# PCI-1603
# PCI-1603L
# 用户手册

北京新超仁达科技有限公司

2007.10 月@XC

注意：PCI-1603L 是 PCI-1603 的简化版，不含 DA 功能，其它功能与 PCI-1603 相同。

# 第一章 简要说明

## 1.1 综述

PCI-1603 卡是 PCI 总线的多功能模入模出接口卡,可方便地应用于装有 PCI 总线插槽的微机，具有即插即用（PnP）功能。PC 操作系统可选用目前流行的 Windows 系列、Unix 等多种操作系统以及专业数据采集分析软件 LabVIEW 、LabWindows/CVI 等环境。

PCI-1603 多功能模入模出接口卡安装使用方便，程序编制简单。使用时只需将接口卡插入微机任一 PCI 总线插槽中。其模拟模入模出信号均由卡上的 37 芯 D 型插座与外部信号源及设备连接。

模入部分：用户可根据实际需要选择单端或双端输入方式。输入通道切换可设置成任意连续通道间自动切换或由用户程序切换。卡上 FPGA 芯片集成了 PCI-CORE，并集成 4K 字的先进先出存储器(以下简称 FIFO),便于采集系统在 Windows 等实时多任务操作系统下工作。本卡的 AD 触发方式可以选用软件触发、定时器触发、外部触发、预置触发等触发方式。系统通过查询板上 FIFO 存储状态、AD 转换完成状态,或响应 FIFO 中断或 AD 转换完成中断的方式实现与板卡的通讯和数据交换。

模出部分：4 路模出有多种输出范围选择，加电输出初始态，设置为加电自动输出范围下限电平。

数字量输入输出部分：有 8 路数字量输入和 8 路数字量输出接口，采用 40P 扁平带缆与外部设备连接。也可经转换电缆从 37 芯 D 型插座输出。其中数字量输出具有锁存功能。8 路数字量输出还具有加电自动清零功能。

定时/计数器部分：FPGA 集成了四路 32 位字长的定时/计数通道，以及 2MHz 的基准时钟。四路定时/计数器通道共有方式 0，方式 2，方式 3 三种工作方式，类似于 8254 的工作方式，用户可根据自己需求，灵活地组合成所需的功能。

## 1.2 技术参数

### 1.2.1 模入部分（以下简称 AD）

- AD 通道数：单端 32 路、双端 16 路；
- AD 信号范围： -1.25V～+1.25V；-2.5V～+2.5V； -5V～+5V；－10V～＋10V；
- 输入阻抗：≥ 10MΩ
- AD 转换分辨率：16 位
- AD 转换速度：250KHZ
- AD 数据先进先出缓冲存储器（FIFO）存储深度：4K 字
- AD 触发方式：软件触发；定时触发；外部触发；预置触发；
- AD 通讯方式：AD 转换结束中断、FIFO 半满中断、程序查询；
- AD 转换非线性误差：±3LSB

### 1.2.2 模出部分（以下简称 DA）

注意：PCI-1603L 不含 DA 功能，其它功能与 PCI-1603 相同。

- DA 通道数：4 路
- DA 范围： 0～5V；0～10V；-5V～+5V； －10V～＋10V；
- DA 转换分辨率：16 位
- DA 转换建立时间： ≤ 10μS
- 4 路 DA 加电输出状态：加电同时自动输出下限电平

### 1.2.3 数字量输入输出部分（以下简称 DI/DO）

- DI：8 路；
- DO：8 路；
- 输入输出电平：TTL／CMOS 电平兼容；

### 1.2.4 定时/计数器部分（以下简称 T／C）

- 内部基准时钟：2MHz，占空比 50％
- 定时/计数通道：4 个 32 位定时/计数通道

## 1.3 电源功耗

- +5V ≤ 600mA
- +15V ≤ 100mA
- -15V ≤ 100mA

## 1.4 使用环境要求

工作温度：0℃～70℃
相对湿度：40％～80％
存贮温度：-55℃～+85℃

## 1.5 外型尺寸：（ 不含档板 ）

外型尺寸(不含档板)：长×高＝160㎜×105㎜ （ 6.2 英寸×4.2 英寸）

# 第二章 工作原理

PCI-1603 多功能模入模出接口卡主要由模数转换电路、数模转换电路、FPGA、数字量输入输出电路，定时/计数接口电路构成。

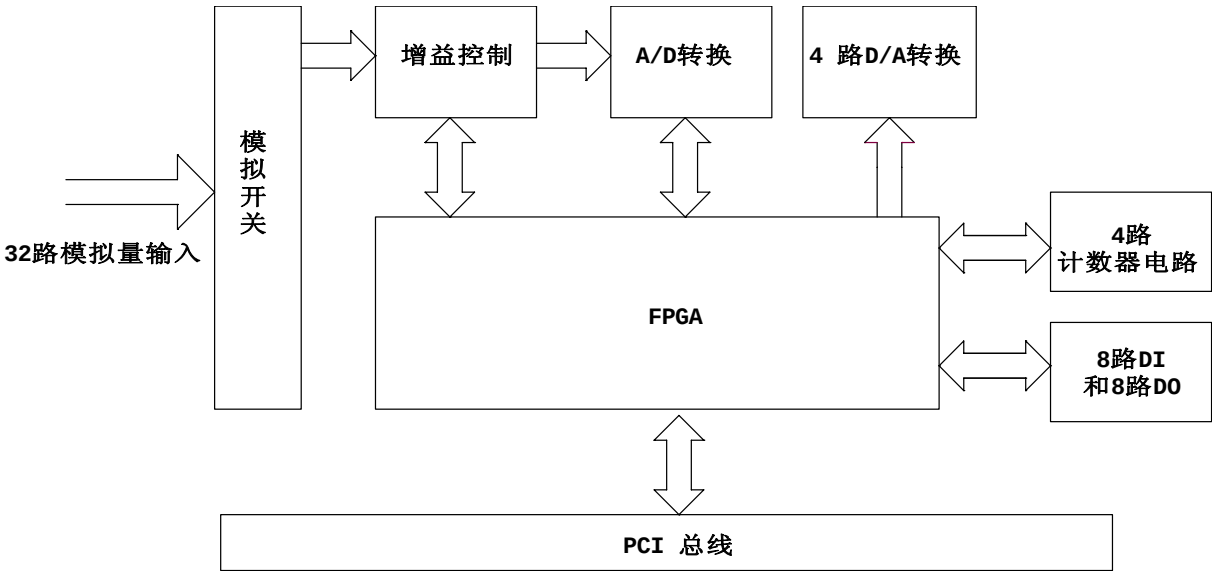



图 1 PCI-1603 工作原理框图

## 2.1 模入部分

外部模拟信号输入首先经多路通道开关的选择后，进入 PGA 放大器。经过 PGA 放大器调理送 AD 转换器，FPAG 控制 AD 芯片进行模数转换，转换后的结果写入 FPGA 内部 FIFO,主机可通过设定好的四种工作方式读取转换结果。

卡上设有通道自动切换电路，程序写入需要循环采集的通道范围（首末通道号），每触发 AD 转换一次，通道开关自动在首末通道之间做加一切换。

模拟量输入方式有单端输入方式和双端输入方式。用户在使用时可根据应用现场的实际需要选择。单端 / 双端输入方式需要软件事先设置。

如用户需要在每个模拟通道输入端设有 RC 滤波电路，可购买 PCLD-880/881 端子板，通过端子板可设定 RC 滤波值。

AD 的量程选择完全由软件实现，每个通道可设置成不同的量程范围，实现了无需跳线的程控增益。

## 2.2 模出部分

DA 输出部分由 DA 转换器件和有关的运放、阻容件和跳线器组成。通过改变跳线器的连接方式，可分别选择不同的电压输出范围。

通过手动设置跨接器 JP1，JP3，JP5，JP7 可分别设置四路 DA 工作在单极性还是双极性输出范围，在加电后自动同时输出范围最低电压。

通过手动设置跨接器 JP2，JP4，JP6，JP8 可分别设置四路 DA 的输出最大幅值是 5V 还是 10V。

## 2.3 数字量输入输出部分

数字量输入输出电路为用户提供 8 路 DI 及 8 路 DO 接口，DO 部分具备加电自动清零功能。

## 2.4 定时/计数器部分

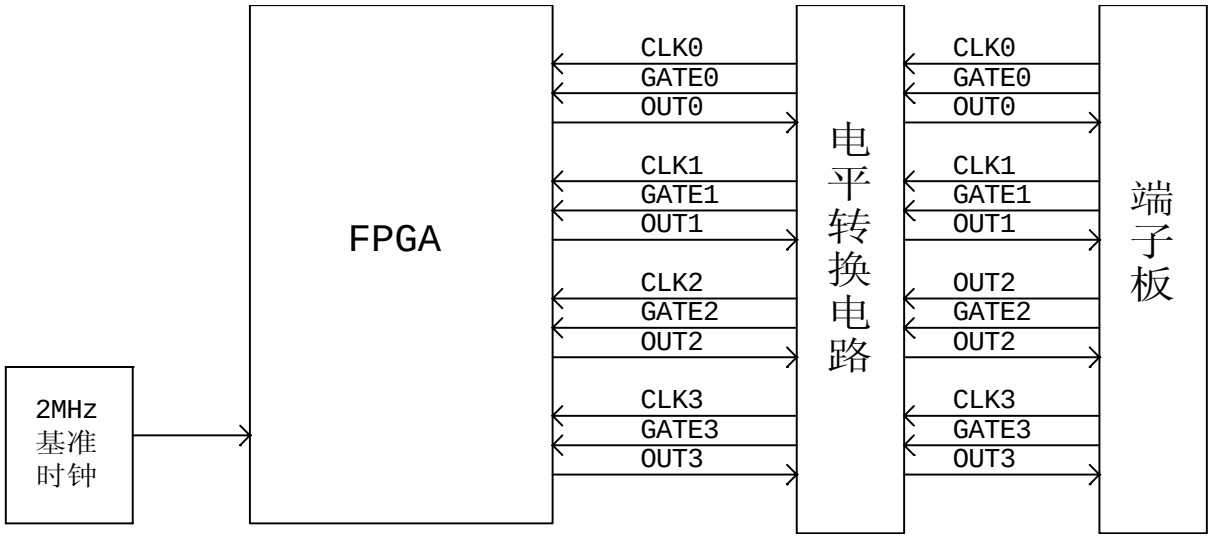



图 2 计数器/计数器工作原理框图

定时/计数器电路由 FPGA，电平转换电路和 2MHZ 基准时钟电路组成。可实现 8254 的方式 0，方式 2 和方式 3 功能，提供 4 路 32 位字长的定时/计数通道，通过灵活的组合可满足用户定时和计数的各种需求。

### 2.4.1 定时/计数器工作方式说明

方式 0：软件装载预置值，启动计数器后，计数器在内时钟或外时钟的上升沿减 1 计数，计数到终点（ 减至 0 ）后输出高电平。在使能外门控时，如果 GATE 为高电电平时停止计数，回到高电平后继续往下计数。再次启动要重新装入计数值或重新编程。

方式 2：比率发生器。编程后重复地循环计数。计数器对内时钟或外时钟的上升沿进行计数，计数到终点时输出一个时钟周期宽度的低电平脉冲，自动初始化后继续计数。在使能外门控时，如果 GATE 为高电电平时停止计数，OUT 输出变为高电平。当 GATE 变为低电平后再继续计数。

方式 3：方波发生器。这种方式是在编程后重复地循环计数，输出波形为方波。计数器对内时钟或外时钟的上升沿进行计数,达到计数终点时输出电平改变。在使能外门控时，如果 GATE 为高电电平时停止计数，回到高电平后继续往下计数，OUT 输出变为高电平。当 GATE 变为低电平后再继续计数。

### 2.4.2 定时/计数器内外时钟方式说明

计数器有两种时钟方式，由软件设定：

1，内时钟方式，内时钟为固定的 2MHz 时钟发生器。

2，外时钟方式，由 40PIN 端子板的 CLK 端子输入时钟信号。

### 2.4.3 定时/计数器门控方式说明

在不使能门控的方式下，计数器工作的停止只受软件控制。

在使能门控的方式下，分两种情况：

1，内门控方式，需要预先设定好门控时间，门控时间以 2MHz 计算。计数器启动后，门控开始，门控到时后，计数器停止输出。

2，外门控方式，软件启动计数器后，是否继续计数由 40PIN 端子板上的 GATE 信号控制，GATE 信号默认为高电平。设计为低电平有效，才开始计数。

# 第三章 安装及使用注意

1. 本卡的安装十分简便，只要将主机机壳打开，在关电情况下，将本卡插入主机的任何一个空余扩展槽中，再将档板固定螺丝压紧即可。37 芯 D 型插座可从主机后面引出并与外设连接。
2. 本卡采用的模拟开关是 COMS 电路，容易因静电击穿或过流造成损坏，所以在安装或用手触摸本卡时，应事先将人体所带静电荷对地放掉，同时应避免直接用手接触器件管脚，以免损坏器件。
3. 当模入通道不全部使用时，应将不使用的通道就近对地短接，不要使其悬空，以避免造成通道间串扰、损坏通道开关。
4. 本卡跳线器较多，使用中应严格按照说明书进行设置操作。电压方式模拟输出时，应避免输出端对地短路。
5. 为保证安全及采集精度，应确保系统地线（计算机及外接仪器机壳）接地良好。特别是使用双端输入方式时，为防止外界较大的共模干扰，应注意对信号线进行屏蔽处理。
6. 禁止带电插拔本接口卡。设置接口卡开关、跨接套和安装接口带缆均应在关电状态下进行。
7. 对外供电端应注意加以保护，严禁短路，否则将造成主机电源损坏，使用中应特别小心。

# 第四章 使用与操作

## 4.1 主要连接器和可调整元件位置图

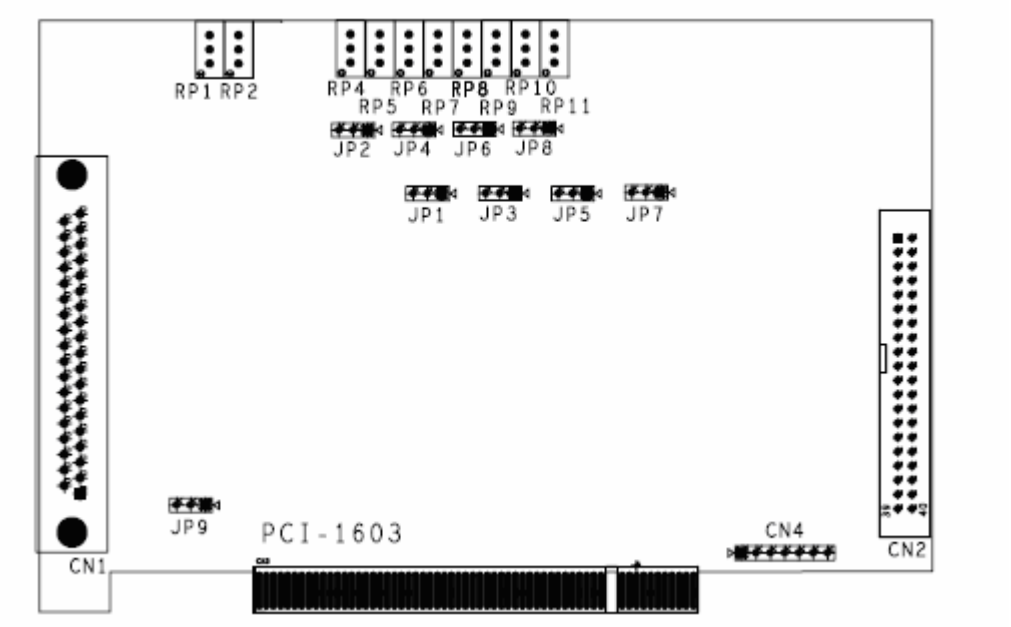



图 4　　主要连接器和可调整元件位置图出厂状态设置

## 4.2 出厂状态设置

- DA 输出范围：0～10V；
- DA 加电输出状态：输出范围最低电压
- 开关量输出 0V

## 4.3 输入输出插座接口定义

### 4.3.1 模拟量输入部分

本卡前端 37 芯 D 型插座（CN1）的信号定义见表 6 。进行 AD 数据采集时，用户可根据需要选择连接信号线(单端)或信号线组(双端)。为减少信号杂波串扰和保护通道开关，凡不使用的信号端应就近与模拟地短接。

表 6　模拟输入插座 CN1 接口定义表

| 插座引脚号 | 信 号 定 义 | 插座引脚号 | 信 号 定 义 |
|---|---|---|---|
| 1 | CH0 | 20 | CH1 |
| 2 | CH2 | 21 | CH3 |
| 3 | CH4 | 22 | CH5 |
| 4 | CH6 | 23 | CH7 |
| 5 | CH8 | 24 | CH9 |
| 6 | CH10 | 25 | CH11 |

| | | | |
|---|---|---|---|
| 7 | CH12 | 26 | CH13 |
| 8 | CH14 | 27 | CH15 |
| 9 | 模 拟 地 | 28 | 模 拟 地 |
| 10 | 模 拟 地 | 29 | 模 拟 地 |
| 11 | CH16 | 30 | CH17 |
| 12 | CH18 | 31 | CH19 |
| 13 | CH20 | 32 | CH21 |
| 14 | CH22 | 33 | CH23 |
| 15 | CH24 | 34 | CH25 |
| 16 | CH26 | 35 | CH27 |
| 17 | CH28 | 36 | CH29 |
| 18 | CH30 | 37 | CH31 |
| 19 | 外触发 | | |

### 4.3.2 数字量输入输出部分

本卡后端 40 芯扁平线插座 CN2 的信号定义见表 7 。

表 7 开关量输入输出信号插座 CN2 端口定义

| 插座引脚号 | 对应 37 针脚号 | 信 号 定 义 | 插座引脚号 | 对应 37 针脚号 | 信 号 定 义 |
|---|---|---|---|---|---|
| 1 | 1 | DO0 | 2 | 20 | DO1 |
| 3 | 2 | DO2 | 4 | 21 | DO3 |
| 5 | 3 | DO4 | 6 | 22 | DO5 |
| 7 | 4 | DO6 | 8 | 23 | DO7 |
| 9 | 5 | DA_OUT0 | 10 | 24 | DA_OUT1 |
| 11 | 6 | 模 拟 地 | 12 | 25 | 模 拟 地 |
| 13 | 7 | DA_OUT2 | 14 | 26 | DA_OUT3 |
| 15 | 8 | 模 拟 地 | 16 | 27 | 模 拟 地 |
| 17 | 9 | DI0 | 18 | 28 | DI1 |
| 19 | 10 | DI2 | 20 | 29 | DI3 |
| 21 | 11 | DI4 | 22 | 30 | DI5 |
| 23 | 12 | DI6 | 24 | 31 | DI7 |
| 25 | 13 | 计数器 0 CLK | 26 | 32 | 计数器 1 CLK |
| 27 | 14 | 计数器 0 GATE | 28 | 33 | 计数器 1 GATE |
| 29 | 15 | 计数器 0 OUT | 30 | 34 | 计数器 1 OUT |
| 31 | 16 | 计数器 2 CLK | 32 | 35 | 计数器 3 CLK |
| 33 | 17 | 计数器 2 GATE | 34 | 36 | 计数器 3 GATE |
| 35 | 18 | 计数器 2 OUT | 36 | 37 | 计数器 3 OUT |
| 37 | 19 | 数字地 | 38 | | 数字地 |
| 39 | | 数字地 | 40 | | 数字地 |

## 4.4 AD 采集方式说明

### 4.4.1 软件触发方式

由软件触发 AD 转换，软件发一次命令，触发一次 AD 转换，软件采用查询方式获取 AD 转换结果。
转换间隔时间由软件决定。

### 4.4.2 定时器触发方式

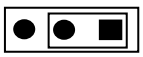
软件设定 AD 采样频率，启动 AD 后，FPGA 按照软件设定好的采样频率控制 AD 转换，转换结果存入 FIFO 中，FIFO 大小为 4K 字，当达到半满时，也就是 2K 字时，产生中断信号通知软件读取转换结果。

### 4.4.3 外部触发方式

外部触发信号每出现一次上升沿，触发一次 AD 转换，转换结果存入 FIFO 中，当 FIFO 半满时，产生中断信号，通知软件读取转换结果。外部触发信号兼容 5V TTL 电平标准。

### 4.4.4 预置触发方式

预置触发方式下，JP9 按照如下跳线。



JP9

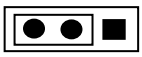
由 DA 通道 3 设置预置触发电平，此时外部触发信号由 CH31 输入。板上设计有比较电路，当外部触发信号穿越触发电平时，开始按照软件设定好的采样频率进行 AD 转换，软件可设定上升沿触发还是下降沿触发，采样的大小为 2K 字的整数倍，由软件灵活设定。

## 4.5 跨接器设置

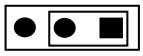
### 4.5.1 模拟信号输出量程选择

DA 的四个通道可分别设置各自的输出量程，JP1,JP3,JP5,JP7 用来选择 DA 输出是单极性还是双极性，JP2,JP4,JP6,JP8 用来选择 DA 输出幅值是 5V 还是 10V.JP1 和 JP2 是对应通道 0 的设置，JP3 和 JP4 是对应通道 1 的设置，JP5 和 JP6 是对应通道 2 的设置，JP7 和 JP8 是对应通道 3 的设置.
下面以通道 0 为例

1， 模拟量输出量程：0V～5V



JP1



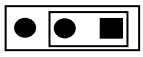
JP2

2，模拟量输出量程：0V～10V

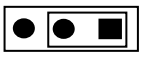
JP1

JP2

3，模拟量输出量程：±5V

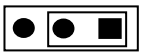


JP1



JP2

4，模拟量输出量程：±10V



JP1

JP2

## 4.5.2 预置触发选择

JP9 出厂默认设置为

JP9

当工作在预置触发方式时，按照如下设置

JP9

## 4.6 模拟信号输入连接方法及注意事项

模拟信号输入有两种输入连接方式：

1. 单端输入方式

32 路模拟电压输入，可按图 5 连成单端输入方式，模拟输入信号连接到 CH0～CH31 输入端，其公共地线连接到 AGND 端。可应用在噪声干扰不高的场合.

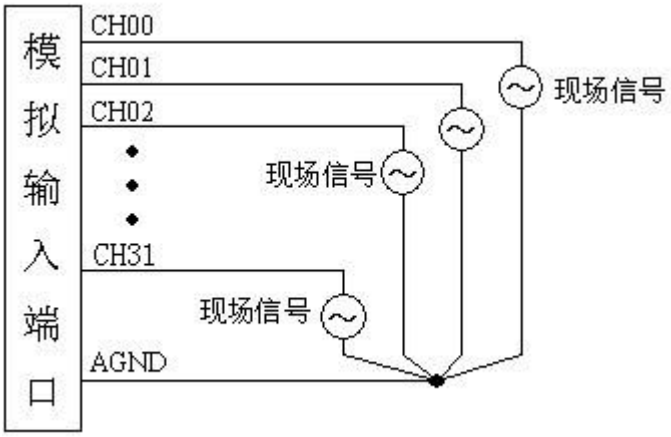



图 5 单端输入方式接线图

2. 双端输入方式

16 路模拟电压输入，可按图 6 连成双端输入方式，模拟输入信号正端分别连接到 CH0、CH2、CH4、CH6…CH30 输入端，负端分别连接到 CH1、CH3、CH5、CH7…CH31 输入端。（其中 CH0 与 CH1 组成一对，CH2 与 CH3 组成一对，CH4 与 CH5 组成一对。。。依此类推，CH30 与 CH31 组成一对。共 16 个差分信号对。）并在距 CN1 插座近处，在 CH1、CH3、CH5、CH7…CH31 端对 AGND 端分别接一只几十 KΩ至几百 KΩ的电阻，为仪表放大器输入电路提供偏置。主要应用在共模干扰较高的场合。

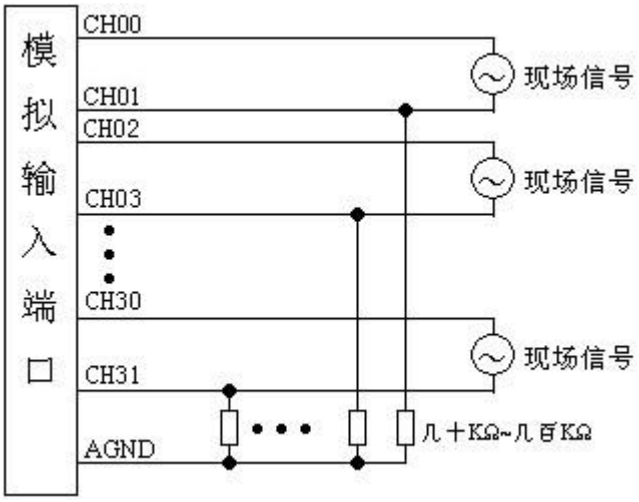



图 6 双端输入方式接线图

# 第五章 调整与校准

## 5.1 准备

产品出厂前，模入部分已按照-10V～+10V 范围调整好,一般情况下用户不需进行调节。本卡的模出部分均已按照单极性 0～10V 范围调整好，如果用户改变了工作模式及范围，可按本节所述方法进行调整，调整应开机预热 20 分钟以上后进行，并准备一块 4 位半以上的数字万用表。

## 5.2 各电位器功能说明

RP1：AD 转换器满度调节。

RP2：AD 转换器零点调节。

## 5.3 模入部分调整

凡改变模入工作方式，如果采样结果偏差大于 20mV 以上的，需要对模入部分进行调整。

1.AD 零点调整：使任一通道与模拟地短接，运行随机带的 test.exe 测试软件(参见第九章),进入采集精度测试菜单,设置好对应的模入范围，启动精度测试采集，即可从屏幕上读取对应通道的采集结果。调整 RP2 使 AD 转换读数值等于零。

2. AD 转换满度调整：在任一通道接入一接近正满度的电压信号，调整 RP1 使 AD 转换读数值等于或接近外信号电压。

3.零点和满度调整应反复进行,以达到理想精度.

## 5.4 模出部分调整

由于本卡出厂前已进行过调整，如改变模出部分量程后，输出误差大，需要对模出部分进行调整。运行随机带的测试软件,进入模拟信号输出测试菜单，选择模出量程 0-10V，以通道 0 为例，做如下调整。

1.零点调整：通过调节 RP4，使 DA0 输出为 0。

2.满度调整：通过调节 RP5，使 DA0 输出为 10V。

3.零点和满度调整应反复进行,以达到理想精度.

说明：RP4 和 RP5 是对应通道 0 的校准，RP6 和 RP7 是对应通道 1 的校准，RP8 和 RP9 是对应通道 2 的校准，RP10 和 RP11 是对应通道 3 的校准.

# 第六章 软件使用说明

## 6.1 简要说明

随机提供的软件是我公司为用户提供的开发包和测试软件。用户如果使用我公司提供的驱动程序，则可以通过随机的开发包，开发自己的用户控制程序；并可以通过测试软件检测

PCI-1603 硬件是否正常工作，并了解 PCI-1603 的参数控制和操作特性。

**注意：**软件应用及其使用说明建立在我公司提供的设备驱动程序上。本说明不涉及用户通过技术说明书编制自己的驱动程序以及根据驱动接口编制的用户控制程序的内容。

## 6.2 提供的软件内容

随板卡提供的软件包括：驱动及安装、测试程序、接口动态连接库和函数说明、编程举例

驱动程序与驱动安装文件,目录路径:

| | |
|---|---|
| \Product_PCI1603\PCI1603 Driver\win2k | Windows 2000 驱动目录 |
| \Product_PCI1603\PCI1603 Driver\winXP | Windows xp 驱动目录 |

接口动态连接库路径

\Product_PCI1603\Develop PCI1603\LibDLL

接口函数定义文件

\Product_PCI1603\Develop PCI1603\Include

测试程序路径

\Product_PCI1603\Test PCI1603

编程举例源码路径

\Product_PCI1603\Samples Source

## 6.3 驱动安装

对于在不同的操作系统下安装驱动程序的方法如下:

### 6.3.1 Windows 2000 驱动安装

1、关闭计算机的电源；
2、将 PCI1603 板卡插入 PCI 查槽中；
3、打开计算机电源，启动 Windows 2000；
4、进入设备管理器，选择其他设备中，未安装程序的 PCI 数据捕获和信号处理控制器
5、打开其属性升级设备驱动程序；
6、选择驱动所在目录，进行安装（\PCI1603 Driver \win2k)；
7、按找到新硬件向导的提示进行下一步；
8、Windows 2000 将显示完成添加/删除硬件向导，单击完成即可完成安装过程
(提示重新启动计算机，则重新启动)

**注：**安装完毕后将在设备管理器中出现一个其他设备(其他设备是问号--不表示设备有问题，只是表示系统 PCI1603 板卡设备类型为其他设备)，设备说明为：PCI_1603 PCI Adapter；

### 6.3.2 Windows XP 驱动安装

1、关闭计算机的电源；
2、将 PCI1603 板卡插入 PCI 查槽中；
3、打开计算机电源，启动 Windows XP；

4、Windows XP 将会显示找到新硬件，可按找到新硬件向导进行下一步；
5、选择从搜索设备的最新驱动程序安装，下一步；
6、选择驱动所在目录，进行安装（\PCI1603 Driver \winXP）；
7、按找到新硬件向导的提示进行下一步；
8、Windows XP 将显示完成添加/删除硬件向导，单击完成即可完成安装过程，重新启动计算机.

**注：**安装完毕后将在设备管理器中出现一个其他设备（其他设备是问号--不表示设备有问题，只是表示系统 PCI1603 板卡设备类型为其他设备），设备说明为：**PCI_1603 PCI Adapter**；

# 第七章 测试程序使用说明

## 7.1 Windows XP 驱动安装

测试程序用于测试 PCI1603 的主要功能工作情况和调试板上参数。可以测试模拟量输入、输出，数字量输入输出，定时器/计数器测试，采集精度测试等，选择程序界面上对应的标签进入各自的测试界面。

## 7.2 模拟量采集(AD)测试

1， 选择采集通道

- 按“输入通道设置”按钮进入“输入通道设置”对话框；
- 选择模拟输入信号方式为单端输入/双端输入；
- 置是否使用通道自动扫描；
- 置通道自动扫描时设置首、末通道值；
- 取消通道自动扫描时设置单通道号值；
- 设置通道值时，最小值为 0，最大值单端输入时=31，双端输入时=15；
- 设置增益选择时，选择全部则所有通道为一种增益，选择增益设置则是为每个通道设置不同增益；按增益设置按钮会弹出设置增益的对话框；
- 按“确认退出”按钮，确认设置，退出对话框；

2， 选择采集方式

- 按“采集方式设置”按钮进入“采集方式控制”对话框；
- 选择 AD 触发方式；
- 如果选择软件触发模式触发 AD，则无需任何设置；
- 如果选择定时器模式触发模式触发 AD，则需要设置定时器出发频率和是否选择 FIFO；
- 如果选择预置触发模式触发 AD，则需要设置定时器出发频率、选择预置触发设置（上升沿或下降沿）和是否选择 FIFO；
- 如果选择外部触发模式触发 AD，则需要设置是否选择 FIFO；
- 按“确认退出”按钮，确认设置，退出对话框。
- 选择记录方式
- 按“记录方式”按钮进入“记录方式”对话框；

3， 选择记录方式

- 选择保存到内存：可以采集数据长度编辑框设置数据到当前内存，最多可保存 163840 点数据；
- 选择内存循环采集：在 163840 长度的数据区内，循环采集数据；
- 选择连续保存到文件：采集数据直到停止采集，数据保存到存储文件名设置的文件中，最大数据长度 163840；
- 选择采集固定长度数据到文件：可以采集数据长度编辑框设置数据到当前内存，最多可保存 163840 点数据；
- 按“确认退出”按钮，确认设置，退出对话框。

4， 选择显示方式

- 按“显示方式”按钮进入“显示方式”对话框；
- 选择波形：左边视图以图形显示一个通道的数据；
- 选择数据值：左边视图以 32 进制显示 32 个通道的数据，对设置的采集通道每屏显示 8 个数据；
- 按“确认退出”按钮，确认设置，退出对话框。

5， 启动采集

- 设置完采集参数和控制方式后，按启动采集按钮，开始采集数据

6， 停止采集

- 按停止采集按钮，停止采集数据

7， 数据显示浏览

- 按“向前一屏”、“向后一屏”浏览数据，波形方式每屏 440 点，数据值方式每屏 8 点；

对波形方式，按“放大”、“衰减”按钮，可调整波形显示幅度。

## 7.3 模拟量采集精度测试

在采集精度测试中进行。按采集精度测试按钮，进入采集精度测试，AD 以软件触发方式工作，用于测试 AD 量程、采集精度等参数；每 400 毫秒刷新一次显示值，同时采集 32 个通道数据，可以以 16 进制方式或电压数据方式显示采集数据。

## 7.4 模拟量输出(DA)测试

界面上有通道显示的选择，可以选择 1、2 通道和 3、4 通道，当选择 1、2 通道时上半部分为模拟输出通道一测试，下半部分为模拟输出通道二测试；当选择 3、4 通道时上半部分为模拟输出通道三测试，下半部分为模拟输出通道四测试

1，选择输出量程

- 按输出量程按钮，进入“输出量程”对话框；
- 择 DA 输出量程，按“确认退出”按钮，确认设置，退出对话框。

2，进行波形设置

- 按输出波形设置，进入“波形设置”对话框；
- 选择周期波形：方波、正弦波、锯齿波输出时，设置波形周期长度，每周期波形需要输出的数据点数，点间延迟由程序循环控制，不同速度主机会产生不同间隔；
- 选择高、低电平，信号按量程的最大最小值输出；
- 选择中间电平，信号按界面中间电平对话框设置的信号电平输出，大（小）于最大（小）量程；按最大（小）量程值输出；

3，按启动输出按钮，模拟信号按设置输出；

4，对周期波形按停止输出按钮，模拟信号停在最后一个点的输出值上；
5，还有带 FIFO 的 DA 模式，需要选上 FIFO，设置起始通道、停止通道和 DA 频率，然后点击启动 DA 按钮就启动 DA 了，点击停止 DA 则停止 DA 输出。

## 7.5 数字量输入输出测试

屏幕上绿灯表示低电平，红灯表示高电平

数字量输出

- 按“数字量输出”按钮，执行数字量输出；可以通过双击输出指示灯切换信号状态，改变输出值；输出值编辑框按 16 进制显示输出数据值；

数字量输入

- 按“数字量输入”按钮，执行数字量输入，指示灯指示输入结果，输入值编辑框按 16 进制显示输入数据值；

## 7.6 定时器/计数器测试

- 通过计数器选择可以选择操作那个计数器；
- 设置计数器的计数值；
- 设置计数器的计数方式（有方式 0、方式 2 和方式 3，等同于 82C54 的方式 0、2、3）
- 选择外时钟则外部脉冲计数；
- 选择使能门控则使能门控，如选择外门控，则使用外部门控信号；如没选择外门控，则需要设置门控计数；
- 按启动按钮，则按照设置好的方式启动计数器；
- 按停止按钮，则按照设置好的方式停止计数器；
- 按回读按钮，则显示各个计数器的当前值。

## 7.7 多块板同时工作测试

- 启动多个测试程序，数量与插入板数相同；
- 对每一个测试程序，按设备选择按钮，进入设备选择对话框，选择一个设备，每个测试程序选择的设备不同，退出设备选择对话框；
- 确认状态编辑框中的是否提示为设备 X 正确打开，提示设备正确打开，则该设备管理正确，硬件资源获取。如果提示打开失败，则程序的设备管理出错。
- 确认打开设备正确后执行各种功能测试，确认各设备工作状态，硬件输入输出及中断正确。

# 第八章 用户编程说明

## 8.1 应用驱动程序编程说明

编程使用本公司提供的驱动时请注意，使用 PCI1603.dll 程序实现 API 接口。仅为用户提供了 VC 格式的 DLL。用户使用 VB 编程时，使用 VC 的 DLL。

控制方式举例由 VC++程序说明，其它语言开发说明请参考在文档中提供的编程举例。

## 8.2 数据结构说明

```
typedef struct tagGAINLIST
{
    USHORT usGainCde;
    FLOAT fMaxGainVal;
    FLOAT fMinGainVal;
    CHAR szGainStr[16];
} GAINLIST;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| usGainCde | Output | USHORT | 该模拟量输入范围的代码 |
| fMaxGainVal | Output | float | 该模拟量输入范围的最大值 |
| fMinGainVal | Output | float | 该模拟量输入范围的最小值 |
| szGainStr | Output | Char 数组 | 该模拟量输入范围的字符描述 |

```
typedef struct tagDEVFEATURES
{
    USHORT    usMaxAIDiffChl;      // Max. number of differential channel
    USHORT    usMaxAISiglChl;      // Max. number of single-end channel

    USHORT    usMaxAOChl;          // Max. number of D/A channel
    USHORT    usMaxDOChl;          // Max. number of digital out channel
    USHORT    usMaxDIChl;          // Max. number of digital input channel
    USHORT    usDIOPort;           // specifies if programmable or not

    USHORT    usMaxTimerChl;       // Max. number of Counter/Timer channel
    USHORT    usMaxAlarmChl;       // Max number of   alram channel
    USHORT    usNumADBit;          // number of bits for A/D converter
    USHORT    usNumADByte;         // A/D channel width in bytes.

    USHORT    usNumDABit;          // number of bits for D/A converter.
    USHORT    usNumDAByte;         // D/A channel width in bytes.

    USHORT    usNumGain ;          // Max. number of gain code
    GAINLIST glGainList[32];       // Gain listing
    DWORD     dwPermutation[4];    // Permutation
} DEVFEATURES, FAR * LPDEVFEATURES;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| usMaxAIDiffChl | Output | USHORT | 差分模拟量输入通道的最大个数 |

| | | | |
|---|---|---|---|
| usMaxAISiglChl | Output | USHORT | 单端模拟量输入通道的最大个数 |
| usMaxTimerChl | Output | USHORT | 计数器/时钟的最大个数 |
| usMaxAlarmChl | Output | USHORT | 报警通道的最大个数 |
| usNumADBit | Output | USHORT | A/D 转换的位数 |
| usNumADByte | Output | USHORT | A/D 转换所需字节个数 |
| usNumGain | Output | USHORT | 输入范围的最大个数 |
| glGainList | Output | GAINLIST 数组 | 所有支持输入范围 |
| dwPermutation | Output | DWORD | 保留 |

```
typedef struct tagPT_DeviceGetFeatures
{
    LPDEVFEATURES buffer;
    USHORT size;
}PT_DeviceGetFeatures, FAR * LPT_DeviceGetFeatures;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| buffer | Output | DEVFEATURES 的指针 | 指向设备特征结构的指针 |
| size | Input | USHORT | 特征结构的数据长度 |

```
typedef struct tagPT_AIConfig
{
    USHORT DasChan;
    USHORT DasGain;
} PT_AIConfig, * LPT_AIConfig;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| DasChan | Input | USHORT | 采样通道 |
| DasGain | Input | USHORT | 采样值范围的代码 |

```
typedef struct tagDEVCONFIG_AI
{
    DWORD     ulChanConfig;          // 0-single ended, 1-differential
    USHORT    usGainCtrMode;
    USHORT    usPolarity;
    USHORT    usDasGain;
    USHORT    usCjcChannel;
} DEVCONFIG_AI, FAR * LPDEVCONFIG_AI;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| ulChanConfig | Output | DWORD | 0 为单端，1 为差分 |

| | | | |
|---|---|---|---|
| usGainCtrMode | Output | USHORT | 保留 |
| usPolarity | Output | USHORT | 保留 |
| usDasGain | Output | USHORT | 保留 |
| usCjcChannel | Output | USHORT | 保留 |

```
// AIGetConfig
typedef struct tagPT_AIGetConfig
{
    LPDEVCONFIG_AI buffer;
    USHORT size;
} PT_AIGetConfig, * LPT_AIGetConfig;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| buffer | Output | DEVONFIG_AI 指针 | 指向设备配置结构的指针 |
| size | Input | USHORT | 该配置结构的大小 |

```
typedef struct tagPT_AIBinaryIn
{
    USHORT chan;
    USHORT TrigMode;
    USHORT *reading;
} PT_AIBinaryIn, * LPT_AIBinaryIn;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| chan | Input | USHORT | 采样通道 |
| TrigMode | Input | USHORT | 触发模式。0 为内部触发；1 为外部触发 |
| reading | Output | USHORT 指针 | 从采样通道读取的原始数据 |

```
typedef struct tagPT_AIScale
{
    USHORT reading;
    FLOAT MaxVolt;
    USHORT MaxCount;
    USHORT offset;
    FLOAT   *voltage;
} PT_AIScale, * LPT_AIScale;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| reading | Input | USHORT | 原始数据 |
| MaxVolt | Input | float | 该输入范围的最大电压值 |
| MaxCount | Input | USHORT | 最大范围值（4095） |

| | | | |
|---|---|---|---|
| offset | Input | USHORT | 0 伏特电压的偏移 |
| voltage | Output | （float）浮点数指针 | 转换后的电压值（伏特） |

```
typedef struct tagPT_AIVoltageIn
{
    USHORT chan;
    USHORT gain;
    USHORT TrigMode;
    FLOAT   *voltage;
} PT_AIVoltageIn, * LPT_AIVoltageIn;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| chan | Input | USHORT | 采样通道 |
| gain | Input | USHORT | 该模拟量输入范围代码 |
| TrigMode | Input | USHORT | 触发模式 |
| voltage | Output | （float）浮点数指针 | 输入的电压，已转换成伏特值的形式 |

```
typedef struct tagPT_MAIConfig
{
    USHORT NumChan;
    USHORT StartChan;
    USHORT *GainArray;
} PT_MAIConfig, * LPT_MAIConfig;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| NumChan | Input | USHORT | 要配置的通道数 |
| StartChan | Input | USHORT | 起始通道 |
| GainArray | Input | USHORT 指针 | 模拟量输入范围代码的数组 |

```
typedef struct tagPT_MAIBinaryIn
{
    USHORT NumChan;
    USHORT StartChan;
    USHORT TrigMode;
    USHORT *ReadingArray;
} PT_MAIBinaryIn, * LPT_MAIBinaryIn;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| NumChan | Input | USHORT | 采样的通道数 |
| StartChan | Input | USHORT | 采样的起始通道 |
| TrigMode | Input | USHORT | 触发模式 |
| ReadingArray | Output | USHORT 指针 | 模拟量输入的原始数据数组 |

```
typedef struct tagPT_MAIVoltageIn
{
    USHORT NumChan;
    USHORT StartChan;
    USHORT *GainArray;
    USHORT TrigMode;
    FLOAT   *VoltageArray;
} PT_MAIVoltageIn, * LPT_MAIVoltageIn;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| NumChan | Input | Unsigned short | 采样的通道数 |
| StartChan | Input | USHORT | 采样的起始通道 |
| GainArray | Input | USHORT 指针 | 各个通道的模拟量输入范围代码 |
| TrigMode | Input | USHORT | 触发模式 |
| VoltageArray | Output | （float）浮点数指针 | 模拟量输入的电压数据数组 |

```
typedef struct tagPT_FAIIntScanStart
{
    USHORT TrigSrc;
    USHORT usRising;
    DWORD SampleRate;
    USHORT NumChans;
    USHORT StartChan;
    USHORT *GainList;
    USHORT *buffer;
    ULONG count;
    USHORT cyclic;
    USHORT IntrCount;
} PT_FAIIntScanStart, * LPT_FAIIntScanStart;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| TrigSrc | Input | USHORT | 触发源:1 为外部触发源, 0 为定时器触发源,2 为预置触发 |
| usRising | Input | USHORT | 0 为上升沿，1 为下降沿（仅预置触发有效） |
| SampleRate | Input | DWORD | 采样频率 |
| NumChans | Input | USHORT | 采样通道数 |

| | | | |
|---|---|---|---|
| StartChan | Input | USHORT | 采样起始通道 |
| GainList | Input | USHORT 指针 | 各个通道输入范围代码 |
| buffer | Output | USHORT 指针 | 用户分配的缓存 |
| count | Input | ULONG | 采样的次数 |
| cyclic | Input | USHORT | 是否循环模式：1 为循环模式，0 为非循环模式 |
| IntrCount | Input | USHORT | 采样多少次一中断，（只有 FIFO 和中断两种采样模式，中断为 1，FIFO 则为半满 FIFO 大小） |

```
typedef struct tagPT_FAITransfer
{
    USHORT ActiveBuf;
    PVOID DataBuffer;
    USHORT DataType;
    ULONG start;
    ULONG count;
    USHORT *overrun;
} PT_FAITransfer, * LPT_FAITransfer;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| ActiveBuf | Input | USHORT | Buffer 的类型，本设备总是为 0 |
| DataBuffer | Output | 指向浮点数或 USHORT 的指针 | 如果 buffer 指针不为空，当数据类型为 USHORT，则该 buffer 的长度应该不小于 **2*count**。当数据类型为浮点数时，则该 buffer 的长度应该不小于 **4*count**。 |
| DataType | Input | USHORT | 数据类型，0 为原始数据，1 为电压值（浮点数） |
| start | Input | ULONG | 从源 buffer 复制到用户 buffer 的起始点 |
| count | Input | ULONG | 从源 buffer 复制到用户 buffer 的采样个数 |
| overrun | Output | ULONG 指针 | overrun 状态<br>0- 无 overrun 发生<br>1- Overrun 发生了 |

```
typedef struct tagPT_FAICheck
{
    USHORT far   *ActiveBuf;
```

```
    USHORT far    *stopped;
    ULONG    far    *retrieved;
    USHORT far    *overrun;
    USHORT far    *HalfReady;
} PT_FAICheck, * LPT_FAICheck;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| ActiveBuf | Output | USHORT 指针 | 本设备总是为 0 |
| stopped | Output | USHORT 指针 | 说明操作是否完成,0 代表未完成，1 代表完成 。 |
| retrieved | Output | ULONG 指针 | A/D 转换的次数 |
| overrun | Output | USHORT 指针 | 是否有 overrun 发生， 0 无 overrun， 1 有 overrun。 |
| HalfReaddy | Output | USHORT 指针 | AD 缓存标志，1 为半满，2 为全满 |

```
typedef struct tagPT_ReadPortByte
{
    USHORT port;
    USHORT *ByteData;
} PT_ReadPortByte, * LPT_ReadPortByte;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| port | Input | USHORT | 端口地址 |
| ByteData | Output | USHORT 指针 | 从端口读取的字节数据 |

```
typedef struct tagPT_WritePortByte
{
    USHORT port;
    USHORT ByteData;
} PT_WritePortByte, * LPT_WritePortByte;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| port | Input | USHORT | 端口地址 |
| ByteData | Input | USHORT 指针 | 写入端口的字节数据 |

```
typedef struct tagPT_ReadPortWord
{
    USHORT port;
    USHORT *WordData;
} PT_ReadPortWord, * LPT_ReadPortWord;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|

| port | Input | USHORT | 端口地址 |
|---|---|---|---|
| WordData | Output | USHORT 指针 | 从端口读取的一个字的数据 |

typedef struct tagPT_WritePortWord
{
    USHORT port;
    USHORT WordData;
} PT_WritePortWord, * LPT_WritePortWord;

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| port | Input | USHORT | 端口地址 |
| WordData | Input | USHORT 指针 | 写入端口的一个字的数据 |

typedef struct tagPT_CheckEvent
{
    USHORT *EventType;
    DWORD Milliseconds;
} PT_CheckEvent, * LPT_CheckEvent;

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| EventType | Output | DWORD | 驱动支持的事件类型，请参考 PT_EnableEvent |
| Milliseconds | Input | USHORT 指针 | 事件超时的毫秒数 |

typedef struct tagPT_EnableEvent
{
    USHORT EventType;
    USHORT Enabled;
    USHORT Count;
} PT_EnableEvent, * LPT_EnableEvent;

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| EventType | Input | USHORT | 希望触发的事件类型 |
| Enabled | Input | USHORT | 使能或取消触发的事件类型。1 为使能；0 为取消 |
| Count | Input | USHORT | 希望事件触发的次数 |

注意：

EventType 的定义：（相应位为 1 表示有该事件的发生）

0 位：中断事件

1 位：buffer change 事件

2 位：termination 事件

3 位：overrun 事件

```
typedef struct tagPT_AOVolCurOut
{
    USHORT chan;
    FLOAT fAOMaxVal;
    FLOAT fAOMinVal;
    FLOAT fValue;
}PT_AOVolCurOut, *LPT_AOVolCurOut;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| chan | Input | USHORT | AO 通道 |
| fAOMaxVal | Input | FLOAT | 输出的最大电压或电流值 |
| fAOMinVal | Input | FLOAT | 输出的最小电压或电流值 |
| fValue | Input | FLOAT | 要输出的电压或电流值 |

```
typedef struct tagPT_AOScale
{
    FLOAT       fAOMaxVal;
    FLOAT       fAOMinVal;
    FLOAT       OutputValue;
    USHORT far * BinData;
} PT_AOScale, * LPT_AOScale;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| fAOMaxVal | Input | FLOAT | 输出的最大电压或电流值 |
| fAOMinVal | Input | FLOAT | 输出的最小电压或电流值 |
| OutputValue | Input | FLOAT | 要输出的电压或电流值 |
| BinData | Output | USHORT 指针 | 转换出来的二进制值 |

```
typedef struct tagPT_FAOIntScanStart
{
    DWORD SampleRate;
    USHORT StartChan;
    USHORT NumChans;
    USHORT far *buffer;
    ULONG count;
    USHORT cyclic;
} PT_FAOIntScanStart, * LPT_FAOIntScanStart;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| SampleRate | Input | DWORD | 采样频率 |
| StartChan | Input | USHORT | 采样起始通道 |
| NumChans | Input | USHORT | 采样通道数 |

| | | | |
|---|---|---|---|
| buffer | Input | USHORT 指针 | 存放采样数据的缓存 |
| count | Input | ULONG | 采样 buffer 大小 |
| cyclic | Input | USHORT | 是否循环采集 |

typedef struct tagPT_FAOCheck
{
    USHORT far *ActiveBuf;
    USHORT far *stopped;
    ULONG far *retrieved;
} PT_FAOCheck, * LPT_FAOCheck;

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| ActiveBuf | Output | USHORT | 本设备总是为 0 |
| stopped | Output | USHORT | 说明操作是否完成，0 代表未完成，1 代表完成 。 |
| retrieved | Output | ULONG | D/A 转换的次数 |

typedef struct tagPT_DioReadBit
{
    USHORT bit;
    USHORT far * state;
} PT_DioReadBit, * LPT_DioReadBit;

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| bit | Input | USHORT | 要读取的 DI 的位 |
| state | Output | USHORT 指针 | 所读出的 bit 位的值 |

typedef struct tagPT_DioWriteBit
{
    USHORT bit;
    USHORT state;
} PT_DioWriteBit, * LPT_DioWriteBit;

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| bit | Input | USHORT | 要写入的 DO 位 |
| state | Input | USHORT | 所写入的 bit 位的值 |

typedef struct tagPT_DioReadWord
{
    USHORT * mask;
    USHORT * value;
} PT_DioReadWord, * LPT_DioReadWord;

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| mask | Input | USHORT 指针 | 要读取 DI 值的掩码 |
| value | Input | USHORT 指针 | 所读出的 DI 值 |

typedef struct tagPT_DioWriteWord
{
    USHORT mask;
    USHORT value;
} PT_DioWriteWord, * LPT_DioWriteWord;

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| mask | Input | USHORT | 要写入的 DO 值的掩码 |
| value | Input | USHORT | 所写入 DO 的值 |

# 8.3 接口函数说明

## 8.3.1 设备管理

LRESULT Pci1603_DeviceOpen(ULONG ulDeviceNum, PVOID *lpDev);
功能：该函数打开要操作的设备。
参数：ulDeviceNum：是 Device Number（拨码开关）。
    lpDev：是返回的设备指针
返回值：如果执行成功，则返回 0；
    如果未成功，则返回错误代码。

LRESULT Pci1603_SelectDevice(HWND hCaller, PULONG ulDevNum);
功能：该函数选择要操作设备的设备号。
参数：hCaller：调用者的窗口句柄；
    ulDevNum：是返回的设备号（拨码开关）。
返回值：如果执行成功，则返回 0；
    如果未成功，则返回错误代码。

LRESULT Pci1603_DeviceClose(PVOID lpDev);
功能：该函数关闭设备。
参数：lpDev：设备指针。
返回值：如果执行成功，则返回 0；
    如果未成功，则返回错误代码。

LRESULT Pci1603_DeviceGetFeatures(PVOID lpDev, LPT_DeviceGetFeatures lpDevFeatures);
功能：该函数获取设备特征。
参数：lpDev：设备指针；

lpDevFeatures:是 PT_DeviceGetFeatures 结构的指针,该结构包含了设备特征参数,结构定义请参照数据结构说明。
返回值:如果执行成功,则返回 0;
如果未成功,则返回错误代码。

LRESULT Pci1603_DeviceConfig( PVOID lpDev, HWND hCaller );
功能:该函数配置和校验设备。
参数:lpDev:设备指针;
hCaller:是调用该函数程序的窗口句柄。
返回值:如果执行成功,则返回 0;
如果未成功,则返回错误代码。

LRESULT Pci1603_DeviceConfigEx( PVOID lpDev, USHORT usChan, USHORT usSinged);
功能:该函数设置某通道单端或是差分。
参数:lpDev:设备指针;
usChan:要设置的通道;
usSinged:单端或是差分(0 为单端,1 为差分);
返回值:如果执行成功,则返回 0;
如果未成功,则返回错误代码。

## 8.3.2 模拟量输出

LRESULT Pci1603_AIConfig (PVOID lpDev, LPT_AIConfig lpConfig);
功能:该函数配置指定的单个模拟量输入通道。
参数:lpDev:设备指针;
lpConfig:是 PT_AIConfig 结构的指针,该结构包含模拟量输入通道配置的参数,结构定义请参照数据结构说明。
返回值:如果执行成功,则返回 0;
如果未成功,则返回错误代码。

LRESULT Pci1603_AIGetConfig (PVOID lpDev, LPT_AIGetConfig lpAIGetConfig);
功能:该函数获取指定的单个模拟量输入通道的配置
参数:lpDev:设备指针;
lpAIGetConfig:是 PT_AIGetConfig 结构的指针,该结构包含要模拟量输入通道配置的参数,结构定义请参照数据结构说明。
返回值:如果执行成功,则返回 0;
如果未成功,则返回错误代码。

LRESULT Pci1603_AIBinaryIn (PVOID lpDev, LPT_AIBinaryIn lpAIBinaryIn);
功能:该函数读取指定的单个通道模拟量输入的二进制值
参数:lpDev:设备指针;
lpAIBinaryIn:是 PT_AIBinaryIn 结构的指针,该结构包含要获取输入的通道、二进制值等的参数,结构定义请参照数据结构说明。
返回值:如果执行成功,则返回 0;

如果未成功，则返回错误代码。

RESULT Pci1603_AIScale (PVOID lpDev, LPT_AIScale lpAIScale);
功能：该函数转换读取到的二进制值为电压值
参数：lpDev：设备指针；
lpAIScale:是 PT_AIScale 结构的指针,该结构包含转换需要的参数和转换后的数值，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_AIVoltageIn (PVOID lpDev, LPT_AIVoltageIn lpAIVoltageIn);
功能：该函数读取模拟量输入的电压值
参数：lpDev：设备指针；
lpAIVoltageIn：是 PT_AIVoltageIn 结构的指针，该结构包含单个模拟量通道电压输入的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_MAIConfig (PVOID lpDev, LPT_MAIConfig lpMAIConfig);
功能：该函数配置指定的多个模拟量输入通道
参数：lpDev：设备指针；
lpMAIConfig:是 PT_MAIConfig 结构的指针,该结构包含配置多个模拟量通道的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_MAIBinaryIn (PVOID lpDev, LPT_MAIBinaryIn lpMAIBinaryIn);
功能：该函数读取指定的多个通道模拟量输入的二进制值
参数：lpDev：设备指针；
lpMAIBinaryIn:是 PT_MAIBinaryIn 结构的指针,该结构包含要获取输入的起始通道、通道数以及二进制值数组等的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_MAIVoltageIn (PVOID lpDev, LPT_MAIVoltageIn lpMAIVoltageIn);
功能：该函数读取多个模拟量通道输入的电压
参数：lpDev：设备指针；
lpMAIVoltageIn：是 PT_MAIVoltageIn 结构的指针，该结构包含多个模拟量通道同时电压输入的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

### 8.3.3 I/O 端口操作

LRESULT Pci1603_ReadPortByte( PVOID lpDev, LPT_ReadPortByte lpReadPortByte );
功能：该函数读一个字节的 I/O 端口数据
参数：lpDev：设备指针；
lpReadPortByte：是 PT_ReadPortByte 结构的指针，该结构包含读取一字节端口数据的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_WritePortByte( PVOID lpDev, LPT_WritePortByte lpWritePortByte );
功能：该函数写一个字节数据到 I/O 端口
参数：lpDev：设备指针；
lpWritePortByte：是 PT_WritePortByte 结构的指针，该结构包含写一个字节数据到端口的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_ReadPortWord( PVOID lpDev, LPT_ReadPortWord lpReadPortWord );
功能：该函数从 I/O 端口读一个字的数据
参数：lpDev：设备指针；
lpReadPortWord：是 PT_ReadPortWord 结构的指针，该结构包含从指定的 I/O 端口读取一个字数据的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_WritePortWord( PVOID lpDev, LPT_WritePortWord lpWritePortWord );
功能：该函数写一个字节数据到 I/O 端口
参数：lpDev：设备指针；
lpWritePortWord：是 PT_WritePortWord 结构的指针，该结构包含写一个字的数据到指定 I/O 端口的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

### 8.3.4 模拟量输入操作

LRESULT Pci1603_CheckEvent (PVOID lpDev, LPT_CheckEvent lpCheckEvent);
功能：该函数检查是否有设定的事件发生
参数：lpDev：设备指针；
lpCheckEvent：是 PT_CheckEvent 结构的指针，该结构包含检测事件状态的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_EnableEvent (PVOID lpDev, LPT_EnableEvent lpEnableEvent);
功能：该函数启动或停止事件机制
参数：lpDev：设备指针；
lpEnableEvent：是 PT_EnableEvent 结构的指针，该结构包含使能事件状态的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_GetFifoSize(PVOID lpDev, PULONG lSize);
功能：该函数获取设备 FIFO 大小
参数：lpDev：设备指针；
lSize：是 long 的指针，该参数返回 FIFO 的大小。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_FAIIntScanStart (PVOID lpDev, LPT_FAIIntScanStart lpFAIIntScanStart);
功能：该函数开始多个通道的模拟量采集
参数：lpDev：设备指针；
lpFAIIntScanStart：是 PT_FAIIntScanStart 结构的指针，该结构包含启动多个通道模拟量数据采集的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_FAITransfer (PVOID lpDev, LPT_FAITransfer lpFAITransfer);
功能：该函数把采集到的数据传输到指定的缓存
参数：lpDev：设备指针；
lpFAITransfer:是 PT_FAITransfer 结构的指针,该结构包含传输数据到缓存的参数，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_FAICheck (PVOID lpDev, LPT_FAICheck lpFAICheck);
功能：该函数检查模拟量输入的状态
参数：lpDev：设备指针；
lpFAICheck：是 PT_FAICheck 结构的指针，该结构包含一些状态的指针，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_ClearOverrun(PVOID lpDev);
功能：该函数清除 Overrun 标志
参数：lpDev：设备指针；

返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_FAITerminate (PVOID lpDev);
功能：该函数停止当前模拟量输入
参数：lpDev：设备指针
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

## 8.3.5 模拟量输出

LRESULT Pci1603_AOBinaryOut(PVOID lpDev, USHORT chan, USHORT BinData);
功能：该函数启动一次模拟量输出操作
参数：lpDev：设备指针；
Chan：是模拟量输出通道；
BinData：模拟量输出的 16 进制值；
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_AOVolCurOut(PVOID lpDev, LPT_AOVolCurOut lpAOVolCurOut);
功能：该函数启动一次模拟量输出操作
参数：lpDev：设备指针；
lpAOVolCurOut：是 PT_AOVolCurOut 结构的指针，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_AOScale(PVOID lpDev, LPT_AOScale lpAOScale);
功能：该函数转换模拟量输出数据为二进制值
参数：lpDev：设备指针；
lpAOScale：是 PT_AOScale 结构的指针，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

## 8.3.6 模拟量输出操作

LRESULT Pci1603_FAOIntScanStart(PVOID lpDev, LPT_FAOIntScanStart lpFAOIntScanStart);
功能：该函数启动高速模拟量输出
参数：lpDev：设备指针；
lpFAOIntScanStart：是 PT_FAOIntScanStart 结构的指针，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；

如果未成功，则返回错误代码。

LRESULT Pci1603_FAOCheck(PVOID lpDev, LPT_FAOCheck lpFAOCheck);
功能：该函数检测当前高速模拟量输出状态
参数：lpDev：设备指针；
lpFAOCheck：是 PT_FAOCheck 结构的指针，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_FAOStop(PVOID lpDev);
功能：该函数停止高速模拟量输出
参数：lpDev：设备指针；
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

## 8.3.7 开关量输入输出操作

LRESULT Pci1603_DioReadBit(PVOID lpDev, LPT_DioReadBit lpDioReadBit);
功能：该函数读取指定通道的输入开关量。
参数：lpDev：设备指针；
lpDioReadBit：是 PT_DioReadBit 结构的指针，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_DioWriteBit(PVOID lpDev, LPT_DioWriteBit lpDioWriteBit);
功能：该函数写指定通道的输出开关量。
参数：lpDev：设备指针；
lpDioWriteBit：是 PT_DioWriteBit 结构的指针，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_DioReadWord(PVOID lpDev, LPT_DioReadWord lpDioReadWord);
功能：该函数读取某几个通道的输入开关量。
参数：lpDev：设备指针；
lpDioReadWord：是 PT_DioReadWord 结构的指针，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_DioWriteWord(PVOID lpDev, LPT_DioWriteWord lpDioWriteWord);
功能：该函数写入某几个通道的输出开关量。
参数：lpDev：设备指针；
lpDioWriteWord：是 PT_DioWriteWord 结构的指针，结构定义请参照数据结构说明。
返回值：如果执行成功，则返回 0；

如果未成功，则返回错误代码。

LRESULT Pci1603_DioGetCurrentDOWord( PVOID lpDev, USHORT far * value );
功能：该函数回读输出的开关量。
参数：lpDev：设备指针；
value：是所读出的输出开关量的值。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

## 8.3.8 定时器/计数器操作

LRESULT Pci1603_StartCounter( PVOID lpDev, USHORT usChanMask);
功能：该函数启动计数器。
参数：lpDev：设备指针；
usChanMask：是启动计数器的掩码，0、1、2、3 有效。为 1 时启动计数。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_StopCounter( PVOID lpDev, USHORT usChanMask);
功能：该函数停止计数器。
参数：lpDev：设备指针；
usChanMask：是启动计数器的掩码，0、1、2、3 有效。为 1 时停止计数。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_SetCounterMode( PVOID lpDev, USHORT usChan, USHORT usMode);
功能：该函数设置计数器模式。
参数：lpDev：设备指针；
usChan：计数器通道。
usMode：计数器模式，低四位有效，0、1 位为计数方式（方式 0、方式 2、方式 3），2 位为是否外部门控（1 为外门控），3 位为是否外时钟（1 为外时钟）；
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_SetCounterValue( PVOID lpDev, USHORT usChan, ULONG ulCount);
功能：该函数设置计数器计数值。
参数：lpDev：设备指针；
usChan：计数器通道。
ulCount：计数值；
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_ReadBackCount( PVOID lpDev, USHORT usChan, ULONG* ulCount);
功能：该函数读取计数器计数值。

参数：lpDev：设备指针；
usChan：计数器通道。
ulCount：计数值；
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_SetCounterGate( PVOID lpDev, USHORT usChanGate);
功能：该函数设置计数器门控。
参数：lpDev：设备指针；
usChanGate：低四位有效，各代表某个计数器是否使能门控（1 为使能门控）。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

LRESULT Pci1603_SetGateCount( PVOID lpDev, USHORT usChan, ULONG ulCount);
功能：该函数设置计数器内门控计数。
参数：lpDev：设备指针；
usChan：计数器通道
ulCount：内门控计数值
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

### 8.3.9 其他功能

LRESULT Pci1603_GetErrorMessage(LRESULT lError, TCHAR* lpMsg);
功能：该函数根据错误代码返回错误信息
参数：lError：错误代码，lpMsg：指向错误信息字符串的指针。
返回值：如果执行成功，则返回 0；
如果未成功，则返回错误代码。

# 第九章 多种语言环境编程实现方法举例

## 9.1 VC 实现功能举例

参照提供的 VC++程序源码，在\Product_PCI1603\Samples Source\VC
用户编程时，在程序中包含接口说明文件 PCI1603.h，在工程中添加动态连接库的导入库 PCI1603.lib。或自己使用现实加载。

### 9.1.1 数字量输出

```
PT_DioWriteWord    ptDioWriteWord;
ptDioWriteWord.mask    = 0xFFFF;
```

```
ptDioWriteWord.value = m_IOOut;
Pci1603_DioWriteWord(glpDevice, (LPT_DioWriteWord)&ptDioWriteWord);
```

### 9.1.2 数字量输入

```
PT_DioReadWord      ptDioReadWord;
ptDioReadWord.mask = &usMask;
ptDioReadWord.value = (USHORT far *)&m_IOIn;
 Pci1603_DioReadWord(glpDevice, (LPT_DioReadWord)&ptDioReadWord);
```

### 9.1.3 软件触发 AD 采集

```
LRESULT         ErrCde;
USHORT    usGainCode[32];
PT_MAIConfig                ptMAIConfig;
ptMAIConfig.NumChan      = 32;
ptMAIConfig.GainArray = (USHORT far *) & usGainCode [0];
if ((ErrCde = Pci1603_MAIConfig(glpDevice,
      (LPT_MAIConfig)&ptMAIConfig)) != 0)
{
      Pci1603_GetErrorMessage(ErrCde,(LPSTR)szErrMsg);
      MessageBox((LPCSTR)szErrMsg,"Test",MB_OK);
      return;
}

float      ADValue[32];
PT_MAIVoltageIn             ptMAIVoltageIn;
ptMAIVoltageIn.StartChan = 0;
ptMAIVoltageIn.NumChan      = 32;
ptMAIVoltageIn.GainArray = (USHORT far *) &usGainCode[0];

ptMAIVoltageIn.TrigMode = 0;                    // internal trigger
ptMAIVoltageIn.VoltageArray = (FLOAT far *)&ADValue[0];

if ((ErrCde = Pci1603_MAIVoltageIn(glpDevice,
      (LPT_MAIVoltageIn)&ptMAIVoltageIn)) != 0)
{
      Pci1603_GetErrorMessage(ErrCde,(LPSTR)szErrMsg);
      MessageBox((LPCSTR)szErrMsg,"Test",MB_OK);
      return;
}
```

## 9.1.4 启动高速带 FIFO 中断采集

```
WORD  *pADData=NULL;
PT_FAIIntScanStart       ptFAIIntScanStart;
PT_EnableEvent             ptEnableEvent;
ptFAIIntScanStart.buffer         = (USHORT far *)pADData;

 ptEnableEvent.EventType = PCI_EVT_INTERRUPT    |
             PCI_EVT_BUFCHANGE    |
             PCI_EVT_TERMINATED |
             PCI_EVT_OVERRUN;
 ptEnableEvent.Enabled = 1;
ptEnableEvent.Count      =    2048;

 if ((ErrCde = Pci1603_EnableEvent(glpDevice,
             (LPT_EnableEvent)&ptEnableEvent)) != 0)
 {
       Pci1603_GetErrorMessage(ErrCde,(LPSTR)szErrMsg);
       MessageBox((LPCSTR)szErrMsg,"1603",MB_OK);
       return ;
 }


ptFAIIntScanStart.TrigSrc       = 0;
ptFAIIntScanStart.usRising = 0;

 ptFAIIntScanStart.StartChan    = 0;
 ptFAIIntScanStart.NumChans      = 32;
 ptFAIIntScanStart.GainList      = &usGainCode[0];
 ptFAIIntScanStart.SampleRate = 1000;
 ptFAIIntScanStart.count           = 163840;
 ptFAIIntScanStart.cyclic         = 0;    // no cyclic mode
ptFAIIntScanStart.IntrCount = 2048;

 if ((ErrCde = Pci1603_FAIIntScanStart(glpDevice,
             (LPT_FAIIntScanStart)&ptFAIIntScanStart)) != 0)
 {
       Pci1603_GetErrorMessage(ErrCde,(LPSTR)szErrMsg);
       MessageBox((LPCSTR)szErrMsg,"1603",MB_OK);
       return ;
 }
```

## 9.1.5 获取高速 AD 采集数据线程

```
PT_FAITransfer      ptFAITransfer;      // FAITransfer table
PT_FAICheck         ptFAICheck;
PT_CheckEvent       ptCheckEvent;

ptCheckEvent.EventType = &usEventType;
ptCheckEvent.Milliseconds = 10000;

if ((ErrCde = Pci1603_CheckEvent(glpDevice,
    (LPT_CheckEvent)&ptCheckEvent)) != 0)
{
    err = false;
    AfxMessageBox("测试出错");
    break;
}

if (usEventType & PCI_EVT_BUFCHANGE)
{
    USHORT gwActiveBuf = 0;      // return by FAICheck
    USHORT gwOverrun   = 0;      // return by FAICheck, FAITransfer
    USHORT gwStopped   = 0;      // return by FAICheck
    ULONG gulRetrieved = 0;      // return by FAICheck
    USHORT gwHalfReady = 0;      // return by FAICheck

    ptFAICheck.ActiveBuf = &gwActiveBuf;
    ptFAICheck.stopped   = &gwStopped;
    ptFAICheck.retrieved = &gulRetrieved;
    ptFAICheck.overrun   = &gwOverrun;
    ptFAICheck.HalfReady = &gwHalfReady;

    if ((ErrCde = Pci1603_FAICheck(glpDevice,
        (LPT_FAICheck)&ptFAICheck)) != 0)
    {
        Pci1603_GetErrorMessage(ErrCde,(LPSTR)szErrMsg);
        MessageBox(hWnd,(LPCSTR)szErrMsg,"Thread
Message",MB_OK);
        return ;
    }

    if (gwHalfReady == 1)    // first ready
    {
```

```
                    ptFAITransfer.ActiveBuf = 0;      // single buffer
                    ptFAITransfer.DataType = 1;
                    ptFAITransfer.start       = 0;
                    ptFAITransfer.count       = 8192;
                    ptFAITransfer.overrun    = &gwOverrun;

                    if ((ErrCde = Pci1603_FAITransfer(glpDevice,
                        (LPT_FAITransfer)&ptFAITransfer)) != 0)
                    {
                        Pci1603_GetErrorMessage(ErrCde,(LPSTR)szErrMsg);
                        MessageBox(hWnd,(LPCSTR)szErrMsg,"Thread
Message",MB_OK);
                        return ;
                    }
                }
                else if (gwHalfReady == 2) // second half ready
                {
                    ptFAITransfer.ActiveBuf = 0;      // single buffer
                    ptFAITransfer.DataType = 1;
                    ptFAITransfer.start       = 8192;
                    ptFAITransfer.count       = 8192;
                    ptFAITransfer.overrun    = &gwOverrun;

                    if ((ErrCde = Pci1603_FAITransfer(glpDevice,
                        (LPT_FAITransfer)&ptFAITransfer)) != 0)
                    {
                        Pci1603_GetErrorMessage(ErrCde,(LPSTR)szErrMsg);
                        MessageBox(hWnd,(LPCSTR)szErrMsg,"Thread
Message",MB_OK);
                        return ;
                    }

                }
            }

            // Process terminate event
            if (usEventType & PCI_EVT_TERMINATED)
            {
                err = true;
                break;
            }
```

### 9.1.6 软件 DA

```
if ((ErrCde = Pci1603_AOBinaryOut(glpDevice, 0, daout)) != SUCCESS)
{
    Pci1603_GetErrorMessage(ErrCde,(LPSTR)szErrMsg);
    MessageBox((LPCSTR)szErrMsg,"Test",MB_OK);
}
```

### 9.1.7 启动高速带 FIFO 的 DA

```
USHORT * pDataOut = NULL;
pDataOut = new USHORT[960];

ZeroMemory(pDataOut, 1920);
PT_FAOIntScanStart    ptFAOIntScanStart;
ptFAOIntScanStart.buffer         = pDataOut;              // analog output data
ptFAOIntScanStart.SampleRate = 1000;             // pacer rate = 1K
ptFAOIntScanStart.StartChan    = 0;      // start channel
ptFAOIntScanStart.NumChans      = 2;      // DA Channel number
ptFAOIntScanStart.count           = 960;      // DA conversion number
ptFAOIntScanStart.cyclic         = 1; // cyclic mode

if ((ErrCde = Pci1603_FAOIntScanStart(glpDevice,
        (LPT_FAOIntScanStart)&ptFAOIntScanStart)) != 0)
{
    Pci1603_GetErrorMessage(ErrCde,(LPSTR)szErrMsg);
    MessageBox((LPCSTR)szErrMsg,"Test",MB_OK);
    delete pDataOut;
    pDataOut = NULL;
}
```

## 9.2 VB 接口方法举例

参照提供的 VB 程序源码，在\Product_PCI1603\Samples Source\VB 用户编程时，在程序中包含接口说明文件 PCI1603.BAS。在其中定义了 VB 调用的接口，说明如下：

(参照 VC 部分数据结构说明)

```
Type DEVLIST
        dwDeviceNum  As Long
        szDeviceName(0 To 49) As Byte
        nNumOfSubdevices As Integer
End Type
```

```
Type GainList
        usGainCde     As Integer
        fMaxGainVal   As Single
        fMinGainVal   As Single
        szGainStr(0 To 15)     As Byte
End Type


Type DevFeatures
        usMaxAIDiffChl  As  Integer     ' Max.  number  of  differential
channel
        usMaxAISiglChl As Integer    ' Max. number of single-end channel

        usMaxAOChl As Integer          ' Max. number of D/A channel
        usMaxDOChl As Integer         ' Max. number of digital out channel
        usMaxDIChl  As  Integer         ' Max.  number  of  digital  input
channel
        usDIOPort As Integer         ' specifies if programmable or not

        usMaxTimerChl  As  Integer      ' Max.  number  of  Counter/Timer
channel
        usMaxAlarmChl As Integer     ' Max number of  alram channel
        usNumADBit As Integer         ' number of bits for A/D converter
        usNumADByte As Integer       ' A/D channel width in bytes.

        usNumDABit As Integer       ' number of bits for D/A converter.
        usNumDAByte As Integer      ' D/A channel width in bytes.

        usNumGain As Integer        ' Max. number of gain code
        glGainList(0 To 31) As GainList   ' Gain listing
        dwPermutation(0 To 3) As Long  ' Permutation
End Type

' DeviceGetFeatures
Type PT_DeviceGetFeatures
    buffer As Long         ' LPDEVFEATURES
    size   As Integer
End Type

' AIConfig
Type PT_AIConfig
        DasChan As Integer
        DasGain As Integer
```

```
End Type


' DEVCONFIG_AI
Type DEVCONFIG_AI
                    ulChanConfig   As Long        ' 0-single ended,
1-differential
                    usGainCtrMode  As  Integer        ' 1-by  jumper,
0-programmable
                    usPolarity  As  Integer             ' 0-bipolar,
1-unipolar
                    usDasGain  As  Integer             ' not  used  if
GainCtrMode = 1
                    usCjcChannel  As  Integer         ' cold  junction
channel
End Type


' AIGetConfig
Type PT_AIGetConfig
        buffer As Long         'LPDEVCONFIG_AI
        size As Integer
End Type

' AIBinaryIn
Type PT_AIBinaryIn
        chan As Integer
        TrigMode As Integer
        reading As Long        ' USHORT far * reading
End Type

' AIScale
Type PT_AIScale
        reading As Integer
        MaxVolt As Single
        MaxCount As Integer
        offset As Integer
        voltage As Long      ' FLOAT far *voltage
End Type

' AIVoltageIn
Type PT_AIVoltageIn
        chan As Integer
        gain As Integer
```

```
        TrigMode As Integer
        voltage As Long      ' FLOAT far *voltage
End Type

' MAIConfig
Type PT_MAIConfig
        NumChan As Integer
        StartChan As Integer
        GainArray As Long    ' USHORT far *GainArray
End Type

' MAIBinaryIn
Type PT_MAIBinaryIn
        NumChan As Integer
        StartChan As Integer
        TrigMode As Integer
        ReadingArray As Long    ' USHORT far *ReadingArray
End Type

' MAIVoltageIn
Type PT_MAIVoltageIn
        NumChan As Integer
        StartChan As Integer
        GainArray As Long            ' USHORT *GainArray
        TrigMode As Integer
        VoltageArray As Long         ' FLOAT  *VoltageArray
End Type

' FAIIntStart
Type PT_FAIIntStart
        TrigSrc As Integer
        SampleRate As Long
        chan As Integer
        gain As Integer
        buffer As Long        ' USHORT *buffer
        count As Long
        cyclic As Integer
        IntrCount As Integer
End Type

' FAIIntScanStart
Type PT_FAIIntScanStart
        TrigSrc  As Integer
        SampleRate As Long
```

```
        NumChans As Integer
        StartChan As Integer
        GainList As Long        'USHORT *GainList
        buffer As Long          'USHORT *buffer;
        count As Long
        cyclic As Integer
        IntrCount As Integer
End Type

' FAITransfer
Type PT_FAITransfer
        ActiveBuf As Integer
        DataBuffer As Long              'PVOID DataBuffer
        DataType As Integer
        start As Long
        count As Long
        overrun As Long       'USHORT *overrun
End Type

' FAICheck
Type PT_FAICheck
        ActiveBuf As Long     'USHORT far  *ActiveBuf
        stopped As Long       'USHORT far  *stopped
        retrieved As Long     'ULONG  far  *retrieved
        overrun As Long       'USHORT far  *overrun
        HalfReady As Long     'USHORT far  *HalfReady
End Type

' ReadPortByte
Type PT_ReadPortByte
        port As Integer
        ByteData As Long        'USHORT *ByteData
End Type

' WritePortByte
Type PT_WritePortByte
        port As Integer
        ByteData As Integer
End Type

' ReadPortWord
Type PT_ReadPortWord
        port As Integer
        WordData As Long    'USHORT *WordData;
```

```
End Type

' WritePortWord
Type PT_WritePortWord
        port As Integer
        WordData As Integer
End Type

' CheckEvent
Type PT_CheckEvent
        EventType As Long       'USHORT *EventType
        Milliseconds As Long
End Type

' EnableEvent
Type PT_EnableEvent
        EventType As Long
        Enabled As Long
        count As Long
End Type

' AOVolCurOut
Type PT_AOVolCurOut
        chan As Integer
        fAOMaxVal As Single
        fAOMinVal As Single
        fValue As Single
End Type

' AOScale
Type PT_AOScale
    fAOMaxVal As Single
    fAOMinVal As Single
    OutputValue As Single
    BinData As Long
End Type

' FAOIntScanStart
Type PT_FAOIntScanStart
    SampleRate As Long
    StartChan As Integer
    NumChans As Integer
    buffer As Long
    count As Long
```

```
    cyclic As Integer
End Type

' FAOCheck
Type PT_FAOCheck
    ActiveBuf As Long
    stopped As Long
    retrieved As Long
End Type

' DioReadBit
Type PT_DioReadBit
        bit As Integer
        state As Long    'USHORT far * state;
End Type

' DioWriteBit
Type PT_DioWriteBit
        bit As Integer
        state As Integer
End Type

' DioReadWord
Type PT_DioReadWord
        mask As Long     'USHORT * mask;
        value As Long    'USHORT * value;
End Type

' DioWriteWord
Type PT_DioWriteWord
        mask As Integer
        value As Integer
End Type
```

## 9.3 API 接口函数说明

(参照 VC 部分函数接口说明)

### 9.3.1 设备管理函数

Declare Function Pci1603_DeviceOpen Lib "PCI1603.dll" (ByVal ulDeviceNum As Long, lpDev As Long) As Long

Declare Function Pci1603_SelectDevice Lib "PCI1603.dll" (ByVal hCaller As Long, ulDevNum

As Long) As Long

Declare Function Pci1603_DeviceClose Lib "PCI1603.dll" (ByVal lpDev As Long) As Long

Declare Function Pci1603_DeviceGetFeatures Lib "PCI1603.dll" (ByVal lpDev As Long, lpDevFeatures As PT_DeviceGetFeatures) As Long

Declare Function Pci1603_DeviceConfig Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal hCaller As Long) As Long

Declare Function Pci1603_DeviceConfigEx Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal usChan As Integer, ByVal usSinged As Integer) As Long

### 9.3.2 模拟量输入函数

Declare Function Pci1603_AIConfig Lib "PCI1603.dll" (ByVal lpDev As Long, ByRef lpAIConfig As PT_AIConfig) As Long

Declare Function Pci1603_AIGetConfig Lib "PCI1603.dll" (ByVal lpDev As Long, ByRef lpAIGetConfig As PT_AIGetConfig) As Long

Declare Function Pci1603_AIBinaryIn Lib "PCI1603.dll" (ByVal lpDev As Long, ByRef lpAIBinaryIn As PT_AIBinaryIn) As Long

Declare Function Pci1603_AIScale Lib "PCI1603.dll" (ByVal lpDev As Long, lpAIScale As PT_AIScale) As Long

Declare Function Pci1603_AIVoltageIn Lib "PCI1603.dll" (ByVal lpDev As Long, lpAIVoltageIn As PT_AIVoltageIn) As Long

Declare Function Pci1603_MAIConfig Lib "PCI1603.dll" (ByVal lpDev As Long, lpMAIConfig As PT_MAIConfig) As Long

Declare Function Pci1603_MAIBinaryIn Lib "PCI1603.dll" (ByVal lpDev As Long, lpMAIBinaryIn As PT_MAIBinaryIn) As Long

Declare Function Pci1603_MAIVoltageIn Lib "PCI1603.dll" (ByVal lpDev As Long, lpMAIVoltageIn As PT_MAIVoltageIn) As Long

### 9.3.3 I/O 端口操作

Declare Function Pci1603_ReadPortByte Lib "PCI1603.dll" (ByVal lpDev As Long, lpWritePortByte As PT_ReadPortByte) As Long

Declare Function Pci1603_WritePortByte Lib "PCI1603.dll" (ByVal lpDev As Long, lpWritePortWord As PT_WritePortByte) As Long

Declare Function Pci1603_ReadPortWord Lib "PCI1603.dll" (ByVal lpDev As Long, lpWritePortByte As PT_ReadPortWord) As Long

Declare Function Pci1603_WritePortWord Lib "PCI1603.dll" (ByVal lpDev As Long, lpWritePortWord As PT_WritePortWord) As Long

### 9.3.4 高速 AD 采集函数

Declare Function Pci1603_CheckEvent Lib "PCI1603.dll" (ByVal lpDev As Long, lpCheckEvent As PT_CheckEvent) As Long

Declare Function Pci1603_EnableEvent Lib "PCI1603.dll" (ByVal lpDev As Long, lpEnableEvent As PT_EnableEvent) As Long

Declare Function Pci1603_GetFifoSize Lib "PCI1603.dll" (ByVal lpDev As Long, lSize As Long) As Long

Declare Function Pci1603_FAIIntScanStart Lib "PCI1603.dll" (ByVal lpDev As Long, lpFAIIntScanStart As PT_FAIIntScanStart) As Long

Declare Function Pci1603_FAITransfer Lib "PCI1603.dll" (ByVal lpDev As Long, lpFAITransfer As PT_FAITransfer) As Long

Declare Function Pci1603_FAICheck Lib "PCI1603.dll" (ByVal lpDev As Long, lpFAICheck As PT_FAICheck) As Long

Declare Function Pci1603_ClearOverrun Lib "PCI1603.dll" (ByVal lpDev As Long) As Long

Declare Function Pci1603_FAITerminate Lib "PCI1603.dll" (ByVal lpDev As Long) As Long

### 9.3.5 模拟量输出函数

Declare Function Pci1603_AOBinaryOut Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal chan

As Integer, ByVal BinData As Integer) As Long

Declare Function Pci1603_AOVolCurOut Lib "PCI1603.dll" (ByVal lpDev As Long, lpAOVolCurOut As PT_AOVolCurOut) As Long

Declare Function Pci1603_AOScale Lib "PCI1603.dll" (ByVal lpDev As Long, lpAOScale As PT_AOScale) As Long

### 9.3.6 高速模拟量输出函数

Declare Function Pci1603_FAOIntScanStart Lib "PCI1603.dll" (ByVal lpDev As Long, lpFAOIntScanStart As PT_FAOIntScanStart) As Long

Declare Function Pci1603_FAOCheck Lib "PCI1603.dll" (ByVal lpDev As Long, lpFAOCheck As PT_FAOCheck) As Long

Declare Function Pci1603_FAOStop Lib "PCI1603.dll" (ByVal lpDev As Long) As Long

### 9.3.7 开关量输入输出函数

Declare Function Pci1603_DioReadBit Lib "PCI1603.dll" (ByVal lpDev As Long, lpDioReadBit As PT_DioReadBit) As Long

Declare Function Pci1603_DioWriteBit Lib "PCI1603.dll" (ByVal lpDev As Long, lpDioWriteBit As PT_DioWriteBit) As Long

Declare Function Pci1603_DioReadWord Lib "PCI1603.dll" (ByVal lpDev As Long, lpDioReadWord As PT_DioReadWord) As Long

Declare Function Pci1603_DioWriteWord Lib "PCI1603.dll" (ByVal lpDev As Long, lpDioWriteWord As PT_DioWriteWord) As Long

Declare Function Pci1603_DioGetCurrentDOWord Lib "PCI1603.dll" (ByVal lpDev As Long, ByRef value As Integer) As Long

### 9.3.8 定时器/计数器操作函数

Declare Function Pci1603_StartCounter Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal usChanMask As Integer) As Long

Declare Function Pci1603_StopCounter Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal usChanMask As Integer) As Long

Declare Function Pci1603_SetCounterMode Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal usChan As Integer, ByVal usMode As Integer) As Long

Declare Function Pci1603_SetCounterValue Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal usChan As Integer, ByVal ulCount As Long) As Long

Declare Function Pci1603_ReadBackCount Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal usChan As Integer, ByRef ulCount As Long) As Long

Declare Function Pci1603_SetCounterGate Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal usChanGate As Integer) As Long

Declare Function Pci1603_SetGateCount Lib "PCI1603.dll" (ByVal lpDev As Long, ByVal usChan As Integer, ByVal ulCount As Long) As Long

### 9.3.9 其他功能函数

Declare Function Pci1603_GetAddress Lib "PCI1603.dll" (lpVoid As Any) As Long

Declare Function Pci1603_GetErrorMessage Lib "PCI1603.dll" (ByVal lError As Long, ByVal lpszszErrMsg As String) As Long

# 附录 A. 产品清单及保修

产品清单：

1. PCI-1603 多功能模入模出接口卡一块。
2. 安装光盘一张，包括驱动及用户手册。
3. 挡片及固定锣钉。
4. 40 芯带缆转 37 芯 D 型孔头一个。
5. 防静电袋一个。

本产品自售出之日起两年内，凡用户遵守贮存、运输及使用要求，而产品质量低于技术指标的，凭保修单免费维修。因违反操作规定和要求而造成损坏的，需交纳维修费。

# 附录 B. LABVIEW 编程说明

为了方便 LABVIEW 语言编写程序，本库（PCI1603.DLL）特意新增了一批库函数（函数尾缀为_LV）。例如：Pci1603_AIConfig 与 Pci1603_AIConfig_LV 函数功能相同,只是 Pci1603_AIConfig_LV 函数的参数方便 LABVIEW 调用。函数中的参数取消了原先的结构体，把结构体改为一个个独立的参数。独立参数的含义请参照该结构体的定义。

例如：

Pci1603_AIConfig 函数原型为：Pci1603_AIConfig (PVOID lpDev, LPT_AIConfig lpConfig);第二个参数为一个结构体；

Pci1603_AIConfig_LV 函数原型为：Pci1603_AIConfig_LV (PVOID lpDev, USHORT DasChan, USHORT DasGain); 其中：DasChan, DasGain 就是结构体 PT_AIConfig 的成员。手册前面章节有它的定义，现复制如下：

```
typedef struct tagPT_AIConfig
{
    USHORT DasChan;
    USHORT DasGain;
} PT_AIConfig, * LPT_AIConfig;
```

Member Description:

| 名称 | 方向 | 类型 | 描述 |
|---|---|---|---|
| DasChan | Input | USHORT | 采样通道 |
| DasGain | Input | USHORT | 采样值范围的代码 |

其它尾缀为_LV 的函数也不一一赘述。

详细请参照本公司的 LABVIEW 例子程序。