PIO-DA

Software Manual

[For Windows 95/98/NT/2000]

Warranty

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for damage consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, ICP DAS assumes no responsibility for its use, or for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

The names used for identification only maybe registered trademarks of their respective companies.

License

The user can use, modify and backup this software **on a single machine.** The user may not reproduce, transfer or distribute this software, or any copy, in whole or in part.

Table of Contents

1. Introduction

The software is a collection of digital I/O subroutines for PIO-DIO series add-on cards for Windows 95/98/NT and Windows NT 2000 Applications. These subroutines are written with C language and perform a variety of digital I/O operations.

The subroutines in PIODA.DLL are easy understanding as its name standing for. It provides powerful, easy-to-use subroutine for developing your data acquisition application. Your program can call these DLL functions by VC++, VB, Delphi, and BORLAND C++ Builder easily. To speed-up your developing process, some demonstration source program are provided.

Please refer to the following user manuals:

- **PnPInstall.pdf:**
  To install the PnP (Plug and Play) driver for PCI card under Windows 95/98.

- **SoftInst.pdf:**
  To install the software package under Windows 95/98/NT/2000.

- **CallDll.pdf:**
  To call the DLL functions with VC++5, VB5, Delphi3 and Borland C++ Builder 3.

- **ResCheck.pdf:**
  To check the resources I/O Port address, IRQ number and DMA number for add-on cards under Windows 95/98/NT/2000.

- **PIO-DA16s.pdf:**
  PIO-DA16/DA8/DA4 Hardware manual.

2. Declaration Files

```
|--\Driver                  ← some device driver
      |--\BCB3              ← for Borland C++ Builder 3
      |      |--\PIODA.H    ← Header file
      |      +--\PIODA.LIB  ← Linkage library for BCB3 only
      |
      |--\Delphi3           ← for Delphi 3
      |      +--\PIODA.PAS  ← Declaration file
      |
      |--\VB5               ← for Visual Basic 5
      |      +--\PIODA.BAS  ← Declaration file
      |
      +--\VC5               ← for Visual C++ 5
            |--\PIODA.H     ← Header file
            +--\PIODA.LIB   ← Linkage library for VC5 only
```

2.1 PIODA.H

```
#ifdef __cplusplus
    #define EXPORTS extern "C" __declspec (dllimport)
#else
    #define EXPORTS
#endif

// return code
#define PIODA_NoError                   0
#define PIODA_DriverOpenError           1
#define PIODA_DriverNoOpen              2
#define PIODA_GetDriverVersionError     3
#define PIODA_InstallIrqError           4
#define PIODA_ClearIntCountError        5
#define PIODA_GetIntCountError          6
#define PIODA_RegisterApcError          7
#define PIODA_RemoveIrqError            8
#define PIODA_FindBoardError            9
#define PIODA_ExceedBoardNumber         10
#define PIODA_ResetError                11

#define PIODA_EEPROMDataError           12
#define PIODA_EEPROMWriteError          13


// to trigger a interrupt when high -> low
#define PIODA_ActiveLow             0
// to trigger a interrupt when low -> high
#define PIODA_ActiveHigh            1

// ID
#define PIO_DA                  0x800400

// Test functions
EXPORTS float       CALLBACK PIODA_FloatSub(float fA, float fB);
EXPORTS short       CALLBACK PIODA_ShortSub(short nA, short nB);
EXPORTS WORD        CALLBACK PIODA_GetDllVersion(void);

// Driver functions
EXPORTS WORD        CALLBACK PIODA_DriverInit(void);
EXPORTS void        CALLBACK PIODA_DriverClose(void);
EXPORTS WORD        CALLBACK PIODA_SearchCard
            (WORD *wBoards, DWORD dwPIOCardID);
EXPORTS WORD        CALLBACK PIODA_GetDriverVersion
            (WORD *wDriverVersion);
EXPORTS WORD        CALLBACK PIODA_GetConfigAddressSpace
```

```
            (WORD wBoardNo,    DWORD *wAddrBase,  WORD *wIrqNo,
             WORD *wSubVendor, WORD  *wSubDevice, WORD *wSubAux,
             WORD *wSlotBus,   WORD  *wSlotDevice );
EXPORTS WORD   CALLBACK PIODA_ActiveBoard( WORD wBoardNo );
EXPORTS WORD   CALLBACK PIODA_WhichBoardActive(void);
EXPORTS WORD   CALLBACK PIODA_SetCounter(WORD wBoardNo,
            WORD wWhichCounter, WORD bConfig, DWORD wValue);
EXPORTS DWORD  CALLBACK PIODA_GetBaseAddress
           (WORD wBoardNo);

// EEPROM functions
EXPORTS WORD   CALLBACK PIODA_EEP_READ
            (WORD wBoardNo, WORD wOffset, WORD *bHi, WORD *bLo);
EXPORTS WORD   CALLBACK PIODA_EEP_WR_EN(WORD wBoardNo);
EXPORTS WORD   CALLBACK PIODA_EEP_WR_DIS(WORD wBoardNo);
EXPORTS WORD   CALLBACK PIODA_EEP_WRITE
            (WORD wBoardNo, WORD wOffset, WORD HI, WORD LO);

// DA functions
EXPORTS WORD   CALLBACK PIODA_Voltage
            (WORD wBoardNo, WORD wChannel, float fValue);
EXPORTS WORD   CALLBACK PIODA_Current
            (WORD wBoardNo, WORD wChannel, float fValue);
EXPORTS WORD   CALLBACK PIODA_CalVoltage
            (WORD wBoardNo, WORD wChannel, float fValue);
EXPORTS WORD   CALLBACK PIODA_CalCurrent
            (WORD wBoardNo, WORD wChannel, float fValue);

// DIO functions
EXPORTS void   CALLBACK PIODA_OutputWord
            (DWORD wBaseAddress, DWORD wOutData);
EXPORTS void   CALLBACK PIODA_OutputByte
            (DWORD wBaseAddress, WORD bOutputValue);
EXPORTS DWORD  CALLBACK PIODA_InputWord
           (DWORD wBaseAddress);
EXPORTS WORD   CALLBACK PIODA_InputByte(DWORD wBaseAddress);
EXPORTS WORD   CALLBACK PIODA_DI
           (WORD wBoardNo, DWORD *wVal);
EXPORTS WORD   CALLBACK PIODA_DO
           (WORD wBoardNo, DWORD wDO);

// Interrupt functions
EXPORTS WORD   CALLBACK PIODA_IntInstall
            (WORD wBoardNo, HANDLE *hEvent,
           WORD wInterruptSource, WORD wActiveMode);
EXPORTS WORD   CALLBACK PIODA_IntRemove(void);
EXPORTS WORD   CALLBACK PIODA_IntResetCount(void);
EXPORTS WORD   CALLBACK PIODA_IntGetCount(DWORD *dwIntCount);
```

2.2 PIODA.BAS

```
Attribute VB_Name = "PIODA"

Global Const PIODA_NoError = 0
Global Const PIODA_DriverOpenError = 1
Global Const PIODA_DriverNoOpen = 2
Global Const PIODA_GetDriverVersionError = 3
Global Const PIODA_InstallIrqError = 4
Global Const PIODA_ClearIntCountError = 5
Global Const PIODA_GetIntCountError = 6
Global Const PIODA_RegisterApcError = 7
Global Const PIODA_RemoveIrqError = 8
Global Const PIODA_FindBoardError = 9
Global Const PIODA_ExceedBoardNumber = 10
Global Const PIODA_ResetError = 11

Global Const PIODA_EEPROMDataError = 12
Global Const PIODA_EEPROMWriteError = 13


' to trigger a interrupt when high -> low
Global Const PIODA_ActiveLow = 0
' to trigger a interrupt when low -> high
Global Const PIODA_ActiveHigh = 1


' ID
Global Const PIO_DA = &H800400          ' PIO-DA16/DA8/DA4


' The Test functions
Declare Function PIODA_ShortSub Lib "PIODA.dll" _
        (ByVal a As Integer, ByVal b As Integer) As Integer
Declare Function PIODA_FloatSub Lib "PIODA.dll" _
        (ByVal a As Single, ByVal b As Single) As Single
Declare Function PIODA_GetDllVersion Lib "PIODA.dll" () As Integer
```

```
' The Driver functions
Declare Function PIODA_DriverInit Lib "PIODA.dll" () As Integer
Declare Sub PIODA_DriverClose Lib "PIODA.dll" ()
Declare Function PIODA_SearchCard Lib "PIODA.dll" _
      (wBoards As Integer, ByVal dwPIOPISOCardID As Long) As Integer
Declare Function PIODA_GetDriverVersion Lib "PIODA.dll" _
      (wDriverVersion As Integer) As Integer
Declare Function PIODA_GetConfigAddressSpace Lib "PIODA.dll" ( _
      ByVal wBoardNo As Integer, wAddrBase As Long, wIrqNo As Integer, _
      wSubVendor As Integer, wSubDevice As Integer, wSubAux As Integer, _
      wSlotBus As Integer, wSlotDevice As Integer) As Integer

Declare Function PIODA_ActiveBoard Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer) As Integer
Declare Function PIODA_WhichBoardActive Lib "PIODA.dll" () As Integer
Declare Function PIODA_SetCounter Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer, ByVal wWhichCounter As Integer, _
       ByVal bConfig As Integer, ByVal wValue As Long) As Long
Declare Function PIODA_GetBaseAddress Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer) As Long

' EEPROM functions
Declare Function PIODA_EEP_READ Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer, ByVal wOffset As Integer, _
       bHi As Integer, bLo As Integer) As Integer
Declare Function PIODA_EEP_WR_EN Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer) As Integer
Declare Function PIODA_EEP_WR_DIS Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer) As Integer
Declare Function PIODA_EEP_WRITE Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer, ByVal wOffset As Integer, _
       ByVal HI As Integer, ByVal LO As Integer) As Integer

' DA functions
Declare Function PIODA_Voltage Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer, ByVal wChannel As Integer, _
       ByVal fValue As Single) As Integer
Declare Function PIODA_Current Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer, ByVal wChannel As Integer, _
       ByVal fValue As Single) As Integer
Declare Function PIODA_CalVoltage Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer, ByVal wChannel As Integer, _
       ByVal fValue As Single) As Integer
Declare Function PIODA_CalCurrent Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer, ByVal wChannel As Integer, _
       ByVal fValue As Single) As Integer
```

```
' DIO functions
Declare Sub PIODA_OutputByte Lib "PIODA.dll" _
      (ByVal wBaseAddress As Long, ByVal dataout As Integer)
Declare Sub PIODA_OutputWord Lib "PIODA.dll" _
      (ByVal wBaseAddress As Long, ByVal dataout As Long)
Declare Function PIODA_InputByte Lib "PIODA.dll" _
      (ByVal wBaseAddress As Long) As Integer
Declare Function PIODA_InputWord Lib "PIODA.dll" _
      (ByVal wBaseAddress As Long) As Long
Declare Function PIODA_DI Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer, wVal As Long) As Integer
Declare Function PIODA_DO Lib "PIODA.dll" _
      (ByVal wBoardNo As Integer, ByVal wDO As Long) As Integer

' Interrupt functions
Declare Function PIODA_IntInstall Lib "PIODA.dll" _
      (ByVal wBoard As Integer, hEvent As Long, _
      ByVal wInterruptSource As Integer, _
      ByVal wActiveMode As Integer) As Integer
Declare Function PIODA_IntRemove Lib "PIODA.dll" () As Integer
Declare Function PIODA_IntResetCount Lib "PIODA.dll" () As Integer
Declare Function PIODA_IntGetCount Lib "PIODA.dll" _
      (dwIntCount As Long) As Integer
```

2.3 PIODA.PAS

```
unit  PIODA;        { PIODA.dll interface unit }

interface

const
  PIODA_NoError                 =0;
  PIODA_DriverOpenError         =1;
  PIODA_DriverNoOpen            =2;
  PIODA_GetDriverVersionError   =3;
  PIODA_InstallIrqError         =4;
  PIODA_ClearIntCountError      =5;
  PIODA_GetIntCountError        =6;
  PIODA_RegisterApcError        =7;
  PIODA_RemoveIrqError          =8;
  PIODA_FindBoardError          =9;
  PIODA_ExceedBoardNumber       =10;
  PIODA_ResetError              =11;

  PIODA_EEPROMDataError         =12;
  PIODA_EEPROMWriteError        =13;

  // to trigger a interrupt when high -> low
  PIODA_ActiveLow               =0;
  // to trigger a interrupt when low -> high
  PIODA_ActiveHigh              =1;

  // ID
  PIO_DA                = $800400;    // PIO-DA16/DA8/DA4


// Test functions
function  PIODA_ShortSub(nA : smallint; nB : smallint) :smallint; StdCall;
function  PIODA_FloatSub(fA :   single; fB : single) :single; StdCall;
function  PIODA_GetDllVersion : word; StdCall;
```

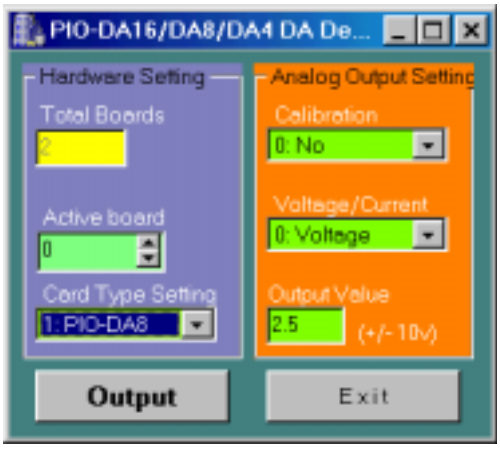
```
// Driver functions
function  PIODA_DriverInit : word; StdCall;
procedure PIODA_DriverClose ; StdCall;
function  PIODA_SearchCard
        (var wBoards:WORD; dwPIOPISOCardID:LongInt):WORD; StdCall;
function  PIODA_GetDriverVersion(var wDriverVer: word):WORD; StdCall;
function  PIODA_GetConfigAddressSpace
        (wBoardNo:word; var wAddrBase:LongInt;  var wIrqNo:word;
        var wSubVerdor:word; var wSubDevice:word; var wSubAux:word;
        var wSlotBus:word;   var wSlotDevice:word ): word; StdCall;
function  PIODA_ActiveBoard(wBoardNo:Word) :WORD; StdCall;
function  PIODA_WhichBoardActive :WORD; StdCall;
function  PIODA_SetCounter(wBoardNo:WORD; wWhichCounter:WORD;
        bConfig:WORD;  wValue:LongInt): WORD; StdCall;
function  PIODA_GetBaseAddress(wBoardNo:WORD):LongInt; StdCall;

// EEPROM functions
function  PIODA_EEP_READ(wBoardNo:WORD; wOffset:WORD;
        var bHi:WORD; var bLo:WORD):WORD; StdCall;
function  PIODA_EEP_WR_EN(wBoardNo:WORD):WORD; StdCall;
function  PIODA_EEP_WR_DIS(wBoardNo:WORD):WORD; StdCall;
function  PIODA_EEP_WRITE( wBoardNo:WORD; wOffset:WORD;
        HI:WORD; LO:WORD):WORD; StdCall;

// DA functions
function  PIODA_Voltage
        (wBoardNo:WORD; wChannel:WORD; fValue:Single):WORD; StdCall;
function  PIODA_Current
        (wBoardNo:WORD; wChannel:WORD; fValue:Single):WORD; StdCall;
function  PIODA_CalVoltage
        (wBoardNo:WORD; wChannel:WORD; fValue:Single):WORD; StdCall;
function  PIODA_CalCurrent
        (wBoardNo:WORD; wChannel:WORD; fValue:Single):WORD; StdCall;

// DIO functions
procedure PIODA_OutputByte
        (wBaseAddress :LongInt; bOutputVal :Word); StdCall;
procedure PIODA_OutputWord
        (wBaseAddress :LongInt; wOutputVal :LongInt); StdCall;
function  PIODA_InputByte(wBaseAddress  :LongInt ) :word; StdCall;
function  PIODA_InputWord(wBaseAddress  :LongInt ) :LongInt; StdCall;
function  PIODA_DI(wBoardNo:WORD; var wVal:LongInt) :word; StdCall;
function  PIODA_DO(wBoardNo:WORD; wDO:LongInt) :word; StdCall;
```

```
// Interrupt functions
function  PIODA_IntInstall(wBoard:Word; var hEvent:LongInt;
        wInterruptSource:Word; wActiveMode:Word):Word; StdCall;
function  PIODA_IntRemove : WORD; StdCall;
function  PIODA_IntResetCount : WORD; StdCall;
function  PIODA_IntGetCount(var dwIntCount:LongInt) : WORD; StdCall;


implementation

// Test functions
function  PIODA_ShortSub;
        external 'PIODA.DLL' name 'PIODA_ShortSub';
function  PIODA_FloatSub;
        external 'PIODA.DLL' name 'PIODA_FloatSub';
function  PIODA_GetDllVersion;
        external 'PIODA.DLL' name 'PIODA_GetDllVersion';

// Driver functions
function  PIODA_DriverInit;
        external 'PIODA.DLL' name 'PIODA_DriverInit';
procedure PIODA_DriverClose;
        external 'PIODA.DLL' name 'PIODA_DriverClose';
function  PIODA_SearchCard;
        external 'PIODA.DLL' name 'PIODA_SearchCard';
function  PIODA_GetDriverVersion;
        external 'PIODA.DLL' name 'PIODA_GetDriverVersion';
function  PIODA_GetConfigAddressSpace;
        external 'PIODA.DLL' name 'PIODA_GetConfigAddressSpace';

function  PIODA_ActiveBoard;
        external 'PIODA.DLL' name 'PIODA_ActiveBoard';
function  PIODA_WhichBoardActive;
        external 'PIODA.DLL' name 'PIODA_WhichBoardActive';
function  PIODA_SetCounter;
        external 'PIODA.DLL' name 'PIODA_SetCounter';
function  PIODA_GetBaseAddress;
        external 'PIODA.DLL' name 'PIODA_GetBaseAddress';

// EEPROM functions
function  PIODA_EEP_READ;
        external 'PIODA.DLL' name 'PIODA_EEP_READ';
function  PIODA_EEP_WR_EN;
        external 'PIODA.DLL' name 'PIODA_EEP_WR_EN';
function  PIODA_EEP_WR_DIS;
        external 'PIODA.DLL' name 'PIODA_EEP_WR_DIS';
function  PIODA_EEP_WRITE;
        external 'PIODA.DLL' name 'PIODA_EEP_WRITE';
```

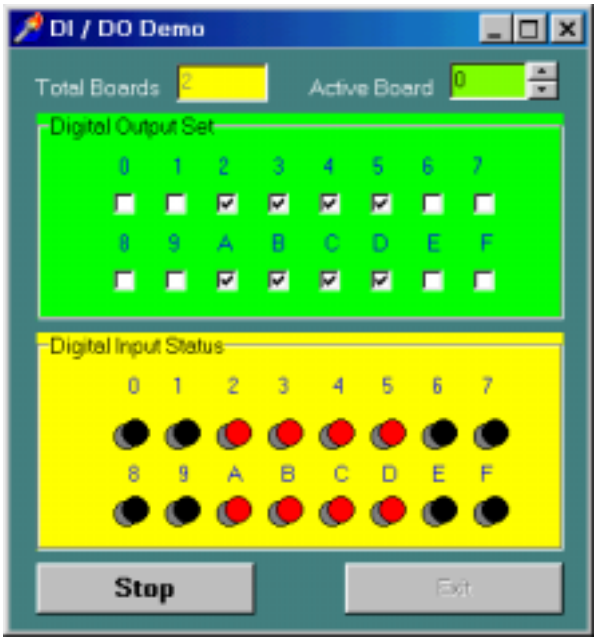
```
// DA functions
function  PIODA_Voltage;
        external 'PIODA.DLL' name 'PIODA_Voltage';
function  PIODA_Current;
        external 'PIODA.DLL' name 'PIODA_Current';
function  PIODA_CalVoltage;
        external 'PIODA.DLL' name 'PIODA_CalVoltage';
function  PIODA_CalCurrent;
        external 'PIODA.DLL' name 'PIODA_CalCurrent';

// DIO functions
procedure PIODA_OutputByte;
        external 'PIODA.DLL' name 'PIODA_OutputByte';
procedure PIODA_OutputWord;
        external 'PIODA.DLL' name 'PIODA_OutputWord';
function  PIODA_InputByte;
        external 'PIODA.DLL' name 'PIODA_InputByte';
function  PIODA_InputWord;
        external 'PIODA.DLL' name 'PIODA_InputWord';
function  PIODA_DI;
        external 'PIODA.DLL' name 'PIODA_DI';
function  PIODA_DO;
        external 'PIODA.DLL' name 'PIODA_DO';

// Interrupt functions
function  PIODA_IntInstall;
        external 'PIODA.DLL' name 'PIODA_IntInstall';
function  PIODA_IntRemove;
        external 'PIODA.DLL' name 'PIODA_IntRemove';
function  PIODA_IntGetCount;
        external 'PIODA.DLL' name 'PIODA_IntGetCount';
function  PIODA_IntResetCount;
        external 'PIODA.DLL' name 'PIODA_IntResetCount';

end.
```

3. Demo Result

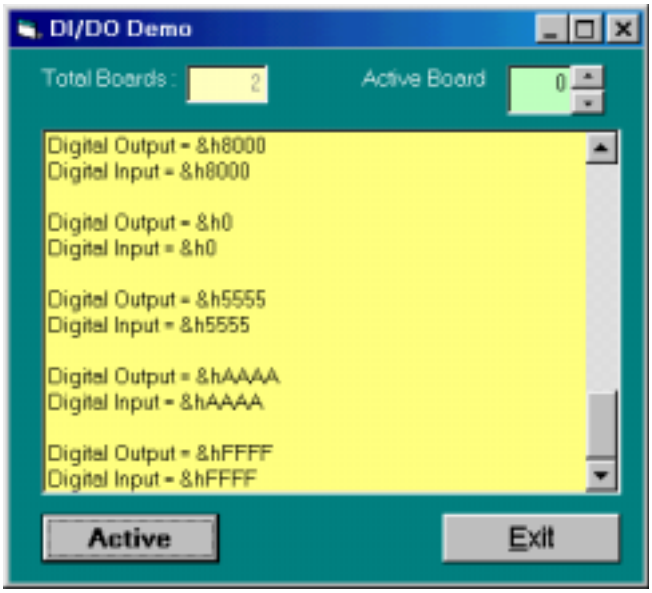
Diagnostic Program



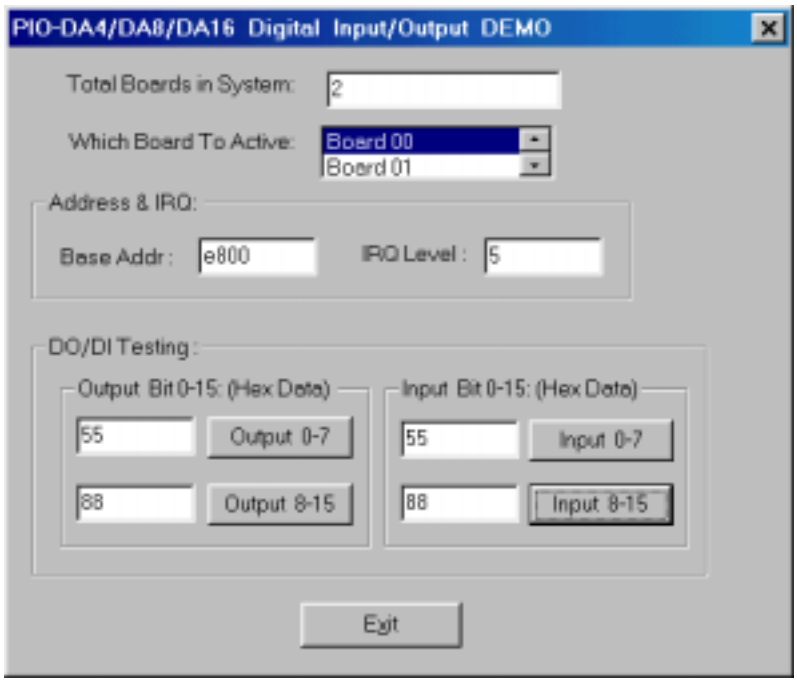
DA Demo for BCB 3



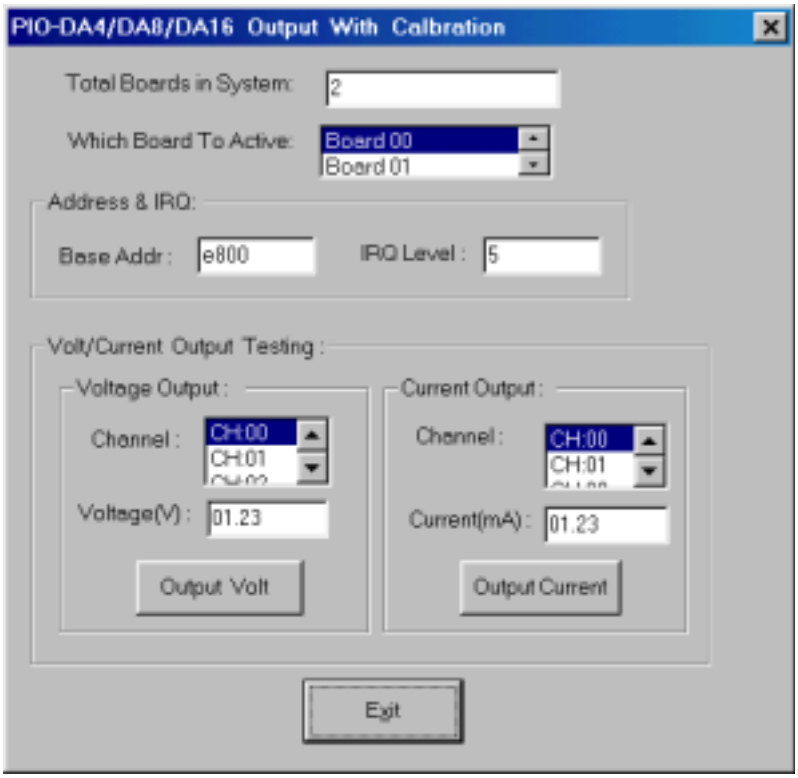
DI/DO Demo for Delphi 3



DI/DO Demo for VB 5



DI/DO Demo for VC 5



DA Demo for VC 5

4. Function Descriptions

In this chapter, we use some keywords to indicate the attribute of Parameters.

Keyword	Setting parameter by user before calling this function ?	Get the data/value from this parameter after calling this function ?
[Input]	Yes	No
[Output]	No	Yes
[Input, Output]	Yes	Yes

Note: All of the parameters need to be allocated spaces by the user.

4.1 Test Functions

4.1.1 PIODA_GetDllVersion

- **Description:**
  To get the version number of PIODA.DLL

- **Syntax:**
  WORD PIODA_GetDllVersion(Void)

- **Parameter:**
  None

- **Return:**
  200(hex) for version 2.00

4.1.2 PIODA_ShortSub

- **Description:**
  To perform the subtraction as nA - nB in short data type. This function is provided for testing DLL linkage purpose.

- **Syntax:**
  short PIODA_ShortSub(short nA, short nB)

- **Parameter:**
  nA :[Input] 2 bytes short data type value
  nB :[Input] 2 bytes short data type value

- **Return:**
  The value of nA - nB

4.1.3 PIODA_FloatSub

- **Description:**
  To perform the subtraction as fA - fB in float data type. This function is provided for testing DLL linkage purpose.

- **Syntax:**
  float PIODA_FloatSub(float fA, float fB)

- **Parameter:**
  fA : [Input] 4 bytes floating point value
  fB : [Input] 4 bytes floating point value

- **Return:**
  The value of fA - fB

4.2 I/O Functions

4.2.1 PIODA_OutputByte

- **Description :**
  This subroutine will send the 8 bits data to the desired I/O port.

- **Syntax :**
  void PIODA_OutputByte(DWORD wPortAddr, WORD bOutputVal);

- **Parameter :**
  wPortAddr : [Input] I/O port addresses, please refer to function PIODA_GetConfigAddressSpace.
  Only the low WORD is valid.
  bOutputVal : [Input] 8 bit data send to I/O port.
  Only the low BYTE is valid.

- **Return:**
  None

4.2.2 PIODA_InputByte

- **Description :**
  This subroutine will input the 8 bit data from the desired I/O port.

- **Syntax :**
  WORD PIODA_InputByte(DWORD wPortAddr);

- **Parameter :**
  wPortAddr : [Input] I/O port addresses, please refer to function
  PIODA_GetConfigAddressSpace().
  Only the low WORD is valid.

- **Return:**
  16 bits data with the leading 8 bits are all 0.
  (Only the low BYTE is valid.)

4.2.3 PIODA_OutputWord

- **Description :**
  This subroutine will send the 16 bits data to the desired I/O port.

- **Syntax :**
  void PIODA_OutputWord(DWORD wPortAddr, DWORD wOutputVal);

- **Parameter :**
  wPortAddr : [Input] I/O port addresses, please refer to function PIODA_GetConfigAddressSpace().
  Only the low WORD is valid.
  wOutputVal : [Input] 16 bit data send to I/O port.
  Only the low WORD is valid.

- **Return:**
  None

4.2.4 PIODA_InputWord

- **Description :**
  This subroutine will input the 16 bit data from the desired I/O port.

- **Syntax :**
  DWORD PIODA_InputWord(DWORD wPortAddr);

- **Parameter :**
  wPortAddr : [Input] I/O port addresses, please refer to function
  PIODA_GetConfigAddressSpace().
  Only the low WORD is valid.

- **Return:**
  16 bit data. Only the low WORD is valid.

4.2.5 PIODA_DO

- **Description :**
  This subroutine will send the 16 bits data to the desired card.

- **Syntax :**
  WORD PIODA_DO(WORD wBoardNo, DWORD wDO);

- **Parameter :**
  wBoardNo : [Input] Which board to active.
  wDO : [Input] The 16-bit data to Digital-Output.
  Only the low WORD is valid.

- **Return:**
  PIODA_NoError : OK

4.2.6 PIODA_DI

- **Description :**
  This subroutine will input the 16 bit data from the desired card.

- **Syntax :**
  WORD PIODA_DI(WORD wBoardNo, DWORD *wVal);

- **Parameter :**
  wBoardNo : [Input] Which board to active.
  wVal : [Output] Stores the Digital-Input value after called this function.
  Only the low WORD is valid.

- **Return:**
  PIODA_NoError : OK

4.3 Driver Functions

4.3.1 PIODA_GetDriverVersion

- **Description :**
  This subroutine will read the version number of PIODA driver.

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

- **Parameter :**
  wDriverVersion : [Output] address of wDriverVersion

- **Return:**
  PIODA_NoError : OK
  PIODA_DriverNoOpen : The PIODA driver no open
  PIODA_GetDriverVersionError : Read driver version error

4.3.2 PIODA_DriverInit

- **Description :**
  This subroutine will open the PIODA driver and allocate the resource for the device. This function must be called once before calling other PIODA functions.

- **Syntax :**
  WORD PIODA_DriverInit();

- **Parameter :**
  None

- **Return:**
  PIODA_NoError : OK
  PIODA_DriverOpenError : open PIODA Driver error

4.3.3 PIODA_DriverClose

- **Description :**
  This subroutine will close the PIODA Driver and release the resource from the device. This function must be called once before exit the user's application.

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

- **Parameter :**
  None

- **Return:**
  None

4.3.4 PIODA_GetConfigAddressSpace

- **Description :**
  Get the I/O address of PIODA board n.

- **Syntax :**

```
WORD PIODA_GetConfigAddressSpace
        ( WORD wBoardNo,   DWORD *wAddrBase,     WORD *wIrqNo,
        WORD *wSubVendor, WORD *wSubDevice,     WORD *wSubAux,
        WORD *wSlotBus,    WORD *wSlotDevice);
```

- **Parameter :**

wBoardNo	: [Input] PIODA board number
wAddrBase	: [Output] The base address of PIODA board. Only the low WORD is valid.
wIrqNo	: [Output] The IRQ number that the PIODA board using.
wSubVendor	: [Output] Sub Vendor ID.
wSubDevice	: [Output] Sub Device ID.
wSubAux	: [Output] Sub Aux ID.
wSlotBus	: [Output] Slot Bus number.
wSlotDevice	: [Output] Slot Device ID.

- **Return:**

PIODA_NoError	: OK
PIODA_FindBoardError	: handshake check error
PIODA_ExceedBoardError	: wBoardNo is invalidated

4.3.5 PIODA_GetBaseAddress

- **Description :**
  Get the I/O address of PIODA board n.

- **Syntax :**
  DWORD PIODA_GetBaseAddress(WORD wBoardNo);

- **Parameter :**
  wBoardNo : [Input] PIODA board number

- **Return:**
  0 : Error
  Other values : The base-address of that board.

4.3.6 PIODA_SearchCard

- **Description :**
  Search the cards by specified Card-ID. This function will automatically read the EEPROM data for each board that found. And will automatically enable the each board.

- **Syntax :**
  WORD PIODA_SearchCard(WORD *wBoards, DWORD dwPIOCardID);

- **Parameter :**
  wBoards : [Output] How many cards be found.
  dwPIOCardID : [Input] What kinds of card to find?
  The user must fill this with PIO_DA.

- **Return:**
  PIODA_NoError : OK

4.3.7 PIODA_SetCounter

- **Description :**
  Set the value to the specified Counter for the Interrupt using.

- **Syntax :**
  WORD PIODA_SetCounter(WORD wBoardNo,
  WORD wWhichCounter, WORD bConfig, DWORD wValue);

- **Parameter :**

wBoardNo	: [Input]	PIODA board number
wWhichCounter	: [Input]	Counter number. (0 to 2)
bConfig	: [Input]	Configuration code. Please refer to 8254 spec.
wValue	: [Input]	Counter value to be set. Only the low-part of word is valid. (16-bit)

- **Return:**
  PIODA_NoError : OK

4.4 EEPROM Functions

4.4.1 PIODA_EEP_Read

- **Description:**
  Read the EEPROM data for the specified board and offset.

- **Syntax:**
  WORD PIODA_EEP_READ
  (WORD wBoardNo, WORD wOffset, WORD *bHi, WORD *bLo);

- **Parameter:**
  wBoardNo : [Input] Which board to be used.
  wOffset : [Input] The offset address for the EEPROM. (0 to 63)
  bHi : [Output] 8-bit data. The high-part of EEPROM data.
  bLo : [Output] 8-bit data. The low-part of EEPROM data.

- **Return:**
  PIODA_NoError : OK

4.4.2 PIODA_EEP_Write

- **Description:**
  Write data into the EEPROM for the specified board and offset. The wrong data may cause the board to output the wrong value (voltage/current). It's recommended not to uses this function. Before using the "PIODA_EEP_Write()" function to write the data into EEPROM, the user must to call the "PIODA_EEP_WR_EN()" function once firstly.

- **Syntax:**
  WORD PIODA_EEP_Write
  (WORD wBoardNo, WORD wOffset, WORD HI, WORD LO);

- **Parameter:**
  wBoardNo : [Input] Which board to be used.
  wOffset : [Input] The offset address for the EEPROM. (0 to 63)
  HI : [Input] 8-bit data. The high-part of EEPROM data.
  LO : [Input] 8-bit data. The low-part of EEPROM data.

- **Return:**
  PIODA_NoError : OK

4.4.3 PIODA_EEP_WR_EN

- **Description:**
  This function will enable the capability of the specified board to write data into EEPROM. The user must call this function once before calling the PIODA_EEP_Write() function.

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

- **Parameter:**
  wBoardNo : [Input] Which board to be used.

- **Return:**
  PIODA_NoError : OK

4.4.4 PIODA_EEP_WR_DIS

- **Description:**
  This function will disable the capability of the specified board to write data into EEPROM.

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

- **Parameter:**
  wBoardNo : [Input] Which board to be set.

- **Return:**
  PIODA_NoError : OK

4.5 DA Functions

4.5.1 PIODA_Voltage

- **Description:**
  This function will output the value of voltage (without the calibration) to the specified board and channel.

- **Syntax:**
  WORD PIODA_Voltage
  (WORD wBoardNo, WORD wChannel, float fValue);

- **Parameter:**
  wBoardNo : [Input] Which board to be used.
  wChannel : [Input] Which channel to output.
  fValue : [Input] What voltage value to output.

- **Return:**
  PIODA_NoError : OK

4.5.2 PIODA_Current

- **Description:**
  This function will output the value of current (without the calibration) to the specified board and channel.

- **Syntax:**
  WORD PIODA_Current
  (WORD wBoardNo, WORD wChannel, float fValue);

- **Parameter:**
  wBoardNo : [Input] Which board to be used.
  wChannel : [Input] Which channel to output.
  fValue : [Input] What current value to output.

- **Return:**
  PIODA_NoError : OK

4.5.3 PIODA_CalVoltage

- **Description:**
  This function will output the value of voltage to the specified board and channel. This function uses the EEPROM data to do the calibration.

- **Syntax:**
  WORD PIODA_CalVoltage
  (WORD wBoardNo, WORD wChannel, float fValue);

- **Parameter:**
  wBoardNo : [Input] Which board to be used.
  wChannel : [Input] Which channel to output.
  fValue : [Input] What voltage value to output.

- **Return:**
  PIODA_NoError : OK

4.5.4 PIODA_CalCurrent

- **Description:**
  This function will output the value of current to the specified board and channel. This function uses the EEPROM data to do the calibration.

- **Syntax:**
  WORD PIODA_CalCurrent
  (WORD wBoardNo, WORD wChannel, float fValue);

- **Parameter:**
  wBoardNo : [Input] Which board to be used.
  wChannel : [Input] Which channel to output.
  fValue : [Input] What current value to output.

- **Return:**
  PIODA_NoError : OK

4.6 Interrupt Functions

4.6.1 PIODA_IntInstall

- **Description:**
  This subroutine will install the IRQ service routine.

- **Syntax:**
  WORD PIODA_IntInstall(WORD wBoardNo, HANDLE *hEvent,
  WORD wInterruptSource, WORD wActiveMode);

- **Parameter:**
  wBoardNo : [Input] Which board to be used.
  hEvent : [Input] Address of a Event handle. The user's program must call the Windows API function "CreateEvent()" to create the event-object.
  wInterruptSource : [Input] What the Interrupt-Source to be used ? Please refer to hardware's manual for the detail information.

Card No.	wInterruptSource	Description
PIO-DA16/DA8/DA4	0	INT0
	1	INT1

  wActiveMode : [Input] When to trigger the interrupt ?
  This can be PIODA_ActiveHigh or PIODA_ActiveLow.

- **Return:**
  PIODA_NoError : OK
  PIODA_InstallIrqError : IRQ installation error

4.6.2 PIODA_IntRemove

- **Description:**
  This subroutine will remove the IRQ service routine.

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

- **Parameter:**
  None

- **Return:**
  PIODA_NoError : OK

4.6.3 PIODA_IntGetCount

- **Description:**
  This subroutine will read the d**wIntCount** defined in device driver.

- **Syntax :**
  WORD PIODA_IntGetCount(WDORD *dwIntCount);

- **Parameter:**
  dwIntCount : [Output] Address of dwIntCount, which will stores the counter value of interrupt.

- **Return:**
  PIODA_NoError : OK
  PIODA_GetIntCountError : **dwIntCount** read error

4.6.4 PIODA_IntResetCount

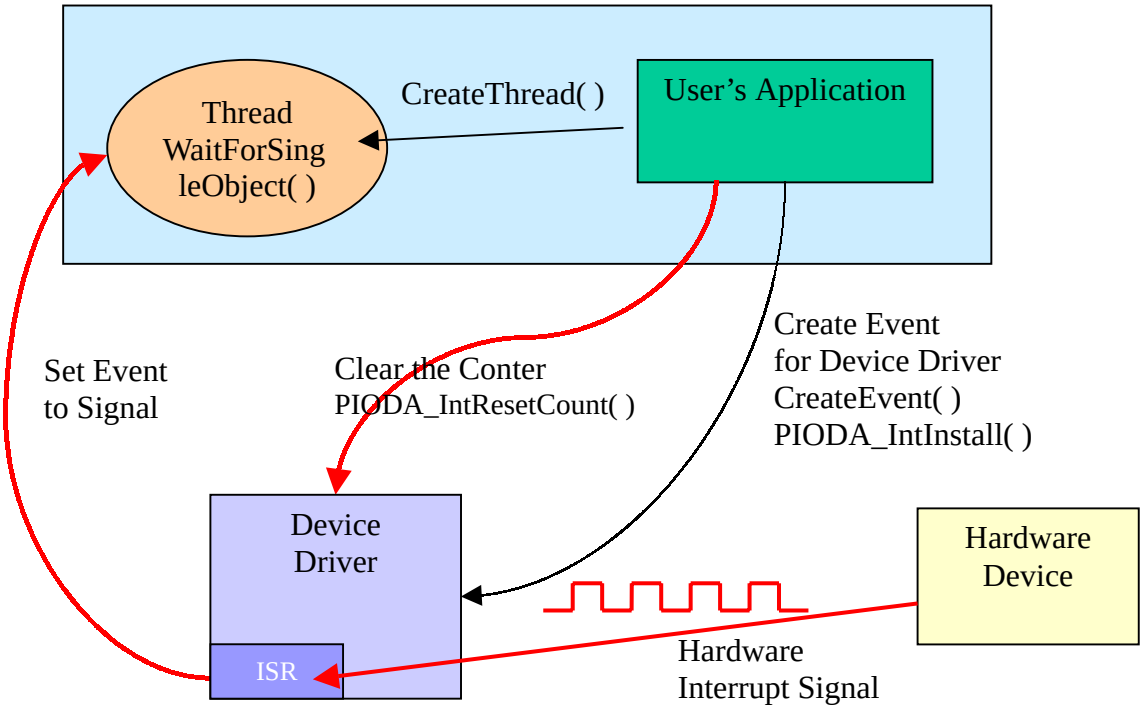
- **Description:**
  This function is used to clear the counter on the device driver for the interrupt.

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

- **Parameter:**
  None

- **Return:**
  PIODA_NoError : OK
  PIODA_ResetError : can't reset the counter

4.6.5 Architecture of Interrupt mode



Please refer to the following Windows API functions:

The following portion description of these functions was copied from MSDN. For the detailed and completely information, please refer to MSDN.

CreateEvent( )

The CreateEvent function creates or opens a named or unnamed event object.

```
HANDLE CreateEvent(
    // pointer to security attributes
    LPSECURITY_ATTRIBUTES lpEventAttributes,
    BOOL bManualReset,     // flag for manual-reset event
    BOOL bInitialState,    // flag for initial state
    LPCTSTR lpName         // pointer to event-object name
);
```

CreateThread( )

The CreateThread function creates a thread to execute within the virtual address space of the calling process.

To create a thread that runs in the virtual address space of another process, use the CreateRemoteThread function.

```
HANDLE CreateThread(
        // pointer to security attributes
        LPSECURITY_ATTRIBUTES lpThreadAttributes,
        DWORD dwStackSize,              // initial thread stack size
         // pointer to thread function
        LPTHREAD_START_ROUTINE lpStartAddress,
        LPVOID lpParameter,             // argument for new thread
        DWORD dwCreationFlags,        // creation flags
        LPDWORD lpThreadId             // pointer to receive thread ID
);
```

WaitForSingleObject( )

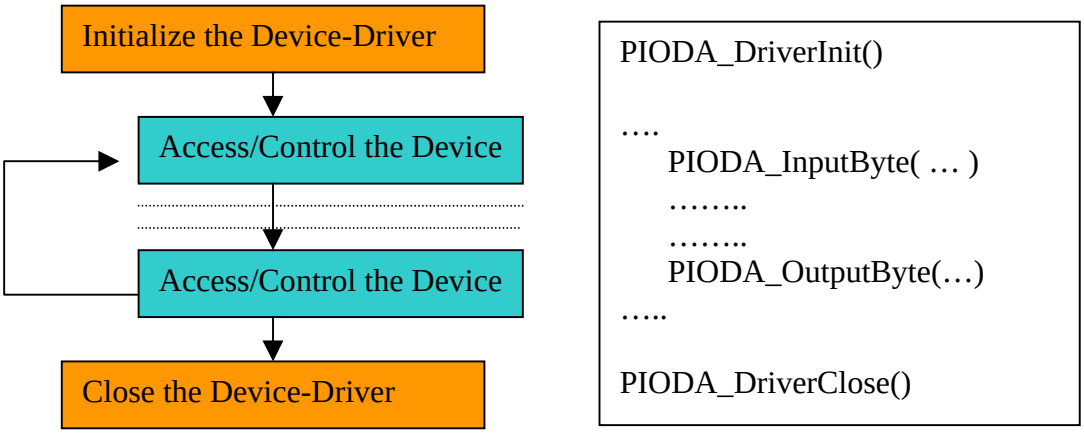
The WaitForSingleObject function returns when one of the following occurs:

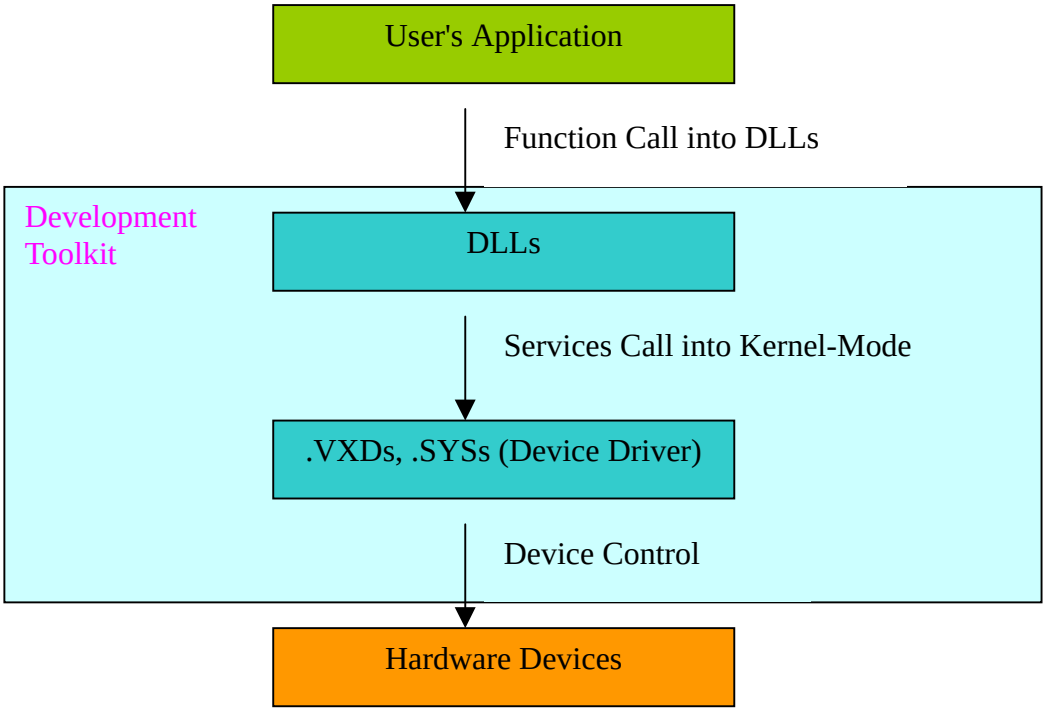
- The specified object is in the signaled state.
- The time-out interval elapses.

To enter an alertable wait state, use the WaitForSingleObjectEx function. To wait for multiple objects, use the WaitForMultipleObjects.

```
DWORD WaitForSingleObject(
            HANDLE hHandle,          // handle to object to wait for
            DWORD dwMilliseconds   // time-out interval in
    milliseconds
);
```

5. Program Architecture





6. Problems Report

Technical support is available at no charge as described below. The best way to report problems is to send electronic mail to

icpdas@ms8.hinet.net
Service@icpdas.com

on the Internet.

When reporting problems, please include the following information:

1) Is the problem reproducible? If so, how?
2) What kind and version of platform that you using? For example, Windows 3.1, Windows for Workgroups, Windows NT 4.0, etc.
3) What kinds of our products that you using? Please see the product's manual.
4) If a dialog box with an error message was displayed, please include the full text of the dialog box, including the text in the title bar.
5) If the problem involves other programs or hardware devices, what devices or version of the failing programs that you using?
6) Other comments relative to this problem or any suggestions will be welcomed.

After we had received your comments, we will take about two business days to test the problems that you said. And then reply as soon as possible to you. Please check that if we had received your comments? And please keeps contact with us.

ICP DAS
E-mail: icpdas@ms8.hinet.net
Service@icpdas.com
Web Site: http://www.icpdas.com
http://www.icpdas.com.tw