User's Guide

OME-PCI-1800/1802

PCI Data Acquisition Boards
Windows Software Manual

omega.com®
OMEGA®

OMEGAnet® Online Service www.omega.com	Internet e-mail info@omega.com

Servicing North America:

USA: ISO 9001 Certified
One Omega Drive, P.O. Box 4047
Stamford CT 06907-0047
TEL: (203) 359-1660 FAX: (203) 359-7700
e-mail: info@omega.com

Canada:
976 Bergar
Laval (Quebec) H7L 5A1, Canada
TEL: (514) 856-6928 FAX: (514) 856-6886
e-mail: info@omega.ca

For immediate technical or application assistance:

USA and Canada: Sales Service: 1-800-826-6342 / 1-800-TC-OMEGA®
Customer Service: 1-800-622-2378 / 1-800-622-BEST®
Engineering Service: 1-800-872-9436 / 1-800-USA-WHEN®
TELEX: 996404 EASYLINK: 62968934 CABLE: OMEGA

Mexico: En Español: (001) 203-359-7803 e-mail: espanol@omega.com
FAX: (001) 203-359-7807 info@omega.com.mx

Servicing Europe:

Benelux:
Postbus 8034, 1180 LA Amstelveen, The Netherlands
TEL: +31 (0)20 3472121 FAX: +31 (0)20 6434643
Toll Free in Benelux: 0800 0993344
e-mail: sales@omegaeng.nl

Czech Republic:
Frystatska 184, 733 01 Karviná, Czech Republic
TEL: +420 (0)59 6311899 FAX: +420 (0)59 6311114
Toll Free: 0800-1-66342 e-mail: info@omegashop.cz

France:
11, rue Jacques Cartier, 78280 Guyancourt, France
TEL: +33 (0)1 61 37 29 00 FAX: +33 (0)1 30 57 54 27
Toll Free in France: 0800 466 342
e-mail: sales@omega.fr

Germany/Austria:
Daimlerstrasse 26, D-75392 Deckenpfronn, Germany
TEL: +49 (0)7056 9398-0 FAX: +49 (0)7056 9398-29
Toll Free in Germany: 0800 639 7678
e-mail: info@omega.de

United Kingdom: ISO 9002 Certified
One Omega Drive, River Bend Technology Centre
Northbank, Irlam, Manchester
M44 5BD United Kingdom
TEL: +44 (0)161 777 6611 FAX: +44 (0)161 777 6622
Toll Free in United Kingdom: 0800-488-488
e-mail: sales@omega.co.uk

Table of Contents

1. INTRODUCTION

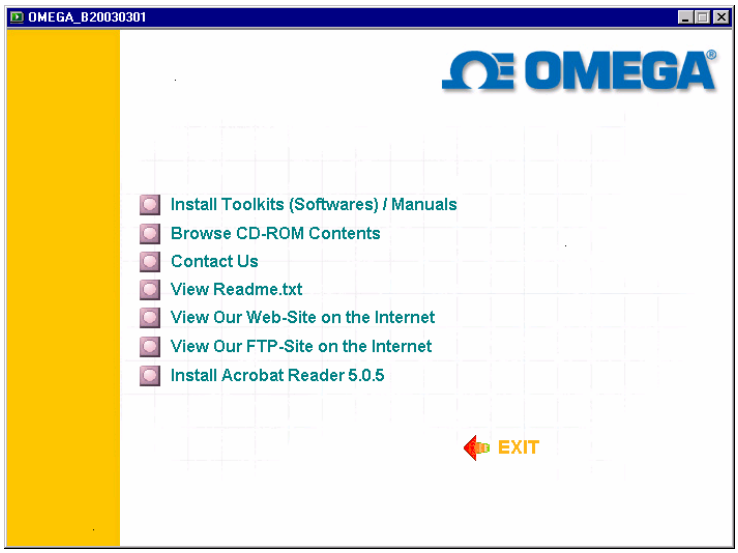
The PCI-180x toolkit is a collection of DLLs for Windows 95/NT/2000/XP applications. The DLLs are 32 bit and can be called by Visual C/C++, BC++, Visual BASIC, Delphi, and LabVIEW.

These DLLs can perform a variety of data acquisition operations such as:

- Get software version
- Initialize data acquisition hardware
- Digital Input/Output
- A/D conversion
- D/A conversion

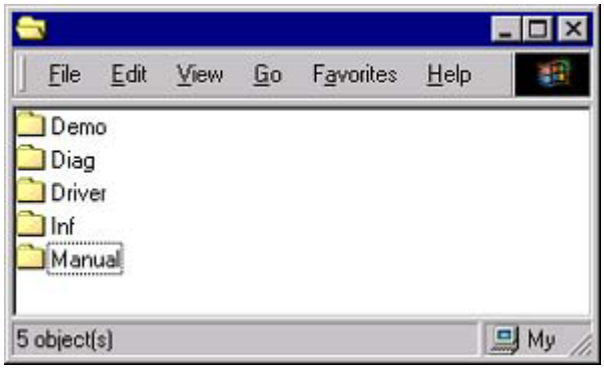
1.1 SOFTWARE INSTALLATION

Insert the CD ROM included with your OME-PCI-180x board and the following installation screen should auto start.



Follow the instructions on the screen to complete the software installation. The software is designed to support the entire OME family of data acquisition hardware, so during the installation, you will be asked to specify your particular hardware (OME-PCI-180x board in this case). During the installation process, you will also be prompted to enter the operating system you will be using.

After installation the following folders will be created on your computer.



Demo Folder
Contains all demonstration programs including the source code. Examples are provided for Visual C++, Borland C++, Visual Basic and Delphi.
Please note: The VC++ demos are developed with VC++ 4.0. After setting up the environment, use the NMAKE.EXE to compiling and linking the demo code. For example,
C:\P1602\DEMO\VC\nmake /f demo1.mak

Driver Folder
Contains software drivers, include files and definition files for the programming languages.

Manual Folder
Contains hardware user manuals, software user manuals and technical notes.

Diag Folder
Contains card diagnostic programs

Inf Folder
Contains tech notes and .INF file for the plug and play installation (only available for operating systems that support plug and play).

1.2 PLUG & PLAY

Then the p180X.inf provides all information needed to complete the Plug & Play installation. **After the installation, the Windows will reserve the resources and populate the registry.** The Plug & Play information is shown in Figure 1. Figure 2 shows the resources that are reserved. If the user executes **"c:\WINNT\regedit.exe"**, the registration information of the PCI-180XH/L will be located under **"My Computer\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\P180x"** as in Figure 3. Figure 4 shows the registration items in detail.

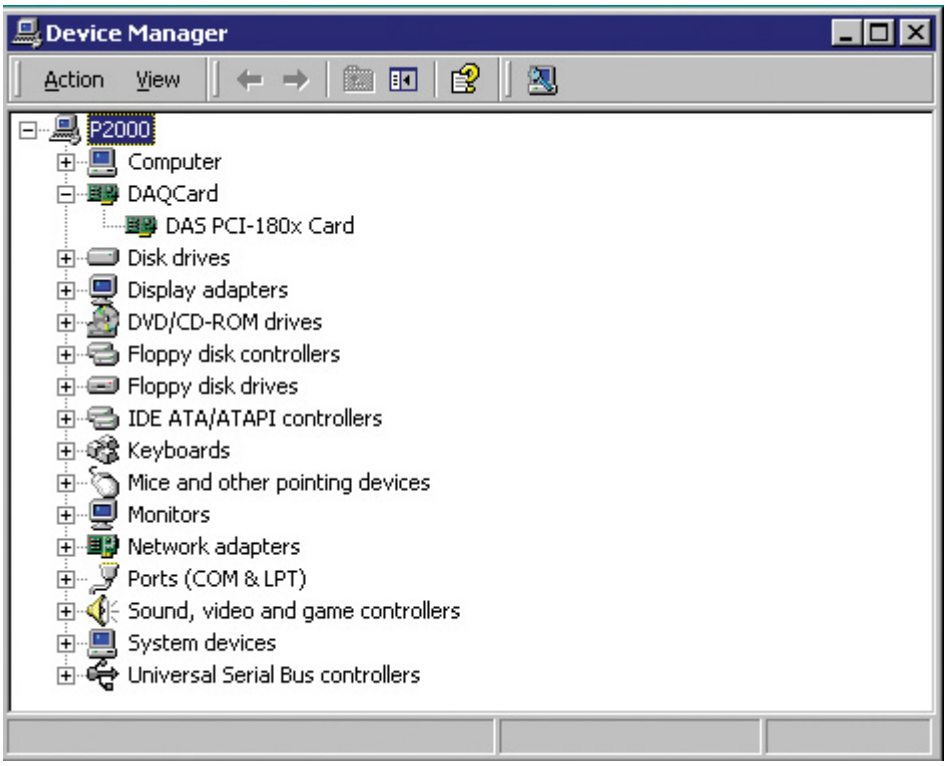


Fig 1. The Plug & Play information

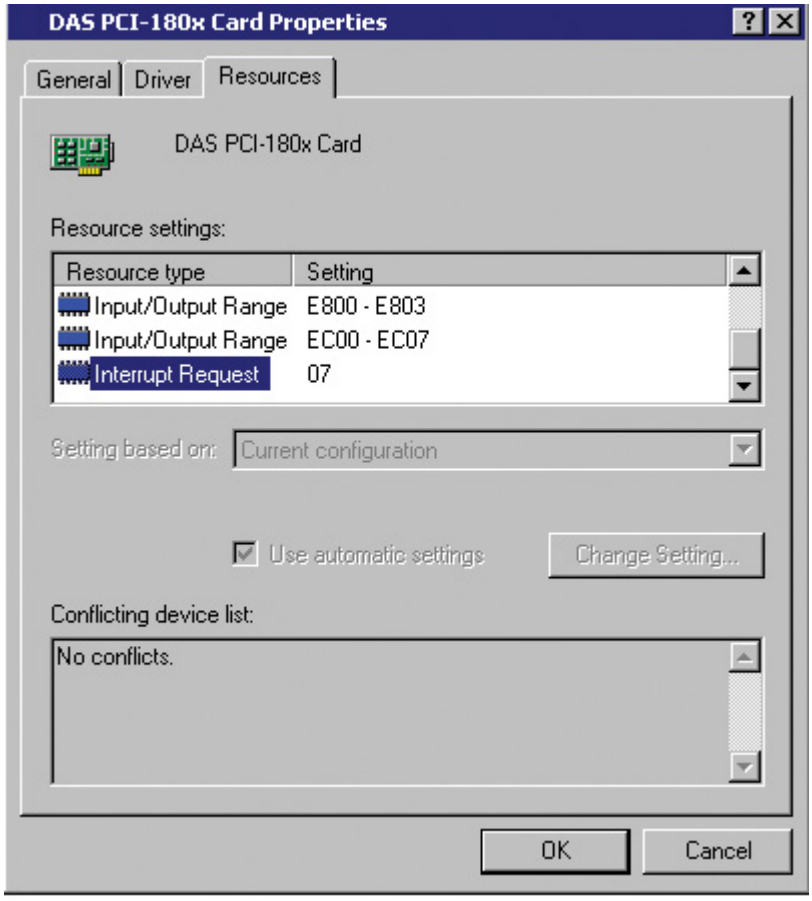


Fig 2. The allocated resources for this PCI-180X.

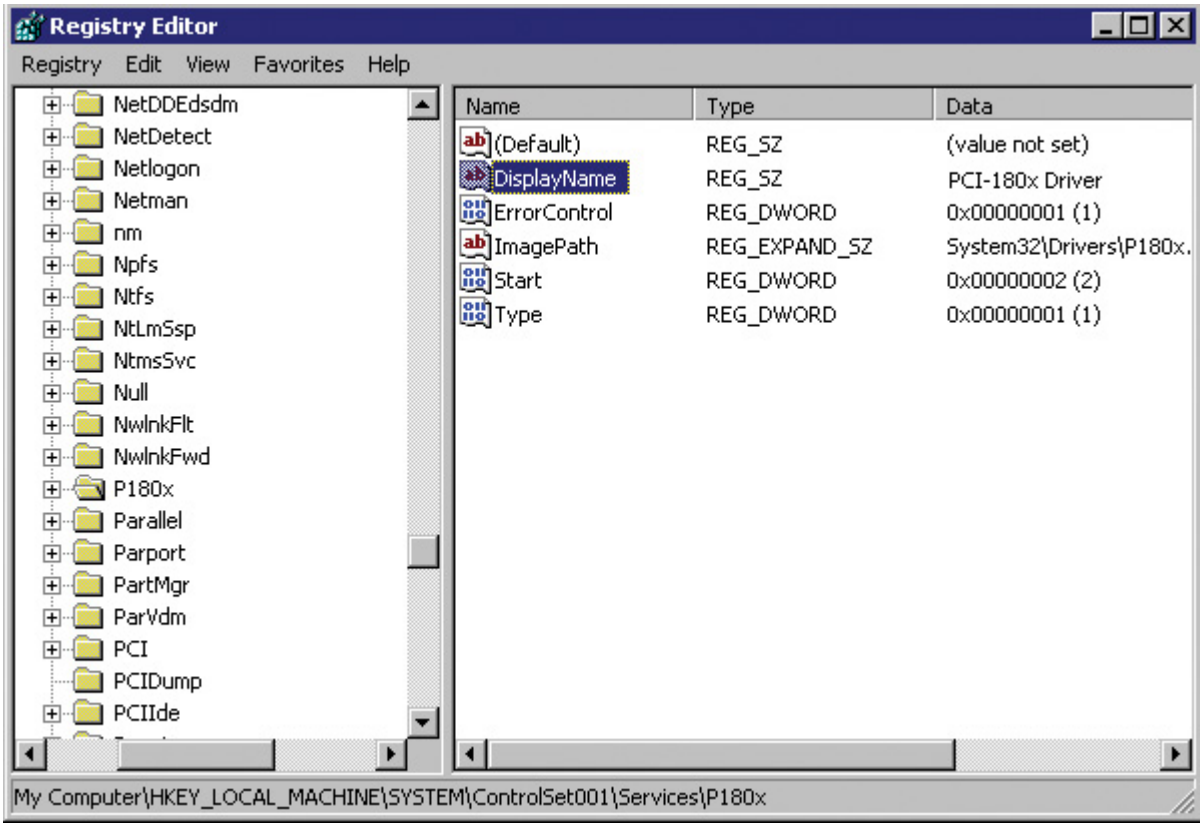


Fig 3. The registry path

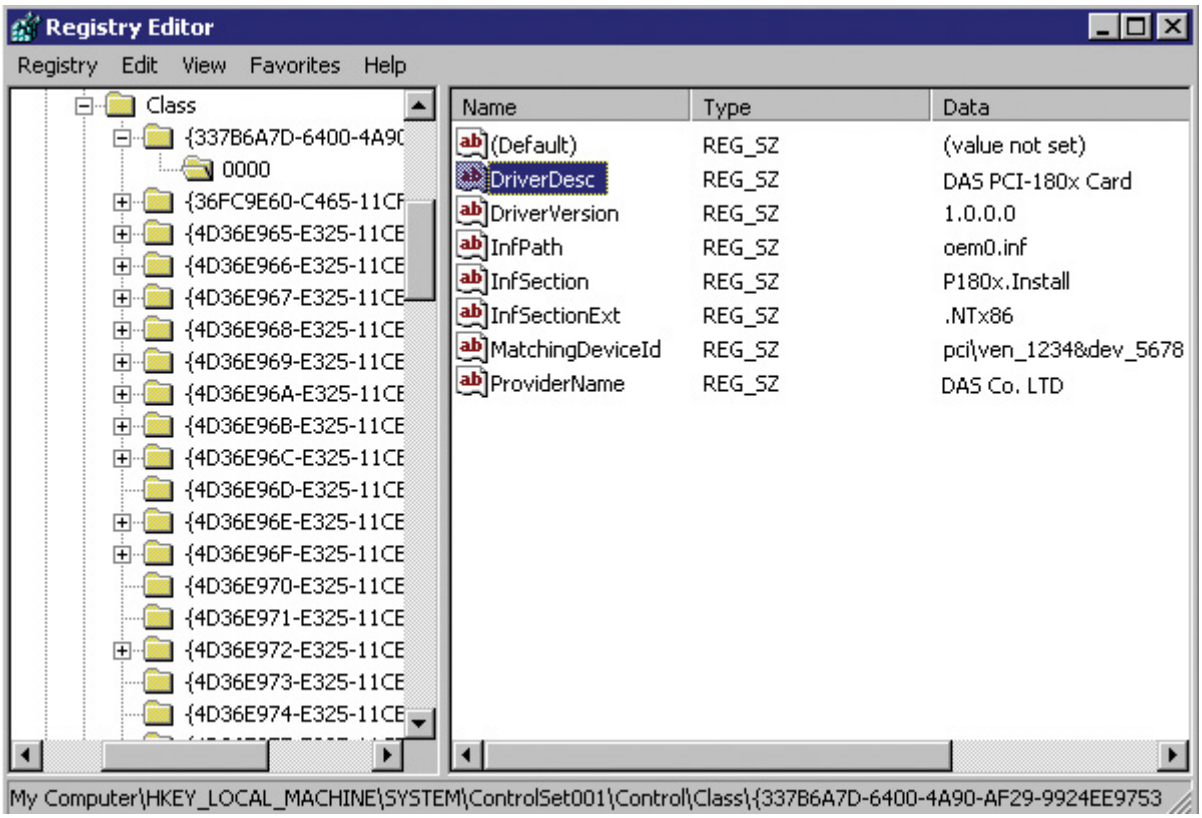


Fig 4. Registry item details

1.3 Using With Visual C++

All demo programs were created with Visual C++5.and tested under Windows 95/NT. Key issues are given below:

1. Make sure the PATH includes the Visual C++ compiler
2. Execute the \MSDEV\BIN\VCVARS32.BAT to setup the environment. The VCVARS32.BAT is provided by Visual C++.
3. The source program must include “P180X.H”
4. Copy the P180X.LIB, import library, to the same directory as the source program.
5. Copy the P180X.DLL, to the same directory as the source program
6. Edit the source program (refer to Demo\VC5\DEMO1.C)
7. Edit the NMAKE file (refer to Demo\VC5\DEMO1.MAK)
8. NMAKE /f DEMO1.MAK
9. Execute DEMO1.EXE

NOTE: The **P180X.lib is used while linking** and the **P180X.DLL is used at run time.**

1.4 Using With MFC

Using the PCI-180x toolkit with MFC is similar as using it with Visual C++. Key issues are given below:

1. Use MFC wizard to create the source code
2. The source program must include “P180X.H”
3. Copy the P180X.LIB, import library, to the same directory as the source program
4. Copy the P180X.DLL, to the same directory as the source program
5. Select **Build/Settings/Link** and key “P180X.lib” in the **object/library modules** field

NOTE: The **P180X.lib is used while linking and** the **P180X.DLL is used in run time.**

1.5 Using With BC++

The DLL was created in Visual C++ 5.0. The P180X.H and P180X.lib are also designed for use with Visual C/C++. BC++ can not use these two files. The following modifications are necessary.

```
#include <conio.h>
#include <windows.h>
HINSTANCE hDLLLib;
// Step 1 : declare a function pointer
float CALLBACK (*FloatSub2)(float fA, float fB);
main()
{
// Step 2 : load dll
hDllLib=LoadLibrary("P180X.dll");
if (hDLLLib)
    {
// Step 3 : get the function address
    FloatSub2=(FARPROC)GetProcAddress(hDLLLib,"P180X_FloatSub2");
    if (FloatSub2)
        {
// Step 4 : call function
        printf("1.2-3.4=%f",FloatSub2(1.2,3.4);
        }
    else printf("get P180X_FloatSub2 function address error");
// Step 5 : free library
    FreeLibrary(hDLLib);
    }
else printf("load P180X.dll error");
getch();
}
```

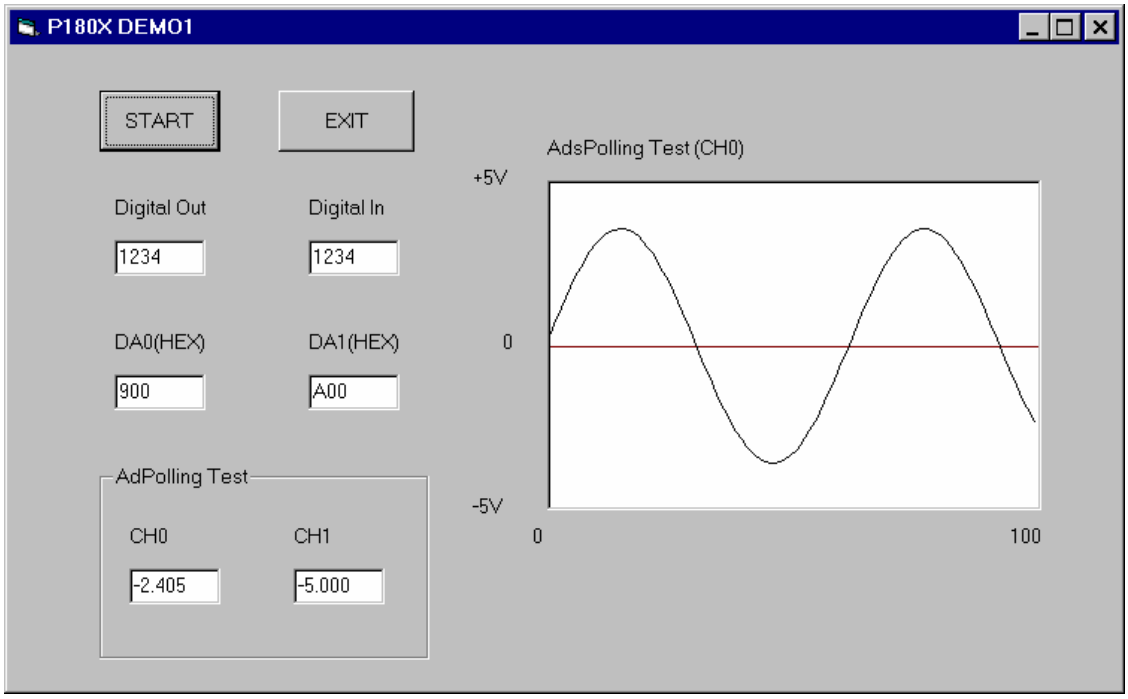
This usage can be divided into **5 steps** listing above. **Using this modification and P180X.DLL, the user can use BC++ to call DLL.**

1.6 Using With Visual Basic

P180X.DLL	**→ DLLs**
DEMO1.FRM	**→ Form file**
P180X.BAS	**→ Module file**
DEMO1.VBP	**→ Project file**

NOTE : 1. Tested under **Windows 95/NT and VB 4.0 (32 bit)**

1.6.1 The Demo1 Result:



1.6.2 The P180X.BAS

```
Attribute VB_Name = "Module1"
Option Explicit
Global Const NoError                    = 0
Global Const DriverHandleError          = 1
Global Const DriverCallError            = 2
Global Const AdControllerError          = 3
Global Const M_FunExecError             = 4
Global Const ConfigCodeError            = 5
Global Const FrequencyCalculateError = 6
Global Const HighAlarm                  = 7
Global Const LowAlarm                   = 8
Global Const AdPollingTimeOut           = 9
Global Const AlarmTypeError             = 10
Global Const FindBoardError             = 11
Global Const AdChannelError             = 12
Global Const DaChannelError             = 13
Global Const InvalidDelay         = 14
Global Const DelayTimeOut               = 15
Global Const InvalidData          = 16
Global Const FifoOverflow               = 17
Global Const TimeOut                    = 18
Global Const ExceedBoardNumber    = 19
Global Const NotFoundBoard              = 20
Global Const OpenError                  = 21
Global Const FindTwoBoardError          = 22
Global Const ThreadCreateError          = 23
Global Const StopError                  = 24
Global Const AllocateMemoryError    = 25

Declare Function P180X_DriverInit Lib "P180X.dll" (wTotalBoards As Integer) As Integer
Declare Sub P180X_DriverClose Lib "P180X.dll" ()

Declare Function P180X_GetDriverVersion Lib "P180X.dll" (wVxdVersion As Integer) As Integer

Declare Function P180X_GetConfigAddressSpace Lib "P180X.dll" (ByVal wBoardNo As Integer, _
            wAddrTimer As Integer, wAddrCtrl As Integer, wAddrDio As Integer, _
             wAddrAdda As Integer) As Integer

Declare Function P180X_ActiveBoard Lib "P180X.dll" (ByVal wBoardNo As Integer) As Integer
Declare Function P180X_WhichBoardActive Lib "P180X.dll" () As Integer

Declare Function P180X_M_FUN_1 Lib "P180X.dll" (ByVal wDaFrequency As Integer, _
              ByVal wDaWave As Integer, ByVal fDaAmplitude As Single, _
              ByVal wAdClock As Integer, ByVal wAdNumber As Integer, _
              ByVal wAdConfig As Integer, fAdBuf As Single, ByVal fLowAlarm As Single, _
              ByVal fHighAlarm As Single) As Integer

Declare Function P180X_M_FUN_2 Lib "P180X.dll" (ByVal wDaNumber As Integer, _
              ByVal wDaWave As Integer, wDaBuf As Integer, ByVal wAdClock As Integer, _
              ByVal wAdNumber As Integer, ByVal wAdConfig As Integer, _
              wAdBuf As Integer) As Integer
```

```
Declare Function P180X_M_FUN_3 Lib "P180X.dll" (ByVal wDaFrequency As Integer, _
                ByVal wDaWave As Integer, ByVal fDaAmplitude As Single, _
                ByVal wAdClock As Integer, ByVal wAdNumber As Integer, _
                wChannelStatus As Integer, wAdConfig As Integer, fAdBuf As Single, _
               ByVal fLowAlarm As Single, ByVal fHighAlarm As Single) As Integer

Declare Function P180X_M_FUN_4 Lib "P180X.dll" (ByVal wType As Integer,  _
            ByVal wDaFrequency As Integer,  _
            ByVal wDaWave As Integer, ByVal fDaAmplitude As Single, _
            ByVal wAdClock As Integer, ByVal wAdNumber As Integer, _
            wChannelStatus As Integer, wAdConfig As Integer, fAdBuf As Single,  _
            ByVal fLowAlarm  As Single, ByVal fHighAlarm As Single) As Integer

Declare Function P180X_Di Lib "P180X.dll" (wDi As Integer) As Integer
Declare Function P180X_Do Lib "P180X.dll" (ByVal wDo As Integer) As Integer

Declare Function P180X_Da Lib "P180X.dll" (ByVal wDaChannel As Integer, _
            ByVal wDaVal As Integer)  As Integer
Declare Function P180X_SetChannelConfig Lib "P180X.dll" (ByVal wAdChannel As Integer,  _
            ByVal wConfig  As Integer) As Integer

Declare Function P180X_AdPolling Lib "P180X.dll" (fAdVal As Single) As Integer
Declare Function P180X_AdsPolling Lib "P180X.dll" (fAdVal As Single, ByVal wNum As Integer)  _
            As Integer
Declare Function P180X_AdsPacer Lib "P180X.dll" (fAdVal As Single, ByVal wNum As Integer,  _
            ByVal wSample As Integer) As Integer

Declare Function P180X_ClearScan Lib "P180X.dll" () As Integer
Declare Function P180X_StartScan Lib "P180X.dll" (ByVal wSampleRate As Integer,  _
            ByVal dwNum As Long, ByVal nPriority As Integer) As Integer
Declare Sub P180X_ReadScanStatus Lib "P180X.dll" (wStatus As Integer, dwLowAlarm As Long,  _
            dwHighAlarm As Long)
Declare Function P180X_AddToScan Lib "P180X.dll" (ByVal wAdChannel As Integer,  _
            ByVal wConfig As Integer, ByVal wAverage As Integer, ByVal wLowAlarm As Integer, _
            ByVal wHighAlarm As Integer, ByVal wAlarmType As Integer) As Integer
Declare Function P180X_SaveScan Lib "P180X.dll" (ByVal wOridinalOrder As Integer,  _
            wBuf As Integer)  As Integer
Declare Sub P180X_WaitMagicScanFinish Lib "P180X.dll" (wStatus As Integer,  _
            wLowAlarm As Integer, _ wHighAlarm As Integer)
Declare Function P180X_StopMagicScan Lib "P180X.dll" () As Integer

Declare Function P180X_DelayUs Lib "P180X.dll" (ByVal wDelayUs As Integer) As Integer

'----------------------- FunA series ----------------------------


'----------------------- FunB series ----------------------------
Declare Function P180X_FunB_Start Lib "P180X.dll" (ByVal wClockDiv As Integer,  _
            wChannel As Integer,  wConfig As Integer, Buffer As Integer,  _
            ByVal dwMaxCount As Long,  ByVal nPriority As Integer) As Integer
Declare Function P180X_FunB_ReadStatus Lib "P180X.dll" () As Integer
Declare Function P180X_FunB_Stop Lib "P180X.dll" () As Integer
Declare Function P180X_FunB_Get Lib "P180X.dll" (P0 As Long) As Integer

Declare Function P180X_Card0_StartScan Lib "P180X.dll" (ByVal wSampleRate As Integer,  _
            wChannelStatus As Integer, wChannelConfig As Integer, ByVal wCount As Integer) As Integer
Declare Function P180X_Card0_ReadStatus Lib "P180X.dll" (wBuf As Integer, wBuf2 As Integer,  _
            dwP1 As Long, dwP2 As Long, wStatus As Integer) As Integer
Declare Sub P180X_Card0_Stop Lib "P180X.dll" ()
```

```
Declare Function P180X_Card1_StartScan Lib "P180X.dll" (ByVal wSampleRate As Integer,  _
        wChannelStatus As Integer, wChannelConfig As Integer,  _
        ByVal wCount As Integer) As Integer
Declare Function P180X_Card1_ReadStatus Lib "P180X.dll" (wBuf As Integer, wBuf2 As Integer,  _
        dwP1 As Long, dwP2 As Long, wStatus As Integer) As Integer
Declare Sub P180X_Card1_Stop Lib "P180X.dll" ()

Declare Function GetTickCount Lib "kernel32" () As Long
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Global AdBuf(10000) As Single
Global Channel(32) As Integer
Global ConfigCode(32) As Integer
Global Buf(10000) As Integer
Global Buf1(10000) As Integer
Global Buf2(10000) As Integer
Global Card0Buf0(10000) As Integer
Global Card0Buf1(10000) As Integer
Global Card1Buf0(10000) As Integer
Global Card1Buf1(10000) As Integer
Global AdNumber As Integer
Global CR
Global LF
```

1.6.3 DEMO1.FRM

```
Public Sub ShowWave()
        Dim a(1000), yc, xc, xl, yt As Single
        Dim ii As Integer
        Dim tmpstr$

        Picture1.Cls
        yc = Picture1.ScaleTop + Picture1.ScaleHeight / 2
        xl = Picture1.ScaleLeft
        xs = Picture1.ScaleWidth / AdNumber
        ys = Picture1.ScaleHeight / 10
        Picture1.Line (xl, yc)-(xl + Picture1.ScaleWidth, yc), QBColor(4)
        Picture1.PSet (X1, yc - ys * AdBuf(0))
        For ii = 1 To AdNumber - 1
           Picture1.Line -(xl + (xs * ii), yc - (ys * AdBuf(ii)))
        Next ii
End Sub

Private Sub DA0Text_Change()
        DA0Text.Text = UCase(DA0Text.Text)
End Sub

Private Sub DA0Text_KeyPress(KeyAscii As Integer)
        If KeyAscii = 13 Then
           Call StartCMD_Click
        End If
End Sub

Private Sub DA1Text_Change()
        DA1Text.Text = UCase(DA1Text.Text)
End Sub


Private Sub DA1Text_KeyPress(KeyAscii As Integer)
        If KeyAscii = 13 Then
           Call StartCMD_Click
        End If
End Sub


Private Sub DoText_Change()
        DoText.Text = UCase(DoText.Text)
End Sub

Private Sub DoText_KeyPress(KeyAscii As Integer)
        If KeyAscii = 13 Then
           Call StartCMD_Click
        End If
End Sub

Private Sub ExitCMD_Click()
        Unload Me
End Sub

Private Sub Form_Load()
```

```
    Dim TotalBoards As Integer
    Dim RetValue As Integer

    CR = Chr$(13)
    LF = Chr$(10)
    RetValue = P180X_DriverInit(TotalBoards)
    If RetValue <> 0 Then
        ret = MsgBox("The Return Error Code = " + Str$(RetValue) + CR + LF +  _
        "The 180X Card Not Found !", 0, "P180X Return Error Code !")
      Exit Sub
    End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call P180X_DriverClose
End Sub

Private Sub StartCMD_Click()
    Dim V0 As Single
    Dim Didata As Integer
    Dim dadata As Integer
    Dim RetValue, ret, cc, Dodata As Integer

    AdNumber = 100
    RetValue = P180X_ActiveBoard(0)
    If RetValue <> 0 Then
      ret = MsgBox("The Return Error Code = " + Str$(RetValue) + CR + LF + "The 180X Card Not Found
      !", 0, "P180X Return Error Code !")
      Exit Sub
    End If
    Dodata = Val("&H" + DoText.Text)
    RetValue = P180X_Do(Dodata)
    If RetValue <> 0 Then
      ret = MsgBox("The Return Error Code = " + Str$(RetValue), 0, "P180X Return Error Code !")
      Exit Sub
    End If
    RetValue = P180X_Di(Didata)
    If RetValue <> 0 Then
      ret = MsgBox("The Return Error Code = " + Str$(RetValue), 0, "P180X Return Error Code !")
     Exit Sub
    End If
    DiText.Text = Hex(Didata)
    dadata = Val("&h" + DA0Text.Text)
    RetValue = P180X_Da(0, dadata)
    dadata = Val("&h" + DA1Text.Text)
    RetValue = P180X_Da(1, dadata)
    If RetValue <> 0 Then
      ret = MsgBox("The Return Error Code = " + Str$(RetValue), 0, "P180X Return Error Code !")
      Exit Sub
    End If
    RetValue = P180X_SetChannelConfig(0, 0) ' // +/- 5V range
    RetValue = RetValue + P180X_DelayUs(23) '       // delay 23 us settling time
    RetValue = RetValue + P180X_AdPolling(V0)



    If RetValue <> 0 Then
      ret = MsgBox("The Return Error Code = " + Str$(RetValue), 0, "P180X Return Error Code !")
      Exit Sub
    End If
```

```
    CH0Text.Text = Format(V0, "#0.000")

    RetValue = P180X_SetChannelConfig(1, 0) ' // +/- 5V range
    RetValue = RetValue + P180X_DelayUs(23) '       // delay 3 us settling time
    RetValue = RetValue + P180X_AdPolling(V0)
    If RetValue <> 0 Then
       ret = MsgBox("The Return Error Code = " + Str$(RetValue), 0, "P180X Return Error Code !")
       Exit Sub
    End If
    CH1Text.Text = Format(V0, "#0.000")
    RetValue = P180X_SetChannelConfig(0, 0) ' Ch:0, +/- 5V range
    RetValue = RetValue + P180X_DelayUs(23) '       // delay 3 us settling time
    RetValue = RetValue + P180X_AdsPolling(AdBuf(0), AdNumber) '

    Call ShowWave
End Sub
```

1.7 Using With Delphi

P180X.PAS	**→ unit file**
P180X.DLL	**→ DLLs**

NOTE : 1. Tested under **Windows 95/NT and Delphi 3.0 (32 bit)**

1.7.1 P180X.PAS

```
unit P180X;

interface

type PSingle=^Single;
type PWord=^Word;

const
// return code
 NoError                  =  0;
 DriverHandleError        =  1;
 DriverCallError          =  2;
 AdControllerError        =  3;
 M_FunExecError           =  4;
 ConfigCodeError          =  5;
 FrequencyCalculateError  =  6;
 HighAlarm                =  7;
 LowAlarm                 =  8;
 AdPollingTimeOut         =  9;
 AlarmTypeError           = 10;
 FindBoardError           = 11;
 AdChannelError           = 12;
 DaChannelError           = 13;
 InvalidDelay             = 14;
 DelayTimeOut             = 15;
 InvalidData          = 16;
 FifoOverflow             = 17;
 TimeOut                  = 18;
```

```
 ExceedBoardNumber      = 19;
 NotFoundBoard          = 20;
 OpenError              = 21;
 FindTwoBoardError      = 22;
 ThreadCreateError      = 23;
 StopError              = 24;
 AllocateMemoryError    = 25;

// Function of Test
function  P180X_FloatSub2(fA:Single; fB:Single):Single ; stdCall;
function  P180X_ShortSub2(nA:SmallInt; nB:SmallInt):SmallInt ; stdCall;
function  P180X_GetDllVersion:WORD  ; stdCall;

// Function of Driver
function      P180X_DriverInit(Var wTotalBoards:Word):WORD ; stdCall;
procedure     P180X_DriverClose; stdCall;
function      P180X_GetDriverVersion(var wDriverVersion:Word):WORD ; stdCall;

function  P180X_GetConfigAddressSpace(wBoardNo:Word;
          var wAddrTimer:Word; var wAddrCtrl:Word; var wAddrDio:Word;
          var wAddrAdda:Word):WORD ; stdCall;
function  P180X_ActiveBoard(wBoardNo:Word):WORD ; stdCall;
function  P180X_WhichBoardActive:WORD ; stdCall;

// Function of M_Fun series
function  P180X_M_FUN_1(wDaFrequency:WORD; wDaWave:WORD;
          fDaAmplitude:Single; wAdClock:WORD; wAdNumber:WORD; wAdConfig:WORD;
          fAdBuf:PSingle;    fLowAlarm:Single;
          fHighAlarm:Single):WORD ; stdCall;

function  P180X_M_FUN_2(wDaNumber:WORD;  wDaWave:WORD;
          wDaBuf:PWord; wAdClock:WORD; wAdNumber:WORD; wAdConfig:WORD;
          wAdBuf:PWord):WORD ; stdCall;

function  P180X_M_FUN_3(wDaFrequency:WORD;  wDaWave:WORD;
          fDaAmplitude:Single; wAdClock:WORD;  wAdNumber:WORD;
          wChannelStatus:PWord; wAdConfig:PWord;  fAdBuf:PSingle;
          fLowAlarm:Single; fHighAlarm:Single):WORD ; stdCall;
```

```
function  P180X_M_FUN_4(wType:WORD; wDaFrequency:WORD;
          wDaWave:WORD; fDaAmplitude:Single; wAdClock:WORD; wAdNumber:WORD;
          wChannelStatus:PWord; wAdConfig:PWord;
          fAdBuf:PSingle; fLowAlarm:Single;
          fHighAlarm:Single):WORD ; stdCall;


// Function of DI/DO
function  P180X_Do(wOutData:Word):Word; stdCall;
function  P180X_Di(var wDiData:Word):WORD ; stdCall;

// Function of AD/DA
function  P180X_Da(wDaChannel:Word; wDaVal:Word):WORD ; stdCall;
function  P180X_SetChannelConfig(wAdChannel:Word;
              wConfig:Word):WORD ; stdCall;
function  P180X_AdPolling(var fAdVal:Single):WORD ; stdCall;
function  P180X_AdsPolling(fAdVal:PSingle; wNum:Word):WORD ; stdCall;
function  P180X_AdsPacer(fAdVal:PSingle; wNum:Word;
                wSamplingDiv:Word ):WORD ; stdCall;

//*******************
function  P180X_ClearScan:WORD ; stdCall;
function  P180X_StartScan(wSampleRateDiv:WORD;   dwNum:LongInt;
              nPriority:SmallInt):WORD ; stdCall;
procedure P180X_ReadScanStatus(var wStatus:WORD;
              var dwLowAlarm:LongInt; var dwHighAlarm:LongInt); stdCall;
function  P180X_AddToScan(wAdChannel:WORD; wConfig:WORD;
              wAverage:WORD; wLowAlarm:WORD;  wHighAlarm:WORD;
              wAlarmType:WORD):WORD ; stdCall;
function  P180X_SaveScan(wAdChannel:WORD; wBuf:PWord):WORD ; stdCall;
procedure P180X_WaitMagicScanFinish(var wStatus:WORD;
     var dwLowAlarm:LongInt; var dwHighAlarm:LongInt); stdCall;
function  P180X_StopMagicScan:WORD ; stdCall;

//*******************
function  P180X_DelayUs(wDelayUs:WORD):WORD ; stdCall;
```

```
//*******************
//function  P180X_Card0_StartScan( wSampleRate:WORD;
//     wChannelStatus:PWORD; wChannelConfig:PWORD;
//     wCount:WORD):WORD ; stdCall;
function  P180X_Card0_StartScan( wSampleRate:WORD;
      wChannelStatus:PWORD;  wChannelConfig:PWORD;
      wCount:WORD):WORD ; stdCall;
function  P180X_Card0_ReadStatus(wBuf:PWORD; wBuf2:PWORD;
      var dwP1:LongInt; var dwP2:LongInt; var wStatus:WORD):WORD ; stdCall;
procedure P180X_Card0_Stop; stdCall;

function  P180X_Card1_StartScan(wSampleRate:WORD;
      wChannelStatus:PWORD; wChannelConfig:PWORD;
      wCount:WORD):WORD ; stdCall;
function  P180X_Card1_ReadStatus(wBuf:PWORD; wBuf2:PWORD;
      var dwP1:LongInt; var dwP2:LongInt;
      var wStatus:WORD):WORD ; stdCall;
procedure P180X_Card1_Stop; stdCall;


//*******************
function  P180X_FunA_Start( wClock0Div:WORD;     wChannel0:PWord;
          wConfig0:PWord;       Buffer0:PWord;          dwMaxCount0:LongInt;
          wClock1Div:WORD;    wChannel1:PWord;    wConfig1:PWord;
          Buffer1:PWord;          dwMaxCount1:LongInt; nPriority:SmallInt):WORD ; stdCall;
function  P180X_FunA_ReadStatus:WORD ; stdCall;
function  P180X_FunA_Stop:WORD ; stdCall;
function  P180X_FunA_Get(var P0:LongInt; var P1:LongInt):WORD ; stdCall;

//*******************
function  P180X_FunB_Start( wClock0Div:WORD; wChannel0:PWord;
           wConfig0:PWord;           Buffer0:PWord;
           dwMaxCount0:LongInt;    nPriority:SmallInt):WORD ; stdCall;
function  P180X_FunB_ReadStatus:WORD ; stdCall;
function  P180X_FunB_Stop:WORD       ; stdCall;
function  P180X_FunB_Get(var P0:LongInt):WORD ; stdCall;
```

```
//*****************************************************************************
//*****************************************************************************
implementation

function  P180X_FloatSub2;   external 'P180X.DLL' name 'P180X_FloatSub2';
function  P180X_ShortSub2;   external 'P180X.DLL' name 'P180X_ShortSub2';
function  P180X_GetDllVersion;
                    external 'P180X.DLL' name 'P180X_GetDllVersion';
function  P180X_GetDriverVersion;
                    external 'P180X.DLL' name 'P180X_GetDriverVersion';


function        P180X_DriverInit;  external 'P180X.DLL' name 'P180X_DriverInit';
procedure     P180X_DriverClose; external 'P180X.DLL' name 'P180X_DriverClose';
function        P180X_GetConfigAddressSpace;
                    external 'P180X.DLL' name 'P180X_GetConfigAddressSpace';
function  P180X_ActiveBoard; external 'P180X.DLL' name 'P180X_ActiveBoard';
function  P180X_WhichBoardActive;
                    external 'P180X.DLL' name 'P180X_WhichBoardActive';

// Function of M_Fun series
function  P180X_M_FUN_1;    external 'P180X.DLL' name 'P180X_M_FUN_1';
function  P180X_M_FUN_2;    external 'P180X.DLL' name 'P180X_M_FUN_2';
function  P180X_M_FUN_3;    external 'P180X.DLL' name 'P180X_M_FUN_3';
function  P180X_M_FUN_4;    external 'P180X.DLL' name 'P180X_M_FUN_4';


function  P180X_Do;             external 'P180X.DLL' name 'P180X_Do';
function  P180X_Di;             external 'P180X.DLL' name 'P180X_Di';


function  P180X_Da;             external 'P180X.DLL' name 'P180X_Da';
function  P180X_SetChannelConfig;
                    external 'P180X.DLL' name 'P180X_SetChannelConfig';
function  P180X_AdPolling;      external 'P180X.DLL' name 'P180X_AdPolling';
function  P180X_AdsPolling;     external 'P180X.DLL' name 'P180X_AdsPolling';
```

```
function  P180X_AdsPacer;     external 'P180X.DLL' name 'P180X_AdsPacer';

//**********************
function  P180X_ClearScan;          external 'P180X.DLL' name 'P180X_ClearScan';
function  P180X_StartScan;          external 'P180X.DLL' name 'P180X_StartScan';
procedure P180X_ReadScanStatus;           external 'P180X.DLL' name
          'P180X_ReadScanStatus';
function  P180X_AddToScan;        external 'P180X.DLL' name 'P180X_AddToScan';
function  P180X_SaveScan;         external 'P180X.DLL' name 'P180X_SaveScan';
procedure P180X_WaitMagicScanFinish;     external 'P180X.DLL' name
          'P180X_WaitMagicScanFinish';
function  P180X_StopMagicScan;    external 'P180X.DLL' name 'P180X_StopMagicScan';

//**********************
function  P180X_DelayUs;            external 'P180X.DLL' name 'P180X_DelayUs';

//********************
function  P180X_Card0_StartScan;       external 'P180X.DLL' name
          'P180X_Card0_StartScan';
function  P180X_Card0_ReadStatus;    external 'P180X.DLL' name
          'P180X_Card0_ReadStatus';
procedure P180X_Card0_Stop;           external 'P180X.DLL' name 'P180X_Card0_Stop';
function  P180X_Card1_StartScan;       external 'P180X.DLL' name
          'P180X_Card1_StartScan';
function  P180X_Card1_ReadStatus;    external 'P180X.DLL' name
          'P180X_Card1_ReadStatus';
procedure P180X_Card1_Stop;           external 'P180X.DLL' name 'P180X_Card1_Stop';

//********************
function  P180X_FunA_Start;             external 'P180X.DLL' name 'P180X_FunA_Start';
function  P180X_FunA_ReadStatus;     external 'P180X.DLL' name
          'P180X_FunA_ReadStatus';
function  P180X_FunA_Stop;              external 'P180X.DLL' name 'P180X_FunA_Stop';
function  P180X_FunA_Get;               external 'P180X.DLL' name 'P180X_FunA_Get';
```

```
//********************
function  P180X_FunB_Start;              external 'P180X.DLL' name 'P180X_FunB_Start';
function  P180X_FunB_ReadStatus;         external 'P180X.DLL' name
          'P180X_FunB_ReadStatus';
function  P180X_FunB_Stop;               external 'P180X.DLL' name 'P180X_FunB_Stop';
function  P180X_FunB_Get;                external 'P180X.DLL' name 'P180X_FunB_Get';


end.
```

1.8 Calling the DLL from LabVIEW

LabVIEW is a graphical programming language developed by National Instruments.

P180X.Dll	**→ DLL**
DEMO1.VI	**→ Demo VI**
MFUN1.VI	**→ Driver VI**

NOTE :

1. Tested under **Windows 95/NT and LabVIEW 4.0**
2. The demo1.VI will call MFUN1.VI to perform M_Functions. The M_Functions can send out an arbitrary waveform to the D/A output port and perform the A/D operation at the same time. If D/A channel_0 is connected to A/D channel_0, M_Functions can measure the D/A output back. The output response is shown in Fig 8 and the connection diagram of demo1.VI is given in Fig. 9.

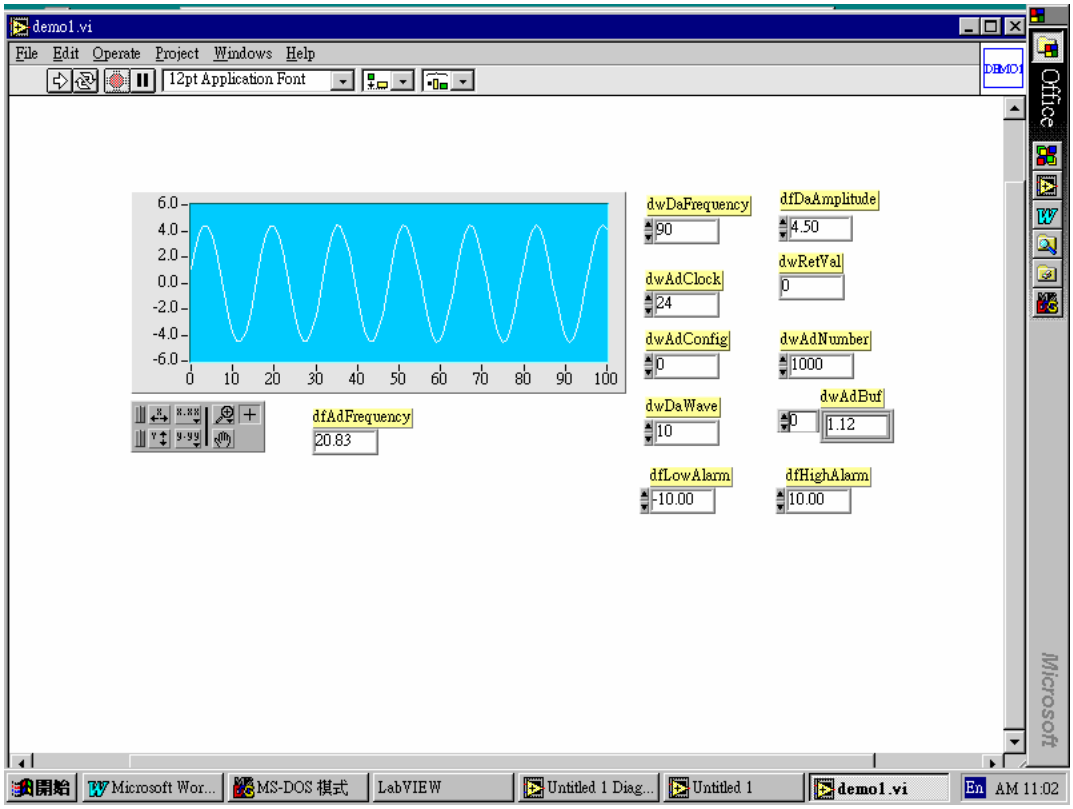


Fig 8. The output of DEMO1.VI (call M_FUN_1)

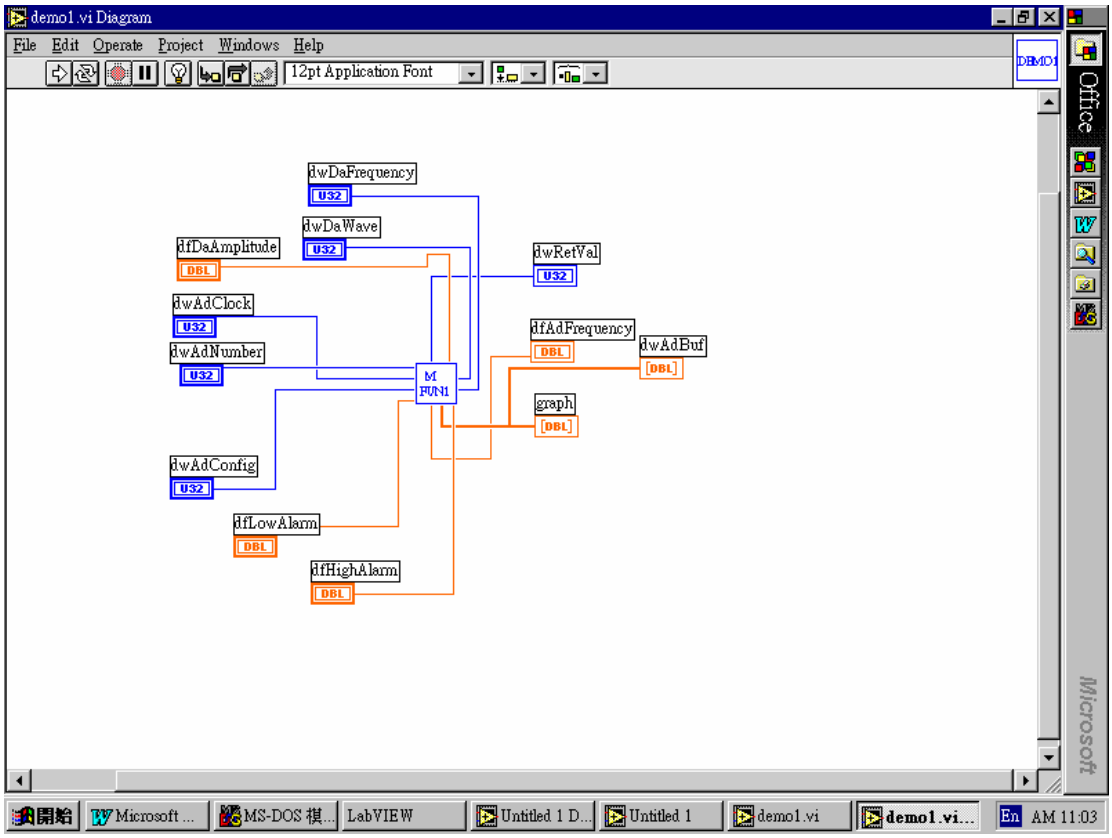


Fig 9. The connection diagram of DEMO1.VI (call MFUN1.VI)

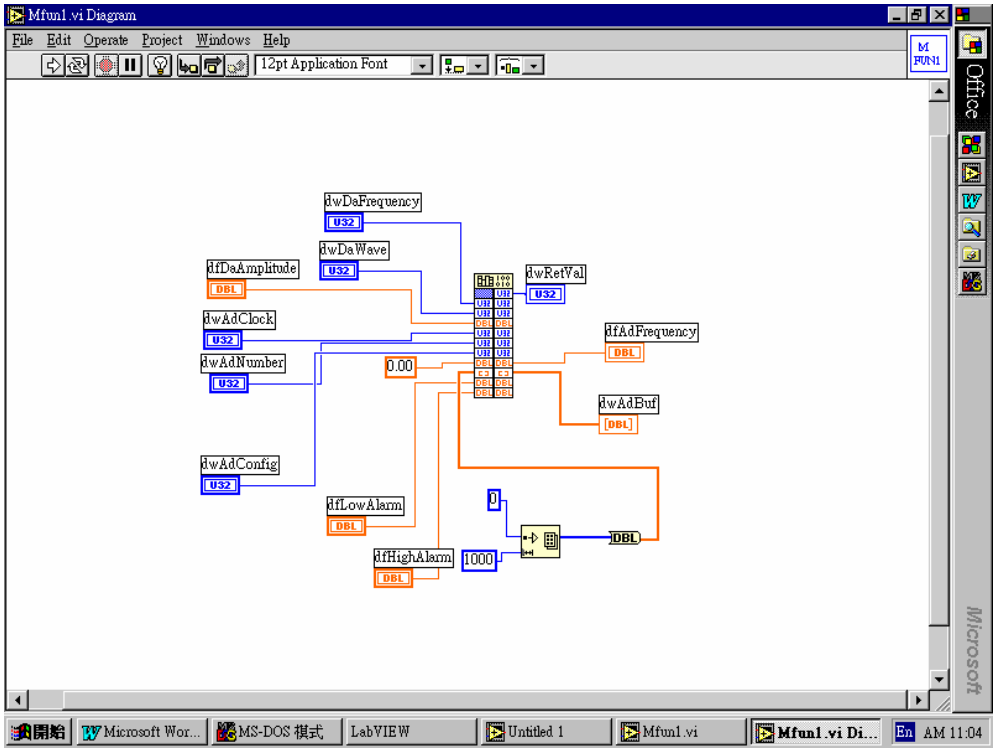


Fig 10. The connection diagram of MFUN1.VI (call DLL M_FUN_1)

1.9 DEMO PROGRAM

We use a common demo program for all p180X.dll demostrations. This demo program will accept the **wDaFreq** and **wAdClk** and call the different functions for used for demonstration.

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "P180X.H"
/*********************************************************************/
/*    DEMO1 program for one P180X card in the PC system.             */
/*   Please set the resolution of your monitor to at least 1024*768.  */
/*********************************************************************/
/* First Card: some P180X function call demo.                        */
/* For the proper operation the P180X, the following functions       */
/* must be used.                                                     */
/*    P180X_DriverInit();     <--   initial the driver               */
/*    P180X_DriverClose();    <--   close the driver                 */
/*********************************************************************/

short nDMA=-1, nIRQ=-1;      // not used
WORD      wBase=0x220,wAdBuf[510],wFlag=0,wAddrCtrl;
int     iLine;

DWORD   dwDaNum=90,dwAdClk=24;
WORD      wTotalBoard,wInitialCode;


void  READ_CMD(char *);
short ASCII_TO_HEX(char);
void  TEST_CMD(HWND, int, int, int, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
void SHOW_WAVE(HWND hwnd);


/* ---------------------------------------------------------------- */

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static char szAppName[] = "P180X Demo1";
    HWND  hwnd ;
    MSG     msg ;
    WNDCLASSEX  wndclass ;

    wndclass.cbSize      = sizeof(wndclass);
    wndclass.style       = CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc   = WndProc;
    wndclass.cbClsExtra    = 0;
    wndclass.cbWndExtra    = 0;
    wndclass.hInstance     = hInstance;
    wndclass.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.hCursor     = LoadCursor(NULL, IDC_ARROW);
```

```
    wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName  = NULL;
    wndclass.lpszClassName = szAppName;
    wndclass.hIconSm           = LoadIcon(NULL, IDI_APPLICATION);



    RegisterClassEx(&wndclass) ;
    hwnd=CreateWindow(szAppName,"P180X Demo1 Program",
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInstance, NULL) ;
    ShowWindow(hwnd,SW_SHOWMAXIMIZED);
    UpdateWindow(hwnd);

    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}

/* ---------------------------------------------------------------- */

LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
       static int  cxChar, cyChar, cxClient, cyClient,cxBuffer;
       static int  cyBuffer, xCaret, yCaret;
       static char         cBuf[80];
       HDC                 hdc;
       TEXTMETRIC   tm;
       PAINTSTRUCT  ps;
       int                 i;

       switch (iMsg)
       {
         case WM_CREATE :  // window initial
          /*****************************************************************/
/* NOTICE: call P180X_DriverInit() to initialize the driver. */
/*****************************************************************/
// Initial the device driver, and return the board number in the PC
       wInitialCode=P180X_DriverInit(&wTotalBoard);
       if( wInitialCode!=NoError )
       {
                     MessageBox(hwnd,"No P180X card in this system !!!",
                                   "P180X Card Error",MB_OK);
       }
       hdc=GetDC(hwnd);
       SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));
       GetTextMetrics(hdc, &tm);
       cxChar=tm.tmAveCharWidth;
       cyChar=tm.tmHeight;
       ReleaseDC(hwnd, hdc);
       return 0;
  case WM_SIZE :
       cxClient=LOWORD(lParam);       // window size in pixels
       cyClient=HIWORD(lParam);
```

```
    cxBuffer=max(1,cxClient/cxChar); // window size in characters
    cyBuffer=max(1,cyClient/cyChar);
    return 0;
case WM_SETFOCUS :
    CreateCaret(hwnd, NULL, cxChar, cyChar);
    SetCaretPos(xCaret * cxChar, yCaret * cyChar);
    ShowCaret(hwnd);
    return 0;
case WM_KILLFOCUS :
    HideCaret(hwnd);
    DestroyCaret();
    return 0;

case WM_CHAR :   // user press KEYBOARD
    for (i = 0 ; i < (int) LOWORD(lParam) ; i++)
    {
        switch (wParam)
          {
           case '\b' :                  // backspace pressed
               if (xCaret > 0)
               {
                 xCaret-- ;
                 cBuf[xCaret]=' ';
                 HideCaret(hwnd);
                 hdc=GetDC(hwnd);
                 SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));
                    TextOut(hdc, xCaret * cxChar, yCaret * cyChar,cBuf+xCaret,1);
                    ShowCaret(hwnd);
                    ReleaseDC(hwnd, hdc);
           }
               break;
           case '\r' :                  // carriage return  pressed
               if (wFlag==1)
               {
                     InvalidRect(hwnd, NULL, TRUE);
                     wFlag=0;
                     break;
               }
               wFlag=1;
            cBuf[xCaret]=0;
               if (xCaret!=0) {xCaret=0; yCaret++;}

               READ_CMD(cBuf);
            TEST_CMD(hwnd,xCaret, cxChar, yCaret,cyChar);

            xCaret=0; yCaret+=iLine;
               if (yCaret >= cyBuffer) InvalidRect(hwnd, NULL, TRUE);
            break ;
           case '\x1B' :                // escape pressed
            InvalidRect (hwnd, NULL, TRUE) ;
            xCaret=yCaret=0;
            break;
          default :                 // other KEY pressed
              cBuf[xCaret]=(char) wParam;
           HideCaret(hwnd);
           hdc=GetDC (hwnd);
           SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));
           TextOut(hdc,xCaret*cxChar,yCaret*cyChar,cBuf+xCaret,1);
         ShowCaret(hwnd);
           ReleaseDC(hwnd, hdc);
```

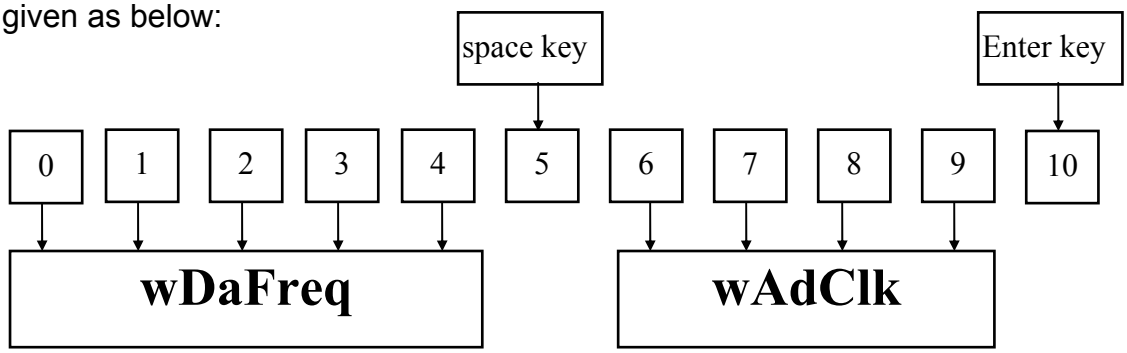
```
                xCaret++;
             break ;
            }
      }
      SetCaretPos(xCaret*cxChar, yCaret*cyChar);
      return 0;
  case WM_PAINT :                  // clr and show HELP
      InvalidRect(hwnd, NULL, TRUE);
      hdc=BeginPaint(hwnd, &ps);
      SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));

      sprintf(cBuf,"Press any key to continue");
      TextOut(hdc,0,0,cBuf,strlen(cBuf));
      xCaret = 0 ; yCaret=1;
      SetCaretPos(0,yCaret*cyChar);

      EndPaint(hwnd, &ps);
      return 0;

  case WM_DESTROY :
 /*****************************************************************/
/* NOTICE: call P180X_DriverClose() to close the driver.      */
/*****************************************************************/
      P180X_DriverClose();         // close the driver
      PostQuitMessage(0);
      return 0 ;
  }
  return DefWindowProc(hwnd, iMsg, wParam, lParam);
}

/* ------------------------------------------------------------------- */
/* [0][1][2][3][4]=wII, [6][7][8][9]=dwAdClk                           */
void READ_CMD(char szCmd[])
{
   DWORD nT1,nT2,nT3,nT4,nT5;

   if(szCmd[0]==0) return;     // only press [Enter]

   nT1=ASCII_TO_HEX(szCmd[0]);       // HEX format
   nT2=ASCII_TO_HEX(szCmd[1]);
   nT3=ASCII_TO_HEX(szCmd[2]);
   nT4=ASCII_TO_HEX(szCmd[3]);
   nT5=ASCII_TO_HEX(szCmd[4]);
   dwDaNum=nT1*10000+nT2*1000+nT3*100+nT4*10+nT5;

   nT1=ASCII_TO_HEX(szCmd[6]);       // HEX format
   nT2=ASCII_TO_HEX(szCmd[7]);
   nT3=ASCII_TO_HEX(szCmd[8]);
   nT4=ASCII_TO_HEX(szCmd[9]);
   dwAdClk=(DWORD)(nT1*1000+nT2*100+nT3*10+nT4);
}

short ASCII_TO_HEX(char cChar)
{
   if(cChar<='9') return(cChar-'0');
   else if (cChar<='F') return(cChar-'A'+10);
   else return(cChar-'a'+10);
}
```

```
/* ---------------------------------------------------------------- */
void TEST_CMD(HWND hwnd, int x, int dx, int y, int dy)
{
```

Test subroutine should be placed here

```
}
```

The READ_COM only accept **fix format** command. The command format is given as below:



- if [0] = [Enter key] → accept current setting of **wDaFreq** and **wAdClk**

The steps to compile and link the demo program are described in Sec. 1.3. All demo programs share the similar interface. The separate testing subroutines are placed in "**TEST_CMD(….) { …….}**". So, only the **TEST_CMD** is listed in the user manual.

2. DESCRIPTION OF FUNCTIONS

The DLL functions are divided into the following groups:

- The Test functions
- The M_Functions function
- The D/I/O functions
- The D/A function
- The A/D fixed-mode functions
- The A/D MagicScan mode functions
- The Batch capture functions
- The Continuous capture functions
- The Plug & Play functions
- Other functions

- The fixed-channel mode functions are as follows:

1. P180X_SetChannelConfig
2. P180X_AdPoling
3. P180X_AdsPolling
4. P180X_AdsPacer

data in float format

- The MagicScan mode functions are as follows:

1. P180X_ClearScan
2. P180X_StartScan
3. P180X_AddToScan
4. P180X_SaveScan
5. P180X_ReadMagicScanResult

data in 12 bits HEX format

- The M_functions functions are as follows:

1. P180X_M_FUN_1
2. P180X_M_FUN_2
3. P180X_M_FUN_3

- The multi-board batch capture functions are as follows: (two boards operating simultaneously)

1. P180X_FunA_Start
2. P180X_FunA_ReadStatus
3. P180X_FunA_Stop
4. P180X_FunA_Get

- The single board batch capture functions are as follows:

1. P180X_FunB_Start
2. P180X_FunB_ReadStatus
3. P180X_FunB_Stop
4. P180X_FunB_Get

- The continuous capture functions are given as follows:

1. P180X_Card0_StartScan
2. P180X_Card0_ReadStatus
3. P180X_Card0_StopScan
4. P180X_Card1_StartScan
5. P180X_Card1_ReadStatus
6. P180X_Card1_StopScan

Group-0: for card_0 continuous capture function

Group-1: for card_1 continuous capture function

2.1 CONFIGURATION CODE

PCI-180XL Configuration Code Table

Bipolar/Unipolar	Input Signal Range	Gain	Settling Time	Configuration Code
Bipolar	+/- 5V	1	3 us	0x00
Bipolar	+/- 2.5V	2	3 us	0x01
Bipolar	+/- 1.25V	4	3 us	0x02
Bipolar	+/- 0.625V	8	3 us	0x03
Bipolar	+/- 10V	0.5	3 us	0x04
Bipolar	+/- 5V	1	3 us	0x05
Bipolar	+/- 2.5V	2	3 us	0x06
Bipolar	+/- 1.25V	4	3 us	0x07
Unipolar	0V to 10V	1	3 us	0x08
Unipolar	0V to 5V	2	3 us	0x09
Unipolar	0V to 2.5V	4	3 us	0x0A
Unipolar	0V to 1.25V	8	3 us	0x0B

PCI-180XH Configuration Code Table

Bipolar/Unipolar	Input Signal Range	Gain	Settling Time	Configuration Code
Bipolar	+/- 5V	1	23 µs	0x10
Bipolar	+/- 0.5V	10	28 µs	0x11
Bipolar	+/- 0.05V	100	140 µs	0x12
Bipolar	+/- 0.005V	1000	1300 µs	0x13
Bipolar	+/- 10V	0.5	23 µs	0x14
Bipolar	+/- 1V	5	28 µs	0x15
Bipolar	+/- 0.1V	50	140 µs	0x16
Bipolar	+/- 0.01V	500	1300 µs	0x17
Unipolar	0V to 10V	1	23 µs	0x18
Unipolar	0V to 1V	10	28 µs	0x19
Unipolar	0V to 0.1V	100	140 µs	0x1A
Unipolar	0V to 0.01V	1000	1300 µs	0x1B

2.2 ERROR CODE

Error Code	Description
NoError	OK! No Error!
DriverHandleError	Device driver open error
DriverCallError	Error while calling the driver functions
AdControllerError	Embedded controller handshake error
M_FunExecError	Failed to execute the M_Functions
ConfigCodeError	Refer to "Section 2.1 Configuration Code"
FrequencyCalculateError	(Not Used)
HighAlarm	fAdBuf[?]>fHighAlarm
LowAlarm	fAdBuf[?]< fLowAlarm
AdPollingTimeOut	Hardware timeout error
AlarmTypeError	The valid range is : 0 to 4
FindBoardError	Can't find the PCI-180X board in the system
AdChannelError	The valid range is 0 to 31
DaChannelError	The valid channel number must be 0 or 1
InvalidDelay	The valid range of dwDelayUs is <= 8191
DelayTimeOut	Timeout
InvalidData	Invalid Data
FifoOverflow	FIFO overflow
TimeOut	Timeout
ExceedBoardNumber	Invalid board number (Valid range: 0 to TotalBoards -1)
NotFoundBoard	Can't detect any PCI-180X boards in the system
OpenError	(Not Used)
FindTwoBoardError	Can't find two PCI-180X boards
ThreadCreateError	Failed to create thread
StopError	Stop Error
AllocateMemoryError	Failed to allocate the memory buffer

2.3 P180X.H

```
#define EXPORTS extern "C" __declspec (dllimport)  // Usage for Allpication
// #define EXPORTS                                 // Usage for DLL

//----------priority setting constant---------------------------------
//
//        THREAD_PRIORITY_LOWEST
//        THREAD_PRIORITY_BELOW_NORMAL
//        THREAD_PRIORITY_NORMAL
//        THREAD_PRIORITY_ABOVE_NORMAL
//        THREAD_PRIORITY_HIGHEST
//
//----------priority setting constant---------------------------------

// return code
#define NoError                 0
#define DriverHandleError       1
#define DriverCallError         2
#define AdControllerError       3
#define M_FunExecError          4
#define ConfigCodeError         5
#define FrequencyCalculateError 6
#define HighAlarm               7
#define LowAlarm                8
#define AdPollingTimeOut        9
#define AlarmTypeError          10
#define FindBoardError          11
#define AdChannelError          12
#define DaChannelError          13
#define InvalidDelay            14
#define DelayTimeOut            15
#define InvalidData             16
#define FifoOverflow            17
#define TimeOut                 18
#define ExceedBoardNumber       19
#define NotFoundBoard           20
#define OpenError               21
#define FindTwoBoardError       22
#define ThreadCreateError       23
#define StopError               24
#define AllocateMemoryError     25

EXPORTS float       CALLBACK P180X_FloatSub2(float fA, float fB);
EXPORTS short       CALLBACK P180X_ShortSub2(short nA, short nB);
EXPORTS WORD    CALLBACK P180X_GetDllVersion(void);

EXPORTS WORD    CALLBACK P180X_DriverInit(WORD *wTotalBoards);
EXPORTS void        CALLBACK P180X_DriverClose(void);
EXPORTS WORD    CALLBACK P180X_GetDriverVersion(WORD *wVxdVersion);

EXPORTS WORD    CALLBACK P180X_GetConfigAddressSpace(WORD wBoardNo,
            WORD *wAddrTimer,WORD *wAddrCtrl, WORD *wAddrDio, WORD *wAddrAdda);

EXPORTS WORD    CALLBACK P180X_ActiveBoard( WORD wBoardNo );
EXPORTS WORD    CALLBACK P180X_WhichBoardActive(void);
```

```
EXPORTS WORD    CALLBACK P180X_M_FUN_1(WORD wDaFrequency,
            WORD wDaWave, float fDaAmplitude, WORD wAdClock, WORD wAdNumber,
            WORD  wAdConfig, float fAdBuf[], float fLowAlarm, float fHighAlarm);

EXPORTS WORD   CALLBACK P180X_M_FUN_2(WORD wDaNumber, WORD wDaWave,
           WORD  wDaBuf[], WORD wAdClock, WORD wAdNumber,
           WORD  wAdConfig, WORD wAdBuf[]);

EXPORTS WORD   CALLBACK P180X_M_FUN_3(WORD wDaFrequency,
            WORD wDaWave, float fDaAmplitude, WORD wAdClock, WORD wAdNumber,
            WORD wChannelStatus[], WORD wAdConfig[],
            float fAdBuf[], float fLowAlarm, float fHighAlarm);

EXPORTS WORD   CALLBACK P180X_M_FUN_4(WORD wType, WORD wDaFrequency,
            WORD wDaWave, float fDaAmplitude, WORD wAdClock, WORD wAdNumber,
            WORD  wChannelStatus[], WORD wAdConfig[],
            float fAdBuf[], float fLowAlarm, float fHighAlarm);

EXPORTS WORD  CALLBACK P180X_Di(WORD *wDi);
EXPORTS WORD  CALLBACK P180X_Do(WORD wDo);

EXPORTS WORD  CALLBACK P180X_Da(WORD wDaChannel, WORD wDaVal);
EXPORTS WORD  CALLBACK P180X_SetChannelConfig(WORD wAdChannel,
            WORD wConfig);

EXPORTS WORD  CALLBACK P180X_AdPolling(float *fAdVal);
EXPORTS WORD  CALLBACK P180X_AdsPolling(float fAdVal[], WORD wNum);
EXPORTS WORD  CALLBACK P180X_AdsPacer(float  fAdVal[],       WORD wNum,
                                      WORD wSample);

EXPORTS WORD  CALLBACK P180X_ClearScan(void);
EXPORTS WORD  CALLBACK P180X_StartScan(WORD wSampleRateDiv,
            DWORD dwNum, SHORT nPriority);
EXPORTS void  CALLBACK P180X_ReadScanStatus(WORD *wStatus,
            DWORD *dwLowAlarm, DWORD *dwHighAlarm);
EXPORTS WORD  CALLBACK P180X_AddToScan(WORD wAdChannel, WORD wConfig,
            WORD wAverage, WORD wLowAlarm, WORD wHighAlarm, WORD wAlarmType);
EXPORTS WORD  CALLBACK P180X_SaveScan(WORD wAdChannel, WORD wBuf[]);
EXPORTS void  CALLBACK P180X_WaitMagicScanFinish(WORD *wStatus,
            DWORD *dwLowAlarm, DWORD *dwHighAlarm);
EXPORTS WORD  CALLBACK P180X_StopMagicScan();

EXPORTS WORD  CALLBACK P180X_DelayUs(WORD wDelayUs);

EXPORTS WORD  CALLBACK P180X_Card0_StartScan(WORD wSampleRate,
            WORD wChannelStatus[], WORD wChannelConfig[],WORD wCount);
EXPORTS WORD  CALLBACK P180X_Card0_ReadStatus(WORD wBuf[], WORD wBuf2[],
            DWORD *dwP1, DWORD *dwP2, WORD *wStatus);
EXPORTS void  CALLBACK P180X_Card0_Stop(void);

EXPORTS WORD  CALLBACK P180X_Card1_StartScan(WORD wSampleRate,
            WORD wChannelStatus[],WORD wChannelConfig[],WORD wCount);
EXPORTS WORD  CALLBACK P180X_Card1_ReadStatus(WORD wBuf[], WORD wBuf2[], DWORD *dwP1,
            DWORD *dwP2,WORD *wStatus);
EXPORTS void  CALLBACK P180X_Card1_Stop(void);
```

```
EXPORTS WORD  CALLBACK P180X_FunA_Start(WORD wClock0Div, WORD wChannel0[],
            WORD  wConfig0[], WORD *Buffer0, DWORD dwMaxCount0,
            WORD  wClock1Div, WORD wChannel1[],WORD wConfig1[],
            WORD  *Buffer1, DWORD dwMaxCount1, SHORT nPriority);
EXPORTS WORD  CALLBACK P180X_FunA_ReadStatus(void);
EXPORTS WORD  CALLBACK P180X_FunA_Stop(void);
EXPORTS WORD  CALLBACK P180X_FunA_Get(DWORD *P0, DWORD *P1);

EXPORTS WORD  CALLBACK P180X_FunB_Start(WORD wClock0Div, WORD wChannel0[],
            WORD  wConfig0[], WORD *Buffer0, DWORD dwMaxCount0, SHORT nPriority);
EXPORTS WORD  CALLBACK P180X_FunB_ReadStatus(void);
EXPORTS WORD  CALLBACK P180X_FunB_Stop(void);
EXPORTS WORD  CALLBACK P180X_FunB_Get(DWORD *P0);

EXPORTS WORD  CALLBACK P180X_StartScanPostTrg(WORD wSampleRateDiv,
            DWORD dwNum, SHORT nPriority);
EXPORTS WORD  CALLBACK P180X_StartScanPreTrg(WORD wSampleRateDiv,
            DWORD dwNum, SHORT nPriority);
EXPORTS WORD  CALLBACK P180X_StartScanMiddleTrg(WORD wSampleRateDiv,
            DWORD dwN1, DWORD dwN2, SHORT nPriority);
EXPORTS WORD  CALLBACK P180X_StartScanPreTrgVerC(WORD wSampleRateDiv,
            DWORD dwNum, SHORT nPriority);
EXPORTS WORD  CALLBACK P180X_StartScanMiddleTrgVerC(WORD wSampleRateDiv, DWORD dwN1,
            DWORD dwN2, SHORT nPriority);
```

2.4 TEST FUNCTIONS

2.4.1 P180X_FloatSub2

- **Description:**

  Calculates C=A-B in **float** format, **float=4 bytes floating point number.** This function is provided to test DLL linkage.

- **Syntax:**

  float P180X_FloatSub2(float fA, float fB);

- **Parameter :**

  fA : 4 bytes floating point value

  fB : 4 bytes floating point value

- **Return :**

  return=fA-fB

2.4.2 P180X_ShortSub2

- **Description :**

  Calculates C=A-B in SHORT format, **SHORT=16 bit signed number.** This function is provided to test DLL linkage.

- **Syntax :**

  short P180X_ShortSub2(Short nA, Short nB);

- **Parameter :**

  nA : 16 bit value

  nB : 16 bit value

- **Return :**

  return=nA-nB

2.4.3 P180X_GetDllVersion

- **Description :**
  Reads the DLL version number of the **P180X.DLL**.

- **Syntax :**
  WORD P180X_GetDllVersion(void);

- **Parameter :**
  None

- **Return :**
  return=0x200 → Version 2.0

2.4.4 P180X_GetDriverVersion

- **Description :**
  This subroutine will read the software version number of NAPPCI.VxD for Windows 95 or NAPPCI.SYS of Windows NT.

- **Syntax :**
  WORD P180X_GetDriverVersion(WORD *wDriverVersion);

- **Parameter :**
  *wDriverVersion : address of **wDriverVersion**
  wDriverVersion=0x200 → Version 2.0

- **Return :**
  NoError: OK
  DriverHandleError: the NAPPCI.VxD open error for Windows 95
  the NAPWNT.SYS open error for Windows NT/2000/XP
  DriverCallError: call NAPPCI.VxD return error
  call NAPWNT.SYS return error

2.5 The M_Functions

2.5.1 P180X_M_FUN_1

- **Description :**

  The P180X_M_FUN_1 will calculate the waveform image automatically. (Refer to "OME-PCI-1800/1802 Hardware Manual" chapter-5 for details) (input=A/D channel_0, output=D/A channel_0)

- **Syntax :**

  WORD P180X_M_FUN_1(WORD wDaFrequency, WORD wDaWave,
  float fDaAmplitude, WORD wAdClock, WORD wAdNumber,
  WORD wAdConfig, float fAdBuf[], float fLowAlarm, float fHighAlarm)

- **Parameter :**

wDaFrequency	: **D/A output frequency = 1.8M/wDaFrequency (Pentium 120MHz)**
wDaWave	: Number of D/A waveforms to be output
fDaAmplitude	: Amplitude of D/A output. NOTE: the hardware jumper J1 must be set to +/-10V
wAdClock	: **A/D sampling clock = 8000000/wAdClock** samples/sec
wAdNumber	: Number of A/D data points to be read
wAdConfig	: **A/D input range configuration code** Refer to "Section 2.1 Configuration Table"
fAdBuf[]	: the starting address of **fAdBuf** which contains the A/D data
fLowAlarm	: low alarm limit. if **fAdBuf[?]< fLowAlarm** → LowAlarm
fHighAlarm	: high alarm limit. if **fAdBuf[?]>fHighAlarm** → HighAlarm

- **Return :**

  NoError: OK
  DriverHandleError: Invalidate VxD/SYS handle
  DriverCallError: VxD/SYS function call error
  ExceedBoardNumber: invalidate board number
  FindBoardError: no PCI-180X board
  AdControllerError: embedded controller handshake error
  M_FunExecError: M_Functions return code error
  ConfigCodeError: **wAdConfig** configuration code error

HighAlarm: **fAdBuf[?]>fHighAlarm**

LowAlarm: **fAdBuf[?]< fLowAlarm**

- **Demo Program :**

  DEMO5.C

2.5.2 P180X_M_FUN_2

- **Description :**

  The P180X_M_FUN_2 will **not** calculate the waveform image automatically. (Refer to “OME-PCI-1800/1802 Hardware Manual” chapter-5 for details) (input=A/D channel_0, output=DA channel_0)

- **Syntax :**

  WORD P180X_M_FUN_2(WORD wDaNumber, WORD wDaWave, WORD wDaBuf[],
  WORD wAdClock, WORD wAdNumber, WORD wAdConfig, WORD wAdBuf[]);

- **Parameter :**

  wDaNumber : Number of D/A samples in one waveform
  wDaWave : Number of D/A waveforms to be output
  wDaBuf[] : The array that contains the D/A waveform image
  wAdClock : **A/D sampling clock = 8000000/wAdClock** samples/sec
  wAdNumber : Number of A/D data points to be read
  wAdConfig : **A/D input range configuration code.**
  Refer to "Section 2.1 Configuration Table"
  wAdBuf[] : the starting address of **fAdBuf** which contains the A/D data

- **Return :**

  NoError : OK
  DriverHandleError : Invalid VxD/SYS handle
  DriverCallError : VxD/SYS function call error
  ExceedBoardNumber: invalidate board number
  FindBoardError: no OME-PCI-180X board
  AdControllererror : embedded controller handshake error
  M_FunExecError : M_Functions return code error
  ConfigCodeError : **wAdConfig** configuration code error

- **Demo Program :**

  DEMO7.C

The D/A output waveform generator is a **machine dependent** function. The D/A output frequency **= 1.8M/wDaNumber** is machine dependent. The following benchmarks have been established:

DA output frequency = 1.8M/dwDaNumber for Pentium 120

DA output frequency = 2.0M/dwDaNumber for Pentium 133

The user must benchmark their system before using M_FUN_1, M_FUN_2 and M_FUN_3.

2.5.3 P180X_M_FUN_3

- **Description :**

The P180X_M_FUN_3 will calculate the waveform image automatically. (Refer to "OME-PCI-1800/1802 Hardware Manual" chapter-5 for details) (input=programable channels, output=DA channel_0) This function will refer to the current active PCI-180X board. Use P180X_ActiveBoard(….) to select the active board. Refer to Sec. 2.4.2 for further details.

- **Syntax :**

```
WORD P180X_M_FUN_3(WORD wDaFrequency, WORD wDaWave,
    float fDaAmplitude, WORD wAdClock, WORD wAdNumber,
    WORD wChannelStatus[], WORD wAdConfig[], float fAdBuf[],
    float fLowAlarm, float fHighAlarm)
```

- **Parameter :**

wDaFrequency	: D/A **output frequency = 1.8M/wDaFrequency (Pentium 120)**
wDaWave	: Number of D/A waveforms to be output
fDaAmplitude	: Amplitude of D/A output. NOTE : the hardware jumper J1 must be set to +/-10V
wAdClock	: **A/D sampling clock = 8000000/wAdClock** samples/sec
wAdNumber	: Number of A/D data points to be read
wAdChannel[]	: 1=scan, 0=no scan
wAdConfig[]	: **configuration code**
fAdBuf[]	: the starting address of **fAdBuf** which contains the A/D data
fLowAlarm	: low alarm limit. if **fAdBuf[?]< fLowAlarm** → LowAlarm
fHighAlarm	: high alarm limit. if **fAdBuf[?]>fHighAlarm** → HighAlarm

- **Return :**

NoError : OK
DriverHandleError : Invalid VxD/SYS handle
DriverCallError : VxD/SYS function call error
ExceedBoardNumber: invalidate board number
FindBoardError: no OME-PCI-180X board
AdControllerError : embedded controller handshake error
M_FunExecError : M_Functions return code error
ConfigCodeError : **wAdConfig** configuration code error

HighAlarm : **fAdBuf[?]>fHighAlarm**

LowAlarm : **fAdBuf[?]< fLowAlarm**

- **Demo Program :**

  DEMO9.C

2.6 The DIO Functions

2.6.1 P180X_Di

- **Description :**

  This subroutine will read the 16 bit data from the digital input (D/I) port. This function will refer to the current active PCI-180X. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  WORD P180X_Di(WORD *wDi);

- **Parameter :**

  *wDi : address of **wDi** which contains the 16 bit D/I data

- **Return :**

  NoError : OK
  FindBoardError : cannot find the OME-PCI-180X board
  ExceedBoardNumber: invalid board number

- **Demo Program :**

  DEMO1.C

2.6.2 P180X_Do

- **Description :**

  This subroutine will send the 16 bit data to the digital output (D/O) port. This function will refer to the current active PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  WORD P180X_Do(WORD wDo);

- **Parameter :**

  wDo : the 16 bit data sent to the D/O port

- **Return :**

  NoError : OK
  ExceedBoardNumber: invalid board number
  FindBoardError : cannot find the OME-PCI-180X board

- **Demo Program :**

  DEMO1.C

2.7 The D/A Functions

2.7.1 P180X_Da

- **Description :**

  This subroutine will send the 12 bit data to the analog output (D/A) port. This function will refer to the current active PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  WORD P180X_Da(WORD wChannel, WORD wDaVal);

- **Parameter :**

  wChannel : 0 for channel_0 D/A, 1 for channel_1 D/A

  wDaVal : 12 bit data sent to D/A port. 0=minimum and 4095=maximum. The D/A output can be +/- 5V or +/- 10V by setting hardware jumper JP1. The software cannot detect the state of JP1.

- **Return :**

  NoError : OK

  FindBoardError : cannot find the OME-PCI-180X board

  ExceedBoardNumber: invalid board number

  DaChannelError : channel number must be 0 or 1

- **Demo Program :**

  DEMO1.C

2.8 The A/D Fixed-mode Functions

2.8.1 P180X_SetChannelConfig

- **Description :**

  This subroutine will set the A/D channel's configuration code. This subroutine will set the active A/D channel for **P180X_AdPolling**, **P180X_AdsPolling** and **P180X_AdsPacer**. This function will refer to the current active PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  WORD P180X_SetChannelConfig(WORD wChannel, WORD wConfig);

- **Parameter :**

  wChannel : A/D channel number

  wConfig : Configuration code. Refer to "Section 2.1 Configuration Table" for details.

- **Return :**

  NoError : OK

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-180X board

  AdControllerError : MagicScan controller hardware handshake error

- **Demo Program :**

  DEMO1.C

2.8.2 P180X_AdPolling

- **Description :**

  This subroutine will perform a single A/D conversion by software polling. The **P180X_SetChannelConfig** subroutine can be used to change channel or configuration code. This function will refer to the current active PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  WORD P180X_AdPolling(float *fAdVal);

- **Parameter :**

  *fAdVal : address of **fAdVal** which contains the A/D data, this data is automatically calculated based on the setting of **P180X_SetChannelConfig**.

- **Return :**

  NoError : OK
  ExceedBoardNumber: invalidate board number
  FindBoardError : cannot find the OME-PCI-180X board
  AdPollingTimeOut : hardware timeout error

- **Demo Program :**

  DEMO1.C

2.8.3 P180X_AdsPolling

- **Description :**

  This subroutine will perform multiple A/D conversions by polling. The **P180X_SetChannelConfig** subroutine can be used to change channel or configuration code. This function will refer to the current active PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  WORD P180X_AdsPolling(float fAdVal[], WORD wNum);

- **Parameter :**

  fAdVal[] : starting address of A/D data buffer, the data will be automatically converted to volts based on the setting of **P180X_SetChannelConfig**.

  wNum : number of A/D conversions to be performed.

- **Return :**

  NoError: OK

  ExceedBoardNumber: invalid board number

  FindBoardError: cannot find the OME-PCI-180X board

  AdPollingTimeOut: hardware timeout error

- **Demo Program :**

  DEMO1.C

2.8.4 P180X_AdsPacer

- **Description :**

  This subroutine will perform multiple A/D conversions by pacer trigger. The **P180X_SetChannelConfig** subroutine can be used to change channel or configuration code and the **P180X_AdsPacer** will refer to that condition in later operation. The hardware pacer will generate a periodic trigger signal to the A/D converter. The A/D data can be used to reconstruct the waveform of the analog input. Since the P180X_AdsPacer function uses the hardware pacer, operating system interrupts will not affect it. For this reason **the P180X_AdsPacer function should be used if a waveform must be precisely reconstructed.** This function will refer to the current active PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  WORD P180X_AdsPacer(float fAdVal[], WORD wNum, WORD wSample);

- **Parameter :**

  fAdVal[] : starting address of the A/D data buffer, the data will automatically be converted to volts based on the setting of **P180X_SetChannelConfig**.

  wNum : number of A/D conversions to be performed.

  wSample : A/D **sample rate = 8M/wSample.**

  for example: wSample=24 → sample rate=8M/24=330K

- **Return :**

  NoError : OK

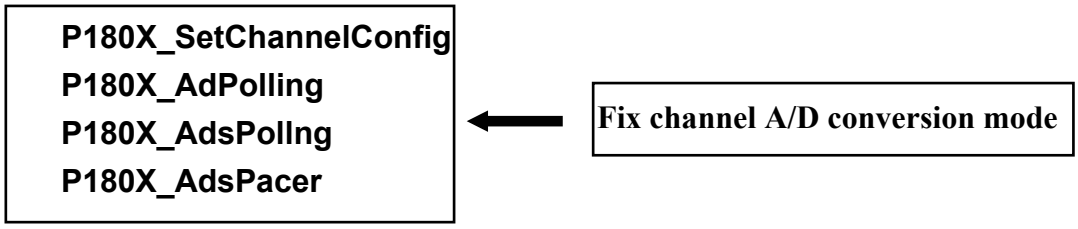
  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-180X board

  AdPollingTimeOut : hardware timeout error

- **Demo Program :**

  DEMO1.C



2.9 The MagicScan Functions

2.9.1 P180X_ClearScan

- **Description :**

  This subroutine will initialize the MagicScan controller. This function will refer to the current active PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  WORD P180X_ClearScan();

- **Parameter :**

  None

- **Return :**

  NoError : OK
  ExceedBoardNumber: invalid board number
  FindBoardError : cannot find the OME-PCI-180X board
  AdControllerError : MagicScan controller hardware handshake error

- **Demo Program :**

  DEMO11.C

2.9.2 P180X_StartScan

- **Description :**

  This subroutine will start the MagicScan operation. **The subroutine will return to the caller before the MagicScan operation is finished.** The **P180X_WaitMagicScanFinish(…)** function or the **P180X_ReadScanStatus(…)** function can be used to check the status of the MagicScan operation. This function will refer to the current active OME-PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  P180X_StartScan(WORD wSampleRateDiv, DWORD dwNum, SHORT nPriority);

- **Parameter :**

  wSampleRateDiv : A/D **sample rate = 8M/wSampleRate.**
  wSampleRate=24 → sample rate=8M/24=330K
  dwNum : Number of **MagicScan cycle** to perform
  nPriority : Used to adjust the priority of the thread.
  The valid values are as followings...

Value	ID (Description)
-2	THREAD_PRIORITY_LOWEST
-1	THREAD_PRIORITY_BELOW_NORMAL
0	THREAD_PRIORITY_NORMAL
1	THREAD_PRIORITY_ABOVE_NORMAL
2	THREAD_PRIORITY_HIGHEST
15	THREAD_PRIORITY_TIME_CRITICAL
others	THREAD_PRIORITY_NORMAL

- **Return :**

  NoError : OK
  ExceedBoardNumber: invalid board number
  FindBoardError : cannot find the OME-PCI-180X board
  AdControllerError : MagicScan controller hardware handshake error

- **Demo Program :**

  DEMO11.C

2.9.3 P180X_ReadScanStatus

- **Description :**

  This subroutine will read the status of the MagicScan operation. This function will refer to the current active OME-PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  void P180X_ReadScanStatus(WORD *wStatus, WORD *wLowAlarm,
  WORD *wHighAlarm);

- **Parameter :**

  wStatus : Store the MagicScan status

Status	Description
0x00	MagicScan initial condition (idle state)
0x01	MagicScan start to operation
0x02	MagicScan stage 1 controller timeout
0x04	MagicScan stage 2 controller timeout
0x08	MagicScan FIFO overflow
0x80	MagicScan function OK

  wLowAlarm : Store the MagicScan alarm status,
  32 bits corresponding to 32 channels,
  every bit: 0 indicates No Low-Alarm, 1 indicates Low-Alarm.
  For example: wLowAlarm=0 indicates all channels are Okay.
  wLowAlarm=3 indicates channel 0 and 1 are LowAlarm.

  wHighAlarm : Store the MagicScan alarm status
  32 bits corresponding to 32 channels,
  every bit: 0 indicates No High-Alarm, 1 indicates High-Alarm.
  For example: wHighAlarm=0 indicates all channels are Okay.
  wHighAlarm=5 indicates channel 0 and 2 are HighAlarm.

- **Return :**

  void

- **Demo Program :**

  DEMO11.C

2.9.4 P180X_AddToScan

- **Description :**

  This subroutine will add one channel to the **MagicScan circular queue**. This function will refer to the current active OME-PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

  void P180X_AddToScan(WORD wAdChannel, WORD wConfig, WORD wAverage,
  WORD wLowAlarm, WORD wHighAlarm, WORD wAlarmType);

- **Parameter :**

  wAdChannel : A/D channel number
  wConfig : the configuration code, Refer to "Section 2.1 Configuration Table"
  wAverage : the factor for the digital average filter
  wLowAlarm : 12 bit low alarm data
  wHighAlarm : 12 bit high alarm data
  wAlarmType : 0=no alarm, 1=high alarm, 2=low alarm, 3=in-alarm, 4=out-alarm

- **Return :**

  NoError : Ok
  ExceedBoardNumber: invalid board number
  FindBoardError : cannot find the OME-PCI-180X board
  AdChannelError : invalid AD channel
  AlarmTypeError : only 0/1/2/3/4 are valid
  AdControllerError : MagicScan controller hardware handshake error

- **Demo Program :**

  DEMO11.C

2.9.5 P180X_SaveScan

- **Description :**

  This subroutine will specify the starting address of A/D data buffer for MagicScan.

- **Syntax :**

  WORD P180X_SaveScan(WORD wAdChannel, WORD wBuf[]);

- **Parameter :**

  wAdChannel : Scan number in the scan queue.
  (Note: not the A/D channel number.)
  wBuf : starting address of A/D data buffer for channel specified in **wAdChannel**

- **Return :**

  NoError : Ok
  ExceedBoardNumber: invalid board number
  FindBoardError : cannot find the OME-PCI-180X board
  AdChannelError : invalid A/D channel

- **Demo Program :**

  DEMO11.C

- **Code Fragment**

```
WORD   wV0[100000];   // A/D ch:0 buffer
WORD   wV2[100000];   // A/D ch:2 buffer
 :
 :
wRetVal=P180X_ClearScan();
//****    For PCI-180XL
wRetVal += P180X_AddToScan(0,0,1,0,0,0); // CH:0 to scan
wRetVal += P180X_SaveScan(0,wV0);
wRetVal += P180X_AddToScan(2,0,1,0,0,0); // CH:2 to scan
wRetVal += P180X_SaveScan(1,wV2);   // Notice: 1 not 2
                                    // ^  Notice: This is a ordinal number in
                                    //    Scan Queue not a channel number.
wSampleRateDiv=80;            //    sample rate=8M/wSampleRateDiv
P180X_StartScan(wSampleRateDiv,DATALENGTH,nPriority);
```

2.9.6 P180X_WaitMagicScanFinish

- **Description :**

This subroutine will wait until the MagicScan operation is finished. This function will refer to the current active OME-PCI-180X board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax :**

void P180X_WaitMagicScanFinish(WORD *wStatus, WORD *wLowAlarm,
WORD *wHighAlarm);

- **Parameter :**

wStatus : Store the MagicScan status

Status	Description
0x00	MagicScan initial condition (idle state)
0x01	MagicScan operation started
0x02	MagicScan stage 1 controller timeout
0x04	MagicScan stage 2 controller timeout
0x08	MagicScan FIFO overflow
0x80	MagicScan function OK

wLowAlarm : Store the MagicScan alarm status,
32 bits corresponding to 32 channels,
every bit: 0 indicates No Low-Alarm, 1 indicates Low-Alarm.
For example: wLowAlarm=0 indicates all channels are Okay.
wLowAlarm=3 indicates channel 0 and 1 are LowAlarm.

wHighAlarm : Store the MagicScan alarm status
32 bits corresponding to 32 channels,
every bit: 0 indicates No High-Alarm, 1 indicates High-Alarm.
For example: wHighAlarm=0 indicates all channels are Okay.
wHighAlarm=5 indicates channel 0 and 2 are HighAlarm.

- **Return :**

Void

- **Demo Program :**

DEMO11.C

2.10 The Plug & Play Functions

2.10.1 P180X_DriverInit

Description : This function will detect all the OME-PCI-1800/1802 boards installed in the system. This function must be called once before the other functions are called.

- **Syntax :**

  WORD P180X_DriverInit(WORD *wTotalBoard);

- **Parameter :**

  *wTotalBoard : address of **wTotalBoard**
  wTotalBoard=1 → one OME-PCI-180X card in the system
  wTotalBoard=n → n OME-PCI-180X cards in the system

- **Return :**

  NoError : OK
  NoFoundBoard: can not detect any OME-PCI-180X
  FindBoardError: handshake check error
  DriverHandleError : the NAPPCI.VxD .open error for Windows 95
  the NAPWNT.SYS .open error for Windows NT
  DriverCallError : call NAPPCI.VxD return error
  call NAPWNT.SYS return error

- **Demo Program :**

  All DEMO programs.

2.10.2 P180X_DriverClose

- **Description :**

  Releases all system resources. This function must be called once before the program is terminated.

- **Syntax :**

  void P180X_DriverClose(void);

- **Parameter :** Void
- **Return :** Void
- **Demo Program :**

  All DEMO programs.

2.10.3 P180X_GetConfigAddressSpace

- **Description :**

  Returns the I/O address of the OME-PCI-1800/1802 board n. This function is for debug purposes only. It is not necessary to call this function.

- **Syntax :**

  WORD P180X_GetConfigAddressSpace(WORD wBoardNo, WORD *wAddrTimer, WORD *wAddrCtrl, WORD *wAddrDio, WORD *wAddrAdda);

- **Parameter :**

wBoardNo	: OME-PCI-1800/1802 board number
wAddrTimer, wAddrCtrl, wAddrDio, wAddrAdda	: refer to “OME-PCI-1800/1802 Hardware manual” chapter-3 for details.

- **Return :**

  NoError : OK
  FindBoardError: handshake check error
  ExceedBoardError: wBoardNo is invalid

- **Demo Program :**

  DEMO1.C

2.10.4 P180X_WhichBoardActive

- **Description:**

  Returns the board number of the active board.

- **Syntax:**

  WORD P180X_WhichBoardActive(void);

- **Parameter:** None
- **Return :**

  board number of the active board.

- **Demo Program:**
  DEMO1.C

2.10.5 P180X_ActiveBoard

- **Description:**

  This function will make active one of the OME-PCI-1800/1802 boards installed in the system. This function must be called once before the D/I/O, A/D, D/A functions are called.

- **Syntax:**

  WORD P180X_ActiveBoard(WORD wBoardNo);

- **Parameter:**

  wBoardNo : board number

- **Return :**

  NoError : OK

  ExceedBoardError: wBoardNo is invalid

- **Demo Program :**

  All DEMO programs.

The P180X_ActiveBoard(…) will affect all functions except the following:

1. P180X_FloatSub2
2. P180X_ShortSub2
3. P180X_GetDriverVersion
4. P180X_DriveInit
5. P180X_DriveClose
6. P180X_GetConfigAddressSpace
7. P180X_Card0_StartScan
8. P180X_Card0_ReadData
9. P180X_Card0_Stop
10. P180X_Card1_StartScan
11. P180X_Card1_ReadData
12. P180X_Card1_Stop

2.11 Multi-board Batch Capture
(Two boards operating simultaneously)

2.11.1 P180X_FunA_Start

- **Description :**

  This function will start the batch capture process for two boards operating simultaneously.

- **Syntax :**

```
WORD P180X_FunA_Start(WORD wClockDiv0, WORD wChannel0[],
    WORD wConfig0[], WORD *Buffer0, DWORD dwMaxCount0,
    WORD wClockDiv1, WORD wChannel1[],
    WORD wConfig1[], WORD *Buffer1, DWORD dwMaxCount1,
    Short  nPriority);
```

- **Parameter :**

wClockDiv0	: the A/D sample rate divisor for first board.
	the sample rate is 8M/wClockDiv0.
wChannel0[]	: (0=no scan, 1=scan) for each channel of the first board
wConfig0[]	: configuration code for each channel of the first board,
	Refer to "Section 2.1 Configuration Table"
*Buffer0	: buffer to store the A/D data of the first board
dwMaxCount0	: to specify the data length of the first board
wClockDiv1	: the A/D sample rate devisor for the second board.
	the sample rate is 8M/wClockDiv1.
WChannel1[]	: (0=no scan, 1=scan) for each channel of the second board
WConfig1[]	: configuration code for each channel of the second board,
	Refer to "Section 2.1 Configuration Table"
*Buffer1	: buffer to store the A/D data of the second board
dwMaxCount1	: sample count for the second board

nPriority : A/D thread priority. The value of nPriority ranges from:

Value	ID (Description)
-2	THREAD_PRIORITY_LOWEST
-1	THREAD_PRIORITY_BELOW_NORMAL
0	THREAD_PRIORITY_NORMAL
1	THREAD_PRIORITY_ABOVE_NORMAL
2	THREAD_PRIORITY_HIGHEST
15	THREAD_PRIORITY_TIME_CRITICAL
other	THREAD_PRIORITY_NORMAL

- **Return :**

NoError : OK

FindTwoBoardError : cannot find two OME-PCI-180X boards

- **Demo Program :**

DEMO20.C

2.11.2 P180X_FunA_ReadStatus

- **Description :**

  This function will read the status of the batch capture process.

- **Syntax :**

  WORD P180X_FunA_ReadStatus(void);

- **Parameter :**

  None

- **Return :**

  0: data is ready

  1: data not ready

- **Demo Program :**

  DEMO20.C

2.11.3 P180X_FunA_Stop

- **Description:**
  This function will stop the batch capture function.

- **Syntax:**
  word P180X_FunA_Stop(void);

- **Parameter:**
  None

- **Return :**
  NoError : OK
  StopError : Stop Error

- **Demo Program :**
  DEMO20.C

2.11.4 P180X_FunA_Get

- **Description:**
  This function will retrieve the number A/D samples acquired.

- **Syntax:**
  word P180X_FunA_Get(DWORD *P0, DWORD *P1);

- **Parameter:**
  *P0: [output] the number of A/D samples that have been acquired for the first board.
  *P1: [output] the number of A/D samples that have been acquired for the second board.

- **Return :**
  NoError : OK

- **Demo Program :**
  DEMO20.C

2.12 The Single Board Batch Capture

2.12.1 P180X_FunB_Start

- **Description :**

  This function will start the batch capture process.

- **Syntax :**

  WORD P180X_FunB_Start(WORD wClockDiv0, WORD wChannel0[], WORD wConfig0[], WORD *Buffer0, DWORD dwMaxCount0, SHORT nPriority);

- **Parameter :**

  wClockDiv0 : the A/D sample rate divisor for this board.
  the sample rate is 8M/wClockDiv0.
  wChannel0[] : (0=no scan, 1=scan) for each channel of this board
  wConfig0[] : configuration code for each channel of this board,
  Refer to "Section 2.1 Configuration Table"
  *Buffer0 : buffer to store the A/D data of this board
  dwMaxCount0 : A/D sample count
  nPriority : Thread priority. The value of nPriority ranges from:

Value	ID (Description)
-2	THREAD_PRIORITY_LOWEST
-1	THREAD_PRIORITY_BELOW_NORMAL
0	THREAD_PRIORITY_NORMAL
1	THREAD_PRIORITY_ABOVE_NORMAL
2	THREAD_PRIORITY_HIGHEST
15	THREAD_PRIORITY_TIME_CRITICAL
others	THREAD_PRIORITY_NORMAL

- **Return :**

  NoError : OK
  FindBoardError : cannot find the OME-PCI-180X board
  AdControllerError : MagicScan controller hardware handshake error

- **Demo Program :**

  DEMO21.C

2.12.2 P180X_FunB_ReadStatus

- **Description :**
  This function provides the status of the batch capture.

- **Syntax :**
  WORD P180X_FunB_ReadStatus(void);

- **Parameter :**
  None

- **Return :**
  0: data is ready
  1: data not ready

- **Demo Program :**
  DEMO21.C

2.12.3 P180X_FunB_Stop

- **Description:**
  This subroutine will stop the batch capture function.

- **Syntax:**
  word P180X_FunB_Stop(void);

- **Parameter:**
  None

- **Return :**
  NoError : OK
  StopError : Stop Error

- **Demo Program :**
  DEMO21.C

2.12.4 P180X_FunB_Get

- **Description:**
  This function will retrieve the number of A/D samples that have been acquired.

- **Syntax:**
  word P180X_FunB_Get(DWORD *P0);

- **Parameter:**
  *P0: [output] how many A/D data has been acquired for first board.

- **Return :**
  NoError: OK

- **Demo Program :**
  DEMO21.C

2.13 The Continuous Capture Functions

2.13.1 P180X_Card0_StartScan

- **Description:**

  This function will start the continuous capture function for card 0. The continuous capture functions are best suited for low speed, long duration data collection. Although computer dependent, sample rates should generally be kept under 40kHz. Refer to the OME-PCI-1800 Hardware User Manual, for additional details on this function.

- **Syntax:**

  WORD P180X_Card0_StartScan(WORD wSampleRate, WORD wChannelStatus[], WORD wChanelConfig[], WORD wCount);

- **Parameter:**

  wSampleRate : A/D **sample rate = 8M/wSampleRate.**
  wSampleRate=240 → sample rate=8M/240=33KHz
  wChannelStatus[] : (0=no scan, 1=scan) for each channel
  wChannelConfig[] : configuration code for each channel,
  Refer to "Section 2.1 Configuration Table"
  wCount : number of A/D data for each scan channel

- **Return :**

  NoError : OK
  FindBoardError : cannot find the OME-PCI-180X board
  AdControllerError : MagicScan controller hardware handshake error

- **Demo Program :**

  DEMO13.C

2.13.2 P180X_Card0_ReadStatus

- **Description:**
  This function will read the data collected by the continuous capture function.

- **Syntax:**
  P180X_Card0_ReadStatus(WORD wBuf[], WORD wBuf2[], DWORD *dwP1,
  DWORD *dwP2, WORD *wStatus);

- **Parameter:**

wBuf[]	: in scan sequence order(012…N012…N……012…N)
wBuf2[]	: in channel sequence order(00000…..11111……22222….NNNNN….)
dwP1	: reserved
dwP2	: reserved
wStatus	: 1=thread start, 2=TimeOut, 8=FIFO overflow, 0x80=thread finish

- **Return :**
  0: data is ready
  1: data not ready

- **Demo Program :**
  DEMO13.C

2.13.3 P180X_Card0_Stop

- **Description:**
  This function will stop the continuous capture function.

- **Syntax:**
  void P180X_Card0_Stop(void);

- **Parameter:** None
- **Return :** Void
- **Demo Program :**
  DEMO13.C

2.13.4 P180X_Card1_StartScan

- **Description:**

  This function will start the continuous capture function for card 1. The continuous capture functions are best suited for low speed, long duration data collection. Although computer dependent, sample rates should generally be kept under 40kHz. Refer to the OME-PCI-1800 Hardware User Manual, for additional details on this function.

- **Syntax:**

  WORD P180X_Card1_StartScan(WORD wSampleRate, WORD wChannelStatus[],
  WORD wChanelConfig[], WORD wCount);

- **Parameter:**

wSampleRate	: A/D **sample rate = 8M/wSampleRate.**
	wSampleRate=240 → sample rate=8M/240=33KHz
wChannelStatus[]	: (0=no scan, 1=scan) for each channel
wChannelConfig[]	: configuration code for each channel,
	Refer to "Section 2.1 Configuration Table"
wCount	: number of A/D data for each scan channel

- **Return :**

  NoError : OK
  FindBoardError : cannot find theOME-PCI-180X board
  AdControllerError : MagicScan controller hardware handshake error

- **Demo Program :**

  DEMO14.C

2.13.5 P180X_Card1_ReadStatus

- **Description:**
  This function will read the data collected by the continuous capture function.

- **Syntax:**
  P180X_Card1_ReadStatus(WORD wBuf[], WORD wBuf2[], DWORD *dwP1,
  DWORD *dwP2, WORD *wStatus);

- **Parameter:**
  wBuf[]: in scan sequence order(012…N012…N……012…N)
  wBuf2[]: in channel sequence order(00000…..11111……22222….NNNNN….)
  dwP1: reserved
  dwP2: reserved
  wStatus: 1=thread start, 2=TimeOut, 8=FIFO overflow, 0x80=thread finish

- **Return :**
  0: data is ready
  1: data not ready

- **Demo Program :**
  DEMO14.C

2.13.6 P180X_Card1_Stop

- **Description:**
  This function will stop the continuous capture process.

- **Syntax:**
  void P180X_Card1_Stop(void);

- **Parameter:** None
- **Return :** Void
- **Demo Program :**
  DEMO14.C

2.14 Other Functions

2.14.1 P180X_DelayUs

- **Description:**

  This is a **machine independent timer.** This function can be used to generate the **settling time delay** or used as a **general purpose machine independent timer**. This function will refer to the current active OME-PCI-1800/1802 board. Use P180X_ActiveBoard(….) to select the active board.

- **Syntax:**

  word P180X_DelayUs(WORD wDelayUs);

- **Parameter:**

  wDelayUs : number of μs to delay, 8191 Max
  wDelayUs=1 → delay 1 μs
  wDelayUs=1000 → delay 1000 us = 1 ms
  wDelayUs=8191 → delay 8191 μs = 8.191 ms (maximum delay)
  wDelayUs=8192 → invalid delay (will return error)

- **Return :**

  NoError : OK
  ExceedBoardNumber: invalid board number
  FindBoardError : cannot find theOME-PCI-180X board
  InvalidDelay : **dwDelayUs** > 8191

- **Demo Program :** DEMO1.C

- **Long Time Delay :**

```
WORD DelayMs(WORD wDelayMs) // maximum delay=4294967.295 sec
{
    WORD wDelay,wRetVal
    wRetVal=0;
    for    (wDelay=0; wDelay<wDelayMs; wDelay++)
        wRetVal+=P180X_DelayUs(1000);
    return(wRetVal);
}
```

3. Demo Programs

The following demonstration programs are provided on the included CD:

- demo1: one board, D/I/O test, D/A test, A/D polling & pacer trigger test, general test
- demo2: two board, same as demo1
- demo3: one board, all 32 channels of A/D by software trigger(by polling)
- demo4: two boards, same as demo3
- demo5: one board, M_function_1 demo
- demo6: two boards, same as demo5
- demo7: one board, M_function_2 demo
- demo8: two boards, same as demo7
- demo9: one board, M_function_3 demo
- demo10: two boards, same as demo9
- demo11: one board, MagicScan demo
- demo12: two boards, same as demo11
- demo13: one board, continuous capture demo
- demo14: two boards, continuous capture demo (Windows 95/NT only)
- demo15: all installed boards, D/I/O test for board number identification
- demo16: one board, performance evaluation demo
- demo17: one board, MagicScan demo, scan sequence: 4→3→5
- demo18: one boards, MagicScan demo, scan 32 channel, show channel 0/1/15/16/17
- demo19: one board, A/D calibration.
- demo20: two boards, P180X_FUNA, continuous capture demo
- demo21: single board, P180X_FUNB, continuous capture demo
- demo23: single board, post-trigger demo
- demo24: single board, pre-trigger demo
- demo25: single board, middle-trigger demo

WARRANTY/DISCLAIMER

OMEGA ENGINEERING, INC. warrants this unit to be free of defects in materials and workmanship for a period of 13 months from date of purchase. OMEGA's WARRANTY adds an additional one (1) month grace period to the normal one (1) year product warranty to cover handling and shipping time. This ensures that OMEGA's customers receive maximum coverage on each product.

If the unit malfunctions, it must be returned to the factory for evaluation. OMEGA's Customer Service Department will issue an Authorized Return (AR) number immediately upon phone or written request. Upon examination by OMEGA, if the unit is found to be defective, it will be repaired or replaced at no charge. OMEGA's WARRANTY does not apply to defects resulting from any action of the purchaser, including but not limited to mishandling, improper interfacing, operation outside of design limits, improper repair, or unauthorized modification. This WARRANTY is VOID if the unit shows evidence of having been tampered with or shows evidence of having been damaged as a result of excessive corrosion; or current, heat, moisture or vibration; improper specification; misapplication; misuse or other operating conditions outside of OMEGA's control. Components which wear are not warranted, including but not limited to contact points, fuses, and triacs.

OMEGA is pleased to offer suggestions on the use of its various products. However, OMEGA neither assumes responsibility for any omissions or errors nor assumes liability for any damages that result from the use of its products in accordance with information provided by OMEGA, either verbal or written. OMEGA warrants only that the parts manufactured by it will be as specified and free of defects. OMEGA MAKES NO OTHER WARRANTIES OR REPRESENTATIONS OF ANY KIND WHATSOEVER, EXPRESS OR IMPLIED, EXCEPT THAT OF TITLE, AND ALL IMPLIED WARRANTIES INCLUDING ANY WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE HEREBY DISCLAIMED. LIMITATION OF LIABILITY: The remedies of purchaser set forth herein are exclusive, and the total liability of OMEGA with respect to this order, whether based on contract, warranty, negligence, indemnification, strict liability or otherwise, shall not exceed the purchase price of the component upon which liability is based. In no event shall OMEGA be liable for consequential, incidental or special damages.

CONDITIONS: Equipment sold by OMEGA is not intended to be used, nor shall it be used: (1) as a "Basic Component" under 10 CFR 21 (NRC), used in or with any nuclear installation or activity; or (2) in medical applications or used on humans. Should any Product(s) be used in or with any nuclear installation or activity, medical application, used on humans, or misused in any way, OMEGA assumes no responsibility as set forth in our basic WARRANTY/DISCLAIMER language, and, additionally, purchaser will indemnify OMEGA and hold OMEGA harmless from any liability or damage whatsoever arising out of the use of the Product(s) in such a manner.

RETURN REQUESTS/INQUIRIES

Direct all warranty and repair requests/inquiries to the OMEGA Customer Service Department. BEFORE RETURNING ANY PRODUCT(S) TO OMEGA, PURCHASER MUST OBTAIN AN AUTHORIZED RETURN (AR) NUMBER FROM OMEGA'S CUSTOMER SERVICE DEPARTMENT (IN ORDER TO AVOID PROCESSING DELAYS). The assigned AR number should then be marked on the outside of the return package and on any correspondence.

The purchaser is responsible for shipping charges, freight, insurance and proper packaging to prevent breakage in transit.

FOR WARRANTY RETURNS, please have the following information available BEFORE contacting OMEGA:

1. Purchase Order number under which the product was PURCHASED,
2. Model and serial number of the product under warranty, and
3. Repair instructions and/or specific problems relative to the product.

FOR NON-WARRANTY REPAIRS, consult OMEGA for current repair charges. Have the following information available BEFORE contacting OMEGA:

1. Purchase Order number to cover the COST of the repair,
2. Model and serial number of the product, and
3. Repair instructions and/or specific problems relative to the product.

OMEGA's policy is to make running changes, not model changes, whenever an improvement is possible. This affords our customers the latest in technology and engineering.

OMEGA is a registered trademark of OMEGA ENGINEERING, INC.

Where Do I Find Everything I Need for Process Measurement and Control? OMEGA...Of Course!

Shop online at www.omega.com

TEMPERATURE

- Thermocouple, RTD & Thermistor Probes, Connectors, Panels & Assemblies
- Wire: Thermocouple, RTD & Thermistor
- Calibrators & Ice Point References
- Recorders, Controllers & Process Monitors
- Infrared Pyrometers

PRESSURE, STRAIN AND FORCE

- Transducers & Strain Gages
- Load Cells & Pressure Gages
- Displacement Transducers
- Instrumentation & Accessories

FLOW/LEVEL

- Rotameters, Gas Mass Flowmeters & Flow Computers
- Air Velocity Indicators
- Turbine/Paddlewheel Systems
- Totalizers & Batch Controllers

pH/CONDUCTIVITY

- pH Electrodes, Testers & Accessories
- Benchtop/Laboratory Meters
- Controllers, Calibrators, Simulators & Pumps
- Industrial pH & Conductivity Equipment

DATA ACQUISITION

- Data Acquisition & Engineering Software
- Communications-Based Acquisition Systems
- Plug-in Cards for Apple, IBM & Compatibles
- Datalogging Systems
- Recorders, Printers & Plotters

HEATERS

- Heating Cable
- Cartridge & Strip Heaters
- Immersion & Band Heaters
- Flexible Heaters
- Laboratory Heaters

ENVIRONMENTAL MONITORING AND CONTROL

- Metering & Control Instrumentation
- Refractometers
- Pumps & Tubing
- Air, Soil & Water Monitors
- Industrial Water & Wastewater Treatment
- pH, Conductivity & Dissolved Oxygen Instruments

M3932/0203