，可使主卡和从卡共享相同的 CLK 时钟和外部启动模式，此时通过软件对主卡发送启动采集的

命令时，从卡也将被同步启动，并进行与主卡时钟同步的采样，由此可实现主从卡的同步采集。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_Start (HANDLE DeviceHandle)

int MPS_Start 函数执行启动设备采集的功能。函数执行成功返回 1，执行失败返回 0。设备上电后，默认进入待机状态，此时设备不采集数据；当 MPS_Start 函数执行成功，设备即启动采集，并准备向计算机传输数据。

HANDLE DeviceHandle：操作所针对的设备句柄。

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

int MPS_DataIn 函数执行读取数据的功能。函数执行成功返回 1，执行失败返回 0。MPS_DataIn 函数用于从设备中读取采集数据，该函数通常在设备启动采集后调用（外部启动时除外），每调用一次读出一批样点数据，样点的个数由参数 SampleNumber 决定。在连续不间断采集的应用中，通过循环调用该函数，并将相邻读取的数据进行首尾连接，可获得连续的数据流。

MPS_DataIn 函数运行时，会与设备硬件通信，要求设备传输指定的样点个数到计算机。如果设备板载的硬件数据缓存（DAQ Buffer）中已有多于指定点数的未读数据存在，函数可直接读取并成功返回；如果未读数据少于指定点数，则函数将在内部等待，直到采集设备采集到足够的数据后才成功返回。在等待过程中，如果程序调用了 MPS_Abort 函数，可强行终止 MPS_DataIn 函数的等待，并使之返回失败值。

如果在设备处于空闲状态时调用 MPS_DataIn 函数，由于设备硬件未处于采集状态，无法获取到样点，会导致函数在内部持续等待无法返回。此时可通过调用 MPS_Start 函数（外部启动模式下，也可以向复用 IO 口发送下降沿）的方式来启动采集，启动后即可获取到数据并成功返回；或者通过调用 MPS_Abort 函数，来强行终止 MPS_DataIn 函数并使之返回失败值。

int * DataBuffer：指向用来保存采集到的数据的缓存区的指针。在调用 MPS_DataIn 函数时，需要预先定义一个 32 位带符号整形的一维数组来作为软件保存数据的缓存区，并取指向该数组首地址的指针作为 DataBuffer 参数。MPS_DataIn 函数执行成功后，采集到的数据将会被写入缓存区，所写入的数据个数由 SampleNumber 参数决定。

写入缓存区 DataBuffer 的数据为带符号的 32 位整形数，如果是在 64 位环境下编译，请务必注意将其定义为 **32 位整形**的数据类型。DataBuffer 的每个数值对应一个采样点数据，数据在缓存区内按采样点对应的通道和时间次序循环分布，规则如下：DataBuffer[0]对应 CH1 的第一个采样点数据，DataBuffer[1]对应 CH2 的第一个采样点数据，DataBuffer[2]对应 CH3 的第一个采样点数据……以此类推，DataBuffer[15]对应 CH16 的第一个采样点数据；此后，DataBuffer[16]对应 CH1 的第二个采样点数据，DataBuffer[17]对应 CH2 的第二个采样点数据，DataBuffer[18]对应 CH3 的第二个采样点数据……以此类推，DataBuffer[31]对应 CH16 的第二个采样点数据……以此类推，DataBuffer[（m - 1）× 16 + （n - 1）]对应 CHn 通道的第 m 个采样点数据。

每个采样点数据都是一个带符号的 32 位整形数据，其值的大小与被测信号的电压

值成正比，对应关系为：*Voltage[i] = ((double)DataBuffer[i]/8388608) * 5.16*，即：电压值等于整形数除以 8388608（带符号 24 位的区间范围），再乘以 5.16V（电压量程范围）。在计算中，建议先将整形数转换为 32 位或 64 位浮点数再进行除法运算，以防在整形除法运算中损失精度。

int SampleNumber：MPS_DataIn 函数执行一次所读取的样点个数。该参数决定 MPS_DataIn 函数执行成功后，函数从设备硬件读出的样点的个数。

SampleNumber 是十六个通道的样点数总和，在 MPS-HD2416 的数据中，每个通道对应的样点数是 SampleNumber 的十六分之一。SampleNumber 的最小取值为 128，最大取值为 1024000，且取值必须为 128 的整倍数。若取值不是 128 的整倍数，函数会自动配置为小于该数的最大的 128 整倍数。

SampleNumber 的值与采样率共同决定采集到这些样点所需要的时间，例如在 64K 采样率下，SampleNumber 设置为 204800，每个通道将采集 12800 个样点,除以 64000 点/秒的采样率，可得对应 0.2 秒的采集时间。采集时间可以近似的决定 MPS_DataIn 函数的执行时长。在连续采集的软件设计中，通常需要通过循环读取数据，此时 MPS_DataIn 函数的执行时长会对软件的操作体验产生影响。假如函数执行时间过长，会使得程序响应变慢，出现卡顿；假如执行的时间过短，则会使函数所在的循环执行过于频繁，降低程序的效率。因此 SampleNumber 应结合软件运行效果选取一个较为合理的值，建议使循环每秒执行 30-50 次为佳。另一方面，在比较简单的单次采集应用（例如 FFT 频谱分析等）设计中，若执行时长影响较小，也可以通过设置较大的 SampleNumber 值的方式，一次性读出足够的数据，来简化程序设计，降低编程难度。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_Stop (HANDLE DeviceHandle)

int MPS_Stop 函数执行停止设备采集的功能。函数执行成功返回 1，执行失败返回 0。在设备处于采集状态时调用此函数，函数执行成功会使设备停止采集，进入待机状态，并清空所有未读数据；在设备处于待机状态时调用此函数，不改变设备当前的待机状态。

在一次连续的采集过程结束后，软件应调用 MPS_Stop 函数，使设备进入待机状态以节省功率，同时清空硬件 DAQ Buffer 中的未读数据，为下一次启动采集做好准备。此外，由于设备的设置参数和获取信息的过程也需要在待机状态下进行，因此建议在采集软件初始化前也先调用一次 MPS_Stop 函数，以确保设备处于待机状态。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_CloseDevice (HANDLE DeviceHandle)

int CloseDevice 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。在软件准备关闭退出，或是准备暂时放弃对设备硬件的控制权时，需要进行关闭设备的操作。若未关闭设备直接退出程序，可能导致再次打开设备时出现异常，此时可关闭所有与设备驱动函数相关联的程序线程，并拔插设备硬件，再重新打开即可。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_TimeOut (int TimeOut_ms, int DeviceHandle)

int MPS_TimeOut 函数执行设置针对 MPS_DataIn 函数等待的超时功能，其返回值为超时的时间设定值。MPS_TimeOut 函数可用于外部启动模式下的触发超时，也可用于读取数据的异常处理。MPS_TimeOut 函数被调用后，如果超时时间设定为 0，则不启用等待超时功能，MPS_DataIn 函数执行时将一直等待数据，直到成功返回或被 MPS_Abort 函数强行终止；如果超时时间设定不为 0 且大于 100，则启用等待超时功能，此时如果 MPS_DataIn 函数超过设定时间仍未返回，将强行终止 MPS_DataIn 函数，并使之返回失败值。

int TimeOut_ms：用于设置超时时间的参数，单位为毫秒。超时时间的范围为 100 毫秒（0.1 秒）至 1000000 毫秒（100 秒），当 TimeOut_ms 的值为 0 或 0-100 之间时，超时时长被强制设置为 0，此时不启用超时功能；当 TimeOut_ms 的值为 100-1000000 时，超时时长将按 TimeOut_ms 进行设置，单位为毫秒；当 TimeOut_ms 的值大于 1000000，超时时长将被强制设置为 1000000 毫秒。

int DeviceHandle：操作所针对的设备句柄。

- int MPS_GetDeviceDescriptor (char * DescriptorBuffer, int BufferSize, HANDLE DeviceHandle)

int MPS_GetDeviceDescriptor 函数执行获取设备型号信息字符串的功能。若函数执行成功，返回 1；执行失败返回 0。函数执行成功后，可获取当前设备的型号信息。通过获取型号信息，可以在软件中实现对硬件的识别，进而进行相应操作。设备型号信息为一组字符串，如：“MPS-HD2416”、“MPS-HD2416Lite”、“MPS-HD2404”等，字符串结束于‘/0’结束符。

char * DescriptorBuffer：用于保存获取到的设备型号信息的缓冲区指针。指针指向一个 char 类型的数组，数组中将保存获取到的型号信息字符串。数组长度必须大于或等于函数中 BufferSize 参数的值。函数成功执行后，DescriptorBuffer 中的前 BufferSize 个字符将被更新为从硬件中读出的设备型号信息，BufferSize 长度之后的信息将被忽略。

int BufferSize：获取信息的长度，函数运行成功后，DescriptorBuffer 中的前 BufferSize 个字符将被更新，BufferSize 之后的内容将被忽略。BufferSize 的取值必须小于或等于 DescriptorBuffer 所指向的缓冲区大小，且最大取值为 64，大于 64 的取值将被自动修正为 64。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_GetDeviceInformation (char * InformationBuffer, int BufferSize, HANDLE DeviceHandle)

int MPS_GetDeviceInformation 函数执行获取设备完整硬件信息字符串的功能。若函数执行成功，返回 1；执行失败返回 0。函数执行成功后，可获取当前设备的完整硬件信息，包括设备型号、序列号、参数描述和用户自定义信息等四部分内容。其中设备型号信息表征此设备的型号，序列号信号表征此设备的唯一硬件序列号，参数描述信息包含此设备预定义的硬件参数，用户自定义信息为此前用户写入到该设备内的自定义

内容。GetDeviceInformation 函数执行耗时较长，建议仅在软件初始化过程中调用，不建议在正常采集期间调用。

char * InformationBuffer：用于保存获取到的设备硬件信息的缓冲区指针。指针指向一个 char 类型的数组，数组中将保存获取到的设备信息字符串。数组长度必须大于或等于函数中 BufferSize 参数的值。函数成功执行后，将从 InformationBuffer 的起始开始，到全部信息写完为止，对 InformationBuffer 进行更新。但如果信息的实际总长度大于 BufferSize 的值，则以 BufferSize 为准进行截止。

int BufferSize：获取信息的长度，函数运行成功后，InformationBuffer 中的前 BufferSize 个字符将被允许更新，BufferSize 之后的内容禁止写入更新。BufferSize 的取值必须小于或等于 InformationBuffer 所指向的缓冲区大小，且最大取值不超过 2048。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_SetUserInformation(char * InformationBuffer, int BufferSize, HANDLE DeviceHandle)

int MPS_SetUserInformation 函数执行写入用户自定义信息字符串的功能。若函数执行成功，返回 1；执行失败返回 0。函数执行成功后，向设备硬件内写入一段用户自定义的信息，信息写入后断电不丢失。MPS_SetUserInformation 函数执行耗时较长，并且存在写入次数上限（约 10 万次至 100 万次），建议仅在必要写入新的自定义内容时调用，非必要不调用。

char * InformationBuffer：指向待写入的自定义信息的指针，指针指向一个 char 类型的数组，数组中需预先写入用户自定义信息的字符串。数组长度必须大于或等于 BufferSize 的值，函数成功执行后，InformationBuffer 中的前 BufferSize 个字符将被写入硬件，BufferSize 个之后的字符将被忽略。

int BufferSize：写入信息字符串的长度，函数成功执行后，InformationBuffer 中的前 BufferSize 个字符将被写入硬件,BufferSize 之后的内容将被忽略。BufferSize 的取值必须小于或等于 InformationBuffer 所指向的缓冲区大小，且最大取值为 500，大于 500 的取值将被自动修正为 500。

HANDLE DeviceHandle：操作所针对的设备句柄。

- int MPS_ResetDevice (int DeviceHandle)

int MPS_ResetDevice 函数执行重置硬件的功能。若函数执行成功，返回 1；执行失败返回 0。函数成功执行后，设备硬件会被重置，恢复为初始状态。USB 连接会自动断开并重新连接，此时重置前的设备句柄将会失效，所有通过该句柄调用的函数都会返回失败值。

硬件重置函数仅在设备软件系统处于异常状态,且无法恢复到正常的采集流程时使用。使用时，如果当前程序或线程已无法传递出正在使用的句柄参数，可在其他程序或线程内调用 MPS_Open 函数，获取一个新的句柄来执行 MPS_ResetDevice 函数，同样可以实现对设备的硬件重置。

int DeviceHandle：操作所针对的设备句柄。

三、 *编程范例*

```
/*采集数据的功能子函数*/
/*基本流程：打开设备—>设置参数—>开始采集—>循环获取数据—>停止采集—>关闭设备*/

/*预定义SampleRate参数值*/
#define SampleRate64K 64000                    //采样率为64K
#define SampleRate32K 32000                    //采样率为32K
#define SampleRate16K 16000                    //采样率为16K
#define SampleRate8K 8000                      //采样率为8K
#define SampleRate4K 4000                      //采样率为4K
#define SampleRate2K 2000                      //采样率为2K
#define SampleRate1K 1000                      //采样率为1K

/*预定义DIOMode参数值*/
#define DIOIdle 0                              //DIO口闲置
#define DIOOuput5V 1                           //DIO口输出V高电平
#define ExternalTrigger 2                      //DIO设置为外部触发启动模式

/*预定义变量*/
#define VoltageRange 5.16                      //MPS-HD2416的电压量程值为5.16V
#define MaxNumberPerCH 3840000                 //每通道电压样点个数的最大值，此处预定义
为约384万点，可支持64K采样率下最多连续采集60s
double Voltage[16][MaxNumberPerCH] = {0};      //保存电压的二维缓存数组，每通道最多可保
存MaxNumberPerCH点

/*采集数据子函数*/
int GetVoltageData(int SampleNumberPerCH = 64000, int SampleRate = SampleRate64K, int DIOMode
= DIOIdle)                                     //采集数据，参数分别为：采样点数、采样率、
DIO的模式
{
    /*函数声明*/
    #define Handle int

    HINSTANCE hDll;                            //打开DLL
    hDll=LoadLibrary("MPS Driver.dll");
    if(NULL==hDll)
    {
        AfxMessageBox("Cann't find DLL");
        return 0;
    }

    typedef Handle(*lpMPS_OpenDevice)(int DeviceNumber);    //打开设备函数的声明
    lpMPS_OpenDevice MPS_OpenDevice=(lpMPS_OpenDevice)GetProcAddress(hDll,"MPS_OpenDevice");
```

```
    if(NULL==MPS_OpenDevice)
    {
        AfxMessageBox("Cann't find <MPS_OpenDevice> function");
    }

    typedef int(*lpMPS_CloseDevice)(Handle DeviceHandle);   //关闭设备函数的声明
    lpMPS_CloseDevice
MPS_CloseDevice=(lpMPS_CloseDevice)GetProcAddress(hDll,"MPS_CloseDevice");
    if(NULL==MPS_CloseDevice)
    {
        AfxMessageBox("Cann't find <MPS_CloseDevice> function");
    }

    typedef int(*lpMPS_Configure)(int SampleRate, int DIOMode, Handle DeviceHandle);    //
设置设备参数函数的声明
    lpMPS_Configure MPS_Configure=(lpMPS_Configure)GetProcAddress(hDll,"MPS_Configure");
    if(NULL==MPS_Configure)
    {
        AfxMessageBox("Cann't find <MPS_Configure> function");
    }

    typedef int(*lpMPS_Start)(Handle DeviceHandle);    //启动设备采集函数的声明
    lpMPS_Start MPS_Start=(lpMPS_Start)GetProcAddress(hDll,"MPS_Start");
    if(NULL==MPS_Start)
    {
        AfxMessageBox("Cann't find <MPS_Start> function");
    }

    typedef int(*lpMPS_Stop)(Handle DeviceHandle); //停止设备采集函数的声明
    lpMPS_Stop MPS_Stop=(lpMPS_Stop)GetProcAddress(hDll,"MPS_Stop");
    if(NULL==MPS_Stop)
    {
        AfxMessageBox("Cann't find <MPS_Stop> function");
    }

    typedef int(*lpMPS_DataIn)(int *DataArray,int SampleNumber, Handle DeviceHandle);   //
读取数据函数的声明
    lpMPS_DataIn MPS_DataIn=(lpMPS_DataIn)GetProcAddress(hDll,"MPS_DataIn");
    if(NULL==MPS_DataIn)
    {
        AfxMessageBox("Cann't find <MPS_DataIn> function");
    }

    typedef int(*lpMPS_TimeOut)( int TimeOut_ms,Handle DeviceHandle);//设置读数超时函数的声
```

明

```
    lpMPS_TimeOut MPS_TimeOut=(lpMPS_TimeOut)GetProcAddress(hDll,"MPS_TimeOut");
    if(NULL==MPS_TimeOut)
    {
        AfxMessageBox("Cann't find <MPS_TimeOut> function");
    }

    /*函数声明结束*/


    /*采集数据*/
    //定义变量
    int Flag = 1;                                   //函数执行成功标志
    int DeviceNumber = 0;                           //操作系统分配的设备硬件序号
    Handle DeviceHandle;                            //设备句柄

    DeviceHandle = MPS_OpenDevice(DeviceNumber);    //先打开设备
    if(DeviceHandle == -1)                          //若打开失败，报错并返回
    {
        AfxMessageBox("OpenDeviceError");
        return 0;
    }

    Flag = MPS_Configure(SampleRate, DIOMode, DeviceHandle);     //设置采样率和复用IO口为闲
置模式

    MPS_TimeOut(5000, DeviceHandle);                //设置等待超时为5s，如果用于外启动超时，
可根据情况修改设置值

    if(DIOMode != ExternalTrigger)Flag = MPS_Start(DeviceHandle);//启动设备采集；外部启动模
式时不通过调用MPS_Start来启动


    //以下为循环调用采集函数的方式来读取数据
    int DataBuffer[16*1024] = {0};                  //用于保存读出的原始整形数据的缓存数组，
请务必注意此变量一定定义为32位整形数
                                                    //用于保存经过计算的电压的数据数组
Voltage已定义为全局变量
    if(SampleNumberPerCH > MaxNumberPerCH)
            SampleNumberPerCH = MaxNumberPerCH;     //每通道获取到的样点个数取值应小于
MaxNumberPerCH

    int Counter = 0;                                //数据点数计数变量
```

```
    while(1)
    {
        Flag = MPS_DataIn(DataBuffer,1024*16,DeviceHandle);//读取数据，每次读取点，十六个通道共1024*8

        if(Flag != 0)                           //如果采集成功
            {
                for(int i = 0; i < 1024; i++)    //数据处理，计算电压值；16个通道的数据在缓存中循环排列；电压值 = (采集值/(65536*128))*量程
                {
                    if(Counter >= SampleNumberPerCH) break;
                                                //如果样点达到SampleNumberPerCH，则跳出for循环
                    Voltage[0][Counter] = ((double)DataBuffer[i*8] / 8388608) * VoltageRange;                                    //通道1的电压
                    Voltage[1][Counter] = ((double)DataBuffer[i*8 + 1] / 8388608) * VoltageRange;                                //通道2的电压
                    Voltage[2][Counter] = ((double)DataBuffer[i*8 + 2] / 8388608) * VoltageRange;                                //通道3的电压
                    Voltage[3][Counter] = ((double)DataBuffer[i*8 + 3] / 8388608) * VoltageRange;                                //通道4的电压
                    Voltage[4][Counter] = ((double)DataBuffer[i*8 + 4] / 8388608) * VoltageRange;                                //通道5的电压
                    Voltage[5][Counter] = ((double)DataBuffer[i*8 + 5] / 8388608) * VoltageRange;                                //通道6的电压
                    Voltage[6][Counter] = ((double)DataBuffer[i*8 + 6] / 8388608) * VoltageRange;                                //通道7的电压
                    Voltage[7][Counter] = ((double)DataBuffer[i*8 + 7] / 8388608) * VoltageRange;                                //通道8的电压
                    Voltage[8][Counter] = ((double)DataBuffer[i*8 + 8] / 8388608) * VoltageRange;                                //通道9的电压
                    Voltage[9][Counter] = ((double)DataBuffer[i*8 + 9] / 8388608) * VoltageRange;                                //通道10的电压
                    Voltage[10][Counter] = ((double)DataBuffer[i*8 + 10] / 8388608) * VoltageRange;                              //通道11的电压
                    Voltage[11][Counter] = ((double)DataBuffer[i*8 + 11] / 8388608) * VoltageRange;                              //通道12的电压
                    Voltage[12][Counter] = ((double)DataBuffer[i*8 + 12] / 8388608) * VoltageRange;                              //通道13的电压
                    Voltage[13][Counter] = ((double)DataBuffer[i*8 + 13] / 8388608) * VoltageRange;                              //通道14的电压
                    Voltage[14][Counter] = ((double)DataBuffer[i*8 + 14] / 8388608) * VoltageRange;                              //通道15的电压
                    Voltage[15][Counter] = ((double)DataBuffer[i*8 + 15] / 8388608) *
```

```
VoltageRange;                                    //通道16的电压


                    Counter++;
                }
            }
            else
            {
                AfxMessageBox("MPS_DataInError");//如果采集失败报错并跳出while循环
                break;
            }

        if(Counter >= SampleNumberPerCH)         //如果读出全部数据，跳出while循环。此处也
可添加其他跳出循环的条件
        {
            break;
        }
    }

    //至此采集过程完成，后面要停止硬件采集并关闭设备
    Flag = MPS_Stop(DeviceHandle);               //采集任务结束后停止采集
    Flag = MPS_CloseDevice(DeviceHandle);        //最后关闭设备

    if (Counter == SampleNumberPerCH)            //判断GetVoltageData函数是否已经获取到了
指定的SampleNumberPerCH个样点数
    {
        AfxMessageBox("MPS_DataInSuccess");
        return 1;
    }
    else return 0;
}




/*获取设备硬件信息的功能子函数*/
int GetInformation()                             //获取设备硬件信息
{
    /*函数声明*/
    #define Handle int

    HINSTANCE hDll;                              //打开DLL
    hDll=LoadLibrary("MPS Driver.dll");
    if(NULL==hDll)
```

```
    {
        AfxMessageBox("Cann't find DLL");
        return 0;
    }

    typedef Handle(*lpMPS_OpenDevice)(int DeviceNumber);//打开设备函数的声明
    lpMPS_OpenDevice MPS_OpenDevice=(lpMPS_OpenDevice)GetProcAddress(hDll,"MPS_OpenDevice");
    if(NULL==MPS_OpenDevice)
    {
        AfxMessageBox("Cann't find <MPS_OpenDevice> function");
    }

    typedef int(*lpMPS_CloseDevice)(Handle DeviceHandle);//关闭设备函数的声明
    lpMPS_CloseDevice
MPS_CloseDevice=(lpMPS_CloseDevice)GetProcAddress(hDll,"MPS_CloseDevice");
    if(NULL==MPS_CloseDevice)
    {
        AfxMessageBox("Cann't find <MPS_CloseDevice> function");
    }

    typedef int(*lpMPS_GetDeviceDescriptor)(char *DescriptorBuffer,int BufferSize,Handle
DeviceHandle);                                          //获取设备型号信息函数的声明
    lpMPS_GetDeviceDescriptor
MPS_GetDeviceDescriptor=(lpMPS_GetDeviceDescriptor)GetProcAddress(hDll,"MPS_GetDeviceDescrip
tor");
    if(NULL==MPS_GetDeviceDescriptor)
    {
        AfxMessageBox("Cann't find <MPS_GetDeviceDescriptor> function");
    }


    typedef int(*lpMPS_GetDeviceInformation)(char *DeviceInformationBuffer,int
BufferSize,Handle DeviceHandle);                        //获取设备完整硬件信息函数的声明
    lpMPS_GetDeviceInformation
MPS_GetDeviceInformation=(lpMPS_GetDeviceInformation)GetProcAddress(hDll,"MPS_GetDeviceInfor
mation");
    if(NULL==MPS_GetDeviceInformation)
    {
        AfxMessageBox("Cann't find <MPS_GetDeviceInformation> function");
    }

    /*函数声明结束*/
```

```
    //定义变量
    int Flag = 1;                                   //函数执行成功标志
    int DeviceNumber = 0;                           //操作系统分配的设备硬件序号
    Handle DeviceHandle;                            //设备句柄

    //读取设备型号信息示例
    char DeviceDescriptor[64] = {0};                //保存设备型号信息的缓存数组

    DeviceHandle = MPS_OpenDevice(DeviceNumber);    //先打开设备
    if(DeviceHandle == -1)                          //若打开失败，报错并返回
    {
        AfxMessageBox("OpenDeviceError");
        return 0;
    }

    Flag = MPS_GetDeviceDescriptor(DeviceDescriptor,64,DeviceHandle);//获取设备型号

    Flag = MPS_CloseDevice(DeviceHandle);           //最后关闭设备

    AfxMessageBox(DeviceDescriptor);                //显示设备型号

    //读取设备硬件信息示例
    char DeviceInformation[2048] = {0};             //保存设备型号信息的缓存数组

    DeviceHandle = MPS_OpenDevice(DeviceNumber);    //先打开设备
    if(DeviceHandle == -1)                          //若打开失败，报错并返回
    {
        AfxMessageBox("OpenDeviceError");
        return 0;
    }

    Flag = MPS_GetDeviceInformation(DeviceInformation,2048,DeviceHandle);//获取设备信息

    Flag = MPS_CloseDevice(DeviceHandle);           //最后关闭设备

    AfxMessageBox(DeviceInformation);               //显示设备信息

    return 1;

}
```

```
/*写入用户自定义信息的功能子函数*/
int SetUserInformation()                                    //写入用户自定义信息
{
    /*函数声明*/
    #define Handle int

    HINSTANCE hDll;                                         //打开DLL
    hDll=LoadLibrary("MPS Driver.dll");
    if(NULL==hDll)
    {
        AfxMessageBox("Cann't find DLL");
        return 0;
    }

    typedef Handle(*lpMPS_OpenDevice)(int DeviceNumber);//打开设备函数的声明
    lpMPS_OpenDevice MPS_OpenDevice=(lpMPS_OpenDevice)GetProcAddress(hDll,"MPS_OpenDevice");
    if(NULL==MPS_OpenDevice)
    {
        AfxMessageBox("Cann't find <MPS_OpenDevice> function");
    }

    typedef int(*lpMPS_CloseDevice)(Handle DeviceHandle);//关闭设备函数的声明
    lpMPS_CloseDevice
MPS_CloseDevice=(lpMPS_CloseDevice)GetProcAddress(hDll,"MPS_CloseDevice");
    if(NULL==MPS_CloseDevice)
    {
        AfxMessageBox("Cann't find <MPS_CloseDevice> function");
    }

    typedef int(*lpMPS_SetUserInformation)(char *UserInformationBuffer,int BufferSize,Handle
DeviceHandle);                                              //写入用户自定义信息函数的声明
    lpMPS_SetUserInformation
MPS_SetUserInformation=(lpMPS_SetUserInformation)GetProcAddress(hDll,"MPS_SetUserInformation
");
    if(NULL==MPS_SetUserInformation)
    {
        AfxMessageBox("Cann't find <MPS_SetUserInformation> function");
    }

    /*函数声明结束*/

    //定义变量
    int Flag = 1;                                   //函数执行成功标志
    int DeviceNumber = 0;                           //操作系统分配的设备硬件序号
```

```
    Handle DeviceHandle;                          //设备句柄

    //写入用户自定义信息示例
    char UserInformation[500] = "This is a test."; //待写入的用户信息的缓存数组

    DeviceHandle = MPS_OpenDevice(DeviceNumber);  //先打开设备
    if(DeviceHandle == -1)                        //若打开失败，报错并返回
    {
        AfxMessageBox("OpenDeviceError");
        return 0;
    }

    Flag = MPS_SetUserInformation(UserInformation,500,DeviceHandle);//写入用户信息

    Flag = MPS_CloseDevice(DeviceHandle);         //最后关闭设备

    AfxMessageBox(UserInformation);               //显示写入的内容

    return 1;

}



/*重置硬件示例*/
int ResetDevice()                                 //重置硬件函数
{
    /*函数声明*/
    #define Handle int

    HINSTANCE hDll;                               //打开DLL
    hDll=LoadLibrary("MPS Driver.dll");
    if(NULL==hDll)
    {
        AfxMessageBox("Cann't find DLL");
        return 0;
    }

    typedef Handle(*lpMPS_OpenDevice)(int DeviceNumber);//打开设备函数的声明
    lpMPS_OpenDevice MPS_OpenDevice=(lpMPS_OpenDevice)GetProcAddress(hDll,"MPS_OpenDevice");
    if(NULL==MPS_OpenDevice)
    {
        AfxMessageBox("Cann't find <MPS_OpenDevice> function");
```

```
    }

    typedef int (*lpMPS_ResetDevice)(Handle DeviceHandle);//重置硬件函数的声明
    lpMPS_ResetDevice MPS_ResetDevice
=(lpMPS_ResetDevice)GetProcAddress(hDll,"MPS_ResetDevice");
    if(NULL==MPS_ResetDevice)
    {
        AfxMessageBox("Cann't find <MPS_ResetDevice> function");
    }

    /*函数声明结束*/

    //定义变量
    int Flag = 1;                                   //函数执行成功标志
    int DeviceNumber = 0;                           //操作系统分配的设备硬件序号
    Handle DeviceHandle;                            //设备句柄

    DeviceHandle = MPS_OpenDevice(DeviceNumber);    //先打开设备
    if(DeviceHandle == -1)                          //若打开失败，报错并返回
    {
        AfxMessageBox("OpenDeviceError");
        return 0;
    }

    Flag = MPS_ResetDevice(DeviceHandle);           //重置硬件函数

    if (Flag != 0)
        {
            AfxMessageBox("MPS_ResetDeviceSuccess");
        }
    else
        {
            AfxMessageBox("MPS_ResetDeviceError");
        }
    return Flag;
}



/*功能测试*/
void Test()
{
    int Flag = 0;
//  Flag = GetInformation();                        //读取设备硬件信息测试，通常需要时在软件
```

```
程序初始化时读一次即可，不必经常读取
//   Flag = SetUserInformation();                    //写入用户自定义信息测试，如不需要写入信
息的话不必执行此函数
     Flag = GetVoltageData(64000,SampleRate64K, DIOIdle);//采集数据测试，测试64K采样率采集1s
//   Flag = ResetDevice();                           //重置硬件测试
}
```

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。
- 设备与计算机 USB 断开后，请间隔 1s 左右再重新接入。
- 设备通常使用计算机 USB 供电即可正常工作，如出现 USB 供电不足，可通过采集卡配套的外接电源适配器来进行外部供电。
- 设备属于精密电子仪器，使用中请注意防尘防潮与防静电。存在人体放电现象时，请预先做好防静电措施，并在使用中尽量避免触碰接头、外壳、信号线及传感器中的金属部分等。设备长期不用时，请做好密封保存。
- 禁止用户自行拆卸设备的下层外壳，一经拆卸将不再享有保修服务，且因此导致的设备故障将由用户自行承担。
- MPS-HD2416 自出厂之日起三年内，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，凭保修卡或订单信息免费维修。因违反操作规定和使用要求造成人为损坏的，需交纳维修费用进行维修。
- 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。