FTDAQ_SILVERX

驱动使用文档

版本历史			
版本号	发布者	日期	描述
V20211117A0	张德志	2021-11-17	首次发布
V20211202A0	张德志	2021-12-02	版本迭代，修改了驱动的使用方式，增加测量外部信号 SNR
V20211227A0	张德志	2021-12-27	更新测量函数说明

广州市方瞳科技有限责任公司
广州市科学城南翔支路 1 号 D 栋 D306
www.finetooling.com

FTDAQ_SilverX 驱动使用文档

使用流程

1. 使用 DriverHelper.InitUsbDAQDevices()初始化所有 USB DAQ 设备
2. 使用 DriverHelper.GetDeviceIds()获取目前系统上所有 DAQ 设备的 id。
3. 选择要使用的 DAQ 设备，使用

```
SilverXDevice device = DriverHelper.GetSilverXDev(devicesId);
device.Initialize();
```

实例化 DAQ 设备，并且完成 DAQ 设备的初始化。

4. 通过 device 调用 DAQ 相应的功能函数进行调用
5. 实例如下：

```
public DAQTest()
{
    //初始化所有 USB DAQ 设备
    DriverHelper.InitUsbDAQDevices();
    //获取目前系统上所有 DAQ 设备的 id
    List<string> devicesIds = DriverHelper.GetSilvexXDevIds();

    if (devicesIds != null && devicesIds.Count > 0)
    {
        //实例化 DAQ 设备，并初始化该设备
        SilverXDevice device = DriverHelper.GetSilverXDev(devicesIds[0]);
        device.Initialize();
        int value = device.DIOReadPort(port);
    }
}
```

与之前版本的不同

1. 之前版本是通过 SilverXDevice.Api 调用设备方法，新版本直接使用 SilverXDevice 调用方法，例如之前读取 IO 口状态使用 device.Api.DIOReadPort(port)；新版本改为：device.DIOReadPort(port)。
2. 新版本的 DriverHelper 兼容了 FT8261，默认处理的设备类型是 DAQDevice，该类被 SilverXDevice 和 FT8261Device 继承，如果要获取 FT8261Device，可以使用以下方法：

```
FT8261Device device = (FT8261Device)DriverHelper.GetDevice(id);
```

3. DriverHelper.DeviceUpdateEvent 事件处理的是 DAQDevice 设备
4. 新设备插入后（或者软件重新打开），不会自动初始化，需要调用 SilverXDevice.Initialize() 方法进行初始化

DriverHelper 类

设备插拔事件

SilverXDAQUpdateEvent

```
/// <summary>
/// 声明 DAQ 插拔委托
/// </summary>
/// <param name="dev">DAQ 设备</param>
/// <param name="isAdd">true:插入，false:拔出</param>
```

```
public delegate void DeviceUpdate(DAQDevice dev, bool isAdd);

/// <summary>
/// SilverX 系列 DAQ 插拔事件
/// </summary>
public static event DeviceUpdate DeviceUpdateEvent;
```

静态方法

InitUsbDAQDevices

```
/// <summary>
/// 初始化所有 USB 设备
/// </summary>
public static void InitUsbDAQDevices()
```

GetDeviceId

```
/// <summary>
/// 获取 DAQ 设备的 ID 信息
/// </summary>
/// <param name="dev">USB 设备</param>
/// <returns>返回{VendorID}-{ProductID}-{SerialNumber}格式的设备 id 信息</returns>
public static string GetDeviceId(USBDevice dev)
```

GetDeviceIds

```
/// <summary>
/// 获取所有 DAQ 设备的 ID
/// </summary>
/// <returns></returns>
public static List<string> GetDeviceIds()
```

GetDeviceIds(DAQModel)

```
/// <summary>
/// 获取所有指定 model 设备的 ID
/// </summary>
/// <returns></returns>
public static List<string> GetDeviceIds(DAQModel model)
```

GetDevice

```
/// <summary>
/// 根据设备 ID 获取设备
/// </summary>
/// <param name="id">silverX 系列 DAQ 的 ID</param>
/// <returns></returns>
public static DAQDevice GetDevice(string id)
```

GetDevices

```
/// <summary>
/// 获取所有 DAQ 设备
/// </summary>
/// <returns></returns>
public static List<DAQDevice> GetDevices()
```

GetDevices

```
/// <summary>
/// 获取所有指定 model 设备
/// </summary>
/// <returns></returns>
public static List<DAQDevice> GetDevices(DAQModel model)
```

GetSilverXDev

```
/// <summary>
/// 根据 silverX 设备 ID 获取设备信息
/// </summary>
/// <param name="id">silverX 系列 DAQ 的 ID</param>
/// <returns></returns>
public static SilverXDevice GetSilverXDev(string id)
```

SilverXDevice 类

属性

```
/// <summary>
```

```
/// DAQ 型号
/// </summary>
public DAQModel Model
/// <summary>
/// 设备 ID
/// </summary>
public string ID
/// <summary>
/// 设备序列号
/// </summary>
public string SN
/// <summary>
/// USB 设备资源
/// </summary>
public CyUSBDevice UsbDev
/// <summary>
/// DAQ 配置信息
/// </summary>
public DAQConfig Config
/// <summary>
/// DAQ 设备信息，主要是 DAQA 设备的厂商、型号、序列号、版本信息等
/// </summary>
public DAQDeviceInfo DeviceInfo
```

构造函数

```
/// <summary>
/// 通过 USB 设备信息 初始化 SilverX 系列的 DAQ 设备
/// </summary>
/// <param name="usbDev"></param>
public SilverXDevice(CyUSBDevice usbDev)
```

Initialize

```
/// <summary>
/// 初始化 DAQ 设备，会停止 AI、AO、Counter、PFI，将 IO 设置为默认值(输入状态)
/// </summary>
/// <returns>返回初始化结果，是否初始化成功</returns>
public bool Initialize()
```

DIOSetPortDirection

```
/// <summary>
/// 按端口设置 DIO 的方向
/// </summary>
/// <param name="port">需要设置方向的端口</param>
/// <param name="direction">方向值：0~255，对应位置 1 为输入，置 0 为输出;
///                         如：0x01,表示 line 0 输入，line1~7 输出</param>
public void DIOSetPortDirection(int port, int direction)
```

DIOSetPortLineDirection

```
/// <summary>
/// 按位设置 DIO 的方向
/// </summary>
/// <param name="port">要操作 DIO 的端口</param>
/// <param name="line">要操作的 IO 脚：0-7</param>
/// <param name="direction">DIO 的方向：In、Out</param>
public void DIOSetPortLineDirection(int port,int line, DIODirection direction)
```

DIOGetPortValue

```
/// <summary>
/// 按端口获取 DIO 的值
/// </summary>
/// <param name="port">要操作 DIO 的端口</param>
/// <returns></returns>
public int DIOGetPortValue(int port)
```

DIOGetPortLineValue

```
/// <summary>
/// 按位获取 DIO 的值
/// </summary>
/// <param name="port">要操作 DIO 的端口</param>
/// <param name="line">要操作的 IO 脚：0-7</param>
/// <returns></returns>
public DIOLevel DIOGetPortLineValue(int port, int line)
```

DIOSetPortValue

```
/// <summary>
/// 按端口输出 DIO
/// </summary>
/// <param name="port">要操作 DIO 的端口</param>
/// <param name="value">需要输出的电平值 0~255,相应位 0:低电平；1：高电平</param>
public void DIOSetPortValue(int port, int value)
```

DIOSetPortLineValue

```
/// <summary>
/// 按位输出 DIO
/// </summary>
/// <param name="port">要操作 DIO 的端口</param>
/// <param name="line">要操作的 IO 脚：0-7</param>
/// <param name="level">要输出的值：Low:低电平，High：高电平</param>
public void DIOSetPortLineValue(int port, int line, DIOLevel level)
```

DIOWritePortLine

```
/// <summary>
/// DIO 按位输出，该方法会将指定引脚设置为输出
/// </summary>
/// <param name="port">要操作 DIO 的端口</param>
/// <param name="line">要操作的 IO 脚：0-7</param>
/// <param name="level">要输出的值：Low:低电平，High：高电平</param>
public void DIOWritePortLine(int port, int line, DIOLevel level)
```

DIOReadPortLine

```
/// <summary>
/// 按位读取 DIO 状态，该方法会将指定引脚设置为输入
/// </summary>
/// <param name="port">要操作 DIO 的端口</param>
/// <param name="line">要操作的 IO 脚：0-7</param>
/// <returns>0：低电平；1：高电平</returns>
public DIOLevel DIOReadPortLine(int port, int line)
```

DIOWritePort

```
/// <summary>
/// 按端口输出 DIO，该方法会将指定 Port 设置为输出
/// </summary>
/// <param name="port">要操作 DIO 的端口</param>
/// <param name="value">需要输出的电平值 0~255,相应位 0:低电平；1：高电平</param>
public void DIOWritePort(int port, int value)
```

DIOReadPort

```
/// <summary>
/// 按端口读取 DIO 值，该方法会将指定 Port 设置为输入
/// </summary>
/// <param name="port">要操作 DIO 的端口</param>
/// <returns>返回 0~255,置 1 的位表示相应的引脚为高电平</returns>
public uint DIOReadPort(int port)
```

PFISetDutyRate

```
/// <summary>
/// PFI 设置频率占空比
/// </summary>
/// <param name="port">PFI 的 port</param>
/// <param name="line">PFI 的 line</param>
/// <param name="duty">输出方波频率的占空比时间，单位为 us</param>
public void PFISetDutyRate(int port, int line, int duty)
```

PFISetFrequency

```
/// <summary>
/// PFI 设置频率
/// </summary>
/// <param name="port">PFI 的 port</param>
/// <param name="line">PFI 的 line</param>
/// <param name="frequency">输出方波频率，单位 Hz</param>
public void PFISetFrequency(int port, int line, int frequency)
```

PFIStart

```
/// <summary>
```

```
/// PFI 开始输出方波
/// </summary>
/// <param name="port">PFI 的 port</param>
/// <param name="line">PFI 的 line</param>
public void PFIStart(int port, int line)
```

PFIStop

```
/// <summary>
/// PFI 停止输出方波
/// </summary>
/// <param name="port">PFI 的 port</param>
/// <param name="line">PFI 的 line</param>
public void PFIStop(int port, int line)
```

~~PFITriAI~~

```
~~/// <summary>~~
~~/// PFI AI 触发信号设置~~
~~/// </summary>~~
~~/// <param name="port">PFI 的 port</param>~~
~~/// <param name="line">PFI 的 line</param>~~
~~/// <param name="edge">上升沿或下降沿触发</param>~~
~~public void PFITriAI(int port, int line,CTEdge edge)~~
```

~~PFITriAO~~

```
~~/// <summary>~~
~~/// PFI AO 触发信号设置~~
~~/// </summary>~~
~~/// <param name="port">PFI 的 port</param>~~
~~/// <param name="line">PFI 的 line</param>~~
~~/// <param name="edge">上升沿或下降沿触发</param>~~
~~public void PFITriAO(int port, int line, CTEdge edge)~~
```

PFIStartSquare

```
/// <summary>
/// PFI 输出指定频率和占空比的方波
/// </summary>
/// <param name="port">PFI 的 port</param>
/// <param name="line">PFI 的 line</param>
```

```
/// <param name="frequency">输出方波频率，单位 Hz</param>
/// <param name="dutyRate">输出方波频率的占空比，值范围：0~1</param>
public void PFIStartSquare(int port, int line,int frequency,double dutyRate = 0.5)
```

ADCConvertSyncVolts

```
/// <summary>
/// 将返回的原始字节数组转为同步采样的数据
/// </summary>
/// <param name="buffer">原始字节数组</param>
/// <param name="totalPoints">实际接收到的采样点数</param>
/// <param name="amplitude">采样量程的幅值，例如使用±10V 的量程，这里填写 10，使用±5V 的量程，这里填写 5</param>
/// <returns></returns>
public List<ChannelDatas> ADCConvertSyncVolts(byte[] buffer, int totalPoints, double amplitude)
```

ADCConvertSyncVolts

```
/// <summary>
/// 将返回的原始字节数组转为同步采样的数据
/// 因为实际接收到的采样点数可能会比设置的点数多，每一个芯片最多多一包数据，
/// 该方法能够将数组按照用户指定的长度(samplePoints)返回
/// </summary>
/// <param name="buffer">原始字节数组</param>
/// <param name="totalPoints">实际接收到的采样点数</param>
/// <param name="samplePoints">每个通道的采样点数</param>
/// <param name="amplitude">采样量程的幅值，例如使用±10V 的量程，这里填写 10，使用±5V 的量程，这里填写 5</param>
/// <returns></returns>
public List<ChannelDatas> ADCConvertSyncVolts(byte[] buffer, int totalPoints, int samplePoints, double amplitude)
```

ADCSampleAsyncVolts

```
/// <summary>
/// 分是复用采样的原始电压值，没有进行分组
/// </summary>
/// <param name="buffer">原始字节数组</param>
/// <param name="samplePoints">采样点数</param>
/// <param name="amplitude">采样量程的幅值，例如使用±10V 的量程，这里填写 10，使用±5V 的量程，这里填写 5</param>
```

```
/// <returns></returns>
public List<double> ADCSampleAsyncVolts(byte[] buffer, int samplePoints, double amplitude)
```

ADCConvertAsyncVolts

```
/// <summary>
/// 分是复用采样的原始电压值，进行分组
/// </summary>
/// <param name="channels">采样的通道</param>
/// <param name="buffer">原始字节数组</param>
/// <param name="samplePoints">采样点数</param>
/// <param name="amplitude">采样量程的幅值，例如使用±10V 的量程，这里填写 10，使用±5V 的量程，这里填写 5</param>
/// <returns></returns>
public List<ChannelDatas> ADCConvertAsyncVolts(List<int> channels, byte[] buffer, int samplePoints, double amplitude)
```

ADCSampleDatas

```
/// <summary>
/// ADC 采集指定长度的数据
/// </summary>
/// <param name="channel">需要采集数据的通道</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samplePoints">需要采集的点数</param>
/// <param name="range">量程：±100mv,±200mv,±500mv,±1v,±2v,±5v,±10v</param>
/// <returns></returns>
public List<double> ADCSampleDatas(int channel, int sampleRate, int samplePoints, VoltRange range)
```

ADCSampleChannels

```
/// <summary>
/// ADC 采集指定长度的数据
/// </summary>
/// <param name="channels">需要采集数据的通道数组</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samplePoints">需要采集的点数</param>
/// <param name="range">量程：±100mv,±200mv,±500mv,±1v,±2v,±5v,±10v</param>
/// <returns>返回各个通道的采样数据</returns>
public List<ChannelDatas> ADCSampleChannels(List<int> channels, int sampleRate,
```

```
int samplePoints, VoltRange range)
```

ADCSampleOnce

```
/// <summary>
/// ADC 采集一帧数据的平均值
/// </summary>
/// <param name="channel">指定采集通道</param>
/// <param name="range">指定量程：±100mv,±200mv,±500mv,±1v,±2v,±5v,±10v</param>
/// <param name="sampleRate">指定采样率</param>
/// <returns>采集结果</returns>
public double ADCSampleOnce(int channel, VoltRange range, int sampleRate = 100000)
```

ADCSampleConfig

```
/// <summary>
/// ADC 采样参数配置，不需要设置参数，传 None,如：ADCChannelMode.None,
ADCSampleMode.None
/// </summary>
/// <param name="channel">需要进行采样的通道，总共 32 位，每位对应 AI0~AI31 通道，相应
位置 1，则采样该通道数据</param>
/// <param name="sampleMode">采样模式:单包(240 个采样点)、固定点数(采集满 samplePoints
个点)、
///                         连续采样(ADCStart 后，一直采样，直到发送 ADCStop 才结
束)</param>
/// <param name="signelType">采集的信号类型：电压、电流</param>
/// <param name="range">采样的量程：±100mv,±200mv,±500mv,±1v,±2v,±5v,±10v</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samplePoints">采样点数，设置固定点数采样时，该值有效</param>
/// <returns>true:参数设置成功，false：参数设置失败</returns>
public bool ADCSampleConfig(int channel, ADCSampleMode sampleMode,
ADCSignelType signelType, VoltRange range, int sampleRate, int samplePoints)
```

ADCSampleStart

```
/// <summary>
/// 开始采样
/// </summary>
/// <returns>true:开始采样；false：开始采样失败</returns>
public bool ADCSampleStart()
```

ADCSampleStop

```
/// <summary>
/// 停止采样
/// </summary>
/// <returns>true:已停止；false:停止失败</returns>
public bool ADCSampleStop()
```

DACGenerateDC

```
/// <summary>
/// DAQ 输出直流电压
/// </summary>
/// <param name="channel">需要输出的 DAC 通道</param>
/// <param name="volt">输出的电压值</param>
public void DACGenerateDC(int channel, double volt)
```

DACGenerateSine

```
/// <summary>
/// DAQ 输出正弦波信号
/// </summary>
/// <param name="channel">需要输出波形的通道</param>
/// <param name="frequency">波形频率</param>
/// <param name="amplitude">波形峰值</param>
/// <param name="sampleRate">采样率</param>
/// <param name="offset">直流偏移</param>
public void DACGenerateSine(int channel, double frequency, double amplitude, int
sampleRate, double offset = 0)
```

DACGenerateTriangle

```
/// <summary>
/// DAQ 输出三角波信号
/// </summary>
/// <param name="channel">需要输出波形的通道</param>
/// <param name="frequency">波形频率</param>
/// <param name="amplitude">波形峰值</param>
/// <param name="sampleRate">采样率</param>
/// <param name="offset">直流偏移</param>
public void DACGenerateTriangle(int channel, double frequency, double amplitude,
int sampleRate,
```

```
    double offset = 0)
```

DACGenerateSquare

```
/// <summary>
/// DAQ 输出方波信号
/// </summary>
/// <param name="channel">DAC 输出通道</param>
/// <param name="amplitude">波形的峰值</param>
/// <param name="frequency">波形的频率</param>
/// <param name="sampleRate">波形的采样率</param>
/// <param name="offset">直流偏移</param>
/// <param name="dutyRatio">方波的正脉宽占空比</param>
public void DACGenerateSquare(int channel,double amplitude, double frequency,
    double sampleRate, double offset = 0, double dutyRatio = 0.5)
```

DACGenerateCustom

```
/// <summary>
/// DAQ 输出自定义波形
/// </summary>
/// <param name="channel">DAC 输出通道</param>
/// <param name="frequency">波形频率</param>
/// <param name="wave">波形数据</param>
public void DACGenerateCustom(int channel, double frequency, double[] wave)
```

DACStopOutput

```
/// <summary>
/// DAQ 停止输出信号
/// </summary>
/// <param name="channel">需要停止输出的通道</param>
public void DACStopOutput(int channel)
```

CounterReadEdge

```
/// <summary>
/// 计数器计算沿跳变数量
/// </summary>
/// <param name="channel">计数器通道</param>
/// <param name="edge">需要计数的沿跳变：上升沿或下降沿</param>
/// <param name="timeMS">计数器计数时长，单位：毫秒(ms)</param>
```

```
/// <returns>统计到的沿跳变个数</returns>
public uint CounterReadEdge(int channel, CTEdge edge, int timeMS)
```

CounterReadPeriod

```
/// <summary>
/// 计数器计算波形周期
/// </summary>
/// <param name="channel">计数器通道</param>
/// <param name="timeMS">计数时长，单位：毫秒(ms)</param>
/// <param name="periodCount">设置需要计算的周期个数，该值越大，计算结果越准确</param>
/// <returns>返回周期长度，单位是 us</returns>
public double CounterReadPeriod(int channel, int timeMS, uint periodCount = 50)
```

CounterReadPulseWidth

```
/// <summary>
/// 计数器计算波形脉宽
/// </summary>
/// <param name="channel">计数器通道</param>
/// <param name="ctmode">脉宽类型：正脉宽/负脉宽</param>
/// <param name="timeMS">计数时长，单位：毫秒(ms)</param>
/// <returns>返回脉宽长度，单位是 us</returns>
public double CounterReadPulseWidth(int channel, CTMode ctmode, int timeMS)
```

CounterStart

```
/// <summary>
/// 设置计时器工作模式并开始计数
/// </summary>
/// <param name="channel">指定计数器的通道</param>
/// <param name="mode">计数器的工作模式：测试沿跳变、周期计数、测试正脉宽、测试负脉宽
</param>
/// <param name="edge">测试沿跳变时：上升沿、下降沿设置</param>
/// <param name="periodCount">周期测试时的计数周期数</param>
public void CounterStart(int channel,CTMode mode,CTEdge edge,uint periodCount)
```

CounterRead

```
/// <summary>
/// 计数器读取计数值
/// </summary>
```

```
/// <param name="channel">指定的计数器通道</param>
/// <param name="mode">计数器工作模式</param>
/// <param name="periodCount">计数器周期测量时，参与运算的周期个数，该值越大计算越准确（50 以上 200 以下比较合适）</param>
/// <returns></returns>
public double CounterRead(int channel, CTMode mode, uint periodCount = 50)
```

CounterStop

```
/// <summary>
/// 计数器停止计数，并清除计数值
/// </summary>
/// <param name="channel">指定的计数器通道</param>
public void CounterStop(int channel)
```

MeasureVoltageAC

```
/// <summary>
/// 根据指定的量程、采样率、采样点数、谐波次数采集并分析指定通道的 AC 电压。
/// 包括分析：最小值、最大值、平均值、rms 值、ACrms 值、峰峰值、基频、THD、SNR、SINAD
/// </summary>
/// <param name="channel">需要采集分析的通道</param>
/// <param name="range">量程：10V、5V、2V、1V、500mV、200mV、100mV。请根据 DAQ 支持的量程填写</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samples">采样点数</param>
/// <param name="startBand">需要进行分析的开始频率</param>
/// <param name="endBand">需要进行分析的结束频率</param>
/// <param name="harmonicNumber">计算 THD 的谐波次数，0 表示不计算谐波，1 表示基波，2 表示二次谐波...</param>
/// <returns>返回结果包含：最小值、最大值、平均值、rms 值、ACrms 值、峰峰值、基频、THD、SNR、SINAD.其中 THD 值返回的是谐波与基频的比值，没有乘以 100%;SNR 和 SINAD 返回值的单位是 dB</returns>
public ACResult MeasureVoltageAC(int channel, VoltRange range, int sampleRate, int samples, int startBand, int endBand,int harmonicNumber = 0)
```

MeasureVoltageACMulti

```
/// <summary>
/// 根据指定的量程、采样率、采样点数、谐波次数采集并分析指定通道的 AC 电压。
/// 包括分析：最小值、最大值、平均值、rms 值、峰峰值、基频、THD、SNR
/// 返回结果按照通道从小到大排序
```

```
/// </summary>
/// <param name="channels">需要采集分析的通道</param>
/// <param name="range">量程：10V、5V、2V、1V、500mV、200mV、100mV。请根据 DAQ 支持的量程填写</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samples">采样点数</param>
/// <param name="startBand">需要进行分析的开始频率</param>
/// <param name="endBand">需要进行分析的结束频率</param>
/// <param name="harmonicNumber">计算 THD 的谐波次数，0 表示不计算谐波，1 表示基波，2 表示二次谐波...</param>
/// <returns>返回结果包含：最小值、最大值、平均值、rms 值、ACrms 值、峰峰值、基频、THD、SNR、SINAD.其中 THD 值返回的是谐波与基频的比值，没有乘以 100%;SNR 和 SINAD 返回值的单位是 dB</returns>
public ACResult[] MeasureVoltageACMulti(int[] channels, VoltRange range, int sampleRate, int samples,int startBand, int endBand, int harmonicNumber = 0)
```

MeasureVoltageACWithDatas

```
/// <summary>
/// 根据指定的量程、采样率、采样点数、谐波次数采集并分析指定通道的 AC 电压，同时返回采集到的原始数据
/// 包括分析：最小值、最大值、平均值、rms 值、ACrms 值、峰峰值、基频、THD、SNR、SINAD.
/// </summary>
/// <param name="channel">需要采集分析的通道</param>
/// <param name="range">量程：10V、5V、2V、1V、500mV、200mV、100mV。请根据 DAQ 支持的量程填写</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samples">采样点数</param>
/// <param name="startBand">需要进行分析的开始频率</param>
/// <param name="endBand">需要进行分析的结束频率</param>
/// <param name="harmonicNumber">计算 THD 的谐波次数，0 表示不计算谐波，1 表示基波，2 表示二次谐波...</param>
/// <returns>返回结果包含：最小值、最大值、平均值、rms 值、ACrms 值、峰峰值、基频、THD、SNR、SINAD.其中 THD 值返回的是谐波与基频的比值，没有乘以 100%;SNR 和 SINAD 返回值的单位是 dB</returns>
public MeasureACResult MeasureVoltageACWithDatas(int channel, VoltRange range, int sampleRate,
    int samples, int startBand, int endBand, int harmonicNumber = 0)
```

MeasureVoltageACMultiWithDatas

```
/// <summary>
/// 根据指定的量程、采样率、采样点数、谐波次数采集并分析指定通道的 AC 电压，同时返回采集
```

```
到的原始数据
/// 包括分析：最小值、最大值、平均值、rms 值、ACrms 值、峰峰值、基频、THD、SNR、SINAD.
/// 返回结果按照通道从小到大排序
/// </summary>
/// <param name="channels">需要采集分析的通道</param>
/// <param name="range">量程：10V、5V、2V、1V、500mV、200mV、100mV。请根据 DAQ 支持的量程填写</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samples">采样点数</param>
/// <param name="startBand">需要进行分析的开始频率</param>
/// <param name="endBand">需要进行分析的结束频率</param>
/// <param name="harmonicNumber">计算 THD 的谐波次数，0 表示不计算谐波，1 表示基波，2 表示二次谐波...</param>
/// <returns>返回结果包含：最小值、最大值、平均值、rms 值、ACrms 值、峰峰值、基频、THD、SNR、SINAD.其中 THD 值返回的是谐波与基频的比值，没有乘以 100%;SNR 和 SINAD 返回值的单位是 dB</returns>
public MeasureACResult[] MeasureVoltageACMultiWithDatas(int[] channels, VoltRange range, int sampleRate,
    int samples, int startBand, int endBand, int harmonicNumber = 0)
```

MeasureVoltageDC

```
/// <summary>
/// 根据量程，采样率，采样点数，测量指定通道的 DC 电压
/// </summary>
/// <param name="channel">测量通道</param>
/// <param name="range">量程：10V、5V、2V、1V、500mV、200mV、100mV。请根据 DAQ 支持的量程填写</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samples">采样点数</param>
/// <returns>返回测量到的 DC 电压平均值</returns>
public double MeasureVoltageDC(int channel, VoltRange range, int sampleRate, int samples)
```

MeasureVoltageDCMulti

```
/// <summary>
/// 根据量程，采样率，采样点数，测量指定通道的 DC 电压。
/// 返回结果按照通道从小到大排序
/// </summary>
/// <param name="channels">测量通道</param>
/// <param name="range">量程：10V、5V、2V、1V、500mV、200mV、100mV。请根据 DAQ 支持的量程填写</param>
```

```
/// <param name="sampleRate">采样率</param>
/// <param name="samples">采样点数</param>
/// <returns>返回测量到的 DC 电压平均值</returns>
public double[] MeasureVoltageDCMulti(int[] channels, VoltRange range, int sampleRate, int samples)
```

MeasureVoltageDCWithDatas

```
/// <summary>
/// 根据量程，采样率，采样点数，测量指定通道的 DC 电压，同时返回采集到的原始数据
/// </summary>
/// <param name="channel">测量通道</param>
/// <param name="range">量程：10V、5V、2V、1V、500mV、200mV、100mV。请根据 DAQ 支持的量程填写</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samples">采样点数</param>
/// <returns>返回测量到的 DC 电压平均值和原始数据</returns>
public MeasureDCResult MeasureVoltageDCWithDatas(int channel, VoltRange range,
    int sampleRate, int samples)
```

MeasureVoltageDCMultiWithDatas

```
/// <summary>
/// 根据量程，采样率，采样点数，测量指定通道的 DC 电压，同时返回采集到的原始数据。
/// 返回结果按照通道从小到大排序
/// </summary>
/// <param name="channel">测量通道</param>
/// <param name="range">量程：10V、5V、2V、1V、500mV、200mV、100mV。请根据 DAQ 支持的量程填写</param>
/// <param name="sampleRate">采样率</param>
/// <param name="samples">采样点数</param>
/// <returns>返回测量到的 DC 电压平均值和原始数据</returns>
public MeasureDCResult[] MeasureVoltageDCMultiWithDatas(int[] channels, VoltRange range, int sampleRate,int samples)
```

MeasureSNR

```
/// <summary>
/// 测量 SNR，计算公式：SNR = 20 * log10(SignalVrms / NoiseVrms)
/// 测量步骤：
/// 1.AO 输出 0V 电压，Sleep 2s 等待设备接收该信号，测量底噪的 rms
/// 2.AO 输出指定频率的电压，Sleep 2s 等待设备接收该信号，测量信号的 rms
```

```
/// 3.根据 SNR 计算公式求出 SNR
/// </summary>
/// <param name="AOChannel">AO 的输出通道</param>
/// <param name="AOFrequency">AO 的输出频率</param>
/// <param name="volt">AO 输出的电压值，单位：V</param>
/// <param name="AIChannels">AI 采集的通道数组</param>
/// <param name="sampleRate">AI 采样率</param>
/// <param name="samplePoints">AI 采样的点数</param>
/// <returns>返回各个通道测量到的 SNR 值数组</returns>
public SNRResult[] MeasureSNR(int AOChannel, int AOFrequency, double volt, int[]
AIChannels, int sampleRate, int samplePoints)
```

*MeasureSignalSNR

```
/// <summary>
/// 采集并分析外部信号的 SNR
/// </summary>
/// <param name="AIChannels">AI 采集通道数组</param>
/// <param name="sampleRate">总共的采样率，每个通道的采样率需要除以通道数</param>
/// <param name="samplePoints">每个通道的采样点数</param>
/// <param name="range">AI 的量程：±100mv,±200mv,±500mv,±1v,±2v,±5v,±10v</param>
/// <returns>返回一个数组，表示每个通道的 SNR 测量结果，结果包括：通道号、噪声 RMS、信号
RMS、SNR</returns>
public SNRResult[] MeasureSignalSNR(int[] AIChannels, int sampleRate, int
samplePoints, VoltRange range)
```

DeviceReadBuffer

```
/// <summary>
/// 读取 DAQ 设备的原始数据
/// </summary>
/// <param name="buffer">返回的原始数据</param>
/// <param name="packet">需要读取的数据帧数，一帧数据 512 字节</param>
/// <returns>是否读取成功</returns>
public bool DeviceReadBuffer(ref byte[] buffer, ref int packet)
```

DeviceParseBuffer

```
/// <summary>
/// 从采集到的原始数据中解析出实际有效的采样点数据
/// </summary>
/// <param name="buffer">采集到的原始数据</param>
```

```
/// <param name="datas">返回采样点数据</param>
/// <param name="len">返回采样点数据长度</param>
public void DeviceParseBuffer(byte[] buffer,ref byte[] datas, ref int len)
```

DeviceReadData

```
/// <summary>
/// 读取 DAQ 设备的采样数据，支持同步采样
/// 由于底层传输时根据 120 个采样点一帧数据，按帧传输，所以返回的数据可能比需要读取的多；
/// 例如读取 121 个采样点，实际会返回 240 个采样点
/// </summary>
/// <param name="channels">DAQ 采样的通道</param>
/// <param name="totalPoints">需要读取的采样点数</param>
/// <param name="maxVolt">采样时的量程</param>
/// <param name="datas">返回的 DAQ 采样数据</param>
/// <returns></returns>
public bool DeviceReadData(List<int> channels,int totalPoints,double maxVolt,
    ref List<ChannelDatas> datas)
```

GetAIChipNumber

```
/// <summary>
/// 根据传入的通道参数，计算出使用的 AI 芯片个数
/// </summary>
/// <param name="channels"></param>
/// <returns>通道包含的 AI 芯片数</returns>
public int GetAIChipNumber(List<int> channels)
```

GetAIChipChannelNumber

```
/// <summary>
/// 获取指定 AI 芯片的通道数量
/// </summary>
/// <param name="channels">使用到的通道号</param>
/// <param name="chip">AI 芯片的编号：0,1,2...</param>
/// <returns>指定 AI 芯片包含的通道数量</returns>
public int GetAIChipChannelNumber(List<int> channels,int chip)
```

GetAIChannelChip

```
/// <summary>
/// 获取指定 AI 通道所在的芯片号
```

```
/// </summary>
/// <param name="channel">要获取 AI 芯片号的通道</param>
/// <returns>通道所在的 AI 芯片编号</returns>
public int GetAIChannelChip(int channel)
```