



驱动函式库使用手册

简体中文版

支持 64 位操作系统

支援 Windows 10

支持大多数 PCI I/O 板卡

► 免责声明

凡使用泓格产品除产品质量造成的损害，泓格科技股份有限公司不承担任何法律责任。泓格科技股份有限公司有义务提供正确及详细的数据，但保留修改权利，且不承担用户非法利用数据对第三方所造成侵害构成的法律责任。

► 版权

版权所有 © 2019 泓格科技股份有限公司,保留所有权利。

► 商标

手册中所涉及所有公司商标，商标名称及产品名称分别属于该商标或名称的拥有者所有。

关于

本手册说明如何透过泓格 UniDAQ 驱动函式库在 Windows 下对泓格数据采集板卡作 I/O 操作。本手册提供了使用泓格板卡的相关讯息。包含 I/O 操作的流程以及每个 API 函数的功能、参数、数据结构的说明。

用户可以使用泓格 UniDAQ 函式库驱动在 Windows 系统下使用 VB、VC、BCB、Delphi、VB.NET、C#.NET、VC.NET、Console 等工具来实作开发。本手册也提供范例程序，利用开发实例向用户说明如何使用泓格 UniDAQ 函式库，提供给使用者参考进行应用开发。

如有本手册未涵盖之内容请来信咨询泓格技术工程师。

Email: service@icpdas.com

Table of Contents

Table Of Contents

Industrial Communication Products

Table of Contents	3
1. 导读	8
1.1. 关于泓格 UniDAQ 驱动程序.....	9
1.2. 支持的泓格产品.....	10
1.3. 系统需求.....	11
2. 开始安装使用	12
2.1. 取得 UniDAQ 驱动函数库安装程序.....	13
2.2. 安装 UniDAQ 驱动程序函数库.....	14
2.3. 移除 UniDAQ 驱动函数库.....	19
3. 开发指南	20
3.1. 应用程序架构.....	21
3.2. 在 Win32 Console.....	22
3.3. 在 Visual Basic 6.0.....	25
3.4. 在 Borland Delphi.....	28
3.5. 在 Borland C++ Builder.....	31
3.6. 在 Visual C++.NET.....	34
3.7. 在 Visual Basic.NET.....	40
3.8. 在 Visual C#.NET.....	46
3.9. 范例程序及文件.....	52
4. 函数应用	53
4.1. 导读.....	54
4.2. 驱动函数库.....	56
4.3. 数字输出输入.....	58
4.3.1. 数字输入.....	59

4.3.2. 数字输出	60
4.4. 模拟输入.....	61
4.5. 模拟输出.....	72
4.6. 计时计数器	74
4.7. 内存存取.....	75
5. 函式参考	76
5.1. 函式支援列表.....	77
5.2. 函式介绍.....	96
5.2.1. 驱动函式集.....	97
Ixud_GetDIIVersion	97
Ixud_DriverInit	97
Ixud_DriverClose.....	98
Ixud_SearchCard	98
Ixud_GetBoardNoByCardID	99
Ixud_GetCardInfo	100
Ixud_ReadPort	101
Ixud_WritePort	102
Ixud_ReadPort32	103
Ixud_WritePort32	104
Ixud_ReadPhyMemory.....	105
Ixud_WritePhyMemory	106
5.2.2. 数字输出输入函式集.....	107
Ixud_SetDIOModes32	107
Ixud_SetDIOMode.....	108
Ixud_ReadDI	109
Ixud_WriteDO.....	110
Ixud_ReadDIBit	111
Ixud_WriteDOBit	112
Ixud_ReadDI32	113

Table Of Contents

Industrial Communication Products

Ixud_WriteDO32.....	114
Ixud_SoftwareReadbackDO.....	115
Ixud_StartDI.....	116
Ixud_StartDO.....	117
Ixud_GetDIBufferH.....	119
Ixud_StopDI.....	120
Ixud_StopDO.....	121
5.2.3. 中断事件函式集.....	122
Ixud_SetEventCallback.....	122
Ixud_RemoveEventCallback.....	125
Ixud_InstallIrq.....	125
Ixud_RemoveIrq.....	127
5.2.4. 模拟输入函式集.....	128
Ixud_ConfigAI.....	128
Ixud_ConfigAIEx.....	130
Ixud_ClearAIBuffer.....	132
Ixud_GetBufferStatus.....	133
Ixud_ReadAI.....	134
Ixud_ReadAIH.....	135
Ixud_PollingAI.....	136
Ixud_PollingAIH.....	137
Ixud_PollingAIScan.....	138
Ixud_PollingAIScanH.....	140
Ixud_StartAI.....	142
Ixud_StartAIScan.....	144
Ixud_StartExtAI.....	146
Ixud_StartExtAnalogTrigger.....	148
Ixud_StartExtAIScan.....	150
Ixud_GetAIBuffer.....	152
Ixud_GetAIBufferH.....	153

Ixud_StopAI.....	154
5.2.5. 模拟输出函式集.....	155
Ixud_ConfigAO.....	155
Ixud_WriteAOVoltage.....	156
Ixud_WriteAOVoltageH.....	157
Ixud_WriteAOCurrent.....	158
Ixud_WriteAOCurrentH.....	159
Ixud_StartAOVoltage.....	160
Ixud_StartAOVoltageH.....	161
Ixud_StopAO.....	163
5.2.6. 计时计数函式集.....	164
Ixud_DisableCounter.....	164
Ixud_ReadCounter.....	165
Ixud_ReadFrequency.....	166
Ixud_SetCounter.....	167
Ixud_SetFCChannelMode.....	168
5.2.7. 内存输出输入函式集.....	170
Ixud_ReadMemory.....	170
Ixud_WriteMemory.....	171
Ixud_ReadMemory32.....	172
Ixud_WriteMemory32.....	173
5.3. 数据型态.....	174
PIXUD_DEVICE_INFO.....	174
PIXUD_CARD_INFO.....	176

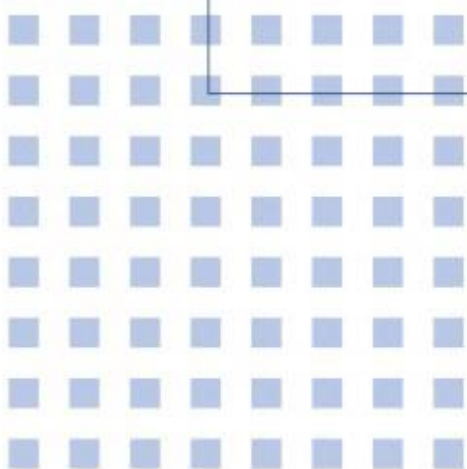
附录 A. 函式回传值与配置码 179

A.1. 函数回传值定义.....	180
A.2. 模块识别号码.....	182
A.3. 配置码定义.....	184
A.3.1. 模拟输入配置码.....	184

A.3.2. 模拟输出配置码(电压).....	187
A.3.3. 模拟输出配置码(电流).....	188
A.3.4. 中断事件配置码	189
A.4. 数字输入端口定义号码.....	190
A.5. 数字输出端口定义号码.....	192
附录 B. 其他.....	194
B.1. 常见问题	195
B.2. 版本修改信息	198

Table Of Contents

Industrial Communication Products



1. 导读

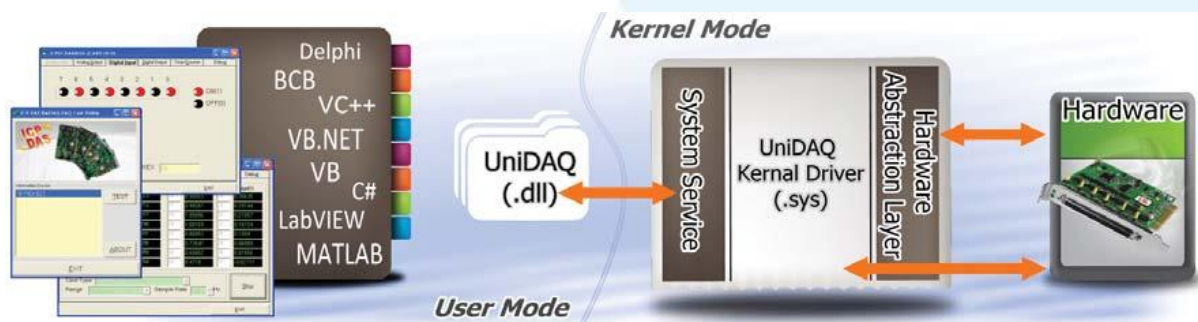
本章节将会简单介绍泓格 UniDAQ 驱动程序库的功能及系统需求

1.1. 关于泓格 UniDAQ 驱动程序

泓格 UniDAQ 驱动函式库提供完整的硬件函式以及最优良效能。在泓格的 UniDAQ 驱动函式库里，不需使用特定的硬件缓存器命令，UniDAQ 提供许多强而有力的函式让泓格板卡的使用者可以在各种编程语言与环境下开发。

泓格 UniDAQ 驱动函式库软件用户直接 I/O 大幅减低 API 对硬件作 I/O 的时间来达到更好的 I/O 速度，另外支持中断及事件通知功能，当硬件中断事件发生时，会透过应用程序通知用户来采取必要的行动，无需手动检查硬件状态，有效的减少程序的复杂度及大幅提升设备的实时可靠性。

泓格 UniDAQ 驱动函式库支持 Windows 2000 之后所有的 32 位及 64 位操作系统，用户就不再需要担心操作系统的兼容性。



1.2. 支持的泓格产品

下表泓格驱动函式库所支持的产品：

型号	型号
PIO-D24/D56/D24U/D56U、PEX-D24/D56	PIO-D48/D48U/D48SU、PEX-D48
PIO-D64/D64U	PIO-D96/D96U/D96SU、PEX-D96S
PIO-D144/D144U/D144LU、PEX-D144LS	PIO-D168/D168U
PCI-D96SU/D128SU	PIO-DA4/DA8/DA16/DA4U/DA8U/DA16U
PISO-DA4U/DA8U/DA16U	PEX-DA4/DA8/DA16
PIO-821L/821H/821LU/821HU	PISO-C64/C64U/P64/P64U
PEX-C64/P64	PISO-A64/A64U/P32A32/P32A32U/ P32A32U-5V、PEX-P32A32
PISO-P32C32/P32C32U/P32C32U-5V/P32S32WU	PEX-P32C32
PISO-P8R8/P8R8U	PISO-P8R8AC/P8R8DC
PISO-P16R16U、PEX-P16R16i/P8R8i	PISO-1730U
PISO-730/730A/730U/730AU、PEX-730/730A	PISO-725/725U
PISO-DA2/DA2U	PISO-813/813U
PCI-TMC12/TMC12A/TMC12AU、PEX-TMC12A	PCI-M128/M256/M512/M512U
PCI-P16R16/P16R16U/P16C16/P16C16U/ P16POR16/P16POR16U/P8R8/P8R8U	PEX-P16POR16i/P8POR8i
PCI-1002L/1002H/1002LU/1002HU	PCI-1202L/1202H/1202LU/1202HU
PEX-1002L/1002H	PEX-1202L/1202H
PCI-1602/1602U,PCI-1602F/1602FU	PCI-1800L/1800H/1800LU/1800HU
PCI-1802L/1802H/1802LU/1802HU	PCI-822LU/826LU
PCI-FC16U	PCI-2602U
PCIe-8620	PCIe-8622

表格 1-1 产品支持列表

1.3. 系统需求

如果您想在计算机上使用泓格驱动程序库，以下是一些系统需求：

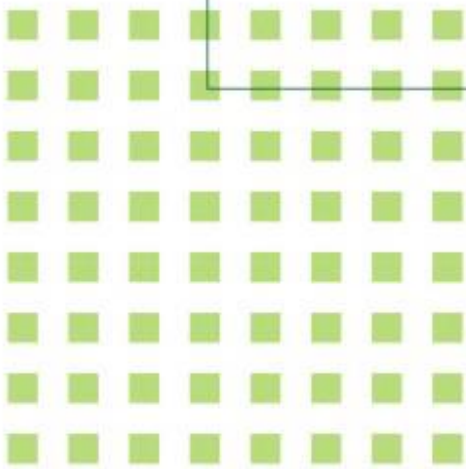
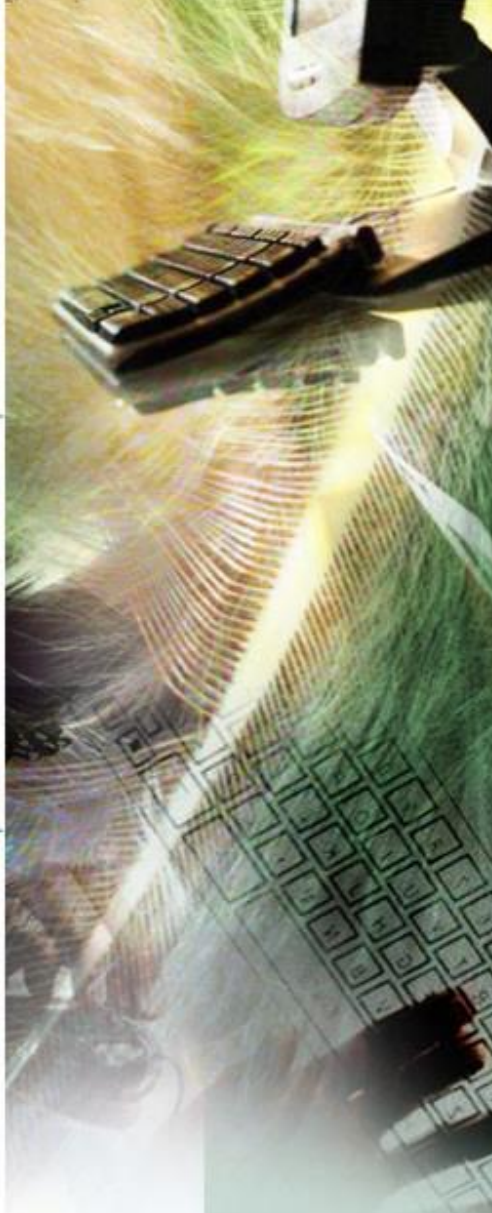
- 使用 266MHz 或更快的 32 位(x86)或 64 位(x64)处理器
- 至少 64 MB 的内存空间
- 兼容于 VGA 的图型显示适配器
- 至少 20 MB 磁盘空间
- 一台相容 DVD/CD-ROM
- Microsoft Windows 2000 以上的 32 位或 64 位操作系统

支持以下的 32 及 64 位的 Windows 操作系统

32 位(x86)	64 位(x64)
Windows 2000	-
Windows XP	Windows XP
Windows Server 2003	Windows Server 2003
Windows Vista	Windows Vista
Windows Server 2008	Windows Server 2008
Windows 7	Windows 7
-	Windows Server 2012
Windows 8/8.1	Windows 8/8.1
Windows 10	Windows 10

表格 1-2 支持操作系统列表

注：不支援 Microsoft Windows 3.1/95/98/ME/NT



2. 开始安装使用

本章节以图解及简易的文字引导用户如何安装及移除驱动程序

2.1. 取得 UniDAQ 驱动函式库安装程序

取得 UniDAQ 驱动函式库安装程序的方法可以从板卡上附的 CD 上取得或是从网络上下载，取得路径请参考下表：



CD:\\ NAPDOS\\PCI\\UniDAQ\\DLL\\Driver



<http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/dll/driver/>



<ftp://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/dll/driver/>

2.2. 安装 UniDAQ 驱动程序函式库

步骤一 安装数据撷取板卡

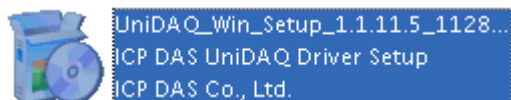
请依照下列步骤安装板卡：

- 1 关掉计算机电源
- 2 打开计算机机壳
- 3 将 I/O 板卡插入至一个未使用的 PCI 或 PCIe 插槽
- 4 装上台壳
- 5 重新启动电源

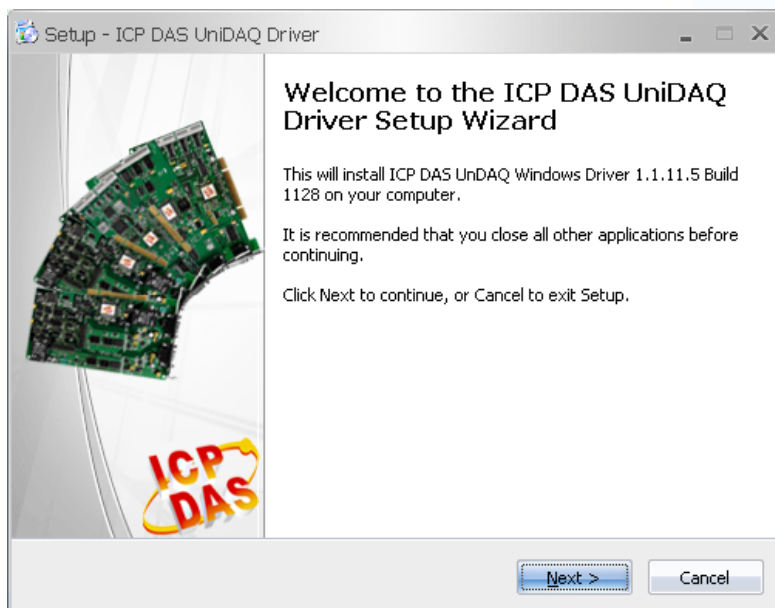
步骤二 安装驱动程序及函式库

请依照下列步骤执行安装

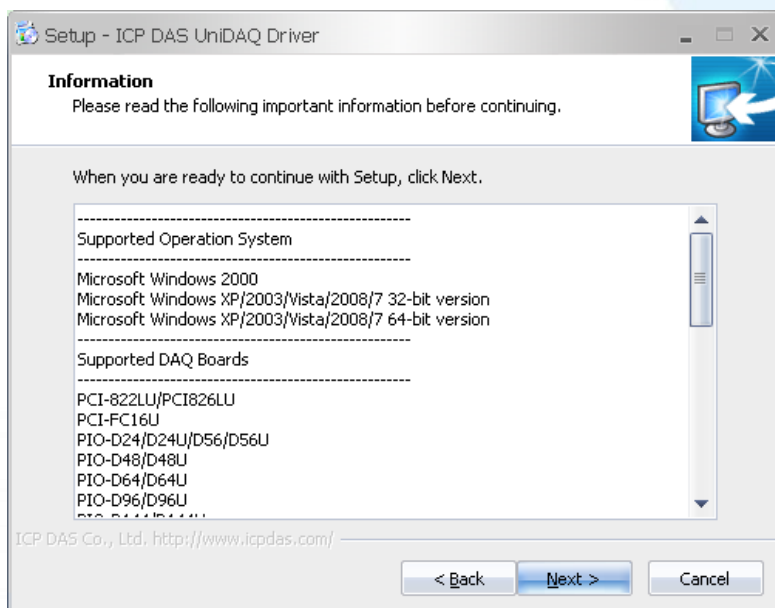
1. 双击 UniDAQ_Win_Setup... 安装驱动函式库。



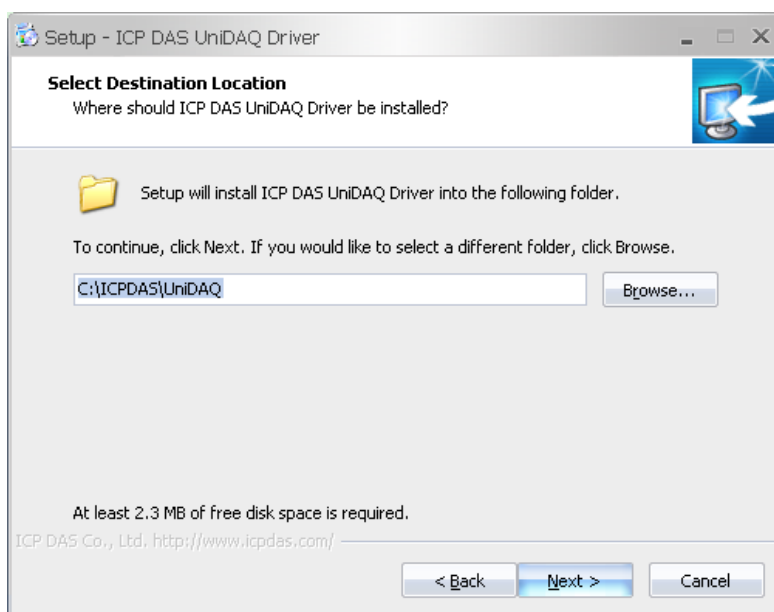
2. 按 **Next>** 到下一个画面。



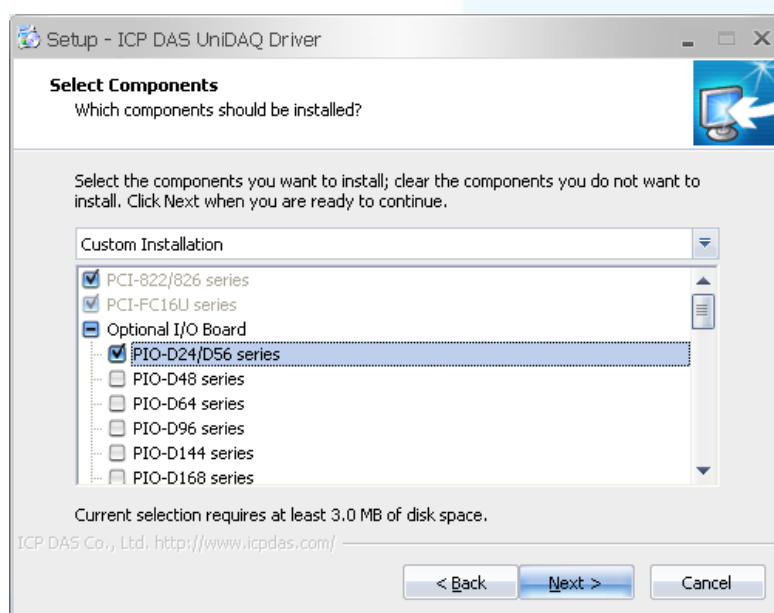
3. 检查您的板卡及系统是否在支持内，按 **Next>** 到下一个画面。



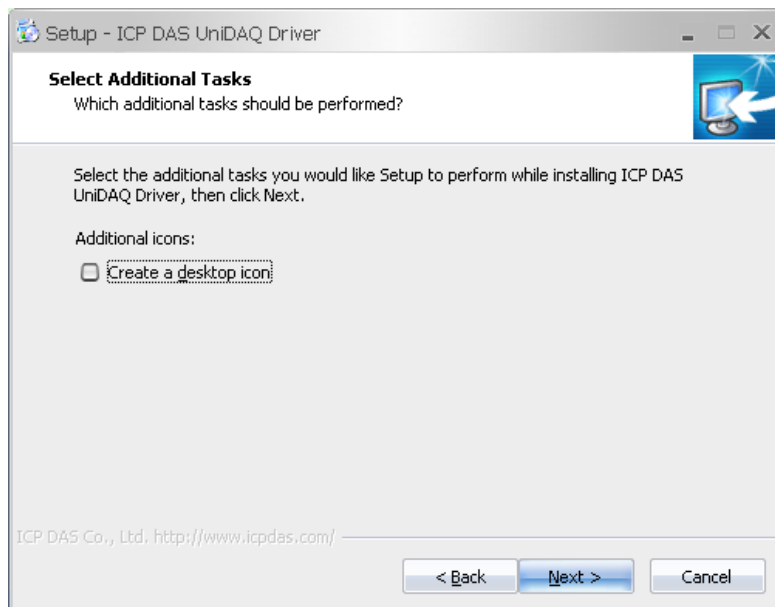
4.选择安装目录，默认为 C:\ICPDAS\UniDAQ，确认后按 **Next>**到下一个画面。



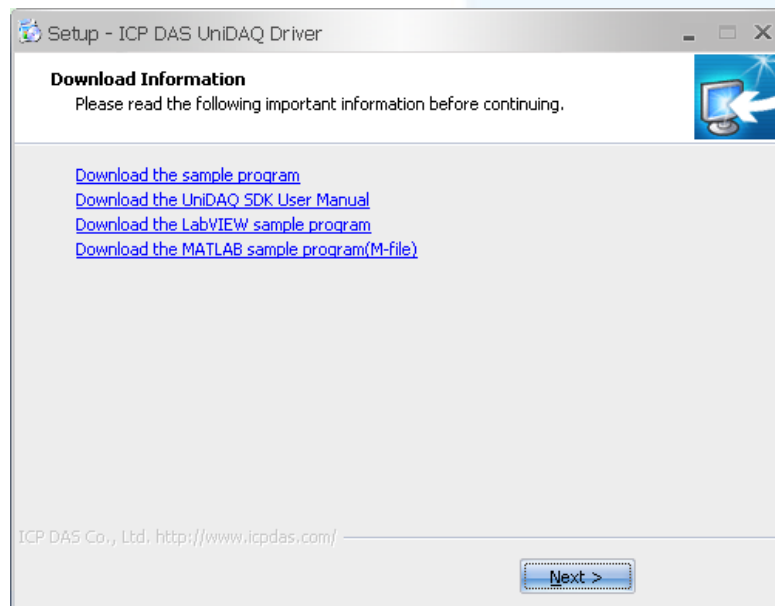
5.在列表内勾选您所需要安装驱动程序的板卡，勾选完后按 **Next>**到下一个画面。



6. 按 Next>到下一个画面。



7. 按 Next>到下一个画面。



8.选择 Yes, restart the computer now 后, 按下 Finish 按键后, 系统会自动重新启动, 在重新启动之后, 泓格 UniDAQ 驱动函数库安装完成。



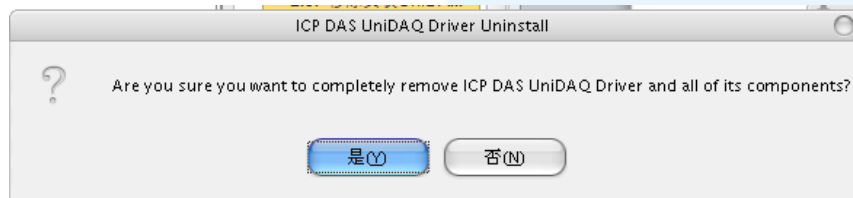
2.3. 移除 UniDAQ 驱动函式库

泓格驱动函式库包括反安装工具来协助您从计算机上移除软件，如果您想要移除软件请完成下列的流程来执行反安装工具。

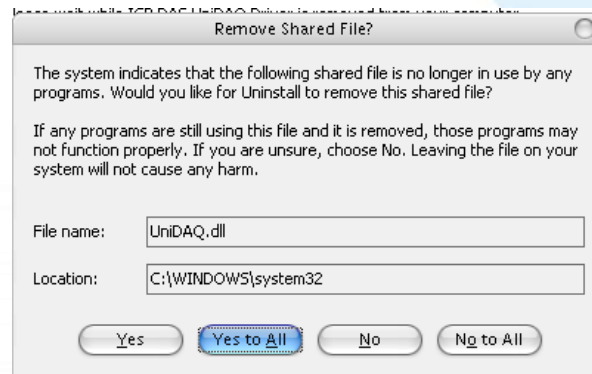
1. 至设定|控制面板|新增或移除程序下。
2. 在选单列表上选择 ICP DAS UniDAQ Windows 项目，并点击最右方的移除按钮。

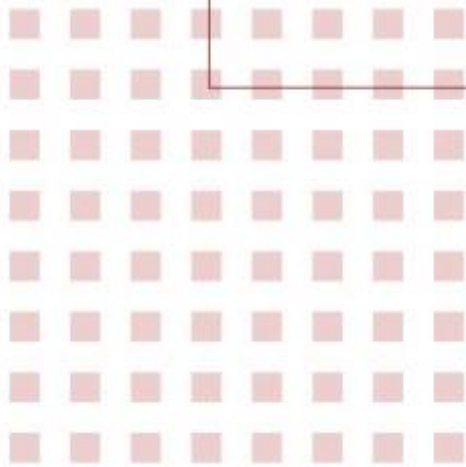


3. 将会跳出一个对话框，并选择是(Y)开始执行反安装。



4. 点击 Yes to All 完全移除 UniDAQ.dll 档案，之后将会完成移除软件的动作。

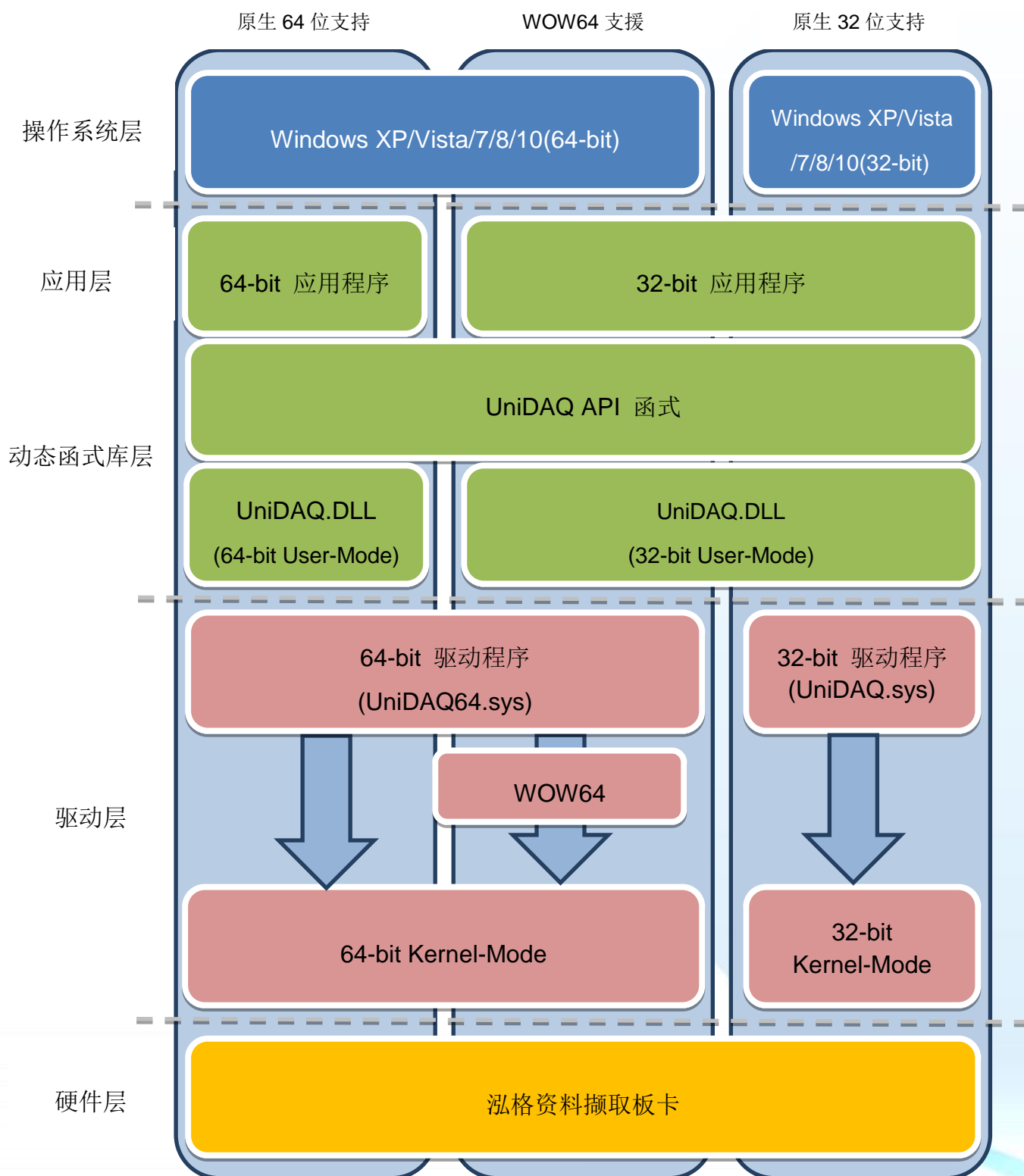




3. 开发指南

引导用户去建构简单应用程序。并且提供在 Win32 Console, VB6, Delphi, BCB, Visual Studio.NET 及 Visual Studio.NET x64 环境下逐步编写程序的范例。

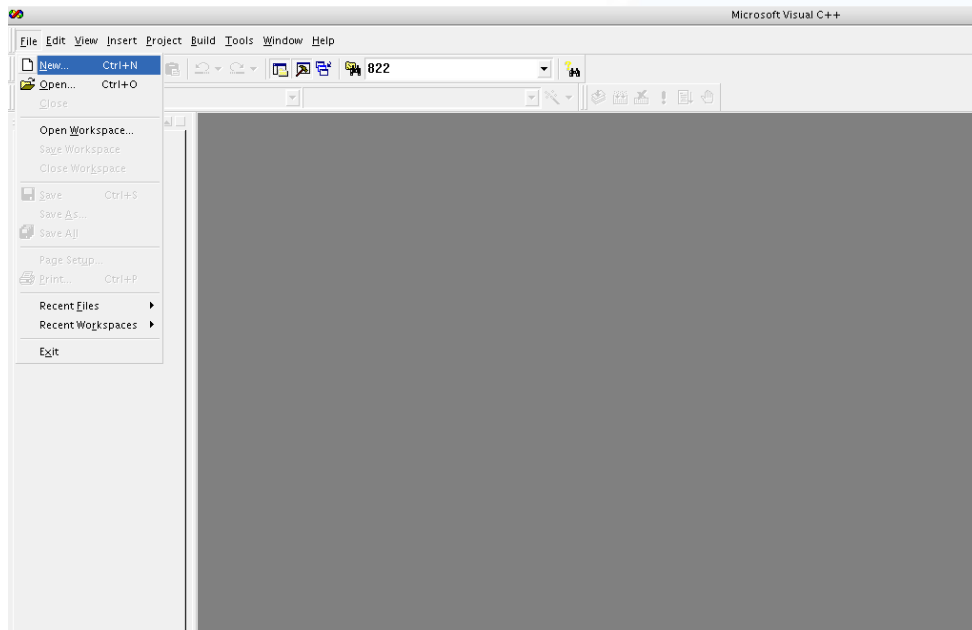
3.1. 应用程序架构



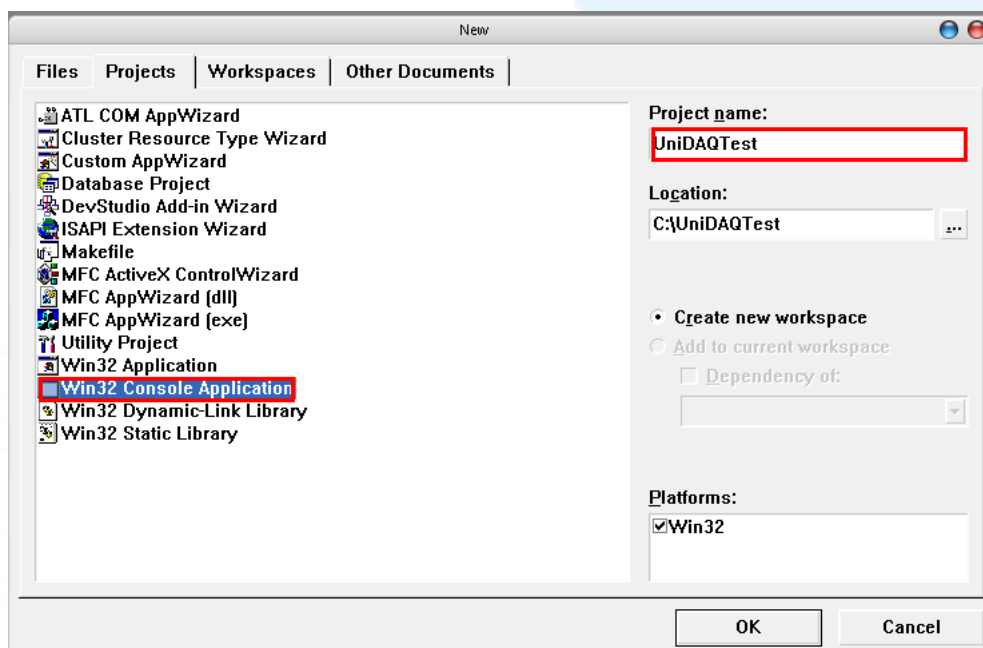
3.2. 在 Win32 Console

步骤 1: 撰写应用程序

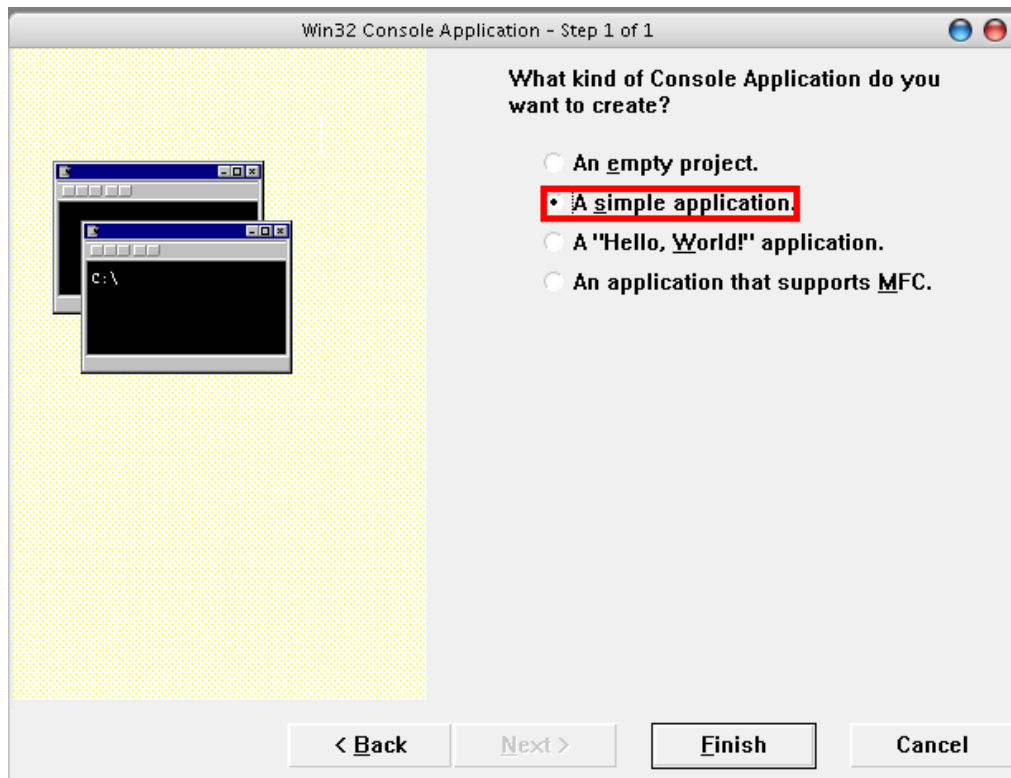
1. 至程序集开启 Microsoft Visual C++ 6.0
2. 从主要选单内选择 File|New...



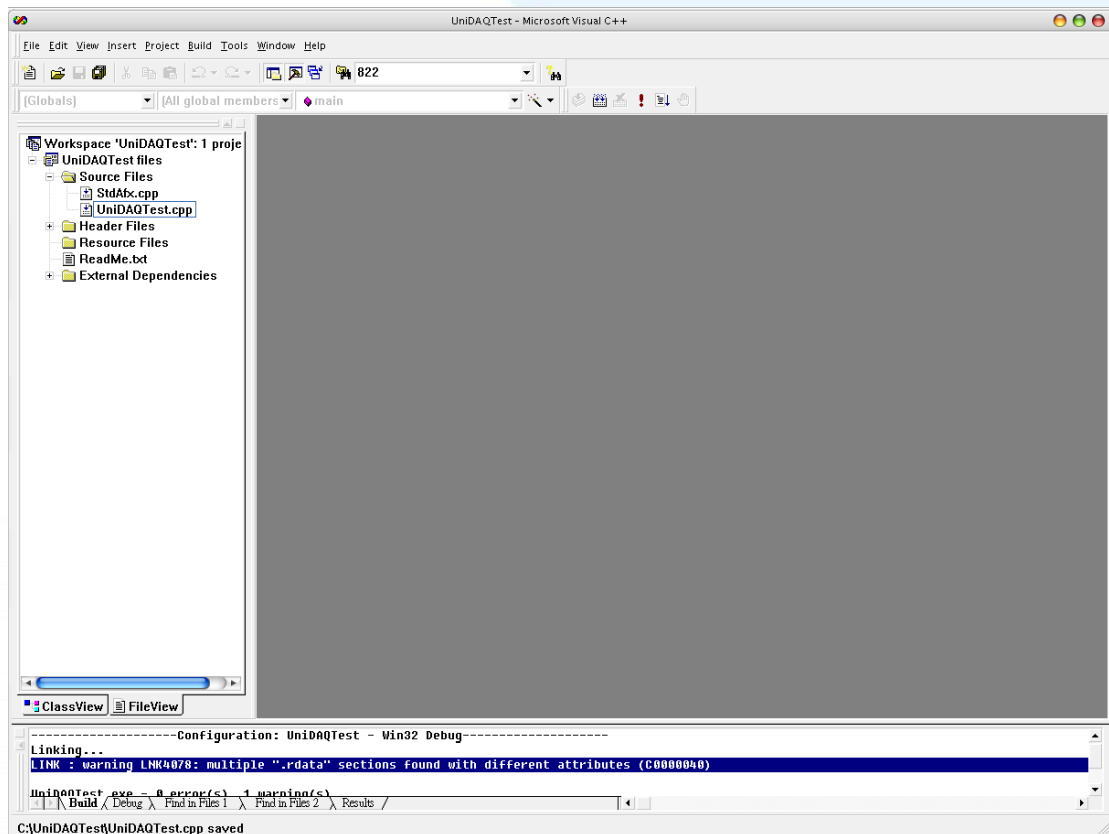
3. 在 dialog box 下的列表点选项目 Win32 Console Application ，并在 Project name 字段输入 UniDAQTest，然后按下 OK 按键。



4. 點選 A simple application 后，按下 Finish 键后，将会产生为使用者产生最基本的程序代码。



5. 双击 UniDAQTest.cpp 开启程序代码写入窗口。



6. 在 UniDAQTest.cpp 填写程序代码如下:

```
#include "stdafx.h"
#include "stdio.h"
#include "UniDAQ.h" //Include the UniDAQ header file
#pragma comment(lib,"UniDAQ.lib") //Include the UniDAQ library file

WORD wRtn;
WORD wBoardNo;
WORD wTotalBoards;

int main(int argc, char* argv[])
{
    WORD wOutPortNo;

    //Initial the resource and get total board number form Driver
    wRtn=Ixud_DriverInit(&wTotalBoards);
    if (wRtn!=Ixud_NoErr)
    {
        printf("\nDriver Init Error(%d)",wRtn);
        return wRtn;
    }
    printf("Write DO Value 0xFF");
    wBoardNo=0;
    wOutPortNo=0;
    //Write DO
    wRtn = Ixud_WriteDO(wBoardNo,wOutPortNo,0xFF);
    //Release the resource from driver
    wRtn = Ixud_DriverClose();
    return 0;
}
```

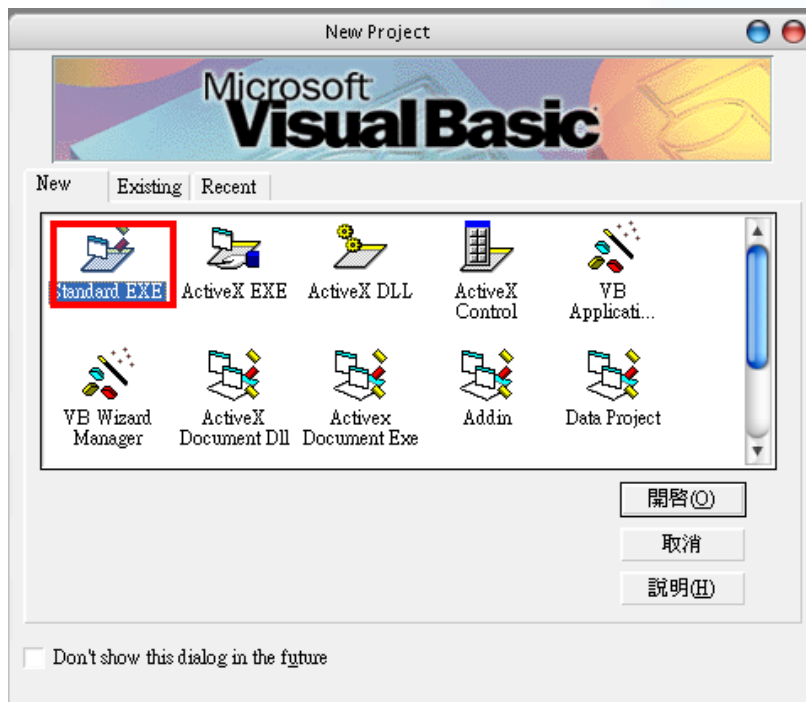
步骤 2:测试应用程序

1. 在 Build 选单点击 Compiler 来编译程序代码。
2. 立即在 DOS Box 下执行程序。

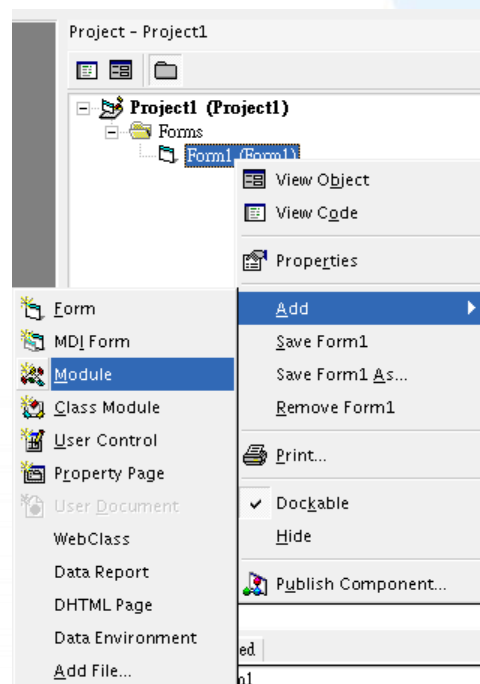
3.3. 在 Visual Basic 6.0

步骤 1: 撰写应用程序

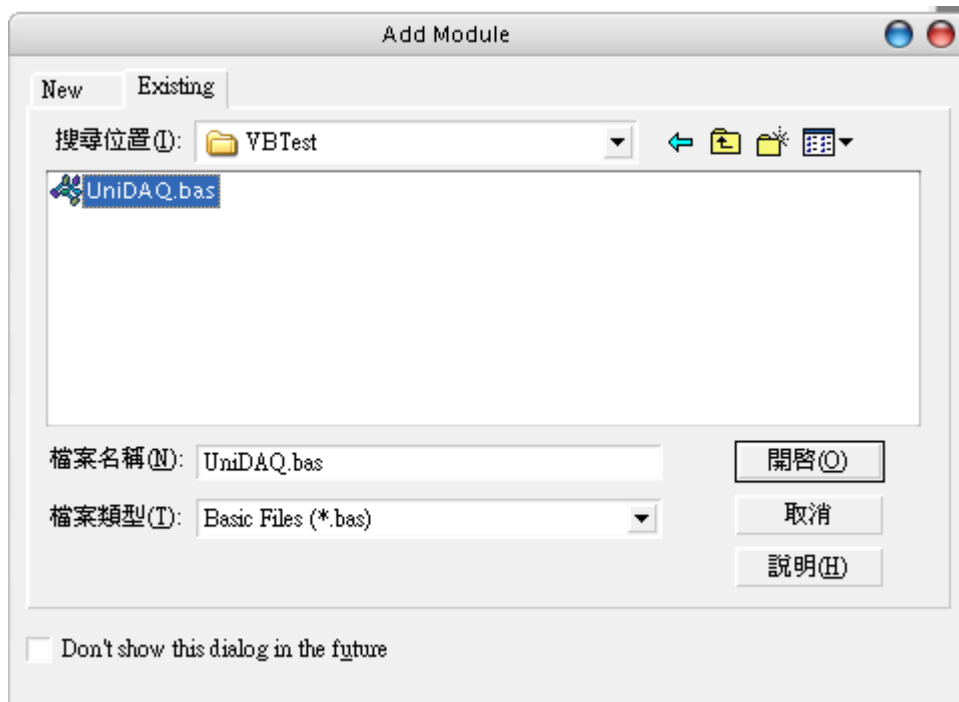
1. 至程序集开启 Microsoft Visual Basic 6.0
2. 选择 Standard EXE 图标并按下开启按钮后将建立一个新的项目。



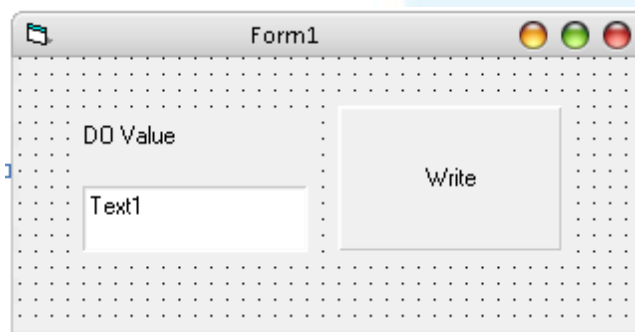
3. 在 Project explorer 开启 Add Module 窗口。



4. 在 Add Module 选择 Existing 后添加宣告档 UniDAQ.bas 至专案里。



5. 设计窗口，在 form1 放置一个 Label 控件并在 Caption 属性上输入 DO Value。接着放置 TextBox 控件，并切换至属性窗口上至 Name 属性输入 txtDOVal，最后放置一个 CommandButton 控件，并修改 Name 属性为 cmdWrite 及在 Caption 属性上输入 Write。



6. 在 cmdWrite 填写程序代码如下：

```
Option Explicit
Dim wTotalBoards As Integer
Dim wBoardNo As Integer
Dim wOutPortNo As Integer
Dim wRtn As Integer

Private Sub cmdWrite_Click()

Dim wBoardIndex As Integer

'//Initial the resource and get total board number form Driver
wRtn = Ixud_DriverInit(wTotalBoards)
If (wRtn) Then
MsgBox ("Driver Initial Error!!Error Code:" + Str(wRtn))
End
End If

wBoardNo:=0;
wOutputportNo =0;

'//Write DO
wRtn = Ixud_WriteDO(wBoardNo, wOutPortNo, Val(txtDOVal.Text))

'//Release the resource form Driver
wRtn = Ixud_DriverClose()
End Sub
```

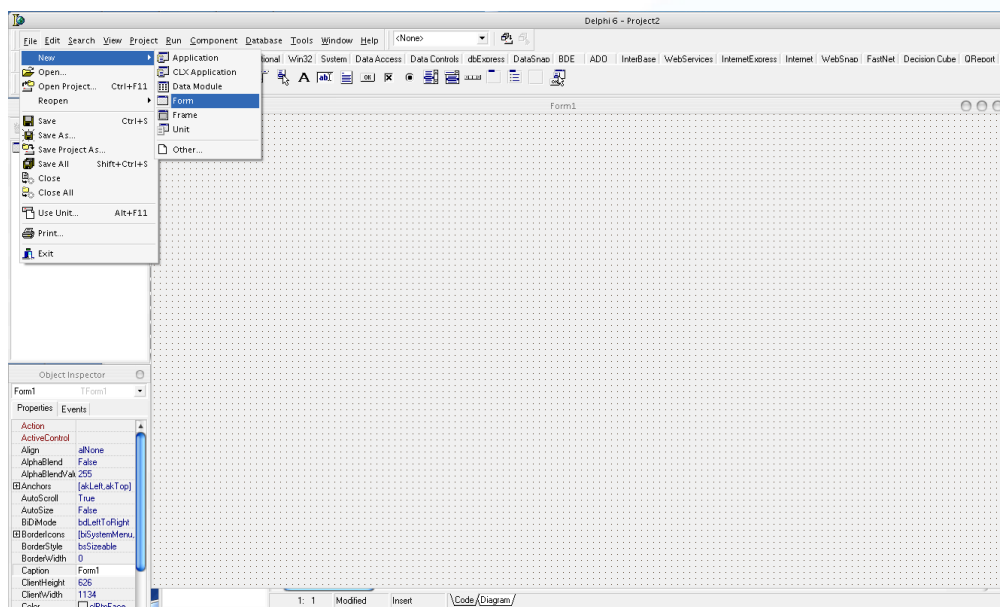
步骤 2:测试应用程序

1. 按下 **F5** 来执行程序。
2. 在 **DO Value** 字段输入数值 **255**。
3. 并按下 **Write** 按键，输出 **DO** 数值 **255**。

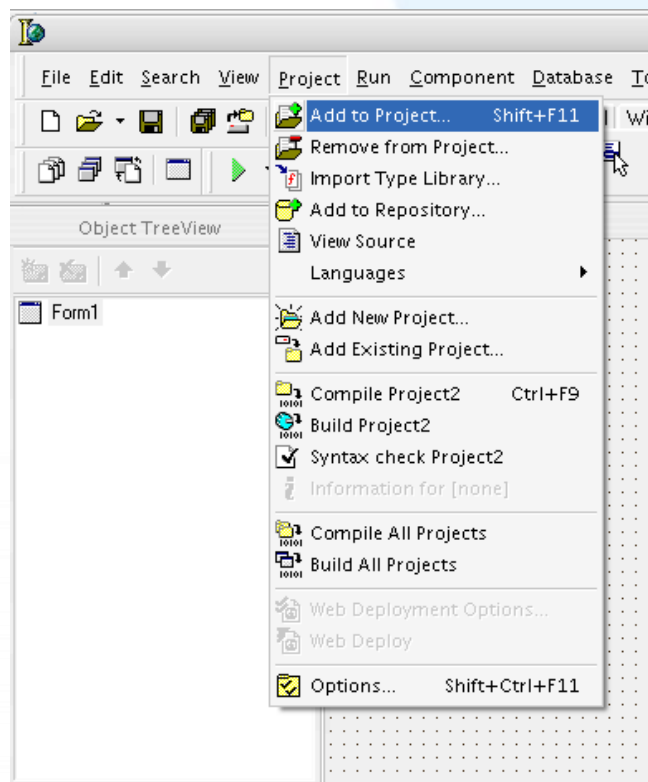
3.4. 在 Borland Delphi

步骤 1: 撰写应用程序

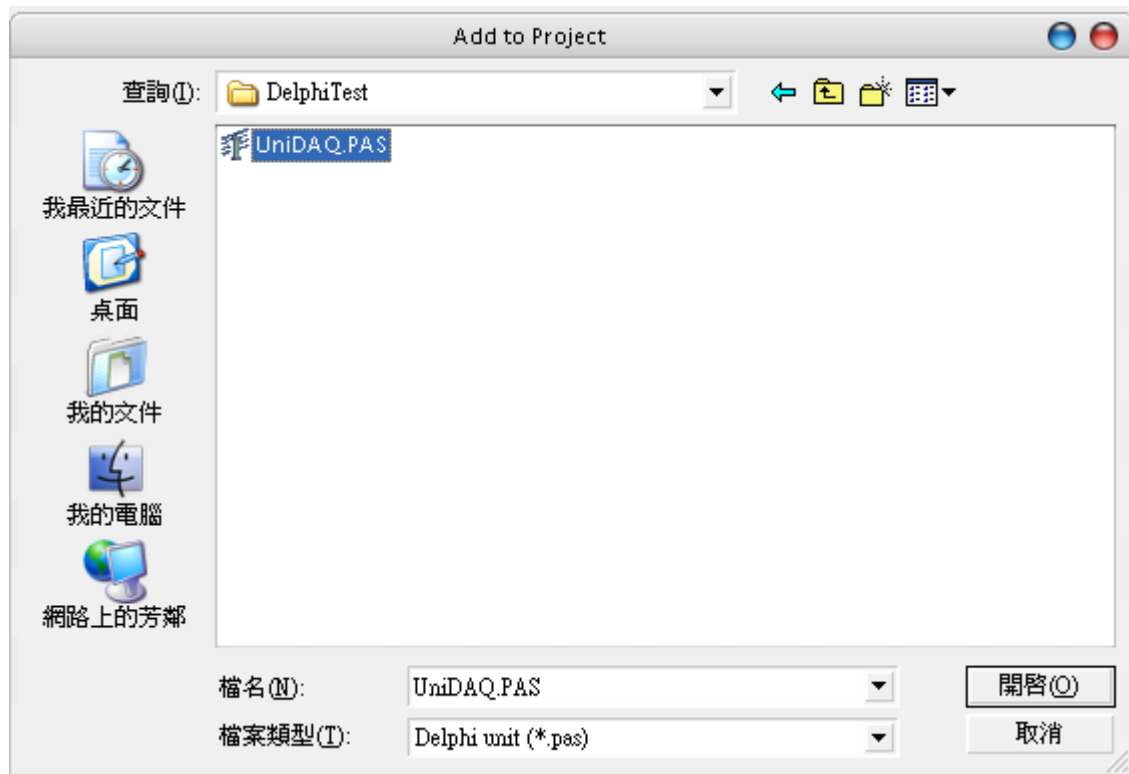
1. 至程序集开启 Delphi 6.0
2. 从主要选单内选择 New|Form 将建立一个新的项目。



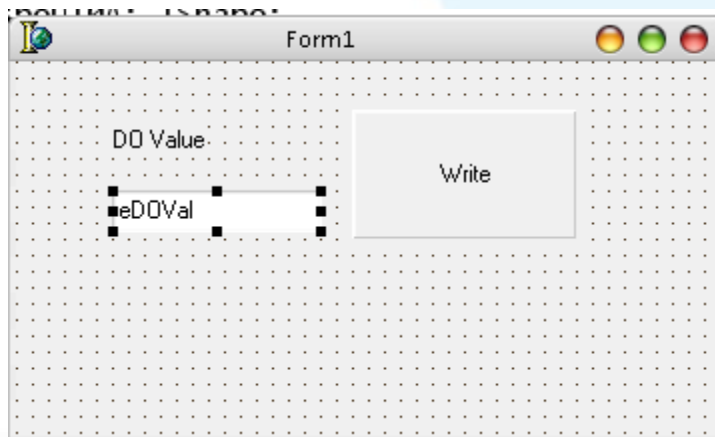
3. 在主要选单内选择 Project|Add to Project 开启一窗口。



4. 在 Add to Project 窗口移至 UniDAQ.pas 的路径下选择 UniDAQ.pas 按下开启(O) 添加宣告檔至专案里。



5. 设计窗口，在 Form1 放置一个 Label 控件并在 Caption 属性上输入 DO Value。接着放置 Edit 控件，并切换至属性窗口上至 Name 属性输入 eDOVal，最后放置一个 Button 控件，并修改 Name 属性为 btnWrite 及在 Caption 属性上输入 Write。



6. 双击 **Form1** 上的 **btnWrite** 按键控件，进入程序代码编辑窗口，在 **implementation** 下加入程序代码如下。

```
implementation
uses UniDAQ;

{$R *.dfm}

procedure TForm1.btnWriteClick(Sender: TObject);
var
    wTotalBoards,wRtn,wBoardNo,wOutputNo:Word;
    dwDOValue : LongInt;
begin
    //Initial resource and get total board number from driver
    wRtn := Ixud_DriverInit(wTotalBoards);
    If wRtn <> Ixud_NoErr Then
    begin
        Application.MessageBox('*** DriverInit Error! ***', 'Error' , IDOK);
        Exit;
    End;
    wBoardNo :=0;
    wOutputNo :=0;

    //Write DO
    wRtn:=Ixud_WriteDO(wBoardNo,wOutputNo,StrToInt(eDOVal.Text));

    //Release the resource from driver
    wRtn := Ixud_DriverClose;

end;

end.
```

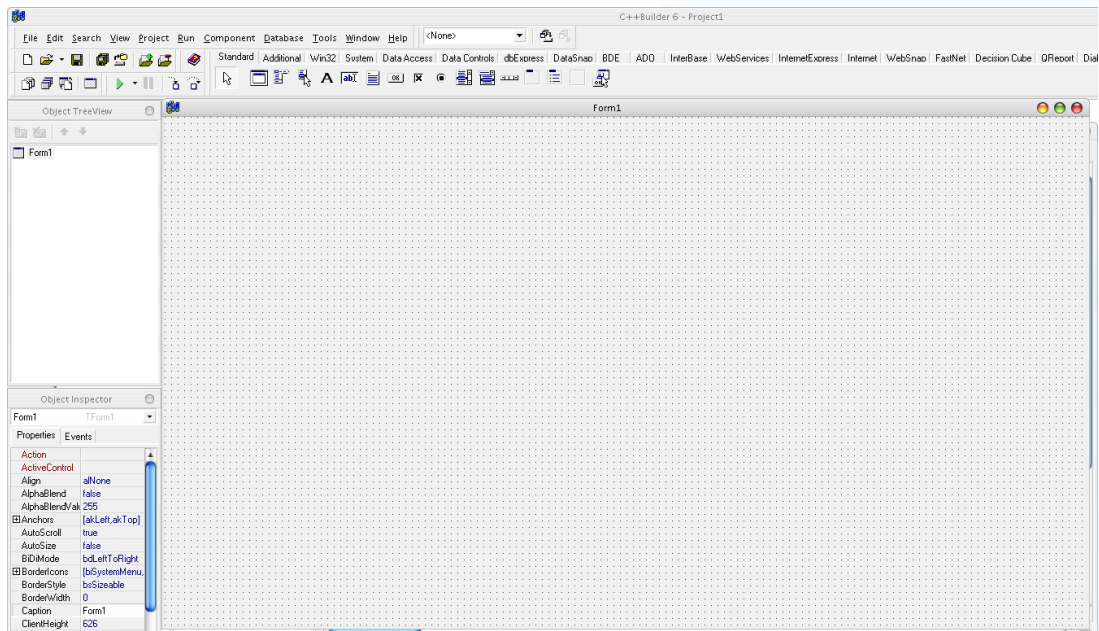
步骤 2:测试应用程序

1. 按下 **F9** 来执行程序。
2. 在 **DO Value** 字段输入数值 **255**。
3. 并按下 **Write** 按键，输出 **DO** 数值 **255**。

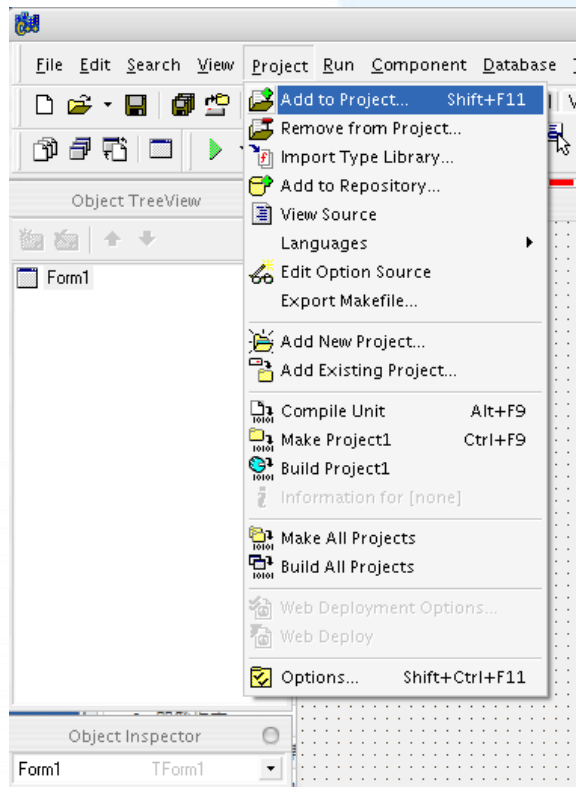
3.5. 在 Borland C++ Builder

步骤 1: 撰写应用程序

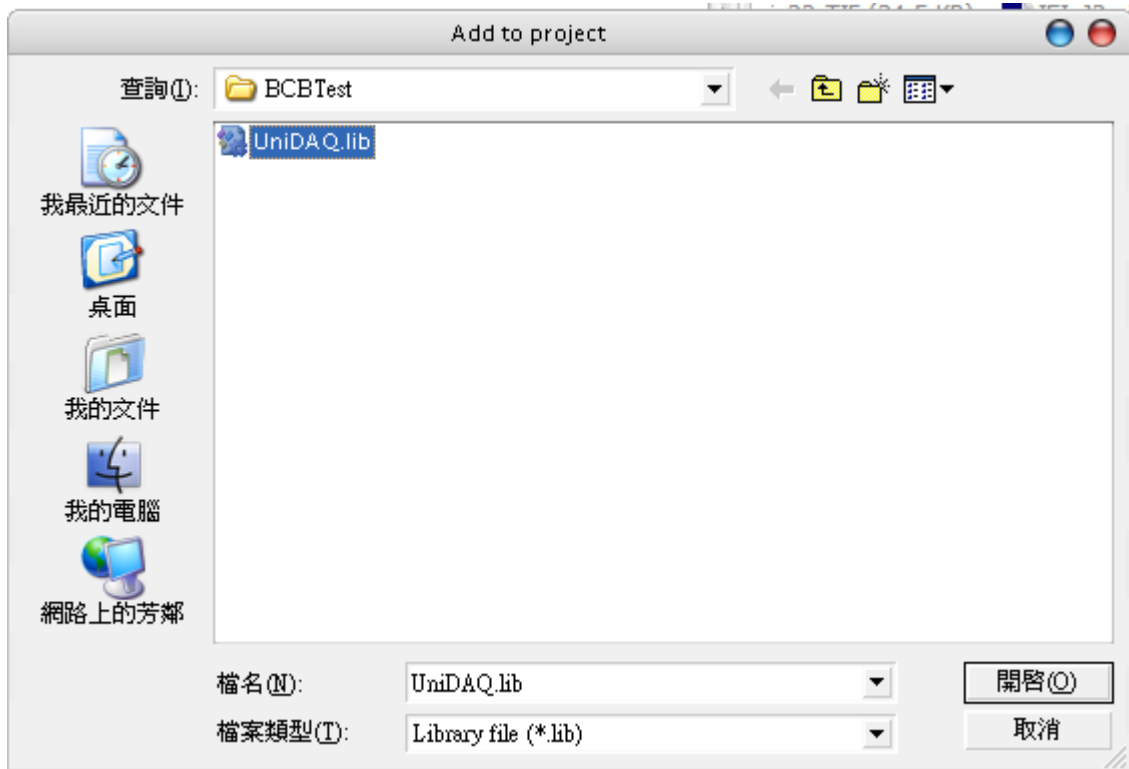
1. 至程序集开启 C++ Builder 6
2. 从主要选单内选择 New|Form 将建立一个新的项目。



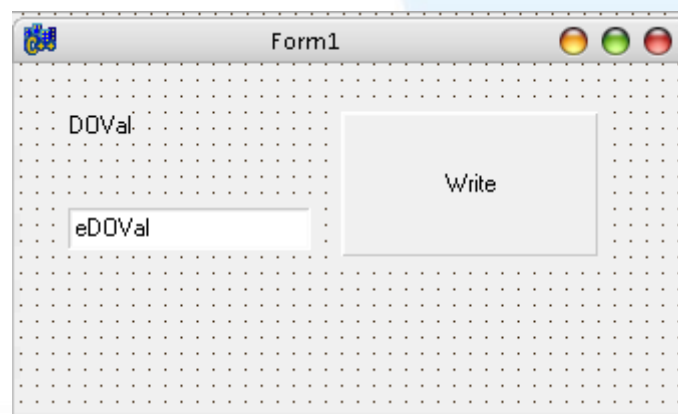
3. 在主要选单内选择 Project|Add to Project 开启一窗口。



4. 在 Add to Project 窗口移至 UniDAQ.lib 的路径下选择 UniDAQ.lib 按下开启(O)添加宣告档至专案里。



5. 设计窗口，在 Form1 放置一个 Label 控件并在 Caption 属性上输入 DO Value。接着放置 Edit 控件，并切换至属性窗口上至 Name 属性输入 eDOVal，最后放置一个 Button 控件，并修改 Name 属性为 btnWrite 及在 Caption 属性上输入 Write。



6. 双击 **Form1** 上的 **btnWrite** 按键控件，进入程序代码编辑窗口，加入程序代码如下。

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "UniDAQ.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
void __fastcall TForm1::btnWriteClick(TObject *Sender)
{
    Word wTotalBoard, wRtn ;
    Word wOutPortNo;
    Word wBoardNo;
    //Initial the resource and get the total board number from driver
    wRtn = Ixud_DriverInit(&wTotalBoard);
    if ( wRtn != Ixud_NoErr )
    {
        ShowMessage( "Driver Initial Err!!Error Code:" + IntToStr(wRtn)) ;
    }
    wOutPortNo=0;
    wBoardNo=0;
    wRtn=Ixud_WriteDO(wBoardNo,wOutPortNo,StrToInt(eDOVal ->Text));

    //Release the resource from driver
    wRtn= Ixud_DriverClose();
}
```

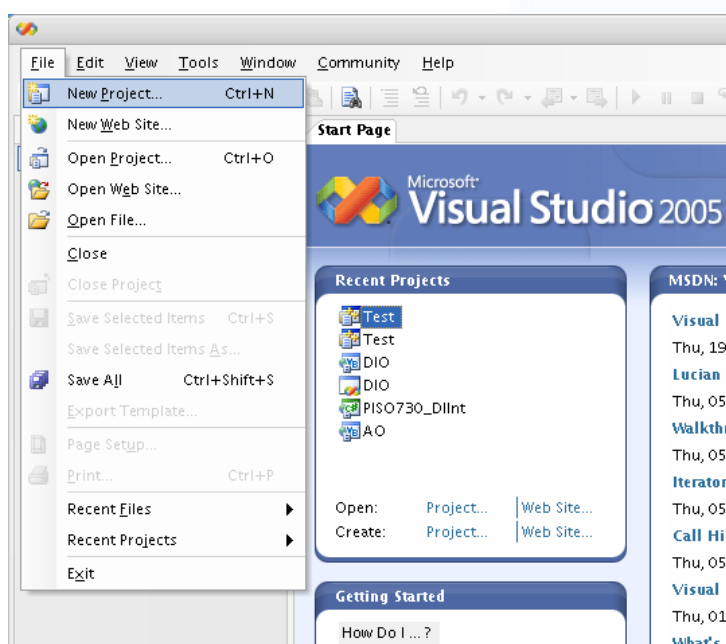
步骤 2:测试应用程序

1. 按下 **F9** 来执行程序。
2. 在 **DO Value** 字段输入数值 **255**。
3. 并按下 **Write** 按键，输出 **DO** 数值 **255**。

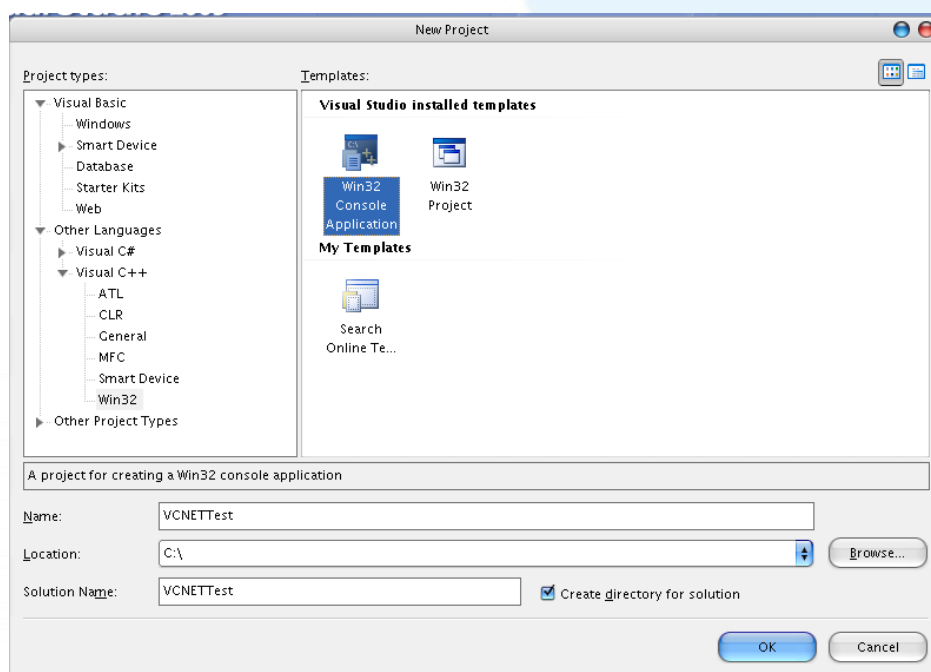
3.6. 在 Visual C++.NET

步骤 1: 撰写应用程序

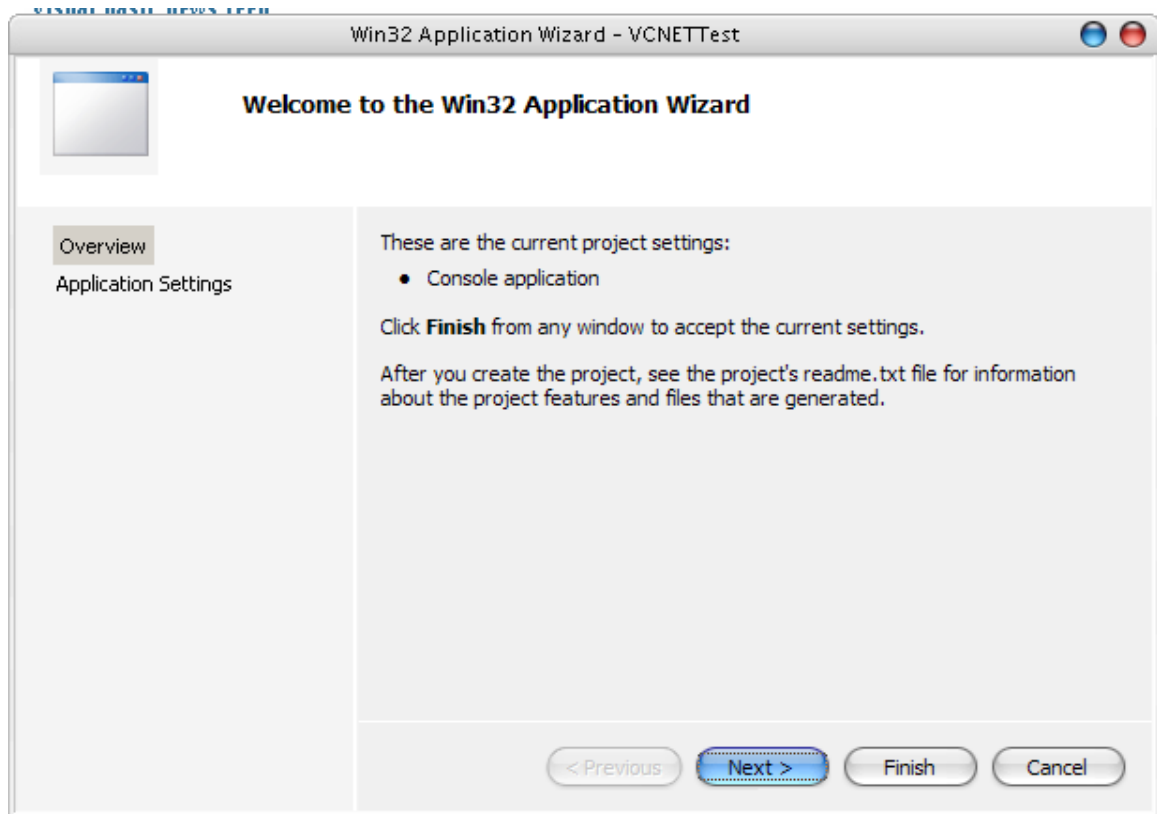
1. 至程序集开启 Microsoft Visual Studio 2005
2. 从主要选单内选择 File|New Project...



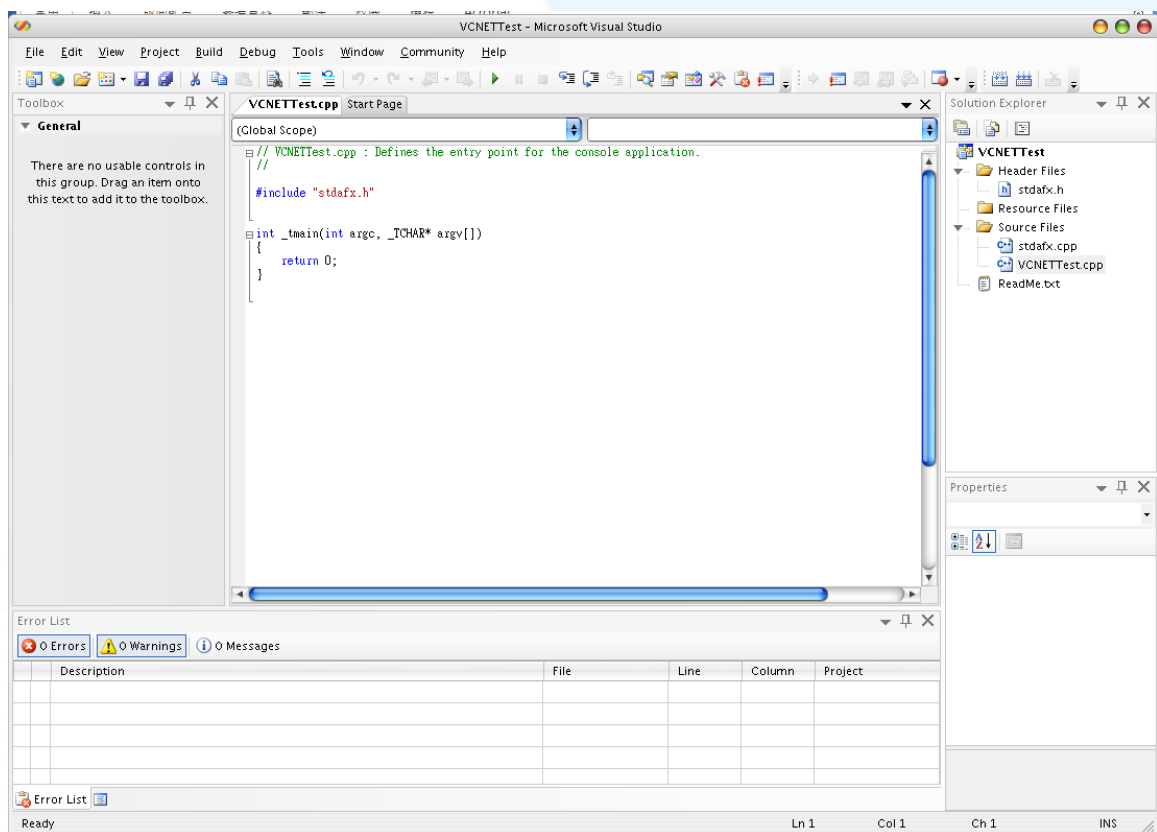
3. 在 Project type 下的列表点选项目 Visual C++ 并在展开选单内选择 Win32, 然后再右方 Templates 表框内选择 Win32 Console Application, 接下来至下方的 Name 键入项目名称 VCNETTest 然后按下 OK 按键。



4. 按下 **Finish** 键后，将会产生为使用者产生最基本的程序代码。



5. 双击 **VCNETTest.cpp** 开启程序代码写入窗口。



6. 在 VCNETTest.cpp 填写程序代码如下:

```
// VCNETTest.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "stdio.h"
#include "UniDAQ.h"
#pragma comment(lib,"UniDAQ.lib")

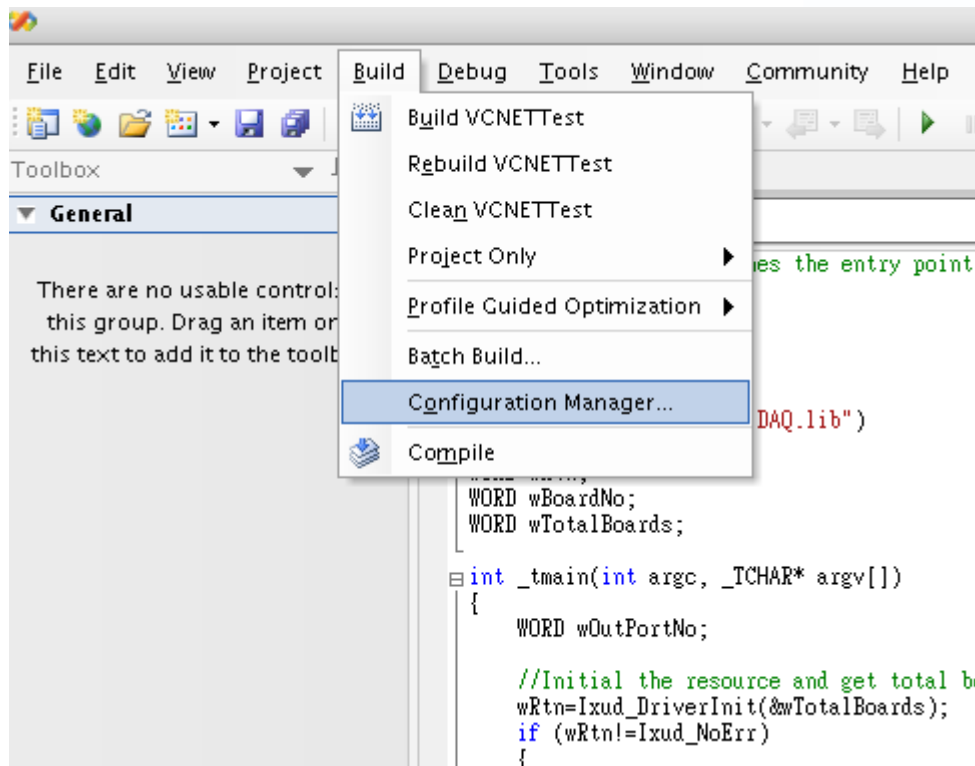
WORD wRtn;
WORD wBoardNo;
WORD wTotalBoards;

int _tmain(int argc, _TCHAR* argv[])
{
    WORD wOutPortNo;

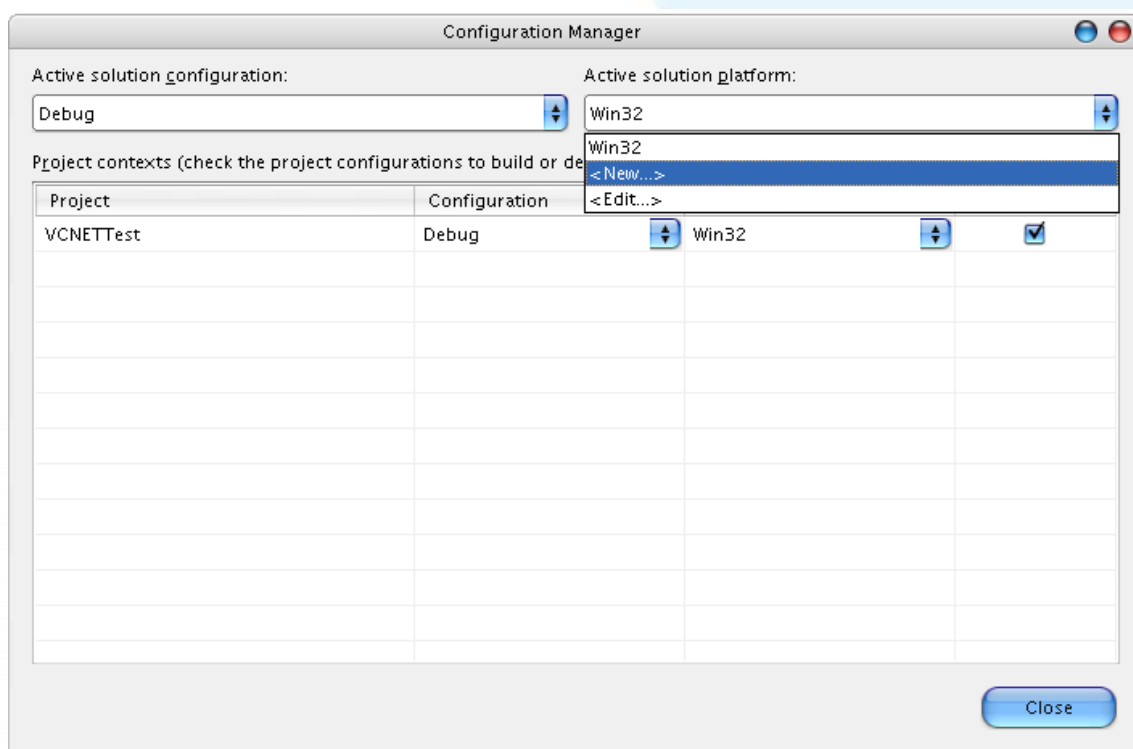
    //Initial the resource and get total board number form Driver
    wRtn=Ixud_DriverInit(&wTotalBoards);
    if (wRtn!=Ixud_NoErr)
    {
        printf("\nDriver Init Error(%d)",wRtn);
        return wRtn;
    }
    printf("Write DO Value 0xFF");
    wBoardNo=0;
    wOutPortNo=0;
    //Write DO
    wRtn = Ixud_WriteDO(wBoardNo,wOutPortNo,0xFF);
    //Release the resource from driver
    wRtn = Ixud_DriverClose();
    return 0;
}
```

步骤 2:编译应用程序

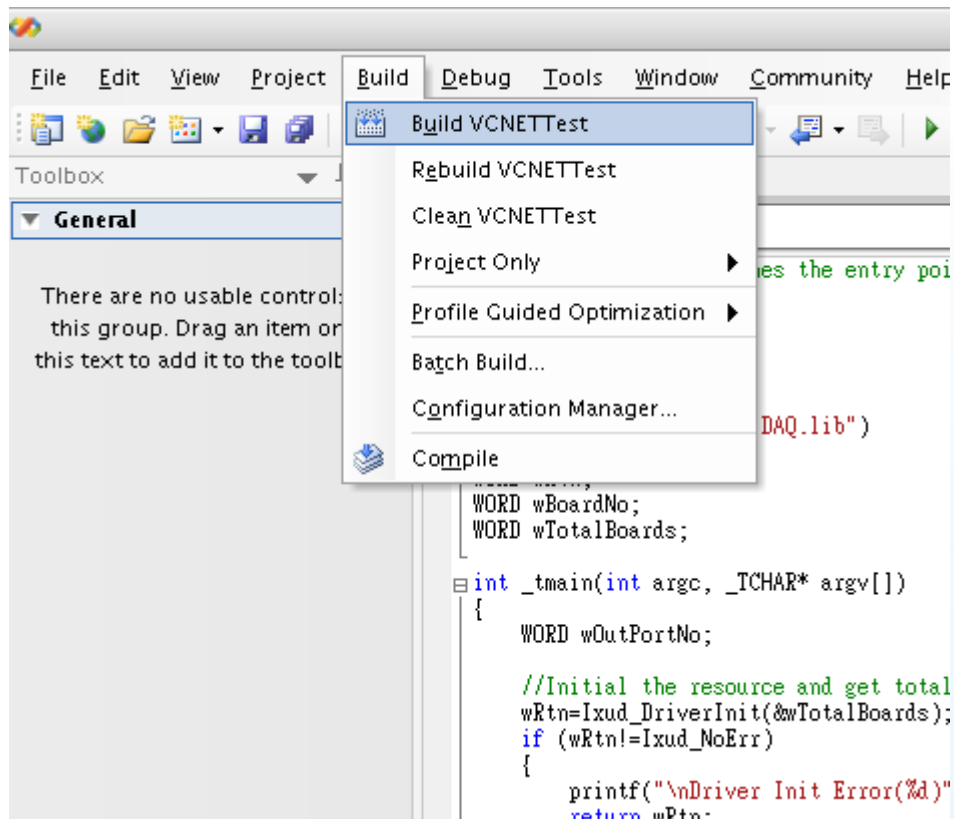
1. 在 Build 选单点击 Configuration Manager。



2. 在 Configuration Manager 点击 Active solution Platform 下拉选单选择<New...>。



5. 从主要选单内选择 Build|Build VCNETTest 开始编译应用程序。



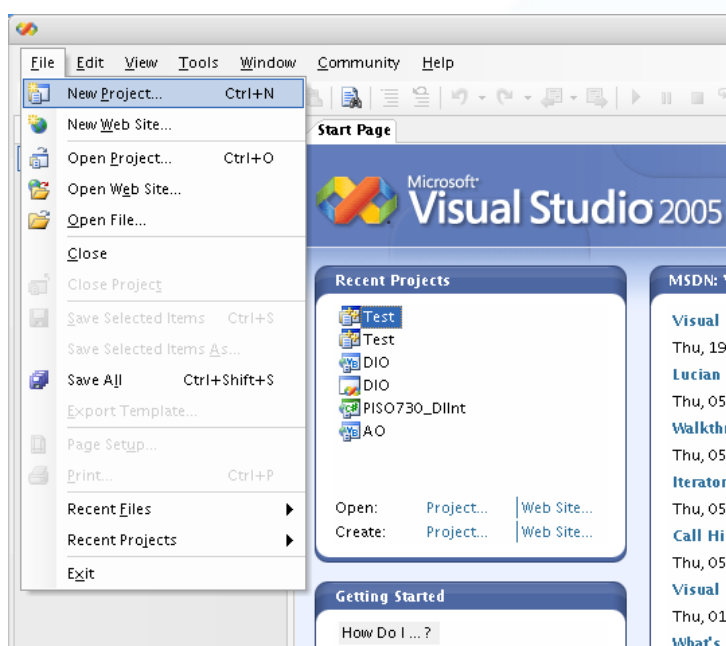
步骤 3:测试应用程序

立即在 DOS Box 下执行程序。

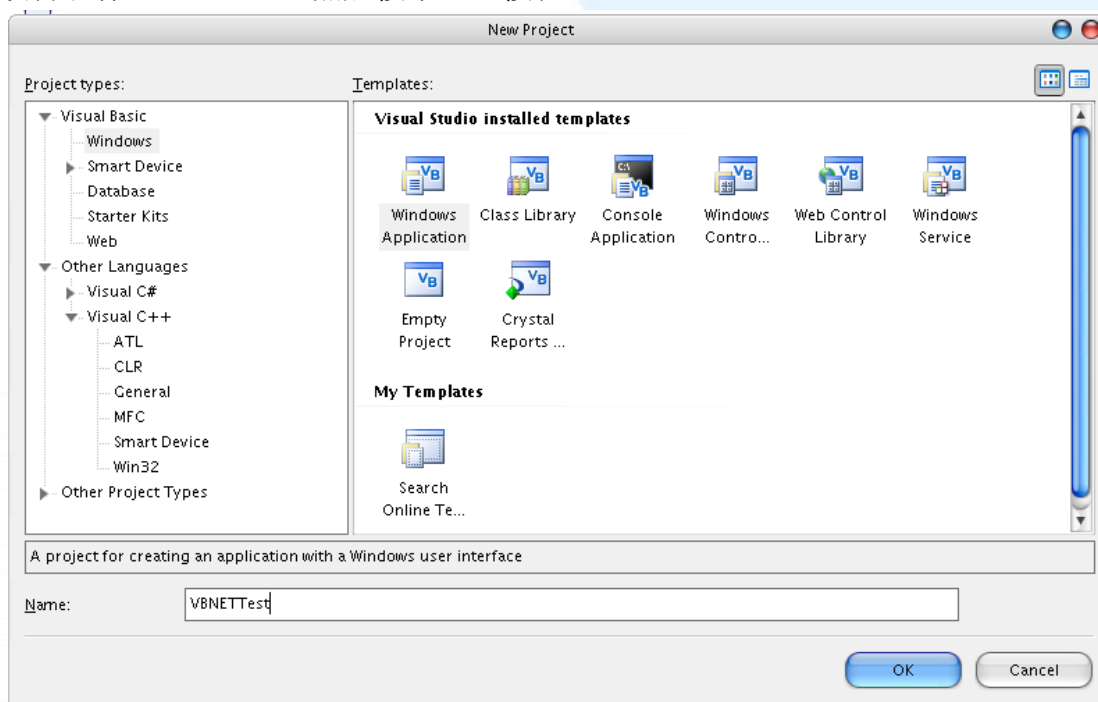
3.7. 在 Visual Basic.NET

步骤 1: 撰写应用程序

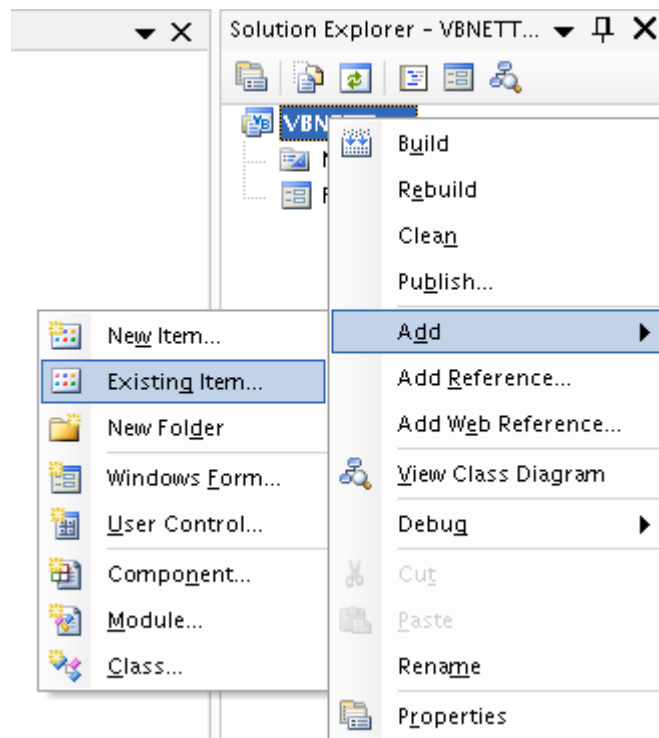
1. 至程序集开启 Microsoft Visual Studio 2005
2. 从主要选单内选择 File|New Project...



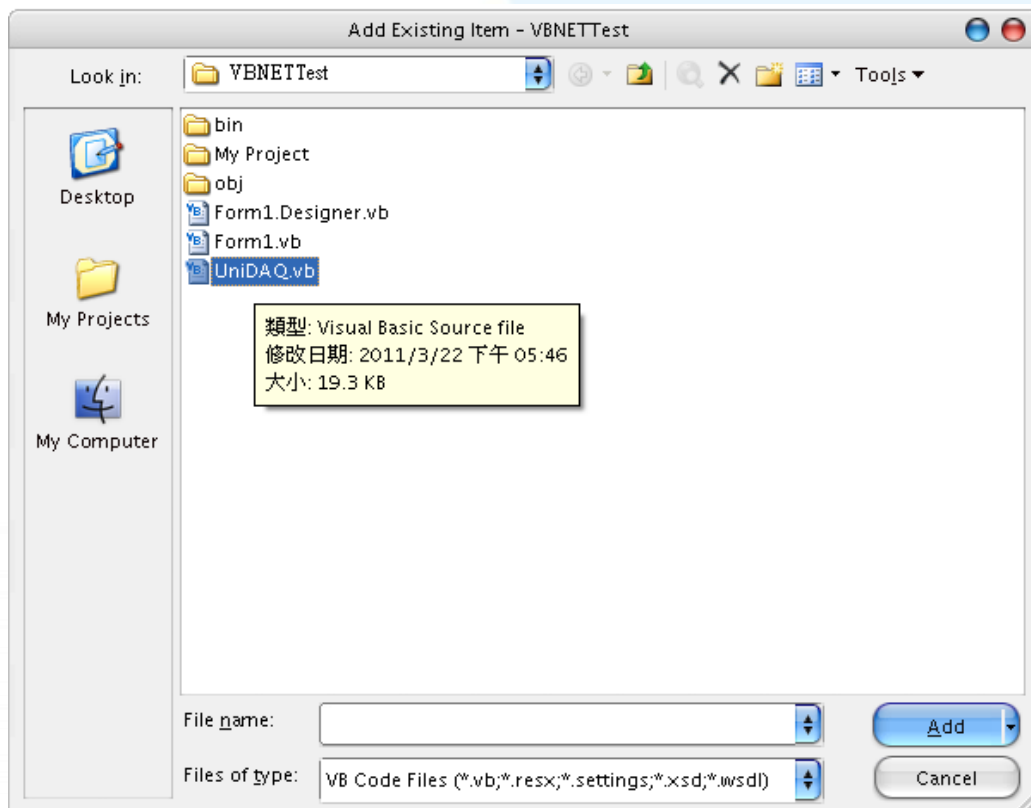
3. 在 Project type 下的列表点选项目 Visual Basic 并在展开选单内选择 Windows，然后再右方 Templates 表框内选择 Windows Application，接下来至下方的 Name 键入项目名称 VBNETTest 然后按下 OK 按键。



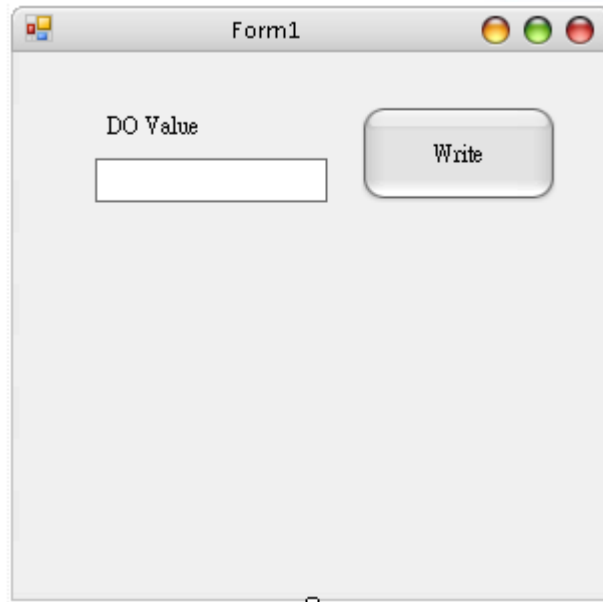
4. 在 Solution Explorer 开启 Add Existion item 窗口。



5. 在 Add Existion item 选择 UniDAQ.vb 后按 Add 按钮添加宣告文件至专案里。



6. 设计窗口，在 Form1 放置一个 Label 控件并在 Text 属性上输入 DO Value。接着放置 TextBox 控件，并切换至属性窗口上至 Name 属性输入 txtDOVal，最后放置一个 Button 控件，并修改 Name 属性为 btnWrite 及在 Text 属性上输入 Write。



7. 在 btnWrite 填写程序代码如下：

```
Private Sub btnWrite_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnWrite.Click
    Dim wTotalBoards As UInteger
    Dim wBoardNo As UInteger
    Dim wOutPortNo As UInteger
    Dim wRtn As UInteger

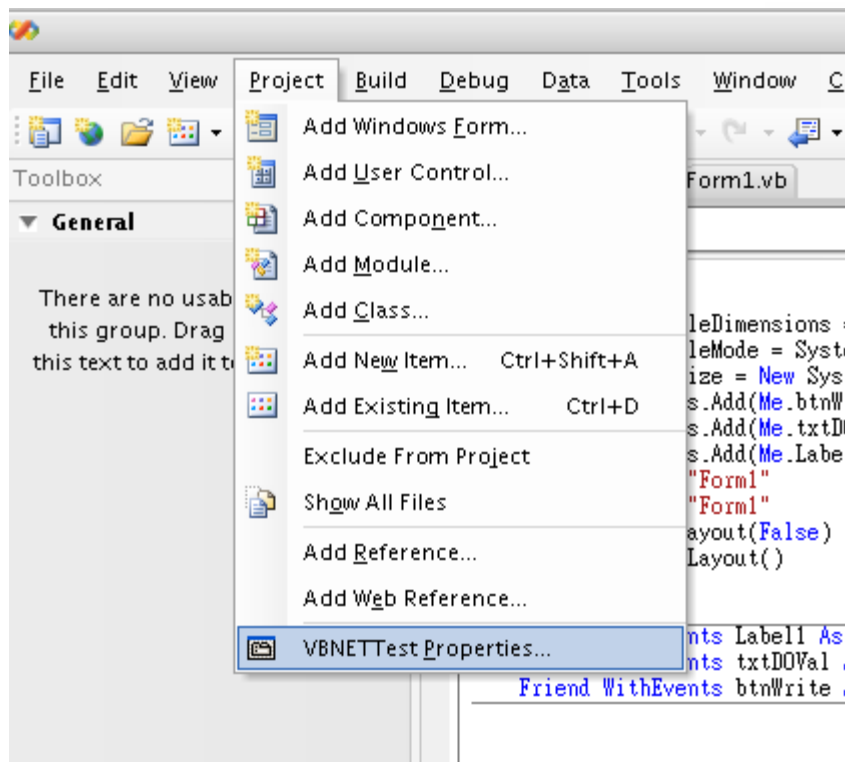
    '//Driver Initial
    wRtn = Ixud_DriverInit(wTotalBoards)
    If (wRtn) Then
        MsgBox("Driver Initial Error!!Error Code:" + Str(wRtn))
    End
End If

    '//Write DO
    wRtn = Ixud_WriteDO(wBoardNo, wOutPortNo, Val(txtDOVal.Text))

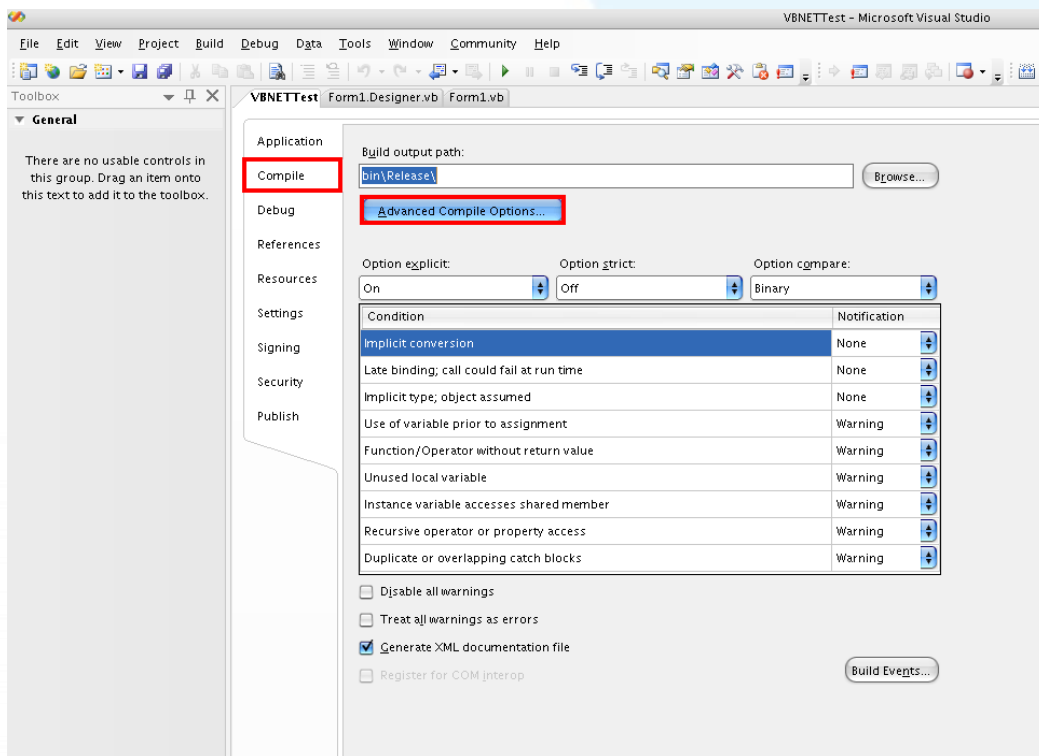
    wRtn = Ixud_DriverClose()
End Sub
```

步骤 2: 编译应用程序

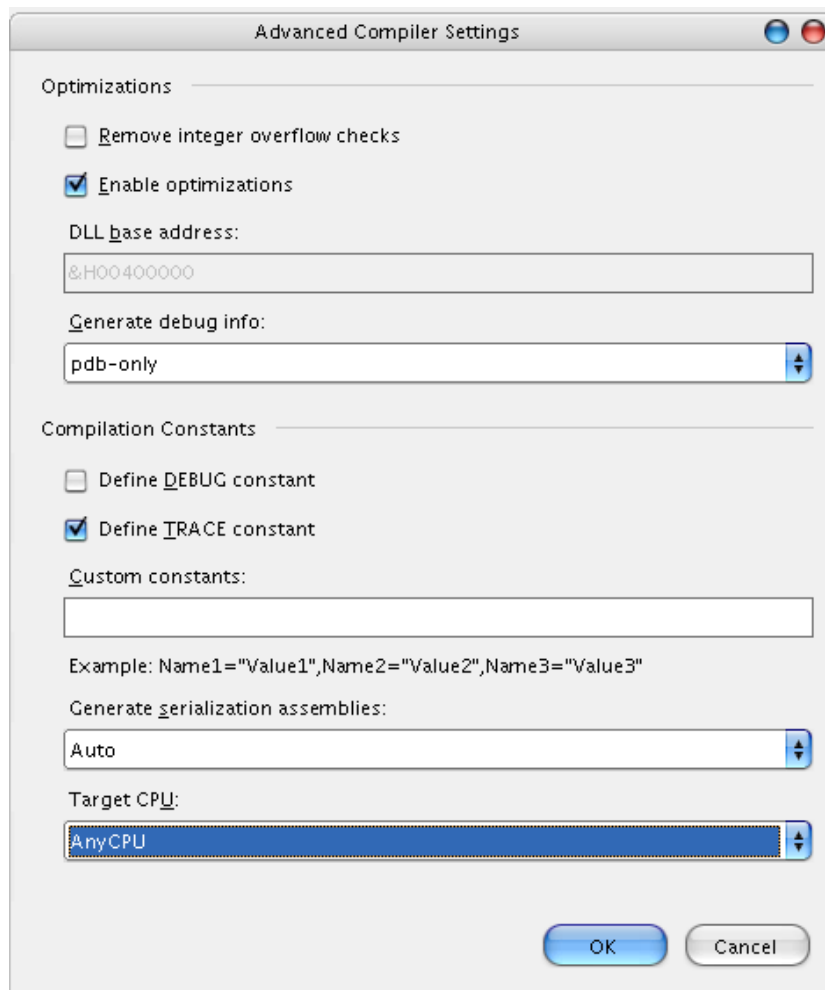
1. 在 Project 选单点击 VBNETTest Properties。



2. 点击 Compiler 后点击 Advanced Compiler Option 按钮，进入 Advanced Compiler Setting 窗口。



3. 在 Advanced Compiler Settings 下的 Target CPU 下拉菜单选择 AnyCPU。



Any CPU 选项-编译出来的执行档，当加载在 64 位作业统上的 64 位版本.NET Framework，程序将会以 64 位的行程来运作，否则将会以 32 位的行程来运作。

x86 选项-不论操作系统或.NET Framework 的版本，执行文件永远以 32 位来运作。

x64 选项-不论操作系统或.NET Framework 的版本，执行文件永远以 64 位来运作。

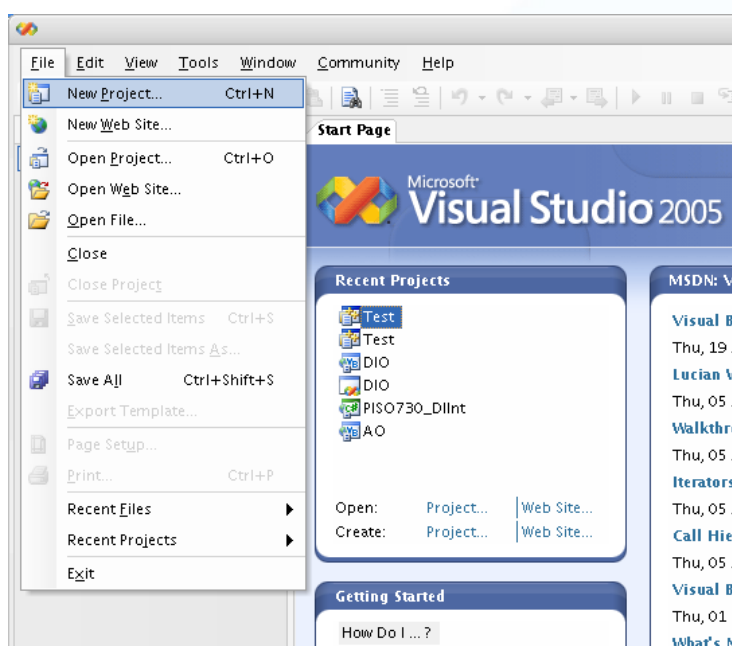
步骤 3:测试应用程序

1. 按下 F5 来执行程序。
2. 在 DO Value 字段输入数值 255。
3. 并按下 Write 按键，输出 DO 数值 255。

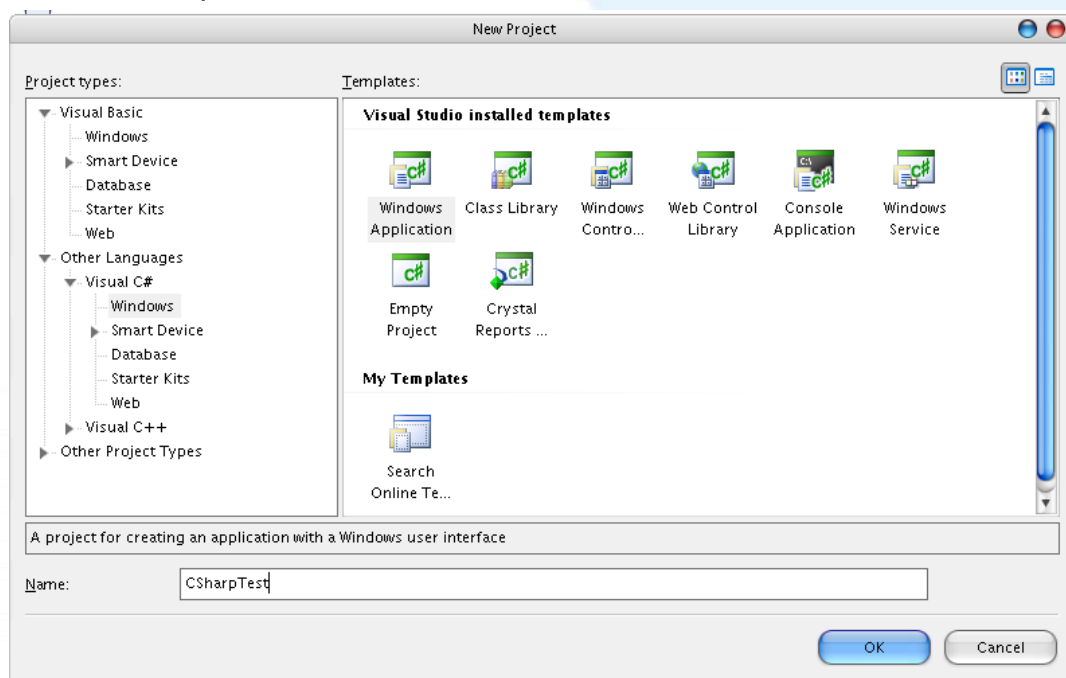
3.8. 在 Visual C#.NET

步骤 1: 撰写应用程序

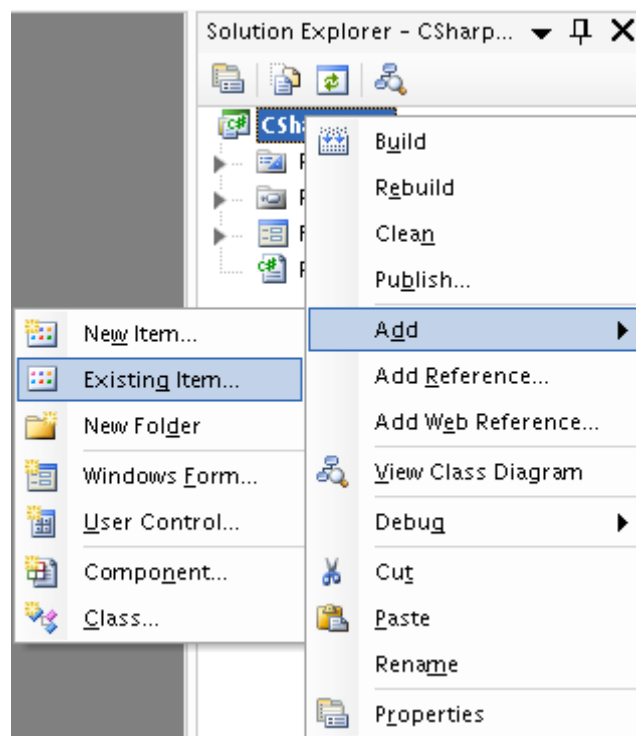
1. 至程序集开启 Microsoft Visual Studio 2005
2. 从主要选单内选择 File|New Project...



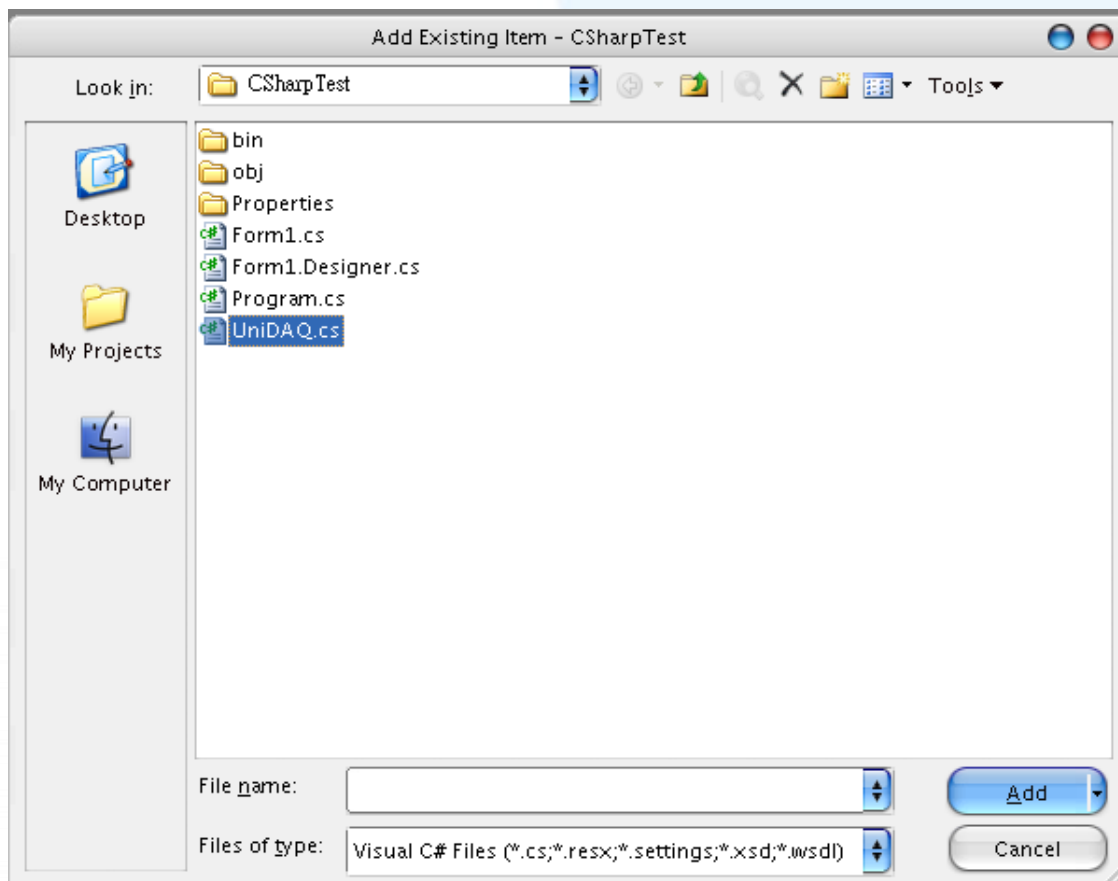
3. 在 Project type 下的列表点选项目 Visual C# 并在展开选单内选择 Windows，然后再右方 Templates 表框内选择 Windows Application，接下来至下方的 Name 键入项目名称 CSharpTest 然后按下 OK 按键。



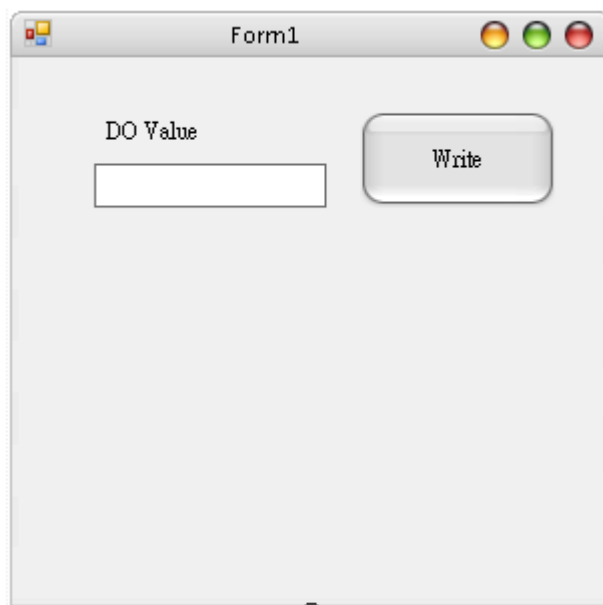
4. 在 Solution Explorer 开启 Add Existion item 窗口。



5. 在 Add Existion item 选择 UniDAQ.cs 后按 Add 按钮添加宣告文件至专案里。



6. 设计窗口，在 **Form1** 放置一个 **Label** 控件并在 **Text** 属性上输入 **DO Value**。接着放置 **TextBox** 控件，并切换至属性窗口上至 **Name** 属性输入 **txtDOVal**，最后放置一个 **Button** 控件，并修改 **Name** 属性为 **btnWrite** 及在 **Text** 属性上输入 **Write**。



7. 在 Form.cs 填写程序代码如下:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using UniDAQ_Ns; //Include the UniDAQ namespace

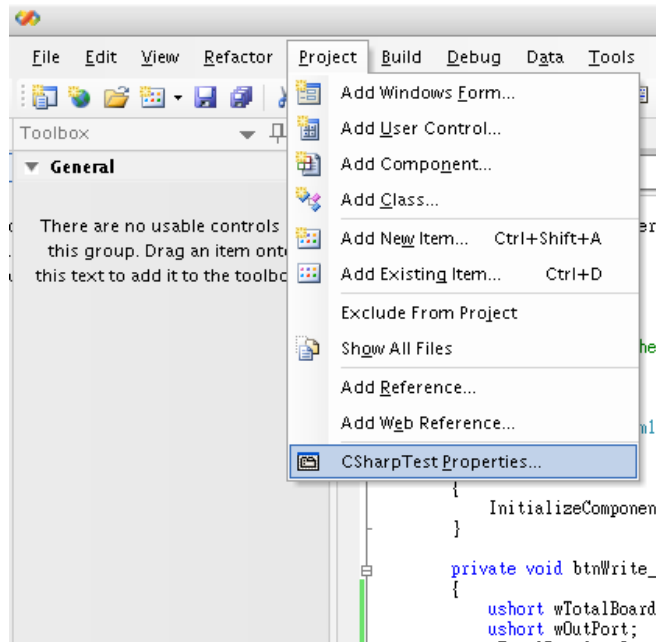
namespace CSharpTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnWrite_Click(object sender, EventArgs e)
        {
            ushort wTotalBoard, wRtn, wBoardNo;
            ushort wOutPort;
            wTotalBoard = 0;
            //Initial the resource and get total board number form Driver
            wRtn = UniDAQ.Ixud_DriverInit(ref wTotalBoard);
            if (wRtn != UniDAQ.Ixud_NoErr)
            {
                MessageBox.Show("Driver Inital Error!!Error Code:" + wRtn.ToString());
                Close();
                return;
            }

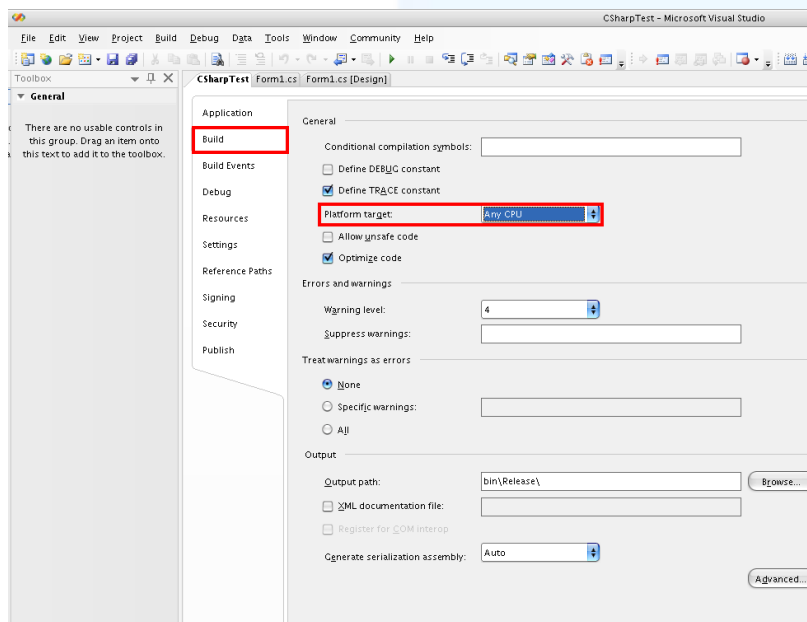
            wBoardNo = 0;
            wOutPort = 0;
            //Write DO
            wRtn = UniDAQ.Ixud_WriteDO(wBoardNo, wOutPort, Convert.ToInt32(txtDOVal.Text));
            //Release the resource from the driver
            wRtn = UniDAQ.Ixud_DriverClose();
        }
    }
}
```

步骤 2: 编译应用程序

1. 在 Project 选单点击 CSharpTest Properties。



2. 点击 Build 至 Platform target 下拉选单选择 Any CPU。



Any CPU 选项-编译出来的执行档，当加载在 64 位作业统上的 64 位版本.NET Framework，程序将会以 64 位的行程来运作，否则将会以 32 位的行程来运作。

x86 选项-不论操作系统或.NET Framework 的版本，执行文件永远以 32 位来运作。




x64 选项-不论操作系统或.NET Framework 的版本，执行文件永远以 64 位来运作。

步骤 3:测试应用程序

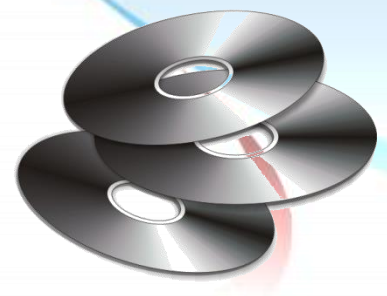
1. 按下 F5 来执行程序。
2. 在 DO Value 字段输入数值 255。
3. 并按下 Write 按键，输出 DO 数值 255。

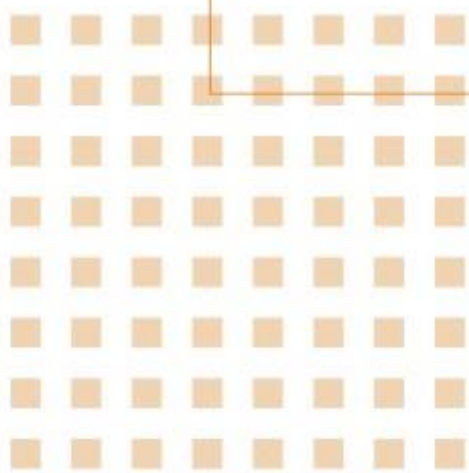
3.9. 范例程序及文件

放置 UniDAQ 相关数据的网址与位置:

	CD:\\ NAPDOS\\PCI\\UniDAQ\\
	http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/
	ftp://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/

UniDAQ 文件夹内档案文件结构





4. 函式应用

提供泓格 UniDAQ 驱动函式库所支持的范例程序行表以及每个范例程序的功能，并且会简单介绍如何使用函式来产生各种应用方案

4.1. 导读

UniDAQ 驱动函式库集成了各种函式,使用者可以利用它们来开发各种应用程序在泓格的装置上。这些 API 函式支持各种开发环境及程序语言,包括了 Microsoft Visual C++, Visual Basic, Borland Delphi, Borland C Builder++, Microsoft Visual C++.NET, Microsoft Visual C#.NET, Microsoft Visual VB.NET。

UniDAQ 提供了几个大类的函式集如下:

1. 驱动函式集:初始化装置资源、取得装置讯息、设定装置及释放装置资源。
2. 数字输出入函式集: 操作控制具有数字输出入功能的装置
3. 中断事件函式集: 支持具有中断功能的装置,当模拟输入及数字输入操作完成时产生通知事件。
4. 模拟输出函式集: 操作装置透过 DAC 输送出电压、电流
5. 模拟输入函式集: 操作装置透过 ADC 撷取电压、电流、压力、应变变量等等数值...
6. 计时计数函式集: 设定及读取计时计数器
7. 内存输出入函式集: 存取内存内的数值。

支持程序语言:

- Microsoft Visual C++ 4.0 or higher
- Microsoft Visual Basic 4.0 or higher
- Borland Delphi 2.0 or higher
- Borland C++ Builder 1.0 or higher
- Microsoft Visual C++.NET 2003 or higher
- Microsoft Visual C#.NET 2003 or higher
- Microsoft Visual Basic.NET 2003 or higher

泓格驱动函式库提供的应用函式集总表:

驱动函式集	数字输出入函式集	中断事件函式集	模拟输入函式集
Ixud_GetDIIVersion	Ixud_SetDIOModes32	Ixud_SetEventCallback	Ixud_ConfigAI
Ixud_OptionMode	Ixud_SetDIOMode	Ixud_RemoveEventCallback	Ixud_ConfigAIEx
Ixud_DriverInit	Ixud_ReadDI	Ixud_InstallIrq	Ixud_ClearAIBuffer
Ixud_DriverClose	Ixud_WriteDO	Ixud_RemoveIrq	Ixud_GetBufferStatus
Ixud_SearchCard	Ixud_ReadDIBit		Ixud_ReadAI
Ixud_GetBoardNoByCardID	Ixud_WriteDIBit		Ixud_ReadAIH
Ixud_GetCardInfo	Ixud_ReadDI32		Ixud_PollingAI
Ixud_ReadPort	Ixud_WriteDO32		Ixud_PollingAIH
Ixud_WritePort	Ixud_SoftwareReadbackDO		Ixud_PollingAIScan
Ixud_ReadPort32	Ixud_StartDI		Ixud_PollingAIScanH
Ixud_WritePort32	Ixud_StopDI		Ixud_StartAI
Ixud_ReadPhyMemory	Ixud_GetDIBufferH		Ixud_StartAIScan
Ixud_WritePhyMemory	Ixud_StartDO		Ixud_StartExtAI
	Ixud_StopDO		Ixud_StartExtAnalogTrigger
			Ixud_StartExtAIScan
			Ixud_GetAIBuffer
			Ixud_GetAIBufferH
			Ixud_StopAI

模拟输出函式集	计时计数函式集	内存输出入函式集
Ixud_ConfigAO	Ixud_ReadCounter	Ixud_ReadMemory
Ixud_WriteAOVoltage	Ixud_SetCounter	Ixud_WriteMemory
Ixud_WriteAOVoltageH	Ixud_DisableCounter	Ixud_ReadMemory32
Ixud_WriteAOCcurrent	Ixud_SetFCChannelMode	Ixud_WriteMemory32
Ixud_WriteAOCcurrentH	Ixud_ReadFrequency	
Ixud_StartAOVoltage		
Ixud_StartAOVoltageH		
Ixud_StopAO		

4.2. 驱动函式库

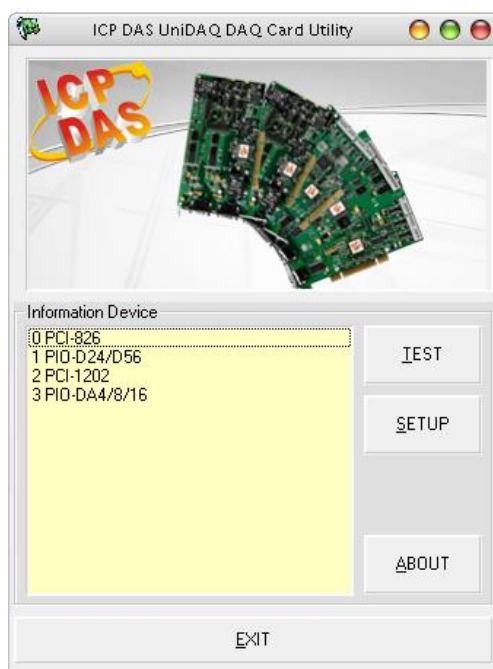
当用户使用泓格 **UniDAQ** 驱动函式库来开发板卡的应用程序时请遵照下列的呼叫流程来初始化及启动驱动程序及函式库。

呼叫流程



Board Num (Type: WORD, Size: 2 bytes)

藉由 Board Num 指定您需要作 I/O 操作的设备。Board Num 由 PCI Configuration space 的 Bus num 及 Device number 决定，如果 Bus number 愈小 Board Number 愈前面，Device Number 愈小 Board Num 愈前面。



在设备项目“0 PCI-826”，Board Num 即为 0，您可以透过 Board Num 直接指定来操作装置。

Ixud_DriverInit 及 Ixud_DriverClose 函式

Ixud_DriverInit 分配取得所有板卡的资源及数量在应用程序启动的时候，所以使用者必需呼叫 Ixud_DriverInit 在应用程序起始点，并且在使用其他函式之前。Ixud_DriverClose 会释放板卡所占用的系统资源，当用户不需要在操作板卡及程序终结前呼叫。

Ixud_GetCardInfo 函式

当使用者有需求知道板卡的名称或相关硬件信息时可透过 Ixud_GetCardInfo 函式取得，一般可忽略呼叫此函式。

4.3. 数字输出输入

数字输入输出函数集执行数字输入及数字输出等操作。在每一个数据采集板卡上，所有的数字输入输出线被分成一个一个的单位称为端口，每一个埠依板卡的设计会有 8、16 或 32 线。

某一些数据撷取板卡(例如：PIO-D24U/D56U/D48U/D96U/144U/168U)的数字输出输入端口，可以被设定为输入或输出。您可以使用 `Ixud_SetDIOModes32` 或 `Ixud_SetDIOMode` 函数来配置指定的端口为输出或输入。

4.3.1. 数字输入

泓格 UniDAQ 驱动函式库的数字输出入函式集可执行数字输入等功能，它支持软件触发数字输入及数字输入中断。

- 软件触发

使用者可以呼叫 `Ixud_ReadDI` 函式读取指定数字输入端口的数据。

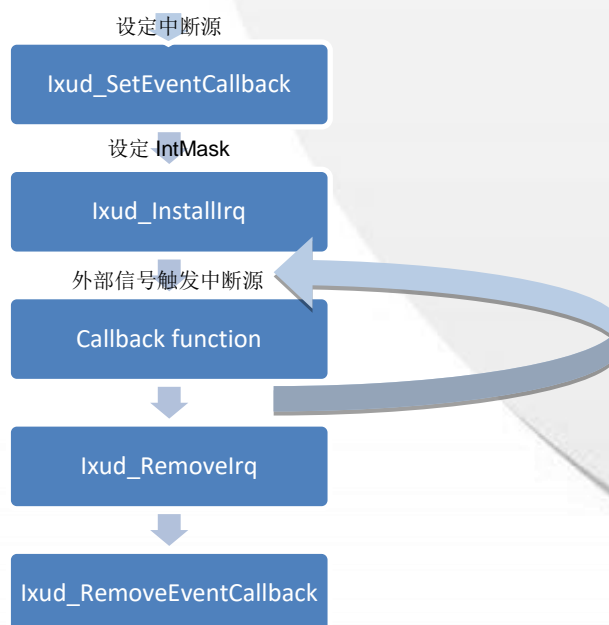
呼叫流程



- 中断触发

用户可透过中断触发功能监控数字输入信道的状态。当输入状态产生变化，输入准位由低变高或由高变低时，它将会透过硬件中断通知驱动程序，所以使用者就不需定时的去轮询数字输入信道。

呼叫流程



4.3.2. 数字输出

泓格 UniDAQ 驱动函式库的数字输出入函式集可执行数字输出等功能。

使用者可以透过 `Ixud_WriteDO` 函式简易的设定数字输出端口的数据，透过 `Ixud_SoftwareReadbackDO` 可取得目前数字输出端口的状态。

呼叫流程

设定埠号及 DO 值

`Ixud_WriteDO`

4.4. 模拟输入

模拟输入函式集运行数据擷取卡上模拟输入的功能。它可以擷取单笔数据、多信道数据或波型数据。模拟输入根据不同的触发模式及数据传输方式提供各种操作方式。

● 软件触发

透过软件触发数据转换来取得模拟数据，函式库提供三种应用方案。第一种是单信道单笔数据读取，第二种是单信道多笔数据读取，最后一种是多信道扫描。



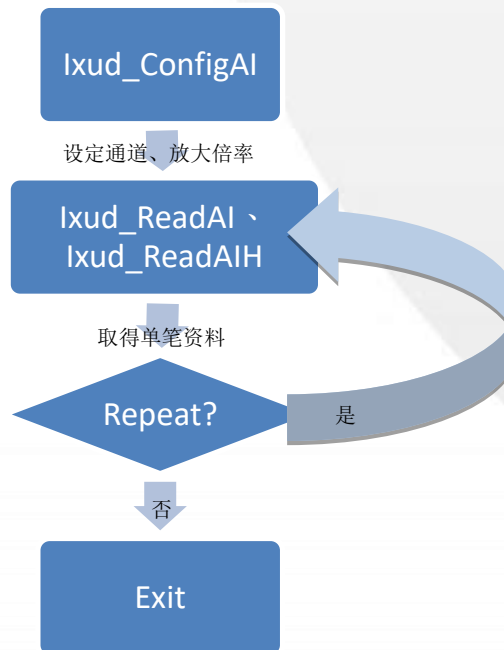
一般在 Windows 下使用软件触发，均容易受到多任务系统的影响，造成取样时间点将会受到其他系统任务的影响造成延迟取样，故不建议用户应用软件触发的方式量测模拟讯号波型，除非量测的波型属于非常低速的波型(低于 500Hz)

■ 单信道单笔数据取样

如果用户需要定时取样多笔模拟数据的功能，用户可以创建一个软件定时器(事件)，定时地呼叫函式 `Ixud_ReadAI` 或 `Ixud_ReadAIH`。

呼叫流程

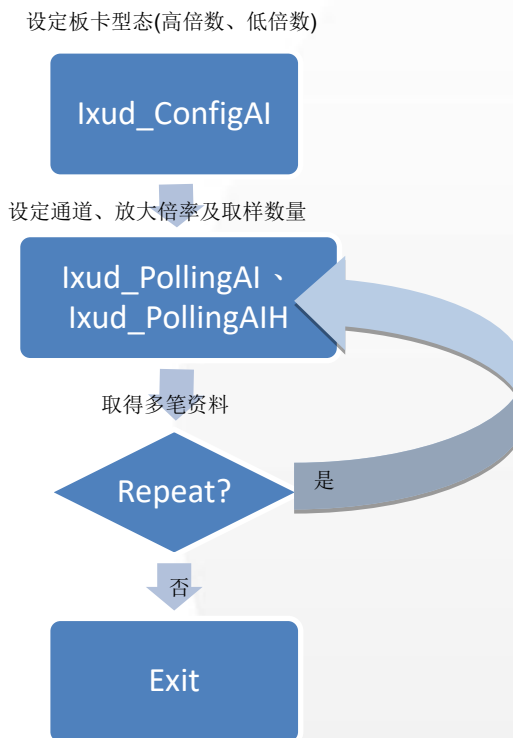
设定板卡型态(高倍数、低倍数)



■ 单信道多笔数据取样

单信道采样的功能与单信道单笔数据取样相似,可以在单一的通道上连续采集一个以上的模拟数据。

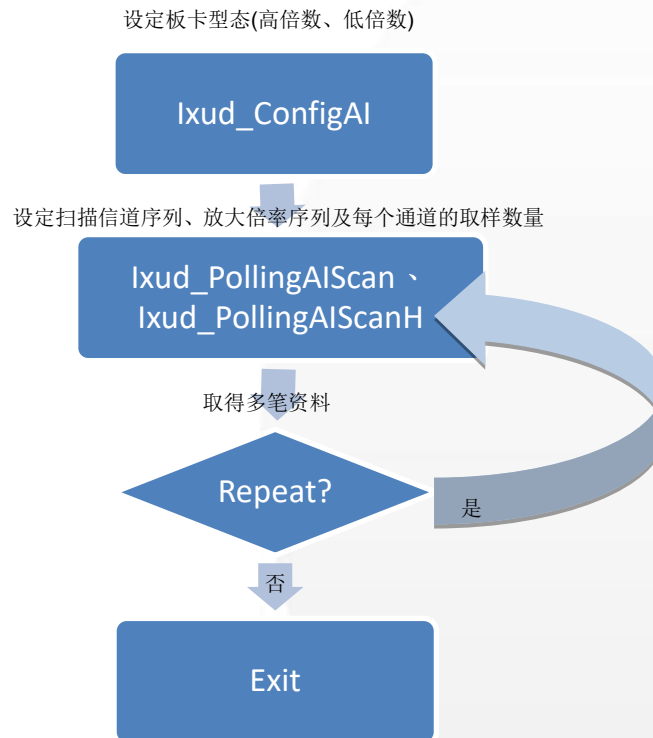
呼叫流程



■ 多通道采样

多信道采样的功能与单信道单笔数据取样相似，除了可以采样多个通道之外，还可以采集一个以上的模拟数据。

呼叫流程



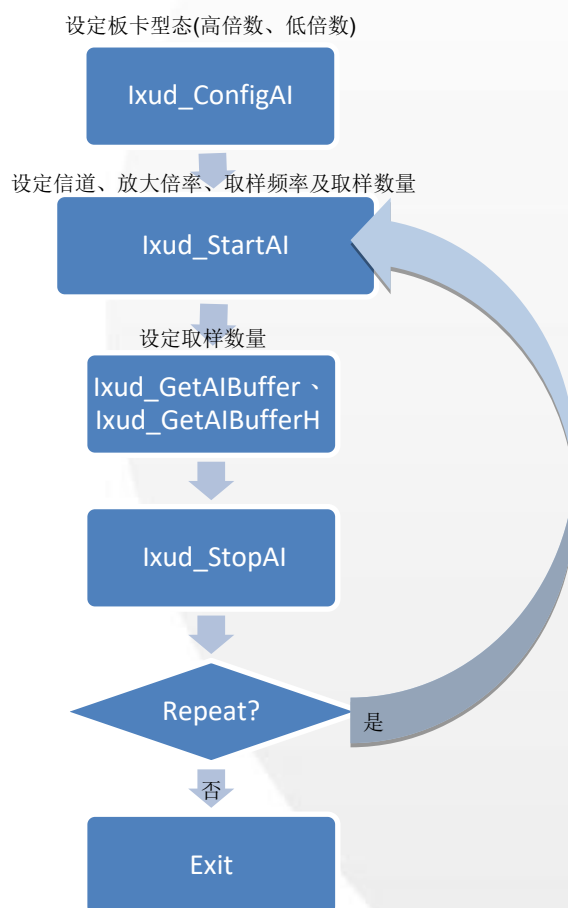
● 读取波型数据

模拟输入功能提供各种方式的应用让使用者取得最精确的波型数据，一般采集波型的触发模式有内部定时触发、中断触发和外部触发。

■ 单通道内部定时触发(single-channel Internal Pacer trigger)

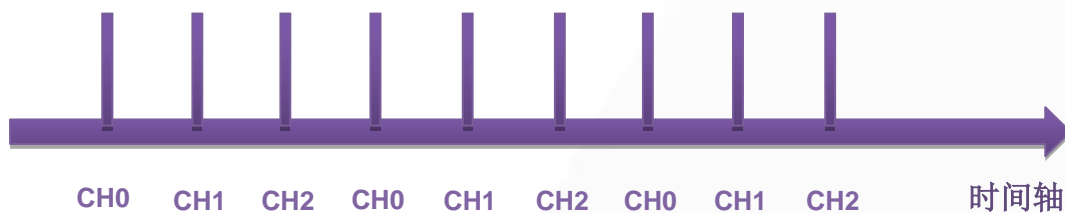
用户设定取样频率后透过板卡上内建的定时器以固定的频率触发 ADC 去采集单个模拟信道的波型数据。

呼叫流程

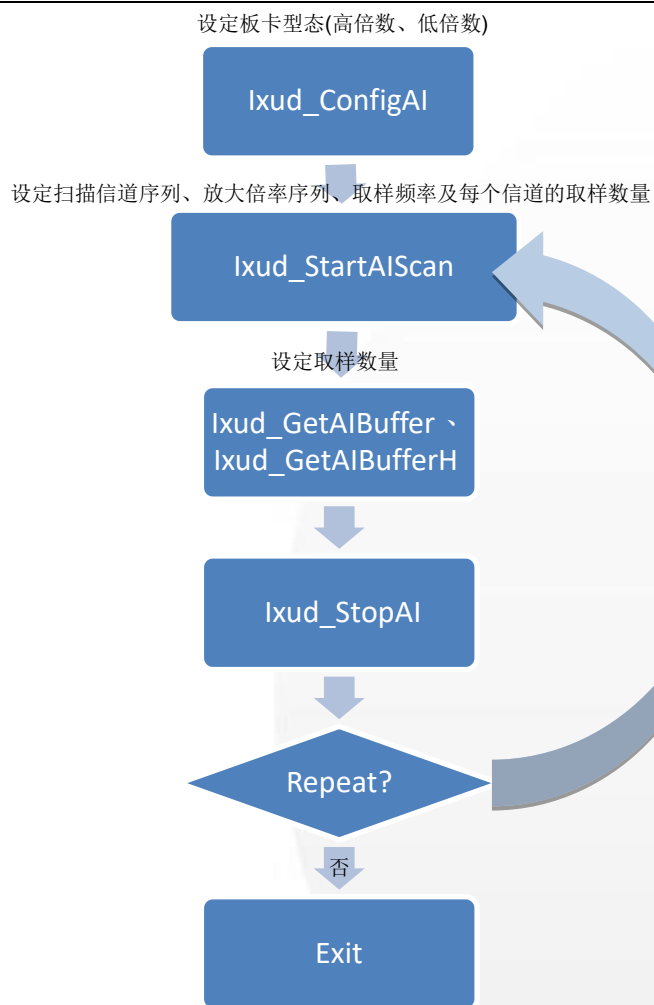


■ **多通道内部定时触发(multi-channel Internal Pacer trigger)**

用户设定取样频率后透过板卡上内建的定时器以固定的频率触发 ADC 去采集多个模拟信道的波型数据。



呼叫流程



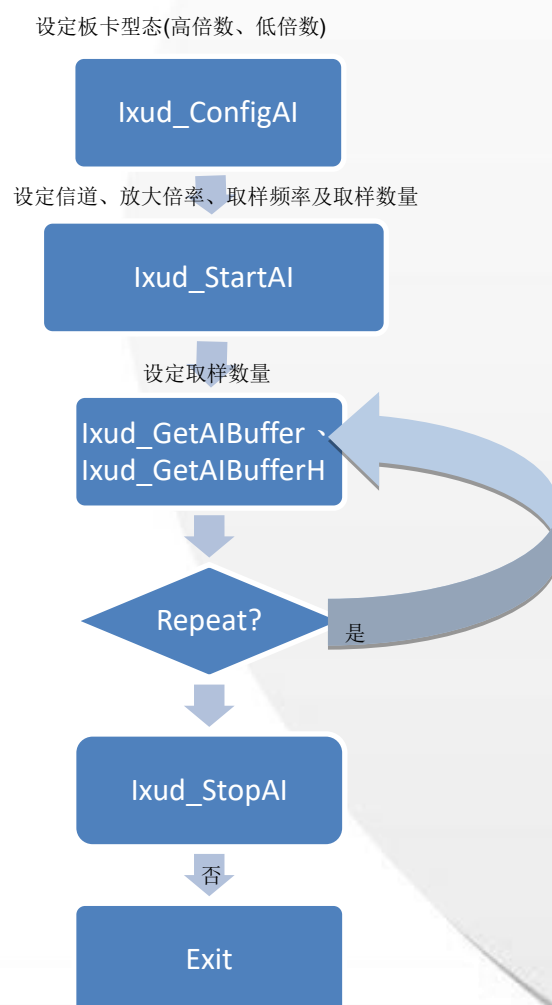
- **模拟讯号长时间监控**

采集模拟数据前将会利用系统内存配置一块缓冲区(默认值 2MB), 用来暂存所采集的模拟资料, 来达到连续采集的功能, 用户可以在采集的过程中从系统缓冲区内提取模拟数值。

- **单通道连续采集(single channel continuous capture)**

使用单一信道来连续采集模拟数据并储存至系统缓冲区等待用户提取。

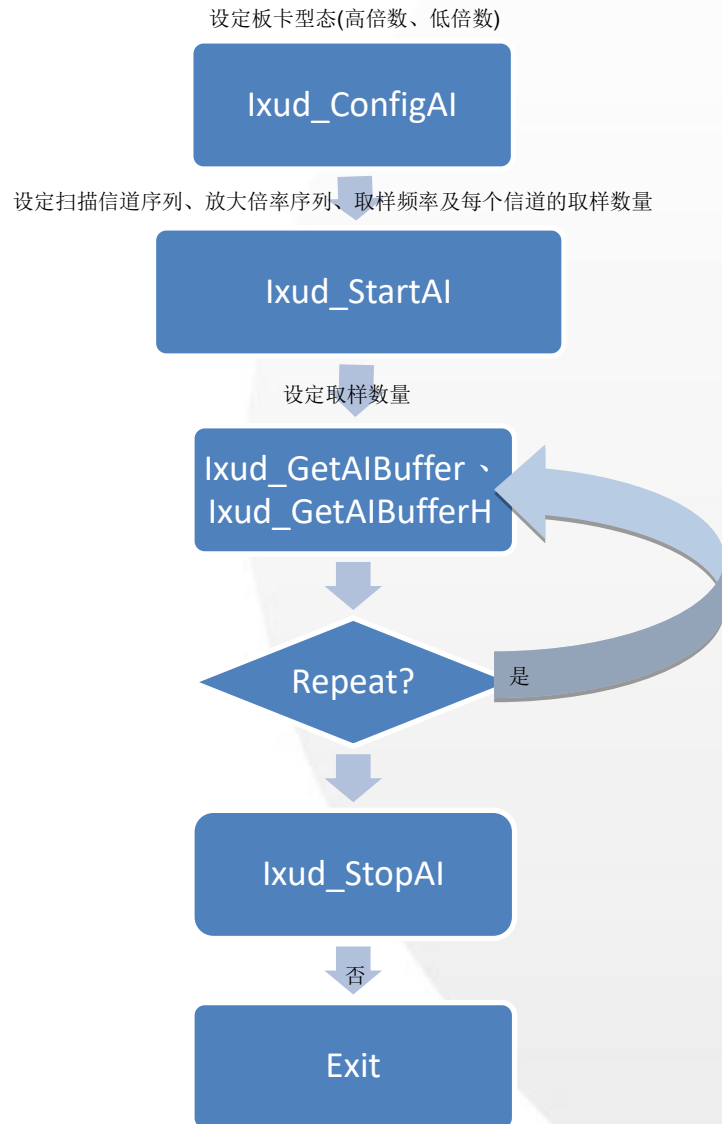
呼叫流程



■ 多通道连续采集(multi-channel continuous capture)

使用多信道来连续采集模拟数据并储存至系统缓冲区等待用户提取。

呼叫流程



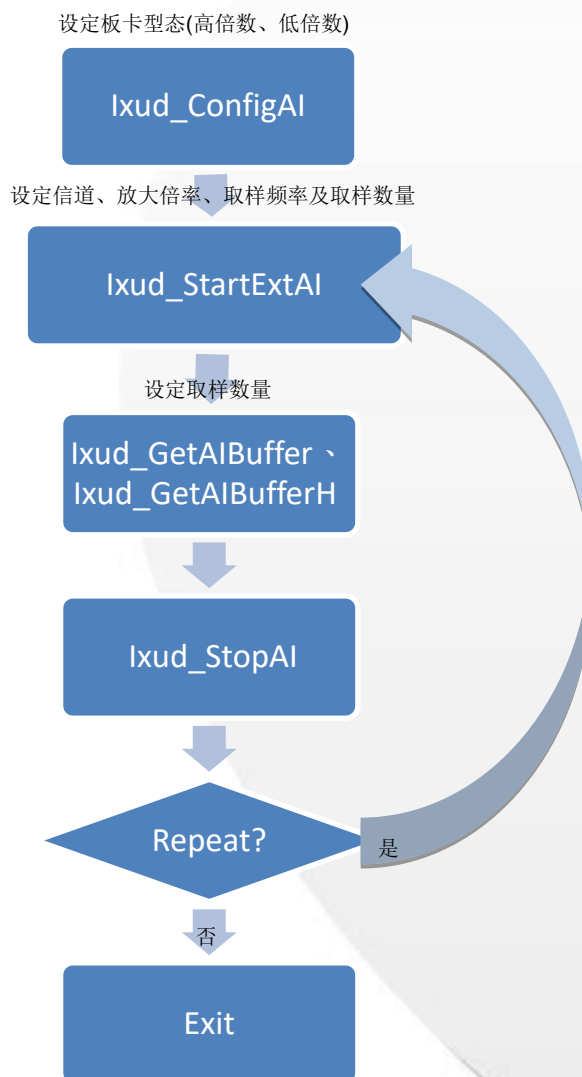
- 外部触发

透过外部信号来决定采样的起始时间，函式集提供三种方式外部触发的功能分别是 post-trigger、pre-trigger 及 mid-trigger。

- 单通道外部触发

透过外部触发信号，来决定单一信道数据采集的时机。

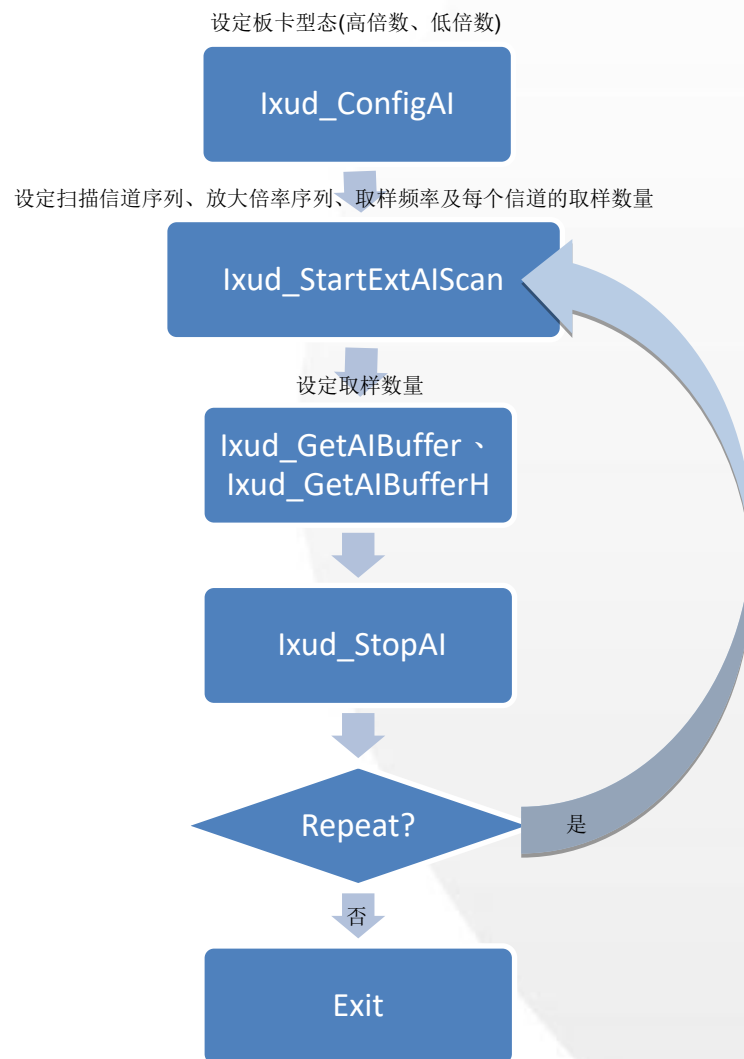
呼叫流程



■ 多通道外部触发波型采集

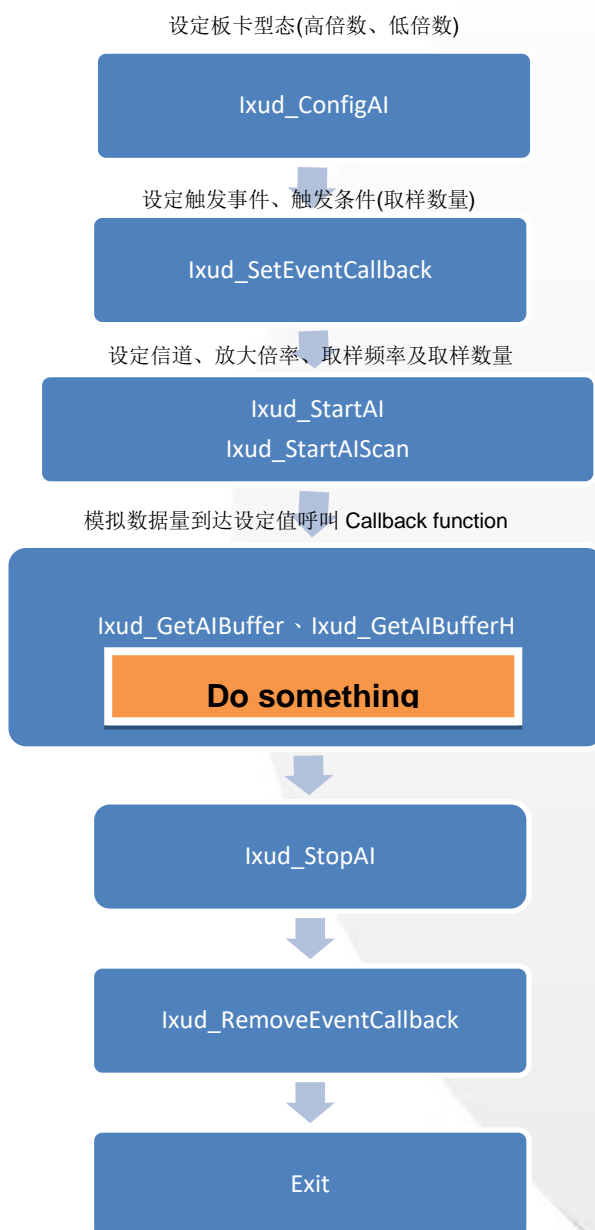
透过外部触发信号，来决定多信道数据采集的时机。

呼叫流程



- 模拟资料事件触发
当模拟数据每到达一定的数量时，驱动函式库会产生出一个事件通知使用者。

呼叫流程

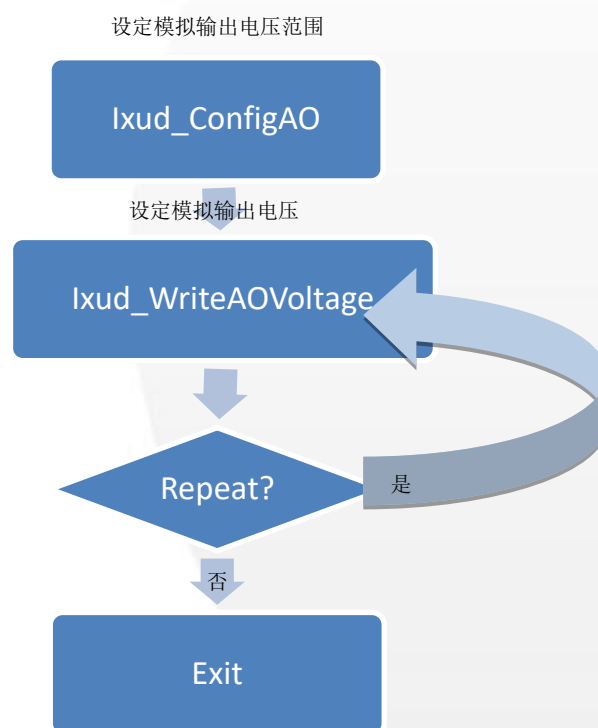


4.5. 模拟输出

泓格 UniDAQ 驱动函式库的模拟输出入函式集可执行模拟输出等功能。

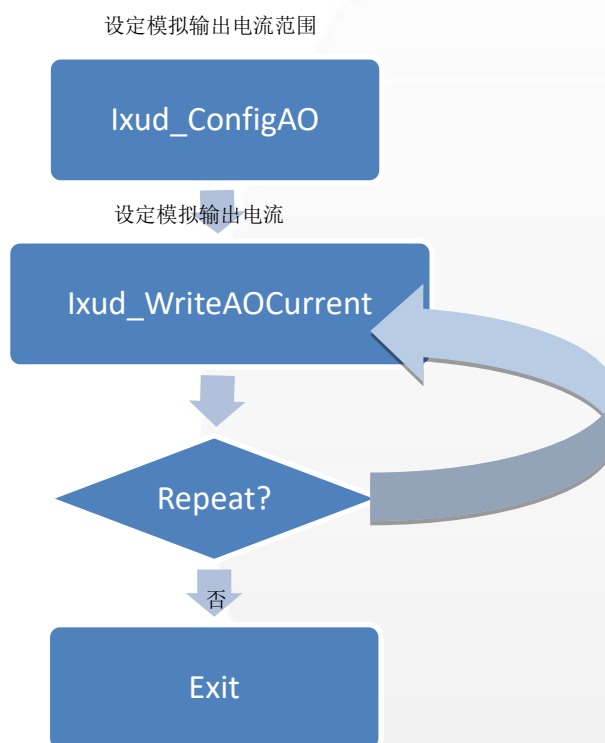
- 静态电压输出
设定模拟输出信道输出一固定的直流电压。

呼叫流程



- 静态电流输出
设定模拟输出通道输出一固定的电流。

呼叫流程



4.6. 计时计数器

泓格 UniDAQ 驱动函式库的计时计数函式集可执行 8254 计时计数器等功能。

- 写入设定计时计数器

呼叫流程

设定计时计数器信道、模式及数值

`Ixud_SetCounter`

- 读取计时计数器

呼叫流程

设定计时计数器通道

`Ixud_ReadCounter`

4.7. 内存存取

泓格 UniDAQ 驱动函式库的内存输出入函式集可执行对内存地址存取等功能。

- 写入内存地址

呼叫流程

设定地址、长度及写入数值

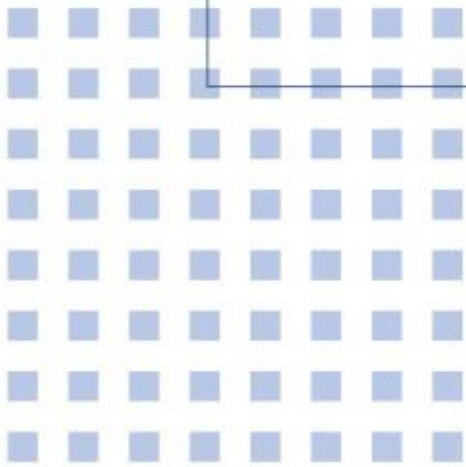
`Ixud_WriteMemory`

- 读取内存地址

呼叫流程

设定地址及长度

`Ixud_ReadCounter`



5. 函式参考

提供泓格 UniDAQ 驱动函式库使用指南，按照功能分组来说明函式的调用方式。

5.1. 函式支援列表

表一

函式名称	Ixud_DriverInit Ixud_DriverClose	Ixud_SearchCard	Ixud_GetCardInfo	Ixud_GetBoardNoByCardID
PIO-D24/D24U/D56/D56U PEX-D24/D56	✓	✓	✓	✓
PIO-D48/D48U/D48SU PEX-D48	✓	✓	✓	✓
PIO-D64/D64U	✓	✓	✓	✓
PIO-D96/D96U/D96SU PEX-D96S	✓	✓	✓	✓
PIO-D144/D144U/D144LU PEX-D144LS	✓	✓	✓	✓
PIO-D168/D168U	✓	✓	✓	✓
PCI-D96SU/D128SU	✓	✓	✓	✓
PISO-DA2/DA2U	✓	✓	✓	✓
PIO-DA4/DA8/DA16 PIO- DA4U/DA8U/DA16U PISO-DA4U/DA8U/DA16U PEX-DA4/DA8/DA16	✓	✓	✓	✓
PISO-813/813U	✓	✓	✓	✓
PCI-P8R8/P8R8U	✓	✓	✓	✓
PCI-P16R16/P16R16U	✓	✓	✓	✓
PCI-P16C16/P16C16U	✓	✓	✓	✓
PCI-P16PRO16/P16POR16U PEX-P16POR16i/P8POR8i	✓	✓	✓	✓
PISO-P8R8UDC/AC	✓	✓	✓	✓
PISO-P8R8/P8R8U PEX-P8R8i	✓	✓	✓	✓
PISO-P16R16U PEX-P16R16i	✓	✓	✓	✓
PISO-P32C32/P32C32U/P32C32U-5V PEX-P32C32	✓	✓	✓	✓
PISO-P32A32/P32A32U/P32A32U-5V PEX-P32A32	✓	✓	✓	✓
PISO-P32S32WU	✓	✓	✓	✓

函式名称	Ixud_DriverInit Ixud_DriverClose	Ixud_SearchCard	Ixud_GetCardInfo	Ixud_GetBoardNoByCardID
PISO-P64/P64U PEX-P64	✓	✓	✓	✓
PISO-A64/A64U/C64/C64U PEX-C64	✓	✓	✓	✓
PISO-725/725U	✓	✓	✓	✓
PISO-730/730A/730AU PEX-730/730A	✓	✓	✓	✓
PISO-1730U	✓	✓	✓	✓
PCI-1002 series PEX-1002 series	✓	✓	✓	✓
PCI-1202 series PEX-1202 series	✓	✓	✓	✓
PCI-1602 series	✓	✓	✓	✓
PCI-1800/1802 series	✓	✓	✓	✓
PIO-821 Series	✓	✓	✓	✓
PCI-822LU/826LU	✓	✓	✓	✓
PCI-2602U	✓	✓	✓	✓
PCIe-8620	✓	✓	✓	✓
PCIe-8622	✓	✓	✓	✓
PCI-M512/M512U	✓	✓	✓	✓
PCI-FC16U	✓	✓	✓	✓
PCI-TMC12/TMC12A/TMC12AU PEX-TMC12A	✓	✓	✓	✓

表二

函数名称	Ixud_ReadPort Ixud_ReadPort32	Ixud_WritePort Ixud_WritePort32	Ixud_SetDIOMode Ixud_SetDIOModes32
PIO-D24/D24U/D56/D56U PEX-D24/D56	✓	✓	✓
PIO-D48/D48U/D48SU PEX-D48	✓	✓	✓
PIO-D64/D64U	✓	✓	✓
PIO-D96/D96U/D96SU PEX-D96S	✓	✓	✓
PIO-D144/D144U/D144LU PEX-D144LS	✓	✓	✓
PIO-D168/D168U	✓	✓	✓
PCI-D96SU/D128SU	✓	✓	✓
PISO-DA2/DA2U	✓	✓	
PIO-DA4/DA8/DA16 PIO- DA4U/DA8U/DA16U PISO-DA4U/DA8U/DA16U PEX-DA4/DA8/DA16	✓	✓	
PISO-813/813U	✓	✓	
PCI-P8R8/P8R8U	✓	✓	
PCI-P16R16/P16R16U	✓	✓	
PCI-P16C16/P16C16U	✓	✓	
PCI-P16PRO16/P16POR16U PEX-P16POR16i/P8POR8i	✓	✓	
PISO-P8R8UDC/AC	✓	✓	
PISO-P8R8/P8R8U PEX-P8R8i	✓	✓	
PISO-P16R16U PEX-P16R16i	✓	✓	
PISO-P32C32/P32C32U/ P32C32U-5V PEX-P32C32	✓	✓	
PISO-P32A32/P32A32U/ P32A32U-5V PEX-P32A32	✓	✓	
PISO-P32S32WU	✓	✓	
PISO-P64/P64U PEX-P64	✓	✓	

函式名称	Ixud_ReadPort Ixud_ReadPort32	Ixud_WritePort Ixud_WritePort32	Ixud_SetDIOMode Ixud_SetDIOModes32
PISO-A64/A64U/C64/C64U PEX-C64	✓	✓	
PISO-725/725U	✓	✓	
PISO-730/730A/730AU PEX-730/730A	✓	✓	
PISO-1730U	✓	✓	
PCI-1002 series PEX-1002 series	✓	✓	
PCI-1202 series PEX-1202 series	✓	✓	
PCI-1602 series	✓	✓	
PCI-1800/1802 series	✓	✓	
PIO-821 Series	✓	✓	
PCI-822LU/826LU	✓	✓	✓
PCI-2602U	✓	✓	✓
PCle-8620	✓	✓	
PCle-8622	✓	✓	
PCI-M512/M512U	✓	✓	
PCI-FC16U	✓	✓	✓
PCI-TMC12/TMC12A/TMC12AU PEX-TMC12A	✓	✓	

表三

函式名称	Ixud_ReadDI	Ixud_WriteDO	Ixud_ReadDI32 Ixud_ReadDI32	Ixud_WriteDO32 Ixud_WriteDO32
PIO-D24/D24U/D56/D56U PEX-D24/D56	✓	✓	✓	✓
PIO-D48/D48U/D48SU PEX-D48	✓	✓	✓	✓
PIO-D64/D64U	✓	✓	✓	✓
PIO-D96/D96U/D96SU PEX-D96S	✓	✓	✓	✓
PIO-D144/D144U/D144LU PEX-D144LS	✓	✓	✓	✓
PIO-D168/D168U	✓	✓	✓	✓
PCI-D96SU/D128SU	✓	✓	✓	✓
PISO-DA2/DA2U				
PIO-DA4/DA8/DA16 PIO- DA4U/DA8U/DA16U PISO-DA4U/DA8U/DA16U PEX-DA4/DA8/DA16	✓	✓	✓	✓
PISO-813/813U				
PCI-P8R8/P8R8U	✓	✓	✓	✓
PCI-P16R16/P16R16U	✓	✓	✓	✓
PCI-P16C16/P16C16U	✓	✓	✓	✓
PCI-P16PRO16/P16POR16U PEX-P16POR16i/P8POR8i	✓	✓	✓	✓
PISO-P8R8UDC/AC	✓	✓	✓	✓
PISO-P8R8/P8R8U PEX-P8R8i	✓	✓	✓	✓
PISO-P16R16U PEX-P16R16i	✓	✓	✓	✓
PISO-P32C32/P32C32U/P32C32U-5V PEX-P32C32	✓	✓	✓	✓
PISO-P32A32/P32A32U/ P32A32U-5V PEX-P32A32	✓	✓	✓	✓
PISO-P32S32WU	✓	✓	✓	✓
PISO-P64/P64U PEX-P64	✓		✓	

函式名称	Ixud_ReadDI	Ixud_WriteDO	Ixud_ReadDI32 Ixud_ReadDI32	Ixud_WriteDOBit Ixud_WriteDO32
PISO-A64/ A64U /C64/C64U PEX-C64		✓		✓
PISO-725/725U	✓	✓	✓	✓
PISO-730/730A/730AU PEX-730/730A	✓	✓	✓	✓
PISO-1730U	✓	✓	✓	✓
PCI-1002 series PEX-1002 series	✓	✓	✓	✓
PCI-1202 series PEX-1202 series	✓	✓	✓	✓
PCI-1602 series	✓	✓	✓	✓
PCI-1800/1802 series	✓	✓	✓	✓
PIO-821 Series	✓	✓	✓	✓
PCI-822LU/826LU	✓	✓	✓	✓
PCI-2602U	✓	✓	✓	✓
PCIe-8620	✓	✓	✓	✓
PCIe-8622	✓	✓	✓	✓
PCI-M512/M512U	✓	✓	✓	✓
PCI-FC16U	✓	✓	✓	✓
PCI-TMC12/TMC12A/TMC12AU PEX-TMC12A	✓	✓	✓	✓

表四

函式名称	Ixud_SoftwareReadbackDO	Ixud_SetEventCallback Ixud_RemoveEventCallback	Ixud_InstallIrq Ixud_RemoveIrq
PIO-D24/D24U/D56/D56U PEX-D24/D56	✓	✓	✓
PIO-D48/D48U/D48SU PEX-D48	✓	✓	✓
PIO-D64/D64U	✓	✓	✓
PIO-D96/D96U/D96SU PEX-D96S	✓	✓	✓
PIO-D144/D144U/D144LU PEX-D144LS	✓	✓	✓
PIO-D168/D168U	✓	✓	✓
PCI-D96SU/D128SU	✓	✓	✓
PISO-DA2/DA2U		✓	✓
PIO-DA4/DA8/DA16 PIO- DA4U/DA8U/DA16U PISO-DA4U/DA8U/DA16U PEX-DA4/DA8/DA16	✓	✓	✓
PISO-813/813U			
PCI-P8R8/P8R8U	✓		
PCI-P16R16/P16R16U	✓		
PCI-P16C16/P16C16U	✓		
PCI-P16PRO16/P16POR16U PEX-P16POR16i/P8POR8i	✓		
PISO-P8R8JDC/AC	✓		
PISO-P8R8/P8R8U PEX-P8R8i	✓		
PISO-P16R16U PEX-P16R16i	✓		
PISO-P32C32/P32C32U/P32C32U-5V PEX-P32C32	✓		
PISO-P32A32/P32A32U/ P32A32U-5V PEX-P32A32	✓		
PISO- P32S32WU	✓		
PISO-P64/P64U PEX-P64	✓		

函式名称	Ixud_SoftwareReadbackDO	Ixud_SetEventCallback Ixud_RemoveEventCallback	Ixud_InstallIrq Ixud_RemoveIrq
PISO-A64/A64U/C64/C64U PEX-C64	✓		
PISO-725/725U	✓	✓	✓
PISO-730/730A/730AU PEX-730/730A	✓	✓	✓
PISO-1730U	✓		
PCI-1002 series PEX-1002 series	✓	✓	
PCI-1202 series PEX-1202 series	✓		
PCI-1602 series	✓		
PCI-1800/1802 series	✓		
PIO-821 Series	✓	✓	
PCI-822LU/826LU	✓	✓	
PCI-2602U	✓		
PCIe-8620	✓	✓	
PCIe-8622	✓	✓	
PCI-M512/M512U	✓		
PCI-FC16U	✓		
PCI-TMC12/TMC12A/TMC12AU PEX-TMC12A	✓	✓	✓

表五

函数名称	Ixud_ConfigAI Ixud_ConfigAIEx	Ixud_ClearAIBuffer	Ixud_GetBufferStatus	Ixud_ReadAI Ixud_ReadAIH
PIO-D24/D24U/D56/D56U PEX-D24/D56				
PIO-D48/D48U/D48SU PEX-D48				
PIO-D64/D64U				
PIO-D96/D96U/D96SU PEX-D96S				
PIO-D144/D144U/D144LU PEX-D144LS				
PIO-D168/D168U				
PCI-D96SU/D128SU				
PISO-DA2/DA2U				
PIO-DA4/DA8/DA16 PIO- DA4U/DA8U/DA16U PISO-DA4U/DA8U/DA16U PEX-DA4/DA8/DA16				
PISO-813/813U	✓			✓
PCI-P8R8/P8R8U				
PCI-P16R16/P16R16U				
PCI-P16C16/P16C16U				
PCI-P16PRO16/P16POR16U PEX-P16POR16i/P8POR8i				
PISO-P8R8UDC/AC				
PISO-P8R8/P8R8U PEX-P8R8i				
PISO-P16R16U PEX-P16R16i				
PISO-P32C32/P32C32U/P32C32U-5V PEX-P32C32				
PISO-P32A32/P32A32U/P32A32U-5V PEX-P32A32				
PISO- P32S32WU				
PISO-P64/P64U PEX-P64				

函式名称	Ixud_ConfigAI Ixud_ConfigAIEx	Ixud_ClearAIBuffer	Ixud_GetBufferStatus	Ixud_ReadAI Ixud_ReadAIH
PISO-A64/A64U/C64/C64U PEX-C64				
PISO-725/725U				
PISO-730/730A/730AU PEX-730/730A				
PISO-1730U				
PCI-1002 series PEX-1002 series	✓	✓	✓	✓
PCI-1202 series PEX-1202 series	✓	✓	✓	✓
PCI-1602 series	✓	✓	✓	✓
PCI-1800/1802 series	✓	✓	✓	✓
PIO-821 Series	✓	✓	✓	✓
PCI-822LU/826LU	✓	✓	✓	✓
PCI-2602U	✓	✓	✓	✓
PCle-8620	✓	✓	✓	✓
PCle-8622	✓	✓	✓	✓
PCI-M512/M512U				
PCI-FC16U				
PCI-TMC12/TMC12A/TMC12AU PEX-TMC12A				

表六

函式名称	Ixud_PollingAI	Ixud_PollingAIH	Ixud_PollingAIScan	Ixud_PollingAIScanH
PIO-D24/D24U/D56/D56U PEX-D24/D56				
PIO-D48/D48U/D48SU PEX-D48				
PIO-D64/D64U				
PIO-D96/D96U/D96SU PEX-D96S				
PIO-D144/D144U/D144LU PEX-D144LS				
PIO-D168/D168U				
PCI-D96SU/D128SU				
PISO-DA2/DA2U				
PIO-DA4/DA8/DA16 PIO- DA4U/DA8U/DA16U PISO-DA4U/DA8U/DA16U PEX-DA4/DA8/DA16				
PISO-813/813U	✓	✓	✓	✓
PCI-P8R8/P8R8U				
PCI-P16R16/P16R16U				
PCI-P16C16/P16C16U				
PCI-P16PRO16/P16POR16U PEX-P16POR16i/P8POR8i				
PISO-P8R8UDC/AC				
PISO-P8R8/P8R8U PEX-P8R8i				
PISO-P16R16U PEX-P16R16i				
PISO-P32C32/P32C32U/P32C32U-5V PEX-P32C32				
PISO-P32A32/P32A32U/P32A32U-5V PEX-P32A32				
PISO- P32S32WU				
PISO-P64/P64U PEX-P64				

函式名称	Ixud_PollingAI	Ixud_PollingAIH	Ixud_PollingAIScan	Ixud_PollingAIScanH
PISO-A64/A64U/C64/C64U PEX-C64				
PISO-725/725U				
PISO-730/730A/730AU PEX-730/730A				
PISO-1730U				
PCI-1002 series PEX-1002 series	✓	✓	✓	✓
PCI-1202 series PEX-1202 series	✓	✓	✓	✓
PCI-1602 series	✓	✓	✓	✓
PCI-1800/1802 series	✓	✓	✓	✓
PIO-821 Series	✓	✓	✓	✓
PCI-822LU/826LU	✓	✓	✓	✓
PCI-2602U	✓	✓	✓	✓
PCIe-8620	✓	✓	✓	✓
PCIe-8622	✓	✓	✓	✓
PCI-M512/M512U				
PCI-FC16U				
PCI-TMC12/TMC12A/TMC12AU PEX-TMC12A				

表七

函数名称	Ixud_StartAI Ixud_StopAI	Ixud_StartAIScan	Ixud_StartExtAI Ixud_StartExtAIScan	Ixud_GetAIBuffer Ixud_GetAIBufferH
PIO-D24/D24U/D56/D56U PEX-D24/D56				
PIO-D48/D48U/D48SU PEX-D48				
PIO-D64/D64U				
PIO-D96/D96U/D96SU PEX-D96S				
PIO-D144/D144U/D144LU PEX-D144LS				
PIO-D168/D168U				
PCI-D96SU/D128SU				
PISO-DA2/DA2U				
PIO-DA4/DA8/DA16 PIO- DA4U/DA8U/DA16U PISO-DA4U/DA8U/DA16U PEX-DA4/DA8/DA16				
PISO-813/813U				
PCI-P8R8/P8R8U				
PCI-P16R16/P16R16U				
PCI-P16C16/P16C16U				
PCI-P16PRO16/P16POR16U PEX-P16POR16i/P8POR8i				
PISO-P8R8UDC/AC				
PISO-P8R8/P8R8U PEX-P8R8i				
PISO-P16R16U PEX-P16R16i				
PISO-P32C32/P32C32U/P32C32U-5V PEX-P32C32				
PISO-P32A32/P32A32U/P32A32U-5V PEX-P32A32				
PISO- P32S32WU				
PISO-P64/P64U PEX-P64				

函式名称	Ixud_StartAI Ixud_StopAI	Ixud_StartAIScan	Ixud_StartExtAI Ixud_StartExtAIScan	Ixud_GetAIBuffer Ixud_GetAIBufferH
PISO-A64/A64U/C64/C64U PEX-C64				
PISO-725/725U				
PISO-730/730A/730AU PEX-730/730A				
PISO-1730U				
PCI-1002 series PEX-1002 series	✓	✓		✓
PCI-1202 series PEX-1202 series	✓	✓		✓
PCI-1602 series	✓	✓		✓
PCI-1800/1802 series	✓	✓		✓
PIO-821 Series	✓			✓
PCI-822LU/826LU	✓	✓	✓	✓
PCI-2602U	✓	✓	✓	✓
PCle-8620	✓	✓		✓
PCle-8622	✓	✓	✓	✓
PCI-M512/M512U				
PCI-FC16U				
PCI-TMC12/TMC12A/TMC12AU PEX-TMC12A				

表八

函式名称	Ixud_ConfigAO	Ixud_WriteAOVoltage Ixud_WriteAOVoltageH	Ixud_WriteAOCurrent Ixud_WriteAOCurrentH
PIO-D24/D24U/D56/D56U PEX-D24/D56			
PIO-D48/D48U/D48SU PEX-D48			
PIO-D64/D64U			
PIO-D96/D96U/D96SU PEX-D96S			
PIO-D144/D144U/D144LU PEX-D144LS			
PIO-D168/D168U			
PCI-D96SU/D128SU			
PISO-DA2/DA2U	✓	✓	✓
PIO-DA4/DA8/DA16 PIO- DA4U/DA8U/DA16U PISO-DA4U/DA8U/DA16U PEX-DA4/DA8/DA16	✓	✓	✓
PISO-813/813U			
PCI-P8R8/P8R8U			
PCI-P16R16/P16R16U			
PCI-P16C16/P16C16U			
PCI-P16PRO16/P16POR16U PEX-P16POR16i/P8POR8i			
PISO-P8R8UDC/AC			
PISO-P8R8/P8R8U PEX-P8R8i			
PISO-P16R16U PEX-P16R16i			
PISO-P32C32/P32C32U/P32C32U-5V PEX-P32C32			
PISO-P32A32/P32A32U/P32A32U-5V PEX-P32A32			
PISO-P32S32WU			
PISO-P64/P64U PEX-P64			

函式名称	Ixud_ConfigAO	Ixud_WriteAOVoltage Ixud_WriteAOVoltageH	Ixud_WriteAOCurrent Ixud_WriteAOCurrentH
PISO-A64/A64U/C64/C64U PEX-C64			
PISO-725/725U			
PISO-730/730A/730AU PEX-730/730A			
PISO-1730U			
PCI-1002 series PEX-1002 series			
PCI-1202 series PEX-1202 series	✓	✓	✓
PCI-1602 series	✓	✓	✓
PCI-1800/1802 series	✓	✓	✓
PIO-821 Series	✓	✓	✓
PCI-822LU/826LU	✓	✓	✓
PCI-2602U	✓	✓	✓
PCle-8620			
PCle-8622	✓	✓	✓
PCI-M512/M512U			
PCI-FC16U			
PCI-TMC12/TMC12A/TMC12AU PEX-TMC12A			

表九

函式名称	Ixud_ReadCounter	Ixud_SetCounter	Ixud_DisableCounter
PIO-D24/D24U/D56/D56U PEX-D24/D56			
PIO-D48/D48U/D48SU PEX-D48	✓	✓	✓
PIO-D64/D64U	✓	✓	✓
PIO-D96/D96U/D96SU PEX-D96S			
PIO-D144/D144U/D144LU PEX-D144LS			
PIO-D168/D168U			
PCI-D96SU/D128SU			
PISO-DA2/DA2U	✓	✓	✓
PIO-DA4/DA8/DA16 PIO-DA4U/DA8U/DA16U PISO-DA4U/DA8U/DA16U PEX-DA4/DA8/DA16	✓	✓	✓
PISO-813/813U			
PCI-P8R8/P8R8U			
PCI-P16R16/P16R16U			
PCI-P16C16/P16C16U			
PCI-P16PRO16/P16POR16U PEX-P16POR16i/P8POR8i			
PISO-P8R8UDC/AC			
PISO-P8R8/P8R8U PEX-P8R8i			
PISO-P16R16U PEX-P16R16i			
PISO-P32C32/P32C32U/P32C32U-5V PEX-P32C32			
PISO-P32A32/P32A32U/P32A32U-5V PEX-P32A32			
PISO-P32S32WU			
PISO-P64/P64U PEX-P64			

函式名称	Ixud_ReadCounter	Ixud_SetCounter	Ixud_DisableCounter
PISO-A64/A64U/C64/C64U PEX-C64			
PISO-725/725U			
PISO-730/730A/730AU PEX-730/730A			
PISO-1730U			
PCI-1002 series PEX-1002 series			
PCI-1202 series PEX-1202 series			
PCI-1602 series			
PCI-1800/1802 series			
PIO-821 Series	✓	✓	✓
PCI-822LU/826LU			
PCI-2602U	✓	✓	✓
PCle-8620			
PCle-8622	✓	✓	✓
PCI-M512/M512U			
PCI-FC16U	✓		
PCI-TMC12/TMC12A/TMC12AU PEX-TMC12A	✓	✓	✓

表十

函数名称	PCI-M512U
Ixud_ReadMemory Ixud_WriteMemory	✓
Ixud_ReadMemory32 Ixud_WriteMemory32	✓

表十一

函数名称	PCI-2602U	PCI-D96SU PCI-D128SU
Ixud_StartDO Ixud_StopDO	✓	✓
Ixud_StartDI Ixud_StopDI Ixud_GetDIBufferH	✓	
Ixud_StartExtAnalogTrigger	✓	
Ixud_StartAOVoltage Ixud_StartAOVoltageH Ixud_StopAO	✓	

5.2. 函式介绍

使用前请注意下列关键词。以方便您的阅读。

关键词	呼叫函式前需由使用者设定该参数	使用者呼叫函式后，会回传参数值
[Input]	Yes	No
[Output]	No	Yes

每一个泓格 UniDAQ 函式皆是如下的形式:

Status = 函式名称(参数 1, 参数 2, ... 参数 n)

每个函式皆会回传一个值在 **status** 变量里，它可以显示函式的呼叫成功或是失败。

Status(值)	结果
0	成功
>0	失败

Status 的格式为一 2 位无号整数值(WORD)，更多相关错误码 **Status** 值的定义请参考 A.1. 函数回传值定义

5.2.1. 驱动函式集

Ixud_GetDllVersion

取得 UniDAQ SDK 函式库的版本编号。

➤ 语法

```
WORD Ixud_GetDllVersion(  
    DWORD *dwDllVer  
);
```

➤ 参数

dwDllVer

[Output] 取得 UniDAQ SDK 函式库的版本值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_DriverInit

呼叫此函式时会向系统要求分配资源，并且开始寻找所有 UniDAQ 有支持的板卡，而对每一张板卡作初使化动作，最后取得板卡的数量。**需在程序起始点，使用其他的函式之前呼叫。**

➤ 语法

```
WORD Ixud_DriverInit(  
    WORD *wTotalBoards  
);
```

➤ 参数

wTotalBoards

[Output] 取得所有的板卡数量。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_DriverClose

呼叫此函式时，会将占用的资源释放归还给系统。**需在程序终结前呼叫。**

➤ 语法

```
WORD Ixud_DriverClose(  
    void  
);
```

➤ 参数

无任何参数。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_SearchCard

此函式提供使用者取得特定的版卡的数量，使用此函式后，此后其余需透过板卡编号来 I/O 的函式将会以此板卡的排序为基准。

➤ 语法

```
WORD Ixud_SearchCard(  
    WORD *wTotalBoards,  
    DWORD dwModelNo  
);
```

➤ 参数

wTotalBoards

[Output] 由用户设定的板卡模块识别号码，取得该板卡的数量。

dwModelNo

[Input] 用户设定板卡模块识别号码，此号码可参考 A.2. 模块识别号码。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_GetBoardNoByCardID

此函式提供给用户，可透过**模块识别号码**或**卡片识别号码**二种参数来取得该板卡排序的编号。

➤ 语法

```
WORD Ixud_GetBoardNoByCardID(  
    WORD *wBoardNo,  
    DWORD dwModelNumber,  
    WORD wCardID  
);
```

➤ 参数

wBoardNo

[Output] 取得板卡排序的编号。

dwModelNumber

[Input] 由用户设定板卡模块号码，此序号可参考 A.2. 模块识别号码。此值设定为 0 时，wBoardNo 会以 CardID 设定为主。

wCardID

[Input] 由用户设定板卡的识别号码。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_GetCardInfo

取得板卡的硬件数据、软件数据及板卡名称。

➤ 语法

```
WORD Ixud_GetCardInfo(  
    WORD wBoardNo,  
    PIXUD_DEVICE_INFO sDevInfo,  
    PIXUD_CARD_INFO sCardInfo,  
    char *szModelName  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

sDevInfo

[Output] 取得板卡在计算机上的系统信息。变量型态为 `PIXUD_DEVICE_INFO` 结构。

sCardInfo

[Output] 取得板卡的硬件规格信息。变量型态为一个 `PIXUD_CARD_INFO` 结构。

szModelName[]

[Output] 取得板卡的名称。请宣告一个 20 个字符大小的字符串。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadPort

从 I/O 埠读取一个 8/16/32bit 值。

➤ 语法

```
WORD Ixud_ReadPort(  
    DWORD dwAddress,  
    WORD wSize,  
    DWORD* dwVal  
);
```

➤ 参数

dwAddress

[Input] 设定 I/O 端口的地址。

wSize

[Input] 设定读取的 bit 长度。

<i>wSize</i>	长度 (bit)
8	8 (Byte)
16	16 (WORD)
32	32 (DWORD)

表格 5-1 *wSize* 参数设定

dwVal

[Output] 取得 I/O 埠的值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WritePort

从 I/O 埠写入一个 8/16/32bit 的值。

➤ 语法

```
WORD Ixud_WritePort(  
    DWORD dwAddress,  
    WORD wSize,  
    DWORD dwVal  
);
```

➤ 参数

dwAddress

[Input] 设定 I/O 端口的地址。

wSize

[Input] 设定写入的 bit 长度。

<i>wSize</i>	长度 (bit)
8	8 (Byte)
16	16 (WORD)
32	32 (DWORD)

表格 5-2 *wSize* 参数设定

dwVal

[Input] 写入 I/O 埠的值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadPort32

从 I/O 埠读取一个 32bit 的值。Visual Basic 6.0 用户建议使用此函数读取 32 位数值。

➤ 语法

```
WORD Ixud_ReadPort32(  
    DWORD dwAddress,  
    DWORD* dwLow,  
    DWORD* dwHigh  
);
```

➤ 参数

dwAddress

[Input] 设定 I/O 端口的地址。

dwLow

[Output] 取得 I/O 端口 32bit 低位部分的值。

dwHigh

[Output] 取得 I/O 端口 32bit 高位部分的值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WritePort32

从 I/O 埠写入一个 32bit 的值。Visual Basic 6.0 的使用者建议使用此函式写入 32 位数值。

➤ 语法

```
WORD Ixud_WritePort32(  
    DWORD dwAddress,  
    DWORD dwLow,  
    DWORD dwHigh  
);
```

➤ 参数

dwAddress

[Input] 设定 I/O 端口的地址。

dwLow

[Input] 写入 I/O 端口 32bit 低位部份的值。

dwHigh

[Input] 写入 I/O 端口 32bit 高位部份的值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadPhyMemory

从 memory mapping I/O 埠读取一个 8/16/32bit 值。

➤ 语法

```
WORD Ixud_ReadPhyMemory(  
    DWORD dwAddress,  
    WORD wSize,  
    DWORD* dwValue  
);
```

➤ 参数

dwAddress

[Input] 设定 I/O 端口的地址。

wSize

[Input] 设定读取的 bit 长度。

<i>wSize</i>	长度 (bit)
8	8 (Byte)
16	16 (WORD)
32	32 (DWORD)

表格 5-3 *wSize* 参数设定

dwValue

[Output] 取得 memory mapping I/O 埠的值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WritePhyMemory

从 memory mapping I/O 埠写入一个 8/16/32bit 的值。

➤ 语法

```
WORD Ixud_WritePortMemory(  
    DWORD dwAddress,  
    WORD wSize,  
    DWORD dwValue  
);
```

➤ 参数

dwAddress

[Input] 设定 memory mapping I/O 端口的地址。

wSize

[Input] 设定写入的 bit 长度。

<i>wSize</i>	长度 (bit)
8	8 (Byte)
16	16 (WORD)
32	32 (DWORD)

表格 5-4 *wSize* 参数设定

dwValue

[Input] 写入 memory mapping I/O 埠的值。

➤ 回传值

请参考 A.1. 函数回传值定义。

5.2.2. 数字输出输入函式集

Ixud_SetDIOModes32

同时设定多个端口的输入/出模式，仅支持含有双向 Digital I/O 功能的端口。

➤ 语法

```
WORD Ixud_SetDIOModes32(  
    WORD wBoardNo,  
    DWORD dwDioModeMask  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

dwDioModeMask

[Input] 设定双向 Digital I/O 端口为输出模式或输入模式，每个 bit 代表一个埠，最高可设定到 32 个埠，bit0 代表第 0 个埠，依此类推。详细端口的对映信息可参考附录 A.4. 数字输入端口定义号码及 A.5. 数字输出端口定义号码。

设定值	端口模式
0	输入模式
1	输出模式

表格 5-5 端口模式参数设定

➤ 回传值

请参考 A.1. 函数回传值定义。

➤ 范例

```
wBoardNo = 0; // 设定第一张板卡  
dwDioModeMask = 5; // 设定端口 0 及 2 为输出模式，端口 1 为输入模式  
wRtn = Ixud_SetDIOModes32(wBoardNo, dwDioModeMask);
```

Ixud_SetDIOMode

由使用者所选择的埠号，设定该端口的输入/出模式，**仅支持含有双向 Digital I/O 功能的端口。**

➤ **语法**

```
WORD Ixud_SetDIOMode(  
    WORD wBoardNo,  
    WORD wPortNo,  
    WORD wDioMode  
);
```

➤ **参数**

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。

wDioMode

[Input] 设定双向 Digital I/O 端口为输出模式或输入模式。详细端口的对映信息可参考附录 A.4. 数字输入端口定义号码及 A.5. 数字输出端口定义号码。

设定值	端口模式
0	输入模式
1	输出模式

表格 5-6 端口模式参数设定

➤ **回传值**

请参考 A.1. 函数回传值定义。

➤ **范例**

```
wRtn = Ixud_SetDIOMode(0, 1, 1); // 设定埠 1 为输出埠
```

Ixud_ReadDI

读取用户所指定的输入端口的数据。

➤ 语法

```
WORD Ixud_ReadDI(  
    WORD wBoardNo,  
    WORD wPortNo,  
    DWORD *dwDIVal  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.4. 数字输入端口定义号码。

dwDIVal

[Output] 读取输入端口里的数据。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WriteDO

写入数据到所用者所指定的输出埠。

➤ 语法

```
WORD Ixud_WriteDO(  
    WORD wBoardNo,  
    WORD wPortNo,  
    DWORD dwDOVal  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.5. 数字输出端口定义号码。

dwDOVal

[Input] 写入数据至输出端口。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadDIBit

读取用户所指定的输入信道内的数据。如果用户需要同步使用多个信道来操作数字输入功能，建议使用 `Ixud_ReadDI` 函数能得到较高的效能。

➤ 语法

```
WORD Ixud_ReadDIBit(  
    WORD wBoardNo,  
    WORD wPortNo,  
    WORD wBitNo,  
    WORD *wDIVal  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.4. 数字输入端口定义号码。

wBitNo

[Input] 由用户指定的通道号。

wDIVal

[Output] 取得输入信道的数据。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WriteDOBit

写入数据到所用者所指定的输出通道。如果用户需要同步使用多个信道来操作数字输出功能，建议使用 `Ixud_WriteDO` 函式能得到较高的效能。

➤ 语法

```
WORD Ixud_WriteDOBit(  
    WORD wBoardNo,  
    WORD wPortNo,  
    WORD wBitNo,  
    WORD wDOVal  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.5. 数字输出端口定义号码。

wBitNo

[Input] 由用户指定的通道号。

wDOVal

[Input] 写入数据至输出信道。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadDI32

从使用者所指定的输出埠读取 32bit 值。不带有无号整数型态的编程语言的用户，如 **Visual Basic 6.0**，建议使用此函式。

➤ 语法

```
WORD Ixud_ReadDI32(  
    WORD wBoardNo,  
    WORD wPortNo,  
    DWORD* dwLow,  
    DWORD* dwHigh,  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.4. 数字输入端口定义号码。

dwLow

[Output] 读取输入端口里 bit 0 ~ 15 的数据。

dwHigh

[Output] 读取输入端口里 bit 16 ~ 31 的数据。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WriteDO32

写入 32-Bit 数据到所用者所指定的输出埠，不带有无号整数型态的编程语言的用户，如 **Visual Basic 6.0** 建议使用此函式。

➤ 语法

```
WORD Ixud_WriteDO32(  
    WORD wBoardNo,  
    WORD wPortNo,  
    DWORD dwLow,  
    DWORD dwHigh,  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.5. 数字输出端口定义号码。

dwLow

[Input] 写入 bit 0 ~ 15 的数据至输入端口。

dwHigh

[Input] 写入 bit16 ~ 31 的数据至输出端口。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_SoftwareReadbackDO

在使用函式库的期间软件将会自动记录数字输出值, 让用户可以透过函式库的协助读取输出端口的值(非缓存器状态)。

➤ 语法

```
WORD Ixud_SoftwareReadbackDO(  
    WORD wBoardNo,  
    WORD wPortNo,  
    DWORD *dwDOVal  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号, 第一张板卡的 *wBoardNo* 为 0, 第二张板卡的 *wBoardNo* 为 1, 依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.5. 数字输出端口定义号码。

dwDOVal

[Output] 回读输出埠的写入值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StartDI

透过此函式来初始化数字输入端口、设定取样频率、采样数据量，并开始启动数字数据撷取，启动后将以等速度撷取单一端口的数字数据并将数据暂存至系统内存内，可透过 `Ixud_GetDIBufferH` 函式取出模拟输入数据，欲停止高速模拟输入采集功能时，需呼叫 `Ixud_StopDI` 函式。



本函式仅支援 PCI-2602U



为了确保数据采集的准确度，使用此函式在数据采集的过程中时将会占用 CPU 一段短暂的时间至采集数据完成，此时间将取决于用户设定的采集数据量及取样频率。

➤ 语法

```
WORD Ixud_StartDI(  
    WORD wBoardNo,  
    WORD wPortNo,  
    DWORD dwReserved,  
    float fSamplingRate,  
    DWORD dwDataCount  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.4. 数字输入端口定义号码。

dwReserved

[Input] 保留。

fSamplingRate

[Input] 为一浮点数，由用户设定数字输入端口的取样频率(次/秒)。

dwDataCount

[Input] 由用户设定需要采集的数据笔数。使用 `Ixud_StopDI` 函式停止采集。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StartDO

依用户指定的数据大小、数据缓冲区及循环模式启动高速数字输出模式。



本函式仅支援 PCI-2602U / PCI-D96SU / PCI-D128SU

➤ 语法

```
WORD Ixud_StartDO(  
    WORD wBoardNo,  
    WORD wPortNo,  
    DWORD dwReserved,  
    float fFrequency,  
    DWORD dwDataCount,  
    DWORD dwCycleNum,  
    DWORD dwDOBuf[ ]  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.5. 数字输出端口定义号码。

dwReserved

[Input] 保留。

fFrequency

[Input] 为一浮点数,由用户设定数字输出端口的输出每笔数字数据的频率(次/秒)。

dwDataCount

[Input] 由使用者设定需要送出一个波型的数据笔数。使用 `Ixud_StopDO` 函式停止采集。

dwCycleNum

[Input] 0:连续模式,如果要终止高速数字输出功能,请使用 `Ixud_StopDO` 函式。

dwDOBuf[]

[Input] 储存模拟输出数据至数字输出缓冲区。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_GetDIBufferH

此函式将会从内存里取得数字输入值，并储存在用户所指令的数组里，其值为十六进制值。呼叫此函式前需先呼叫 **Ixud_StartDI** 函式。



本函式仅支援 PCI-2602U

➤ 语法

```
WORD Ixud_GetDIBufferH(  
    WORD wBoardNo,  
    WORD wPortNo,  
    DWORD dwDataCount,  
    DWORD hValue[ ]  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.4. 数字输入端口定义号码。

dwDataCount

[Input] 由用户设定需要的数据笔数。

hValue[]

[Output] 请宣告一个 **DWORD** 数组。用来取得数字输入值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StopDI

呼叫此函式，用来停止高速数字输入撷取的运作。



本函式仅支援 PCI-2602U

➤ 语法

```
WORD Ixud_StopDI(  
    WORD wBoardNo,  
    WORD wPortNo  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.4. 数字输入端口定义号码。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StopDO

呼叫此函式，用来停止高速数字输出的运作。



本函式仅支援 PCI-2602U / PCI-D96SU / PCI-D128SU

➤ 语法

```
WORD Ixud_StopDO(  
    WORD wBoardNo,  
    WORD wPortNo  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wPortNo

[Input] 由使用者指定的埠号。详细端口的对映信息可参考附录 A.5. 数字输出端口定义号码。

➤ 回传值

请参考 A.1. 函数回传值定义。

5.2.3. 中断事件函数集

Ixud_SetEventCallback

此函数用来设定中断事件发生时的 Callback function，当不再使用 Callback 功能时请呼叫 Ixud_RemoveEventCallback 函数来关闭此功能。

➤ 语法

```
WORD Ixud_SetEventCallback(  
    WORD wBoardNo,  
    WORD wEventType,  
    WORD wInterruptSource,  
    HANDLE *hEvent,  
    PVOID CallbackFun,  
    DWORD dwCallBackParameter  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wEventType

[Input] 由使用者设定事件发生的条件，每个 bit 代表一种模式，可同时开启。设定值请参考 A.3.4. 中断事件配置码

wInterruptSource

[Input] 由使用者设定事件发生时所启用的中断源。各板卡使用的中断源设定值请参考下面表格。

wInterrupt Source	PIO-D24U PEX-D24 PIO-D56U PEX-D56	PIO-D48U PIO-D48SU PEX-D48	PIO-D64U	PIO-D96U PIO-D96SU PEX-D96S	PIO-D144U PIO-D144LU PIO-D168U PEX-D144LS	PCI-D96SU	PCI-D128SU
1	P2C0	P2C3/P2C7	EXTIRQ	P2C0	P2C0	Port 0	Port 0
2	P2C1	P5C3/P5C7	EVTIRQ	P5C0	P2C1	Port 1	Port 1
3	P2C2	COU0	TMRIRQ	P8C0	P2C2	Port 2	Port 2
4	P2C3	COU2	-	P11C0	P2C3		Port 3

wInterrupt Source	PIO-DA4U PIO-DA8U PIO-DA16U PISO-DA4U PISO-DA8U PISO-DA16U	PEX-DA4 PEX-DA8 PEX-DA16	PISO-730 PISO-730A PISO-730U PISO-730AU PEX-730 PEX-730A	PISO-725 PISO-725U	PCI-TMC12 PCI-TMC12A PCI-TMC12AU PEX-TMC12A	PIO-821 PCI-1002 PEX-1002	PCI-822LU PCI-826LU PCI-2602U PCIe-8620 PCIe-8622
1	COUT0	COUT0	DI0	IDI0	COUT3/6/9/12/Ext	AD Data	AD Data
2	COUT2	COUT2	DI1	IDI1	-	-	-
3	-	-	-	IDI2	-	-	-
4	-	-	-	IDI3	-	-	-
5	-	-	-	IDI4	-	-	-
6	-	-	-	IDI5	-	-	-
7	-	-	-	IDI6	-	-	-
8	-	-	-	IDI7	-	-	-



hEvent

[Input] 由使用者传入事件的指标，此事件请使用 Windows API CreateEvent(..) 函式创造，或设定为 0，系统会自动生成一个事件。

CallbackFun

[Input] 由使用者设定事件所使用的 Callback Function。

dwCallbackParameter

[Input] 由使用者设定事件所使用 Callback Function 的参数，wEventType 设置为 ADC Ready 模式时此参数可用来设定 ADCReady 的数据量。

➤ 回传值

请参考 A.1. 函数回传值定义。

➤ 范例

DI Callback

```
//Set DI Callback function
// Use Source = 1 Event Type = Active High +Hardware Interrupt
wRtn = Ixud_SetEventCallback(wBoardNo, IXUD_HARDWARE_INT|IXUD_ACTIVE_HIGH , 1, hEvent0,
Callbackfun0, 0);

//Use Source = 3 Event Type = Active Low +Hardware Interrupt
wRtn = Ixud_SetEventCallback(wBoardNo, IXUD_HARDWARE_INT|IXUD_ACTIVE_LOW, 3, hEvent2,
Callbackfun2, 0);
```

AI Callback

```
// Set AI Callback function
// Use Source = 1 Event Type = APC Ready+Hardware Interrupt 每笔AD数据产生一次Callback Event
DataNum=1000;
wRtn = Ixud_SetEventCallback(wBoardNo, IXUD_HARDWARE_INT|IXUD_APC_READY_INT , 1, hEvent0,
Callbackfun0,DataNum);
```

Ixud_RemoveEventCallback

此函式用来关闭或移除中断事件所发生的 **Callback function**。使用此函式需在使用 **Ixud_SetEventCallback** 函式之后，终结使用 **Callback function** 前呼叫。

➤ 语法

```
WORD Ixud_RemoveEventCallback(  
    WORD wBoardNo,  
    WORD wInterruptSource  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wInterruptSource

[Input] 由使用者设定事件发生时所启用的中断源，第一个中断源的 **wInterruptSource** 为 1，依此类推。如果设置为 0 将会一次移除该板卡所使用的事件。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_InstallIrq

此函式用来安装设定中断例程，并支持同时启动多个中断源。

注：模拟输入中断功能的用户请勿呼叫此函式。

➤ 语法

```
WORD Ixud_InstallIrq(  
    WORD wBoardNo,  
    DWORD dwInterruptMask  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

dwInterruptMask

[Input] 由使用者设定安装中断源，每个 bit 代表一个中断源，bit0 为第 0 个中断源，依此类推。

Bit	PIO-D24U PEX-D24 PIO-D56U PEX-D56	PIO-D48U PIO-D48SU PEX-D48	PIO-D64U	PIO-D96U PIO-D96SU PEX-D96S	PIO-D144U PIO-D144LU PIO-D168U PEX-D144LS	PCI-D96SU	PCI-D128SU
0	P2C0	P2C3/P2C7	EXTIRQ	P2C0	P2C0	Port 0	Port 0
1	P2C1	P5C3/P5C7	EVTIRQ	P5C0	P2C1	Port 1	Port 1
2	P2C2	COUT0	TMRIRQ	P8C0	P2C2	Port 2	Port 2
3	P2C3	COUT2	-	P11C0	P2C3	-	Port 3

Bit	PIO-DA4U PIO-DA8U PIO-DA16U PISO-DA4U PISO-DA8U PISO-DA16U	PEX-DA4 PEX-DA8 PEX-DA16	PISO-730 PISO-730A PISO-730U PISO-730AU PEX-730 PEX-730A	PISO-725 PISO-725U	PCI-TMC12 PCI-TMC12A PCI-TMC12AU PEX-TMC12A
0	COUT0	COUT0	DI0	IDI0	COUT3/6/9/12/EXT
1	COUT2	COUT2	DI1	IDI1	-
2	-	-	-	IDI2	-
3	-	-	-	IDI3	-
4	-	-	-	IDI4	-
5	-	-	-	IDI5	-
6	-	-	-	IDI6	-
7	-	-	-	IDI7	-

	Digital Input
	Timer/Counter

➤ 回传值

请参考 A.1. 函数回传值定义。

➤ 范例

```
dwInterruptMask = 0xF //(开启INT_0,INT_1,INT_2及INT_3)  
wRtn=Ixud_InstallIrq(wBoardNo,dwInterruptMask);
```

Ixud_RemoveIrq

此函式用来关闭中断例程。

➤ 语法

```
WORD Ixud_RemoveIrq(  
    WORD wBoardNo  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

➤ 回传值

请参考 A.1. 函数回传值定义。

5.2.4. 模拟输入函数集

Ixud_ConfigAI

- 此函数用来设定板卡模拟输入功能的参数值，**使用模拟输入函数集前必需先呼叫此函数。**

- 语法

```
WORD Ixud_ConfigAI(  
    WORD wBoardNo,  
    WORD wFIFOSizeKB,  
    DWORD dwBufferSizeCount,  
    WORD wCardType,  
    WORD wDelaySettlingTime  
);
```

- 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wFIFOSizeKB

[Input] 由使用者设定板卡内建 FIFO 的大小，单位为 KByte。建议设定值为 0 时，会自动依板卡最适合的 FIFO 大小来设定。

dwBufferSizeCount

[Input] 由用户设定模拟输入缓冲区数据笔数，此缓冲区将于配置于系统内存上之上作为暂存模拟数据值使用，设定值为 0 时，直接使用默认值 524288 笔 DWORD 大小的数据量约占用 2MB 的系统内存。

wCardType

[Input] 由用户依板卡规格作设定。如果您的板卡为低倍数板卡请设定为 0，若为高倍数板卡请设定为 1。此设定值将会影响将会影响擷取数据的精度及量测范围，详细设定请参考下列表格。

wCardType	PISO-813	PIO-821	PCI-1002 PEX-1002	PCI-1202 PEX-1202	PCI-1602	PCI-1802 PCI-1800	PCI-822 PCI-826
0	JP1 = 20 V	PIO-821L	PCI-1002LU PEX-1002L	PCI-1202LU	PCI-1602U	PCI-1802LU PCI-1800LU	PCI-822LU PCI-826LU
1	JP1 = 10V	PIO-821H	PCI-1002HU PEX-1002H	PCI-1202HU	PCI-1602FU	PCI-1802HU PCI-1800HU	-

wCardType	PCI-2602U	PCle-8620 PCle-8622
0	PCI-2602U	PCle-8620 PCle-8622
1	-	

表格 5-7 wCardType 参数设定

wDelaySettingTime

[Input] 板卡的模拟数字转换器的设定加权时间，单位为微秒(μs)。此时间的设定会影响到模拟输入的效能，建议设定值为 0。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ConfigAIEx

- 此函数用来设定板卡模拟输入功能的参数值，使用模拟输入函数集前必需先呼叫此函数。

- 语法

```
WORD Ixud_ConfigAIEx(  
    WORD wBoardNo,  
    WORD wFIFOSizeKB,  
    DWORD dwBufferSizeCount,  
    WORD wCardType,  
    WORD wDelaySettlingTime,  
    DWORD dwMode  
);
```

- 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wFIFOSizeKB

[Input] 由使用者设定板卡内建 FIFO 的大小，单位为 KByte。建议设定值为 0 时，会自动依板卡最适合的 FIFO 大小来设定。

dwBufferSizeCount

[Input] 由用户设定模拟输入缓冲区数据笔数，此缓冲区将于配置于系统内存上之上作为暂存模拟数据值使用，设定值为 0 时，直接使用默认值 524288 笔 DWORD 大小的数据量约占用 2MB 的系统内存。

wCardType

[Input] 由用户依板卡规格作设定。如果您的板卡为低倍数板卡请设定为 0，若为高倍数板卡请设定为 1。此设定值将会影响将会影响擷取数据的精度及量测范围，详细设定请参考下列表格。

wCardType	PISO-813	PIO-821	PCI-1002 PEX-1002	PCI-1202 PEX-1202	PCI-1602	PCI-1802 PCI-1800	PCI-822 PCI-826
0	JP1 = 20 V	PIO-821L	PCI-1002LU PEX-1002L	PCI-1202LU	PCI-1602U	PCI-1802LU PCI-1800LU	PCI-822LU PCI-826LU
1	JP1 = 10V	PIO-821H	PCI-1002HU PEX-1002H	PCI-1202HU	PCI-1602FU	PCI-1802HU PCI-1800HU	-

wCardType	PCI-2602U	PCle-8620 PCle-8622
0	PCI-2602U	PCle-8620 PCle-8622
1	-	

表格 5-8 wCardType 参数设定

wDelaySettingTime

[Input] 板卡的模拟数字转换器的设定加权时间，单位为微秒(μs)。此时间的设定会影响到模拟输入的效能，建议设定值为 0。

dwMode

[Input] The analog input data transfer mode.

dwMode	PCI-2602U PCle-8620 PCle-8622
ENABLEDMAAI	使用 DMA 传输

Table 5-9 dwMode Parameters setting

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ClearAIBuffer

清除系统内存里模拟输入缓冲区的数据。

➤ 语法

```
WORD Ixud_ClearAIBuffer(  
    WORD wBoardNo  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_GetBufferStatus

取得模拟输入缓冲区的状态代码及数据量。

➤ 语法

```
WORD Ixud_GetBufferStatus(  
    WORD wBoardNo,  
    WORD *wBufferStatus,  
    DWORD *dwDataCount  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wBufferStatus

[Output] 取得模拟缓冲区的使用状态。状态代码请参考下图。

<i>wBufferStatus</i>	状态简述
0	无任何资料。
1	正常。已有资料且未溢满。
2	缓冲区溢满。
3	系统未分配缓冲区。
4	FIFO 溢满。
5	其他异常状态。

表格 5-10 模拟缓冲区状态代码说明

dwDataCount

[Output] 取得模拟缓冲区的数据量(笔数)。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadAI

读取一笔的模拟输入信道内的电压值，单位为 volts。

➤ 语法

```
WORD Ixud_ReadAI(  
    WORD wBoardNo,  
    WORD wChannel,  
    WORD wConfig,  
    float *fValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定撷取模拟输入数据的信道号码。

wConfig

[Input] 由用户设定数据撷取模拟输入数据的配置码，不同型号的板卡因设计上的不同也会有不同的模拟输入范围，配置码及板卡支持的输入范围请查阅附录 A.3.1. 模拟输入配置码。此配置码将会影响撷取数据的精度及量测范围。

fValue

[Output] 读取模拟输入信道内的数据。单位为 volts。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadAIH

读取一笔的模拟输入通道内的数值，此值为十六进制。

➤ 语法

```
WORD Ixud_ReadAIH(  
    WORD wBoardNo,  
    WORD wChannel,  
    WORD wConfig,  
    DWORD *dwValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定撷取模拟输入数据的信道号码。

wConfig

[Input] 由用户设定数据撷取模拟输入数据的配置码，不同型号的板卡因设计上的不同也会有不同的模拟输入范围，配置码及板卡支持的输入范围请查阅附录 A.3.1. 模拟输入配置码。此配置码将会影响撷取数据的精度及量测范围。

dwValue

[Output] 读取模拟输入信道内的数据。此值为十六进制值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_PollingAI

此函式以**软件轮询**的方式取得模拟输入通道里复数笔数的数据。

➤ 语法

```
WORD Ixud_PollingAI(  
    WORD wBoardNo,  
    WORD wChannel,  
    WORD wConfig,  
    DWORD dwDataCount,  
    float fValue[]  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定撷取模拟输入数据的信道号码。

wConfig

[Input] 由用户设定数据撷取模拟输入数据的配置码，不同型号的板卡因设计上的不同也会有不同的模拟输入范围，配置码及板卡支持的输入范围请查阅附录 A.3.1. 模拟输入配置码。**此配置码将会影响撷取数据的精度及量测范围。**

dwDataCount

[Input] 由用户设定需要撷取的模拟输入数据笔数。

fValue[]

[Output] 请宣告一个浮点数数组(数组大小为 *dwDataCount*)，将轮询后得到的每笔模拟输入数据储存在此数组里，单位为 **volts**。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_PollingAIH

此函式以**软件轮询**的方式一次取得同一个信道里复数笔数的模拟输入值。

➤ 语法

```
WORD Ixud_PollingAIH(  
    WORD wBoardNo,  
    WORD wChannel,  
    WORD wConfig,  
    DWORD dwDataCount,  
    DWORD dwValue[ ]  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定撷取模拟输入数据的信道号码。

wConfig

[Input] 由用户设定数据撷取模拟输入数据的配置码，不同型号的板卡因设计上的不同也会有不同的模拟输入范围，配置码及板卡支持的输入范围请查阅附录 A.3.1. 模拟输入配置码。**此配置码将会影响撷取数据的精度及量测范围。**

dwDataCount

[Input] 由用户设定需要撷取的模拟输入数据笔数

dwValue[]

[Output] 请宣告一个 **DWORD** 数组(数组大小为 *dwDataCount*)，将轮询后得到的每笔模拟输入数据储存在此数组里，此数值为十六进制。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_PollingAIScan

此函式以**软件轮询**的方式取得**多个通道**里复数笔数的模拟输入数据，单位为 volts。

➤ 语法

```
WORD Ixud_PollingAIScan(  
    WORD wBoardNo,  
    WORD wChannels,  
    WORD wChannelList[ ],  
    WORD wConfigList[ ],  
    DWORD dwDataCountPerChannel,  
    float fValue[ ]  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannels

[Input] 由用户设定将撷取 *wChannels* 个信道撷取数据的配置值。

wChannelList[]

[Input] 一个 WORD 数组(数组大小等于 *wChannels*)。由用户设定欲使用来做撷取数据的模拟输入信道。请从数组第 0 个元素开始设定第一个模拟输入信道号码。

wConfigList[]

[Input] 请宣告一个 WORD 数组(数组大小至少大于 *wChannels*)。由用户设定每个信道的配置值，从数组第 0 个元素开始设定第一个模拟输入信道的配置码。配置值代码请参考 A.3.1. 模拟输入配置码。

dwDataCountPerChannel

[Input] 设定**每一个通道**的取样数量。

fValue[]

[Output] 请宣告一个浮点数数组，数组大小为 **wChannels** 乘以 **dwDataCountPerChannel**。将轮询用户设定的每个通道后所得到的每笔模拟输入数据储存在此数组里，此数据为浮点数，此数组储存排列方式请参考表格 5-11。

➤ 回传值

请参考 A.1. 函数回传值定义。

➤ 使用范例

```
DWORD dwDataCountPerChannel = 2 //每个通道撷取2个模拟数据
wChannels = 3 //总共使用3个通道
float fValue[dwDataCounterPerChannel*wChannels]; //宣告一个2 x3大小的模拟数据数组
wChannelList[0]= 5 //第1次撷取模拟输入信道5的数据
wChannelList[1]= 3 //第2次撷取模拟输入信道3的数据
wChannelList[2]= 6 //第3次撷取模拟输入信道6的数据
wConfigList[0]= IXUD_BI_10V //第1次撷取模拟输入信道5的数据量测范围为+/-10V
wConfigList[1]= IXUD_BI_5V //第2次撷取模拟输入信道3的数据量测范围为+/-5V
wConfigList[2]= IXUD_BI_2V5 //第3次撷取模拟输入信道6的数据量测范围为+/-2.5V
wRtn = lxud_PollingAIScan(wBoardNo, wChannels, wChannelList, wConfigList, dwDataCountPerChannel,
fValue)
```

呼叫函式后各个信道的输入值储存至数组(*fValue[]*)里，而储存方式如下图：

0	Channel 5	Value 0
1	Channel 3	Value 0
2	Channel 6	Value 0
3	Channel 5	Value 1
4	Channel 3	Value 1
5	Channel 6	Value 1

表格 5-11 数组储存方式

Ixud_PollingAIScanH

此函式以**软件轮询**的方式一次取得**多个信道**里复数笔数的模拟输入值，此值为十六进值。

➤ 语法

```
WORD Ixud_PollingAIScanH(  
    WORD wBoardNo,  
    WORD wChannels,  
    WORD wChannelList[ ],  
    WORD wConfigList[ ],  
    DWORD dwDataCountPerChannel,  
    DWORD dwValue[ ]  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannels

[Input] 由用户设定将撷取 *wChannels* 个信道撷取数据的配置值。

wChannelList[]

[Input] 一个 WORD 数组(数组大小等于 *wChannels*)。由用户设定欲使用来做撷取数据的模拟输入信道。请从数组第 0 个元素开始设定第一个模拟输入信道号码。

wConfigList[]

[Input] 请宣告一个 WORD 数组(数组大小至少大于 *wChannels*)。由用户设定每个信道的配置值，从数组第 0 个元素开始设定第一个模拟输入信道的配置码。配置值代码请参考 A.3.1. 模拟输入配置码。

dwDataCountPerChannel

[Input] 设定**每一个通道**的取样数量。

dwValue[]

[Output] 请宣告一个双倍无号整数数组，数组大小至少大于 **wChannels** 乘以 **dwDataCountPerChannel**。将轮询用户设定的每个通道后所得到的每笔模拟输入数据储存在此数组里，此数据为十六进值，此数组储存排列方式请参考表格 5-12。

➤ 回传值

请参考 A.1. 函数回传值定义。

➤ 使用范例

```
DWORD dwDataCountPerChannel = 2 //每个通道撷取2个模拟数据
wChannels = 3 //总共使用3个通道
DWORD dwValue[dwDataCounterPerChannel*wChannels]; //宣告一个2 x3大小的模拟数据数组
wChannelList[0] = 5 //第1次撷取模拟输入信道5的数据
wChannelList[1] = 3 //第2次撷取模拟输入信道3的数据
wChannelList[2] = 6 //第3次撷取模拟输入信道6的数据
wConfigList[0] = IXUD_BI_10V //第1次撷取模拟输入信道5的数据量测范围为+/-10V
wConfigList[1] = IXUD_BI_5V //第2次撷取模拟输入信道3的数据量测范围为+/-5V
wConfigList[2] = IXUD_BI_2V5 //第3次撷取模拟输入信道6的数据量测范围为+/-2.5V
```

呼叫函式后各个信道的输入值储存至数组(dwValue[])里，而储存方式如下表：

0	Channel 5	Val0
1	Channel 3	Val0
2	Channel 6	Val0
3	Channel 5	Val1
4	Channel 3	Val1
5	Channel 6	Val1

表格 5-12 数组储存方式

Ixud_StartAI

透过此函式来初始化 ADC、设定取样频率、采样数据量，并开始启动模拟数据撷取，启动后将以等速度撷取单一信道的模拟数据并将数据暂存至系统内存内，可透过 `Ixud_GetAIBuffer` 函式或 `Ixud_GetAIBufferH` 函式取出模拟输入数据，欲停止模拟输入采集功能时，需呼叫 `Ixud_StopAI` 函式。



为了确保数据采集的准确度，使用此函式在数据采集的过程中将会占用 CPU 一段短暂的时间至采集数据完成，此时间将取决于用户设定的采集数据量及取样频率。

➤ 语法

```
WORD Ixud_StartAI(  
    WORD wBoardNo,  
    WORD wChannel,  
    WORD wConfig,  
    float fSamplingRate,  
    DWORD dwDataCount  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

wChannel

[Input] 由用户设定撷取模拟输入数据的信道号码。

wConfig

[Input] 由用户设定数据撷取模拟输入数据的配置码，不同型号的板卡因设计上的不同也会有不同的模拟输入范围，配置码及板卡支持的输入范围请查阅附录 A.3.1. 模拟输入配置码。此配置码将会影响撷取数据的精度及量测范围。

fSamplingRate

[Input] 为一浮点数，由用户设定模拟输入信道的取样频率(次/秒)。

dwDataCount

[Input] 由用户设定需要采集的数据笔数。当 `dwDataCount` 设定为 0 时，此时将会启动连续摄取模式，除非用户使用 `Ixud_StopAI` 函式停止采集。



使用连续摄取模式时的注意事项

1. 需注意频率的大小，如果取样频率设定过大，由于采集数据量变大，将会很容易造成 FIFO 的溢满。
2. 连续摄取模式中会将数据暂存在内存之中，用户在适当的时间应将数据取出，否则随着时间的增长内存缓冲区将会溢满。
3. 采集过程中请尽量缩短对系统负荷过重的动作，例如档案处理等等，否则将会提高忆体缓冲区溢满的机率。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StartAIScan

透过此函式来初始化 ADC、设定取样频率、采样数据量，并开始启动模拟数据撷取，启动后将以等速度撷取多个信道的模拟数据并将数据暂存至系统内存内，可透过 `Ixud_GetAIBuffer` 函式或 `Ixud_GetAIBufferH` 函式取出模拟输入数据，欲停止模拟输入采集功能时，需呼叫 `Ixud_StopAI` 函式。



为了确保数据采集的准确度，使用此函式在数据采集的过程中时将会占用 CPU 一段短暂的时间至采集数据完成，此时间将取决于用户设定的采集数据量及取样频率。

➤ 语法

```
WORD Ixud_StartAIScan(  
    WORD wBoardNo,  
    WORD wChannels,  
    WORD wChannelList[ ],  
    WORD wConfigList[ ],  
    float fSamplingRate,  
    DWORD dwDataCountPerChannel  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

wChannels

[Input] 由用户设定将撷取 `wChannels` 个信道撷取数据的配置值。

wChannelList[]

[Input] 一个 **WORD** 数组(数组大小等于 `wChannels`)。由用户设定欲使用来做撷取数据的模拟输入信道。请从数组第 0 个元素开始设定第一个模拟输入信道号码。

wConfigList[]

[Input] 请宣告一个 **WORD** 数组(数组大小至少大于 `wChannels`)。由用户设定每个信道的配置值，从数组第 0 个元素开始设定第一个模拟输入信道的配置码。配置值代码请参考 A.3.1. 模拟输入配置码。

fSamplingRate

[Input] 为一浮点数，由用户设定模拟输入信道的取样频率(次/秒)。

dwDataCountPerChannel

[Input] 设定**每一个通道**的取样数量。设定值为 0，将会启动连续撷取模式，除非用户使用 `Ixud_StopAI` 函式停止采集。



使用连续撷取模式时的注意事项

1. 需注意频率的大小，如果取样频率设定过大，由于采集数据量变大，将会很容易造成 FIFO 的溢满。
2. 连续撷取模式中会将数据暂存在内存之中，用户在适当的时间应将数据取出，否则随着时间的增长内存缓冲区将会溢满。
3. 采集过程中请尽量缩短对系统负荷过重的动作，例如档案处理等等，否则将会提高忆体缓冲区溢满的机率。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StartExtAI

透过此函式来初始化 ADC、设定取样频率、采样数据量，并开始启动外部触发 (TTL level) 模拟数据撷取，支持三种触发模式：post-trigger、middle-trigger 及 pre-trigger 启动后将以等速度撷取单一信道的模拟数据并将数据暂存至系统内存内，可透过 Ixud_GetAIBuffer 函式或 Ixud_GetAIBufferH 函式取出模拟输入数据，欲停止模拟输入采集功能时，需呼叫 Ixud_StopAI 函式。



为了确保数据采集的准确度，使用此函式在数据采集的过程中时将会占用 CPU 一段短暂的时间至采集数据完成，此时间将取决于用户设定的采集数据量及取样频率。

➤ 语法

```
WORD Ixud_StartExtAI(  
    WORD wBoardNo,  
    WORD wActive,  
    WORD wChannel,  
    WORD wConfig,  
    float fSamplingRate,  
    DWORD dwPostDataCount  
    DWORD dwPreDataCount  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wActive

[Input] 设定外部触发条件。

<i>dwActive</i>	PCI-822LU PCI-826LU	PCI-2602U PCIe-8622
0	下降缘触发	任意
1	上升缘触发	任意

wChannel

[Input] 由用户设定撷取模拟输入数据的信道号码。

wConfig

[Input] 由用户设定数据摄取模拟输入数据的配置码，不同型号的板卡因设计上的不同也会有不同的模拟输入范围，配置码及板卡支持的输入范围请查阅附录 A.3.1. 模拟输入配置码。此配置码将会影响摄取数据的精度及量测范围。

fSamplingRate

[Input] 为一浮点数，由用户设定模拟输入信道的取样频率(次/秒)。

dwPostDataCount

[Input] 由用户设定外部触发后需要的数据笔数。

dwPreDataCount

[Input] 由用户设定外部触发前需要的数据笔数。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StartExtAnalogTrigger

透过此函式来初始化 ADC、设定取样频率、采样数据量，并开始启动外部触发(模拟信号)模拟数据撷取，启动后将以等速度撷取单一信道的模拟数据并将数据暂存至系统内存内，可透过 `Ixud_GetAIBuffer` 函式或 `Ixud_GetAIBufferH` 函式取出模拟输入数据，欲停止模拟输入采集功能时，需呼叫 `Ixud_StopAI` 函式。



本函式仅支援 PCI-2602U



为了确保数据采集的准确度，使用此函式在数据采集的过程中时将会占用 CPU 一段短暂的时间至采集数据完成，此时间将取决于用户设定的采集数据量及取样频率。

➤ 语法

WORD `Ixud_StartExtAnalogTrigger`(

WORD `wBoardNo`,

WORD `wActive`,

WORD `wChannel`,

WORD `wConfig`,

float `fSamplingRate`,

DWORD `dwDataCount`,

DWORD `dwReserved`,

float `fAboveTrgVoltage`,

float `fBelowTrgVoltage`

);

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

wActive

[Input] 设定外部触发条件。

<i>dwActive</i>	PCI-2602U
IXUD_ANALOGTRIGGER_ABOVE	Above High
IXUD_ANALOGTRIGGER_BELOW	Below Low
IXUD_ANALOGTRIGGER_LEAVE	Leave Region
IXUD_ANALOGTRIGGER_ENTRY	Entry Region

wChannel

[Input] 由用户设定截取模拟输入数据的信道号码。

wConfig

[Input] 由用户设定数据截取模拟输入数据的配置码，不同型号的板卡因设计上的不同也会有不同的模拟输入范围，配置码及板卡支持的输入范围请查阅附录 A.3.1. 模拟输入配置码。此配置码将会影响截取数据的精度及量测范围。

fSamplingRate

[Input] 为一浮点数，由用户设定模拟输入信道的取样频率(次/秒)。

dwDataCount

[Input] 由用户设定外部触发后需要的数据笔数。

dwReserved

[Input] 保留。

fAboveTrgVoltage

[Input] 设定 Above trigger 电压范围。

fBelowTrgVoltage

[Input] 设定 Below trigger 电压范围。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StartExtAIScan

透过此函式来初始化 ADC、设定取样频率、采样数据量，并开始启动外部触发(模拟信号)模拟数据撷取，启动后将以等速度撷取多个信道的模拟数据并将数据暂存至系统内存内，可透过 `Ixud_GetAIBuffer` 函式或 `Ixud_GetAIBufferH` 函式取出模拟输入数据，欲停止模拟输入采集功能时，需呼叫 `Ixud_StopAI` 函式。



为了确保数据采集的准确度，使用此函式在数据采集的过程中将会占用 CPU 一段短暂的时间至采集数据完成，此时间将取决于用户设定的采集数据量及取样频率。

➤ 语法

```
WORD Ixud_StartExtAIScan(  
    WORD wBoardNo,  
    WORD wChannels,  
    WORD wActive,  
    WORD wChannelList[ ],  
    WORD wConfigList[ ],  
    float fSamplingRate,  
    DWORD dwPostDataCountPerChannel,  
    DWORD dwPreDataCountPerChannel  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

wChannels

[Input] 由用户设定将撷取 `wChannels` 个信道撷取数据的配置值。

wActive

[Input] 设定外部触发条件。

<i>dwActive</i>	PCI-822LU PCI-826LU	PCI-2602U PCIe-8622
0	下降缘触发	任意
1	上升缘触发	任意

wChannelList[]

[Input] 一个 WORD 数组(数组大小等于 *wChannels*)。由用户设定欲使用来做摄取数据的模拟输入信道。请从数组第 0 个元素开始设定第一个模拟输入信道号码。

wConfigList[]

[Input] 请宣告一个 WORD 数组(数组大小至少大于 *wChannels*)。由用户设定每个信道的配置值，从数组第 0 个元素开始设定第一个模拟输入信道的配置码。配置值代码请参考 A.3.1. 模拟输入配置码。

fSamplingRate

[Input] 为一浮点数，由用户设定模拟输入信道的取样频率(次/秒)。

dwPostDataCountPerChannel

[Input] 由用户设定每一个通道外部触发后需要的数据笔数。

dwPreDataCountPerChannel

[Input] 由用户设定每一个通道外部触发前需要的数据笔数。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_GetAIBuffer

此函式将会从内存里取得模拟输入值，并储存在用户所指令的数组里，其值为浮点数。呼叫此函式前需先呼叫 `Ixud_StartAI`、`Ixud_StartAIScan`、`Ixud_StartExtAI` 及 `Ixud_StartExtAIScan` 函式。

➤ 语法

```
WORD Ixud_GetAIBuffer(  
    WORD wBoardNo,  
    DWORD dwDataCount,  
    float fValue[ ]  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

dwDataCount

[Input] 由用户设定需要的数据笔数。

fValue[]

[Output] 请宣告一个浮点数数组。用来取得从模拟输入数据值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_GetAIBufferH

此函式将会从内存里取得模拟输入值，并储存在用户所指令的数组里，其值为十六进制值。呼叫此函式前需先呼叫 `Ixud_StartAI`、`Ixud_StartAIScan`、`Ixud_StartExtAI` 及 `Ixud_StartExtAIScan` 函式。

➤ 语法

```
WORD Ixud_GetAIBufferH(  
    WORD wBoardNo,  
    DWORD dwDataCount,  
    DWORD hValue[ ]  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 `wBoardNo` 为 0，第二张板卡的 `wBoardNo` 为 1，依此类推。

dwDataCount

[Input] 由用户设定需要的数据笔数。

hValue[]

[Output] 请宣告一个 **DWORD** 数组。用来取得从模拟输入值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StopAI

呼叫此函式，用来停止模拟输入擷取的运作。

➤ 语法

```
WORD Ixud_StopAI(  
    WORD wBoardNo  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

➤ 回传值

请参考 A.1. 函数回传值定义。

5.2.5. 模拟输出函数集

Ixud_ConfigAO

此函数用来设定模拟输出的参数值，使用模拟输出函数集前必需先呼叫此函数。

➤ 语法

```
WORD Ixud_ConfigAO(  
    WORD wBoardNo,  
    WORD wChannel,  
    WORD wCfgCode  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定模拟输出信道的号码。

wCfgCode

[Input] 由用户设定模拟输出信道的配置码，配置码的代码请查阅附录 A.3.2. 模拟输出配置码(电压)及 A.3.3. 模拟输出配置码(电流)。此配置码将会影响模拟输出的精度及输出范围。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WriteAOVoltage

呼叫此函式控制模拟输出信道输出一固定的电压，此电压设定值为浮点数。

➤ 语法

```
WORD Ixud_WriteAOVoltage(  
    WORD wBoardNo,  
    WORD wChannel,  
    float fValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wChannel

[Input] 由用户设定模拟输出信道的号码。

fValue

[Input] 由用户设定模拟输出电压值，此值为浮点数，单位为伏特(V)。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WriteAOVoltageH

呼叫此函式控制模拟输出信道输出一固定的电压，此电压设定值为十六进制值。

➤ 语法

```
WORD Ixud_WriteAOVoltageH(  
    WORD wBoardNo,  
    WORD wChannel,  
    DWORD hValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wChannel

[Input] 由用户设定模拟输出信道的号码。

hValue

[Input] 由用户设定模拟输出电压值，此值为十六进制值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WriteAOCurrent

呼叫此函式控制模拟输出通道输出一固定的电流，此电流设定值为浮点数。

➤ 语法

```
WORD Ixud_WriteAOCurrent(  
    WORD wBoardNo,  
    WORD wChannel,  
    float fValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wChannel

[Input] 由用户设定模拟输出信道的号码。

fValue

[Input] 由使用者设定模拟输出电流值，此值为浮点数，单位为 mA。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WriteAOCurrentH

呼叫此函式控制模拟输出通道输出一固定的电流，此电流设定值为十六进制值。

➤ 语法

```
WORD Ixud_WriteAOCurrentH(  
    WORD wBoardNo,  
    WORD wChannel,  
    DWORD hValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wChannel

[Input] 由用户设定模拟输出信道的号码。

hValue

[Input] 由使用者设定模拟输出电流值，此值为十六进制值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StartAOVoltage

依用户指定的数据大小、数据缓冲区及循环模式启动高速模拟输出模式，模拟输出缓冲区使用浮点数电压。



本函式仅支援 PCI-2602U

➤ 语法

WORD *Ixud_StartAOVoltage*(

WORD *wBoardNo*,

WORD *wChannel*,

WORD *wCfgCode*,

float *fFrequency*,

DWORD *dwDataCount*,

DWORD *dwCycleNum*,

float *fAOBuf* []

);

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定模拟输出信道的号码。

wCfgCode

[Input] 由用户设定模拟输出信道的配置码，配置码的代码请查阅附录 A.3.2. 模拟输出配置码(电压)及 A.3.3. 模拟输出配置码(电流)。此配置码将会影响模拟输出的精度及输出范围。

fFrequency

[Input] 为一浮点数，由用户设定数字输出端口的输出每笔模拟数据的频率(次/秒)。

dwDataCount

[Input] 由使用者设定需要输出一个波型的数据笔数。使用函式停止采集。

dwCycleNum

[Input] 0:连续模式, 如果要终止高速模拟输出功能, 请使用 `Ixud_StopAO` 函式。

fAOBuf[]

[Input] 储存模拟输出数据(浮点数电压)至数字输出缓冲区。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StartAOVoltageH

依用户指定的数据大小、数据缓冲区及循环模式启动高速模拟输出模式, 模拟输出缓冲区使用十六进制值。



本函式仅支援 PCI-2602U

➤ 语法

WORD `Ixud_StartAOVoltageH`(

WORD `wBoardNo`,

WORD `wChannel`,

WORD `wCfgCode`,

float `fFrequency`,

DWORD `dwDataCount`,

DWORD `dwCycleNum`,

DWORD `dwAOBuf[]`)

);

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号, 第一张板卡的 `wBoardNo` 为 0, 第二张板卡的 `wBoardNo` 为 1, 依此类推。

wChannel

[Input] 由用户设定模拟输出信道的号码。

wCfgCode

[Input] 由用户设定模拟输出信道的配置码，配置码的代码请查阅附录 A.3.2. 模拟输出配置码(电压)及 A.3.3. 模拟输出配置码(电流)。此配置码将会影响模拟输出的精度及输出范围。

fFrequencye

[Input] 为一浮点数，由用户设定数字输出端口的输出每笔模拟数据的频率(次/秒)。

dwDataCount

[Input] 由使用者设定需要输出一个波型的数据笔数。使用函式停止采集。

dwCycleNum

[Input] 0:连续模式，如果要终止高速模拟输出功能，请使用 `Ixud_StopAO` 函式。

dwAOBuf[]

[Input] 储存模拟输出数据(十六进制值)至数字输出缓冲区。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_StopAO

呼叫此函式，用来停止高速模拟输出的运作。



本函式仅支援 PCI-2602U

➤ 语法

```
WORD Ixud_StopAO(  
    WORD wBoardNo,  
    WORD wChannel  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wChannel

[Input] 由用户设定模拟输出信道的号码。

➤ 回传值

请参考 A.1. 函数回传值定义。

5.2.6. 计时计数函数集

Ixud_DisableCounter

关闭(停止)计数器通道。

➤ 语法

```
WORD Ixud_DisableCounter(  
    WORD wBoardNo,  
    WORD wChannel  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定计数器通道的编号，第一个通道 *wChannel* 为 0，第二个通道 *wChannel* 为 1，依此类推。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadCounter

呼叫此函式可读取计数器通道的计数值。

➤ 语法

```
WORD Ixud_ReadCounter(  
    WORD wBoardNo,  
    WORD wChannel,  
    DWORD *dwValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 **wBoardNo** 为 0，第二张板卡的 **wBoardNo** 为 1，依此类推。

wChannel

[Input] 由用户设定计数器通道的编号，第一个通道 **wChannel** 为 0，第二个通道 **wChannel** 为 1，依此类推。

dwValue

[Output] 取得计数器的计数值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadFrequency

读取信号源的频率。



本函式仅支援 PCI-FC16U

➤ 语法

```
WORD Ixud_ReadFrequency(  
    WORD wBoardNo,  
    WORD wChannel,  
    float *fFrequency,  
    DWORD dwTimeOutMs,  
    WORD *wStatus  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定计数器通道的编号，第一个通道 *wChannel* 为 0，第二个通道 *wChannel* 为 1，依此类推。

fFrequency

[Output] 储存从信号源读取出来的频率，单位为 Hz。

dwTimeOutMs

[Input] 由使用者设定计数器的逾时时间，单位为 ms。

wStatus

[Output] 取得计数器通道的状态。

<i>wStatus</i>	状态描述
0	取得频率中
1	逾时
2	闪烁住频率

表格 5-13 *wStatus* 参数设定

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_SetCounter

设定计数器通道的计数值及计数模式。

➤ 语法

```
WORD Ixud_SetCounter(  
    WORD wBoardNo,  
    WORD wChannel,  
    WORD wMode,  
    DWORD dwValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定计数器通道的编号，第一个通道 *wChannel* 为 0，第二个通道 *wChannel* 为 1，依此类推。

wMode

[Input] 由用户设定计数器信道的计数模式。计数模式的详细说明请参考 Intel 8254 Datasheet。

<i>wMode</i>	模式名称
0	数完时中断
1	可程序单击闸触发
2	比率产生器
3	方波产生器
4	软件触发摄取
5	硬件触发摄取

表格 5-14 *wMode* 参数设定

dwValue

[Input] 由用户设定计数器通道的计数值。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_SetFCChannelMode

设定每个信道的计数模式。



本函式仅支援 PCI-FC16U

➤ 语法

WORD *Ixud_SetFCChannelMode*(

WORD *wBoardNo*,

WORD *wChannel*,

WORD *wMode*,

WORD *wDelayMs*

);

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

wChannel

[Input] 由用户设定计数器通道的编号，第一个通道 *wChannel* 为 0，第二个通道 *wChannel* 为 1，依此类推。

wMode

[Input] 由用户设定计数器信道的计数模式。

<i>wMode</i>	模式名称
0	-
1	-
2	倒数模式(down count)
3	-
4	-
5	-

表格 5-15 *wMode* 参数设定

wDelayMs

[Input] 由使用者设定计数器的延迟时间，此时间的长短取决于用户欲量的频率的范围，单位为 **ms**。

➤ 回传值

请参考 A.1. 函数回传值定义。

5.2.7. 内存输出输入函数集

Ixud_ReadMemory

从用户所指定的内存地址里读取出 8/16/32 bit 的数值。

➤ 语法

```
WORD Ixud_ReadMemory(  
    WORD wBoardNo,  
    DWORD dwOffsetByte,  
    WORD wSize,  
    DWORD *dwValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

dwOffsetByte

[Input] 由用户指定欲读取内存的位移地址。

wSize

[Input] 由用户设定读取数据的长度。

<i>wSize</i>	长度
8	8-bit
16	16-bit
32	32-bit

表格 5-16 *wSize* 参数设定

dwValue

[Output] 读取内存内的数据。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WriteMemory

写入 8/16/32 bit 的数值至用户所指定的内存地址里。

➤ 语法

```
WORD Ixud_WriteMemory(  
    WORD wBoardNo,  
    DWORD dwOffsetByte,  
    WORD wSize,  
    DWORD dwValue  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

dwOffsetByte

[Input] 由用户指定欲写入内存的位移地址。

wSize

[Input] 由用户设定写入数据的长度。

<i>wSize</i>	长度
8	8-bit
16	16-bit
32	32-bit

表格 5-17 *wSize* 参数设定

dwValue

[Input] 写入数据至内存内。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_ReadMemory32

从用户所指定的内存地址里读取出 32bit 值，此函式支持不带有无号整数型态的编辑器，如 Visual Basic 6.0。

➤ 语法

```
WORD Ixud_ReadMemory32(  
    WORD wBoardNo,  
    DWORD dwOffsetByte,  
    DWORD *dwLow,  
    DWORD *dwHigh  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

dwOffsetByte

[Input] 设定读取内存的位移地址。

dwLow

[Output] 读取内存内 Bit 0~15 的数据。

dwHigh

[Output] 读取内存内 Bit 16~31 的数据。

➤ 回传值

请参考 A.1. 函数回传值定义。

Ixud_WriteMemory32

从内存地址里写入一个 32bit 的值，此函式支持不带有无号整数型态的编辑器，如 **Visual Basic 6.0**。

➤ 语法

```
WORD Ixud_WriteMemory32(  
    WORD wBoardNo,  
    DWORD dwOffsetByte,  
    DWORD dwLow,  
    DWORD dwHigh  
);
```

➤ 参数

wBoardNo

[Input] 由使用者指定的板卡编号，第一张板卡的 *wBoardNo* 为 0，第二张板卡的 *wBoardNo* 为 1，依此类推。

dwOffsetByte

[Input] 设定写入内存的位移地址。

dwLow

[Input] 设定写入内存的 bit 0~15 的数据。

dwHigh

[Input] 设定写入内存的 bit16~31 的数据。

➤ 回传值

请参考 A.1. 函数回传值定义。

5.3. 数据型态

PIXUD_DEVICE_INFO

➤ 语法

```
typedef struct _IXUD_DEVICE_INFO_  
{  
    DWORD dwSize;  
    WORD wVendorID;  
    WORD wDeviceID;  
    WORD wSubVendorID;  
    WORD wSubDeviceID;  
    DWORD dwBAR[6];  
    UCHAR BusNo;  
    UCHAR DevNo;  
    UCHAR IRQ;  
    UCHAR Aux;  
    DWORD dwBarVirtualAddress [6];  
}IXUD_DEVICE_INFO,*PIXUD_DEVICE_INFO;
```

➤ 成员

dwSize

[Output] 取得此数据结构的大小，单位为 byte。

wVendorID

[Output] 取得板卡的 Vendor ID。

wDeviceID

[Output] 取得板卡的 Device ID。

wSubVendorID

[Output] 取得板卡的 Sub Vendor ID。

wSubDeviceID

[Output] 取得板卡的 Sub Device ID。

dwBAR[]

[Output] 取得板卡的 Base Address。

Base Address	dwBAR [Index]
Bar 0	dwBAR[0]
Bar 1	dwBAR[1]
Bar 2	dwBAR[2]
Bar 3	dwBAR[3]
Bar 4	dwBAR[4]
Bar 5	dwBAR[5]

BusNo

[Output] 取得板卡的 Bus 号码。

DevNo

[Output] 取得板卡的 Device 号码。

IRQ

[Output] 取得板卡的 IRQ 号码。

Aux

[Output] 取得板卡的 Aux 号码。

dwBarVirtualAddress []

[Output] 取得 memory mapping I/O 虚拟内存地址。

Virtual Memory Address	dwBAR [Index]
Bar 0	dwBarVirtualAddress [0]
Bar 1	dwBarVirtualAddress [1]
Bar 2	dwBarVirtualAddress [2]
Bar 3	dwBarVirtualAddress [3]
Bar 4	dwBarVirtualAddress [4]
Bar 5	dwBarVirtualAddress [5]

PIXUD_CARD_INFO

➤ 语法

```
typedef struct _IXUD_CARD_INFO_  
{  
    DWORD dwSize;  
    DWORD dwModelNo;  
    UCHAR CardID;  
    UCHAR wSingleEnded;  
    WORD wAIOResolution;  
    WORD wAIChannels;  
    WORD wAOChannels;  
    WORD wDIPorts;  
    WORD wDOPorts;  
    WORD wDIOPorts;  
    WORD wDIOPortWidth;  
    WORD wCounterChannels;  
    WORD wMemorySize;  
    DWORD dwReserved1[6];  
}IXUD_CARD_INFO,*PIXUD_CARD_INFO;
```

➤ 成员

dwSize

[Output] 取得此数据结构的大小，单位为 byte。

dwModelNo

[Output] 取得板卡的模块识别号码，模块识别号码可参考 A.2. 模块识别号码。

CardID

[Output] 取得板卡的卡片识别号码，若取得数值为 255 代表板卡未支持此功能。

wSingleEnded

[Output] 取得板卡模拟输入接线设定值，设定值请参考下面表格。

数值	数值(Hex)	接线设定
1	1	单端式(SE)
2	2	差动式(DIFF)
255	FF	未支持此设定

wAIOResolution

[Output] 取得板卡模拟输出分辨率，高位为模拟输入信道分辨率((*wAIOResolution*>>8)&0xFF)，低位为模拟输出信道分辨率(*wAIOResolution*&0xFF)。

数值	数值(Hex)	分辨率
12	C	12-bit
14	E	14-bit
16	10	16-bit

wAIChannels

[Output] 取得板卡模拟输入的通道数量。

wAOChannels

[Output] 取得板卡模拟输出的通道数量。

wDIPorts

[Output] 取得板卡单向数字输入端口的端口数量。

wDOPorts

[Output] 取得板卡单向数字输出端口的端口数量。

wDIOPorts

[Output] 取得板卡双向数字输入输出端口的端口数量。

wDIOPortWidth

[Output] 取得板卡数字输出端口的宽度。

数值	宽度
8	8-bit
16	16-bit
32	32-bit

wCounterChannels

[Output] 取得板卡计时计数的通道数量。

wMemorySize

[Output] 取得板卡的内建内存大小，单位为 kByte。

dwReserved1[]

[Output] 取得保留信息。

A

附录 A. 函式回传值与配置码

本章节列出了泓格 UniDAQ 驱动函式库所返回的状态回传值及配置设定代码。

A.1. 函数回传值定义

呼叫所有的函式，都会回传出一个整数值。透过这个值可以得知该函数运作的状况，下表提供每个回传值所代表的定义。

回传值	定义	说明
0	Ixud_NoErr	正确
1	Ixud_OpenDriverErr	开启驱动程序发生错误
2	Ixud_PnPDriverErr	Plug&Play 时发生错误
3	Ixud_DriverNoOpen	驱动程序未开启
4	Ixud_GetDriverVersionErr	取得驱动程序版本错误
5	Ixud_ExceedBoardNumber	板卡号码错误
6	Ixud_FindBoardErr	找不到任何的板卡
7	Ixud_BoardMappingErr	板卡对象索引(Board Mapping)错误
8	Ixud_DIOModesErr	数字输出输入模式设定错误
9	Ixud_InvalidAddress	不合法的地址
10	Ixud_InvalidSize	不合法的大小
11	Ixud_InvalidPortNumber	不合法的埠号
12	Ixud_UnSupportedModel	未支援此板卡
13	Ixud_UnSupportedFun	未支援此函式
14	Ixud_InvalidChannelNumber	不合法的信道号码
15	Ixud_InvalidValue	不合法的值
16	Ixud_InvalidMode	不合法的模式
17	Ixud_GetAIStatusTimeOut	取得模拟输入状态超时
18	Ixud_TimeOutErr	超过时间发生异常
19	Ixud_CfgCodeIndexErr	找不到适合的配置码表格索引
20	Ixud_ADCCTLTimeoutErr	ADC 控制器愈期
21	Ixud_FindPCIIndexErr	找不到适合的 PCI 表格索引值
22	Ixud_InvalidSetting	不合法的设定值
23	Ixud_AllocateMemErr	分配内存空间失败
24	Ixud_InstallEventErr	安装中断事件失败
25	Ixud_InstallIrqErr	安装中断失败
26	Ixud_RemoveIrqErr	移除中断失败
27	Ixud_ClearIntCountErr	清除中断计数数量失败
28	Ixud_GetSysBufferErr	取得系统缓冲区失败
29	Ixud_CreateEventErr	CreateEvent 失败
30	Ixud_UnSupportedResolution	未支持此分辨率
31	Ixud_CreateThreadErr	CreateThread 失败

32	Ixud_ThreadTimeOutErr	线程超时
33	Ixud_FIFOOverFlowErr	FIFO 溢满
34	Ixud_FIFOTimeOutErr	FIFO 超时
35	Ixud_GetIntInstStatus	取得中断安装状态
36	Ixud_GetBufStatus	取得 SYS 缓冲区状态
37	Ixud_SetBufCountErr	设定缓冲区大小错误
38	Ixud_SetBufInfoErr	设定缓冲区数据错误
39	Ixud_FindCardIDErr	找不到卡片标识符
40	Ixud_EventThreadErr	事件执行绪错误
41	Ixud_AutoCreateEventErr	自动创建事件失败
42	Ixud_RegThreadErr	注册线程失败
43	Ixud_SearchEventErr	寻找事件失败
44	Ixud_FifoResetErr	FIFO 清除失败
45	Ixud_InvalidBlock	不合法的 EEPROM 区块
46	Ixud_InvalidAddr	不合法的 EEPROM 地址
47	Ixud_AcquireSpinLock	获得旋转锁失败
48	Ixud_ReleaseSpinLock	释放旋转锁失败
49	Ixud_SetControlErr	模拟输入设定错误
50	Ixud_InvalidChannels	不合法的通道数
51	Ixud_SearchCardErr	不合法的板卡号码
52	Ixud_SetMapAddressErr	设定映像地址失败
53	Ixud_ReleaseMapAddressErr	释放映像地址失败
54	Ixud_InvalidOffset	不合法的位移
55	Ixud_ShareHandleErr	开启 Share Memory 失败
56	Ixud_InvalidDataCount	不合法的数据量
57	Ixud_WriteEEPErr	写入 EEPROM 失败
58	Ixud_CardIOErr	使用 CardIO 失败
59	Ixud_IOErr	使用 MemoryIO 失败
60	Ixud_SetScanChannelErr	设置 channel scan number 失败
61	Ixud_SetScanConfigErr	设置 channel scan configuration 失败
62	Ixud_GetMMIOMapStatus	取得 Memory Mapping IO 状态失败

A.2. 模块识别号码

标识符	值(十六进制)	支持的数据摄取板卡
PIOD56	800140	PIO-D24/D56/D24U/D56U
PEXD56	800140	PEX-D24/D56
PIOD48	800130	PIO-D48/D48U/D48SU
PEXD48	800130	PEX-D48
PIOD64	800120	PIO-D64/D64U
PIOD96	800110	PIO-D96/D96U/D96SU
PEXD96	800110	PEX-D96S
PIOD144	800100	PIO-D144
PEXD144	800100	PEX-D144LS
PIOD168	800150	PIO-D168
PIODA	800400	PIO-DA4/DA8/DA16/DA4U/DA8U/DA16U/PISO-DA4U/DA8U/DA16U
PEXDA	800400	PEX-DA4/DA8/DA16
PIO821	800310	PIO-821 L/H/LU/HU
PISOP16R16U	1800FF	PISO-P16R16U/P16R16E
PEXP16R16	1800FF	PEX-P16R16i
PEXP8R8	1800FF	PEX-P8R8i
PISOC64	800800	PISO-C64
PEXC64	800800	PEX-C64
PISOP64	800810	PISO-P64
PEXP64	800810	PEX-P64
PISOA64	800850	PISO-A64/A64U
PISOP32C32	800820	PISO-P32C32/P32C32U/P32S32WU
PEXP32C32	800820	PEX-P32C32
PISO1730	800820	PISO-1730U
PISOP32A32	800870	PISO-P32A32/P32A32U/ P32A32U-5V
PEXP32A32	800870	PEX-P32A32
PISOP8R8	800830	PISO-P8R8/PISO-P8R8AC/PISO-P8R8DC/ P8SSR8AC
PISO730	800840	PISO-730
PEX730	800840	PEX-730
PISO730A	800880	PISO-730A/730AU
PEX730A	800880	PEX-730A
PISO725	8008FF	PISO-725/725U
PISODA2	800B00	PISO-DA2
PISO813	800A00	PISO-813/813U
PCITMC12	DF2962	PCI-TMC12/PCI-TMC12A/TMC12AU
PEXTMC12	DF2962	PEX-TMC12A
PCIM512	DE9562	PCI-M512
PCIM256	DE92A6	PCI-M256

PCIM128	DE9178	PCI-M128
PCIFC16	B13017	PCI-FC16U
PCID64	DE3513	PCI-D64
PCI822	DE3823	PCI-822 LU
PCI826	DE3827	PCI-826 LU
PCI2602	2CB656	PCI-2602U
PCI100X	341002	PCI-1002 LU/HU
PEX100X	341002	PEX-1002
PCI1202	345672	PCI-1202 L/H ,PCI-1202U L/H
PEX1202	345672	PEX-1202 L/H
PCI1602	345676	PCI-1602/1602U,PCI-1602 F
PCI180X	345678	PCI-1800 L/H, PCI-1802 L/H
PCIP8R8	D6102B	PCI-P8R8/P8R8U
PEXP8POR8	D6102B	PEX-P8POR8i
PCIP16R16	D61E39	PCI-P16R16/P16R16U/P16C16/ P16C16U/P16POR16/ P16POR16U
PEXP16POR16	D61E39	PEX-P16POR16i
PISO1730	800820	PISO-1730U
PCIE8620	658627	PCIe-8620
PCIE8622	658629	PCIe-8622
PCID96	80D096	PCI-D96SU
PCID128	80D128	PCI-D128SU

A.3. 配置码定义

透过配置码可用来设定各种硬件的功能来产生不同的应用。

A.3.1. 模拟输入配置码

- 使用者可查询下表在回传值

请参考 A.1. 函数回传值定义。

设定板卡模拟输入的电压范围及极性，每张板卡支持的模拟输入电压范围及极性皆有所不同，更详细的信息可以参考板卡的硬件手册或参考本节**泓格板卡支持模拟输入配置码**表格。

配置码	定义	极性	电压范围
0	IXUD_BI_10V	Bipolar	+/- 10V
1	IXUD_BI_5V	Bipolar	+/- 5V
2	IXUD_BI_2V5	Bipolar	+/- 2.5V
3	IXUD_BI_1V25	Bipolar	+/- 1.25V
4	IXUD_BI_0V625	Bipolar	+/- 0.625V
5	IXUD_BI_0V3125	Bipolar	+/- 0.3125V
6	IXUD_BI_0V5	Bipolar	+/- 0.5V
7	IXUD_BI_0V05	Bipolar	+/- 0.05V
8	IXUD_BI_0V005	Bipolar	+/- 0.005
9	IXUD_BI_1V	Bipolar	+/- 1V
10	IXUD_BI_0V1	Bipolar	+/- 0.1V
11	IXUD_BI_0V01	Bipolar	+/- 0.01V
12	IXUD_BI_0V001	Bipolar	+/- 0.001V
13	IXUD_UNI_20V	Unipolar	0 ~ 20V
14	IXUD_UNI_10V	Unipolar	0 ~ 10V
15	IXUD_UNI_5V	Unipolar	0 ~ 5V
16	IXUD_UNI_2V5	Unipolar	0 ~ 2.5V
17	IXUD_UNI_1V25	Unipolar	0 ~ 1.25V
18	IXUD_UNI_0V625	Unipolar	0 ~ 0.625V
19	IXUD_UNI_1V	Unipolar	0 ~ 1V
20	IXUD_UNI_0V1	Unipolar	0 ~ 0.1V
21	IXUD_UNI_0V01	Unipolar	0 ~ 0.01V
22	IXUD_UNI_0V001	Unipolar	0 ~ 0.001V
23	IXUD_BI_20V	Bipolar	+/- 20V

泓格板卡支持模拟输入配置码表

电压范围	PIO-821L PIO-821LU	PIO-821H PIO-821HU	PISO-813 PIO-813U (JP1=10V)	PISO-813 PIO-813U (JP1=20V)	PCI-1002LU PEX-1002L	PCI-1002HU PEX-1002H
+/- 10V				✓	✓	✓
+/- 5V	✓	✓	✓	✓	✓	
+/- 2.5V	✓		✓	✓	✓	
+/- 1.25V	✓		✓	✓	✓	
+/- 0.625V	✓		✓	✓		
+/- 0.3125V						
+/- 0.5V		✓				
+/- 0.05V		✓				
+/- 0.005		✓				
+/- 1V						✓
+/- 0.1V						✓
+/- 0.01V						✓
+/- 0.001V						
0 ~ 20V						
0 ~ 10V			✓			
0 ~ 5V			✓			
0 ~ 2.5V			✓			
0 ~ 1.25V			✓			
0 ~ 0.625V			✓			
0 ~ 1V						
0 ~ 0.1V						
0 ~ 0.01V						
0 ~ 0.001V						

泓格板卡支持模拟输入配置码表

电压范围	PCI-1202LU PCI-1800LU PCI-1802LU PEX-1202L	PCI-1202HU PCI-1800HU PCI-1802HU PEX-1202H	PCI-1602 PCI-1602U PCI-1602F PCI-1602FU	PCI-822LU PCI-826LU	PCI-2602U	PCIe-8620 PCIe-8622
+/- 10V	✓	✓	✓	✓	✓	✓
+/- 5V	✓	✓	✓	✓	✓	✓
+/- 2.5V	✓		✓	✓	✓	
+/- 1.25V	✓		✓	✓	✓	
+/- 0.625V	✓				✓	
+/- 0.3125V						
+/- 0.5V		✓				
+/- 0.05V		✓				
+/- 0.005		✓				
+/- 1V		✓				
+/- 0.1V		✓				
+/- 0.01V		✓				
+/- 0.001V						
0 ~ 20V						
0 ~ 10V	✓	✓				
0 ~ 5V	✓					
0 ~ 2.5V	✓					
0 ~ 1.25V	✓					
0 ~ 0.625V						
0 ~ 1V		✓				
0 ~ 0.1V		✓				
0 ~ 0.01V		✓				
0 ~ 0.001V						

PCI-2602U 模拟输入配置电压表

电压设定值	实际电压
+/- 10V	+/- 10.24V
+/- 5V	+/- 5.12V
+/- 2.5V	+/- 2.56V
+/- 1.25V	+/- 1.28V
+/- 0.625V	+/- 0.64V

A.3.2. 模拟输出配置码(电压)

使用者可查询下表在模拟输出函数集设定板卡模拟输出的电压范围及极性, 每张板卡支持的模拟输出电压范围及极性皆有所不同, 更详细的信息可以参考板卡的硬件手册或参考本节**泓格板卡支持模拟输出配置码**表格。

配置码	定义	电压范围
0	IXUD_AO_UNI_5V	0 ~ 5V
1	IXUD_AO_BI_5V	+/- 5V
2	IXUD_AO_UNI_10V	0 ~ 10V
3	IXUD_AO_BI_10V	+/- 10V
4	IXUD_AO_UNI_20V	0 ~ 20V
5	IXUD_AO_BI_20V	+/- 20V

泓格板卡支持模拟输出配置码表(电压)

配置码	电压范围	PIO-DA4U PIO-DA8U PIO-DA16U	PISO-DA4U PISO-DA8U PISO-DA16U	PIO-821L PIO-821H PIO-821LU PIO-821HU	PISO-DA2U	CI-1202 PCI-1602 PCI-1800 PCI-1802 PEX-1202	PCI-822 PCI-826 PCI-2602U	PCIe-8622
0	0 ~ 5V	-	-	✓	✓	-	✓	✓
1	+/- 5V	-	-	✓	✓	✓	✓	✓
2	0 ~ 10V	-	-	-	✓	-	✓	✓
3	+/- 10V	✓	✓	-	✓	✓	✓	✓

A.3.3. 模拟输出配置码(电流)

使用者可查询下表在模拟输出函数集设定板卡模拟输出的电流范围及极性, 每张板卡支持的模拟输出电流范围及极性皆有所不同, 更详细的信息可以参考板卡的硬件手册或参考本节**泓格板卡支持模拟输出配置码(电流)**表格。

配置码	定义	电流范围
16	IXUD_AO_I_0_20_MA	0 ~ 20 mA
17	IXUD_AO_I_4_20_MA	4 ~ 20 mA

泓格板卡支持模拟输出配置码表(电流)

配置码	电流范围	PIO-DA4U	PISO-DA4U	PEX-DA4	PISO-DA2U
		PIO-DA8U	PISO-DA8U	PEX-DA8	
		PIO-DA16U	PISO-DA16U	PEX-DA16	
16	0 ~ 20	✓	✓	✓	✓
17	4~20	-	-		✓

A.3.4. 中断事件配置码

下表配置码定义中断事件

配置码	定义	叙述
1	IXUD_HARDWARE_INT	硬件中断
2	IXUD_APC_READY_INT	模拟数据准备完成产生中断
4	IXUD_ACTIVE_LOW	下降缘触发产生事件
8	IXUD_ACTIVE_HIGH	上升缘触发产生事件

A.4. 数字输入端口定义号码

DI 埠号	PIO-D24U PEX-D24	PIO-D56U PEX-D56	PIO-D48U PIO-D48SU PEX-D48	PIO-D64U	PIO-D96U PIO-D96SU PEX-D96S	PIO-D144 PIO-D144U PIO-D144LU PEX-D144LS	PIO-D168U	PISO-P64 PISO-P64U PEX-P64
0	CN3 Port0	CN3 Port0	CN1 Port0	CN2 DI 0 ~ 7	CN1 Port0	CN1 Port0	CN1 Port0	IDI 0 ~ 7
1	CN3 Port1	CN3 Port1	CN1 Port1	CN2 DI 8 ~ 15	CN1 Port1	CN1 Port1	CN1 Port1	IDI 8 ~ 15
2	CN3 Port2	CN3 Port2	CN1 Port2	CN4 DI 0 ~ 7	CN1 Port2	CN1 Port2	CN1 Port2	IDI 16 ~ 23
3	-	CN2 DI 0 ~ 7	CN2 Port3	CN4 DI 8 ~ 15	CN2 Port3	CN2 Port3	CN2 Port3	IDI 24 ~ 31
4	-	CN2 DI 8 ~ 15	CN2 Port4	-	CN2 Port4	CN2 Port4	CN2 Port4	IDI 32 ~ 39
5	-	-	CN2 Port5	-	CN2 Port5	CN2 Port5	CN2 Port5	IDI 40 ~ 47
6	-	-	-	-	CN3 Port6	CN3 Port6	CN3 Port6	IDI 48 ~ 55
7	-	-	-	-	CN3 Port7	CN3 Port7	CN3 Port7	IDI 56 ~ 63
8	-	-	-	-	CN3 Port8	CN3 Port8	CN3 Port8	
9	-	-	-	-	CN4 Port9	CN4 Port9	CN4 Port9	
10	-	-	-	-	CN4 Port10	CN4 Port10	CN4 Port10	
11	-	-	-	-	CN4 Port11	CN4 Port11	CN4 Port11	
12	-	-	-	-	-	CN5 Port12	CN5 Port12	
13	-	-	-	-	-	CN5 Port13	CN5 Port13	
14	-	-	-	-	-	CN5 Port14	CN5 Port14	
15	-	-	-	-	-	CN6 Port15	CN6 Port15	
16	-	-	-	-	-	CN6 Port16	CN6 Port16	
17	-	-	-	-	-	CN6 Port17	CN6 Port17	
18	-	-	-	-	-	-	CN6 Port18	
19	-	-	-	-	-	-	CN6 Port19	
20	-	-	-	-	-	-	CN6 Port20	

DI 埠号	PISO-P32A32U PISO-P32A32U-5V PISO-P32C32U PISO-P32S32WU PISO-1730U PEX-P32C32 PEX-P32A32	PISO-P16R16U PEX-P16R16i PCI-P16R16U	PISO-P8R8U PISO-P8SSR8AC PEX-P8R8i PCI-P8R8U PISO-725 PISO-725U	PISO-730 PISO-730U PISO-730A PISO-730AU PEX-730 PEX-730A	PCI-P8R8 PEX-P8POR8i	PCI-P16R16 PCI-P16C16 PCI-P16C16U PEX-P16POR16i PCI-P16POR16U
0	CN1 IDI 0 ~ 7	CN1 IDI 0 ~ 7	CN1 IDI 0 ~ 7	CN1 IDI 0 ~ 7	IDI 0 ~ 7	IDI 0 ~ 15
1	CN1 IDI 8 ~ 15	CN2 IDI 8 ~ 15	-	CN1 IDI 8 ~ 15	-	-
2	CN2 IDI 16 ~ 23	-	-	CN2 DI 0 ~ 7	-	-
3	CN2 IDI 24 ~ 31	-	-	CN2 DI 8 ~ 15	-	-

DI 埠号	PCI-822LU PCI-826LU PCI-FC16U	PIO-821L PIO-821H PIO-821LU PIO-821HU	PIO-DA4U PIO-DA8U PIO-DA16U	PISO-DA4U PISO-DA8U PISO-DA16U	PEX-DA4 PEX-DA8 PEX-DA16	PCI-1002 PEX-1002	PCI-1202 PEX-1202	PCI-1602 PCI-1800 PCI-1802
0	PA 0 ~ 15	DI 0~7	DI 0 ~ 7	DI 0 ~ 7	DI 0 ~ 7	DI 0 ~ 15	DI 0 ~ 15	DI 0 ~ 15
1	PB 0 ~ 15	DI 8~15	DI 8 ~ 15	DI 8 ~ 15	DI 8 ~ 15	-	-	-

DI 埠号	PCI-M512 PCI-M512U	PCI-TMC12 PCI-TMC12A PCI-TMC12AU PEX-TMC12A	PCI-2602U	PCI-D96SU	PCI-D128SU	PCIe-8620	PCIe-8622
0	DI 0 ~ 11	DI 0 ~ 15	PA0~7 PB0~7 PC0~7 PD0~7	CON1 Port0	CON1 Port0	DI 0~3	DI 0~11
1				CON1 Port1	CON1 Port1		
2				CON1 Port2	CON1 Port2		
3					CN1/2 Port3		

双向数字输出输入端口
 数字输入端口



双向数字输出输入端口，在使用前需使用 `Ixud_SetDIOModes32` 及 `Ixud_SetDIOMode` 函式设定 I/O 模式为输入模式

A.5. 数字输出端口定义号码

DO 埠号	PIO-D24U PEX-D24	PIO-D56U PEX-D56	PIO-D48U PIO-D48SU PEX-D48	PIO-D64U	PIO-D96U PIO-D96SU PEX-D96S	PIO-D144 PIO-D144U PIO-D144LU PEX-D144LS	PIO-D168U	PISO-A64 PISO-A64U PISO-C64 PISO-C64U PEX-C64
0	CN3 Port0	CN3 Port0	CN1 Port0	CN1 DO 0 ~ 7	CN1 Port0	CN1 Port0	CN1 Port0	IDO 0 ~ 7
1	CN3 Port1	CN3 Port1	CN1 Port1	CN1 DO 8 ~ 15	CN1 Port1	CN1 Port1	CN1 Port1	IDO 8 ~ 15
2	CN3 Port2	CN3 Port2	CN1 Port2	CN3 DO 0 ~ 7	CN1 Port2	CN1 Port2	CN1 Port2	IDO 16 ~ 23
3	-	CN1 DO 0 ~ 7	CN2 Port3	CN3 DO 8 ~ 15	CN2 Port3	CN2 Port3	CN2 Port3	IDO 24 ~ 31
4	-	CN1 DO 8 ~ 15	CN2 Port4	-	CN2 Port4	CN2 Port4	CN2 Port4	IDO 32 ~ 39
5	-	-	CN2 Port5	-	CN2 Port5	CN2 Port5	CN2 Port5	IDO 40 ~ 47
6	-	-	-	-	CN3 Port6	CN3 Port6	CN3 Port6	IDO 48 ~ 55
7	-	-	-	-	CN3 Port7	CN3 Port7	CN3 Port7	IDO 56 ~ 63
8	-	-	-	-	CN3 Port8	CN3 Port8	CN3 Port8	
9	-	-	-	-	CN4 Port9	CN4 Port9	CN4 Port9	
10	-	-	-	-	CN4 Port10	CN4 Port10	CN4 Port10	
11	-	-	-	-	CN4 Port11	CN4 Port11	CN4 Port11	
12	-	-	-	-	-	CN5 Port12	CN5 Port12	
13	-	-	-	-	-	CN5 Port13	CN5 Port13	
14	-	-	-	-	-	CN5 Port14	CN5 Port14	
15	-	-	-	-	-	CN6 Port15	CN6 Port15	
16	-	-	-	-	-	CN6 Port16	CN6 Port16	
17	-	-	-	-	-	CN6 Port17	CN6 Port17	
18	-	-	-	-	-	-	CN6 Port18	
19	-	-	-	-	-	-	CN6 Port19	
20	-	-	-	-	-	-	CN6 Port20	

DO 埠号	PISO-P32A32U PISO-P32A32U-5V PISO-P32C32U PISO-P32S32WU PISO-1730U PEX-P32C32 PEX-P32A32	PISO-P16R16U PEX-P16R16i PCI-P16R16U	PISO-P8R8U PISO-P8SSR8 AC PEX-P8R8i PCI-P8R8U PISO-725 PISO-725U	PISO-730 PISO-730A PISO-730U PISO-730AU PEX-730 PEX-730A	PCI-P8R8 PEX-P8POR8i	PCI-P16R16 PCI-P16C16 PCI-P16C16U PEX-P16POR16i PCI-P16POR16U
0	CN1 IDO 0 ~ 7	CN1 IDO 0 ~ 7	CN1 IDO 0 ~ 7	CN1 IDO 0 ~ 7	IDO 0 ~ 7	IDO 0 ~ 15
1	CN1 IDO 8 ~ 15	CN2 IDO 8 ~ 15	-	CN1 IDO 8 ~ 15	-	-
2	CN2 IDO 16 ~ 23	-	-	CN2 DO 0 ~ 7	-	-
3	CN2 IDO 24 ~ 31	-	-	CN2 DO 8 ~ 15	-	-

DO 埠号	PCI-822LU PCI-826LU PCI-FC16U	PIO-821L PIO-821H PIO-821LU PIO-821HU	PIO-DA4U PIO-DA8U PIO-DA16U	PISO-DA4U PISO-DA8U PISO-DA16U	PEX-DA4 PEX-DA8 PEX-DA16	PCI-1002 PEX-1002	PCI-1202 PEX-1202	PCI-1602 PCI-1802
0	PA 0 ~ 15	DO 0~7	DO 0 ~ 7	DO 0 ~ 7	DO 0 ~ 7	DO 0 ~ 15	DO 0 ~ 15	DO 0 ~ 15
1	PB 0 ~ 15	DO 8~15	DO 8 ~ 15	DO 8 ~ 15	DO 8 ~ 15	-	-	

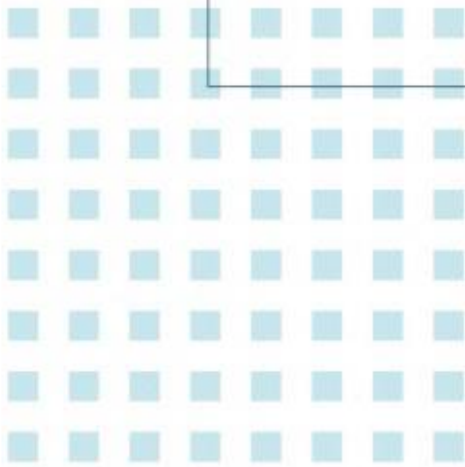
DO 埠号	PCI-M512	PCI-TMC12 PCI-TMC12A PCI-TMC12AU PEX-TMC12A	PCI-2602U	PCI-D96SU	PCI-D128SU	PCIe-8620	PCIe-8622
0	DO 0 ~ 15	DO 0 ~ 15	PA 0 ~ 7 PB 0 ~ 7 PC 0 ~ 7 PD 0 ~ 7	CON1 Port0	CON1 Port0	DO 0~3	DO 0~11
	-	-	-	CON1 Port1	CON1 Port1	-	-
	-	-	-	CON1 Port2	CON1 Port2	-	-
	-	-	-	-	CN1/2 Port3	-	-

双向数字输出输入端口
 数字输出端口



双向数字输出端口, 在使用前需使用 `Ixud_SetDIOModes32` 及 `Ixud_SetDIOMode` 函式设定 I/O 模式为输出模式

B



附录 B. 其他

本章节将会提供一些其他的补充数据。

B.1. 常见问题

系统与安装

Q. UniDAQ 支持 64-bit 的操作系统吗？

A. 支持，由于 UniDAQ 驱动程序函式库支持 64 位操作系统。

Q. 如果我原来使用 Classic 版本的驱动程序，换成使用 UniDAQ 驱动程序，我需要修改软件吗？

A. 需要，为了加速用户开发的速度及整合泓格所有 I/O 板卡，所以 UniDAQ 驱动程序的 API 接口设计已与 Classic 版本的驱动程序完全不兼容。

Q. 我不知道我安装的驱动程序是 Classic 系列的或是 UniDAQ，请问该如何判别？

A. 请您至设备管理器，检查板卡装置名称，如果您安装的是 UniDAQ 驱动程序，装置名称前方会有[UniDAQ]字样，如果没有即代表安装的是 Classic 系列驱动程序。

Q. 如果系统需要再增加第二张不同的泓格板卡来开发新的项目，一张因为原先就是使用 Classic 驱动程序，因为我不想变更软件，我可以让第二张板卡使用 UniDAQ 的函式库来作开发吗？

A. 可以，请您第一张保持使用 Classic 驱动程序，第二张使用 UniDAQ 驱动程序即可。

Q. UniDAQ 支援 ISA 总线的板卡吗？

A. UniDAQ 目前尚未支持任何 ISA 总线的板卡。

数字输出

Q. 使用 PIO-D24U/D56U/D48U/D96U/D144U/D168U 时，输出埠无法输出，输入埠无法输入？

A. 因为 PIO-D24U/D56U/D48U/D96U/D144U/D168U 的输出输入埠为双向埠，需先设定规画埠，请先使用 `Ixud_SetDIOModes32` 或 `Ixud_SetDIOMode` 函式设定端口的模式，再对端口作输出或输入的动作。

模拟输出

Q. 使用 PIO-DA4U/8U/16U 或 PISO-DA4U/8U/16U 板卡，为什么我设定模拟输出范围 $\pm 5V$ ， $0 \sim 10V$ ， $0 \sim 5V$ 及 $4 \sim 20 \text{ mA}$ 时后使用模拟输出函式却输出不正确的电压或电流。

A. 因为 PIO-DA4U/8U/16U 或 PISO-DA4U/8U/16U 的硬件设计仅支持 $\pm 10V$ 的电压输出及 $0 \sim 20 \text{ mA}$ 的电流输出，如果您设定成其他未支持的电压或电流范围，将会输出不正确的电压及电流。

Q. 使用模拟输出函式输出一电压或电流值为何输出不正确的电压或电流。

A. 请您检查您的电压范围设定是否正确，并利用 `Ixud_ConfigAO` 设定正确输出范围，再使用 `Ixud_WriteAOVoltage` 或 `Ixud_WriteAOCurrent` 函式输出电压或电流。

函式错误码故障排除

Q. 错误码 1.

A. 请重新安装泓格 UniDAQ 驱动函式库或重新启动。

Q. 错误码 2.

A. (1)请在使用 UniDAQ 函式前使用 Ixud_DriverInit 作初始化的动作。
(2)wBoardNo 有误, 请重新检查 wBoardNo。

Q. 错误码 5.

A. wBoardNo 有误, 请重新检查 wBoardNo。

Q. 错误码 6.

A. 未找到任何板卡, 请您安装泓格板卡再开始程序。

Q. 错误码 13

A. 此板卡不支持此函式功能。

Q. 错误码 19

A. 请设定正确的类入输入范围。

B.2. 版本修改信息

Revision	Date	Description
1.0	Sep. 2009	初版
1.3	Sep. 2011	增加函式
2.0	June. 2012	新增安装教学、开发指南及函式应用说明
2.1	Dec. 2012	修正中断事件配置码 修正中断支持列表
2.2	May. 2013	修正 PISO-813 的 CardType 参数设定 新增 PCI-1002 支援 Channel Scan
2.3	Feb. 2014	新增新产品 新增新的 API 函式
2.5	Aug. 2019	修正部分 API 错误说明、新增新产品
2.6	Dec. 2019	修正函式支持列表、部分表格