LabJack U12
多功能数据采集控制器用户说明书

版本1.08

2004年4月10 日

北京迪阳世纪科技有限公司

BEIJING DIYANG SCIENCE &TECHNLLOGY CO.; LTD.

www.vlink.com.cn

地址:北京市海淀区北三环西路甲30号 邮编：100086

电话：010-62169728，62156134 传真：010-68400238

章节目录

插图目录

1 安装

LabJack U12 要求计算机操作系统是 Windows（视窗）98SE，ME，2000 或 XP。要确定操作系统版本，点击开始->设置->控制面板->系统->常规，确认版本号是 4.10.2222 或更高（Win98SE=4.10.2222, WinME=4.90.3000，Win2000=5.0.2195, WinXP=5.1.xxxx）。

对软硬件安装的先后次序没有要求。

如果在 Windows 98SE 上安装遇到困难，在求助于我们之前，请先阅读 Win98sehid.zip 中的 readme 文件。

1.1 硬 件 安 装

在计算机正常运行状态下，用提供的连接线把 LabJack U12 接到计算机的 USB 口上。这根 USB 连接线为 LabJack U12 提供了电源以及它与计算机间的通讯。状态发光二极管会快速闪 4 次（频率大约 4 赫兹），然后保持暗状态，这说明计算机正在访问查寻。

访问查寻是计算机操作系统对一个 USB 器件进行信息读取的过程，这些信息是用来描述被查询器件的名称及其功能。Windows 本身自带 LabJack U12 所需的底层驱动程序，接上 LabJack 后，系统自动开始访问查寻。Windows 对一个新器件进行首次访问查寻时可能需要一两分钟，Windows 可能会告诉你它正在安装驱动程序，如果 Windows 出现提示，请接受所有的缺省值。如果必要，重新启动计算机，此时可能需要视窗的安装光盘，请确定所提供光盘的视窗版本是否正确。访问查寻在每次连接时都会发生，但只需几秒钟。

访问查寻完成后，发光二极管会闪两次并保持亮状态，这说明视窗已正常地访问查寻了 LabJack。

如果 LabJack 不能被访问查询：

- 检查计算机的操作系统的版本号是 4.10.2222 或更高；
- 尝试将 LabJack 连到另一台计算机上；

- 尝试将其他 USB 设备连到该计算机上；
- 联系我们。

1.2 软件安装

虽然 Windows 已包含了 LabJack 所需的 USB 底层驱动程序，但是它还需要高级驱动程序来进行数据的传送和接收。随机提供的安装光盘将安装高级驱动程序、一些应用程序和一些范例程序的源代码。

插入安装光盘前，关闭所有打开的应用程序，特别是与 LabJack 相关的软件。安装程序通常会开始运行。如果安装程序没自动运行，你可用鼠标双击在光盘上的 LabJackVXXX.exe。

LabJack 安装结束后会安装美国国家仪器仪表公司的 LabView 运行引擎（LVRTE)。所安装的应用程序是需要该引擎才能运行的。如果有重启的提示，请照办。一些病毒扫描程序和 LVRTE 的安装可能有冲突。如果运行应用程序出错，重复以上安装直到 LVRTE 正确为止。

要测试安装正确与否，运行 LJTEST 程序。

开始=>程序=>LabJack=>LJtest

确定没选择“Test Fixture Installed”和“Continuous”，按运行（RUN）按钮，LJtest 应该一项项地测试并通过 8 个独立的测试。

2 硬件说明

LabJack U12 的外部特征有：

- USB 接口
- DB25 数字 I/O 连接器(D 线)
- 发光二极管状态指示灯
- 30 个螺丝接线柱

USB 接口提供电源和通讯。LabJack 无需外接电源。在各处的+5 伏电压接点都是输出口，千万不要接到电源上。

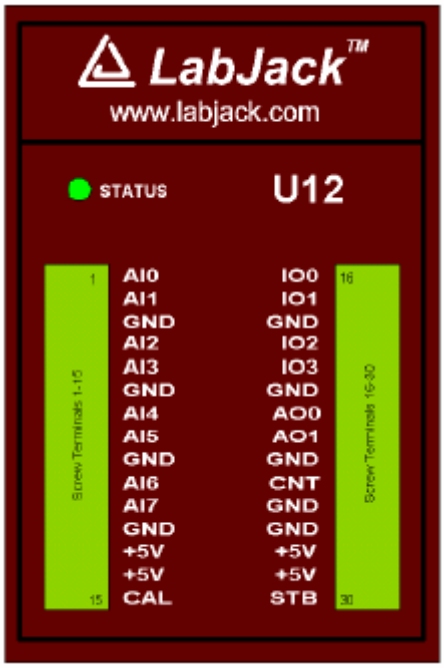


图 2-1. LabJack U12 的正面图

图 2-1 是 LabJack U12 的正面图。USB 和 DB25 的接口没有现出来。它们位于顶部的边上。DB25 插座有 16 根数字线，叫 D0-D15，它还有地线和+5 伏电源线。所有的线（包括 D0－D15）都接到那 30 个螺丝接线端上，见图 2-1。每个螺丝接线端都有标注，从 AI0 到 STB。

上电时，状态发光二极管会闪4次，在被计算机识别后会再闪1次后停留在亮的状态。若它没有被禁用，在进行短时读（Burst）和连续读（Stream）操作时也会闪。通过软件，如AISample, AIBurst, 或 AIStreamStart 等函数可以控制该发光二极管的亮暗。由于它要用4-5毫安的电流，在某些应用中，可以禁用它以减少负载。

2.1 AI0-AI7

硬件

LabJack U12 有 8 个模拟输入用的螺丝接线口，它们可以分别设置成单端输入或差动输入。每个输入具有 12 位的精度，输入偏置电流为±90 μA。

- 单端输入：输入的测量范围为±10 伏特
- 差动输入：差动输入可利用可编程放大器，放大倍数可达到 20 倍。这样可使得有效分辨率高于 16 位。在差动模式下，每个模拟输入端相对于地的电压必须在±10 伏之间，但两个输入端间的电压差范围由放大倍数(G)决定的。其关系如下：

序号	放大倍数	电压(伏)	序号	放大倍数	电压(伏)
1	G=1	±20	5	G=8	±2.5
2	G=2	±10	6	G=10	±2
3	G=4	±5	7	G=16	±1.25
4	G=5	±4	8	G=20	±1

在G=1时，范围为±20伏的原因是AI0可能是对地+10伏，AI1是-10伏，这样它们之间的电压便是+20伏；反之当AI0=-10伏，AI1=+10伏时，它们间的电压就是-20伏。PGA - 可编程增益放大器仅对差动输入有效。它对进入数模转换器前的信号进行放大。高级驱动程序会将读数除以放大倍数来获得实际的电压值。

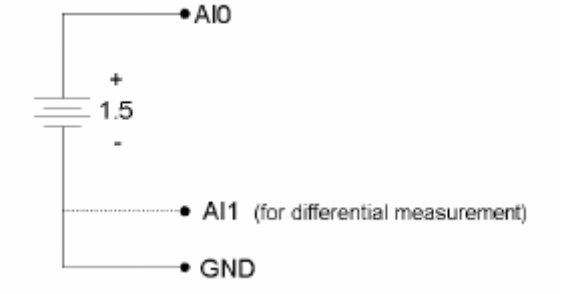


图 2-2. 单端方式的测量电路

图2-2是典型的单端测量电池电压电路。可以用差动的方法来做同样的测量，这样便可以用上可编程增益放大器。一般来说，任何单端的测量都可以用差动方式来实现，只要把信号接到偶数的输入端口，而相应的奇数端口接地，如图2-2的点虚线所示。

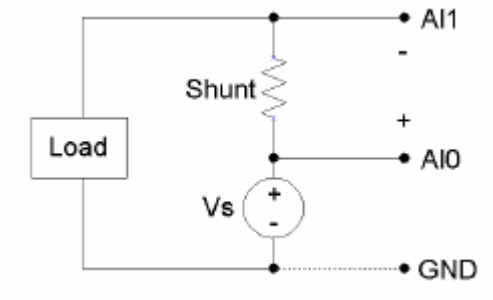


图 2-3. 差动测量接线图

图2-3是典型的差动测量接线图，它用于测量电阻上的电压。该电阻的任何一段都不是直接接地，所以必须用差动方式。但必须保证AI0和AI1对地的电压在±10伏之间。比如说图中的Vs是交流120伏，虽然AI0和AI1之间的电压可能很小，但AI0和AI1对地的电压会达到170伏左右,这样会严重地损坏LabJack。是否要连接地线(GND)完全取决于Vs的大小。

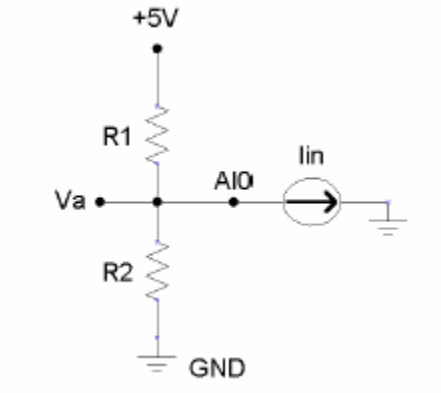


图 2-4. 单端方式的分压测量电路。

图2-4是用单端方式测量一个典型分压电路的输出电压。分压电路是一个把电阻量的变化（如感温电阻，光电阻，电位器等）转换成电压变化量的简单方法。如果Va端没有连接任何东西，R2的电阻值可以用下面的公式计算：

R2 = Va*R1 / (Vs-Va)

这里Vs是电源电压（在图中是+5伏）。

当Va接到AI0（如图），LabJack的偏置电流会影响到该分压器电路。如果R1和R2的值很大，这影响就必须考虑。所有测量设备都有输入偏置电流，该电流可达到毫安级。LabJack的偏置电流在+70到-94微安(μA)之间。这相当于100 kΩ 的输入阻抗。但因为在0伏时该电流不等0，最好使用下面的公式来建立模拟输入的电流模型：

$$I_{in} = 8.181 * Va - 11.67\ \mu A$$

已知输入偏置电流是输入电压的函数，简单的分压电路的方程可以修正成：

R2 = Va / [((Vs-Va)/R1) - (8.181μ * Va) + 11.67μ]

我们也可以用一个单电压跟随运放来消除偏置电流的影响从而使用简单的分压电路公式。这方法对在0-5伏间的单端信号都有效。可以用以下几种运放。

- TLV2462
- LMC6482
- MAX4166

软件

函数 EAnalogIn, AISample, AIBurst, and AIStreamRead 都返回模拟输入值。

EAnalogIn 是简化的函数（E就是指Easy - 容易），它返回一个模拟通道的值。其执行时间为20ms。

AISample 返回1到4通道的值，执行时间20ms，这样每通道的最高采样频率为50赫兹。

AIBurst 对1到4通道进行多次采样，硬件控制的采样频率为400到8192赫兹。采样可以由IO0或IO1的状态变化来触发。该函数也返回IO脚的状态。在内部，实际的采集和传输的样本数是2的次方，从64到4096，它最少是和numSamples（样本数）一样大。执行时间可以用下面公式来估计。

快速模式（此为默认模式）：

30+(1000*numSamplesActual/sampleRate)+(0.4*numSamplesActual)

一般模式：

30+(1000*numSamplesActual/sampleRate)+(2.5*numSamplesActual)

其中：

numSamples = numScans * numChannels

sampleRate = scanRate * numChannels

在连续读（Stream）模式下，AIStreamStart函数启动连续读响应，周期调用AIStreamRead函数来读取数据。每次读取将从LabJack的缓冲区中返回1-4通道的多个样本，也返回IO脚的状态。硬件定时采样频率在2000到1200之间。在连续读操作被启动后调用任何除AIStreamRead以外的函数都将停止连续读模式。

2.2 AO0 & AO1

LabJack U12有两个模拟电压输出的螺丝接线端。其电压输出在0和电源电压(一般为+5伏)之间，具有10位分辨率。输出电压是线性的，精度一般为±5%（见附录A）。如果要输出+5伏电压，其实际电压值会是100%的电源电压值。同样，要输出2.5伏，其实际值将是电源电压值的50%。在没有负载情况下，最大输出电压值几乎等于+5

伏的100%。但它会随着负载的增加而降低。

如果需要较高的精度，可以用模拟输入通道来测量+5伏的电压，然后计算出实际输出的电压值。比如要输出2.5伏的电压，+5伏的电源电压测量值为5.1伏，那么实际的输出就是2.55伏（空载情况下）。另一种更好的方法是用模拟通道来直接测量模拟输出端的值。

在每个模拟输出通道上有一个低通滤波器，其3dB截止频率为22赫兹。

在上电或复位时，模拟输出电压初始化成0伏。

模拟输出通道可以承受连续对地短路，不论其输出值大小。

模拟输出口是输出口，千万不要把他们接到任何电压源上。这些输出口上虽然有瞬间过压保护，比如对ESD（静电），耦合电压引起的连续过压。但是施加超过保护能力的电压有可能损坏LabJack板上的R63（AO0）或R62（AO1）电阻。损坏的症状是输出电压降低，尤其是在有负载的情况下。可以通过对R62/R63的测量来判断，它们的正常值应该小于50欧，如果大于50欧就说明有损坏。简单的修理方法是将电阻去掉并将其短接。

软件

用EAnalogOut或AOUpdate来输出电压，它们的执行时间为20ms，这相当于每通道的最大刷新频率为50赫兹。AOUpdate还控制和读取20个数字输入输出口的状态和计数器的计数值。

2.3 IO0 - IO3

这四个数字输入输出口是LabJack 20个输入输出口中的4个，它们接到LabJack的螺丝接线口上。它们可以分别设置成输入，高电平输出或低电平输出。这四个数字输入输出口各串联一个1.5KΩ电阻，该电阻起过压或短路保护作用。每个通道还有一个1MΩ的电阻接地。

通电或复位时，所有数字输入输出口均被设置成输入口。

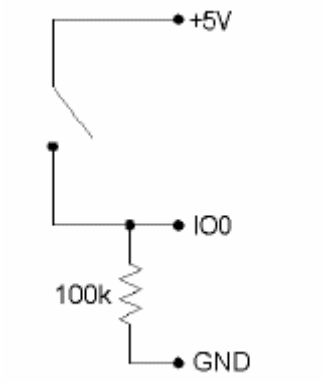


图 2-5. 数字输入口用于测量一个开关的状态，

图2-5，开关开时，IO0则读低电平，当开关关闭时，IO0则读高电平。1.5KΩ的电阻还限制了输出口的电流。比如当输出电流为1毫安时，该串联电阻会有1.5伏的压降，使得输出的电压为3.5伏。

软件

简单函数EDigitalIn 或 EDigitalOut 用于读写数字线的状态，它们都需要20毫秒执行时间。

函数AOUpdate 和 DigitalIO 可以设置每个口的输入输出方向和读写它们的数字状态。它们的执行时间也是20毫秒，即每个口的最大更新速率为50赫兹。

AISample 可以读写每个数字口,但对那些被设置成输入的数字口进行的写操作是无效的。函数Counter只能读取每个数字口的状态。

AIBurst 和 AIStreamRead 函数在读取模拟量的同时也读取这些数字量。这四个数字状态是被同时读取的，在短时读（Burst）模式下最高读取频率为2048赫兹，在连续读（Stream）模式下的读取频率为300赫兹。实际上这些函数并没有每次读取数字量，而是每四次才读取一次。

2.4 D0 - D15

这16路数字输入输出通道是通过DB25插座连接的，它们没有过压或短路保护。每路可提供25毫安的进出电流（但16路的电流能力总和是200毫安）。这使得这些口

可以用来直接驱动某些继电器。除D13到D15外各路都是CMOS输出和TTL输入。D13到D15是史密特（SCHMITT）触发输入。每个口都通过一个1 MΩ的电阻接地。

所有输入输出口在上电或复位时都被设置为输入。

DB25插座的接线如下表：

1: D0	6: D5	11: +5V	16: GND	21: D21
2: D1	7: D6	12: +5V	17: GND	22: D12
3: D2	8: D7	13: +5V	18: D8	23: D13
4: D3	9: NC	14: GND	19: D9	24: D14
5: D4	10:+5V	15: GND	20: D10	25：D15

这些口也可以像IO0-IO3一样用来检测开关的状态。

由于这些口没有过压/短路保护，用户要特别小心。一个串联电阻可以起到十分有效的保护作用。下列是一些可能会造成对这些口，乃至对LabJack的损坏：

- 高电平输出短接到地（或任何一个不是+5V的电压源）
- 低电平输出短接到一个非零的电压源（如+5V）
- 超过附录A指定的电压范围。

软件

简单函数EDigitalIn 或 EDigitalOut 用于读写数字线的状态，它们都需要20毫秒执行时间。

函数AOUpdate 和 DigitalIO 可以设置每个口的输入输出方向和读写数字状态。另外DigitalIO还返回目前输入输出的方向和各输出寄存器的状态。它们的执行时间也是20毫秒，即每个口的最大更新速率为50赫兹。

2.5 CNT

螺丝接线端CNT是接到一个32位计数器上的,它在检测到一个下降沿和一个上升沿后触发计数器加一。这意味着如果你在信号为低电平时复位，信号通过高-低-高后才会使计数器进一。在一些首次计数重要的情况下，你不应该使用复位函数，而

是把最终的计数减掉初始计数。它能对高达1MHZ的频率脉冲进行计数。

软件

ECount, AOUpdate 和 Counter 用来复位或读取计数器。 这些函数需要20毫秒执行时间，即最高更新率为50赫兹。

AIStreamRead 也读取计数器，其更新率可高达300赫兹。

2.6 CAL - STB

这两端口是用来测试和校正用的，校正端（CAL）是一个2.5伏的参考电压。它可以在正常操作模式下使用，但必须遵守附录A中指定的电流范围。虽然CAL端有一些保护措施（如ESD和过压），但严重的过压（长时间或瞬时）将会损坏CAL端，导致所有模拟输入口都不能使用。

2.7 +5V

LabJack有一个标称+5伏的内部电源。它被引到标有+5伏的接线端或者DB25的+5V脚。大多数台式计算机和带电源的USB盒可以提供450毫安的总输出电流。总输出电流是+5伏端口、模拟输出口和数字输出口输出的电流总和。一些手提电脑和总线驱动的集线器会限制最大电流在50毫安左右。

USB 规范要求所有USB控制器和集线器有过流保护。如果用户在LabJack的+5伏端接上过大的负载（包括+5伏端对地短路），USB控制器和集线器应负责过流保护。

2.8 GND

地线可在标有GND的螺丝端口和DB25插座得到,它是LabJack所有功能的公共地。

2.9 OEM 版本

LabJack U12 有两种OEM (Original Equipment Manufacturer) 版本:

- LJU12-PH: 这版本上没有螺丝接线柱，取而代之的是双列针式插头。所有

元器件都在原件面（包括发光二极管），反面没有安装任何东西。

• LJU12-NTH: 这版本上没有安装任何穿孔的元器件（如DB25接插件，USB接插件，发光二极管和接线头）。该板是让用户直接将接线焊到印刷电路板上，或自行安装接插件。

注意：这些板子不带任何附件。

3 应用范例程序

安装光盘上附有9个应用范例:LJconfig, LJcounter, LJfg, LJlogger, LJscope, LJstream, Ljtest, LJSHT, 和 LJSHTmulti。

LJconfig: 该程序列出所有连接在该计算机上的 LabJack。可以用它来设定某个 LabJack 的设备号。这对有多个 LabJack 的情况特别有用。每个 LabJack 都有自己的设备号（LocalID）和序号（serial number)。设备号是一个在 0 和 255 之间的数值，而该值可以由用户改变。序号是在 256 到 2,147,483,647 之间是一个数值。每个 LabJack 都有一个唯一的序号，用户是不能改变它的。

LJcounter: 用来读 LabJack 的计数器，它提供频率或计数。

LJfg （函数发生器）: 输出一些基本波形到 AO0 上的。

LJlogger: 在指令/响应模式下发送或接收来自各通道的数据。保存数据到存储介质，写数据到网页上和在时间触发下执行各种任务（包括发送电子邮件)。

LJscope: 使用短时读（Burst）模式读取两个模拟输入通道来模拟一个示波器

LJstream: 使用连续读（Stream）模式来读取 4 个模拟输入通道的数据，并对它们制图和存盘。

LJtest: 对 LabJack 本身进行一系列测试。用户一般不选“Test Fixture Installed”，而且 LabJack 除 USB 线外不接任何东西。

LJSHT: 读取并记录来自一个或两个 EI-1050 的数字温度/湿度测量仪的数据。

LJSHTmulti:显示从多达 20 个 EI-1050 数字温度/湿度测量仪数据。

大多数的 LabView 程序源代码在 Examples 的目录下。大多数的程序有中英文两个版本。

3.1 LJconfig

每个 LabJack 都有自己的设备号（Local ID）和序号（serial number）。设备号是一个在 0 和 255 之间的数值，而该值可以由用户改变。序号是在 256 到 2,147,483,647 之间是一个数值。每个 LabJack 都有一个唯一的序号，用户是不能

改变它的。LJConfig 是用来设定某个 LabJack 的设备号的。

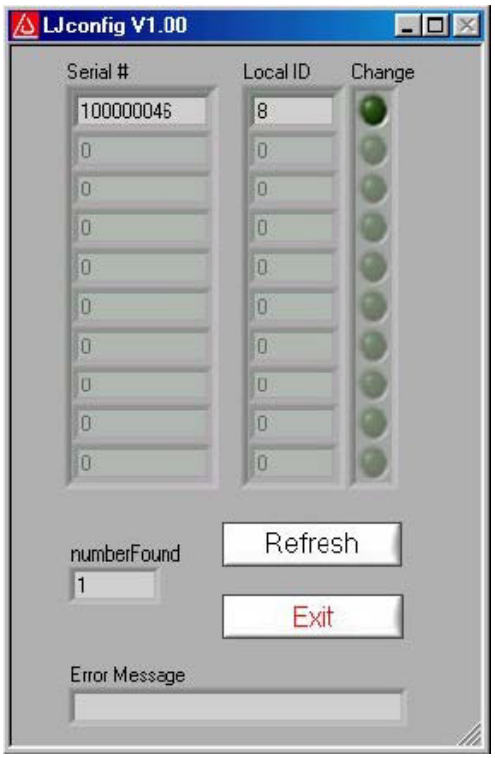


图 3-1. LJconfig

图 3-1 是当 LJConfig 运行时的窗口。当按下 Refresh 键时，该程序会查询 USB 口来获得所有 LabJack 的信息。要改变某个 LabJack 的设备号（Local ID），则按下在其相应处的 Change（改变）按钮，图 3-2 就会出现。

图 3-2. LJconfig Change Local ID

输入一个新的设备号（在 0 到 255 之间）后按 Change 按钮。这个新的设备号就会被写进 LabJack 并重新进行访问查询。

该程序只有英文版。

3.2 LJcounter

用来读 LabJack 的计数器以获得当前的频率或计数。

图 3-3. LJcounter

图 3-3 是 LJcounter 的窗口：

- 间隔（秒）： 指定调用 DLL 函数 “Counter” 的时间间隔。
- 测量方式： 如选“频率”，程序将读到的计数器数据除以间隔来计算出频率，单位是赫兹。每次读后计数器复位。如选“计数”，测量值则为计数器的当前值。
- 测量值： 显示频率或计数。取决于选中的测量方式。

3.3 LJfg

该程序用LabJack来作为简单的函数发生器。每25毫秒 AOUpdate 被调用一次，即更新率为 40 赫兹，因此输出有意义的信号只有几赫兹。

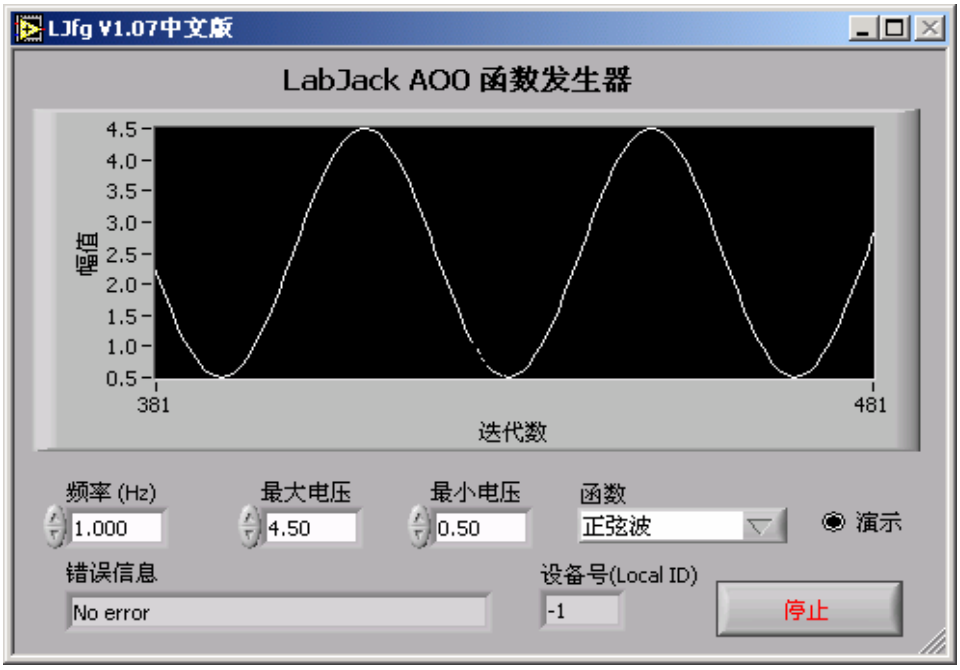


图 3-4. LJfg

3.4 LJlogger

该程序调用了两个“AISample”和一个“AOUpdate”，在指令/响应模式下收发数据。它可以把数据存盘（最快两赫兹）和在事件触发下执行各种任务。

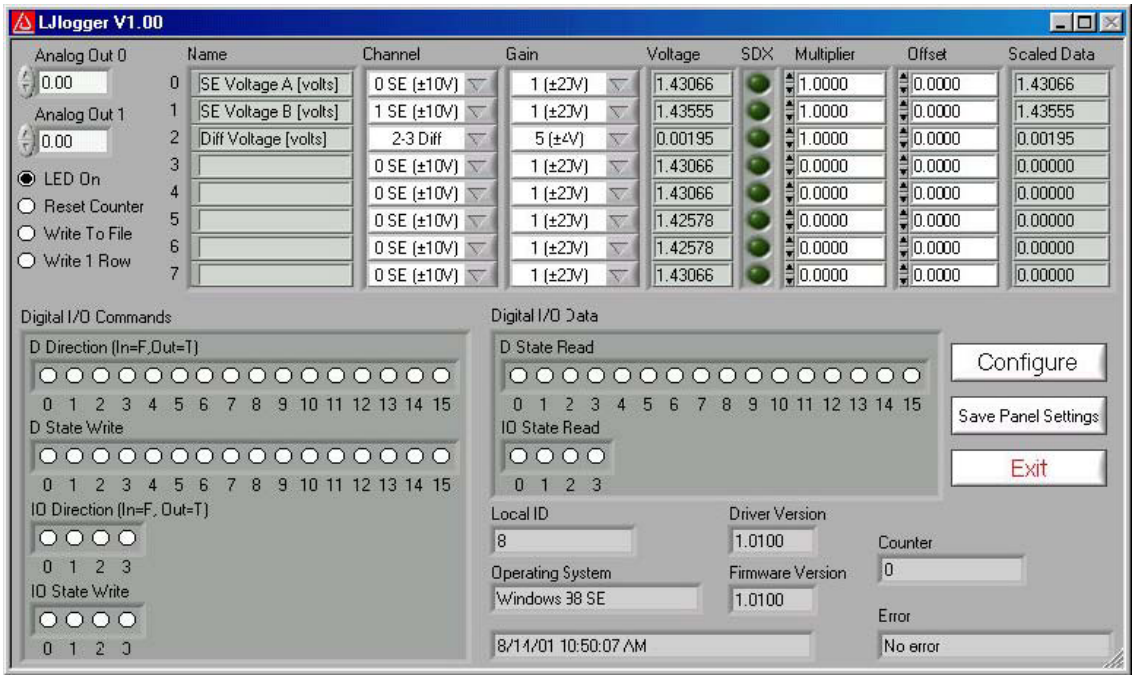


图 3-5. LJlogger

该程序的主窗口如图 3-5 所示。白色区域和“SDX”按键是用户输入或选择的项目。灰色的区域是用来显示信息的。按下“Save Panel Settings”会将当前的各输入值设置成默认值。如果某个输入口的“SDX”被选，那么相应的 SDX 动态链接库会用来计算 “Scaled Data”。用户可以编写自己的 SDX 动态链接库（详见源程序）来进行复杂的运算。该程序只有英文版本。

按 “Configure” 键 会进入图 3-6 所示窗口。

- **Working Directory:** 这是数据和配置文件存储的目录。
- **Data File Name:** 这是数据文件名。数据将写入该文件，新数据也会被不断加到该文件的末端。
- **Data File Write Interval:** 指定数据存文件的时间间隔。最小为 0.5 秒。
- **HTML Write Interval:** 决定了更新 HTML 文件的时间间隔。

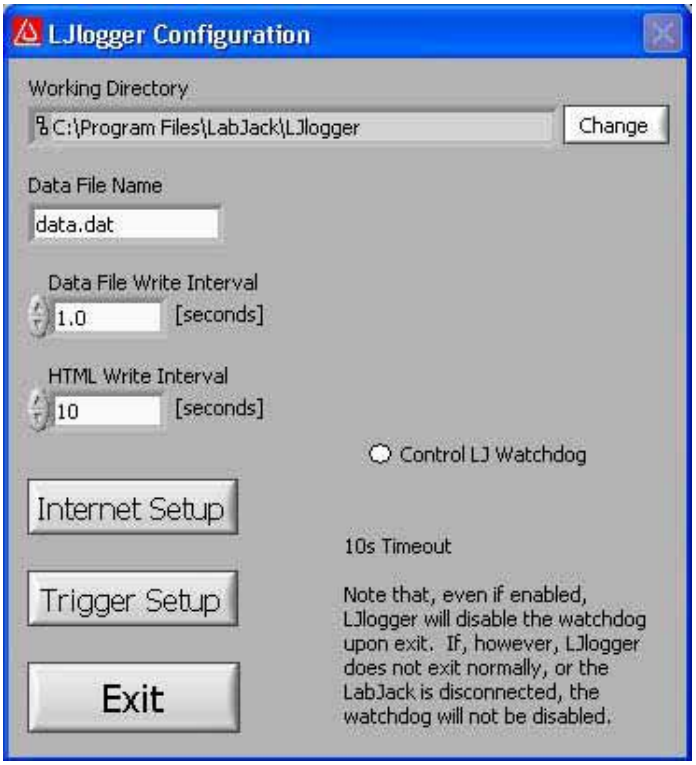


图 3-6. LJlogger Configuration

按 “Internet Setup” 键 （图 3-6）会弹出英特网的配置窗口，见图 3-7。对 HTML 文件的基本配置可以按 “Advanced HTML Configuration” 键弹出图 3-8

的窗口。

在图 3-6 所示窗口中按 “Trigger Setup”键会弹出如图 3-9 的窗口。

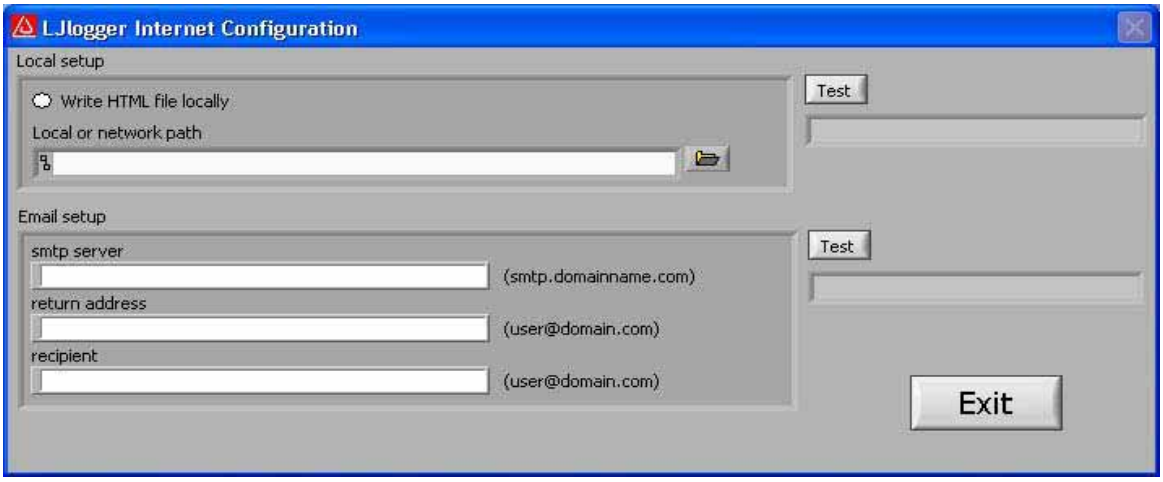


图 3-7. LJlogger Internet Configuration

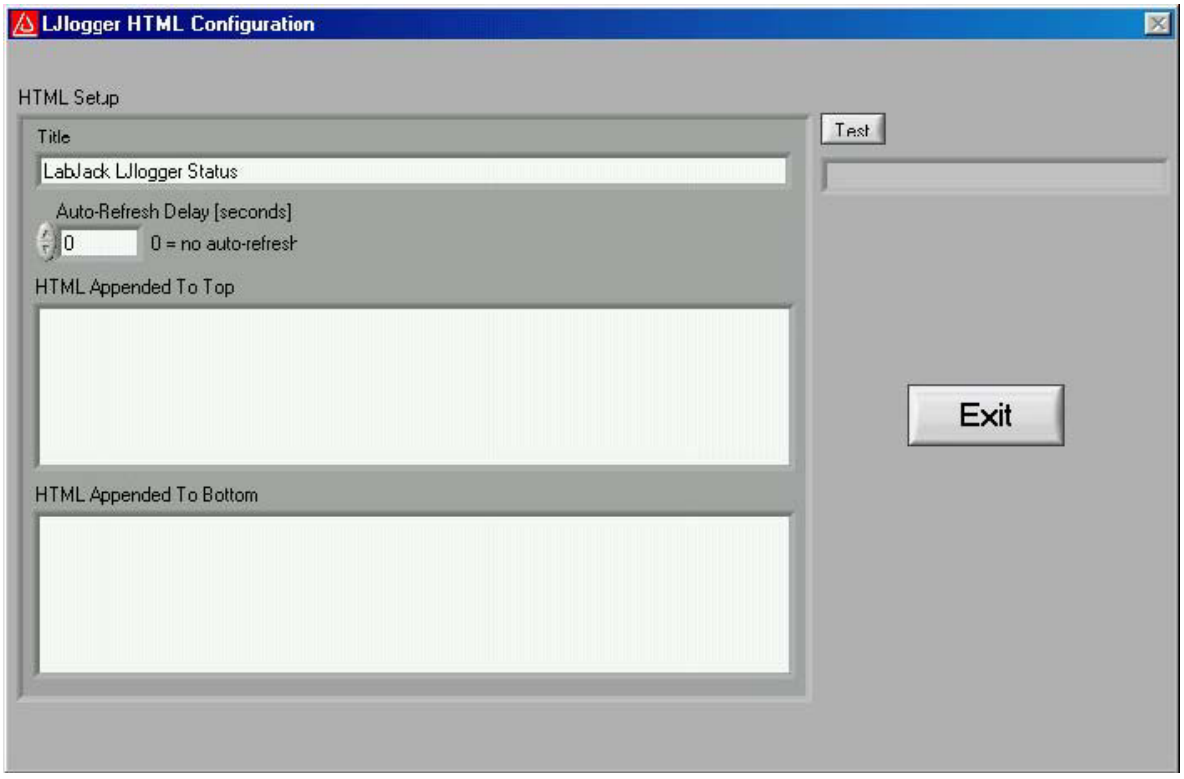


图 3-8. LJlogger HTML Configuration

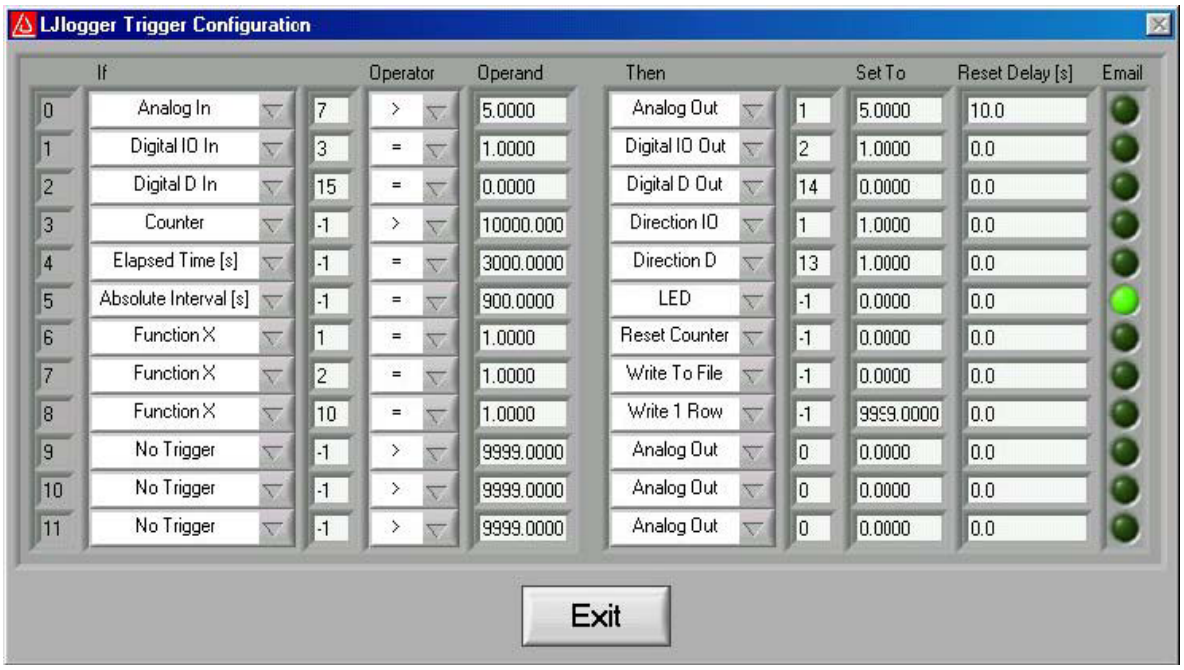


图 3-9. LJlogger Trigger Configuration

图3-9 展示了九种触发例子：

- **触发事件#0：** 如果图3-5中模拟输入第七行的Scaled Data（计算值）大于第五行，那么将AO1设置成5伏。每次触发后要等待10秒事件才会再次被触发。
- **触发事件#1：** 如果IO3是高电平，将IO2 置高。“Reset delay”在此为零，所以只要IO3是高电平，每次更新数据时都会触发该事件。
- **触发事件#2：** 如果D15位低电平，置D14为低。
- **触发事件#3：** 如果计数器大于10,000，把IO1口设置为输出口。
- **触发事件#4：** 如果LJLogger运行多于3000秒，则把D13设置为输出口。
- **触发事件#5：** 计算机时钟每隔15分钟，状态发光二极管就暗掉并送出一个电子邮件。
- **触发事件#6：** 从function1.dll 这个动态链接库中调用函数“FunctionX”。如果函数返回值为真（TRUE），则将计数器复位。 用户可以编写自己的动态链接库，详见源代码。
- **触发事件#7：** 从function2.dll 这个动态链接库中调用函数“FunctionX”。如果函数返回值为真（TRUE），则停止向文件写数据。

• **触发事件#8：** 从function10.dll 这个动态链接库中调用函数“FunctionX”。如果函数返回值为真（TRUE），则将第一行数据写入数据文件。

3.5 LJscope

LJscope 以短时读（Burst）模式来读取两个模拟输入通道，它是个模拟的示波器。

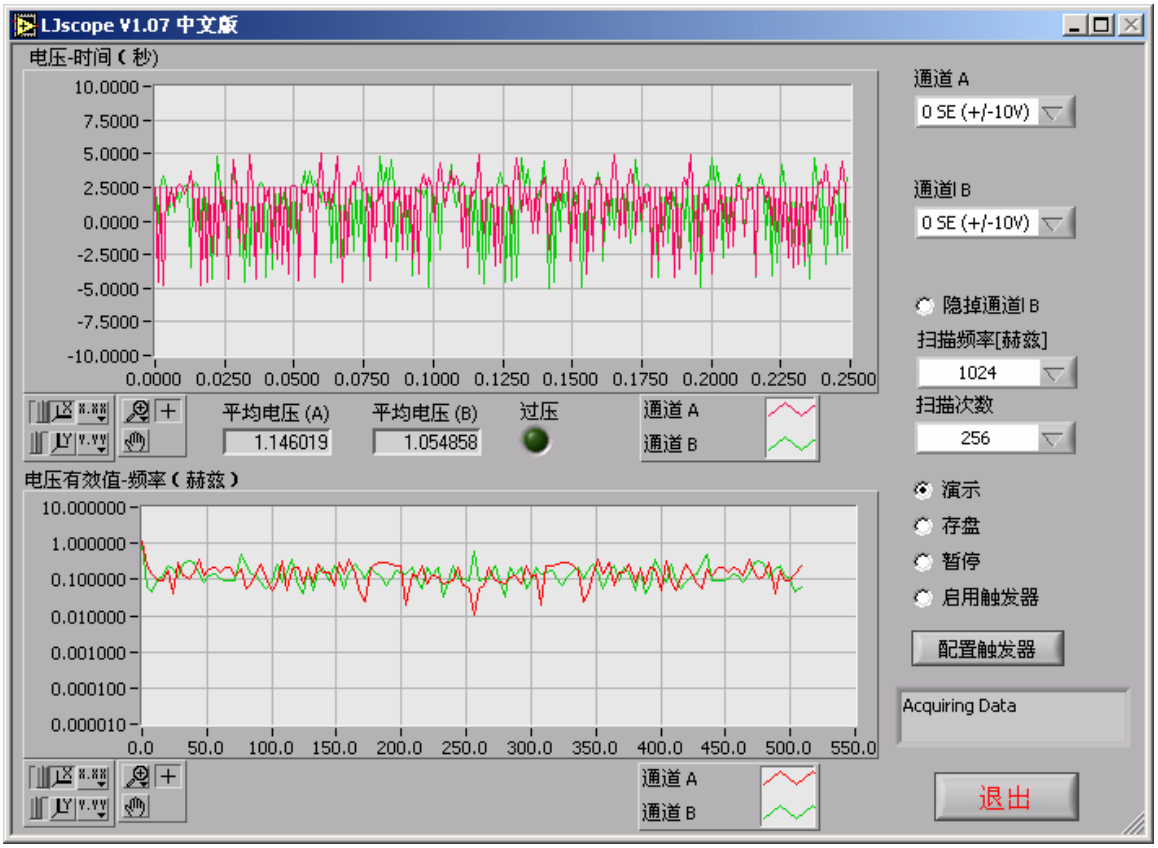
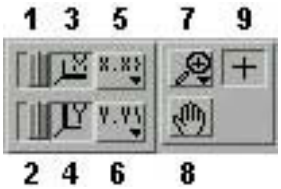


图 3-10. LJscope

在图 3-10 所示的 LJScope 主窗口上有两个图形显示器。它们分别显示电压-时间的关系和电压-频率的关系。各图都有相应的工具来对显示器进行操作，如自动调整显示范围大小，图形方大等。 工具的操作解释如下。



1. 按下它会锁住按键 3。

2. 按下它会锁住按键 4。

3. 按下它会自动调节 X 轴。

4. 按下它会自动调节 Y 轴。

5. 其它 X 轴的调接功能。

6. 其它 Y 轴的调节功能。

7. 放大工具。按下它会得到许多不同的放大工具。在采集数据过程中，放大功能不会工作，除非自动调节功能被禁用。

8. 按下它你可以移动曲线到任何位置。

9. 没有用到。

其它 LJscope 的控制包括：

• **通道 A 和通道 B:** 选择被采集的两个模拟通道。如果选择了差动输入通道，增益控制器会显示出来。

• **隐掉通道 B:** 按下它 B 通道就不会显示在图上。

• **扫描频率 [赫兹]:** (256 到 4096) 指定了每秒所进行的扫描数。

• **扫描次数:** (32 到 2048) 指定了采集的扫描数，即总的采集时间。例如以 4096 赫兹的扫描速率来采集 1024 个扫描次数,那么它将采集四分之一秒的数据。(如图 3-10 所示)

• **演示:** 在演示模式下调用 “AIBurst” 函数，这样所有时序和数据都是内部产生的。

• **存盘:** 选择它会弹出要求输入文件名的对话框，当前采集的数据将被存入文本格式的文件（时间，通道 A，通道 B)。

• **暂停:** 暂停数据采集。

• **启用触发器:** 启用数字输入输出口触发器。

• **配置触发器:** 弹出图 3-11 所示的窗口。选择触发口和触发状态（高电平触发或低电平触发)。指定一个超时时间，这样使程序不会无休止地等待触发信号。

图 3-11. LJscope

3.6 LJstream

是用连续读（Stream）模式来采集 4 个模拟通道的信号，显示它们的图形，把它们存入文件中。详情还可见连续读的函数 (AIStreamStart, AIStreamRead, 和 AIStreamClear)。

- **启用连续读:** 开始和结束连续数据采集。
- **扫描频率:** 指定了每秒的扫描数(在 50 到 300 之间)。
- **扫描次数:** 指定了每次更新所采集的扫描数，从而决定了该程序对数据更新的速度。
- **演示:** 在演示模式下调用“AIStream” 函数。所有数据将是内部产生的。
- **读计数器:** 如被选中，采集 1 个模拟输入和计数器的计数值。
- **配置通道:** 按下它会弹出如图 3-13 所示的通道配置窗口。
- **存储当前设置:** 把当前的设置，包括通道的设置都保存起来。
- **图形历史:** 指定了图形上显示多少个历史数据。

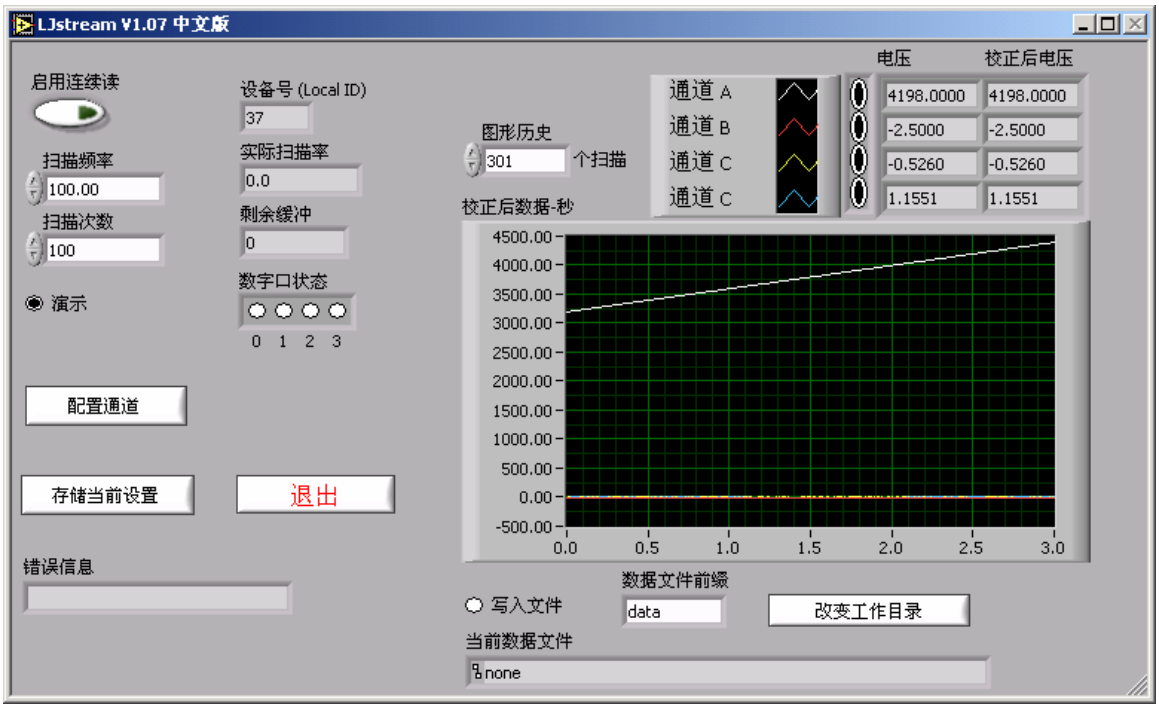


图 3-12. LJstream

图 3-13 是 LJstream 的通道配置窗口。你可以选择模拟输入通道，增益和计算公式。用“测试数据”来看计算公式的效应(“v”列是实测的电压值，”y”列是计算后的值)。“手动输入/采样数据”选择了“测试数据”中“v”列数据的来源。

图 3-13. LJstream Channel Configuration

3.7 LJtest

LJtest 对 LabJack 本身进行一系列的测试。 用户一般不选“Test Fixture Installed”， 而且 LabJack 除 USB 线外不接任何东西。

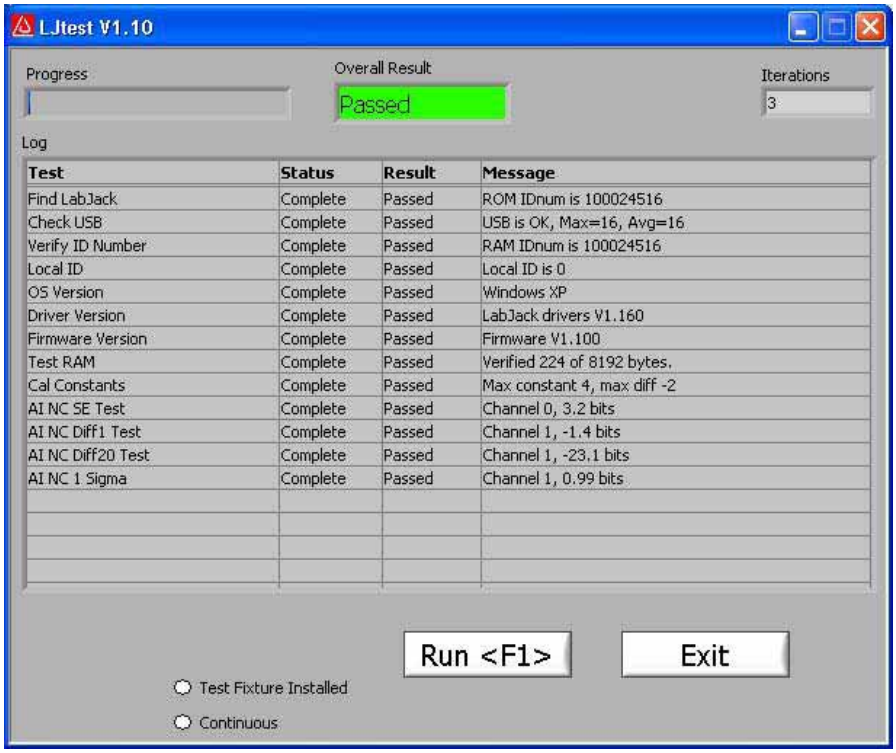


图 3-14. LJtest

如果除“OS Version”（操作系统版本）和“Driver Version”（驱动程序版本）外的测试都失败，一般来说其原因是 LJTest 没有检测到 LabJack 的存在。检查上电时 LabJack 上的状态发光二极管是否闪烁。如果使用 Windows 98 SE， 则请查看 Win98sehid.zip 中的有关文件。

如果“Find LabJack”（查找 LabJack）是唯一的失败项，通常是因为有多个 LabJack 的缘故。

“Check USB”（检查 USB）对 USB 进行了一些很基本的检查，用来检查 USB 是否有什么很明显的问题。该检查需要对 LabJack U12 进行正常的通讯。

“Local ID”（设备号）在出厂时总被设置为零。如果用户把它改变为一个非零值，这一项在该测试中会有黄色的警告。

“Test RAM”（测试内存）或任何“AI …”（对模拟通道的测试）测试项的失败提示你 LabJack 可能已被损坏。先检查 LabJack 上没有接任何除 USB 线外的东西，如果还是不能通过测试，请与技术服务点联系。在 LabJack 不接任何东西下，“AI …”（对模拟通道的测试）的黄色警告提示你可能需要运行一次自校正了。

“Cal Constants”（校正常数）测试项的黄色警告一般是因为所有的常数都已被置零。更经常是因为用户选择了“Test Fixture Installed”或者连接不当。

要写入一组新的校正数据需要使用该程序运行自校正。你需要准备 12 个小跳线（大约 5 厘米长即可）。

1)照下列接线：

- AI0 <=> AI2 <=> AI4 <=> AI6 <=> +5V
- AI1 <=> AI3 <=> AI5 <=> AI7 <=> +5V
- IO0 <=> IO1
- IO2 <=> AO0
- IO3 <=> AO1
- CNT <=> STB

2) 运行 LJTest 并选中"Test Fixture Installed" 和 "Prompt During Cal"，然后按下“Run”（运行）。

3) LJtest 将进行一步步的测试，在出现提示“connect GND to all 8 AI channels”时，将所有 8 个模拟输入端接地。在出现提示“Connect CAL to AI 0,2,4,6 and GND to AI 1,3,5,7”时，将所有偶数的模拟输入口接到 CAL，而将所有奇数的模拟输入口接地（GND）。在出现提示“Connect CAL to all 8 AI channels”时，将所有的模拟输入口接到 CAL。在出现提示“Connect GND to AI 0,2,4,6 and CAL to AI 1,3,5,7”时，将所有奇数的模拟输入口接到 CAL，而将所有偶数的模拟输入口接地（GND）。

4) 完成后除去所有的跳线并拔出 USB 线。重新连上 USB 线后新的校正数据会被使用。然后再运行 LJTest 一次，这次就不选“Test Fixture Installed”，确认所有测试通过。

3.8 LJSHT

用于从 EI-1050 数字温度/湿度传感器读取数据并记录数据。

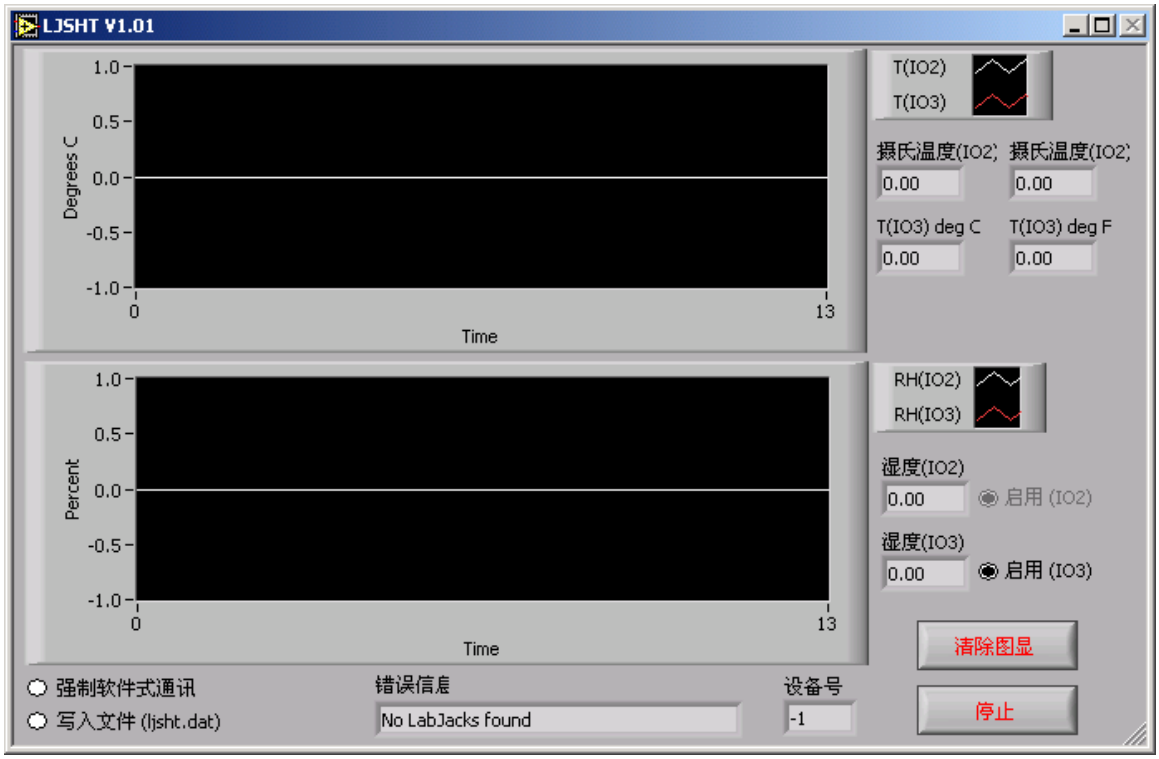


图 3-15. LJSHT

图3-15是LJSHT的窗口。

• **启用(IO2/IO3)**：至少要连接一个EI-1050。IO2 用来作为该传感器的控制线。如有两个传感器，则要用IO3 作为第二个的控制线。

• **强制软件式通讯**：即使 LabJack U12 的内在程序版本是 V1.10 或更高，你也可以用该项来强行使用软件式 SHT1X 通讯模式。

• **写入文件**：使数据被加到一个文本文件末尾。该文件是在当前目录下，叫“ljsht.dat”

3.9 LJSHTmulti

用一个 LabJack 来读取和显示来自多达 20 个 EI-1050 数字温度/湿度传感器。

图 3-16. LJSHTmulti

图3-16是LJSHTmulti的窗口:

- **启用控制口:** 选择一个LabJack的输出口来控制每个EI-1050传感器。
- **强制软件式通讯:** 即使LabJack U12 的内在程序版本是V1.10 或更高，你也可以用该项来强行使用软件式SHT1X通讯模式。
- **控制延时:** 在每次读操作之间加入一个延时。这常在调试时用。

4 编程参考

安装光盘会安装高级驱动程序(ljackuw.dll)、高级驱动程序的 ActiveX 接口和调用这些动态链接库的 LabView 子程序。DLL 和 OCX 安装在 Windows 的系统目录中。如果安装程序能找到 LabView 的目录，它会把 LabView 的子程序拷贝到“\vi.lib\addons\”目录下。这样他们会出现在 LabView 的功能板上。否则，这些子程序会被复制到” c:\Program Files\LabJack)\drivers\labview”目录中。

LabJack DLL 中有 38 个函数，OCX 和 LabView 子程序中也有相应的函数。由于 ActiveX 的限制性，OCX 中有两个附加的函数。除了 AIBurst 和 AIStreamStart/Read/Clear，所有函数都是指令/相应模式。

大多数函数使用了下列两个参数：

- errorcode - LabJack 特定的错误码。0指没有错误，2指没有找到LabJack。用“GetErrorString”函数可以获得错误信息或见本文件中4.24段。
- idnum - 该参数可以是设备号，系列号，或者-1。设备号或系列号指定某个LabJack，而-1指所能找到的第一个LabJack。每个LabJack都有设备号和系列号。设备号是在0和255之间的一个数，用户可以改变它。系列号是256 到 2,147,483,647 之间的一个数。每个LabJack都有一个唯一的系列号，用户是不能改变它的。

为了能让更多的编程语言调用，尽量使用基本的变量类型。所有声明都使用C写的。在ActiveX中如有不同，我们都会详细说明的。

参数前的“*”号说明该参数是个指针。这样的参数可以是输入，也可以使输出，而非指针参数一定是输入。有时指针不是指向一个单一值的，而是指向一个数组。这在参数说明中都会提到。

一些数字口的参数用一个值来包含每位 I/O 口的信息,每一位 I/O 口在参数中都有其对应的位（如参数 trisD 中的第 0 位对应设置数字口 D0 的输入输出方向）。比如在 DigitalIO 函数中，参数*trisD 是指向以内存的指针，而该内存的值表示了 16 个数字线的方向：

• 如果*trisD 指向的值是0，那么所有的数字线将都是输入线。

• 如果*trisD 指向的值是1 (2^0)，那么D0是输出，D1-D15 是输入。

• 如果*trisD 指向的值是5 (2^0 + 2^2)，那么D0 和 D2 使输出，而其他都是输入。

• 如果*trisD 指向的值是65535 (2^0 + … + 2^15)，则D0-D15 都是输出。

*trisD 所致的值的范围是 0 到 65535。当调用 DigitalIO 时，如果 updateDigital >1，那么所有的数字线都会根据 *trisD 所指向的值被设定成输入或输出。当 DigitalIO 返回时，*trisD 所指向的值也和 LabJack U12 中方向寄存器中的状态相对应。

4.1 EAnalogIn

这是个简单函数，是 AISample 函数的简化版本，它只读一个模拟输入通道的电压。调用该函数会使状态发光二极管变亮或保持亮的状态。

执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

声明:

```
long EAnalogIn(  long    *idnum,
                 long     demo,
                 long     channel,
                 long     gain,
                 long     *overVoltage,
                 float    *voltage )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

• ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）

• **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。

• **channel** - 采集通道。单端时用0-7，差动时用8-11。

• **gain** - 增益。0=〉1, 1=〉2, …, 7=〉20。只在差动通道采集时有效。

输出:

- *idnum - 返回设备号或-1（如果没有找到LabJack的话）
- *overVoltage - 如果大于0， 则说明在被选择的通道上信号超界。
- *voltage - 返回采集的电压值。

4.2 EAnalogOut

这是 AOUpdate 的简化版本，它用来输出两个模拟输出通道的电压。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

如果要输出的值小于 0，DLL 将输出上次的输出值。这样就可以只改变一个通道的电压值。注意当 DLL 刚刚载入时，它并不知道原先的值是什么，它只能认为它们都是 0 伏。同样当 LabJack 不是被 DLL 复位的情况下，DLL 记忆的原先的电压值可能不是 0，而实际的值却已经为 0。

声明：

```
long EAnalogOut ( long  *idnum,
                  long  demo,
                  float analogOut0,
                  float analogOut1 )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- *idnum - 设备号，系列号或-1（所能找到的第一个设备）
- demo - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- analogOut0 - 要输出到AO0的0.0到5.0伏的电压。
- analogOut1 - 要输出到AO1的0.0到5.0伏的电压。

输出:

- *idnum - 返回设备号或-1（如果没有找到LabJack的话）

4.3 ECount

这是 AOUpdate 的简化版本，它用来读取计数器或复位计数器。调用该函数会使 STB 被禁用（STB 被禁用是默认的状态）。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

声明：

```
long ECount( long *idnum,
             long demo,
             long resetCounter,
             double *count,
             double *ms )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有 LabJack的情况下调用该函数。
- **resetCounter** - 如果它大于0， 计数器则被复位。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）
- ***count** - 被复位前的计数值。
- ***ms** - 读取计数器时的时间。该事件是Windows毫秒定时器的时间，单位是毫秒。请注意：该Windows毫秒定时器每50天循环一次。一般来说该毫秒计数器在计算机重启后会重新从0开始计数。

4.4 EDigitalIn

这是 DigitalIO 的简化版本。 它是用来读取一个数字输入口的状态。它还把指

定的口设置成输入口并保持它的输入方向。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

注意：该函数可以对所有 20 个数字口进行操作。DLL 总是记住当前的数字口的输入输出方向并保持所有输出口的状态。所以该函数只对某个口进行操作而不改变其它口的情况。 但是当 DLL 刚刚载入时，它并不知道方向和状态，所以只能认为所有口的方向是输入，在所有口上的电平为低。

声明：

```
long EDigitalIn ( long *idnum,
                  long demo,
                  long channel,
                  long readD,
                  long *state )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **channel** - 要读取的数字口号。IO口是0-3，D口是0-15。
- **readD** - 大于0时读取D口，否则读取IO口。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）
- ***state** - 非零值表示高电平，0 表示低电平。

4.5 EDigitalOut

这是 DigitalIO 的简化版本，它用来输出一个数字量。 它还把指定的数字口设置成输出口并保持它的输出方向。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

注意：该函数可以对所有 20 个数字口进行操作。DLL 总是记住当前的数字口的

输入输出方向并保持所有输出口的状态。所以该函数只对某个口进行操作而不改变其它口的情况。 但是当 DLL 刚刚载入时，它并不知道方向和状态，所以只能认为所有口的方向是输入，在所有口上的电平为低。

声明：

```
long EDigitalOut ( long *idnum,
                   long  demo,
                   long  channel,
                   long  writeD,
                   long  state )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack设备的情况下调用DLL
- **channel** - 要读取的数字口号。IO口是0-3，D口是0-15。
- **writeD** - 大于0时读取D口，否则读取IO口。
- **state** - 大于零则输出高电平，否则输出低电平。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）

4.6 AISample

从 1，2，或 4 个模拟输入口读取电压。还可以控制和读取 4 个 IO 口。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

声明：

```
long AISample ( long *idnum,
                long demo,
                long *stateIO,
                long updateIO,
                long ledOn,
```

```
long numChannels,
long *channels,
long *gains,
long disableCal,
long *overVoltage,
float *voltages )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- ***stateIO** - 要输出的IO0-IO3的状态。如果IO是被设置成输入口，输出状态位对其没有作用。可以用其他的函数来把它们设置成输出方向。
- **updateIO** - 大于0，则输出状态值，否则只进行读操作。
- **ledOn** - 大于0时，LabJack的发光二极管被点亮。
- **numChannels** - 要读取的模拟通道数(1,2, 或 4)。
- ***channels** - 这是一个数组的指针，该数组至少要有numChannels 个单元。 单端输入用0-7，查动输入用 8-11。
- ***gains** - 这是一个数组的指针，该数组至少要有numChannels 个单元。个单元值是响应通道的增益指令。0=〉1, 1=〉2, …, 7=〉20。 只在差动通道采集时有效。
- **disableCal** - 大于0时，返回的电压是没有用校正系数进行校正计算的值。（叫做原始值）。
- ***voltages** - 这是一个数组的指针，返回的电压值将在该数组中。可以送一个4个单元均为0的数组。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）
- ***stateIO** - 返回的 IO0-IO3 的状态。

• *overVoltage - 如果大于0， 则说明在被选择的通道上信号超界。

• *voltages - 这是一个数组的指针，返回的numChannels个电压值在该数组中。

ActiveX 函数的不同点:

在 ActiveX 函数中，“channels” 和 “gains” 数组被换成“channelsPacked” 和 “gainsPacked”。 OCX 中有一个叫 “FourPack” 的函数，该函数把4个单元的数组元素（element[ ]）转换成一个数值（PackedValue）。其公式为：

PackedValue = element[0] + (element[1] * 2^8) + (element[2] * 2^16) + (element[3] * 2^24)。

“voltages” 数组被换成4个当独的参数。

声明 (ActiveX):

```
long AISampleX ( long FAR* idnum,
                 long demo,
                 long FAR* stateIO,
                 long updateIO,
                 long ledOn,
                 long numChannels,
                 long channelsPacked,
                 long gainsPacked,
                 long disableCal,
                 long FAR* overVoltage,
                 float FAR* voltageA,
                 float FAR* voltageB,
                 float FAR* voltageC,
                 float FAR* voltageD )
```

4.7 AIBurst

从 1，2，或 4 个模拟通道，以指定的扫描率（可高到 8192 赫兹）读取指定扫描数的电压。首先采集的数据是放在 LabJack 的 4096 个内存缓冲器中。然后才被传送到计算机上。

如果发光二极管被启用，那么它在等待出发期间会以 4 赫兹的频率闪烁，在采

集过程中会暗掉，在数据传送到计算机的过程中会快速地闪烁，结束后再变暗。

该函数的执行时间取决于指令本身，可以用下面的公式来估计。实际采样的样本数是 2 的次方，从 64 到 4096。 它大于或等于 numScans*numChannels。 在下面该值被表示成 numSamplesActual。

正常模式：运行时间 =

30+(1000*numSamplesActual/sampleRate)+(2.5*numSamplesActual)

快速模式：运行时间 =

30+(1000*numSamplesActual/sampleRate)+(0.4*numSamplesActual)

声明：

```
long AIBurst ( long *idnum,
               long  demo,
               long  stateIOin,
               long  updateIO,
               long  ledOn,
               long  numChannels,
               long  *channels,
               long  *gains,
               float *scanRate,
               long  disableCal,
               long  triggerIO,
               long  triggerState,
               long  numScans,
               long  timeout,
               float (*voltages)[4],
               long  *stateIOout,
               long  *overVoltage,
               long  transferMode )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。

• *stateIOin - 要输出的IO0-IO3的状态。如果IO是被设置成输入口，输出状态位对其没有作用。可以用其他的函数来把它们设置成输出方向。

• updateIO - 大于0，则输出状态值，否则只进行读操作。

• ledOn - 大于0时，LabJack的发光二极管被点亮。

• numChannels - 要读取的模拟通道数(1,2, 或 4)。

• *channels - 这是一个数组的指针，该数组至少要有numChannels 个单元。 单端输入用0-7，差动输入用 8-11。

• *gains - 这是一个数组的指针，该数组至少要有numChannels 个单元。个单元值是响应通道的增益指令。0=〉1, 1=〉2, …, 7=〉20。 只在差动通道采集时有效。

• *scanRate - 每秒的扫描数。一个扫描意味着对所有指定通道的一次采集。 采样频率（等于 scanRate * numChannels) 必须在 400 和 8192 之间。

• disableCal - 大于0时，返回的电压是没有用校正系数进行校正计算的值。（叫做原始值）。

• triggerIO - 出发信号的IO口号。(0=不用触发，1=IO0，2=IO1)。

• triggerState - 大于0时，采集将在所选的IO口变高电平时开始。

• numScans - 指定多少个扫描将被返回。最小值为1。numSamples 最大值为4096，其中 numSamples = numScans * numChannels。

• timeout - 超时时间(单位为秒)。该函数执行时间超过规定的超时时间时，即使还没有采到一个扫描，也会立即退出，同时返回超时错误码。

• *voltages - 这是一个指向一个 4096 乘 4 数组的指针。该数组用于放置采集的电压值。 将该数组元素置零输出。

• transferMode - 0=自动，1=正常，2=快速。如果是自动，当 timeout >= 4 或 numScans/scanRate >=4 时，实际使用模式是正常，否则是快速。

输出:

• *idnum - 返回设备号或-1（如果没有找到LabJack的话）

• *scanRate - 返回实际的扫描频率。由于时钟的分辨率的原因，实际的扫描率不会完全和希望的扫描率相同。

• *voltages - 这是一个数组的指针，返回的numChannels个电压值在该数组中。

• *stateIOout - 这是一个指向一个大小为4096的数组的指针。读取的IO状态值存在该数组中。没有用到单元的值是9999.0。

• *overVoltage - 如果大于0， 则说明在被选择的通道上信号超界。

ActiveX 函数的不同点:

在 ActiveX 函数中，“channels” 和 “gains” 数组被换成“channelsPacked” 和 “gainsPacked”。 OCX 中有一个叫 “FourPack” 的函数，该函数把4个单元的数组元素（element[ ]）转换成一个数值（PackedValue）。其公式为：

PackedValue = element[0] + (element[1] * 2^8) + (element[2] * 2^16) + (element[3] * 2^24)。

参数“demo”,“ledOn”,“disableCal”,“transferMode”,“updateIO”和“stateIOin”被“optionBits”代替。调用 OCX 函数 “BuildOptionBits” (4.38) 来获得这个参数。

“voltages”和 “stateIOout”是字符串格式。浮点数据用 13 个字符来表示(XXXX.XXXXXXXX)。整数用十个字符来表示 (XXXXXXXXXX)。0 被用于填充那些没有用到的字符。“voltages” 字符串的字节总长度是 13*numSamples。“stateIOout” 字符串的总字节是 string is 10*numScans。 注意：为了避免内存泄漏，这些字符串在每次调用 AIBustX 后要清空。

声明 (ActiveX):

```
long AIBurstX ( long FAR* idnum,
                long numChannels,
                long channelsPacked,
                long gainsPacked,
                float FAR* scanRate,
                long triggerIO,
                long triggerState,
                long numScans,
                long timeout,
                BSTR FAR* voltages,
```

```
BSTR FAR* stateIOout,
long FAR* overVoltage,
long optionBits)
```

4.8 AIStreamStart

该指令开始了硬件定时的连续数据采集。数据先是存放在 LabJack 的缓冲区内，然后同时传送到计算机的内存中。调用该函数后必须不断调用 AIStreamRead，最后调用 AIStreamClear。 注意在连续采集的过程中，LabJack 是不可能作任何其它工作的。如果调用除了 AIStreamRead 外的函数，连续读操作会被终止。

其执行时间为 30 毫秒或更少（在 Windows 下其典型值为 24 毫秒）。

如果启用发光二极管，它会在采集过程中每采 40 个样本变化一次亮暗状态，在操作停止时变亮。

声明：

```
long AIStreamStart ( long *idnum,
                     long demo,
                     long stateIOin,
                     long updateIO,
                     long ledOn,
                     long numChannels,
                     long *channels,
                     long *gains,
                     float *scanRate,
                     long disableCal,
                     long reserved1,
                     long readCount )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **stateIOIn** - 要输出的IO0-IO3的状态。

• updateIO - 大于0，则输出状态值，否则只进行读操作。

• ledOn - 大于0时，LabJack的发光二极管被点亮。

• numChannels - 要读取的模拟通道数(1,2, 或 4)。

• *channels - 这是一个数组的指针，该数组至少要有numChannels 个单元。 单端输入用0-7，差动输入用 8-11。

• *gains - 这是一个数组的指针，该数组至少要有numChannels 个单元。各单元值是相应通道的增益指令。0=〉1, 1=〉2, …, 7=〉20。 只在差动通道采集时有效。

• *scanRate - 每秒的扫描数。一个扫描意味着对所有指定通道的一次采集。 采样频率（等于 scanRate * numChannels) 必须在 200 和 1200 之间。

• disableCal - 大于0时，返回的电压是没有用校正系数进行校正计算的值。（叫做原始值）。

• reserved1 - 为今后而预留的参数，送0给它。

• readCount - 若大于0，函数不返回第2，3，4模拟通道的值，取而代之的是计数器的计数值。在原第2通道值处放置计数器的第0到11位，第12到23位放在原第3通道值处,第24道31位放在第4通道上。该功能在LabJack内在程序版本1.03 中被加入的。该参数此以前的版本中无效。

输出:

• *idnum - 返回设备号或-1（如果没有找到LabJack的话）

• *scanRate - 返回实际的扫描频率。由于时钟的分辨率的原因，实际的扫描率不会完全和希望的扫描率相同。

ActiveX 函数的不同点:

在 ActiveX 函数中，“channels” 和 “gains” 数组被换成“channelsPacked” 和 “gainsPacked”。 OCX 中有一个叫 “FourPack” 的函数，该函数把4个单元的数组元素（element[ ]）转换成一个数值（PackedValue）。其公式为：

PackedValue = element[0] + (element[1] * 2^8) + (element[2] * 2^16) + (element[3] * 2^24)。

参数“demo”,“ledOn”,“disableCal”,“transferMode”,“updateIO”和“stateIOin”被“optionBits”代替。调用 OCX 函数 “BuildOptionBits” (4.38) 来获得这个参数。

声明(ActiveX):

```
long AIStreamStartX (  long FAR* idnum,
                       long numChannels,
                       long channelsPacked,
                       long gainsPacked,
                       float FAR* scanRate,
                       long optionBits,
                       long readCount)
```

4.9 AIStreamRead

等待指定的扫描数完成后读取数据。之前要调用 AIStreamStart，结束连续读操作时要调用 AIStreamClear。注意在连续采集的过程中，LabJack 是不可能作任何其它工作的。如果调用除了 AIStreamRead 外的函数，连续读操作会被终止。

声明：

```
long AIStreamRead ( long localID,
                    long numScans,
                    long timeout,
                    float (*voltages)[4],
                    long *stateIOout,
                    long *reserved,
                    long *ljScanBacklog,
                    long *overVoltage )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***localID** - 把从AIStreamStart得到的LocalID作为该输入。
- **numScans** - 函数会等待该指定的扫描完成后才读取数据。其值最小为1，最大的 numSamples 为 4096，其中 numSamples = numScans * numChannels。 在内部，该函数从LabJack中一次获取64个样本，所以建议 numSamples 最少是 64。

• **timeout** - 超时时间(单位为秒)。

• ***voltages** - 这是一个指向一个 4096 乘 4 数组的指针。该数组用于放置采集的电压值。 将该数组元素置零输出。

• ***stateIOout** - 这是一个指向一个大小为4096的数组的指针。该数组用于放置IO 的状态值。 将该数组元素置零输出。

输出：

• ***voltages** - 这是一个指向一个 4096 乘 4 数组的指针。读取的电压存在该数组中。没有用到单元的值是9999.0。

• ***stateIOout** - 这是一个指向一个大小为4096的数组的指针。读取的IO状态值存在该数组中。没有用到单元的值是9999.0。

• **reserved** - 为今后而预留的参数，送0给它。

• ***ljScanBacklog** - 返回留在LabJack内存缓冲区内尚未读取的扫描数。其最大值为 4096/numChannels。

• ***overVoltage** - 如果大于0， 则说明在被选择的通道上信号超界。

ActiveX 函数的不同点:

“voltages” 和 “stateIOout” 是字符串格式。浮点数据用13个字符来表示(XXXX.XXXXXXXX)。整数用十个字符来表示 (XXXXXXXXXX)。0 被用于填充那些没有用到的字符。“voltages” 字符串的字节总长度是 13*numSamples。“stateIOout” 字符串的总字节是string is 10*numScans。 注意：为了避免内存泄漏，这些字符串在每次调用AIStreamReadX后要清空。

声明 (ActiveX):

```
long AIStreamReadX ( long localID,
                     long numScans,
                     long timeout,
                     BSTR FAR* voltages,
                     BSTR FAR* stateIOout,
                     long FAR* ljScanBacklog,
                     long FAR* overVoltage)
```

4.10 AIStreamClear

该函数停止连续读的操作。在连续读结束后必须要调用该函数。典型的调用次序为：AIStreamStart, AIStreamRead, AIStreamRead, AIStreamRead, …, AIStreamClear。

声明：

```
long AIStreamClear (  long localID )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）

4.11 AOUpdate

输出模拟电压。它还控制和读取所有 20 个数字口的状态的计数器。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

如果要输出的值小于0，DLL将输出上次的输出值。这样就可以只改变一个通道的电压值。注意当DLL刚刚载入时，它并不知道原先的值是什么，它只能认为它们都是0伏。同样当LabJack不是被DLL复位的情况下，DLL记忆的原先的电压值可能不是0，而实际的值却已经为0。

声明：

```
long AOUpdate ( long *idnum,
                long demo,
                long trisD,
                long trisIO,
                long *stateD,
                long *stateIO,
                long updateDigital,
                long resetCounter,
                unsigned long *count,
                float analogOut0,
                float analogOut1)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **trisD** - D0-D15的输入输出方向。 0=输入, 1=输出。
- **trisIO** - IO0-IO3 的输入输出方向。 0=输入, 1=输出。
- ***stateD** - D0-D15 的输出状态。
- ***stateIO** - IO0-IO3 的输出状态。
- **updateDigital** - 大于0，则输出状态值，否则只进行读操作。
- **resetCounter** - 大于0，则计数器在被读取后复位为0。
- **analogOut0** - 要输出到AO0的0.0到5.0伏的电压。
- **analogOut1** - 要输出到AO1的0.0到5.0伏的电压。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）
- ***stateD** - D0-D15的状态。
- ***stateIO** - IO0-IO3的状态。
- ***count** – 当前32位计数器的计数值，该指示在复位前读取的。

ActiveX 函数的不同点:

计数值用的不再是长整型（unsigned long），而是双精度（Double）。

声明 (ActiveX):

```
long AOUpdateX ( long FAR* idnum,
                 long demo,
                 long trisD,
                 long trisIO,
                 long FAR* stateD,
                 long FAR* stateIO,
                 long updateDigital,
                 long resetCounter,
```

```
double FAR* count,
float analogOut0,
float analogOut1)
```

4.12 AsynchConfig

在LabJack内在程序V1.1 或更高的版本中才有效。 该函数对 asynch 寄存器进行写操作并设置D口的方向。

其执行时间为 60 毫秒或更少（在 Windows 下其典型值为 48 毫秒）。

实际1位的时间大约是 1.833 + full-delay (us)

实际1/2位的时间大约是 1.0 plus a half-delay (us)

full/half-delay = 0.833 + 0.833C + 0.667BC + 0.5ABC (us)

常用的波特率	full-delay的A，B，C	half-delay的A，B，C
1	55，153，232	114，255，34
10	63,111,28 ;	34,123,23
100	51,191,2	33,97,3
300	71,23,4	84,39,1
600	183,3,6	236,7,1
1000	33,29,2	123,8,1
1200	23,17,4	14,54,1
2400	21,37,1	44,3,3
4800	10,18,2	1,87,1
7200	134,2,1	6,9,2
9600	200,1,1	48,2,1
10000	63,3,1	46,2,1
19200	96,1,1	22,2,1
38400	3,5,2	9,2,1
57600	3,3,2	11,1,1
100000	3,3,1	1,2,1

115200	9,1,1	2,1,1 或 1,1,1

当使用大于38.4kbps是，要考虑下面的因素：

- 在读第一个字节时，其起始位是在TX的停止位开始后11.5 秒时先被检验的。
- 在读以后的字节时，其起始位是在前8个比特被读后 full-delay + 11 微秒时先被检验的，这大约发生在第8位的中点处。

STB可以被其用来帮助调试异步读过程：

- STB 在上一个TX停止位开始后大约6微秒时，或在前第8位读后的full-delay + 6 微秒时被置为高电平。
- STB在RX的起始位被检测到后大约0-2 微秒时被清零。
- STB 在half-delay后被置1。
- STB在full-delay后被清零。
- 1 微秒后读 Bit 0。
- Bit 0 读后1微秒时STB 被置1。
- STB在full-delay后被清零。
- 1 微秒后读 Bit 1。
- Bit 1 读后1微秒时STB 被置1。
- STB在full-delay后被清零。
- 这过程持续到读完所有8位和停止位为止，然后STB保持低电平。

声明：

```
long AsynchConfig( long *idnum,
                   long demo,
                   long timeoutMult,
                   long configA,
                   long configB,
                   long configTE,
                   long fullA,
                   long fullB,
                   long fullC,
                   long halfA,
```

long halfB,
long halfC)

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **timeoutMult** - 若被启用，读超时时间大约是100 毫秒乘该值（0-255）。
- **configA** - 若大于零，D8 输出高电平，D9 设置为输入口。
- **configB** - 若大于零，D10 输出高电平，D11 设置为输入口。
- **configTE** - 若大于零，D12 输出低电平。
- **fullA/B/C** - full-delay 的时间常数 (1-255)。
- **halfA/B/C** - half-delay的时间常数 (1-255)。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）

4.13 Asynch

在LabJack 内在程序 V1.1 或更高的版本中才有效。 该函数在两对 D 线(8,n,1)中的其中一对上先读后写半双工异步数据。调用 AsynchConfig 来设置波特率。它和 RS232 相似，但逻辑不同 RS232。它的逻辑是正常的 CMOS/TTL 逻辑，即：0=低=地电平，1=高=+5 伏，传输线的闲置状态为高电平。 要接到 RS232 设备上可能需要转换芯片，如 MAX233。

读写 4 个字节的执行时间大约为 20 毫秒，每增加 4 个字节则增加 20 毫秒。低波特率下需要的执行时间更长。

A 口（PortA） => TX 是 D8 and RX 是 D9

B 口（PortB） => TX 是 D10 and RX 是 D11

传送启用线是 D12

可读写多达 18 个字节。在读写多与 4 个字节以上的情况下，该函数用 Up to 18 bytes can be WriteMem/ReadMem 来读取 LabJack 的数据缓冲。

声明：

```
long Asynch( long *idnum,
             long demo,
             long portB,
             long enableTE,
             long enableTO,
             long enableDel,
             long baudrate,
             long numWrite,
             long numRead,
             long *data)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **portB** - 若大于零，则用B口（port B）而不用A口（port A）。
- **enableTE** - 若大于零，D12（传送启用线）传送时置高，接收时清零。
- **enableTO** - 若大于零，在接收每个字节时启用超时控制。
- **enableDel** - 若大于零，在传送每个字节之间加入1位的延时。
- **baudrate** - 这是由AsynchConfig设置的波特率。 Asynch 函数需要知道它才能确定传输需要多少时间。
- **numWrite** - 要写出的字节数 (0-18)。
- **numRead** - 要读入的字节数(0-18)。
- ***data** - 串行数据缓冲。是一个有18元素的数组。没有用到的元素要置零。

输出:

• ***idnum** – 返回设备号或-1（如果没有找到LabJack的话）

• ***data** – 串行数据缓冲。返回读到的串行数据。没有用到的元素值为9999。

ActiveX 函数的不同点:

最大读写的字节为 6 个(numWrite 和 numRead 应是 0-6)。 *data 数组被 6 个单独的数据指针代替。

声明 (ActiveX):

```
long AsynchX( long FAR* idnum,
              long demo,
              long portB,
              long enableTE,
              long enableTO,
              long enableDel,
              long baudrate,
              long numWrite,
              long numRead,
              long FAR* data0,
              long FAR* data1,
              long FAR* data2,
              long FAR* data3,
              long FAR* data4,
              long FAR* data5)
```

4.14 BitsToVolts

把一个 12 位二进制值转换成 LabJack 的电压值。这里没有任何硬件操作。

Volts=((2*Bits*Vmax/4096)-Vmax)/Gain

其中 Vmax=10（单端输入时），20（差动输入时）

声明：

```
long BitsToVolts ( long chnum,
                   long chgain,
                   long bits,
                   float *volts )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- **chnum** - 通道号。单端输入用0-7，差动输入用 8-11。
- **chgain** - 增益。0=〉1, 1=〉2, …, 7=〉20。
- **bits** - 二进制值（0-4095）

输出：

- ***volts** - 电压值

4.15 VoltsToBits

把一个电压值转换成 12 位二进制值。这里没有任何硬件操作。

Bits=(4096*((Volts*Gain)+Vmax))/(2*Vmax)

其中 Vmax=10（单端输入时），20（差动输入时）

声明：

```
long VoltsToBits ( long chnum,
                   long chgain,
                   float volts,
                   long *bits )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- **chnum** - 通道号。单端输入用0-7，差动输入用 8-11。
- **chgain** - 增益。0=〉1, 1=〉2, …, 7=〉20。
- ***volts** - 电压值

输出：

- **bits** - 二进制值（0-4095）

4.16 Counter

控制和读取计数器。当看门狗定时器被启用时，该计数器被禁用。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

声明：

```
long Counter ( long *idnum,
               long demo,
               long *stateD,
               long *stateIO,
               long resetCounter,
               long enableSTB,
               unsigned long *count )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **resetCounter** - 若大于0，计数器在读取后被清零。
- **enableSTB** - 若大于0,STB 被启用。用于测试和校正。(该输入对LabJack内在程序V1.02 或更早的版本没有效果。STB总是被启用的)

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）
- ***stateD** - D0-D15的状态
- ***stateIO** - IO0-IO3的状态
- ***count** - 当前32位计数器的计数值，该指示在复位前读取的。

ActiveX 函数的不同点:

计数值用的不再是长整型（unsigned long），而是双精度（Double）。

声明 (ActiveX):

```
long CounterX(long FAR* idnum,
              long demo,
              long FAR* stateD,
              long FAR* stateIO,
              long resetCounter,
              long enableSTB,
```

double FAR* count)

4.17 DigitalIO

读写所有的 20 个 I/O。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

声明：

```
long DigitalIO (long *idnum,
                long demo,
                long *trisD,
                long trisIO,
                long *stateD,
                long *stateIO,
                long updateDigital,
                long *outputD )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- ***trisD** - D0-D15的输入输出方向。 0=输入, 1=输出
- **trisIO** - IO0-IO3 的输入输出方向。 0=输入, 1=输出。
- ***stateD** - D0-D15 的输出状态。
- ***stateIO** - IO0-IO3 的输出状态。
- **updateDigital** - 大于0，则输出状态值，否则只进行读操作。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）
- ***trisD** - 返回 D0-D15 的方向
- ***stateD** - D0-D15的状态。
- ***stateIO** - IO0-IO3的状态。

- ***outputD** - 返回 D0-D15 的输出值。

4.18 GetDriverVersion

返回 LabJack 驱动程序的版本号。没有硬件操作。

声明：：

```
float GetDriverVersion ( void )
```

参数说明:

返回驱动程序（DLL）的版本号。

ActiveX 函数的不同点:

用参数来获得 DLL 和 OCX 的版本号。

声明： (ActiveX):

```
void GetDriverVersionX ( float FAR* dllVersion,
                         float FAR* ocxVersion)
```

4.19 GetErrorString

把另一个函数返回的错误码转换成错误信息字符串。

声明：：

```
void GetErrorString ( long errorcode,
                      char *errorString )
```

参数说明:

没有返回值。

输入：

- **errorcode** - LabJack 的错误码。
- ***errorString** - 一个指向有50个字符元素的数组。

输出：

- ***errorString** - 指向错误信息字符串的指针。没有用的单元被填0。

4.20 GetFirmwareVersion

获取 LabJack 内在程序的版本号

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

声明：：

```
float GetFirmwareVersion ( long *idnum )
```

参数说明:

返回版本号或 0（表示错误）。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）。若有错误，返回值为一般的错误码加上512。

4.21 GetWinVersion

用 Windows API 来读操作系统的版本号。

声明：：

```
long GetWinVersion ( unsigned long *majorVersion,
                     unsigned long *minorVersion,
                     unsigned long *buildNumber,
                     unsigned long *platformID,
                     unsigned long *servicePackMajor,
                     unsigned long *servicePackMinor )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输出：

	Platform	Major	Minor	Build
Windows 3.1	0	-	-	-

Windows 95	1	4	0	950
Windows 95 OSR2	1	4	0	1111
Windows 98	1	4	10	1998
Windows 98SE	1	4	10	2222
Windows Me	1	4	90	3000
Windows NT 3.51	2	3	51	-
Windows NT4.0	2	4	0	1381
Windows2000	2	5	0	2195
WindowsXP	2	5	1	-

ActiveX 函数的不同点:

计数值用的不再是长整型（unsigned long），而是双精度（Double）。

4.22 ListAll

查找 USB 上所有的 LabJacks，并返回它们的系列号和设备号。

声明：:

```
long ListAll ( long *productIDList,
               long *serialnumList,
               long *localIDList,
               long *powerList,
               long (*calMatrix)[20],
               long *numberFound,
               long *reserved1,
               long *reserved2 )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入：

- ***productIDList** - 指向127个单元的数组。数组先置0。
- ***serialnumList** - 指向127个单元的数组。数组先置0。

• *localIDList - 指向127个单元的数组。数组先置0。

• *powerList - 指向127个单元的数组。数组先置0。

• *calMatrix - 指向127 x 20 个单元的数组。数组先置0。

输出：

• *serialnumList - 指向127个单元的数组，存系列号，没用的单元置9999.0。

• *localIDList - 指向127个单元的数组，存设备号，没用的单元置9999.0。

• *numberFound - 找到的LabJackde个数。

ActiveX 函数的不同点：

数组都由字符串来表示，每个数用 10 个字符（XXXXXXXXXX）。字符串在需要处补零。

声明： (ActiveX)：

```
long ListAllX (  BSTR FAR* productIDList,
                 BSTR FAR* serialnumList,
                 BSTR FAR* localIDList,
                 BSTR FAR* powerList,
                 BSTR FAR* calMatrix,
                 long FAR* numberFound,
                 long FAR* reserved1,
                 long FAR* reserved2)
```

4.23 LocalID

用来改变设备号。这改变要在 LabJack 被重新访问查寻或复位后才有效（你可以拔出 USB 线后再接入或调用 ReEnum 或 Reset 函数）。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

声明：：

```
long LocalID ( long *idnum,
               long localID )
```

参数说明：

返回值：LabJack 的错误码，无错时为 0。

输入:

- *idnum - 设备号，系列号或-1（所能找到的第一个设备）
- localID - 新的设备号。

输出：

- *idnum - 设备号，系列号或-1（所能找到的第一个设备）

4.24 NoThread

在 Windows 98/SE 上使用 TestPoint 和 LabJack DLL 接口时要用到该函数（可详见 ljackuw.h)。调用该函数来启用/禁用开辟其他函数的线程。一般情况下开辟线程应该被启用，但是在 TestPoint 上调用 LabJack 的函数时，它必须被禁用。 另外在 Visual C Debugger 上运行对时间要求和高的应用程序时禁用线程的开辟可能有好处。Visual C Debugger 开辟线程缓慢是已知的问题。

其执行时间为 80 毫秒或更少。

如果线程被禁用，在 AIBurst 和 AIStreamRead 中的超时功能也会被禁用。

声明:

```
long NoThread ( long *idnum,
                long noThread )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- *idnum - 设备号，系列号或-1（所能找到的第一个设备）
- noThread - 若大于零，线程不被使用。

输出:

- *idnum - 返回设备号或-1（如果没有找到LabJack的话）

4.25 PulseOut

在 LabJack 内在程序 V1.1 或更高的版本中才有效。它在任何一个或所有 DO -

D7 上产生脉冲。 所选择的 D 口必须事先设置成输出口（可用 DigitalIO 或 AOUpdate)。 所有口上的脉冲会是一样的。

其执行时间为 20 毫秒加上脉冲的输出时间。

该函数指定每个脉冲的前半周期和后半周期的时间。每个时间是由值 B 和 C 来表示的。

前半周期时间（微秒）= ~17 + 0.83*C + 20.17*B*C,

后半周期时间（微秒）= ~12 + 0.83*C + 20.17*B*C,

可用下面的简化公式估计。

时间（微秒） = 20*B*C

C 越小,其估计精度越高。B 和 C 必须在 1 到 255 之间。因此每半周期在 38/33 到 1.3 秒之间。

若 LabJack 的看门狗功能被启用，请确认它的超时值大于输出脉冲的时间。

该函数的超时时间被设为：5000+numPulses*((B1*C1*0.02)+(B2*C2*0.02))

声明：：

```
long PulseOut(long *idnum,
              long demo,
              long lowFirst,
              long bitSelect,
              long numPulses,
              long timeB1,
              long timeC1,
              long timeB2,
              long timeC2)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为 0，大于 0 时为演示模式。演示模式允许用户在没有 LabJack 的情况下调用该函数。
- **lowFirst** - 若大于 0，则脉冲是先低后高，否则先高后低。

• bitSelect - D0-D7 相应位被置 1 时，改口就有脉冲输出。范围是 0-255。

• numPulses - 输出的脉冲数 (1-32767)。

• timeB1 - 前半周期的 B 值 (1-255)。

• timeC1 - 前半周期的 C 值 (1-255)。

• timeB2 - 后半周期的 B 值 (1-255)。

• timeC2 - 后半周期的 C 值 (1-255)。

输出:

• *idnum - 返回设备号或-1（如果没有找到 LabJack 的话）

4.26 PulseOutStart

在 LabJack 内在程序 V1.1 或更高的版本中才有效。PulseOutStart 和 PulseOutFinish 是 PulseOut 的另一种用法。 PulseOutStart 开始脉冲输出后不等待其结束就返回。而 PulseOutFinish 则等待 LabJack 结束脉冲输出后的响应。

其执行时间为 10 毫秒。

如果在调用 OulseOutStart 后调用除 PulseOutFinish 外的函数，脉冲输出会被中断，LabJack 将执行该新命令。 在脉冲输出结束前不断调用 PulseOutStart 可以得到相当好的连续脉冲输出。

注意 LabJack U12 启动后会进行检验，如果在 LabJack 复位或刚上电后的第一个指令就是 PulseOutStart，那么 LabJack 就不会送响应给 PulseOutFinish。实际上这是不会发生的，因为在调用 PulseOutStart 前必须调用其它函数把相应的 D 口设成输出口。

另外请注意必须在 LabJack 结束输出脉冲之前调用 PulseOutFinish 否则 LabJack 的响应会丢失，从而会得到一个超时错误。

声明：：

```
long PulseOutStart( long *idnum,
                    long demo,
                    long lowFirst,
                    long bitSelect,
                    long numPulses,
```

```
                    long timeB1,
                    long timeC1,
                    long timeB2,
                    long timeC2)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **lowFirst** - 若大于0，则脉冲是先低后高，否则县高后低。
- **bitSelect** - D0-D7 相应位被置1时，改口就有脉冲输出。范围是0-255。
- **numPulses** - 输出的脉冲数 (1-32767)。
- **timeB1** - 前半周期的B值 (1-255)。
- **timeC1** - 前半周期的C值 (1-255)。
- **timeB2** - 后半周期的B值 (1-255)。
- **timeC2** - 后半周期的C值 (1-255)。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）

4.27 PulseOutFinish

在 LabJack 内在程序 V1.1 或更高的版本中才有效。详情参见 PulseOutStart。

声明：：

```
long PulseOutFinish( long *idnum,
                     long demo,
                     long timeoutMS)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

• ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）

• **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。

• **timeoutMS** - 超时值（毫秒）。指定了该函数等待响应的时间限制。

输出:

• ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）

4.28 PulseOutCalc

在 LabJack 内在程序 V1.1 或更高的版本中才有效。该函数用来计算 PulseOut 和 PulseOutStart 中的周期时间。

声明：：

```
long PulseOutCalc( float *frequency,
                   long *timeB,
                   long *timeC)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入：

• ***frequency** - 频率的期望值（赫兹）。

输出：

• ***frequency** - 实际找到的最佳频率（赫兹）

• ***timeB** - 后半周期的B值。

• ***timeC** - 后半周期的C值。

4.29 ReEnum

使 LabJack 重新被访问查寻。设备号和校正常数之后会被更新。

声明：：

```
long ReEnum ( long *idnum )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）

4.30 Reset (or ResetLJ)

使 LabJack 2 秒后被复位。复位后 LabJack 会被重新访问查寻。 Reset 和 ResetLJ are 完全相同的。

声明：:

long Reset (long *idnum)

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）

4.31 SHT1X

该函数读取 SHT1X 传感器的温度/湿度数据。数据传输率大约是 2kbps（V1.1 或更高的版本使用硬件来进行通讯)。如果版本是低于 V1.1 或 softComm 为 TRUE（真)，那么数据传输率为 20bps。

DATA = IO0 （IO0 是数据口）

SCK = IO1 （IO1 是通讯控制口）

EI-1050 有另外一根启用控制线,它使得一根 DATA 线和一根 SCK 线可以同时连

接多个传感器。该函数不能控制着跟启用控制线。

该函数自动将 IO0 设置成输入口，IO1 设置成输出口。

注意该函数实际上是对 IO0，IO1 的方向和状态进行操作。要操作任何一根 IO 口，LabJack 需要对所有 4 个口进行操作。 但 DLL 会记住所有口的方向和状态，所以该函数可以在不改变 IO2，IO3 的情况下对 IO0 和 IO1 进行操作。是当 DLL 刚刚载入时，它并不知道方向和状态，所以只能认为所有口的方向是输入，在所有口上的电平为低。

声明:

```
long SHT1X( long *idnum,
            long demo,
            long softComm,
            long mode,
            long statusReg,
            float *tempC,
            float *tempF,
            float *rh)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **softComm** - 若大于0，使用软件通讯，否则只有在内在程序版本低于V1.1时才使用软件通讯。
- **mode** - 0=温度和湿度，1=温度，2=湿度。 如果 mode = 2，必须用 *tempC 来传进当前的温度值，温度只是用来校正湿度用的。
- **statusReg** - SHT1X寄存器中的当前值。除非你用了一些高级的函数写过这个寄存器，否则它的值是0。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）

• ***tempC** - 返回摄氏温度值。 如果mode = 2， 要传进当前的温度值来校正湿度。

• ***tempF** - 返回华氏温度值。

• ***rh** - 返回湿度的百分比。

4.32 SHTComm

这是个低级函数，用于给 SHT1X 传感器收发多大 4 个字节的数据。数据传输率大约是 2kbps（V1.1 或更高的版本使用硬件来进行通讯)。如果版本是低于 V1.1 或 softComm 为 TRUE（真)，那么数据传输率为 20bps。

DATA = IO0 （IO0 是数据口）

SCK = IO1 （IO1 是通讯控制口）

EI-1050 有另外一根启用控制线,它使得一根 DATA 线和一根 SCK 线可以同时连接多个传感器。该函数不能控制着跟启用控制线。

该函数自动将 IO0 设置成输入口，IO1 设置成输出口。

注意该函数实际上是对 IO0，IO1 的方向和状态进行操作。要操作任何一根 IO 口，LabJack 需要对所有 4 个口进行操作。 但 DLL 会记住所有口的方向和状态，所以该函数可以在不改变 IO2，IO3 的情况下对 IO0 和 IO1 进行操作。是当 DLL 刚刚载入时，它并不知道方向和状态，所以只能认为所有口的方向是输入，在所有口上的电平为低。

声明：：

```
long SHTComm( long *idnum,
              long softComm,
              long waitMeas,
              long serialReset,
              long dataRate,
              long numWrite,
              long numRead,
              unsigned char *datatx,
              unsigned char *datarx)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

• ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）

• **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。

• **softComm** - 若大于0，使用软件通讯，否则只有在内在程序版本低于V1.1时才使用软件通讯。

• **waitMeas** - 若大于0，它是一个温度或湿度（T or RH）测量的请求。

• **serialReset** - 若大于0，在收发前会先发一个串行复位。

• **dataRate** - 0=没有额外的延时（默认值），1=中度延时，2=最大延时。

• **numWrite** - 要写出的字节数(0-4)。

• **numRead** - 要读进的字节数(0-4)。

• ***datatx** - 要发送的 0-4 字节的数组。数组大小至少要等与 numWrite

输出:

• ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）。

• ***datarx** - 返回0-4字节，由 numRead 确定。

ActiveX 函数的不同点:

收发的数组被 4 个单独的参数代替。每个参数存一个字节。

声明: (ActiveX):

```
long SHTCommX( long FAR* idnum,
               long softComm,
               long waitMeas,
               long serialReset,
               long dataRate,
               long numWrite,
               long numRead,
               long FAR* data0,
               long FAR* data1,
               long FAR* data2,
               long FAR* data3)
```

4.33 SHTCRC

检查 SHT1X 通讯的 CRC（纠错字）。Datarx 的最后一个字节是 CRC。如果 CRC 是好的返回 0，否则返回 SHT1X_CRC_ERROR_LJ。

声明:

```
long SHTCRC(long statusReg,
            long numWrite,
            long numRead,
            unsigned char *datatx,
            unsigned char *datarx)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入：

- **statusReg** - SHT1X寄存器中的当前值。
- **numWrite** - 送出的字节数(0-4)。
- **numRead** - 收到的字节数(0-4)。
- ***datatx** - 要发送的 0-4 字节的数组。数组大小至少要等与 numWrite
- ***datatx** - 送出的 0-4 字节的数组。
- ***datarx** - 收到的 0-4 字节的数组。

ActiveX 函数的不同点:

收发的数组被 4 个单独的参数代替。每个参数存一个字节。

声明: (ActiveX):

```
long SHTCRCX( long statusReg,
              long numWrite,
              long numRead,
              long datatx0,
              long datatx1,
              long datatx2,
              long datatx3,
              long datarx0,
              long datarx1,
              long datarx2,
              long datarx3)
```

4.34 Synch

在 LabJack 内在程序 V1.1 或更高的版本中才有效。该函数进行 SPI 通讯。在没有额外延时的情况下数据传输率是 160kbps，但是每位是可以加进 100 微秒或 1 毫秒的延时。

读写 4 个字节的执行时间大约为 20 毫秒，每增加 4 个字节则增加 20 毫秒。如果 configD 为真（TRUE）则要花另外 20 毫秒。如果开启延时，花时会更多。

D0-D7 在该函数中可以作为“片选”控制信号。任何一个数字口都可以作为“片选”信号。

MOSI 是 D13

MISO 是 D14

SCK 是 D15

若用 CB25（数字口接线板），那么那些在 SPI 连接线上的保护电阻可能要短接。

在 CS 变低前，SCK 的初始状态 会被该函数适当地设置（CPOL）。在 CS 变高前，SCK 的状态也会被该函数适当地设置。如果认为地处理“片选”信号，必须小心地把 SCK 的初始值设成 CPOL。

支持所有模式 (A, B, C, 和 D)

模式 A: CPHA=1, CPOL=1

模式 B: CPHA=1, CPOL=0

模式 C: CPHA=0, CPOL=1

模式 D: CPHA=0, CPOL=0

如果时钟相位 (CPHA) 是 1, 数据在变到 CPOL 沿有效。如果 CPHA 是 0, 数据在离开 CPOL 沿有效。 时钟极性 (CPOL) 确定了 SCK 的闲置状态。

可以读写多达 18 个字节。通讯是全工的（full duplex），所以读写一个字节是发生在同一时间的。如果读写 4 个字节以上，该函数用 WriteMem/ReadMem 来读取 LabJack 的数据缓冲。

该函数用 configD 来自动为以下信号配置默认状态和方向：MOSI (D13 输出), MISO (D14 输入), SCK (D15 输出 CPOL), CS (D0-D7 为!CS 输出高电平)。 该函

数调用 DigitalIO 来做这些工作。 和 EDigitalIn，EDigitalOut 相似，DLL 记住当前所有数字口的方向和输出状态，所以可以在不影响其它数字口的情况下对着 4 根 D 线进行配置。当 DLL 刚刚载入时，它并不知道方向和状态，所以只能认为所有口的方向是输入，在所有口上的电平为低。

声明:

```
long Synch( long *idnum,
            long demo,
            long mode,
            long msDelay,
            long husDelay,
            long controlCS,
            long csLine,
            long csState,
            long configD,
            long numWriteRead,
            long *data)
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **mode** - 指定SPI的操作模式(0-3)：0=A,1=B,2=C,3=D。
- **msDelay** - 若大于0，每位间会加入一个1毫秒的延时。
- **husDelay** - 若大于0，每位间会加入一个100微秒的延时。
- **controlCS** - 若大于0，D0-D7 自动被控制成“片选”信号（CS）。CS的方向和状态只在控制被启用的情况下被检测。
- **csLine** - 若被启用，D线用作CS。(0-7)
- **csState** - CS 线的工作状态。它一般为0，即使用正常的!CS， 若大于0，则用较少用的CS。
- **configD** - 若大于0，D13, D14, D15, !CS的方向和状态被配置。

• numWriteRead - 要读写的字节数 (1-18)

• *data - 串行数据缓冲。是一个有18个元素的数组。没有用到的元素要置零。

输出:

• *idnum - 返回设备号或-1（如果没有找到LabJack的话）

• *data - 串行数据缓冲。返回读到的串行数据。没有用到的元素值为9999。

ActiveX 函数的不同点:

可读写的最多字节数被限为 5 个(numWriteRead 应是 1-5)。 数组被 5 个单独字节的指针代替。

声明: (ActiveX):

```
long SynchX( long FAR* idnum,
             long demo,
             long mode,
             long msDelay,
             long husDelay,
             long controlCS,
             long csLine,
             long csState,
             long configD,
             long numWriteRead,
             long FAR* data0,
             long FAR* data1,
             long FAR* data2,
             long FAR* data3,
             long FAR* data4,
             long FAR* data5)
```

4.35 Watchdog

用来控制 LabJack 看门狗的功能。如果检测到 LabJack 和计算机的通讯有问题，被启用后看门狗的功能可以在指定的时间内改变数字的 I/O 状态。该信号可以用于重启计算机已达到长期无人监守的可靠性。 当看门狗的功能被启用时，32 位的计数器（CNT）就会被禁用。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

如果你让看门狗去复位 LabJack，并选择太小的超时时间，那么可能难以让LabJack 脱离复位状态。 要禁用他的话，把 IO0 短接到 STB 再复位 LabJack，然后除去短接线，再次复位 LabJack。

声明:

```
long Watchdog ( long *idnum,
                long demo,
                long active,
                long timeout,
                long reset,
                long activeD0,
                long activeD1,
                long activeD8,
                long stateD0,
                long stateD1,
                long stateD8 )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **active** - 启用看门狗的功能，32位的计数器（CNT）就会被禁用。
- **timeout** - 超时时间(1-715秒)。
- **reset** - 若大于零，超时后LabJack被复位。
- **activeDn** - 若大于0，经过超时延时后把第n位的D线置成stateDn状态。
- **stateDn** - 超时后第n个D线的状态。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）

4.36 ReadMem

从 LabJack 非易失性存储器的指定地址中读取 4 个字节。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

声明：：

```
long ReadMem ( long *idnum,
               long address,
               long *data3,
               long *data2,
               long *data1,
               long *data0 )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **address** - 数据的起始地址 (0-8188)

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）
- ***data3** - 第一个地址中的值。
- ***data2** - 第二个地址中的值。
- ***data1** - 第三个地址中的值。
- ***data0** - 第四个地址中的值。

4.37 WriteMem

在 LabJack 非易失性存储器的指定地址中写进 4 个字节。写后数据会被读回以确认。内存 0-511 是用来保存配置和校正数据的。内存 512-1023 是给用户用的。这样数据起始地址是 512-1020。内存 1024-8191 是用作数据缓冲的（在 Burst 和 Stream 模式时用）。

其执行时间为 20 毫秒或更少（在 Windows 下其典型值为 16 毫秒）。

声明：：

```
long WriteMem ( long *idnum,
                long unlocked,
                long address,
                long data3,
                long data2,
                long data1,
                long data0 )
```

参数说明:

返回值: LabJack 的错误码，无错时为 0。

输入:

- ***idnum** - 设备号，系列号或-1（所能找到的第一个设备）
- **unlocked** - 若大于零，地址0-511被打开来写数据。
- **address** - 写入数据的起始地址 (0-8188)
- **data3** - 第一个地址中的字节。
- **data2** - 第二个地址中的字节。
- **data1** - 第三个地址中的字节。
- **data0** - 第四个地址中的字节。

输出:

- ***idnum** - 返回设备号或-1（如果没有找到LabJack的话）

4.38 BuildOptionBits (ActiveX only)

该函数只在 OCX 中有。用来获得 AIBurst 和 AIStreamStart 使用的 optionBits 参数。

optionBits 是由下列位组成的，它常常只是被设置成 2。

- bit 0 => demo
- bit 1 => ledOn
- bit 2 => disableCal
- bits 3,4 => transferMode
- bit 5 => updateIO
- bit 6 => stateIOin(0)

- bit 7 => stateIOin(1)
- bit 8 => stateIOin(2)
- bit 9 => stateIOin(3)

声明：：

```
long BuildOptionBits (  long demo,
                        long ledOn,
                        long disableCal,
                        long transferMode,
                        long updateIO,
                        long stateIOin )
```

参数说明:

返回 optionBits

输入:

- **demo** - 正常操作时为0，大于0时为演示模式。演示模式允许用户在没有LabJack的情况下调用该函数。
- **ledOn** - 大于0时，LabJack的发光二极管被点亮。
- **disableCal** - 大于0时，返回的电压是没有用校正系数进行校正计算的值。（叫做原始值）。
- **transferMode** - 设为 0（自动）。
- **updateIO** - 大于0，则输出状态值，否则只进行读操作。
- ***stateIOin** - 要输出的IO0-IO3的状态。如果IO是被设置成输入口，输出状态位对其没有作用。

4.39 FourPack (ActiveX only)

该函数只在 OCX 中有。用于把 4 个元素的数组转换成一个整数。转换值计算如下：

valueA+(valueB * 2^8)+(valueC * 2^16+(valueD * 2^24)。其中 A，B，C，D 是数组的元素值。

声明:

```
long FourPack ( long valueA,
```

```
                long valueB,
                long valueC,
                long valueD)
```

参数说明:

返回转换后的值。

输入：

- valueA - 数组中第一单元的元素值。
- valueB - 数组中第二单元的元素值。
- valueC - 数组中第三单元的元素值。
- valueD - 数组中第四单元的元素值。

4.40 错误码解释

我们建议用 GetErrorString 来解释错误码。下列是简要的说明。

- 0 - No error（没有错误）
- 1 - Unknown error（不明错误原因）
- 2 - No LabJacks found（没找到LabJack硬件）
- 3 - LabJack n not found（第n个LabJack没有找到）
- 4 - Set USB buffer error（设置USB缓冲错误）
- 5 - Open handle error（打开句柄错误）
- 6 - Close handle error（关闭句柄错误）
- 7 - Invalid ID（无效的设备号）
- 8 - Invalid array size or value（无效的数组大小或值）
- 9 - Invalid power index（无效的次方数）
- 10 - FCDD size too big（FCDD太大）
- 11 - HVC size too big（HVC太大）
- 12 - Read error（读操作错误）
- 13 - Read timeout error（读超时错误）
- 14 - Write error（写错误）
- 15 - Turbo error（快速模式错误）
- 16 - Illegal channel index（无效的通道数）
- 17 - Illegal gain index（无效的增益数）
- 18 - Illegal AI command（无效的AI命令）
- 19 - Illegal AO command（无效的AO命令）
- 20 - Bits out of range（位超界）
- 21 - Illegal number of channels（无效的总通道数）

- 22 - Illegal scan rate（无效的扫描速率）
- 23 - Illegal number of samples（无效的样本总数）
- 24 - AI response error（AI响应错误）
- 25 - LabJack RAM checksum error（LabJack的RAM检查错误）
- 26 - AI sequence error（AI执行次序错误）
- 27 - Maximum number of streams（最大的连续数据数错误）
- 28 - AI stream start error（AI 连续读启动错误）
- 29 - PC buffer overflow（计算机缓冲溢出）
- 30 - LabJack buffer overflow（LabJack缓冲溢出）
- 31 - Stream read timeout（连续读超时错误）
- 32 - Illegal number of scans（无效的扫描数）
- 33 - No stream was found（没找到连续读的数据）
- 40 - Illegal input（无效的参数输入）
- 41 - Echo error（回复错误）
- 42 - Data echo error（数据回复错误）
- 43 - Response error（响应错误）
- 44 - Asynch read timeout error（Asynch读超时错误）
- 45 - Asynch read start bit error（Asynch读启始位错误）
- 46 - Asynch read framing error（Asynch读字节长度错误）
- 47 - Asynch DIO config error（Asynch DIO配置错误）
- 48 - Caps error（能力错误）
- 49 - Caps error（能力错误）
- 50 - Caps error（能力错误）
- 51 - HID number caps error（HID数字能力错误）
- 52 - HID get attributes warning（HID取特性警告）
- 57 - Wrong firmware version error（错误内在程序版本）
- 58 - DIO config error（数字口配置错误）
- 64 - Could not claim all LabJacks（不能得到所有的LabJack）
- 65 - Error releasing all LabJacks（释放所有LabJack时出错）
- 66 - Could not claim LabJack（不能得到LabJack）
- 67 - Error releasing LabJack（释放LabJack时出错）
- 68 - Claimed abandoned LabJack（不能获得已放弃的LabJack）
- 69 - Local ID -1 thread stopped（第一个找到的LabJack的线程已停止）
- 70 - Stop thread timeout（停止线程是超时）
- 71 - Thread termination failed（停止线程失败）
- 72 - Feature handle creation error（创建特性句柄错误）
- 73 - Create mutex error（创建mutex出错）
- 80 - Synchronous CS state or direction error（同步CS状态或方向出错）
- 81 - Synchronous SCK direction error（同步SCK方向错误）

- 82 - Synchronous MISO direction error（同步MISO方向错误）
- 83 - Synchronous MOSI direction error（同步MOSI方向错误）
- 89 - SHT1X CRC error（SHT1X CRC错误）
- 90 - SHT1X measurement ready error（SHT1X 测量预备信号）
- 91 - SHT1X ack error（SHT1X ACK错误）
- 92 - SHT1X serial reset error（SHT1X 串行复位错误）

如果第 8 位是 1，则该错误发生在 Stream 的线程中。第 10 位为 1 表示 Windows API 出错。

5 附录 A. 技术性能指标

参数	条件	最小值	典型值	最大值	单位
一般项					
USB 线长度				3	米
用户连接线长度	CE 兼容			3	米
电源电流（1）			20		毫安
运行温度		-40		85	℃
时钟误差	~25℃			±30	ppm
	0 - 70℃			±50	ppm
	-40 - 85℃			±100	ppm
+5 伏电源					
电压（Vs）（2）	自供电	4.5		5.25	伏
	总线供电	4.1		5.25	伏
输出电流（2）（3）	自供电	450		500	毫安
	总线供电	50		100	毫安
模拟输入（A0-A7）					
线形输入范围	AIx 对 地，单端	-10		10	伏
	AIx 对 地，差动	-10		20	伏
最大输入范围	AIx 对 地	-40		40	伏
输入电流（4）	Vin = +10 伏		70.1		微安
	Vin = 0 伏		-11.7		微安
	Vin = -10 伏		-93.5		微安
分辨率（无误码）	C/R 和 Stream		12		位
	Burst， 差动		12		位
	Burst， 单端		11		位
偏置	G = 1 到 20		±1*G		位
绝对精度	单端		±0.2		%满标
	差动		±1		%满标

噪声	C/R 和 Stream		±1		位
积分线性误差			±1		位
差分线性误差			±0.5		位
重复性			±1		位
CAL 精度	CAL = 2.5 伏		±0.05	±0.25	%
CAL 电流	流出			1	毫安
	流入	20	100		微安
触发延迟	Burst	25		50	微秒
触发脉冲宽度	Burst	40			微秒
模拟输出（AO0，AO1）					
最大电压（6）	无负载		Vs		伏
	1 毫安时		0.99Vs		伏
	5 毫安时		0.96Vs		伏
源阻抗			20		Ω
输出电流	每个 AO			30	毫安
IO					
低电平输入电压				0.8	伏
高电平输入电压		3		15	伏
输入泄漏电流（7）			±1		微安
输出短路电流（8）	输出高		3.3		毫安
输出电压（8）	无负载	Vs-0.4	Vs		伏
	1 毫安时		Vs-1.5		伏
D					
低电平输入电压（9）	D0 - D12			0.8	伏
	D13 - D15			1	伏
高电平输入电压（9）	D0 - D12	2		Vs+0.3	伏
	D13 - D15	4		Vs+0.3	伏
输入泄漏电流（7）			±1		微安
输出电流（9）	每个口			25	毫安

	D0-D15 之和			200	毫安
输出低电平电压				0.6	伏
输出高电平电压		Vs-0.7			伏
计数器（CNT）					
低电平电压（10）		地		1	伏
高电平电压（10）		4		15	伏
斯密特触发起滞回			20-100		毫伏
输入泄漏电流（7）			±1		微安
最小高电平时间				500	纳秒
最小低电平时间				500	纳秒
做大输入频率		1			兆赫

（1） LabJack 通过 USB 消耗的电流。状态发光二极管消耗了其中的 4-5 毫安的电流。

（2） 所有已知的台式电脑和一些手提电脑的 USB 控制器向 USB 集线器提供电源。总线供电设备没有向USB接线盒以及一些手提电脑的USB控制器供电。

（3） 这是能够从+5V，模拟输出和数字口流出的总电流。

（4） 每个模拟输入口的输入电流是该口上对地电压的函数。计算公式为：（8.181*Vin-11.67）微安。

（5） 单端 Burst 模式返回二进制偶校验码，所以精度为 11 位。另外，外界噪声会进一步减低其有效精度。

（6） 无负载时，最大的模拟输出电压是等于电源电压。

（7） 必须考虑 1MΩ 接地电流。

（8） 每个 IO 线上有一个 1500Ω 的串联电阻。

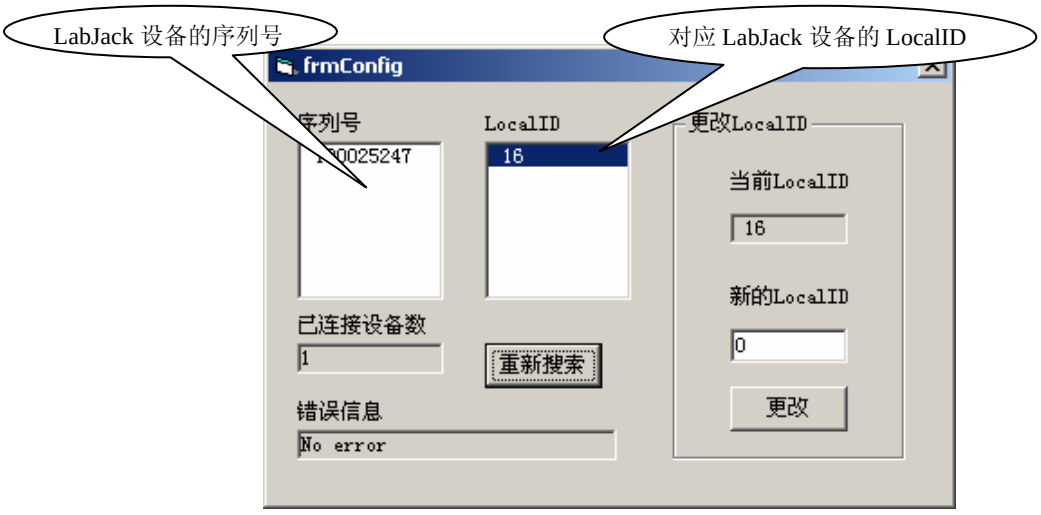
（9） 这些线上没有串联电阻。用户要保证最大电压和电流没有超值。

（10）计数器是斯密特触发输入。

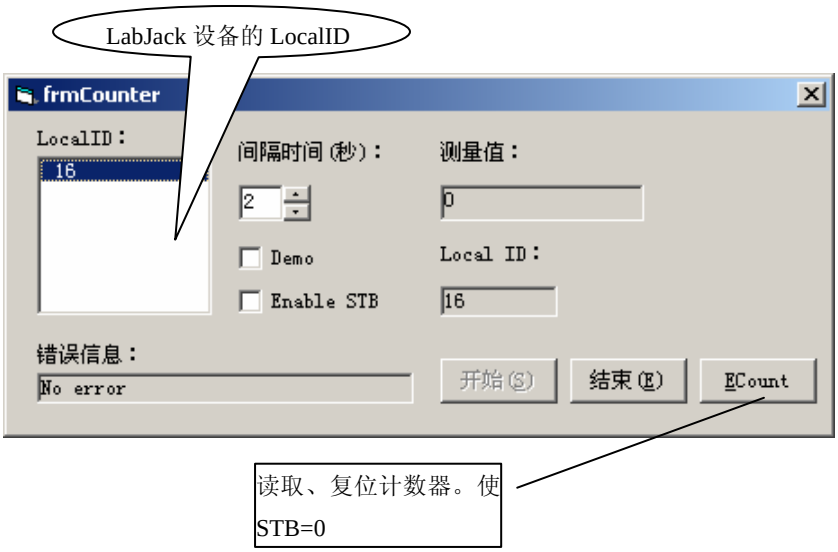
6 附录 B. LabJack 设备的 VB 范例程序

6.1 范例说明

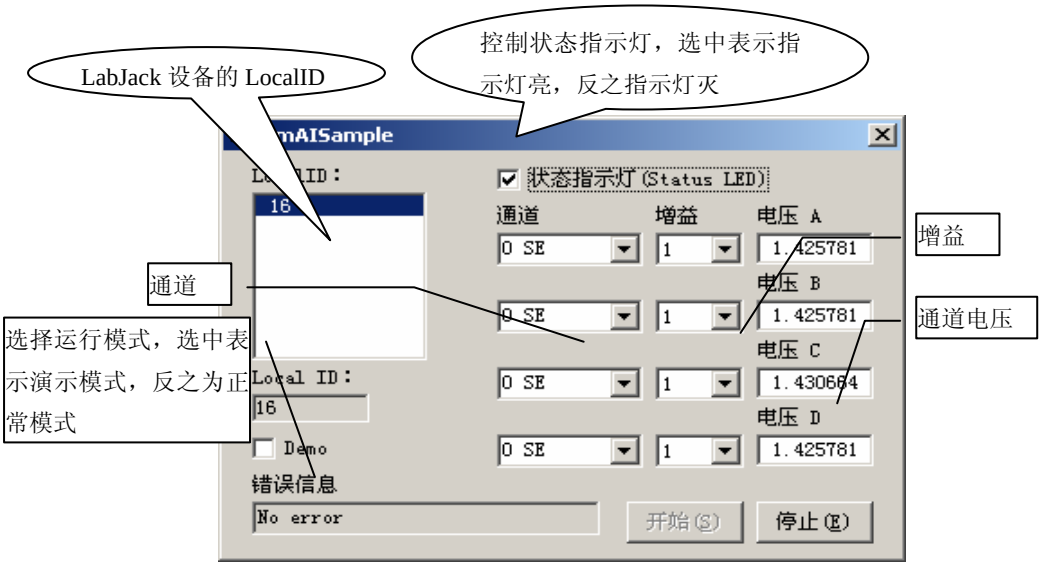
1、VbLJConfig：演示如何使用 ListAll 获取连接在 USB 接口上的所有 LabJack 设备，返回序列号，及 Local ID，运用 LocalID、ReEnum 等函数修改指定设备的 Local ID。同时应用 GetErrorString 来获取错误信息。



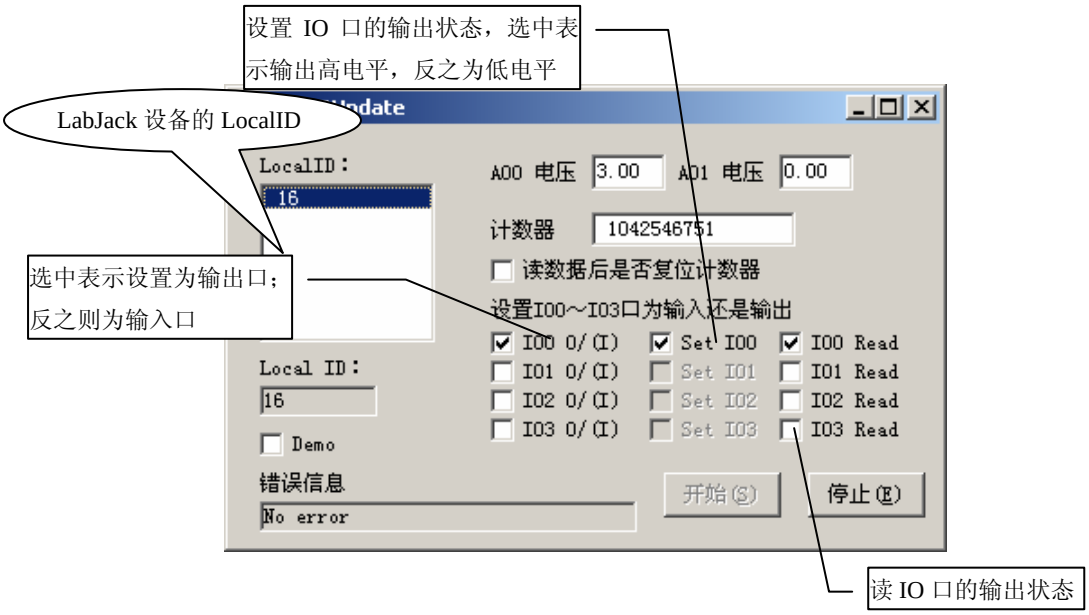
2、VbLJCount：演示如何使用 Counter、Ecount 等函数，控制、读取计数器。



3、VbLJAISample：演示如何使用 AISample 等函数，读取 1、2 或 4 个通道的模拟输入电压，控制、读取 4 个 I/O 口。

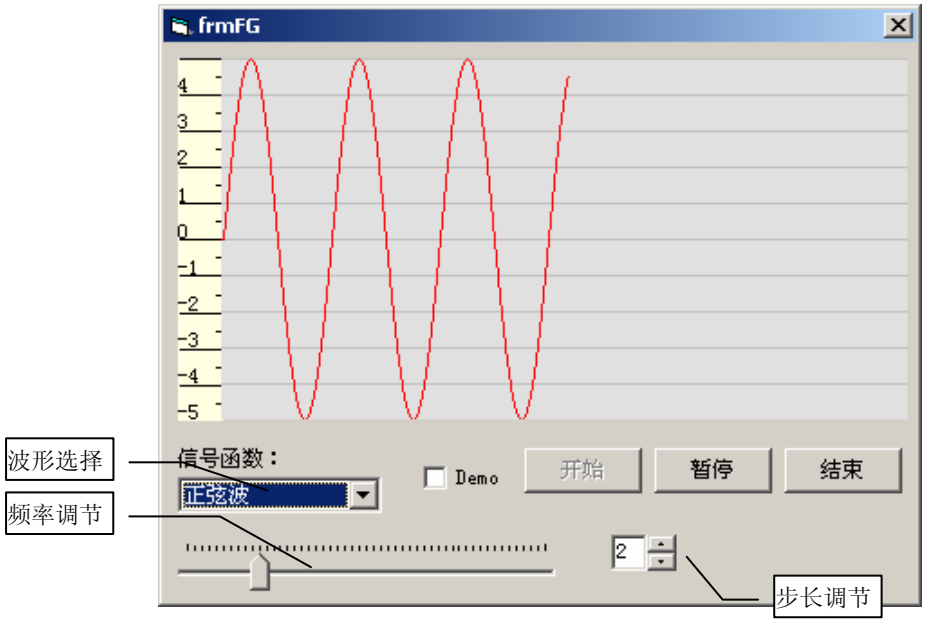


4、VbLJAOUpdate：设置模拟输出电压，控制、读取所有 20 个数字 I/O 口和计数器。

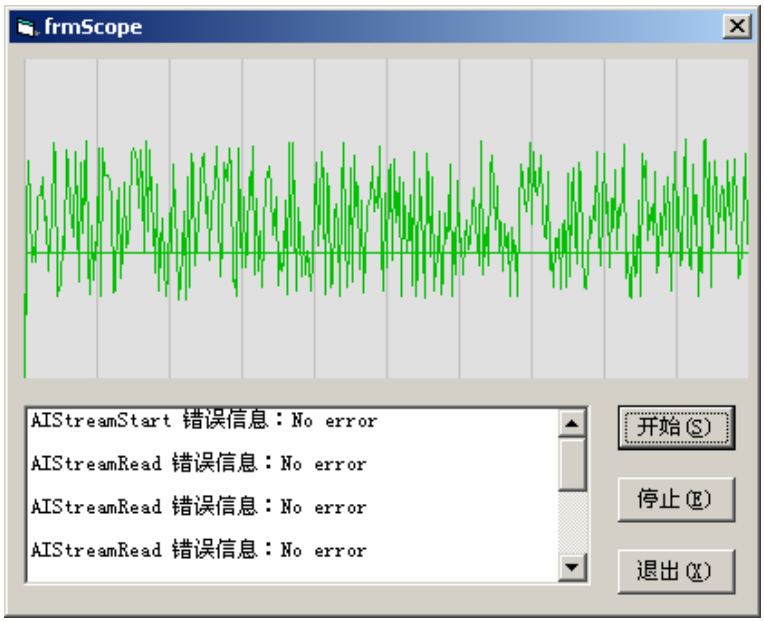


注：AO0、AO1 的输出电压为：0.0 ～ 5.0 伏

5、VbLJFG：演示如何使用 EanalogOut 或 AOUpdate 函数，设置模拟输出电压，并使用图形显示（多种波形选择）。



6、VbLJScope：演示如何使用 AIStreamStart、AIStreamRead、AIStreamClear 等函数，批量读、连续读 1、2 或 4 个通道的模拟输入电压，并使用图形显示。



6.2 函数声明

1、 AISample 函数的简化版本。读取单个模拟输入口的电压。调用本函数后，状态指示灯亮

```
Public Declare Function EAnalogIn Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByVal lngChannel As Long, _
```

```
    ByVal lngGain As Long, _
    ByRef lpOverVoltage As Long, _
    ByRef lpVoltage As Single) As Long
```

2、 AOUpdate 函数的简化版本。设置模拟输出电压。
sngAnalogOut0 < 0 表示保持不变

```
Public Declare Function EAnalogOut Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByVal sngAnalogOut0 As Single, _
    ByVal sngAnalogOut1 As Single) As Long
```

3、 Counter 函数的简化版本。读取、复位计数器。调用本函数，使 STB=0
lpCount: 复位前的当前计数器值

```
Public Declare Function ECount Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByVal lngResetCounter As Long, _
    ByRef lpCount As Long, _
    ByRef lpMs As Long) As Long
```

4、DigitalIO 函数的简化版本。读取单个数字输入口的状态。

```
Public Declare Function EDigitalIn Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByVal lngChannel As Long, _
    ByVal lngReadD As Long, _
    ByRef lpState As Long) As Long
```

5、DigitalIO 函数的简化版本。读取单个数字输出口的状态。

```
Public Declare Function EDigitalOut Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByVal lngChannel As Long, _
    ByVal lngWriteD As Long, _
    ByRef lpState As Long) As Long
```

6、查找连接在 USB 接口上的所有 LabJack 设备。
返回序列号（Serial Number）以及每个本地 ID（Local ID）

```
Public Declare Function ListAll Lib "ljackuw.dll" _
    (ByRef productIDList As Any, _
    ByRef serialnumList As Any, _
    ByRef localIDList As Any, _
    ByRef powerList As Any, _
```

```
    ByRef calMatrix As Any, _
    ByRef numberFound As Long, _
    ByRef Reserved1 As Long, _
    ByRef reserved2 As Long) As Long
```

7、改变指定 LabJack 设备的 Local ID

```
Public Declare Function LocalID Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
   ByVal lngLocalID As Long) As Long
```

8、控制、读取计数器 （监控定时器使能时，关闭计数器）

```
Public Declare Function Counter Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByRef lpStateD As Long, _
    ByRef lpStateIO As Long, _
    ByVal lngResetCounter As Long, _
    ByVal lngEnableSTB As Long, _
    ByRef lpCount As Long) As Long
```

9、设置模拟输出电压。也可以控制/读取所有 20 个数字 I/O 口和计数器

```
Public Declare Function AOUpdate Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByVal lngTrisD As Long, _              //设置 D0～D15
    ByVal lngTrisIO As Long, _
    ByRef lpStateD As Long, _
    ByRef lpStateIO As Long, _
    ByVal lngUpdateDigital As Long, _      //AOUpdate 函数除了进行读操
作，还实行写操作。
    ByVal lngResetCounter As Long, _
    ByRef lngCount As Long, _
    ByVal sngAnalogOut0 As Single, _
    ByVal sngAnalogOut1 As Single) As Long
```

10、读取 1、2、或 4 的模拟输入电压。也可以控制/读取 4 个 I/O 口

```
Public Declare Function AISample Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByRef lpStateIO As Long, _
    ByVal lngUpdateIO As Long, _           //AISample 函数除了进行读操
作，还实行写操作。
    ByVal lngLEDOn As Long, _
    ByVal lngNumChannels As Long, _
```

```
    ByRef alngChannels As Any, _
    ByRef alngGains As Any, _
    ByVal lngDisableCal As Long, _            //若>0，将返回未处理过的
电压值
    ByRef lpOverVoltage As Long, _
    ByRef asngVoltages As Any) As Long
```

11、读写所有 20 个数字 I/O 口

```
Public Declare Function DigitalIO Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByRef lngTrisD As Long, _
    ByVal lngTrisIO As Long, _
    ByRef lpStateD As Long, _
    ByRef lpStateIO As Long, _
    ByVal lngUpdateDigital As Long, _
    ByRef lngOutputD As Long) As Long
```

12、从 1、2 或 4 模拟输入口，以指定的扫描速率（高达 4096）读数据。

数据先存储在 LabJack 的内存缓冲区，然后再传送到 PC。若状态指示灯开，等待触发的时候灯以 4Hz 闪烁。读取时灯灭。传送到 PC 时灯快闪，完成后灯亮。

```
Public Declare Function AIBurst Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByVal lngStateIO As Long, _
    ByVal lngUpdateIO As Long, _
    ByVal lngLEDOn As Long, _
    ByVal lngNumChannels As Long, _
    ByRef alngChannels As Any, _
    ByRef alngGains As Any, _
    ByRef SngScanRate As Single, _
    ByVal lngDisableCal As Long, _
    ByVal lngTriggerIO As Long, _
    ByVal lngTriggerState As Long, _
    ByVal lngNumScans As Long, _
    ByVal lngTimeout As Long, _
    ByRef lpVoltages As Any, _
    ByRef lpStateIOout As Long, _
    ByRef lpOverVoltage As Long, _
    ByVal lngTransferMode As Long) As Long
```

13、数据采样，并存储在 LabJack 的内存缓冲区，同时传送到 PC 应用程序

除了 AIStreamRead 以外的函数调用，将终止 Stream

```
Public Declare Function AIStreamStart Lib "ljackuw.dll" _
```

```
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByVal lngStateIOIn As Long, _
    ByVal lngUpdateIO As Long, _
    ByVal lngLEDOn As Long, _
    ByVal lngNumChannels As Long, _
    ByRef alngChannels As Any, _
    ByRef alngGains As Any, _
    ByRef alngScanRate As Single, _
    ByVal lngDisableCal As Long, _
    ByVal lngReserved1 As Long, _
    ByVal lngReadCount As Long) As Long
```

14、定时扫描，并读取数据

最大样本数是 4096，样本数 ＝ numScans × numChannels。

64 个样本为一块，

Timeout 为函数自身跳出的时间（秒）

```
    Dim Voltages(4, 4096) As Single
    Dim StateIOout(4096) As Long
Public Declare Function AIStreamRead Lib "ljackuw.dll" _
    (ByVal lngIDNum As Long, _
    ByVal lngNumScans As Long, _
    ByVal lngTimeout As Long, _
    ByRef lpVoltages As Any, _
    ByRef lpStateIOout As Long, _
    ByRef lpReserved As Long, _
    ByRef lpLjScanBacklog As Long, _
    ByRef lpOverVoltage As Long) As Long
```

15、结束 Stream

```
Public Declare Function AIStreamClear Lib "ljackuw.dll" _
    (ByVal lpIDNum As Long) As Long
```

16、写异步寄存器，设置 D line 的输入输出方向

```
    full/half delay = 0.833 + 0.833C + 0.667BC + 0.5ABC (us)
    Common baud rates (full A,B,C; half A,B,C):
    1               55,153,232 ;     114,255,34
    10              63,111,28 ;      34,123,23
    100             51,191,2 ;       33,97,3
    300             71,23,4 ;        84,39,1
    600             183,3,6 ;        236,7,1
    1000            33,29,2 ;        123,8,1
    1200            23,17,4 ;        14,54,1
```

2400	21,37,1 ;	44,3,3
4800	10,18,2 ;	1,87,1
7200	134,2,1 ;	6,9,2
9600	200,1,1 ;	48,2,1
10000	63,3,1 ;	46,2,1
19200	96,1,1 ;	22,2,1
38400	3,5,2 ;	9,2,1
57600	3,3,2 ;	11,1,1
100000	3,3,1 ;	1,2,1
115200	9,1,1 ;	2,1,1 or 1,1,1

```
Public Declare Function AsynchConfig Lib "ljackuw.dll" _
    (ByVal lngIDNum As Long, _
    ByVal lngTimeoutMult As Long, _
    ByVal lngConfigA As Long, _
    ByVal lngConfigB As Long, _
    ByVal lngConfigTE As Long, _
    ByVal lngFullA As Long, _
    ByVal lngFullB As Long, _
    ByVal lngFullC As Long, _
    ByVal lngHalfA As Long, _
    ByVal lngHalfB As Long, _
    ByVal lngHalfC As Long) As Long
```

17、控制 LabJack 定时狗。

在 lngActive=1 时，如果 LabJack 与 PC 通讯失败，定时狗在指定的时间内改变数字 I/O 口的状态。

在可靠的无人值守时允许 PC 重启。启动定时狗，计数器暂停使用。

lngTimeout=1～715

lngReset>0,LabJack 在指定的时间复位

```
Public Declare Function Watchdog Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long, _
    ByVal lngDemo As Long, _
    ByVal lngActive As Long, _
    ByVal lngTimeout As Long, _
    ByVal lngReset As Long, _
    ByVal lngActiveD0 As Long, _
    ByVal lngActiveD1 As Long, _
    ByVal lngActiveD8 As Long, _
    ByVal lngStateD0 As Long, _
    ByVal lngStateD1 As Long, _
    ByVal lngStateD8 As Long) As Long
```

18、转换 12 位二进制数据成电压值。与硬件无关。

lngBits = 0～4095
Volts=((2*Bits*Vmax/4096)-Vmax)/Gain

```
Public Declare Function BitsToVolts Lib "ljackuw.dll" _
    (ByVal lngChannel As Long, _
    ByVal lngGain As Long, _
    ByVal lngBits As Long, _
    ByRef lpVolts As Single) As Long
```

19、把电压值转换成 12 位二进制数据。与硬件无关。
Bits=(4096*((Volts*Gain)+Vmax))/(2*Vmax)

```
Public Declare Function VoltsToBits Lib "ljackuw.dll" _
    (ByVal lngChannel As Long, _
    ByVal lngGain As Long, _
    ByVal lngVolts As Single, _
    ByRef lpBits As Long) As Long
```

以下为其他一些辅助的 API 函数声明

20、复位

```
Public Declare Function Reset Lib "ljackuw.dll" _
    (ByRef lpIDNum As Long) As Long
```

21、获取错误信息

```
Public Declare Sub GetErrorString Lib "ljackuw.dll" _
    (ByVal lngErrorcode As Long, _
    ByVal lpErrorString As String)
```

22、获取硬件版本信息

```
Public Declare Function GetFirmwareVersion Lib "ljackuw.dll"_
    (ByRef lpIDNum As Long) As Single
```

23、获取驱动版本信息

```
Public Declare Function GetDriverVersion Lib "ljackuw.dll" () As Single
```

24、获取操作版本信息

```
Public Declare Function GetWinVersion Lib "ljackuw.dll" _
    (ByRef lpMajorVersion As Long, _
    ByRef lpMinorVersion As Long, _
    ByRef lpBuildNumber As Long, _
    ByRef lpPlatformID As Long, _
    ByRef lpServicePackMajor As Long, _
    ByRef lpServicePackMinor As Long) As Long
```

6.3 备注

用户在 VB 开发平台中需调用以上函数时可直接引用 modDll 模块。