

YAV USB 接口采集卡二次开发即 DLL

使用手册

武汉亚为电子科技有限公司
2019.05

USB6000

目 录

| | |
|--------------------------|----|
| 版本说明..... | 3 |
| 64 位开发平台使用方法..... | 3 |
| 函数说明..... | 3 |
| 查找设备 FindDevice..... | 3 |
| 打开设备 OpenYavDevice..... | 4 |
| 读取数据 GetYavData..... | 4 |
| 配置设备 SetYavParam..... | 5 |
| 单次读取数据 GetYavOnce..... | 6 |
| 单次设置数据 SetYavOnce..... | 6 |
| 获取传感器数据 YavSencer..... | 7 |
| 获取感应器状态 YavDI..... | 7 |
| 关闭设备 CloseYavDevice..... | 7 |
| 配置命令..... | 9 |
| 加载流程..... | 11 |
| 多卡同步使用..... | 12 |
| SDK 及例程..... | 13 |
| labview 例程..... | 13 |
| VC 例程..... | 13 |
| C#例程..... | 13 |
| Matlab 例程..... | 18 |
| Delhpi 例程..... | 18 |
| 其他平台例程..... | 19 |

YAV USB 接口采集卡 二次开发及 DLL 使用手册

武汉亚为电子科技有限公司

版本说明

版本号：ADIO86.dll(32 位开发平台)/ADIO64.dll(64 位开发平台)，V20180115

适用范围：带有 YAV 标识，或者武汉亚为电子科技有限公司产品标注，且具备 USB（包括但不限于方口 USB、Micro USB）通信功能的采集卡，均可利用本指南。

特别声明：该指南适应于 2017 年 9 月 1 日之后 YAV 采集卡

64 位开发平台使用方法

YAV USB 接口的采集卡，无需安装驱动，可利用 ADIO86.dll 与其他开发平台产生数据交互，dll 基于 VS2010 开发。部分操作系统加载 dll 时出错，需 msucr100d.dll、msucr120d.dll 配合使用，直接放 ADIO86.dll 同文件夹下即可，如果是 64 位开发平台（注意：64 位操作系统如果用的是 32 位开发平台，依然需要使用 ADIO86 版本），可 ADIO64.dll 改名为 ADIO86.dll，替换并重启软件。该 dll 可用于通用开发平台，例如 VC++、VB、C#、Delphi 等。

函数说明

 调用 ADIO86.dll，使用时多检查参数正确性，勿要过于怀疑函数本身。


查找设备 FindDevice

int FindDevice(void)，初始化 DLL 函数，返回设备数量 n，返回 0 或者 -1，表示没有识别任何设备，此函数推荐单卡用户使用。

int FindYavDevice(char *string)，初始化 DLL 函数，查找 YAV 设备，并返回设备数量 n，返回 0，表示没有识别任何设备，此函数推荐多卡用户使用。string 参数为亚为采集卡设备型号返回的字符串，声明长度不小于 500 字节。

返回：n 设备数量，0 查找失败。

例 num = FindDevice();

 以上两种函数二选一，后者主要用于多卡同一电脑使用，不可两种函数同时都用。该函数如果无法调用，证明用户开发平台 DLL 调用功能没有配置正确。

打开设备 OpenYavDevice

int OpenYavDevice (unsigned short TaskID)

返回：设备临时 ID，一个卡就是 1，两个卡就分别为 1/2，返回值为负数，表示设备异常。-1 无设备，-257/258 通信错误，-259 类型错误。返回数值 255，表示超时。

TaskID: USB 设备编号，单设备为 0（默认），多个设备为 0 1 2.....。

例如 OpenYavDevice (0)

读取数据 GetYavData

int GetYavData(unsigned short TaskID,int *DAQDataBuffer, unsigned int DataSize, int *YavParam, int *CNTBuffer, unsigned int *IOBuffer)

返回：设备临时 ID，一个卡就是 0，两个卡就分别为 0/1，返回值为负数，表示设备异常。-1 无设备，-257/258 通信错误，-259 类型错误。返回数值 255，表示超时。

为了开发方便，此函数可在不调用 **FindDevice** 、 **OpenYavDevice** 函数的情况下，单独使用，第一次调用报错，第二次之后便可实现连续采集。

例如 int DAQDataBuffer[64]={0},YavParam[1]={0},CNTBuffer[2]={0},IOBuffer[2]={0}

GetYavData(0,DAQDataBuffer,64,YavParam,CNTBuffer,IOBuffer)

其中：

TaskID: USB 设备编号，单设备为 0（默认），多个设备为 0 1 2.....。

DAQDataBuffer: 缓存数据数组，初始化长度必须大于 DataSize；LabVIEW 中数据类型为无符号 32 位整形，其他平台均为 unsigned int。数据根据通道数全通道交叉排列，【X00 X01...X0N.....X10 X11...X1N.....XMN】，其中 M 是通道数，M 由采集卡型号决定，N 单次是采样长度，必须为 64 的整数倍。例如两路采集卡，是 0 1 0 1.....,8 路是 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7....., 16 路是 0 1 2~15 0 1 2.....,

循环排列。8IO 采集卡，测频计数功能，返回的依次是 DI0 计数、测频，DI1 计数、测频……DI7 计数、测频值。编码器功能，返回的是编码器 1，编码器 2 的计数值。交替连接。

数据解算：单极性采集卡，例如 0-10V 量程，AI 精度 12 位，为无符号数，转换为十进制，除以 4095，再乘以量程。例如 10V 量程，再乘以 10（实际电阻会有偏差，9.83 会比较精确，具体可用其他第三方设备做校准）会得到通道数据，即真值 $X \times 9.83 / 4095$ 。正负量程的双极性采集卡，例如 AI 精度 16（或 24 位），为有符号数，转换为十进制，除以 $2^{(16-1)-1}$ ，再乘以量程。例如 $\pm 10V$ 量程，再乘以 20。

DataSize：单次采样长度，必须为 32 的整数倍，例如 32 256 512 1024 2048，最大为 81920，是几个通道加起来的数据长度，例如 1024 表示每个通道 512，不可用其他数值，越小越快，越大每次采集的越多更新越慢。为了显示通畅，一般设置为采样率的四分之一到一半，例如 1k 采样率，设置 512 为宜。

YavParam：采集卡参数数组，初始化长度必须大于 1。依次为报警、采样率、通道数、量程。具体参数范围及含义见“配置命令一览表”。

CNTBuffer：数字量读数数组，初始化长度必须大于 2。分别对应通道 1 通道 2 计数/测频。

IOBuffer：IO 量状态数组，初始化长度必须大于 2。分别对应 DI、DO 状态。0 代表 DI 都为低电平，1 代表 DI0 高，DI1 低，3 代表都为高电平。



务必对每一个参数提供正确初始化值，尤其是数组必须注意长度，否则 VB\VC 等平台调用会出现错误或者崩溃。

配置设备 SetYavParam

int SetYavParam(unsigned short TaskID,unsigned char CMD,unsigned char *SetParam)可用于设置 DO 状态。USB 8IO 12IO 采集卡，只用此单个函数，通信效率最高，无需调用 Find、Open、Get 等函数。

此函数可以多线程随意调用，互不影响。调用时间间隔要大于 1ms，否则不加限制的话，影响其他线程的效率。

返回：CMD 代表成功，非 CMD 表示失败。

为了开发方便，此函数可在不调用 FindDevice 、OpenYavDevice 函数的情况下，单独使用，第一次调用报错，第二次之后便可实现连续采集。

例如 int SetParam [16]={0,1,43,4,56,67,8}

GetYavData(0,FD,SetParam)

其中：

TaskID: USB 设备编号, 单设备为 0 (默认), 多个设备为 0 1 2.....。

CMD: 命令参数, 详见“配置命令一览表”。

SetParam: SetParam 数组功能详见“配置命令一览表”, 最多可一次性写入 14 个数据。

该函数功能较多, 可以控制采集卡的工作方式、设置采样率等等。例如设置采样率, 其等级为 05-0F, 对应十进制为 5-15, 对应 200 500 1K 2K 5K 10K 20K 50K 100K 200K Hz (最高采样率根据卡型号视情而定)。
SetYavParam(0,FA, [A,0,0,0])设置设备 0 的采样率为 10K。功能详见“配置命令一览表”。

单次读取数据 GetYavOnce

int GetYavOnce(unsigned short TaskID, unsigned int *DAQDataBuffer)

返回: 设备临时 ID, 一个卡就是 0, 两个卡就分别为 0/1, 返回值为负数, 表示设备异常。-1 无设备, -257/258 通信错误, -259 类型错误。返回数值 255, 表示超时。

该函数为不用调用 FindDevice\OpenYavDevice\GetYavData 等函数, 简易高速单次读取所有通道参数的函数, 能自动打开设备, 读取后关闭设备的方法, 简单易用, 对于采样率要求不高的开发者非常实用。

例如 int DAQDataBuffer[36]={0}

GetYavData(0,DAQDataBuffer)

其中:

TaskID: USB 设备编号, 单设备为 0 (默认), 多个设备为 0 1 2.....。

DAQDataBuffer: 缓存数据数组, 初始化为 36 个长度的数组, 0-31 为通道 AI, 32-35 分别为 DI0/DI1 的测频计数、DIDO。LabVIEW 中数据类型为无符号 32 位整形, 其他平台均为 unsigned int。数据根据通道数全通道交叉排列, 【X00 X01...X0N.....X10 X11...X1N.....XMN】, 其中 M 是通道数, M 由采集卡型号决定, N 单次是采样长度。例如两路采集卡, 是 0 1 0 1....., 8 路是 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7....., 16 路是 0 1 2~15 0 1 2....., 循环排列。

单次设置数据 SetYavOnce

int SetYavOnce(unsigned short TaskID,unsigned char *YavParam)

返回: 设备临时 ID, 一个卡就是 0, 两个卡就分别为 0/1, 返回值为负数, 表示设备异常。-1 无设备, -257/258 通信错误, -259 类型错误。返回数值 255, 表示超时。

例如 int YavParam[16]={0,1,2,3,4,5,6,7}

GetYavData(0,YavParam)

其中：

该函数为不用调用 FindDevice\OpenYavDevice\SetYavParam 等函数，简易高速单次读取所有通道参数的函数，能自动打开设备，读取后关闭设备的方法，简单易用，对于采样率要求不高的开发者非常实用。

YavParam:地址+命令+数据，联合单次下发，最多可一次性写入 16 个数据。

获取传感器数据 YavSencer

```
int YavSencer(unsigned short TaskID, char *string, int *SencerData)
```

返回：设备临时 ID，一个卡就是 0，两个卡就分别为 0/1，返回值为负数，表示设备异常。-1 无设备，-257/258 通信错误，-259 类型错误。返回数值 255，表示超时。

该函数为不用调用 FindDevice\OpenYavDevice\SetYavParam 等函数，简易高速单次读取亚为传感器的数据，能自动打开设备。

其中：

TaskID: USB 设备编号，单设备为 0（默认），多个设备为 0 1 2.....。

string:传感器类型字符串，申请变量长度不小于 20 字符。

SencerData: 获取传感器数据数组，申请数组长度不小于 5，返回几组数据就代表几个。

获取感应器状态 YavDI

```
int YavDI( )
```

返回：感应器状态，此函数专门用于亚为 USB 感应器。其他采集卡无效。

该函数为不用调用其他任何函数，简易单次读取亚为 USB 感应器的数据，能自动打开设备。返回数据即为感应器的状态，例如 0，表示感应状态为低，1 表示一个感应器状态为高，3 表示两个感应器状态均为高。

返回值为负数，表示设备异常。-1 无设备，-257/258 通信错误，-259 类型错误。

关闭设备 CloseYavDevice

```
int CloseYavDevice(unsigned short TaskID)
```

返回：1 代表成功，非 1 失败，此函数一般无需调用。系统能自动关闭。

例如 CloseYavDevice(0)



注意：

本 DLL 适用于任何可以调用 dll 的开发平台，经过测试，目前能支持几乎所有平台所有版本的数据采集，不存在 dll 无法调用或者错误的情况。如果您在开发的过程中发现错误，多半是 dll 加载不正确，或者输入的参数有误。64 位平台注意加载 ADIO64.dll，编译时注意选择 x86 还是 x64。

请注意数据类型，以及数组缓冲区大小。如果缓冲区太小，就会出现内存错误。所以申请数组，可以尽量大一些空间。

配置命令

配置命令一览表

| 序号 号 | 功 能 | | 命令 CMD | 参数数组 16 进制 | | | | | 备注 |
|---------|--------------|------------|---------------|---|----------------------------|---------|--------|-------|-------------------|
| | | | | D0 | D1 | D2 | D3 | D4-13 | |
| 1 | 采集 | 停止采集 | 0xA0 | —— | —— | —— | —— | —— | |
| 2 | | 单次采集 | 0xA1 | —— | —— | —— | —— | —— | |
| 3 | | AI0 触发采集 | 0xA2 | 触发电平（高位） | 触发电平（低位） | 长度（高位） | 长度（低位） | —— | 需定制 |
| | | | | 0-4096 | | 32 的整数倍 | | —— | |
| 4 | | AI0 高速采集 | 0xA3 | —— | —— | —— | —— | —— | |
| 5 | | 测频/计数※ | 0xFC /0xA8 | FF(CH0 测频) CC(CH0 开始计数) | FF(CH1 测频) CC(CH1 开始计数) | —— | —— | —— | 测频/计数二选一。双通道可独立控制 |
| 6 | | AI 连续采集 | 0xAA | N（0-F）参考 FA | —— | —— | —— | —— | 默认 |
| 7 | | 特殊采集 | 0xAF | —— | —— | —— | —— | —— | 备用 |
| 6 | 系数 校准 | 读取校准系数 | 0xB0 | —— | —— | —— | —— | —— | 读漂移参数 |
| 7 | | CH0-3 系数 | 0xB1 | AI0 系数 8 位数组，可设置 950-1250 默认 1000，代表 1 倍，1101 代表 1.101 倍 | | | | | |
| 8 | | CH4-7 系数 | 0xB2 | AI8~AI15 系数 8 位数组 | | | | | |
| 9 | | CH8-11 系数 | 0xB3 | 以上类推 | | | | | |
| 10 | | CH12-15 系数 | 0xB4 | 1、用精密电压源（或精密电流源，具体由通道量程而定）作校准电源连到待校准通道 AI 和 GND。 | | | | | |
| 11 | | CH16-19 系数 | 0xB5 | | | | | | |
| 12 | | CH20-23 系数 | 0xB6 | 2、电源调节到通道量程的最大值（满量程），用 UMS 测量值除以信号源的实际值，换算为 1000 基准的数据，减 900，以 U8 通过 SetYavData 函数发送下去。950-1250，默认 1000，代表 1 倍，1101 代表 1.101 倍。例如 0.99 倍，实际是 990，发送数据为 90，U8 的十六进制为 0x5A。硬件可记忆参数。 | | | | | |
| 13 | | CH24-27 系数 | 0xB7 | | | | | | |
| 14 | | CH28-31 系数 | 0xB8 | | | | | | |
| 15 | 零漂 校准 | 读取零漂 | 0xC0 | 可设置 0-255，I8，127 代表 0 | | | | | 读漂移参数 |
| 16 | | CH0-3 零漂 | 0xC1 | 1、用精密电压源（或精密电流源，具体由通道量程而定）作校准电源连到待校准通道 AI 和 GND。 | | | | | |
| 17 | | CH4-7 零漂 | 0xC2 | | | | | | |
| 18 | | CH8-11 零漂 | 0xC3 | 2、将电源调节到通道量程的最小值（零点，一般是接 GND，不可悬空），把数值的 16 进制 U8 格式，用此命令，通过 SetYavData 函数发送下去。零漂一般都非常小，例如 0x04。硬件可记忆参数。 | | | | | |
| 19 | | CH12-15 零漂 | 0xC4 | | | | | | |
| 20 | | CH16-19 零漂 | 0xC5 | | | | | | |
| 21 | | CH20-23 零漂 | 0xC6 | | | | | | |
| 22 | | CH24-27 零漂 | 0xC7 | | | | | | |

| | | | | | | | | | |
|----|------|---------------|------|---|-------------------|-----------------------------|---------|-------|----------------|
| 23 | | CH28-31 零漂 | 0xC8 | | | | | | |
| 24 | 输出控制 | 数字量 DO 控制※ | 0xD0 | 0X00-0XFF | —— | —— | —— | —— | |
| | | | | D0 每位 BIT15-BIT0 对应 DO15-DO0, 1: 高电平输出, 0: 低电平输出 | | | | | |
| 25 | | 单路 DO 控制 | 0xD1 | 0X00-0X0F 通道号 | 0/1 状态 | —— | —— | —— | |
| | | | | D1 指令,独立控制单路的输出状态 | | | | | |
| 26 | | 状态自复位时间 ms | 0xD9 | 时间高位 0X00-0XFF | 时间低位 0X00-0XFF | —— | —— | —— | |
| | | | | D9 用于控制 DO 输出的保持毫秒时间,超时状态自动复位, D0D1 组成 U16, 最大 65534ms,发送 0XFFFF 表示状态持续保持。 | | | | | |
| 27 | | 模拟量 DA 输出※ | 0xDA | 通道号 0-7 | 高低 | 低位 | 后一通道类推 | | |
| | | | | 输出 3.3V, 则 3.3*4096/10, 变为十六进制, 再分配给高低位, 12 位 | | | | —— | |
| 28 | | PWM 输出※频率 | 0xDB | 频率 (高位) | 频率 (低位) | 频率 (高位) | 频率 (低位) | 频率 | |
| | | | | 通道 1, 1-FFFF 代表 1Hz~65.535KHz | | | 通道 2 | 通道 n | |
| 29 | | PWM 输出※占空比 | 0xDC | 占空比 1 | 占空比 2 | 占空比 3 | 占空比 4 | 占空比 n | |
| 30 | | | | 0-255 对应 0-100% | | | | | |
| 31 | | PWM 输出※ | 0xFD | 频率 (高位) | 频率 (低位) | 占空比 1 | 占空比 2 | —— | MAX MAX PRO |
| | | | | 两通道同频, 1-FFFF 代表 1Hz~65.535KHz | | 每个通道占空比不同代表 0-255 对应 0-100% | | —— | |
| | | | | 通道号 | 频率 (高位) | 频率 (低位) | 占空比 1 | —— | 4AD PLUS |
| 32 | 辅助 | 设备地址 | 0xDD | N | —— | —— | —— | —— | |
| 33 | | 恢复出厂 | 0xF0 | —— | —— | —— | —— | —— | |
| 34 | | 设备量程 | 0xF1 | 0X00-0XFF | —— | —— | —— | —— | 用户不可设置 |
| | | | | 0-F 对应 0-100mV /3V /5V / 10V /15V /30V /60V 4-20mA 0-20mA 0-20mV -5-5V -10-10V -20-20V -30-30V -60-60V -20-20mV, FF 代表前后各一半通道的量程, 例如 38, 也就是前一半是 10V, 后一半是 0-20mA 量程 | | | | —— | |
| 35 | | 设备 ID | 0xF2 | —— | —— | —— | —— | —— | 用 USB 助手读 |
| 36 | | 封锁控制 | 0xF3 | 00/F3 | —— | —— | —— | —— | 用户不可控 |
| 37 | | 信号处理 | 0xF4 | 00 为默认 | —— | —— | —— | —— | 备用 |
| 38 | | AI 精度 | 0xF5 | 00 默认精度, 01 为 10 位, 02 为 12 位, 04 为 14 位, 06 为 16 位, 08 位 18 位, 09 位 24 位 | —— | —— | —— | —— | 用户不可设置 |
| 39 | | 兼容性 | 0xF6 | —— | —— | —— | —— | —— | 备用 |
| 40 | | 采集卡型号 | 0xF7 | 接口类型, 6 7 8 | 通道数 | 防护 | —— | —— | 据命名规则表 |
| 41 | | 功 能 | 0xF8 | 75:754 真值 38:8 位 AD (U8) 37:12 位 AD (U12 无符号) 36:16 位 AD (I16 有符号) 35: 24 位 AD (I32 有符号) 21: DI 高速 22: 测频 (I32 位) 23: 计数 (I32 位) 24: 计数 (I32) 测频 (I16) 25: 编码器 (I32 计数 I16 测频) | —— | —— | —— | —— | 用户不可设置 |

| | | | | | | | | | |
|----|----|--------|------|--|--------------------------------|---|---|---|---------------------------|
| 42 | | 通道使能 | 0xF9 | FF 开启全部,01 开 CH0, 02 开 CH0 和 1 | — | — | — | — | 不记忆 |
| 43 | | 采样率 | 0xFA | 采样等级 (05-0F) D0 设置采样率等级的 05-0F, 对应十进制为 5-15, 对应 500 1K 2K 5K 10K 20K 50K 100K 200K Hz (最高采样率根据卡型号视情而定) | — | — | — | — | |
| 44 | | 重 启 | 0xFB | — | — | — | — | — | |
| 45 | 测量 | 测频/计数※ | 0xFC | FF(CH0 测频) CC(CH0 开始计数) | FF (CH1 测频) CC(CH1 开始计数) | — | — | — | 测频/计数二选 一。双通道可 独立控制 |
| 46 | 输出 | 用户参数 | 0xDF | — | — | — | — | — | 备用 |
| 47 | 帮助 | 版本信息 | 0xFE | 01, 2014 02, 20160501 03, 20170901 | — | — | — | — | 用 USB 助手读 |
| 48 | | 帮 助 | 0xFF | — | — | — | — | — | 用 USB 助手读 |

说明：—代表参数无效，可以设置为 0；※代表部分采集卡不具备改功能；编程中不要频繁设置参数，否则影响采集速度。

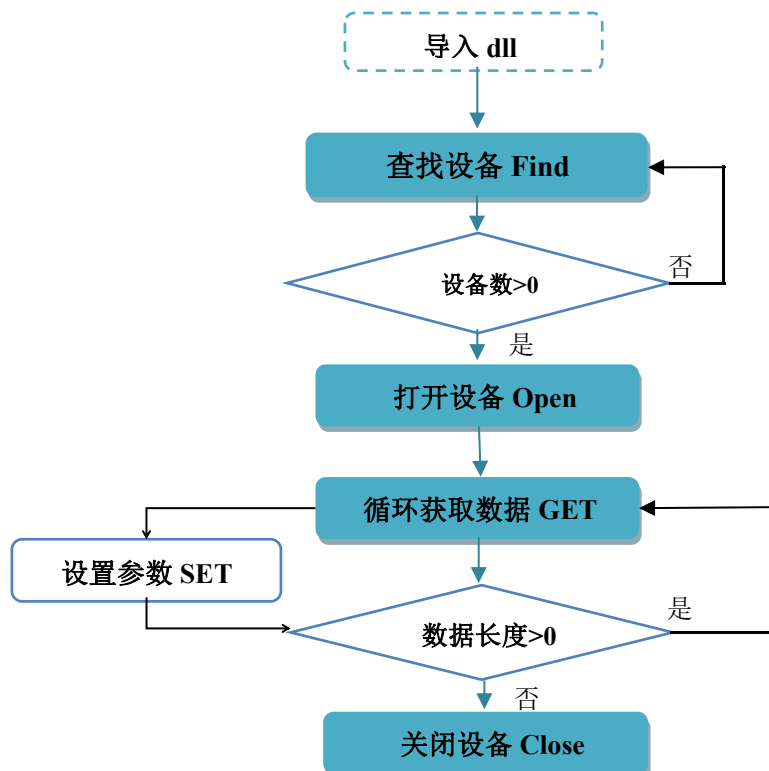
例如：设置 PWM SetYavParam(0, FC,05,00,50,50)。设置的瞬间，影响采集速度。

设备 0 输出为高电平 SetYavParam(0, D0, FF,0,0,0)

DO0 端口输出状态 1，SetYavParam(0,D0,[0,1])

DO1 端口输出状态 0，SetYavParam(0,D0,[1,0])

加载流程



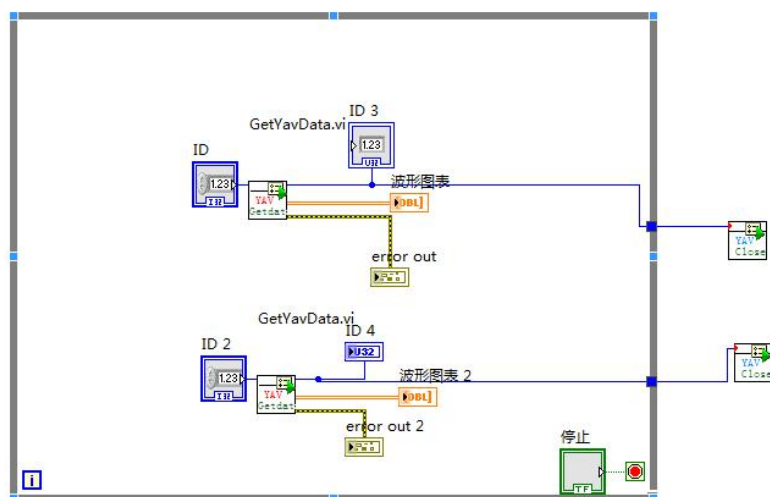
深色背景为必须操作的函数。读取数据返回 255 为超时。在采集速度要求较高的情况下，不要反复配置设备参数与改变 DO 状态，否则影响采集数据时间精度。

多卡同步使用

多卡同步使用，需要复制多份 dll，根据卡的索引 TaskID 号（TaskID 号自动分配，从 0 开始，0、1、2、3.....依次增加，先插入的为 0），调用不同 dll 的函数。调用方法和上面一致。读取数据，可以根据设备号（设备号用户可设置）解算数据。

SDK 及例程

labview 例程



注意：程序框图中重要参数不可调节，否则会引起系统崩溃，尤其是注明不可修改的部分。
以上框图，仅供参考，如有更新，恕不另行通知。

VC++例程

仅供参考，具体以前文的函数说明为准。

```
static int lookup_functions()
{
//  lib_handle = LoadLibraryA("ADIO86.dll");
lib_handle = LoadLibrary(_T("../Debug\\ADIO86.dll")); //64 位平台，加载 ADIO64.dll
if (lib_handle)
{
    ADIOD_FindDevice = (ADIOD_FindDevice_)GetProcAddress(lib_handle, (LPCSTR)"FindDevice");
    ADIOD_FindYavDevice = (ADIOD_FindYavDevice_)GetProcAddress(lib_handle,
(LPCSTR)"FindYavDevice");
    ADIOD_OpenYavDevice = (ADIOD_OpenYavDevice_)GetProcAddress(lib_handle,
(LPCSTR)"OpenYavDevice");
    ADIOD_GetYavData = (ADIOD_GetYavData_)GetProcAddress(lib_handle, (LPCSTR)"GetYavData");
    ADIOD_SetYavParam = (ADIOD_SetYavParam_)GetProcAddress(lib_handle, (LPCSTR)"SetYavParam");
    ADIOD_GetYavOnce = (ADIOD_GetYavOnce_)GetProcAddress(lib_handle, (LPCSTR)"GetYavOnce");
}
```

```
ADIOD_YavSencer = (ADIOD_YavSencer_)GetProcAddress(lib_handle, (LPCSTR)"YavSencer");
ADIOD_SetYavOnce = (ADIOD_SetYavOnce_)GetProcAddress(lib_handle, (LPCSTR)"SetYavOnce");
ADIOD_GetYavDI = (ADIOD_GetYavDI_)GetProcAddress(lib_handle, (LPCSTR)"YavDI");
ADIOD_CloseYavDevice = (ADIOD_CloseYavDevice_)GetProcAddress(lib_handle,
(LPCSTR)"CloseYavDevice");
}
else
    return -1;
return 0;
}
int adio_exit(void)
{
    if (lib_handle)
        FreeLibrary(lib_handle);
    lib_handle = NULL;
    initialized = FALSE;
    return 0;
}
int adio_init(void)
{
    if (!initialized) {
        if (lookup_functions() < 0) {
            adio_exit();
            return -1;
        }
        initialized = TRUE;
    }
    return 0;
}
int _tmain(int argc, _TCHAR* argv[])
{
    int err = 0;
    //本程序可使用于所有亚为的 USB 采集卡
    adio_init();
    if (initialized == FALSE)
        return 0;
    int num = ADIOD_FindDevice();
    if (num)
    {
        printf("YAV: 发现%d 设备! \n", num);
    }
    else
    {
        printf("YAV: 未发现设备! \n");
    }
}
```

```
    adio_exit();
    return -1;
}
err = ADIOD_OpenYavDevice(0);
if (err == 0)
{
    printf("YAV: 设备打开成功! \n");
}
else
{
    adio_exit();
    printf("YAV: 设备打开失败! \n");
}
while (1)
{
    //1. 参数设置例子
    unsigned char m_SetParam[13] = { 0 };
    unsigned char a;
    printf("ADIO: 请输入 1 或者 2 来控制 D00 或者 D01! \n");
    scanf_s("%d", &a);
    m_SetParam[0] = a;
    err = ADIOD_SetYavParam(0, 0xD0, m_SetParam);
    if (err == 0xD0)
    {
        printf("ADIO: 设备输出状态修改成功! \n");
    }
    else
    {
        printf("ADIO: 设备输出状态修改失败! \n");
    }
}
```

//2. 获取采集卡数据例子

```
err = ADIOD_GetYavData(0, m_DAQDataBuffer, 64, m_YavParam, m_CNTBuffer, m_IOBuffer);
if (err == m_YavParam[0])
{
    printf("设备获取数据成功! \n");
    // Print out the returned buffer.
    printf(" 模拟数据: ");
    for (int i = 0; i < 64; i++)
    {
        printf("%02hhx ", m_DAQDataBuffer[i]);
    } //本程序可使用于所有亚为的 USB 采集卡
    printf("\n AI0: %02hhx \n AI1: %02hhx \n .....", m_DAQDataBuffer[0], m_DAQDataBuffer[1]);
    //16 进制 AAABBB, 12 位精度, AAA 三位是 AI0, BBB 三位是 AI1, 以此类推。每个数据包, 根据采集卡
```

通道数循环排列

```
printf("\n 地址: %d \n 采样率: %d \n 通道数: %d \n 量程: %d \n 功能码: %d \n",
m_YavParam[0],
    m_YavParam[1], m_YavParam[2], m_YavParam[3], m_YavParam[4]);
printf(" 通道 1 计数: %d \n 通道 2 计数: %d \n", m_CNTBuffer[0], m_CNTBuffer[1]);
printf(" 输入状态: %d \n 输出状态: %d \n ", m_IIOBuffer[0], m_IIOBuffer[1]);
}
else
{
    printf(" 设备获取数据失败! \n");
}
//3. 获取传感器数据例子
int m_SencerData[4] = { 0 };
char devname[20] = { 0 };
err = ADIOD_YavSencer(0, devname, m_SencerData);
if (err != 0)
{
    printf(" 传感器名称: %s \n 传感器数据: %d %d %d %d \n", devname, m_SencerData[0],
m_SencerData[1], m_SencerData[2], m_SencerData[3]);
}
else
{
    printf(" 设备获取数据失败! \n");
}
//4. 获取红外传感器例子
int m_Data = ADIOD_GetYavDI();
Sleep(300);
}
adio_exit();
return 0;
}
```

C#例程

仅供参考，具体以前文的函数说明为准。

```
[DllImport("ADIO86.dll", EntryPoint = "FindDevice", CharSet = CharSet.Ansi, CallingConvention =
CallingConvention.Cdecl)]
```

```
public static extern UInt32 FindDevice();
//public static extern UInt32 FindDevice(UInt32 uVid, UInt32 uPid);
```

```
[DllImport("ADIO86.dll", EntryPoint = "OpenYavDevice ", CharSet = CharSet.Ansi,
CallingConvention = CallingConvention.Cdecl)]
```

```
public static extern bool OpenYavDevice (UInt32 ID);
```

```
[DllImport("ADIO86.dll", EntryPoint = "GetYavData", CharSet = CharSet.Ansi, CallingConvention
= CallingConvention.Cdecl)]
```

```
public static extern UInt32 GetYavData(UInt32 ID, ref UInt32 ADDDataBuffer, UInt32 len, ref
```



```
UInt32 YavParam, ref UInt32 CNTDataBuffer, ref UInt32 DIODDataBuffer);
```

```
[DllImport("ADIO86.dll", EntryPoint = "SetYavParam", CharSet = CharSet.Ansi,  
CallingConvention = CallingConvention.Cdecl)]
```

```
public static extern bool SetYavParam(UInt32 ID, Byte CMD, Byte Data0, Byte Data1, Byte Data2,  
Byte Data3);
```

```
UInt32 ADtemp;
```

```
UInt32 num;//采样数，必须 64 的整数倍。
```

```
double val1, val2;
```

```
num = (UInt32)Number_TD.Value;
```

```
ADtemp = ADIO.GetYavData(0, ref ADbuf[0], num, ref YavParam[0], ref CNTbuf[0], ref  
DIObuf[0]);
```

```
if (n > 1000)
```

```
{
```

```
    x1.Clear();
```

```
    y1.Clear();
```

```
    x2.Clear();
```

```
    y2.Clear();
```

```
    x3.Clear();
```

```
    y3.Clear();
```

```
    x4.Clear();
```

```
    y4.Clear();
```

```
    n = 0;
```

```
}
```

```
for (int i = 0; i < num / 2; i++)//2 为通道数，USB 2ADIO 设置 2,8AD 设置 8 16ad 设置 16
```

```
{
```

```
    val1 = ADbuf[2 * i] * 10.0 / 4096;
```

```
    val2 = ADbuf[2 * i + 1] * 10.0 / 4096;
```

```
    x1.Add(n);
```

```
    y1.Add((float)val1);
```

```
    x2.Add(n);
```

```
    y2.Add((float)val2);
```

```
    n++;
```

```
}
```

```
//for (int i = 0; i < num / 8; i++)//2 为通道数，USB 2ADIO 设置 2,8AD 设置 8 16ad 设置 16
```

```
//{
```

```
//    val1 = ADbuf[8 * i] * 10.0 / 4096;
```

```
//    val2 = ADbuf[8 * i + 1] * 10.0 / 4096;
```

```
//    val3 = ADbuf[8 * i + 2] * 10.0 / 4096;
```

```
//    val4 = ADbuf[8 * i + 3] * 10.0 / 4096;
```

```
//    val5 = ADbuf[8 * i + 4] * 10.0 / 4096;
```

```
//    val6 = ADbuf[8 * i + 5] * 10.0 / 4096;
```

```
// val7 = ADbuf[8 * i + 6] * 10.0 / 4096;  
// val8 = ADbuf[8 * i + 7] * 10.0 / 4096;  
// x1.Add(n);  
// y1.Add((float)val1);  
// x2.Add(n);  
// y2.Add((float)val2);  
// n++;  
//}
```

C#: <http://www.cnblogs.com/kevin-top/archive/2010/06/04/1751425.html>

Matlab 例程

仅供参考，具体以前文的函数说明为准。

加载 DLL：把编译连接之后产生的 ADIO86.dll 和 ADIO86.h 文件拷贝到 Matlab 的当前工作目录下，输入 `loadlibrary('ADIO86','ADIO86.h');`（一定要有这步，如果 dll 不成功，会显示错误原因）

查看 DLL 中导出的函数

```
libfunctions MatlabDllTest -full
```

-full 选项会列出导出函数的详细输入和输出参数，这是输出信息如下：

Functions in library MatlabDllTest:

```
uint32_t FindDevice( )
```

```
uint32_t OpenYavDevice (uint_t ID)
```

调用函数

```
calllib('ADIO86','OpenYavDevice',0)
```

此时就会输出正确的结果 1

Matlab: http://www.eeworld.com.cn/Test_and_measurement/2013/0730/article_7605.html

Delphi 例程

仅供参考，具体以前文的函数说明为准。

// 以下静态调用 DLL,注意：调用 ADIO86.dll，千万不要弄错！！

```
function FindYavDevice(id:PInteger):integer;stdcall;external 'ADIO86.dll';//发现设备函数
```

```
function OpenYavDevice(id:integer):integer;stdcall;external 'ADIO86.dll';//打开设备函数
```

```
function SetYavParam(id:integer;cmd:Byte;setparam:PChar):integer;stdcall;external 'ADIO86.dll';//写设备
```

命令函数

```
function
```

```
GetYavData(id:integer;ADBuffer:PInteger;DataSize:Word;DABuffer:PInteger;CNTBuffer:PInteger;IOBuffer:PInteger):i  
nteger;stdcall;external 'ADIO86.dll';//写设备命令函数
```

```
function CloseYavDevice(id:integer):integer;stdcall;external 'ADIO86.dll';//关闭设备函数
```

```
{SR *.dfm}
```

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
    AD: array[0..1024] of Integer;
    DA: array[0..4] of Integer;
    CNT: array[0..2] of Integer;
    IO: array[0..2] of Integer;
    ID: array[0..2] of Integer;
    SETPARAM: array[0..6] of Byte;
begin

    edit1.Text:=inttostr(FindYavDevice(@ID));
    edit2.Text:=inttostr(OpenYavDevice(0));
    edit3.Text:=inttostr(SetYavParam(0,$a1,@SETPARAM));//0XA1 命令，单次采集
// 例如要发送(0,$a1,1,2,3,4)，设置 SETPARAM[0]=1;SETPARAM[1]=2;SETPARAM[2]=3;SETPARAM[3]=4;
//参考《YAV USB 接口采集卡二次开发说明》V201703

    edit4.Text:=inttostr(GetYavData(0,@AD,64,@DA,@CNT,@IO));
    //显示 AI0 路 AD 值，12 位采集卡满量程是 4096
    edit5.Text:=inttostr(AD[0]);// AD[0]-AI0 AD[1]-AI1 AD[2]-AI2 依次类推，然后重复 AD 采样值
    edit6.Text:=inttostr(AD[1]);
    edit7.Text:=inttostr(AD[2]);
    edit8.Text:=inttostr(AD[3]);
end;
end.
```

其他平台例程

dll 可供各种开发平台调用。上上页加载流程图中深色背景为必须操作的函数。所有函数，返回 0 为正确，非零为错误。读取数据返回 4 为超时。以下是各平台调用 dll 方法资料：

VB: <http://www.cnblogs.com/wuyifu/p/3376084.html>

附录：函数原型

```
int HID_API_EXPORT_CALL FindDevice( char *string )
int HID_API_EXPORT_CALL HID_API_CALL OpenYavDevice (unsigned short TaskID)
int HID_API_EXPORT_CALL HID_API_CALL GetYavData(unsigned short TaskID,int
*DAQDataBuffer, unsigned int DataSize, int *YavParam, int *CNTBuffer, unsigned int *IOBuffer)
int HID_API_EXPORT_CALL HID_API_CALL SetYavParam(unsigned short
TaskID,unsigned char CMD,unsigned char *SetParam )
int HID_API_EXPORT_CALL HID_API_CALL GetYavOnce(unsigned short
TaskID,unsigned int *DAQDataBuffer)
int HID_API_EXPORT_CALL HID_API_CALL SetYavOnce(unsigned short TaskID,unsigned
char YavParam[])
int HID_API_EXPORT_CALL HID_API_CALL CloseYavDevice( unsigned short TaskID )
```