User's Guide

OME-PCI-1602

PCI Data Acquisition Board
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

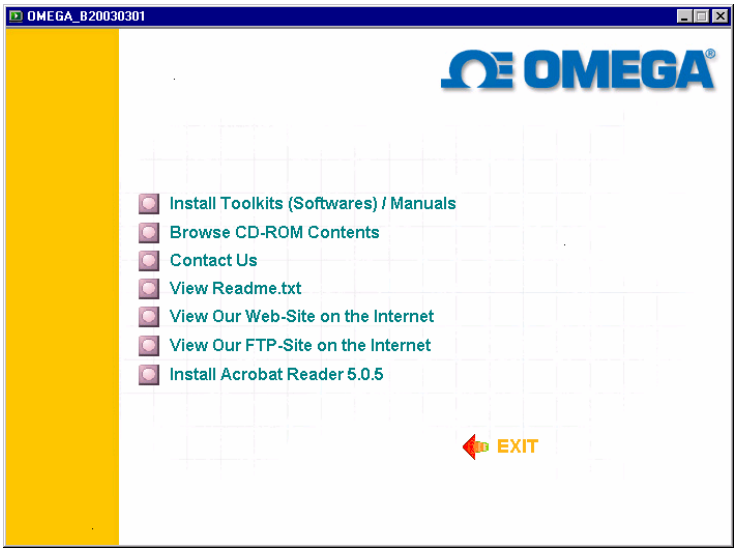
The driver is a collection of DLLs for Windows 95/98/NT/2000/XP applications. These DLLs are standard Win32 DLLs (32 bits) and can be called from Visual C/C++, BC++, Visual BASIC, Delphi, BCB and LabVIEW.

These DLLs can perform a variety of data acquisition operations including:

- Get software version
- Initialization
- Digital Input/Output
- A/D conversion
- D/A conversion

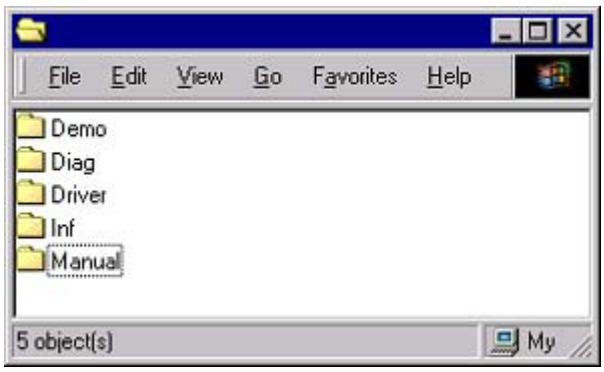
1.1 Software Installation

Insert the CD ROM included with your OME-PCI-1602 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-1602 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 their source codes. 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 NMAKE.EXE to compiling and link 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 Reference:

Please refer to the following user manuals for more information.

CallDll.pdf:

Calling the DLL with VB5, VC5, Delphi3, and BCB3.

ResCheck.pdf:

How to check the system resources.(IRQ, IO Port, and DMA)

PnPInstall.pdf:

Installing the information file (.inf) under Windows 95/98/2000/XP with Plug & Play.

SoftInst.pdf:

Installing the software driver (package).

2. Declaration files

2.1 The P1602.h

```
#define EXPORTS extern "C" __declspec (dllimport)
//#define EXPORTS

// return code
#define NoError                 0
#define DriverHandleError       1
#define DriverCallError         2
#define AdControllerError       3
#define M_FunExecError          4
#define ConfigCodeError         5
#define FrequencyComputeError   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 P1602_FloatSub2(float fA, float fB);
EXPORTS short CALLBACK P1602_ShortSub2(short nA, short nB);
EXPORTS WORD   CALLBACK P1602_GetDllVersion(void);

EXPORTS WORD   CALLBACK P1602_DriverInit(WORD *wTotalBoards);
EXPORTS void   CALLBACK P1602_DriverClose(void);
EXPORTS WORD   CALLBACK P1602_GetDriverVersion(WORD *wVxdVersion);

EXPORTS WORD   CALLBACK P1602_GetConfigAddressSpace(WORD wBoardNo,   WORD *wAddrTimer,WORD *wAddrCtrl, WORD *wAddrDio, WORD *wAddrAdda);

EXPORTS WORD   CALLBACK P1602_ActiveBoard(WORD wBoardNo);
EXPORTS WORD   CALLBACK P1602_WhichBoardActive(void);

EXPORTS WORD   CALLBACK P1602_M_FUN_1(WORD wDaFrequency, WORD wDaWave, float fDaAmplitude, WORD wAdClock, WORD wAdNumber, WORD wAdConfig, float fAdBuf[],   float fLowAlarm, float fHighAlarm);

EXPORTS WORD   CALLBACK P1602_M_FUN_2(WORD wDaNumber, WORD wDaWave, WORD wDaBuf[], WORD wAdClock, WORD wAdNumber, WORD wAdConfig, WORD wAdBuf[]);

EXPORTS WORD   CALLBACK P1602_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 P1602_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 P1602_Di(WORD *wDi);
EXPORTS WORD CALLBACK P1602_Do(WORD wDo);

EXPORTS WORD CALLBACK P1602_Da(WORD wDaChannel, WORD wDaVal);
EXPORTS WORD CALLBACK P1602_SetChannelConfig(WORD wAdChannel,

WORD wConfig);

EXPORTS WORD CALLBACK P1602_AdPolling(float *fAdVal);
EXPORTS WORD CALLBACK P1602_AdsPolling(float fAdVal[], WORD wNum);
EXPORTS WORD CALLBACK P1602_AdsPacer(float fAdVal[], WORD wNum,
WORD wSample);

EXPORTS WORD CALLBACK P1602_ClearScan(void);
EXPORTS WORD CALLBACK P1602_StartScan(WORD wSampleRateDiv,
DWORD dwNum, SHORT nPriority);
EXPORTS void CALLBACK P1602_ReadScanStatus(WORD *wStatus,
DWORD *dwLowAlarm, DWORD *dwHighAlarm);
EXPORTS WORD CALLBACK P1602_AddToScan(WORD wAdChannel, WORD
wConfig, WORD wAverage, WORD wLowAlarm, WORD wHighAlarm, WORD
wAlarmType);
EXPORTS WORD CALLBACK P1602_SaveScan(WORD wAdChannel, WORD
wBuf[]);
EXPORTS void CALLBACK P1602_WaitMagicScanFinish(WORD *wStatus,
DWORD *dwLowAlarm, DWORD *dwHighAlarm);
EXPORTS WORD CALLBACK P1602_StopMagicScan();

EXPORTS WORD CALLBACK P1602_DelayUs(WORD wDelayUs);

EXPORTS WORD CALLBACK P1602_Card0_StartScan(WORD wSampleRate,
WORD wChannelStatus[], WORD wChannelConfig[],WORD wCount);
EXPORTS WORD CALLBACK P1602_Card0_ReadStatus(WORD wBuf[], WORD
wBuf2[], DWORD *dwP1, DWORD *dwP2, WORD *wStatus);
EXPORTS void CALLBACK P1602_Card0_Stop(void);

EXPORTS WORD CALLBACK P1602_Card1_StartScan(WORD wSampleRate,
WORD wChannelStatus[], WORD wChannelConfig[],WORD wCount);
EXPORTS WORD CALLBACK P1602_Card1_ReadStatus(WORD wBuf[], WORD
wBuf2[], DWORD *dwP1, DWORD *dwP2, WORD *wStatus);
EXPORTS void CALLBACK P1602_Card1_Stop(void);

EXPORTS WORD CALLBACK P1602_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 P1602_FunA_ReadStatus(void);
EXPORTS WORD CALLBACK P1602_FunA_Stop(void);
EXPORTS WORD CALLBACK P1602_FunA_Get(DWORD *P0, DWORD *P1);

EXPORTS WORD CALLBACK P1602_FunB_Start(WORD wClock0Div, WORD wChannel0[], WORD wConfig0[], WORD *Buffer0, DWORD dwMaxCount0, SHORT nPriority);
EXPORTS WORD CALLBACK P1602_FunB_ReadStatus(void);
EXPORTS WORD CALLBACK P1602_FunB_Stop(void);
EXPORTS WORD CALLBACK P1602_FunB_Get(DWORD *P0);

EXPORTS WORD CALLBACK P1602_MemoryStatus(DWORD *dwTotalPhys, DWORD *dwAvailPhys, DWORD *dwTotalPageFile, DWORD *dwAvailPageFile, DWORD *dwTotalVirtual, DWORD *dwAvailVirtual);
EXPORTS WORD CALLBACK P1602_AllocateMemory(HGLOBAL *hMem, WORD *Buffer, DWORD dwSize);
EXPORTS WORD CALLBACK P1602_FreeMemory(HGLOBAL hMem);
EXPORTS WORD CALLBACK P1602_StartScanPostTrg(WORD wSampleRateDiv, DWORD dwNum, SHORT nPriority);
EXPORTS WORD CALLBACK P1602_StartScanPreTrg(WORD wSampleRateDiv, DWORD dwNum, SHORT nPriority);
EXPORTS WORD CALLBACK P1602_StartScanMiddleTrg(WORD wSampleRateDiv, DWORD dwN1, DWORD dwN2, SHORT nPriority);
EXPORTS WORD CALLBACK P1602_StartScanPreTrgVerC(WORD wSampleRateDiv, DWORD dwNum, SHORT nPriority);
EXPORTS WORD CALLBACK P1602_StartScanMiddleTrgVerC(WORD wSampleRateDiv, DWORD dwN1, DWORD dwN2, SHORT nPriority);

2.2 The P1602.BAS

```
Attribute VB_Name = "P1602"
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 FrequencyComputeError = 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 P1602_DriverInit Lib "P1602.dll" (wTotalBoards As Integer) As
Integer
Declare Sub P1602_DriverClose Lib "P1602.dll" ()

Declare Function P1602_GetDriverVersion Lib "P1602.dll" (wVxdVersion As
Integer) As Integer

Declare Function P1602_GetConfigAddressSpace Lib "P1602.dll" (ByVal wBoardNo
As Integer, wAddrTimer As Integer, wAddrCtrl As Integer, wAddrDio As Integer,
wAddrAdda As Integer) As Integer

Declare Function P1602_ActiveBoard Lib "P1602.dll" (ByVal wBoardNo As Integer)
As Integer
Declare Function P1602_WhichBoardActive Lib "P1602.dll" () As Integer

Declare Function P1602_M_FUN_1 Lib "P1602.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 P1602_M_FUN_2 Lib "P1602.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 P1602_M_FUN_3 Lib "P1602.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 P1602_M_FUN_4 Lib "P1602.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 P1602_Di Lib "P1602.dll" (wDi As Integer) As Integer
Declare Function P1602_Do Lib "P1602.dll" (ByVal wDo As Integer) As Integer

Declare Function P1602_Da Lib "P1602.dll" (ByVal wDaChannel As Integer, ByVal
wDaVal As Integer)  As Integer
Declare Function P1602_SetChannelConfig Lib "P1602.dll" (ByVal wAdChannel As
Integer, ByVal wConfig  As Integer) As Integer

Declare Function P1602_AdPolling Lib "P1602.dll" (fAdVal As Single) As Integer
Declare Function P1602_AdsPolling Lib "P1602.dll" (fAdVal As Single, ByVal
```

```
wNum As Integer) As Integer
Declare Function P1602_AdsPacer Lib "P1602.dll" (fAdVal As Single, ByVal wNum
As Integer,  ByVal wSample As Integer) As Integer

Declare Function P1602_ClearScan Lib "P1602.dll" () As Integer
Declare Function P1602_StartScan Lib "P1602.dll" (ByVal wSampleRate As Integer,
ByVal dwNum As Long, ByVal nPriority As Integer) As Integer
Declare Sub P1602_ReadScanStatus Lib "P1602.dll" (wStatus As Integer,
dwLowAlarm As Long,  dwHighAlarm As Long)
Declare Function P1602_AddToScan Lib "P1602.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 P1602_SaveScan Lib "P1602.dll" (ByVal wOridinalOrder As
Integer,  wBuf As Integer)  As Integer
Declare Sub P1602_WaitMagicScanFinish Lib "P1602.dll" (wStatus
As Integer,  wLowAlarm As Integer, _ wHighAlarm As Integer)
Declare Function P1602_StopMagicScan Lib "P1602.dll" () As Integer

Declare Function P1602_DelayUs Lib "P1602.dll" (ByVal wDelayUs As Integer) As
Integer

'------------------------ FunB series ----------------------------
Declare Function P1602_FunB_Start Lib "P1602.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 P1602_FunB_ReadStatus Lib "P1602.dll" () As Integer
Declare Function P1602_FunB_Stop Lib "P1602.dll" () As Integer
Declare Function P1602_FunB_Get Lib "P1602.dll" (P0 As Long) As Integer

Declare Function P1602_Card0_StartScan Lib "P1602.dll" (ByVal wSampleRate As
Integer,  wChannelStatus As Integer, wChannelConfig As Integer, ByVal wCount As
Integer) As Integer
Declare Function P1602_Card0_ReadStatus Lib "P1602.dll" (wBuf As Integer, wBuf2
As Integer, dwP1 As Long, dwP2 As Long, wStatus As Integer) As Integer
Declare Sub P1602_Card0_Stop Lib "P1602.dll" ()

Declare Function P1602_Card1_StartScan Lib "P1602.dll" (ByVal wSampleRate As
Integer,  wChannelStatus As Integer, wChannelConfig As Integer,  ByVal wCount As
Integer) As Integer
Declare Function P1602_Card1_ReadStatus Lib "P1602.dll" (wBuf As Integer, wBuf2
As Integer,  dwP1 As Long, dwP2 As Long, wStatus As Integer) As Integer
Declare Sub P1602_Card1_Stop Lib "P1602.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
```

2.3 The P1602.PAS

```
unit P1602;

interface

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

const
// return code
 NoError                 =  0;
 DriverHandleError       =  1;
 DriverCallError         =  2;
 AdControllerError       =  3;
 M_FunExecError          =  4;
 ConfigCodeError         =  5;
 FrequencyComputeError =  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  P1602_FloatSub2(fA:Single; fB:Single):Single ; stdCall;
function  P1602_ShortSub2(nA:SmallInt; nB:SmallInt):SmallInt ; stdCall;
function  P1602_GetDllVersion:WORD  ; stdCall;

// Function of Driver
function  P1602_DriverInit(Var wTotalBoards:Word):WORD ; stdCall;
procedure P1602_DriverClose; stdCall;
function  P1602_GetDriverVersion(var wDriverVersion:Word):WORD ; stdCall;

function  P1602_GetConfigAddressSpace(wBoardNo:Word;var wAddrTimer:Word;
            var wAddrCtrl:Word; var wAddrDio:Word;
            var wAddrAdda:Word):WORD ; stdCall;
function  P1602_ActiveBoard(wBoardNo:Word):WORD ; stdCall;
function  P1602_WhichBoardActive:WORD ; stdCall;

// Function of M_Fun series
function  P1602_M_FUN_1(wDaFrequency:WORD; wDaWave:WORD;
fDaAmplitude:Single;  wAdClock:WORD; wAdNumber:WORD; wAdConfig:WORD;
 fAdBuf:PSingle;     fLowAlarm:Single;  fHighAlarm:Single):WORD ; stdCall;

function  P1602_M_FUN_2(wDaNumber:WORD;  wDaWave:WORD;
wDaBuf:PWord; wAdClock:WORD; wAdNumber:WORD; wAdConfig:WORD;
  wAdBuf:PWord):WORD ; stdCall;

function  P1602_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  P1602_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  P1602_Do(wOutData:Word):Word; stdCall;
function  P1602_Di(var wDiData:Word):WORD ; stdCall;

// Function of AD/DA
function  P1602_Da(wDaChannel:Word; wDaVal:Word):WORD ; stdCall;
function  P1602_SetChannelConfig(wAdChannel:Word; wConfig:Word):WORD ;
stdCall;
function  P1602_AdPolling(var fAdVal:Single):WORD ; stdCall;
function  P1602_AdsPolling(fAdVal:PSingle; wNum:Word):WORD ; stdCall;
function  P1602_AdsPacer(fAdVal:PSingle; wNum:Word;
                wSamplingDiv:Word ):WORD ; stdCall;

//*******************
function  P1602_ClearScan:WORD ; stdCall;
function  P1602_StartScan(wSampleRateDiv:WORD;   dwNum:LongInt;
nPriority:SmallInt):WORD ; stdCall;
procedure P1602_ReadScanStatus(var wStatus:WORD; var dwLowAlarm:LongInt;
                    var dwHighAlarm:LongInt); stdCall;
function  P1602_AddToScan(wAdChannel:WORD; wConfig:WORD;
wAverage:WORD; wLowAlarm:WORD;  wHighAlarm:WORD;
wAlarmType:WORD):WORD ; stdCall;
function  P1602_SaveScan(wAdChannel:WORD; wBuf:PWord):WORD ; stdCall;
procedure P1602_WaitMagicScanFinish(var wStatus:WORD; var
dwLowAlarm:LongInt;  var dwHighAlarm:LongInt); stdCall;
function  P1602_StopMagicScan:WORD ; stdCall;

//*******************
function  P1602_DelayUs(wDelayUs:WORD):WORD ; stdCall;

//*******************
//function  P1602_Card0_StartScan( wSampleRate:WORD; wChannelStatus:PWORD;
//                    wChannelConfig:PWORD; wCount:WORD):WORD ; stdCall;
function  P1602_Card0_StartScan( wSampleRate:WORD; wChannelStatus:PWORD;
                    wChannelConfig:PWORD; wCount:WORD):WORD ; stdCall;
function  P1602_Card0_ReadStatus(wBuf:PWORD; wBuf2:PWORD;
                    var dwP1:LongInt; var dwP2:LongInt;
                    var wStatus:WORD):WORD ; stdCall;
procedure P1602_Card0_Stop; stdCall;
```

```
function  P1602_Card1_StartScan(wSampleRate:WORD; wChannelStatus:PWORD;
                    wChannelConfig:PWORD; wCount:WORD):WORD ; stdCall;
function  P1602_Card1_ReadStatus(wBuf:PWORD; wBuf2:PWORD; var
dwP1:LongInt;  var dwP2:LongInt; var wStatus:WORD):WORD ; stdCall;
procedure P1602_Card1_Stop; stdCall;


//*******************
function  P1602_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  P1602_FunA_ReadStatus:WORD ; stdCall;
function  P1602_FunA_Stop:WORD ; stdCall;
function  P1602_FunA_Get(var P0:LongInt; var P1:LongInt):WORD ; stdCall;


//*******************
function  P1602_FunB_Start( wClock0Div:WORD; wChannel0:PWord;
wConfig0:PWord;   Buffer0:PWord; dwMaxCount0:LongInt;
nPriority:SmallInt):WORD ; stdCall;
function  P1602_FunB_ReadStatus:WORD ; stdCall;
function  P1602_FunB_Stop:WORD       ; stdCall;
function  P1602_FunB_Get(var P0:LongInt):WORD ; stdCall;


//*****************************************************************/
/*****************************************************************
implementation

function  P1602_FloatSub2;   external 'P1602.DLL' name 'P1602_FloatSub2';
function  P1602_ShortSub2;   external 'P1602.DLL' name 'P1602_ShortSub2';
function  P1602_GetDllVersion; external 'P1602.DLL' name 'P1602_GetDllVersion';
function  P1602_GetDriverVersion; external 'P1602.DLL' name
'P1602_GetDriverVersion';
```

```
function  P1602_DriverInit;  external 'P1602.DLL' name 'P1602_DriverInit';
procedure P1602_DriverClose; external 'P1602.DLL' name 'P1602_DriverClose';
function  P1602_GetConfigAddressSpace;
                    external 'P1602.DLL' name 'P1602_GetConfigAddressSpace';
function  P1602_ActiveBoard; external 'P1602.DLL' name 'P1602_ActiveBoard';
function  P1602_WhichBoardActive;
                    external 'P1602.DLL' name 'P1602_WhichBoardActive';

// Function of M_Fun series
function  P1602_M_FUN_1;    external 'P1602.DLL' name 'P1602_M_FUN_1';
function  P1602_M_FUN_2;    external 'P1602.DLL' name 'P1602_M_FUN_2';
function  P1602_M_FUN_3;    external 'P1602.DLL' name 'P1602_M_FUN_3';
function  P1602_M_FUN_4;    external 'P1602.DLL' name 'P1602_M_FUN_4';

function  P1602_Do;         external 'P1602.DLL' name 'P1602_Do';
function  P1602_Di;         external 'P1602.DLL' name 'P1602_Di';

function  P1602_Da;         external 'P1602.DLL' name 'P1602_Da';
function  P1602_SetChannelConfig;
                    external 'P1602.DLL' name 'P1602_SetChannelConfig';
function  P1602_AdPolling;  external 'P1602.DLL' name 'P1602_AdPolling';
function  P1602_AdsPolling; external 'P1602.DLL' name 'P1602_AdsPolling';

function  P1602_AdsPacer;   external 'P1602.DLL' name 'P1602_AdsPacer';

//*********************
function  P1602_ClearScan;        external 'P1602.DLL' name 'P1602_ClearScan';
function  P1602_StartScan;        external 'P1602.DLL' name 'P1602_StartScan';
procedure P1602_ReadScanStatus;   external 'P1602.DLL' name
'P1602_ReadScanStatus';
function  P1602_AddToScan;      external 'P1602.DLL' name 'P1602_AddToScan';
function  P1602_SaveScan;         external 'P1602.DLL' name 'P1602_SaveScan';
procedure P1602_WaitMagicScanFinish; external 'P1602.DLL' name
'P1602_WaitMagicScanFinish';
function  P1602_StopMagicScan; external 'P1602.DLL' name
'P1602_StopMagicScan';

//*********************
```

```
function  P1602_DelayUs;          external 'P1602.DLL' name 'P1602_DelayUs';

//*******************
function  P1602_Card0_StartScan;  external 'P1602.DLL' name
'P1602_Card0_StartScan';
function  P1602_Card0_ReadStatus; external 'P1602.DLL' name
'P1602_Card0_ReadStatus';
procedure P1602_Card0_Stop;       external 'P1602.DLL' name 'P1602_Card0_Stop';
function  P1602_Card1_StartScan;  external 'P1602.DLL' name
'P1602_Card1_StartScan';
function  P1602_Card1_ReadStatus; external 'P1602.DLL' name
'P1602_Card1_ReadStatus';
procedure P1602_Card1_Stop;       external 'P1602.DLL' name 'P1602_Card1_Stop';

//*******************
function  P1602_FunA_Start;       external 'P1602.DLL' name 'P1602_FunA_Start';
function  P1602_FunA_ReadStatus;  external 'P1602.DLL' name
'P1602_FunA_ReadStatus';
function  P1602_FunA_Stop;        external 'P1602.DLL' name 'P1602_FunA_Stop';
function  P1602_FunA_Get;         external 'P1602.DLL' name 'P1602_FunA_Get';

//*******************
function  P1602_FunB_Start;       external 'P1602.DLL' name 'P1602_FunB_Start';
function  P1602_FunB_ReadStatus;  external 'P1602.DLL' name
'P1602_FunB_ReadStatus';
function  P1602_FunB_Stop;        external 'P1602.DLL' name 'P1602_FunB_Stop';
function  P1602_FunB_Get;         external 'P1602.DLL' name 'P1602_FunB_Get';

end.
```

2.4 LabVIEW Call DLLs

LabVIEW is an industrial graphical programming system developed by National Instruments. With LabVIEW, the user can quickly design a user interface and application program as a block diagram.

NAPPCI\Win\VIEW\P1602.Dll	**→ DLLs**
NAPPCI\Win\VIEW\DEMO1.VI	**→ Demo VI**
NAPPCI\Win\VIEW\MFUN1.VI	**→ Driver VI**

NOTE:

1. Tested under **Windows 95/98/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 A/D input at the same time. If you connect D/A channel_0 to A/D channel_0, the M_Functions can measure the D/A output back. The output response is shown in Fig 8 and the connecting diagram of demo1.VI is given in Fig. 9.

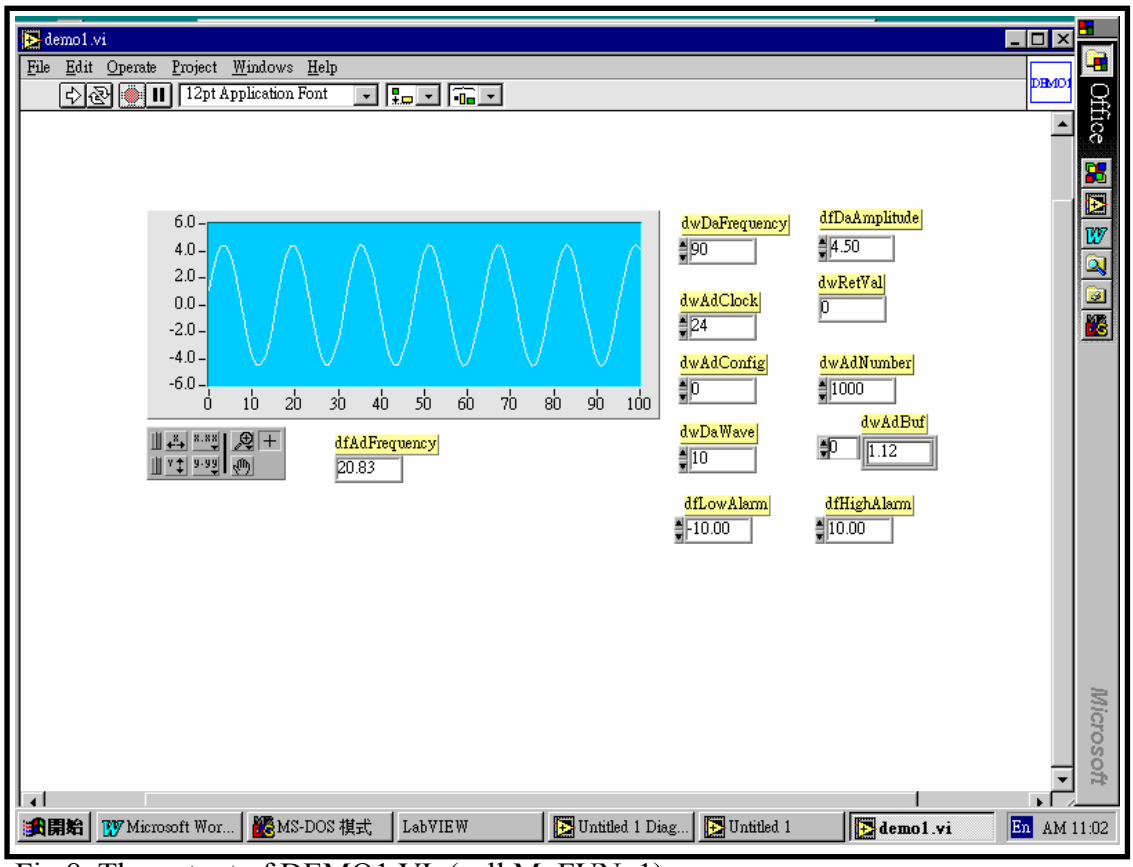


Fig 8. The output of DEMO1.VI. (call M_FUN_1)

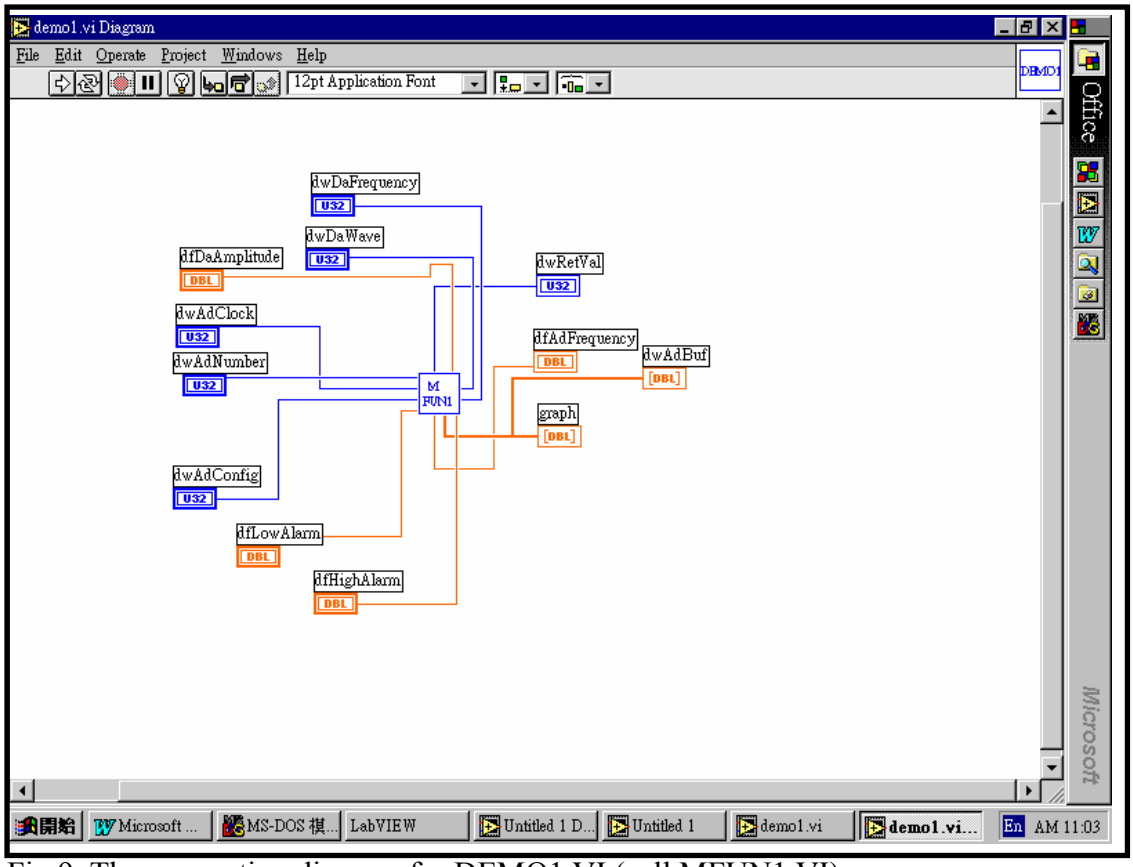


Fig 9. The connection diagram for DEMO1.VI (call MFUN1.VI)

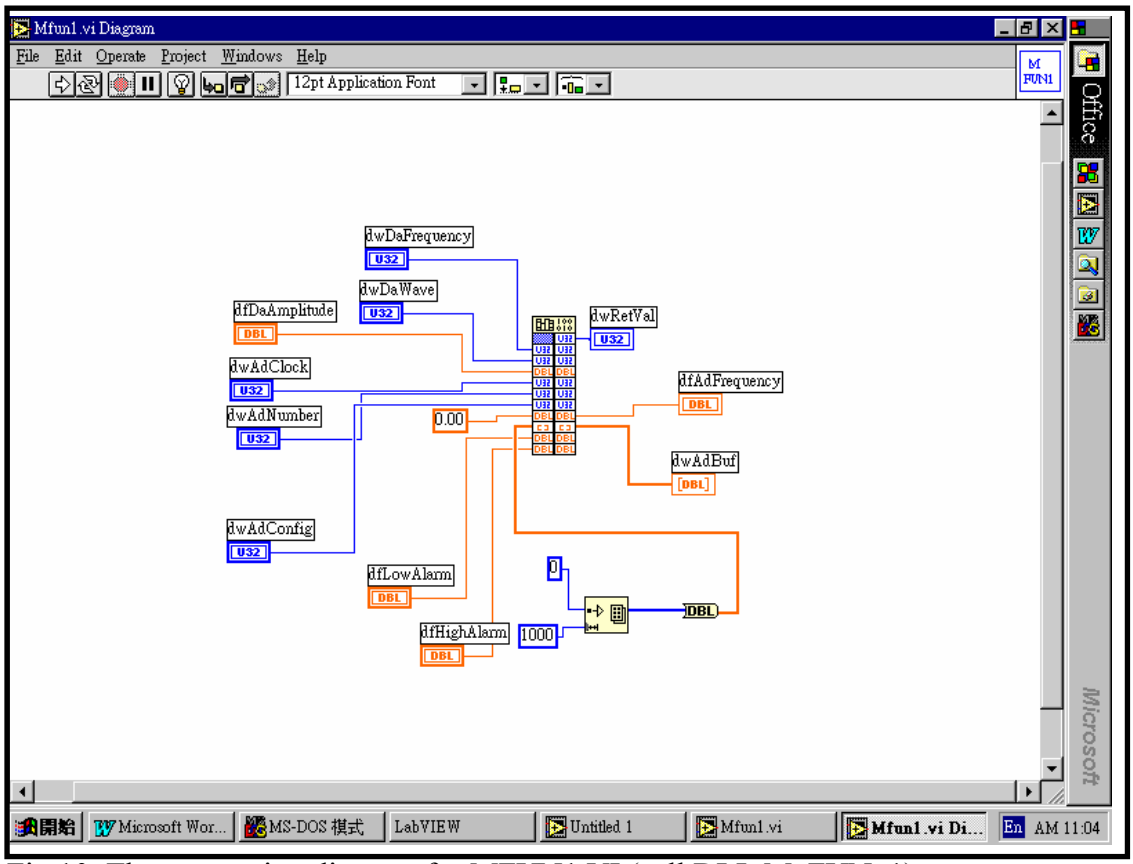


Fig 10. The connection diagram for MFUN1.VI (call DLL M_FUN_1)

2.5 Demo Program

A common demo program is used for all P1602.dll examples. The demo program will accept **wDaFreq** and **wAdClk** and call the different driver functions for demonstration purposes.

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "P1602.H"
/*********************************************************************/
/*    DEMO1 program for one P1602 cards in the PC system.            */
/*   Please set the resolution of your monitor to at least 1024*768 */
/*********************************************************************/
/* First Card: some P1602 function call demo.                        */
/* For the proper operation the P1602, the following functions       */
/* must be used.                                                     */
/*    P1602_DriverInit();     <--   initial the driver               */
/*    P1602_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[] = "P1602 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,"P1602 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 P1602_DriverInit() to initialize the driver. */
/*************************************************************/
// Initialize the device driver, and return the board number in the PC
       wInitialCode=P1602_DriverInit(&wTotalBoard);
       if( wInitialCode!=NoError )
       {
            MessageBox(hwnd,"No P1602 card in this system !!!",
                    "P1602 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);
```

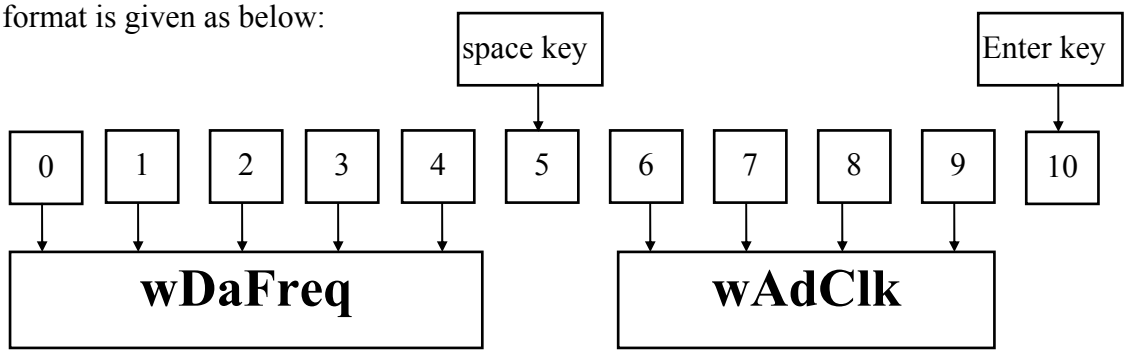
```
      ShowCaret(hwnd);
      return 0;
 case WM_KILLFOCUS :
      HideCaret(hwnd);
      DestroyCaret();
      return 0;

 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 P1602_DriverClose() to close the driver.     */
/*************************************************************/
        P1602_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 placed here

}

The READ_COM only accepts **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 similar setup code as above. The separate testing functions are placed in "**TEST_CMD(….) { …….}**". Only the code that would be placed in **TEST_CMD** is shown in subsequent listings in this manual.

3. Description of Functions

The DLL functions are divided 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 A/D continuous capture functions
- The A/D batch capture functions
- The Plug & Play functions
- Other functions

- The functions of the fixed-channel mode are as follows:

1. P1602_SetChannelConfig
2. P1602_AdPoling
3. P1602_AdsPolling
4. P1602_AdsPacer

data in float format

- The functions of the MagicScan mode are as follows:

1. P1602_ClearScan
2. P1602_StartScan
3. P1602_AddToScan
4. P1602_SaveScan
5. P1602_ReadMagicScanResult

data in 16 bits HEX format

- The functions of the M_functions are as follows:

1. P1602_M_FUN_1
2. P1602_M_FUN_2
3. P1602_M_FUN_3

- The multiboard batch capture functions are as follows:

1. P1602_FunA_Start
2. P1602_FunA_ReadStatus
3. P1602_FunA_Stop
4. P1602_FunA_Get

- The single board batch capture functions are as follows:

1. P1602_FunB_Start
2. P1602_FunB_ReadStatus
3. P1602_FunB_Stop
4. P1602_FunB_Get

- The continuous capture functions are as following:

1. P1602_Card0_StartScan
2. P1602_Card0_ReadStatus
3. P1602_Card0_StopScan
4. P1602_Card1_StartScan
5. P1602_Card1_ReadStatus
6. P1602_Card1_StopScan

Group-0: for card_0 continuous capture function

Group-1: for card_1 continuous capture function

3.1 The Configuration Code Table

OME-PCI-1602 Configuration Code Table

Bipolar/Unipolar	Input Signal Range	Gain	Settling Time	Configuration Code
Bipolar	+/- 10V	1	10 us	0
Bipolar	+/- 5V	2	10 us	1
Bipolar	+/- 2.5V	4	10 us	2
Bipolar	+/- 1.25V	8	10 us	3

OME-PCI-1602F Configuration Code Table

Bipolar/Unipolar	Input Signal Range	Gain	Settling Time	Configuration Code
Bipolar	+/- 10V	1	5 us	0
Bipolar	+/- 5V	2	5 us	1
Bipolar	+/- 2.5V	4	5 us	2
Bipolar	+/- 1.25V	8	5 us	3

3.2 The Test Functions

3.2.1 P1602_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 P1602_FloatSub2(float fA, float fB);
- **Input Parameter :**

fA : 4 bytes floating point value

fB : 4 bytes floating point value

- **Return Value :** return=fA-fB
- **Demo Program : DEMO1.C**

3.2.2 P1602_ShortSub2

- **Description :**

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

- **Syntax :** short P1602_ShortSub2(Short nA, Short nB);
- **Input Parameter :**

nA : 16 bits value

nB : 16 bits value

- **Return Value :** return=nA-nB
- **Demo Program : DEMO1.C**

3.2.3 P1602_GetDllVersion

- **Description :**
  Reads the version of the P1602.DLL.

- **Syntax :** WORD P1602_GetDllVersion(void);

- **Input Parameter :** void

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

- **Demo Program : DEMO1.C**

3.2.4 P1602_GetDriverVersion

- **Description :** This function will read the software version number of Nappci.VxD for Windows 95/98 or Napwnt.SYS for Windows NT.

- **Syntax :** WORD P1602_GetDriverVersion(WORD *wDriverVersion);

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

- **Return Value :**
  NoError : OK
  DriverHandleError : the NAPPCI.VxD open error for Windows 95/98
  the Napwnt.SYS open error for Windows NT
  DriverCallError : call NAPPCI.VxD return error
  call Napwnt.SYS return error

- **Demo Program : DEMO1.C**

3.3 The M_Functions

3.3.1 P1602_M_FUN_1

- **Description :**

The P1602_M_FUN_1 will calculate the waveform image automatically. (Refer to the "OME-PCI-1602 Hardware Manual" chapter-5 for details) (input=A/D channel_0, output=D/A channel_0)

- **Syntax :**

WORD P1602_M_FUN_1(WORD wDaFrequency, WORD wDaWave, float fDaAmplitude, WORD wAdClock, WORD wAdNumber, WORD wAdConfig, float fAdBuf[], float fLowAlarm, float fHighAlarm)

- **Input Parameter :**

wDaFrequency : **D/A output frequency = 1.8M/wDaFrequency (Pentium 120)**

wDaWave : Number of D/A waveform to be output

fDaAmplitude : Amplitude of D/A output. NOTE : the hardware J1 must select +/-10V

wAdClock : **A/D sampling clock = 8000000/wAdClock** samples/sec

wAdNumber: Number of A/D data to be read

wAdConfig : **A/D input range configuration code**

Refer to "Section 3.1 Configuration Table"

fAdBuf[] : the starting address of **fAdBuf** which store the A/D data

fLowAlarm : low alarm limit. if **fAdBuf[?]< fLowAlarm** → LowAlarm

fHighAlarm : high alarm limit. if **fAdBuf[?]>fHighAlarm** → HighAlarm

- **Return Value :**

NoError : OK

DriverHandleError : Invalid VxD/SYS handle

DriverCallError : VxD/SYS function call error

ExceedBoardNumber: invalid board number

FindBoardError: no OME-PCI-1602 board

AdControllerError : embedded controller handshake error

M_FunExecError : M_Functions return code error

ConfigCodeError : **wAdConfig** configuration code error

Refer to "Section 3.1 Configuration Table"

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

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

- **Demo Program : DEMO5.C**

3.3.2 P1602_M_FUN_2

- **Description :**

The P1602_M_FUN_2 will **not** compute the waveform image automatically. (Refer to "OME-PCI-1602 Hardware Manual" chapter-5 for details) (input=A/D channel_0, output=D/A channel_0)

- **Syntax :**

WORD P1602_M_FUN_2(WORD wDaNumber, WORD wDaWave, WORD wDaBuf[], WORD wAdClock, WORD wAdNumber, WORD wAdConfig, WORD wAdBuf[]);

- **Input Parameter :**

wDaNumber: number of D/A samples in one waveform

wDaWave : Number of D/A waveform to be output

wDaBuf[] : The array stores the D/A waveform image

wAdClock : **A/D sampling clock = 8000000/wAdClock** samples/sec

wAdNumber: Number of A/D data to be read

wAdConfig : **A/D input range configuration code.**

Refer to "Section 3.1 Configuration Table"

wAdBuf[] : the starting address of **fAdBuf** which store the A/D data

- **Return Value :**

NoError : OK

DriverHandleError : Invalid VxD/SYS handle

DriverCallError : VxD/SYS function call error

ExceedBoardNumber: invalid board number

FindBoardError: no OME-PCI-1602 board

AdControllererror : embedded controller handshake error

M_FunExecError : M_Functions return code error

ConfigCodeError : **wAdConfig** configuration code error, Refer to "Section 3.1"

- **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. Some benchmarks are shown below:

D/A output frequency = 1.8M/dwDaNumber for Pentium 120

D/A output frequency = 2.0M/dwDaNumber for Pentium 133

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

3.3.3 P1602_M_FUN_3

- **Description :**

The P1602_M_FUN_3 will calculate the waveform image automatically. (Refer to "OME-PCI-1602 Hardware Manual" chapter-5 for details) (input=programable channels, output=D/A channel_0) This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board. Refer to Sec. 2.4.2 for more information.

- **Syntax :**

WORD P1602_M_FUN_3(WORD wDaFrequency, WORD wDaWave, float fDaAmplitude, WORD wAdClock, WORD wAdNumber, WORD wChannelStatus[], WORD wAdConfig[], float fAdBuf[], float fLowAlarm, float fHighAlarm)

- **Input Parameter :**

wDaFrequency : D/A **output frequency = 1.8M/wDaFrequency (Pentium 120)**

wDaWave : Number of D/A waveform 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 samples to be read

wAdChannel[]: 1=scan, 0=no scan

wAdConfig[]: **configuration code**

Refer to "Section 3.1 Configuration Table"

fAdBuf[] : the starting address of **fAdBuf** which store the A/D data

fLowAlarm : low alarm limit. if **fAdBuf[?]< fLowAlarm** → LowAlarm

fHighAlarm : high alarm limit. if **fAdBuf[?]>fHighAlarm** → HighAlarm

- **Return Value :**

NoError : OK

DriverHandleError : Invalid VxD/SYS handle

DriverCallError : VxD/SYS function call error

ExceedBoardNumber: invalid board number

FindBoardError: no OME-PCI-1602 board

AdControllerError : embedded controller handshake error

M_FunExecError : M_Functions return code error

ConfigCodeError : **wAdConfig** configuration code error, Refer to "Section 3.1"

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

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

- **Demo Program : DEMO9.C**

3.4 The DIO Functions

3.4.1 P1602_Di

- **Description :** This function will read the 16 bit data from the digital input (D/I) port. This function will refer to the current active OME-PCI-1602. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax :** WORD P1602_Di(WORD *wDi);
- **Input Parameter :**

  *wDi : address of **wDi** which contains the 16 bits of D/I data
- **Return Value :**

  NoError : OK

  FindBoardError : cannot find the OME-PCI-1602 board

  ExceedBoardNumber: invalid board number
- **Demo Program : DEMO1.C**

3.4.2 P1602_Do

- **Description :** This function will write the 16 bit data to the digital output(D/O) port. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax :** WORD P1602_Do(WORD wDo);
- **Input Parameter :**

  wDo : the 16 bits of data sent to the D/O port
- **Return Value :**

  NoError : OK

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-1602 board
- **Demo Program : DEMO1.C**

3.5 The D/A Functions

3.5.1 P1602_Da

Description: This function will write the 12 bit data to D/A port. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.

- **Syntax :** WORD P1602_Da(WORD wChannel, WORD wDaVal);
- **Input 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 depending on the setting hardware jumper JP1. The software cannot detect the state of JP1, the user must be aware of its state.
- **Return Value :**

  NoError : OK

  FindBoardError : cannot find the OME-PCI-1602 board

  ExceedBoardNumber: invalid board number

  DaChannelError : channel number must be 0 or 1
- **Demo Program : DEMO1.C**

3.6 The A/D Fixed-mode Functions

3.6.1 P1602_SetChannelConfig

- **Description :** This function will set the A/D channel's configuration code. This function will set the active A/D channel for **P1602_AdPolling**, **P1602_AdsPolling** and **P1602_AdsPacer** functions. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax :** WORD P1602_SetChannelConfig(WORD wChannel, WORD wConfig);
- **Input Parameter :**

  wChannel : A/D channel number

  wConfig : Configuration code. Refer to "Section 3.1 Configuration Table"
- **Return Value :**

  NoError : OK

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-1602 board

  AdControllerError : MagicScan controller hardware handshake error
- **Demo Program : DEMO1.C**

3.6.2 P1602_AdPolling

- **Description :** This function will perform a single A/D conversion by software polling. The **P1602_SetChannelConfig** function can be used to change channel or configuration code used by the **P1602_AdPolling** function. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax :** WORD P1602_AdPolling(float *fAdVal);
- **Input Parameter :**

  *fAdVal : address of **fAdVal** which will contain the A/D data (16 bits), this data is automatically converted to volts based on the setting **P1602_SetChannelConfig**.
- **Return Value :**

  NoError : OK

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-1602 board

  AdPollingTimeOut : hardware timeout error
- **Demo Program : DEMO1.C**

3.6.3 P1602_AdsPolling

- **Description :** This function will perform multiple A/D conversions by polling. The **P1602_SetChannelConfig** function can be used to change the channel or configuration code used by the **P1602_AdsPolling** function. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax :** WORD P1602_AdsPolling(float fAdVal[], WORD wNum);
- **Input Parameter :**

  fAdVal[]: starting address of A/D data buffer (16 bit), the data will be automatically converted to volts based on the setting of the **P1602_SetChannelConfig** function.

  wNum: number of A/D conversions to be performed.
- **Return Value :**

  NoError: OK

  ExceedBoardNumber: invalid board number

  FindBoardError: cannot find the OME-PCI-1602 board

  AdPollingTimeOut: hardware timeout error
- **Demo Program : DEMO1.C**

3.6.4 P1602_AdsPacer

- **Description :** This function will perform multiple A/D conversions by pacer trigger. The **P1602_SetChannelConfig** function can be used to change channel or configuration code. The hardware pacer will generate a periodic trigger signal to the A/D converter. Software polling is used by the P1602_AdsPolling function, so the A/D conversion process could be interrupted by the computer's operating system. Since the P1602_AdsPacer function uses the hardware pacer, operating system interrupts will not affect it. For this reason **the P1602_AdsPacer function should be used if a waveform must be precisely reconstructed.** This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.

- **Syntax :** WORD P1602_AdsPacer(float fAdVal[], WORD wNum, WORD wSample);

- **Input Parameter :**

  fAdVal[] : starting address of the A/D data buffer (16 bit), the data will be automatically converted to volts based on the setting of the **P1602_SetChannelConfig** function.

  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 Value :**

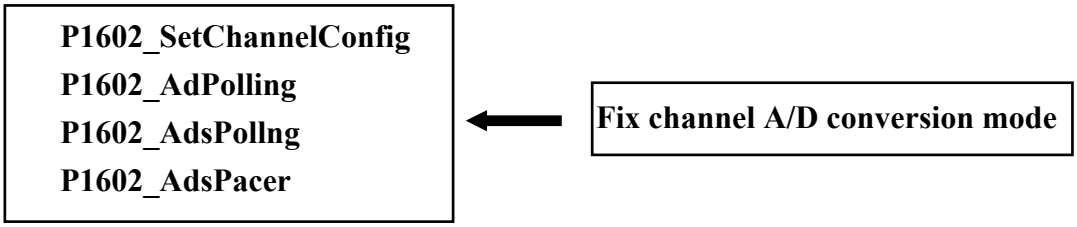
  NoError : OK

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-1602 board

  AdPollingTimeOut : hardware timeout error

- **Demo Program : DEMO1.C**



3.7 The MagicScan Functions

3.7.1 P1602_ClearScan

- **Description :** This function will initialize the MagicScan controller. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax :** WORD P1602_ClearScan();
- **Input Parameter :** void
- **Return Value :**

  NoError : OK

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-1602 board

  AdControllerError : MagicScan controller hardware handshake error
- **Demo Program : DEMO11.C**

3.7.2 P1602_StartScan

- **Description :** This function will start the MagicScan operation. **This function will return to the caller before the MagicScan operation finishes.** The user can use the **P1602_WaitMagicScanFinish(…)** function or the **P1602_ReadScanStatus(…)** function to check the state of MagicScan operation. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax :** WORD P1602_StartScan(WORD wSampleRate, WORD wNum);
- **Input Parameter :**

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

  wSampleRate=24 → sample rate=8M/24=330K

  wNum : Number of **MagicScan cycles** to perform
- **Return Value :**

  NoError : OK

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-1602 board

  AdControllerError : MagicScan controller hardware handshake error
- **Demo Program : DEMO11.C**

3.7.3 P1602_ReadScanStatus

- **Description :** This function will read the status of the MagicScan operation. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax :** void P1602_ReadScanStatus(WORD *wStatus, WORD *wLowAlarm, WORD *wHighAlarm);
- **Input Parameter :**

*wStatus : address of **wStatus** which will contain the MagicScan status

*wLowAlarm : address of **wLowAlarm** which will contain the MagicScan alarm status

*wHighAlarm : address of **wHighAlarm** which will contain the MagicScan alarm status

- Return Value : void
- **Demo Program : DEMO11.C**

wStatus	**= 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 → 32 bits corresponding to 32 channels
→ 0 = no low alarm
→ 1 = is low alarm
wLowAlarm=0 → all channels OK, no low alarm
wLowAlarm=1 → channel_0 is low alarm, others are OK
wLowAlarm=3 → channel_0 and channel_1 are low alarm, others are OK

wHighAlarm → 32 bits corresponding to 32 channels
→ 0 = no high alarm
→ 1 = is high alarm
wHighAlarm=0 → all channels OK, no high alarm
wHighAlarm=1 → channel_0 is high alarm, others are OK
wHighAlarm=3 → channel_0 and channel_1 are high alarm, others are OK

3.7.4 P1602_AddToScan

- **Description :** This function will add one channel to the **MagicScan circular queue**. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) function to select the active board.
- **Syntax :** word P1602_AddToScan(WORD wAdChannel, WORD wConfig, WORD wAverage, WORD wLowAlarm, WORD wHighAlarm, WORD wAlarmType);
- **Input Parameter :**

  wAdChannel : A/D channel number

  wConfig : the configuration code

  Refer to "Section 3.1 Configuration Table"

  wAverage : the digital average filter factor

  wLowAlarm : 16 bit low alarm data

  wHighAlarm : 16 bit high alarm data

  wAlarmType : 0=no alarm, 1=high alarm, 2=low alarm, 3=in-alarm, 4=out-alarm
- Return Value :

  NoError : Ok

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-1602 board

  AdChannelError : invalid A/D channel

  AlarmTypeError : only 0/1/2/3/4 are valid

  AdControllerError : MagicScan controller hardware handshake error
- **Demo Program : DEMO11.C**

3.7.5 P1602_SaveScan

- **Description :** This function will specify the starting address of A/D data buffer for MagicScan.
- **Syntax :** void P1602_SaveScan(WORD wAdChannel, WORD wBuf[]);
- **Input Parameter :**

  wAdChannel : Scan number in the scan queue.

  (Note: not the A/D channel number.)

  wBuf : starting address of the A/D data buffer for the channel specified in **wAdChannel**
- Return Value :

  NoError : Ok

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-1602 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=P1602_ClearScan();
//****    For OME-PCI-1602L
 wRetVal += P1602_AddToScan(0,0,1,0,0,0); // CH:0 to scan
 wRetVal += P1602_SaveScan(0,wV0);
 wRetVal += P1602_AddToScan(2,0,1,0,0,0); // CH:2 to scan
 wRetVal += P1602_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
P1602_StartScan(wSampleRateDiv,DATALENGTH,nPriority);
```

3.7.6 P1602_WaitMagicScanFinish

- **Description :** This function will delay until the MagicScan operation is finished. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax :** void P1602_WaitMagicScanFinish(WORD *wStatus, WORD *wLowAlarm, WORD *wHighAlarm);
- **Input Parameter :**

  *wStatus : address of **wStatus** which will contain the MagicScan status

  *wLowAlarm : address of **wLowAlarm** which will contain the MagicScan alarm status

  *dwHighAlarm : address of **wHighAlarm** which will contain the MagicScan alarm status
- Return Value : void
- **Demo Program : DEMO11.C**

wStatus	**= 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	→ 32 bits corresponding to 32 channels
	→ 0 = no low alarm
	→ 1 = is low alarm
wLowAlarm=0	→ all channels OK, no low alarm
wLowAlarm=1	→ channel_0 is low alarm, others are OK
wLowAlarm=3	→ channel_0 and channel_1 are low alarm, others are OK

wHighAlarm	→ 32 bits corresponding to 32 channels
	→ 0 = no high alarm
	→ 1 = is high alarm
wHighAlarm=0	→ all channels OK, no high alarm
wHighAlarm=1	→ channel_0 is high alarm, others are OK
wHighAlarm=3	→ channel_0 and channel_1 are high alarm, others are OK

3.8 The Pulg&Play Functions

3.8.1 P1602_DriverInit

- **Description:** This function will detect all of the OME-PCI-1602 boards installed in the system. This function must be called once before any other function is called.

- **Syntax :** WORD P1602_DriverInit(WORD *wTotalBoard);

- **Input Parameter :** *wTotalBoard: address of **wTotalBoard**
  wTotalBoard=1 → one OME-PCI-1602 card in the system
  wTotalBoard=n → n*OME-PCI-1602 cards in the system

- **Return Value :**
  NoError : OK
  NoFoundBoard: can not detect any OME-PCI-1602
  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.**

3.8.2 P1602_DriverClose

- **Description:** Returns all resources to the system. This function should be called before the program is terminated.
- **Syntax :** void P1602_DriverClose(void);
- **Input Parameter :** void
- **Return Value :** void
- **Demo Program: All DEMO programs.**

3.8.3 P1602_GetConfigAddressSpace

- **Description:** Gets the I/O address of OME-PCI-1602 board n. This function is for debugging purposes only. It is not necessary to call this function.
- **Syntax :** WORD P1602_GetConfigAddressSpace(WORD wBoardNo, WORD *wAddrTimer,WORD *wAddrCtrl, WORD *wAddrDio, WORD *wAddrAdda);
- **Input Parameter :**

  wBoardNo: OME-PCI-1602 board number

  wAddrTimer, wAddrCtrl, wAddrDio, wAddrAdda: refer to the "OME-PCI-1602 Hardware manual" chapter-3 for details.
- **Return Value :**

  NoError : OK

  FindBoardError: handshake check error

  ExceedBoardError: wBoardNo is invalid
- **Demo Program : DEMO1.C**

3.8.4 P1602_WhichBoardActive

- **Description:** Returns the board number of the active board.
- **Syntax:** WORD P1602_WhichBoardActive(void);
- **Input Parameter:** void
- **Return Value:** board number of the active board.
- **Demo Program: DEMO1.C**

3.8.5 P1602_ActiveBoard

- **Description:** This function will make active one of the OME-PCI-1602 boards installed in the system. This function must call once before the D/I/O, A/D, D/A functions are called.

- **Syntax:** WORD P1602_ActiveBoard(WORD wBoardNo);

- **Input Parameter:**
  wBoardNo: board number

- **Return Value :**
  NoError : OK
  ExceedBoardError: wBoardNo is invalid

- **Demo Program : All DEMO programd.**

The P1602_ActiveBoard(…) function will effect all functions except the following:

1. P1602_FloatSub2
2. P1602_ShortSub2
3. P1602_GetDriverVersion
4. P1602_DriveInit
5. P1602_DriveClose
6. P1602_GetConfigAddressSpace
7. P1602_Card0_StartScan
8. P1602_Card0_ReadData
9. P1602_Card0_Stop
10. P1602_Card1_StartScan
11. P1602_Card1_ReadData
12. P1602_Card1_Stop

3.9 Multiboard Batch Capture
(Two boards operating simultaneously)

3.9.1 P1602_FunA_Start

- **Description:** This function will start the batch capture process for two boards operating simultaneously.

- **Syntax :**

```
WORD P1602_FunA_Start(WORD wClockDiv0, WORD wChannel0[],
                      WORD wConfig0[], WORD *Buffer0, DWORD dwMaxCount0,
                      WORD wClockDiv1, WORD wChannel1[],
                      WORD wConfig1[], WORD *Buffer1, DWORD dwMaxCount1
                      Short  nPriority);
```

- **Input 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 3.1 Configuration Table"
*Buffer0: buffer to store the A/D data of the first board
dwMaxCount0: sample count for the first board
wClockDiv1: the A/D sample rate divisor 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 3.1 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 range from:

-2:	THREAD_PRIORITY_LOWEST
-1:	THREAD_PRIORITY_BELOW_NORMAL
0:	THREAD_PRIORITY_NORMAL
1:	THREAD_PRIORITY_ABOVE_NORMAL
2:	THREAD_PRIORITY_HIGHEST
Other:	THREAD_PRIORITY_NORMAL

- **Return Value :**

  NoError : OK

  FindTwoBoardError : cannot find out two OME-PCI-1602 boards

- **Demo Program : DEMO20.C**

3.9.2 P1602_FunA_ReadStatus

- **Description :** This function will read the status of the batch capture process.

- **Syntax :**

  WORD P1602_FunA_ReadStatus(void);

- **Input Parameter :**

  void;

- **Return Value :**

  0: data is ready

  1: data not ready

- **Demo Program : DEMO20.C**

3.9.3 P1602_FunA_Stop

- **Description:** This function will stop the batch capture function.
- **Syntax:**
  word P1602_FunA_Stop(void);
- **Input Parameter:**
  void
- **Return Value :**
  NoError : OK
  StopError : Stop Error
- **Demo Program : DEMO20.C**

3.9.4 P1602_FunA_Get

- **Description:** This function will retrieve the number A/D samples acquired.
- **Syntax:**
  word P1602_FunA_Get(DWORD *P0, DWORD *P1);
- **Input 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 Value :**
  NoError : OK
- **Demo Program : DEMO20.C**

3.10 The Single Board Batch Capture

3.10.1 P1602_FunB_Start

- **Description :** This function will start the batch capture process.

- **Syntax :**

```
WORD P1602_FunB_Start(WORD wClockDiv0, WORD wChannel0[],
                      WORD wConfig0[], WORD *Buffer0, DWORD dwMaxCount0,
                      SHORT nPriority);
```

- **Input 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 3.1 Configuration Table"
*Buffer0: buffer to store the A/D data of this board
dwMaxCount0: to specify the data length of this board
nPriority: Thread priority. The value of nPriority ranges from:

-2:	THREAD_PRIORITY_LOWEST
-1:	THREAD_PRIORITY_BELOW_NORMAL
0:	THREAD_PRIORITY_NORMAL
1:	THREAD_PRIORITY_ABOVE_NORMAL
2:	THREAD_PRIORITY_HIGHEST
Other:	THREAD_PRIORITY_NORMAL

- **Return Value :**

NoError : OK
FindBoardError : cannot find the OME-PCI-1602 board
AdControllerError : MagicScan controller hardware handshake error

- **Demo Program : DEMO21.C**

3.10.2 P1602_FunB_ReadStatus

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

- **Syntax :**

  WORD P1602_FunB_ReadStatus(void);

- **Input Parameter :**

  void;

- **Return Value :**

  0: data is ready

  1: data not ready

- **Demo Program : DEMO21.C**

3.10.3 P1602_FunB_Stop

- **Description:** This function will stop the batch capture function.
- **Syntax:**
  word P1602_FunB_Stop(void);
- **Input Parameter:**
  void
- **Return Value :**
  NoError : OK
  StopError : Stop Error
- **Demo Program : DEMO21.C**

3.10.4 P1602_FunB_Get

- **Description:** This function will retrieve the number of A/D samples that have been acquired.
- **Syntax:**
  word P1602_FunB_Get(DWORD *P0);
- **Input Parameter:**
  *P0: [output] the number of A/D samples that have been acquired.
- **Return Value :**
  NoError : OK
- **Demo Program : DEMO21.C**

3.10.5 P1602_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 collection. Although computer dependent, sample rates should generally be kept under 40kHz. Refer to the OME-PCI-1602 Hardware User Manual, for additional details on this function.
- **Syntax :** WORD P1602_Card0_StartScan(WORD wSampleRate, WORD wChannelStatus[], WORD wChanelConfig[], WORD wCount);
- **Input 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 3.1 Configuration Table"

  wCount: number of A/D data for each scan channel
- **Return Value :**

  NoError : OK

  FindBoardError : cannot find the OME-PCI-1602 board

  AdControllerError : MagicScan controller hardware handshake error
- **Demo Program : DEMO13.C**

3.10.6 P1602_Card0_ReadStatus

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

- **Syntax :** P1602_Card0_ReadStatus(WORD wBuf[], WORD wBuf2[], DWORD *dwP1, DWORD *dwP2, WORD *wStatus);
- **Input 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 Value :**

  0: data is ready

  1: data not ready
- **Demo Program : DEMO13.C**

3.10.7 P1602_Card0_Stop

- **Description :** This function will stop the continuous capture function.
- **Syntax :** void P1602_Card0_Stop(void);
- **Input Parameter :void**
- **Return Value :void**
- **Demo Program : DEMO13.C**

3.10.8 P1602_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 collection. Although computer dependent, sample rates should generally be kept under 40kHz. Refer to the OME-PCI-1602 Hardware User Manual, for additional details on this function.
- **Syntax :** WORD P1602_Card1_StartScan(WORD wSampleRate, WORD wChannelStatus[], WORD wChanelConfig[], WORD wCount);
- **Input 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 3.1 Configuration Table"

  wCount: number of A/D data for each scan channel
- **Return Value :**

  NoError : OK

  FindBoardError : cannot find the OME-PCI-1602 board

  AdControllerError : MagicScan controller hardware handshake error
- **Demo Program : DEMO14.C**

3.10.9 P1602_Card1_ReadStatus

- **Description:** This function will read the data collected by the continuous capture function.
- **Syntax:** P1602_Card1_ReadStatus(WORD wBuf[], WORD wBuf2[], DWORD *dwP1, DWORD *dwP2, WORD *wStatus);
- **Input 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 Value:**

  0: data is ready

  1: data not ready
- **Demo Program: DEMO14.C**

3.10.10 P1602_Card1_Stop

- **Description:** This function will stop the continuous capture function.
- **Syntax:** void P1602_Card1_Stop(void);
- **Input Parameter:** void
- **Return Value:** void
- **Demo Program: DEMO14.C**

3.11 Other Functions

3.11.1 P1602_DelayUs

- **Description:** This is a **machine independent timer.** This function can be used to create the **settling time delay** or used as a **general purpose machine** independent timer. This function will refer to the current active OME-PCI-1602 board. Use the P1602_ActiveBoard(….) to select the active board.
- **Syntax:** word P1602_DelayUs(WORD wDelayUs);
- **Input Parameter:**

  wDelayUs : number of us to delay, 8191 Max

  wDelayUs=1 → delay 1 μs

  wDelayUs=1000 → delay 1000 us = 1 ms

  wDelayUs=8191 → delay 8191 us = 8.191 ms (maximum delay)

  wDelayUs=8192 → invalid delay (will return error)
- **Return Value:**

  NoError : OK

  ExceedBoardNumber: invalid board number

  FindBoardError : cannot find the OME-PCI-1602 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+=P1602_DelayUs(1000);
return(wRetVal);
}
```

4. 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 boards, 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 95/98/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 board, MagicScan demo, scan 32 channel, show channel 0/1/15/16/17
- demo19: one board, A/D calibration.
- demo20: two boards, P1602_FUNA, batch capture demo
- demo21: single board, P1602_FUNB, batch capture 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

M3931/0203