

PIO-DIO Series Classic Driver DLL Software Manual

Version 1.6, Jun. 2014

Supports

Board includes PIO-D24/D24U/D56/D56U, PIO-D48/D48U/D48SU, PIO-D64/D64U, PIO-D96/D96U/D96SU, PIO-D144/D144U/D144LU, PIO-D168A/D168/D168U, PEX-D24/D56, PEX-D48, PEX-D96S and PEX-D144LS.

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 damages 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, no responsibility is assumed by ICP DAS for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

Names are used for identification only and may be registered trademarks of their respective companies.

Contact US

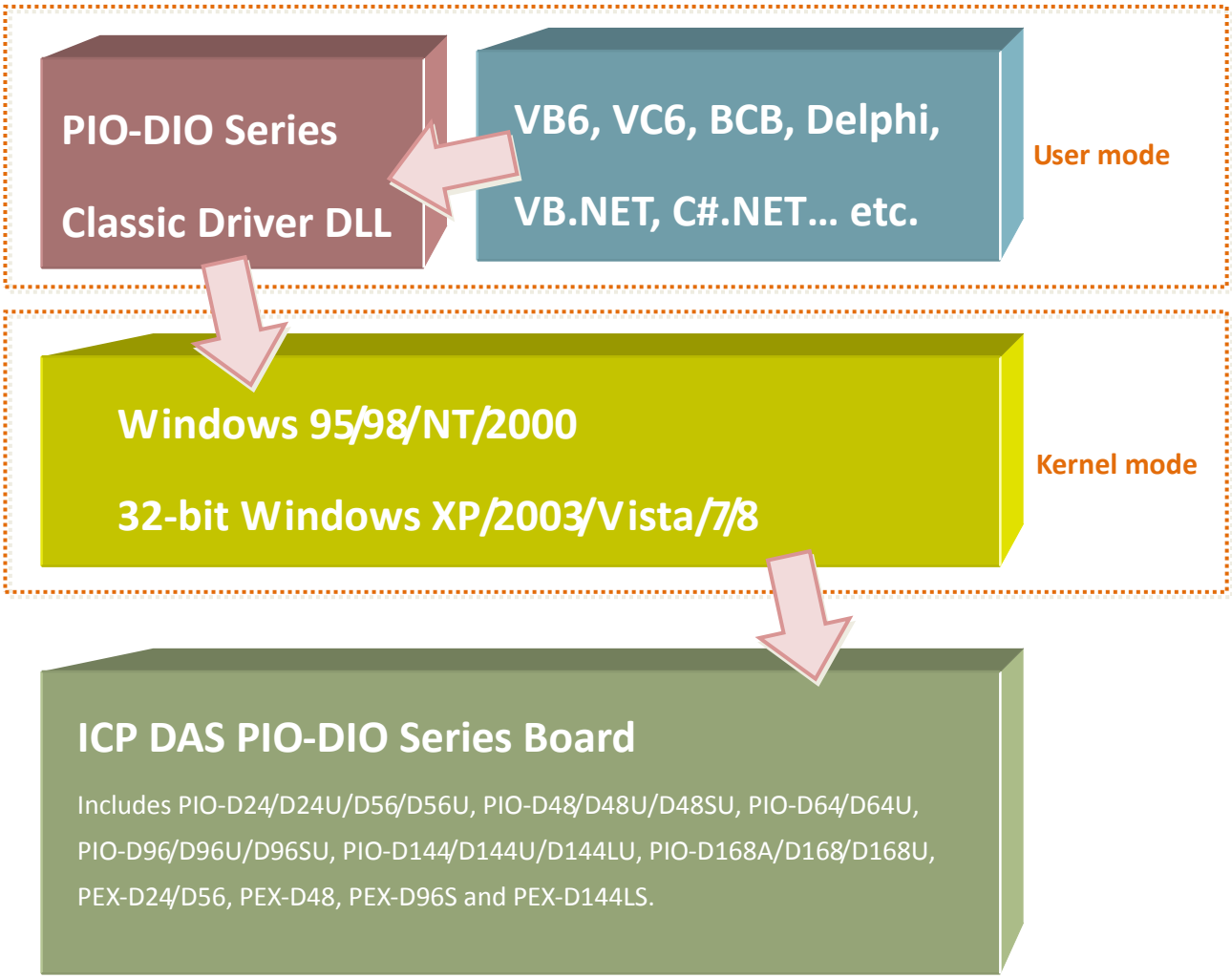
If you have any question, please feel to contact us at:
service@icpdas.com; service.icpdas@gmail.com
We will give you quick response within 2 workdays.

TABLE OF CONTENTS

1. Introduction

The software is a collection of digital I/O subroutines for PIO-DIO series card add-on cards for **Windows 95/98/NT, Windows 2000 and 32-bit Windows XP/2003/Vista/7/8** applications. The application structure is presented in the following diagram.

The subroutines in **PIODIO.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 **VB, VC, Delphi, BCB, VB.NET 2005 and C#.NET 2005** easily. To speed-up your developing process, some demonstration source program are provided.



1.1 Obtaining the Driver Installer Package

PIO-DIO series card can be used on Linux and Windows 95/98/NT/2000 and 32-bit XP/2003/Vista/7/8 based systems, and the drivers are fully Plug & Play (PnP) compliant for easy installation.

The driver installer package for the PIO-DIO series can be found on the supplied CD-ROM, or can be obtained from the ICP DAS FTP web site. The location and addresses are indicated in the table below:

	**CD:\\ NAPDOS\PCI\PIO-DIO\DLL_OCX**
	ftp://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pio-dio/dll_ocx/
	http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pio-dio/dll_ocx/

Install the appropriate driver for your operating system, as follows:

Name	OS
PIO-DIO_Win_xxx.exe	For Windows 95, Windows 98, Windows NT, Windows 2000, 32-bit Windows XP, 32-bit Windows 2003, 32-bit Windows Vista, 32-bit Windows 7 and 32-bit Windows 8 .
lxpio.tar.gz	For Linux Kernel 2.4.x, 2.6.x and 3.12.x. For detail information about Linux software installation, refer to Linux software manual, The download addresses are show below: http://www.icpdas.com/download/pci/linux/

1.2 Driver Installing Procedure

Before the driver installation, you must complete the hardware installation. For detailed information about the hardware installation, please refer to appropriate hardware user manual for your PIO-DIO series card.

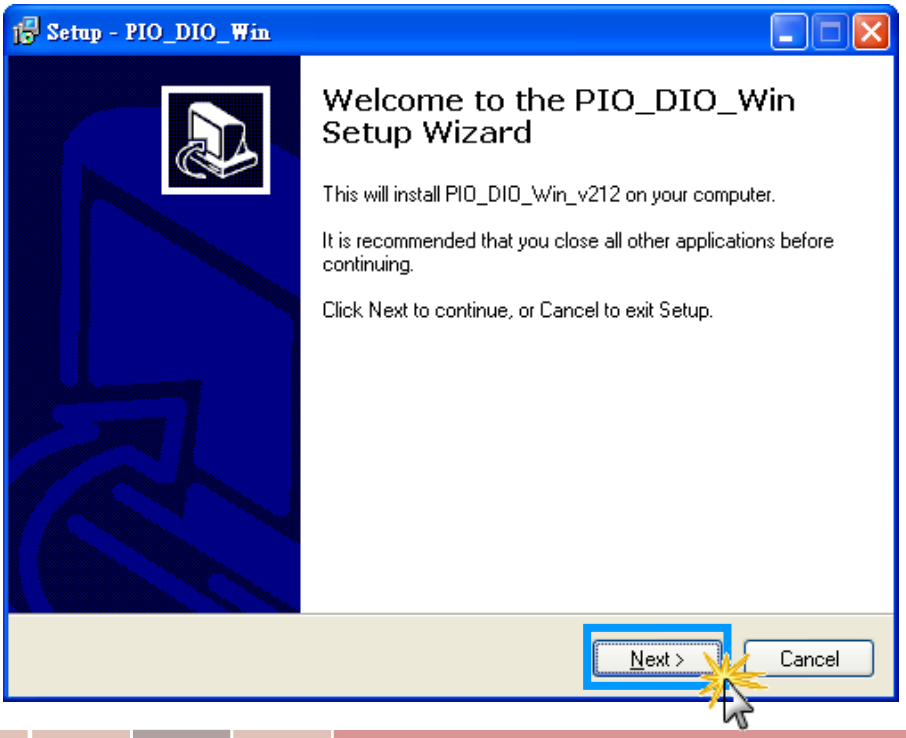
The hardware user manual is contained in:

	CD:\NAPDOS\PCI\PIO-DIO \Manual\
	http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pio-dio/manual/

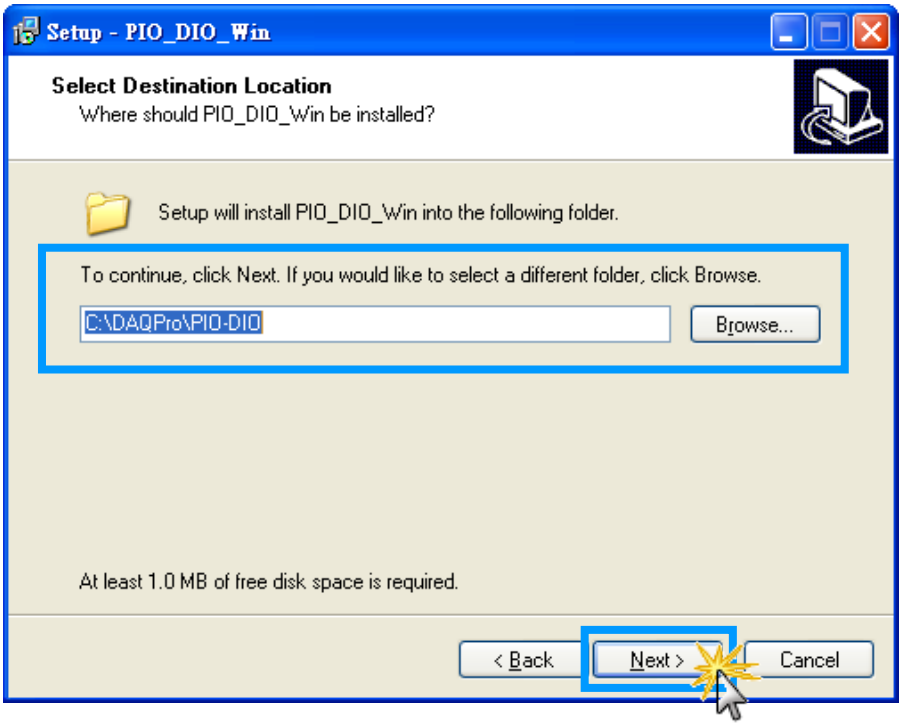
To install the PIO-DIO series classic drivers, follow the procedure described below:

Step 1: Double-Click **"PIO-DIO_Win_xxxx.exe"** to install driver.

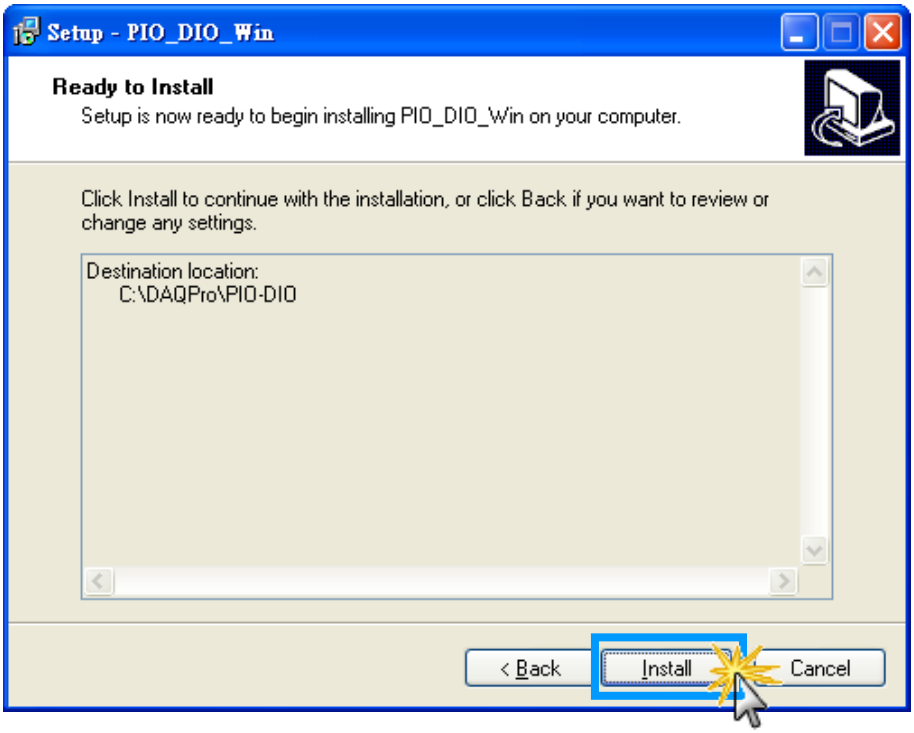
Step 2: Click the **"Next>"** button to start the installation on the **"Setup – PIO_DIO_Win"** window.



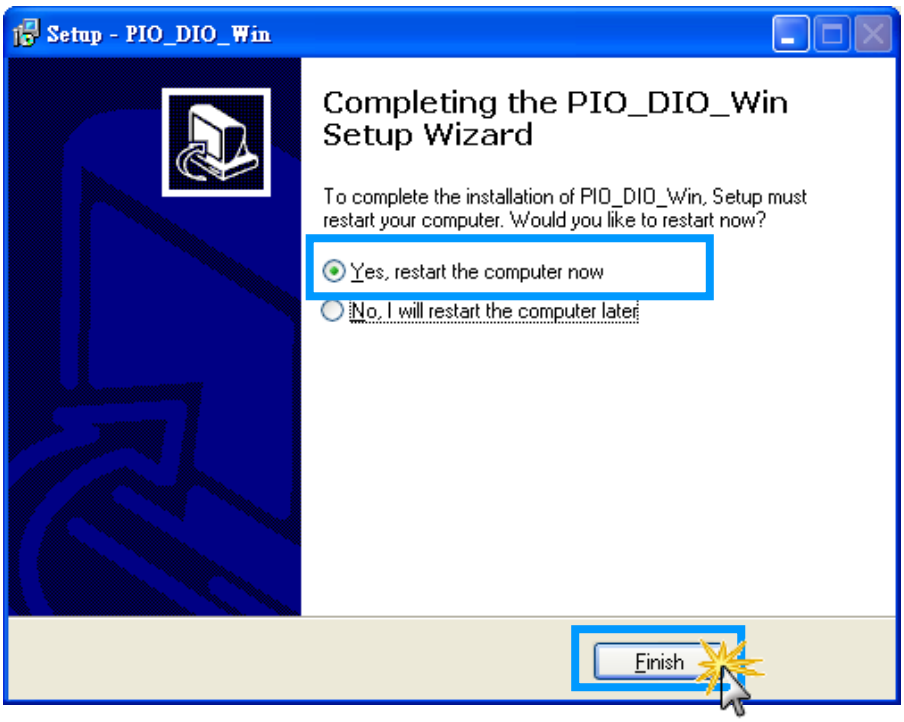
Step 3: Click the **"Next>"** button to install the driver into the **default** folder.



Step 4: Click the **"Install"** button to continue.



Step 5: Selection**"Yes, restart computer now"**and then click the **"Finish"** button.

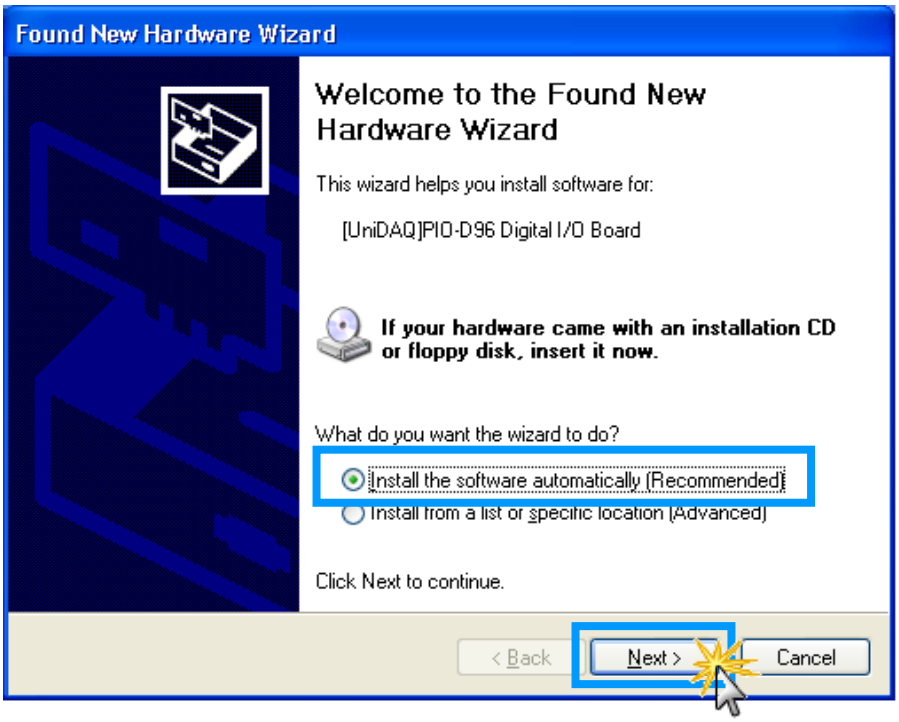


1.3 PnP Driver Installation

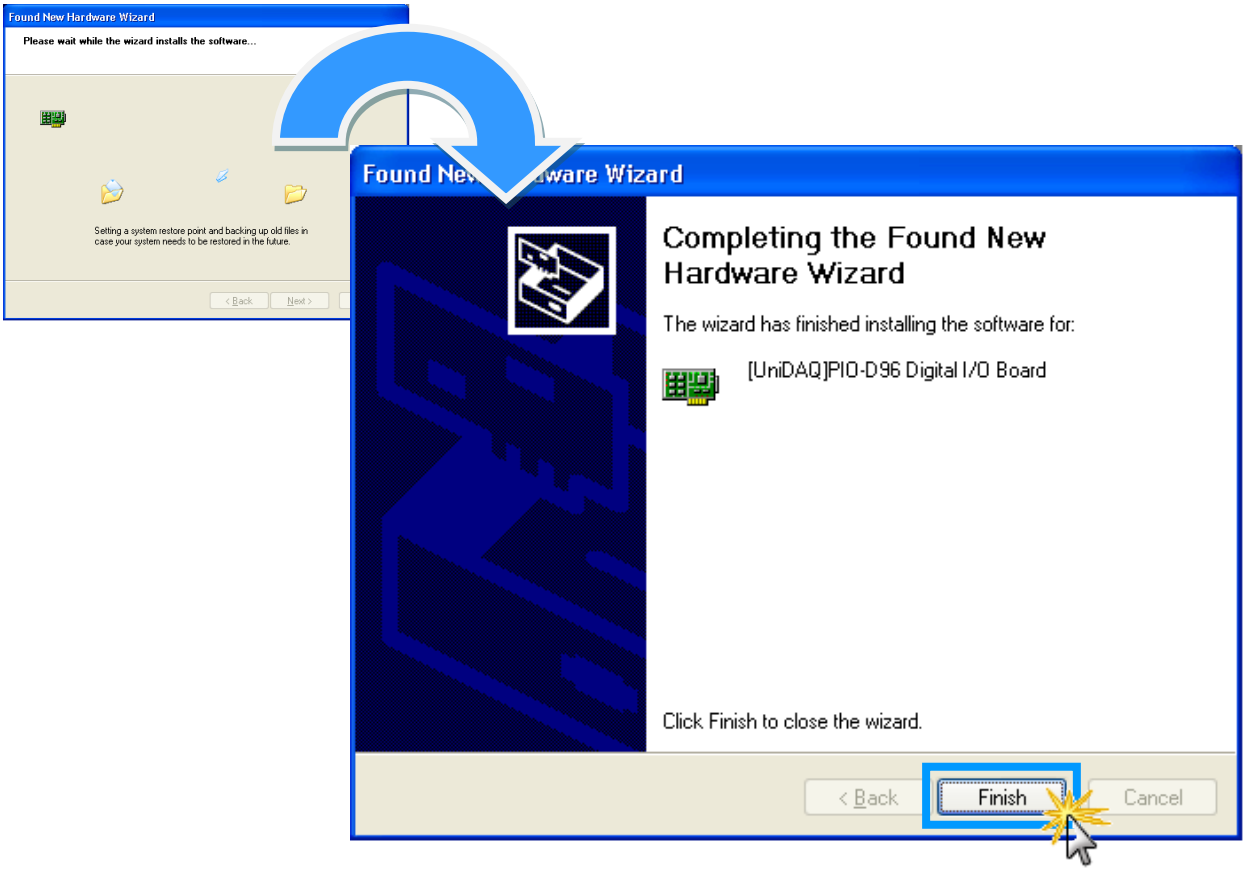
Step 1: The system should find the new card and then continue to finish the Plug&Play steps.

Note: Some operating system (such as Windows Vista/7) will find the new card and make it work automatically, so the Step2 to Step4 will be skipped.

Step 2: Select **"Install the software automatically [Recommended]"** and click the **"Next>"** button.



Step 3: Click the **"Finish"** button.



Step 4: Windows pops up **"Found New Hardware"** dialog box again.

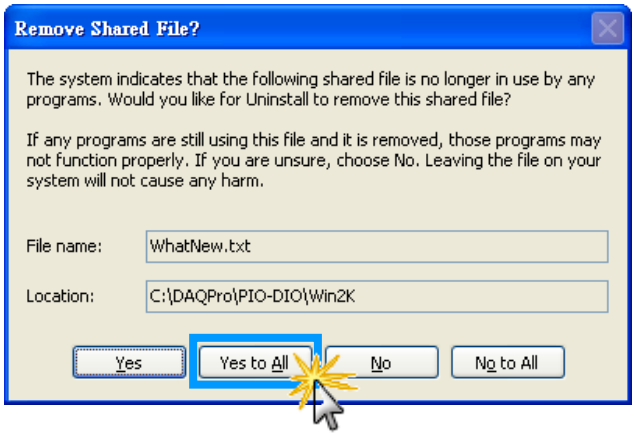
1.4 Uninstalling the PIO-DIO Series Classic Driver

The ICP DAS PIO-DIO series classic driver includes an uninstallation utility that allows you remove the software from your computer. To uninstall the software, follow the procedure described below:

Step 1: Double clock the **unins000.exe** uninstaller application, which can be found in the following folder: **C:\DAQPro\PIO-DIO**.

Step 2: A dialog box will be displayed asking you to confirm that you want to remove the utility program. Click the "**Yes**" button to continue.

Step 3: The **"Remove Shared File?"** dialog box will then be displayed to confirm whether you want to remove the share files. Click the **"Yes to All"** button to continue.



Step 4: After the uninstallation process is complete, a dialog box will be displayed to you that the driver was successfully removed. Click the **"OK"** button to finish the uninstallation process.

2. DLL Function Descriptions

All of the functions provided for PIO-DIO series card are listed below in Tables 2-1 to 2-4. This list of functions is expanded on in the text that follows. However, in order to make a clear and simplified description of the functions, the attributes of the input and output parameters for every function is indicated as [input] and [output] respectively, as shown in following table. Furthermore, the error code of all functions supported by PIO-DIO series card is also listed in Section 2-1.

Keyword	Parameter must be set by the user **before** calling the function	Data/value from this parameter is retrieved **after** calling the function
[Input]	**Yes**	No
[Output]	No	**Yes**
[Input, Output]	**Yes**	**Yes**

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

Table2-1: Test Functions Table

Section	Function Definition
2.3	**Test Functions**
	WORD **PIODIO_GetDllVersion**(void);
	short **PIODIO_ShortSub**(shor **nA**, short **nB**);
	float **PIODIO_FloatSub**(float **fA**, float **fB**);

Table2-2: Driver Relative Functions Table

Section	**Function Definition**
2.4	**Driver Relative Functions**
	WORD **PIODIO_GetDriverVersion**(WORD ***wDriverVersion**);
	WORD **PIODIO_DriverInit**(void);
	WORD **PIODIO_SearchCard**(WORD ***wBoards**, DWORDn **dwPIOCardID**);
	WORD **PIODIO_GetConfigAddressSpace**(WORD **wBoardNo**, DWORD ***wAddrBase**, WORD ***wIrqNo**, WORD ***wSubVendor**, WORD ***wSubDevice**, WORD ***wSubAux**, WORD ***wSlotBus**, WORD ***wSlotDevice**);

	viod **PIODIO_DriverClose**(void);
	WORD **PIODIO_ActiveBoard**(WORD **wBoardNo**);
	WORD **PIODIO_WhichBoardActive**(void);

Table2-3: Digital I/O Functions Table

Section	Function Definition
2.5	**Digital I/O Functions**
	void **PIODIO_OutputByte**(DWORD **wPortAddr**, WORD **bOutputValue**);
	WORD **PIODIO_InputByte**(DWORD **wPortAddr**);
	void **PIODIO_OutputWord**(DWORD **wPortAddress**, DWORD **wOutData**);
	DWORD **PIODIO_InputWord**(DWORD **wPortAddress**);

Table2-4: Interrupt Functions Table

Section	Function Definition
2.6	**Interrupt Functions**
	WORD **PIODIO_IntResetCount**(void);
	WORD **PIODIO_IntGetCount**(DWORD ***dwIntCount**);
	WORD **PIODIO_IntInstall**(WORD **wBoardNo**, HANDLE ***hEvent**, WORD **wInterruptSource**, WORD **wActiveMode**);
	WORD **PIODIO_IntRemove**(void);

Table2-5: PIO-D48 Interrupt Functions Table

Section	Function Definition
2.7	**PIO-D48 Interrupt Functions**
	WORD **PIOD48_IntGetCount**(DWORD ***dwIntCount**);
	WORD **PIOD48_IntInstall**(WORD **wBoardNo**, HANDLE ***hEvent**, WORD **wIrqMask**, WORD **wActiveMode**);
	WORD **PIOD48_IntGetActiveFlag**(WORD ***bActiveHighFlag**, WORD ***bActiveLowFlag**);
	WORD **PIOD48_IntRemove**(void);

Table2-6: PIO-D48 Counter Functions Table

Section	Function Definition
2.8	**PIO-D48 Counter Functions**
	void **PIOD48_SetCounter**(DWORD **dwBase**, WORD **wCounterNo**, WORD **bCounterMode**, DWORD **wCounterValue**);
	DWORD **PIOD48_ReadCounter**(DWORD **dwBase**, WORD **wCounterNo**, WORD **bCounterMode**);
	void **PIOD48_SetCounterA**(WORD **wCounterNo**, WORD **bCounterMode**, DWORD **wCounterValue**);
	DWORD **PIOD48_ReadCounterA**(WORD **wCounterNo**, WORD **bCounterMode**);

Table2-7: PIO-D48 Frequency Functions Table

Section	Function Definition
2.9	**PIO-D48 Frequency Functions**
	DWORD **PIOD48_Freq**(DWORD **dwBase**);
	DWORD **PIOD48_FreqA**();

Table2-8: PIO-D64 Counter Functions Table

Section	Function Definition
2.10	**PIO-D64 Counter Functions**
	void **PIOD64_SetCounter**(DWORD **dwBase**, WORD **wCounterNo**, WORD **bCounterMode**, DWORD **wCounterValue**);
	DWORD **PIOD64_ReadCounter**(DWORD **dwBase**, WORD **wCounterNo**, WORD **bCounterMode**);
	void **PIOD64_SetCounterA**(WORD wCounterNo, WORD bCounterMode, DWORD wCounterValue);
	DWORD **PIOD64_ReadCounterA**(WORD wCounterNo, WORD bCounterMode);

2.1 Error Code Table

For the most errors, it is recommended to check:

1. Does the device driver installs successful?
2. Does the card have plugged?
3. Does the card conflicts with other device?
4. Close other applications to free the system resources.
5. Try to use another slot to plug the card.
6. Restart your system to try again.

Error Code	Error ID	Error String
0	PIODIO_NoError	OK (No Error)
1	PIODIO_DriverOpenError	Device driver can't be opened
2	PIODIO_DriverNoOpen	The PIODIO_DriverInit() function must be called first
3	PIODIO_GetDriverVersionError	Get driver version error
4	PIODIO_InstallIrqError	Install IRQ error
5	PIODIO_ClearIntCountError	Clear counter value error
6	PIODIO_GetIntCountError	Get interrupt counter error
7	PIODIO_RegisterApcError	Get register APC error
8	PIODIO_RemoveIrqError	Remove IRQ error
9	PIODIO_FindBoardError	Cannot find board
10	PIODIO_ExceedBoardNumber	The board number exceeds the maximum board number (7).
11	PIODIO_ResetError	Can't reset the interrupt count
12	PIODIO_IrqMaskError	Irq-Mask is 1,2,4,8 or 1 to 0xF
13	PIODIO_ActiveModeError	Active Mode is 1,2 or 1 to 3
14	PIODIO_GetActiveFlagError	Can't get the interrupt active flag
15	PIODIO_ActiveFlagEndOfQueue	The flag queue is empty

2.2 Sub IDs Table

PIO-DIO Series	Sub_Vendor ID	Sub_Device ID	Sub_AUX ID
PIO-D168	0x9880	0x01	0x50
PIO-D168A	0x80	0x01	0x50
PIO-D168U	0x9880	0x01	0x50
PIO-D144	0x80	0x01	0x00
PIO-D144 (Rev 4.0 or above)	0x5C80	0x01	0x00
PIO-D144U	0x1C80	0x01	0x00
PIO-D144LU	0x1C80	0x01	0x00
PEX-D144LS	0x1C80	0x01	0x00
PIO-D96	0x80	0x01	0x10
PIO-D96 (Rev 4.0 or above)	0x5880	0x01	0x10
PIO-D96U	0x5880	0x01	0x10
PIO-D96SU	0x1880	0x01	0x10
PEX-D96S	0x1880	0x01	0x10
PIO-D64	0x80	0x01	0x20
PIO-D64 (Rev 2.0 or above)	0x4080	0x01	0x20
PIO-D64U	0x4080	0x01	0x20
PIO-D56	0x80	0x01	0x40
PIO-D56 (Rev 5.0 or above)	0x8080, 0xC080	0x01	0x40
PIO-D56U	0x8080, 0xC080	0x01	0x40
PEX-D56	0x8080, 0xC080	0x01	0x40
PIO-D48	0x80	0x01	0x40
PIO-D48U	0x0080	0x01	0x40
PIO-D48SU	0x0080	0x01	0x40
PEX-D48	0x0080	0x01	0x30
PIO-D24	0x80	0x01	0x40
PIO-D24 (Rev 5.0 or above)	0x8080, 0xC080	0x01	0x40
PIO-D24U	0x8080, 0xC080	0x01	0x40
PEX-D24	0x8080, 0xC080	0x01	0x40

2.3 Test Functions

PIODIO_GetDllVersion

This function is used to retrieve the version number of the PIODIO.DLL.

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

- **Parameters:**
  None

- **Returns:**
  DLL version information.
  For example: If 200(hex) value is return, it means driver version is 2.00.

PIODIO_ShortSub

This function is used to perform the subtraction (as **nA - nB** in short data type), and is provided for testing DLL linkage purposes.

- **Syntax:**
  short **PIODIO_ShortSub**(short **nA**, short **nB**);

- **Parameters:**

  nA
  [Input] 2 bytes short data type value

  nB
  [Input] 2 bytes short data type value

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

PIODIO_FloatSub

This function is used to perform the subtraction (as **fA - fB** in float data type), and is provided for testing DLL linkage purpose.

- **Syntax:**

  float **PIODIO_FloatSub**(float **fA**, float **fB**);

- **Parameters:**

  fA

  [Input] 4 bytes floating point value

  fB

  [Input] 4 bytes floating point value

- **Returns:**

  The value of fA - fB

2.4 Driver Relative Functions

PIODIO_GetDriverVersion

This function is used to read the version number information from the PIODIO driver.

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

- **Parameters:**

  wDriverVersion
  [Output] wDriverVersion address

- **Returns:**
  Refer to "Section 2.1 Error Code Table"

PIODIO_DriverInit

This function is used to open the PIODIO driver and allocate the computer resource for the device. This function must be called once before applying other PIODIO functions.

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

- **Parameters:**

  None

- **Returns:**
  Refer to "Section 2.1 Error Code Table"

PIODIO_SearchCard

This function can be used to search for the installed card and determine total number of boards. This function must be called once before applying other PIODIO functions.

- **Syntax:**

  WORD **PIODIO_SearchCard**(WORD ***wBoards**, DWORDn **dwPIOCardID**);

- **Parameters:**

  wBoardNo

  [Output] Determine the total number of boards.

  DwPIOCardID

  [Input] Sub IDs of the PIODIO card. Refer to Section 2.2 Sub IDs Table for more details.

  Note:

  Different versions of the PIO-DIO series cards may have different Sub IDs. This function will determine the total number of PIO-DIO series cards including all versions; no matter what version Sub ID is input.

  For example:

  wRtn=PIODIO_SearchCard(&wBoards, 0x800100);

  Will determine the total number of PIO-D144 cards installed in the PC regardless of version.

- **Returns:**

  Refer to "Section 2.1 Error Code Table"

PIODIO_GetConfigAddressSpace

This function is used to obtain the I/O address and other information for the PIO-DIO series cards.

- **Syntax:**

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

- **Parameters:**

wBoardNo
[Input] PIO-DIO series card number.

wAddrBase
[Output] The bases address of PIO-DIO series cards. Only the low WORD is valid.

wIrqNo
[Output] The IRQ number that is being used by the PISO-DIO series cards using.

wSubVendor
[Output] Sub Vendor ID.

wSubDevice
[Output] Sub Device ID.

wSubAux
[Output] Sub Aux ID.

wSlotBus

[Output] Slot Bus number.

wSlotDevice

[Output] Sub Device ID.

- **Returns:**

  Refer to "Section 2.1 Error Code Table"

PIODIO_DriverClose

This function is used to close the PIODIO Driver and release the device resources from the computer. This function must be called once before exiting the user's application.

- **Syntax:**

  void **PIODIO_DriverClose**();

- **Parameters:**

  None

- **Returns:**

  None

PIODIO_ActiveBoard

This function is used to active one of the PIO-DIO boards installed in the system. This function must call once before the digital input, digital output and interrupt functions are called.

- **Syntax:**
  void **PIODIO_ActiveBoard**(WORD **wBoardNo**);
- **Parameters:**

  wBoardNo
  [Input] The board numbers to active.
- **Returns:**
  Refer to "Section 2.1 Error Code Table"

PIODIO_WhichBoardActive

This function is used to return the board number of the active board.

- **Syntax:**
  WORD **PIODIO_WhichBoardActive**(void);
- **Parameters:**

  None
- **Returns:**
  Return the board number of the active board.

2.5 Digital I/O Functions

PIODIO_OutputByte

This function is used to send 8 bits of data to the specified I/O port.

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

- **Parameters:**

  wPortAddr
  [Input] I/O port addresses. Refer to the PIODIO_GetConfigAddressSpace() function.
  Only the low WORD is valid.

  bOutputVal
  [Input] 8 bits of data sent to the I/O port. Only the low BYTE is valid.

- **Returns:**
  None

PIODIO_InputByte

This function is used to read 8 bits of data from the specified I/O port.

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

- **Parameters:**

  wPortAddr
  [Input] I/O port addresses. Refer to the PIODIO_GetConfigAddressSpace() function.
  Only the low WORD is valid.

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

PIODIO_OutputWord

This function is used to send 16 bits of data to the specified I/O port.

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

- **Parameters:**

  wPortAddr
  [Input] I/O port addresses. Refer to PIODIO_GetConfigAddressSpace() function.
  Only the low WORD is valid.

  wOutputVal
  [Input] 16 bit data sent to the I/O port. Only the low WORD is valid.

- **Returns:**
  None

PIODIO_InputWord

This function is used to read 16 bits of data from the specified I/O port.

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

- **Parameters:**

  wPortAddr
  [Input] I/O port addresses. Refer to PIODIO_GetConfigAddressSpace() function.
  Only the low WORD is valid.

- **Returns:**
  16 bits of data. Only the low WORD is valid.

2.6 Interrupt Functions

PIODIO_IntResetCount

This function is used to clear the counter value of the device driver for the interrupt.

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

- **Parameters:**

  None

- **Returns:**
  Refer to "Section 2.1 Error Code Table"

PIODIO_IntGetCount

This function is used to read the **dwIntCount** value defined in the device driver.

- **Syntax:**
  WORD **PIODIO_IntGetCount**(WORD ***dwIntCount**);

- **Parameters:**

  **dwIntCount*
  [Output] Address of dwIntCount, which is used of store the value of the interrupt counter.

- **Returns:**
  Refer to "Section 2.1 Error Code Table"

PIODIO_IntInstall

This function is used to install the IRQ service routine.

- **Syntax:**

WORD **PIODIO_IntInstall**(WORD **wBoardNo,**
HANDLE ***hEvent,**
WORD **wInterruptSource,**
WORD **wActiveMode**);

- **Parameters:**

wBoardNo

[Input] The board to be used.

hEvent

[Input] Address of an Event handle. The user's program must call the Windows API function "Create Event()" to create an event object.

wInterruptSource

[Input] The Interrupt Source to be used. Refer to hardware's manual of PIO-DIO series for the detail information.

Model	wInterruptSource	Description
PIO-D48 series	0	PC3/PC7 from Port-2
	1	PC3/PC7 from Port-5
	2	Cout0
	3	Cout2
PIO-D56/D24 series	0	PC0
	1	PC1
	2	PC2
	3	PC3
PIO-D64 series	0	EXTIRQ
	1	EVTIRQ
	2	TMRIRQ
PIO-D96 series	0	P2C0
	1	P5C0
	2	P8C0
	3	P11C0

Model	wInterruptSource	Description
PIO-D144/D168 series	0	P2C0
	1	P2C1
	2	P2C2
	3	P2C3

wActiveMode

[Input] The mode for triggering the interrupt.

wActiveMode	Description
0	PIODIO_ActiveLow
1	PIODIO_ActiveHigh

- **Returns:**

  Refer to "Section 2.1 Error Code Table"

PIODIO_IntRemove

This function is used to remove the IRQ service routine.

- **Syntax:**

  WORD **PIODIO_IntRemove**(void);

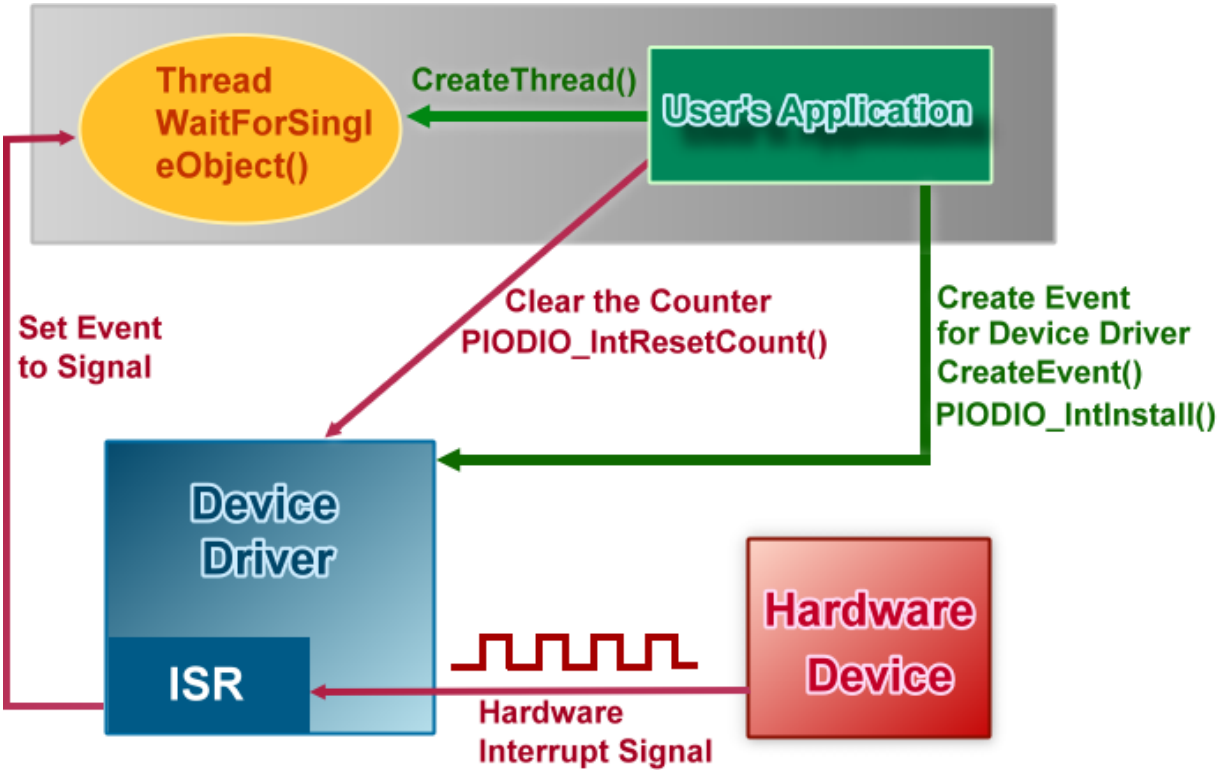
- **Parameters:**

  None

- **Returns:**

  Refer to "Section 2.1 Error Code Table"

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
);
```

2.7 PIO-D48 Interrupt Functions

The following PIOD48_XXX series function is designed for PIO-D48 series card only. They cannot be used with other cards.

The most different between the PIO-DIO and PIO-D48 interrupt functions is the PIO-DIO supports only one interrupt-source at a time and the PIO-D48 supports 4 interrupt-source at a time.

PIOD48_IntGetCount

This subroutine will read the **Interrupt-Counter** value in the device driver. The Interrupt-Counter will be increased (in the ISR) when the interrupt is triggered. When the interrupt setting to Active-High only or Active-Low only, some of the interrupt signal will be ignored and the Interrupt-Counter will not increase.

- **Syntax:**
  WORD **PIOD48_IntGetCount**(DWORD ***dwIntCount**);

- **Parameters:**

  **dwIntCount*
  [Output] Address of dwIntCount, which is used of store the value of the interrupt counter.

- **Returns:**
  Refer to "Section 2.1 Error Code Table"

PIOD48_IntInstall

This subroutine will install the IRQ service routine. This function supports multiple interrupt-source and the Active-Mode can setting to "Active-Low only", "Active-High only" and "Active-Low or Active-High".

- **Syntax:**

WORD **PIOD48_IntInstall**(WORD **wBoardNo,**
HANDLE ***hEvent,**
WORD **wIrqMask,**
WORD **wActiveMode**);

- **Parameters:**

wBoardNo

[Input] The board to be used.

hEvent

[Input] Address of an Event handle. The user's program must call the Windows API function "Create Event()" to create an event object.

wIrqMask

[Input] The IRQ Mask to be used. Refer to hardware's manual of PIO-D48 series for the detail information.

wIrqMask	Description
1	NT_CHAN_0: PC3/PC7 from Port-2
2	INT_CHAN_1: PC3/PC7 from Port-5
4	INT_CHAN_2: Cout0
8	INT_CHAN_3: Cout2

This function supports 4 interrupt-source at a time, thus users can use multiple interrupt-source like 1 +2 +4 +8.

wActiveMode

[Input] When the ISR will service the interrupt ?

wActiveMode	Description
1	PIOD48_ActiveLow The interrupt is occurred when the Interrupt-Source status is low.
2	PIOD48_ActiveHigh The interrupt is occurred when the Interrupt-Source status is high.

This can be 1 (Active-High), 2(Active-Low) or 1 + 2 (Both of the High and Low will active the interrupt).

- **Returns:**

Refer to "Section 2.1 Error Code Table"

PIOD48_IntGetActiveFlag

This subroutine will read the Active-High and Active-Low flag from the device driver's memory queues (First-in-First-out, Buffer Size: 2000 flags for High/Low).

The Active-Flag is used to records the Active-State-change of interrupt-source when the interrupt occurred. The Active-High-Flag records which interrupt-source changed to high state and the Active-Low-Flag records which interrupt-source changed to low state. Users can uses these flags to indicate which interrupt-source has changed.

If the Active-Mode is set to Active-Low(/Active-High) only and the state for the Active-Low(/Active-High) is equal to zero, then the ISR will not increased the interrupt-counter, and the Active-Flag for High and Low will not recorded.

If users have not calling this function to retrieve the flags from device driver's memory queues, these queues will stop record the flags (lost data) while the buffer is full. But the interrupt-counter will still counting while the ISR services the interrupt.

- **Syntax:**

  WORD **PIOD48_IntGetActiveFlag**(WORD ***bActiveHighFlag,** WORD ***bActiveLowFlag**);

- **Parameters:**

  bActiveHighFlag

  [Output] Returns a flag that indicates which interrupt-source changed to High-State.

  bActiveLowFlag

  [Output] Returns a flag that indicates which interrupt-source changed to Low-State.

- **Returns:**

  Refer to "Section 2.1 Error Code Table"

PIOD48_IntRemove

This function is used to remove the IRQ service routine.

- **Syntax:**

  WORD **PIOD48_IntRemove**(void);

- **Parameters:**

  None

- **Returns:**

  Refer to "Section 2.1 Error Code Table"

2.8 PIO-D48 Counter Functions

The following PIOD48_XXX series function is designed for PIO-D48 series card only.

PIOD48_SetCounter

This subroutine is used to set the 8254 counter's mode and value.

- **Syntax:**

  WORD **PIOD48_SetCounter**(WORD **dwBase,**
  WORD **wCounterNo**,
  WORD **bCounterMode**,
  DWORD **wCounterValue**);

- **Parameters:**

  dwBase
  [Input] I/O port addresses. Refer to the PIODIO_GetConfigAddressSpace() function.
  Only the low WORD is valid.

  wCounterNo
  [Input] The 8254 Counter-Number: 0 to 2.

  wCounterMode
  [Input] The 8254 Counter-Mode: 0 to 5. Refer to the hardware's manual of PIO-D48 series card for details.

  wCounterValue
  [Input] The 16 bits value for the timer/counter to count. Only the low WORD is valid.

- **Returns:**

  None

PIOD48_ReadCounter

This subroutine is used to read the 8254 counter's value.

- **Syntax:**

  WORD **PIOD48_ReadCounter**(WORD **dwBase,**
  WORD **wCounterNo**,
  WORD **bCounterMode**);

- **Parameters:**

  dwBase
  [Input] I/O port addresses. Refer to the PIODIO_GetConfigAddressSpace() function.
  Only the low WORD is valid.

  wCounterNo
  [Input] The 8254 Counter-Number: 0 to 2.

  wCounterMode
  [Input] The 8254 Counter-Mode: 0 to 5. Refer to the hardware's manual of PIO-D48 series card for details.

- **Returns:**

  Returns the 16 bits counter value. (Only the low WORD is valid.)

PIOD48_SetCounterA

This subroutine is used to set the 8254 counter's mode and value. Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax:**

  WORD **PIOD48_SetCounterA**(WORD **wCounterNo,**
  WORD **bCounterMode**,
  WORD **wCounterValue**);

- **Parameters:**

  wCounterNo

  [Input] The 8254 Counter-Number: 0 to 2.

  wCounterMode

  [Input] The 8254 Counter-Mode: 0 to 5. Refer to the hardware's manual of PIO-D48 series card for details.

  wCounterValue

  [Input] The 16 bits value for the counter to count. Only the low WORD is valid.

- **Returns:**

  None

PIOD48_ReadCounterA

This subroutine is used to read the 8254 counter's value. Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax:**

  WORD **PIOD48_ReadCounterA**(WORD **wCounterNo,**
  WORD **bCounterMode**);

- **Parameters:**

  wCounterNo
  [Input] The 8254 Counter-Number: 0 to 2.

  wCounterMode
  [Input] The 8254 Counter-Mode: 0 to 5. Refer to the hardware's manual of PIO-D48 series card for details.

- **Returns:**

  Returns the 16 bits counter value. (Only the low WORD is valid.)

2.9 PIO-D48 Frequency Functions

The following PIOD48_XXX series function is designed for PIO-D48 series card only.

PIOD48_Freq

This subroutine is used to measurement the signal frequency. Users have to connect the signal(+) with CN1.Pin29, and connect the signal(-) with CN1.Pin19.
It will uses the Counter-0 and Counter-1 to measure the frequency, thus users shouldn't use Counter-0 and Counter-1 for other purposes.

- **Syntax:**
  WORD **PIOD48_Freq**(WORD **dwBase**);

- **Parameters:**

  dwBase
  [Input] I/O port addresses. Refer to the PIODIO_GetConfigAddressSpace() function.
  Only the low WORD is valid.

- **Returns:**
  Return the frequency value. (Only the low WORD is valid.)

PIOD48_FreqA

Please refer to the description of "PIOD48_Freq()" function. Users have to calling the "PIODIO_ActiveBoard()" function before calling this function.

- **Syntax:**

  WORD **PIOD48_FreqA**();

- **Parameters:**

  None

- **Returns:**

  Return the frequency value. (Only the low WORD is valid.)

2.10 PIO-D64 Counter Functions

The following PIOD64_XXX series function is designed for PIO-D64 series card only.

PIOD64_SetCounter

This subroutine is used to set the 8254 counter's mode and value.

- **Syntax:**

WORD **PIOD64_SetCounter**(WORD **dwBase,**
WORD **wCounterNo**,
WORD **bCounterMode**,
DWORD **wCounterValue**);

- **Parameters:**

dwBase
[Input] I/O port addresses. Refer to the PIODIO_GetConfigAddressSpace() function.
Only the low WORD is valid.

wCounterNo
[Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)

wCounterMode
[Input] The 8254 Counter-Mode: 0 to 5. Refer to the hardware's manual of PIO-D64 series card for details.

wCounterValue
[Input] The 16 bits value for the counter to count. Only the low WORD is valid.

- **Returns:**

None

PIOD64_ReadCounter

This subroutine is used to read the 8254 counter's value.

- **Syntax:**

  WORD **PIOD64_ReadCounter**(WORD **dwBase,**
  WORD **wCounterNo**,
  WORD **bCounterMode**);

- **Parameters:**

  dwBase
  [Input] I/O port addresses. Refer to the PIODIO_GetConfigAddressSpace() function.
  Only the low WORD is valid.

  wCounterNo
  [Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)

  wCounterMode
  [Input] The 8254 Counter-Mode: 0 to 5. Refer to the hardware's manual of PIO-D64 series card for details.

- **Returns:**

  Returns the 16 bits counter value. (Only the low WORD is valid.)

PIOD64_SetCounterA

This subroutine is used to set the 8254 counter's mode and value. Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax:**

  WORD **PIOD64_SetCounterA**(WORD **wCounterNo,**
  WORD **bCounterMode**,
  WORD **wCounterValue**);

- **Parameters:**

  wCounterNo

  [Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)

  wCounterMode

  [Input] The 8254 Counter-Mode: 0 to 5. Refer to the hardware's manual of PIO-D64 series card for details.

  wCounterValue

  [Input] The 16 bits value for the counter to count. Only the low WORD is valid.

- **Returns:**

  None

PIOD64_ReadCounterA

This subroutine is used to read the 8254 counter's value. Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax:**

  WORD **PIOD64_ReadCounterA**(WORD **wCounterNo,**
  WORD **bCounterMode**);

- **Parameters:**

  wCounterNo

  [Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)

  wCounterMode

  [Input] The 8254 Counter-Mode: 0 to 5. Refer to the hardware's manual of PIO-D64 series card for details.

- **Returns:**

  Returns the 16 bits counter value. (Only the low WORD is valid.)

3. DOS Lib Function

3.1 Error Code Table

Error Code	Error ID	Error String
0	NoError	OK (No Error)
1	DriverHandleError	Device driver opened error
2	DriverCallError	Got the error while calling the drier functions
3	FindBoardError	Can’t find the board on the system
4	TimeOut	Timeout
5	ExceedBoardNumber	Invalidate board number (Valid range: 0 to TotalBoard -1)
6	NotFoundBoard	Can’t detect the board on the system

PIO_DriverInit

This function can detect all PIO/PISO series cards in the system. It is implemented based on the PCI Plug & Play mechanism-1. It will find all PIO/PISO series cards installed in this system and save all their resources in the library.

- **Syntax:**

  WORD **PIO_DriverInit**(WORD ***wBoards**,
  WORD **wSubVendorID**,
  WORD **wSubDeviceID**,
  WORD **wSubAuxID**);

- **Parameters:**

  wBoards
  [Output] Number of boards found in the PC.

  wSubVendorID
  [Input] SubVendor ID of the PIO/PISO series board.

  wSubDeviceID
  [Input] SubDevice ID of the PIO/PISO series board.

  wSubAuxID
  [Input] SubAux ID of the PIO/PISO series board.

- **Returns:**

  Refer to "Section 3.1 Error Code Table"

PIO_GetDriverVersion

This subroutine will obtain the version number of PIO/PISO driver.

- **Syntax:**

  WORD **PIO_GetDriverVersion**(WORD ***wDriverVersion**);

- **Parameters:**

  **wDriverVersion*

  [Output] wDriverVersion address.

- **Returns:**

  Refer to "Section 3.1 Error Code Table"

PIO_GetConfigAddressSpace

The user can use this function to save the resources found on all the PIO/PISO cards installed on the system. Then the application program can control all the functions of PIO/PISO series cards directly.

- **Syntax:**

```
WORD PIO_GetConfigAddressSpace(wBoardNo,
                               *wBase,
                               *wIrq,
                               wSubVendor,
                               *wSubDevice,
                               *wSubAux,
                               *wSlotBus,
                               *wSlotDevice);
```

- **Parameters:**

wBoardNo
[Input] Number of boards found in the PC.

**wBase*
[Output] The base address of the PIO/PISO series board.

**wIrq*
[Output] The IRQ number that the PIO/PISO using.

wSubVendor
[Output] SubVendor ID of the PIO/PISO series board.

**wSubDevice*
[Output] SubDevice ID of the PIO/PISO series board.

**wSubAux*
[Output] SubAux ID of the PIO/PISO series board.

**wSlotBus*
[Output] Slot Bus number.

**wSlotDevice*
[Output] Slot Device ID

- **Returns:**

Refer to "Section 3.1 Error Code Table"

ShowPIOPISO

This function will show a text string for a special Sub_ID. This text string is the same as that defined in PIO.H.

- **Syntax:**

  WORD **ShowPIOPISO**(**wSubVendor**, **wSubDevice**, **wSubAux**);

- **Parameters:**

  wSubVendor

  [Input] SubVendor ID of the board

  wSubDevice

  [Input] SubDevice ID of the board

  wSubAux

  [Input] SubAux ID of the board

- **Returns:**

  Refer to "Section 3.1 Error Code Table"

4. Demo Programs

4.1 For Microsoft Windows

ICP DAS PIO-DIO Series Classic Driver DLL contains a set of functions. It can be used in various application programs for PIO-DIO series card. The API functions supports many development environments and programming languages, including Microsoft Visual C++，Visual Basic，Borland Delphi，Borland C Builder++，Microsoft Visual C#.NET，Microsoft Visual VB.NET.

The demo programs of Windows OS for the PIO-DIO series can be found on the supplied CD-ROM, or can be obtained from the ICP DAS FTP web site. The location and addresses are indicated in the table below:

	CD:\NAPDOS\PCI\PIO-DIO\DLL_OCX\Demo\
	http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pio-dio/dll_ocx/demo/

- BCB4 → for Borland C++ Builder 4
  PIODIO.H → Header files
  PIODIO.LIB → Linkage library for BCB only
- Delphi4 → for Delphi 4
  PIODIO.PAS → Declaration files
- VC6 → for Visual C++ 6
  PIODIO.H → Header files
  PIODIO.LIB → Linkage library for VC only
- VB6 → for Visual Basic 6
  PIODIO.BAS → Declaration files
- VB.NET2005 → for VB.NET2005
  PIODIO.vb → Visual Basic Source files
- CSharp2005 → for C#.NET2005
  PIODIO.cs → Visual C# Source files

Select the appropriate demo for your PIO-DIO series card, as follows:

Folder	The list of demo programs
D24	For PIO-D24/D24U, PEX-D24 DIO Demo Int Demo IntAPC demo
D56	For PIO-D56/D56U, PEX-D56 DIO_1 Demo DIO_2 Demo Int Demo IntAPC Demo
D48	For PIO-D48/D48U/D48SU, PEX-D48 DIO Demo Freq Demo Int Demo Int1APC Demo Int2 Demo Int2APC Demo Int3 Int3APC Demo Int4 Int4APC Demo Read Counter Demo
D64	For PIO-D64/D64U DIO Demo Int Demo IntAPC Demo Counter Demo 32bitCounter Demo
D96	For PIO-D96/D96U/D96SU, PEX-D96S DIO Demo Int Demo IntAPC Demo

D144	For PIO-D144/D144U/D144LU, PEX-D144S - DIO Demo - DIO2 Demo - DO Demo - Int Demo - IntAPC Demo
D168	For PIO-D168/D168A/D168U - DIO Demo - DIO2 Demo - DO Demo - Int Demo - IntAPC Demo

4.2 For DOS

The demo program is contained in:

	CD:\NAPDOS\PCI\PIO-DIO\DOS\
	http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pio-dio/dos/

- \TC*.* → for Turbo C 2.xx or above
- \MSC*.* → for MSC 5.xx or above
- \BC*.* → for BC 3.xx or above

- \TC\LIB*.* → for TC Library
- \TC\DEMO*.* → for TC demo program
- \TC\DIAG*.* → for TC diagnostic program

- \TC\LIB\Large\PIO.H → TC Declaration File
- \TC\LIB\Large\TCPIO_L.LIB → TC Large Model Library File
- \TC\LIB\Huge\TCPIO_H.LIB → TC Huge Model Library File

- \MSC\LIB\Large\PIO.H → MSC Declaration File
- \MSC\LIB\Large\MSCPIO_L.LIB → MSC Large Model Library File
- \MSC\LIB\Huge\MSCPIO_H.LIB → MSC Huge Model Library File

- \BC\LIB\Large\PIO.H → BC Declaration File
- \BC\LIB\Large\BCPIO_L.LIB → BC Large Model Library File
- \BC\LIB\Huge\BCPIO_H.LIB → BC Huge Model Library File

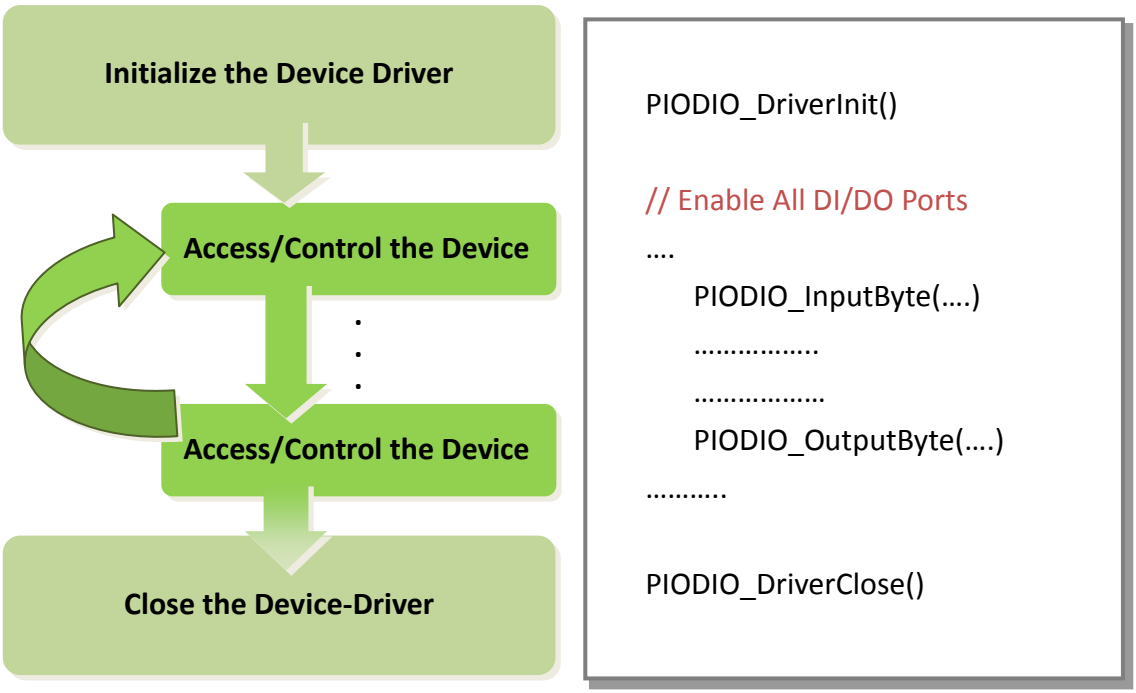
Select the appropriate demo for your PIO-DIO series card, as follows:

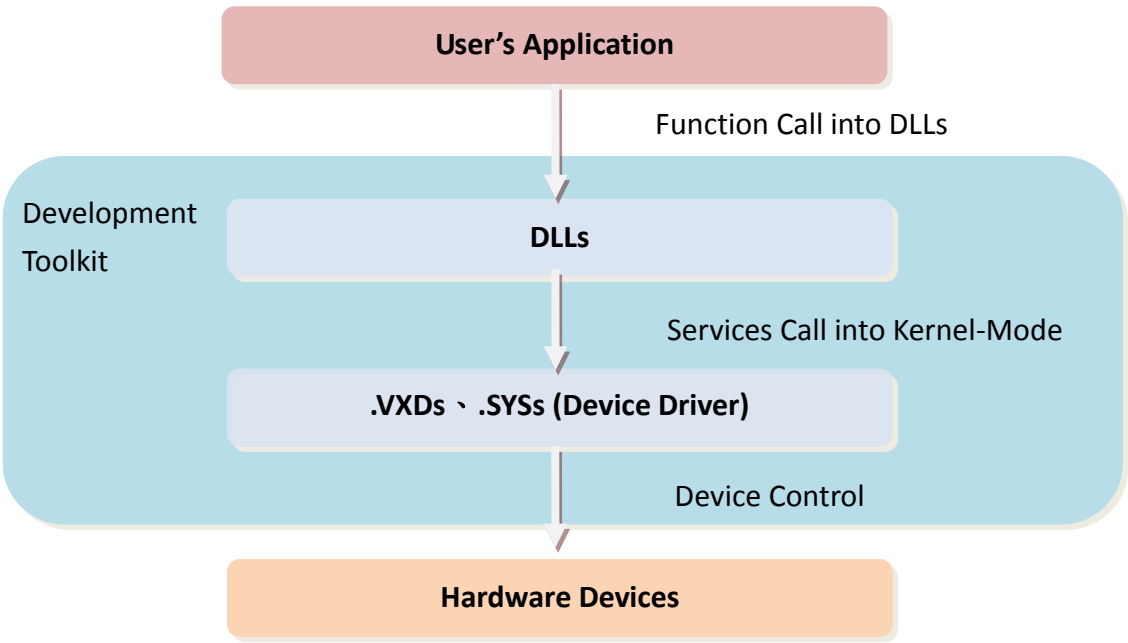
Folder	The list of demo programs
diag	For all PISO/DIO series card ⊕ PIO_PISO.exe
D2456	For PIO-D24/D24U/D56/D56U, PEX-D24/D56 ⊕ Demo1: DO demo of CON3 ⊕ Demo2: DI/O demo of CON1, CON2 and CON3 ⊕ Demo3: Count high pulse of PC0 (Initial low & active high) ⊕ Demo4: Count high pulse of PC0 (Initial high & active low) ⊕ Demo5: Four Interrupt Source
D48	For PIO-D48/D48U/D48SU, PEX-D48 ⊕ Demo1: DO demo of CN1 and CN2 ⊕ Demo2: DI demo of CN1 and CN2 ⊕ Demo3: DI/O demo of CN1 and CN2 ⊕ Demo4: INT_CHAN_3, timer interrupt ⊕ Demo5: INT_CHAN_2, 16-bit event counter (no interrupt), init_HIGH & active_LOW signal to PC0 of port-2. ⊕ Demo6: INT_CHAN_2, 16-bit event counter (no interrupt), init_LOW & active_HIGH signal to PC0 of port-2. ⊕ Demo7: INT_CHAN_2, 16-bit down-counter (using interrupt), init_HIGH & active_LOW signal to PC3 of port-2. (Note: The PC7 of port_2 is used to enable the interrupt) ⊕ Demo8: INT_CHAN_0, interrupt demo, init_HIGH & active_LOW signal to PC3 of port-2. (Note: The PC7 of port_2 is don't case) ⊕ Demo9: INT_CHAN_0, interrupt demo, init_HIGH & active_LOW signal to PC3 of port-2. (Note: The PC7 of port_2 is used to enable the interrupt) ⊕ Demo10: INT_CHAN_1, interrupt demo, init_HIGH & active_LOW signal to PC3 of port-5. (Note: The PC7 of port_5 is used to enable the interrupt) ⊕ Demo11: INT_CHAN_0 & INT_CHAN_1, interrupt demo, init_HIGH & active_LOW signal to PC3 of port-2 (port-5). (Note: The PC7 of port-2(port_5) is don't care)

D64	For PIO-D64/D64U ⊕ Demo1: DO demo ⊕ Demo2: DI/O demo ⊕ Demo3: Use external int. to measure pulse width (high level) ⊕ Demo4: Use EVTIRQ to count event ⊕ Demo5: Use TMRIRQ to generate 0.5 Hz squa. ⊕ Demo6: Use TMRIRQ to generate 0.5 Hz squa. EVTIRQ to count
D96	For PIO-D96/D96U/D96SU, PEX-D96S ⊕ Demo1: DO demo of CN1 ⊕ Demo2: DI/O demo of CN2 and CN3 ⊕ Demo3: Count high pulse of P2C0 (initial Low & active High) ⊕ Demo4: Count high pulse of P2C0 (initial High & active Low) ⊕ Demo5: Four Interrupt Source
D144	For PIO-D144/D144U/D144LU, PEX-D144S ⊕ Demo1: DO of CN1 ⊕ Demo2: DO of CN1 to CN6 ⊕ Demo3: Interrupt of P2C0 (Initial low & active high) ⊕ Demo4: Interrupt of P2C0 (Initial high & active low) ⊕ Demo5: 4 Interrupt sources ⊕ Demo6: DO demo ⊕ Demo10: Find card number
D168	For PIO-D168/D168A/D168U ⊕ Demo1: DO of CN1 ⊕ Demo2: DO of the CN1 to CN6 ⊕ Demo3: Interrupt of P2C0 (Initial low & active high) ⊕ Demo4: Interrupt of P2C0 (Initial high & active low) ⊕ Demo5: 4 Interrupt sources

Note that all of the hardware control functions need to be provided and processed by user themselves.

5. Programs 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 Service@icpdas.com or Service.icpdas@ gmail.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 98, Windows 2000 or 32-bit Windows XP/2003/Vista/7/8.
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 test 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 you comments? And please keeps contact with us.