

PISO-730 Series Card User Manual

32-ch Isolated DIO and 32-ch TTL DIO Board Version 4.2, Jun. 2018

SUPPORTS

This manual relates to the following boards:

PCI Express	PEX-730, PEX-730A
Universal PCI	PISO-730U, PISO-730U-5V
PCI bus	PISO-730, PISO-730A, PISO-730A-5V

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

Copyright © 2018 by ICP DAS. All rights are reserved.

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. We will give you quick response within 2 workdays.

Email: service@icpdas.com, service.icpdas@gmail.com

TABLE OF CONTENTS

PACKING LIST 4

1. INTRODUCTION 5

1.1 FEATURES 6
1.2 SPECIFICATIONS 7
1.2.1 PEX-730 and PISO-730 Series 7
1.2.2 PEX-730A and PISO-730A Series 8

2. HARDWARE CONFIGURATION 9

2.1 BOARD LAYOUT 9
2.1.1 PISO-730 and PISO-730A(-5V) 9
2.1.2 PISO-730U(-5V) and PEX-730/730A Series 10
2.2 I/O OPERATION 11
2.2.1 Non-isolation DO Port Architecture (CON3) 11
2.2.2 Non-isolation DI Port Architecture (CON2) 12
2.2.3 Isolation DO Port Architecture (CON1) 13
2.2.4 Isolation DI Port Architecture (CON1) 14
2.3 INTERRUPT OPERATION 15
2.3.1 Interrupt Block Diagram 16
2.3.2 INT_CHAN_0 17
2.3.3 INT_CHAN_1 18
2.3.4 Initial_high, Active_low Interrupt Source 19
2.3.5 Initial_low, Active_high Interrupt Source 21
2.3.6 Multiple Interrupt Source 23
2.4 CARD ID SWITCH 25
2.5 PIN ASSIGNMENTS 26

3. HARDWARE INSTALLATION 27

4. SOFTWARE INSTALLATION 31

4.1 OBTAINING/INSTALLING THE DRIVER INSTALLER PACKAGE 31
4.2 PNP DRIVER INSTALLATION 34
4.3 VERIFYING THE INSTALLATION 36
4.3.1 Accessing Windows Device Manager 36
4.3.2 Check that the Installation 39

5. BOARD TESTING 40

5.1 Self-test Wiring 40
5.1.1 Non-isolation DIO Test Wiring 40
5.1.2 Isolation DIO Test Wiring 41
5.2 Launch the Test Program 46

6. I/O CONTROL REGISTER 48

6.1 How to Find the I/O Address 48
6.1.1 PIO_PISO.EXE Utility for Windows 50
6.1.2 PIO_PISO.EXE Utility for DOS 51
6.2 The Assignment of I/O Address 52
6.3 The I/O Address Map 54
6.3.1 RESET\ Control Register 55
6.3.2 AUX Control Register 55
6.3.3 AUX Data Register 56
6.3.4 INT Mask Control Register 56
6.3.5 AUX Status Register 57
6.3.6 Interrupt Polarity Control Register 57
6.3.7 I/O Data Register 58
6.3.8 DO Readback Register 59
6.3.9 Card ID Register 60
6.3.10 Version Number Register 60

7. DEMO PROGRAMS 61

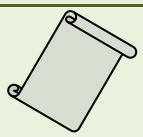
7.1 Demo Program for Windows 61
7.2 Demo Program for DOS 63
7.2.1 Demo1: DO Demo 65
7.2.2 Demo2: DIO Demo 67
7.2.3 Demo3: Interrupt (DI0 initial high) 69
7.2.4 Demo4: Interrupt (DI0 initial low) 71
7.2.5 Demo5: Interrupt (Multi interrupt source) 74

APPENDIX: DAUGHTER BOARD 77

A1. DB-16P Isolated Input Board 77
A2. DB-16R Relay Board 78
A3. DB-24PR, DB-24POR and DB-24C 79
A4. Daughter Board Comparison Table 80

Packing List

The shipping package includes the following items:

	One multi-function card as follows: PEX Series: PEX-730, PEX-730A PISO-730 series: PISO-730U, PISO-730U-5V, PISO-730 PISO-730A series: PISO-730A, PISO-730A-5V
	One printed Quick Start Guide
	One Software Utility CD
	One CA-4002 D-Sub Connect

Note:

If any of these items is missing or damaged, contact the dealer from whom you purchased the product. Save the shipping materials and carton in case you need to ship or store the product in the future.

1. Introduction

The PEX-730/730A and PISO-730U card is the new generation product that ICP DAS provides to meet RoHS compliance requirement. The PISO-730U card is designed as a drop-in replacement for the PISO-730, while the PEX-730/730A card is designed as easy replacement for the PISO-730/730A. Users can replace the PISO-730(A) by the PEX-730(A)/PISO-730U directly without software/driver modification.

PEX-730(A)/PISO-730U(-5V)/PISO-730A(-5V) cards provide 32 isolated digital I/O channels (16 x DI and 16 x DO) and 32 TTL-level digital I/O channels (16 x DI and 16 x DO). Both the isolated DI and DO channels use a short optical transmission path to transfer an electronic signal between the elements of a circuit and keep them electrically isolated. With 3750 V_{rms} isolation protection, these DIO channels allow the input signals to be completely floated so as to prevent ground loops and isolate the host computer from damaging voltages. Each digital output offers a **NPN (Current Sinking for PEX-730/PISO-730U(-5V)/PISO-730)** or **PNP (Current Sourcing for PEX-730A/PISO-730A(-5V))** transistor and integrated suppression diode for the inductive load. The open collector outputs (DO channels) are typically used for alarm and warning notification, signal output control, control for external circuits that require a higher voltage level, and signal transmission applications, etc.

The PEX-730/730A and PISO-730U(-5V) also adds a Card ID switch on-board. Users can set Card ID and then recognize the board by the ID via software when using two or more cards in one computer.

These cards support various OS versions, such as Linux, DOS, 32/64-bit Windows 10/8/7/2008/2003/XP. DLL and Active X control together with various language sample programs based on Turbo C++, Borland C++, Microsoft C++, Visual C++, Borland Delphi, Borland C++ Builder, Visual Basic, C#.NET, Visual Basic.NET and LabVIEW are provided in order to help users quickly and easily develop their own applications.

➢ **Comparison Table**

Model Name	Bus	DI			DO		
		Non-Isolated (5V/TTL)	Isolated		Non-Isolated (5V/TTL)	Isolated	
		Channel	Channel	Input Voltage	Channel	Channel	Type
PEX-730	PCI Express	16	16	Logic 1: 9 ~ 24 V	16	16	Sink, NPN
PISO-730U	Universal PCI	16	16	Logic 1: 9 ~ 24 V	16	16	Sink, NPN
PISO-730U-5V	Universal PCI	16	16	Logic 1: 5 ~ 12 V	16	16	Sink, NPN
PISO-730 (Phased-out)	5 V PCI	16	16	Logic 1: 5 ~ 12 V	16	16	Sink, NPN
PEX-730A	PCI Express	16	16	Logic 1: 9 ~ 24 V	16	16	Source, PNP
PISO-730A	5 V PCI	16	16	Logic 1: 9 ~ 24 V	16	16	Source, PNP
PISO-730A-5V	5 V PCI	16	16	Logic 1: 5 ~ 12 V	16	16	Source, PNP

1.1 Features

- Support the +5V PCI bus for PISO-730/PISO-730A(-5V)
- Support the +3.3/+5 V PCI bus for PISO-730U(-5V)
- Supports PCI Express x 1 for PEX-PEX-730/PEX-730A
- 64 DIO channels (32 Isolated channels and 32 Non-isolated channels)
- 3750 V_{rms} photo-isolated protection
- Built-in DC/DC converter with 3000 V_{DC} isolated
- Output status readback function for PISO-730U(-5V)/PEX-730/PEX-730A
- Supports Card ID (SMD Switch) for PISO-730U(-5V)/PEX-730/PEX-730A
- Two Interrupt sources
- SMD, Sort card, power saving
- Supports Plug & Play to obtain I/O resources
- No more manually setting of I/O address and IRQ

1.2 Specifications

1.2.1 PEX-730 and PISO-730 Series

Model Name		PEX-730	PISO-730U	PISO-730 (Phased-out)	PISO-730U-5V
Digital Input					
Isolation Voltage		3750 Vrms			
Channels	Isolated	16			
	Non-Isolated	16			
Compatibility	Isolated	Optical			
	Non-Isolated	5V/TTL			
Input Voltage	Isolated	Logic 0: 0 ~ 1 V Logic 1: 9 ~ 24 V (Logic 1: Min. 7 V; Max. 30 V)			Logic 0: 0 ~ 1 V Logic 1: 5 ~ 12 V (Logic 1: Min. 3.5 V; Max. 16 V)
	Non-Isolated	Logic 0: 0.8 V max.; Logic 1: 2.0 V min.			
Input Impedance		1.2 KΩ, 1 W			
Response Speed	Isolated	4 kHz (Typical)			
	Non-Isolated	500 kHz	1.2 MHz		
Digital Output					
Isolation Voltage		3750 Vrms			
Channels	16	16			
	16	16			
Compatibility	Isolated	Sink, Open Collector			
	Non-Isolated	5V/TTL			
Output Voltage	Non-Isolated	Logic 0: 0.4 V max.; Logic 1: 2.4 V min.			
Output Capability	Isolated	100 mA/+30 V for one channel @ 100% duty			
	Non-Isolated	Sink: 2.4 mA @ 0.8 V; Source: 0.8 mA @ 2.0 V			
Response Speed	Isolated	4 kHz (Typical)			
	Non-Isolated	500 kHz	500 kHz		
General					
Bus Type		PCI Express x 1	3.3 V / 5 V Universal PCI, 32-bit, 33 MHz	5 V PCI, 32-bit, 33 MHz	3.3 V / 5 V Universal PCI, 32-bit, 33 MHz
Data Bus		8-bit			
Card ID		Yes (4-bit)		-	Yes (4-bit)
I/O Connector		Female DB37 x 1; 20-pin box header x 2			
Dimensions (L x W x D) (Units: mm)		163 x 116 x 22	180 x 105 x 22		
Power Consumption		350 mA @ +3.3 V 250 mA @ +12 V	600 mA @ +5 V		
Operating Temperature		0 ~ 60 °C			
Storage Temperature		-20 ~ 70 °C			
Humidity		5 ~ 85% RH, non-condensing			

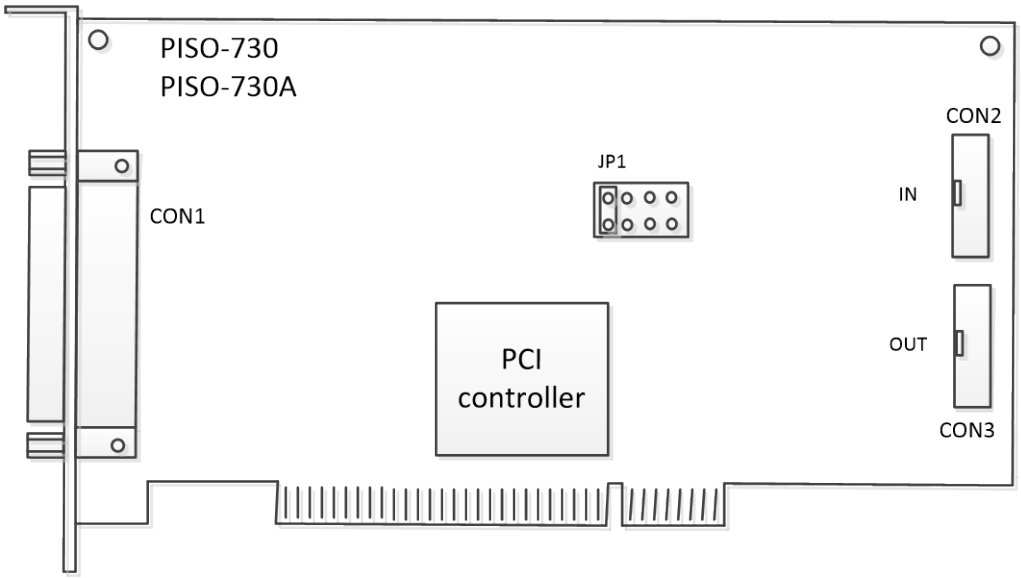
1.2.2 PEX-730A and PISO-730A Series

Model Name		PEX-730A	PISO-730A	PISO-730A-5V
Digital Input				
Isolation Voltage		3750 Vrms		
Channels	Isolated	16		
	Non-Isolated	16		
Compatibility	Isolated	Optical		
	Non-Isolated	5V/TTL		
Input Voltage	Isolated	Logic 0: 0 ~ 1 V Logic 1: 9 ~ 24 V (Logic 1: Min. 7 V; Max. 30 V)		Logic 0: 0 ~ 1 V Logic 1: 5 ~ 12 V (Logic 1: Min. 3.5 V; Max. 16 V)
	Non-Isolated	Logic 0: 0.8 V max.; Logic 1: 2.0 V min.		
Input Impedance		1.2 KΩ, 1 W		
Response Speed	Isolated	4 kHz (Typical)		
	Non-Isolated	1.2 MHz		
Digital Output				
Isolation Voltage		3750 Vrms		
Channels	16	16		
	16	16		
Compatibility	Isolated	Source, Open Collector		
	Non-Isolated	5V/TTL		
Output Voltage	Non-Isolated	Logic 0: 0.4 V max.; Logic 1: 2.4 V min.		
Output Capability	Isolated	100 mA/+30 V for one channel @ 100% duty		
	Non-Isolated	Sink: 2.4 mA @ 0.8 V; Source: 0.8 mA @ 2.0 V		
Response Speed	Isolated	4 kHz (Typical)		
	Non-Isolated	1.2 MHz		
General				
Bus Type		PCI Express x 1	5 V PCI, 32-bit, 33 MHz	
Data Bus		8-bit		
Card ID		Yes (4-bit)	-	
I/O Connector		Female DB37 x 1; 20-pin box header x 2		
Dimensions (L x W x D) (Units: mm)		163 x 110 x 22	180 x 105 x 22	
Power Consumption		350 mA @ +3.3 V 250 mA @ +12 V	640 mA @ +5 V	
Operating Temperature		0 ~ 60 °C		
Storage Temperature		-20 ~ 70 °C		
Humidity		5 ~ 85% RH, non-condensing		

2. Hardware Configuration

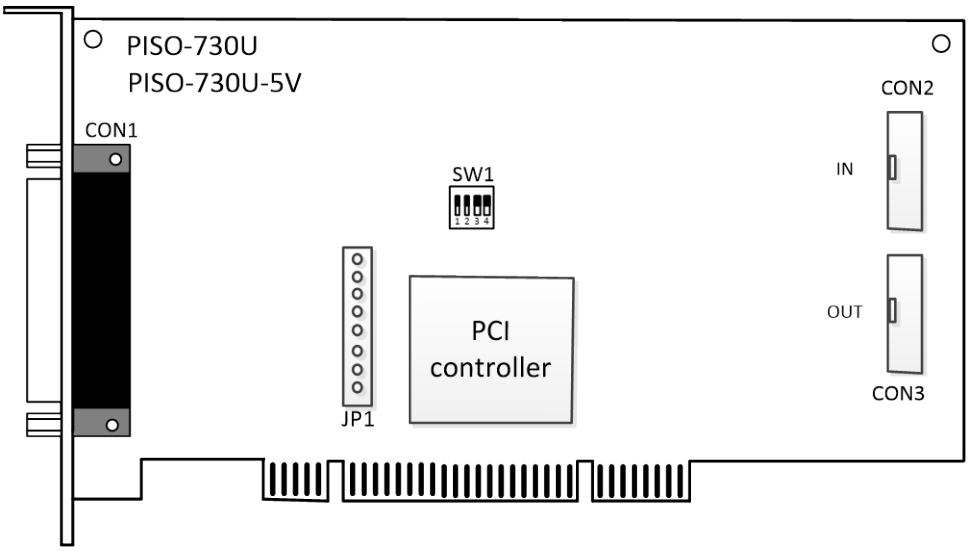
2.1 Board Layout

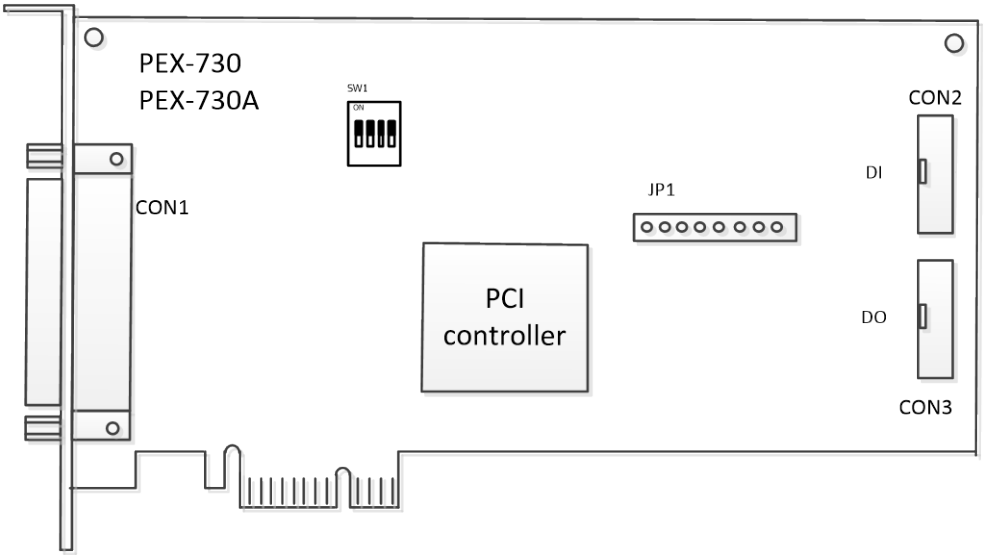
2.1.1 PISO-730 and PISO-730A(-5V)



Items	Description	
CON1	The terminal for isolation Digital Input/Output	Refer to Section 2.5 "Pin Assignments" for more detailed about pin assignments information.
CON2	The terminal for TTL Digital Input	
CON3	The terminal for TTL Digital Output	
JP1	Reserved	-

2.1.2 PISO-730U(-5V) and PEX-730/730A Series





Items	Description	
CON1	The terminal for isolation Digital Input/Output	Refer to Section 2.5 "Pin Assignments" for more detailed about pin assignments information.
CON2	The terminal for TTL Digital Input	
CON3	The terminal for TTL Digital Output	
JP1	Reserved	-
SW1	Card ID function	Refer to Section 2.4 "Card ID Switch" for more details.

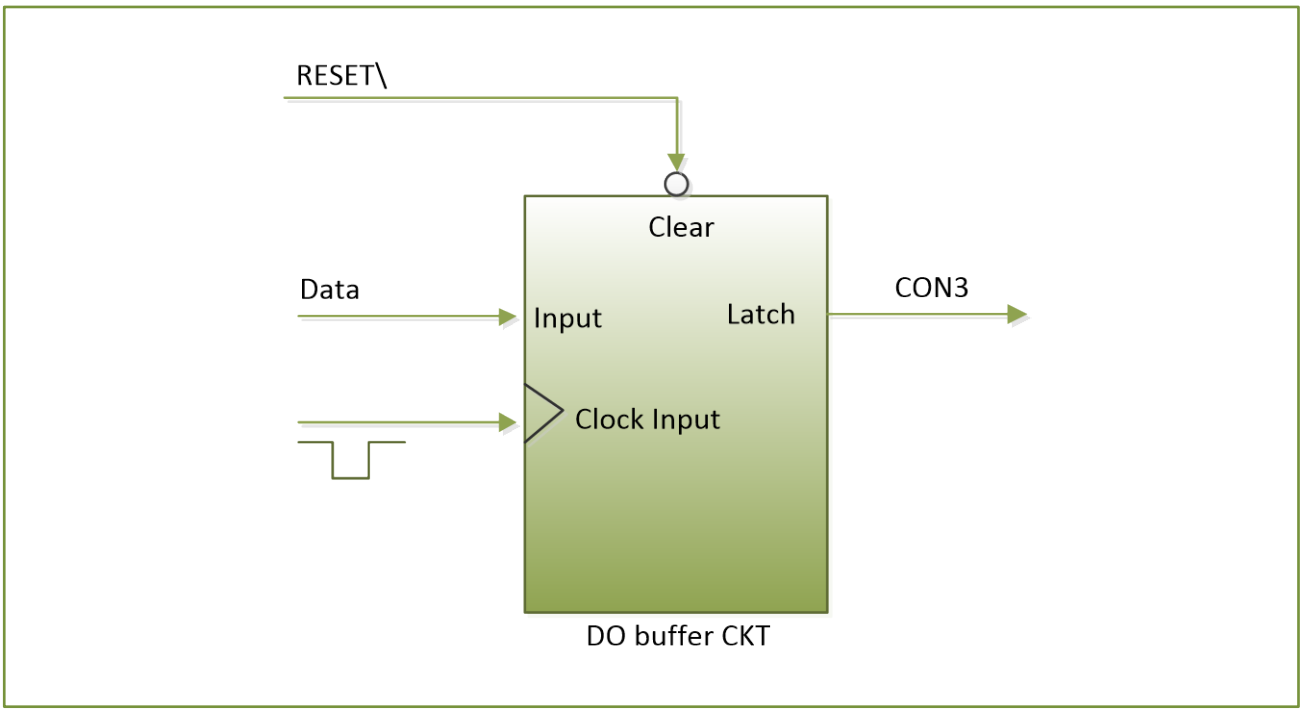
2.2 I/O Operation

2.2.1 Non-isolation DO Port Architecture (CON3)

When the PC is powered-up, all operations of non-isolated DO states are cleared to low state. The RESET\ signal is used to clear non-isolated DO states. Refer to Section 6.3.1 "RESET\ Control Register" for more information about the RESET\ signal.

- **The RESET\ is in Low-state → all non-isolated DO states are clear to low state**

The block diagram of Non-isolated DO is as follows:

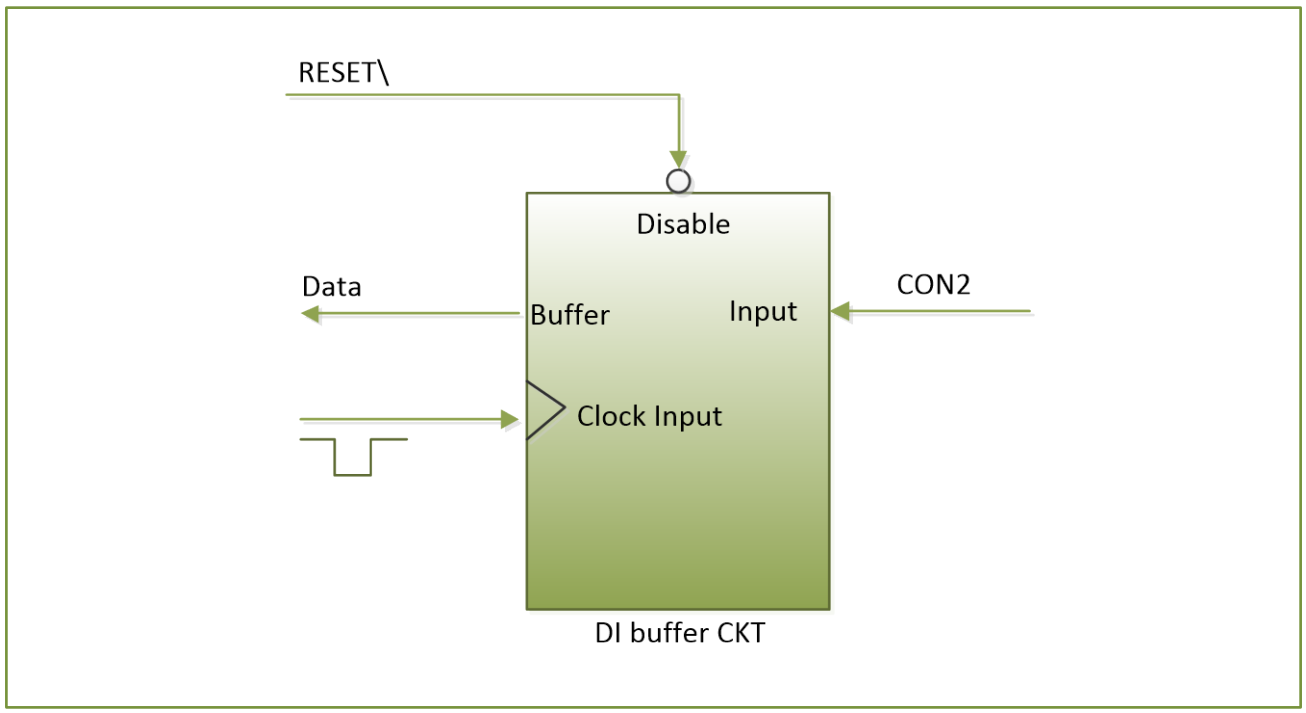


2.2.2 Non-isolation DI Port Architecture (CON2)

When the PC is powered-up, non-isolated DI port operations are disabled. The enable/disable for non-isolated DI ports is controlled by the RESET\ signal. Refer to Section 6.3.1 "RESET\ Control Register" for more information about the RESET\ signal.

- **The RESET\ is in Low-state → all non-isolated DI operation are disabled**
- **The RESET\ is in High-state → all non-isolated DI operation are enabled**

The block diagram of Non-isolated DI is as follows:

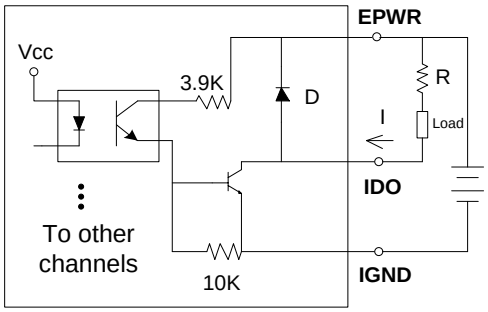
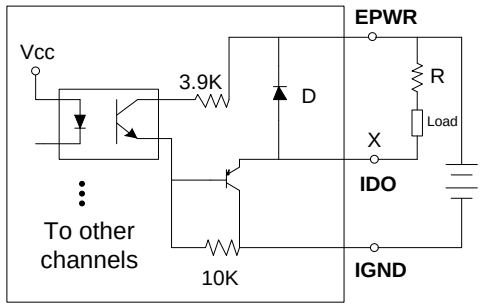
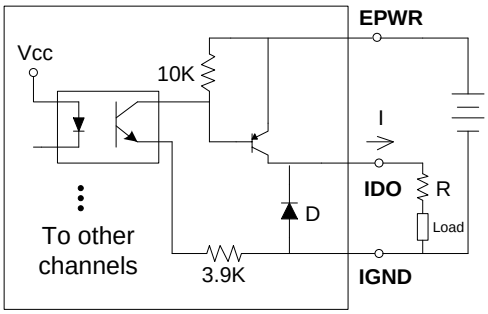
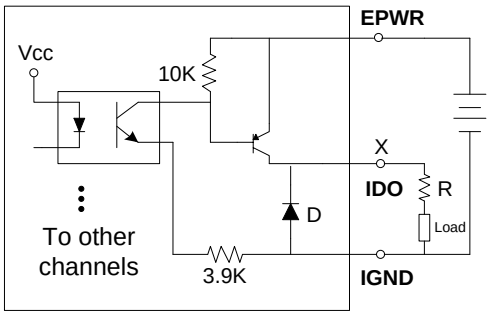


2.2.3 Isolation DO Port Architecture (CON1)

When the PC is powered-up, all operations of isolated DO states are cleared to low state. The RESET\ signal is used to clear isolated DO states. Refer to Section 6.3.1 "RESET\ Control Register" for more information about RESET\ signal.

➢ **The RESET\ is in Low-state → all isolated DO states are cleared to low state**

The IDO0 to IDO7 open-collector output channels use external power EPWR1, while the IDO8 to IDO15 channels use EPWR2.

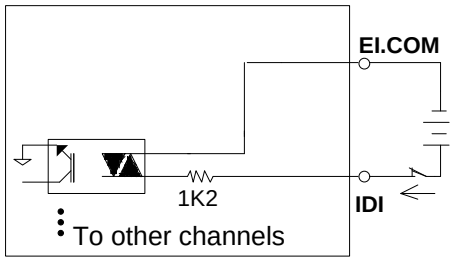
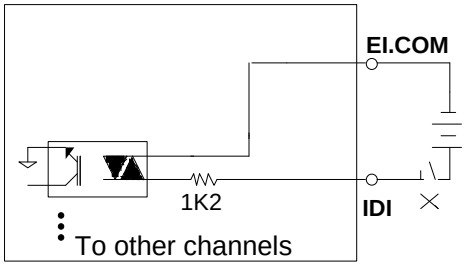
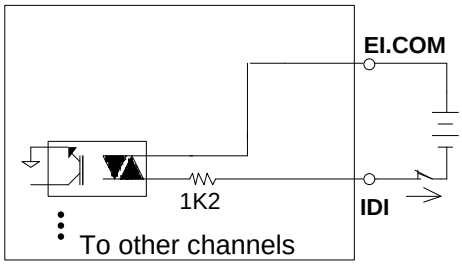
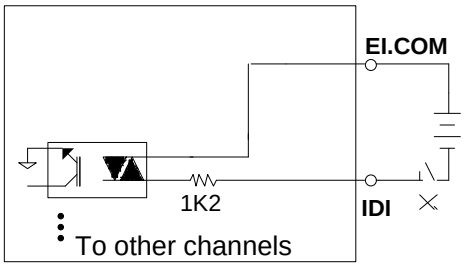
DO Group	ON State Readback as 1	OFF State Readback as 0
DO **(Sink, NPN)**	+(5)/+10 ~ +30 VDC	Open
		
DO **(Source, PNP)**	+(5)/+10 ~ +30 VDC	Open
		

Note:

It is necessary to connect a diode in the external device end as means of preventing damage from the counter emf, if your external device is inductive load, e.g. Relay...etc.

2.2.4 Isolation DI Port Architecture (CON1)

The PISO-730 series cards provide 16-channel isolated Digital Input. The **PEX-730/730A and PISO-730(U)/730A** each of the isolated Digital Input can accept **voltages from +9 VDC to +30 VDC**. The **PISO-730U-5V/730A-5V** each of the isolated digital input can accept **voltages from +5 VDC to +12 VDC**. The IDI0 to IDI7 isolated input channels use external common end point EI.COM1, while the IDI8 to IDI15 channels use EI.COM2.

Input Type	ON State as 0	OFF State as 1
Wet Contact (Sink)	+(5)/+10 ~ +30 VDC	+4 VDC Max.
		
Wet Contact (Source)	+(5)/+10 ~ +30 VDC	+4 VDC Max.
		

2.3 Interrupt Operation

There are two interrupt sources in PISO-730 series cards. These two signals are named as INT_CHAN_0 and INT_CHAN_1. Their signal sources are given as follows:

INT_CHAN_0: DI0
INT_CHAN_1: DI1

If only one interrupt signal source is used, the interrupt service routine does not have to identify the interrupt source. Refer to Section 7.2.3 “DEMO3.C” and Section 7.2.4 “DEMO4.C” for more information.

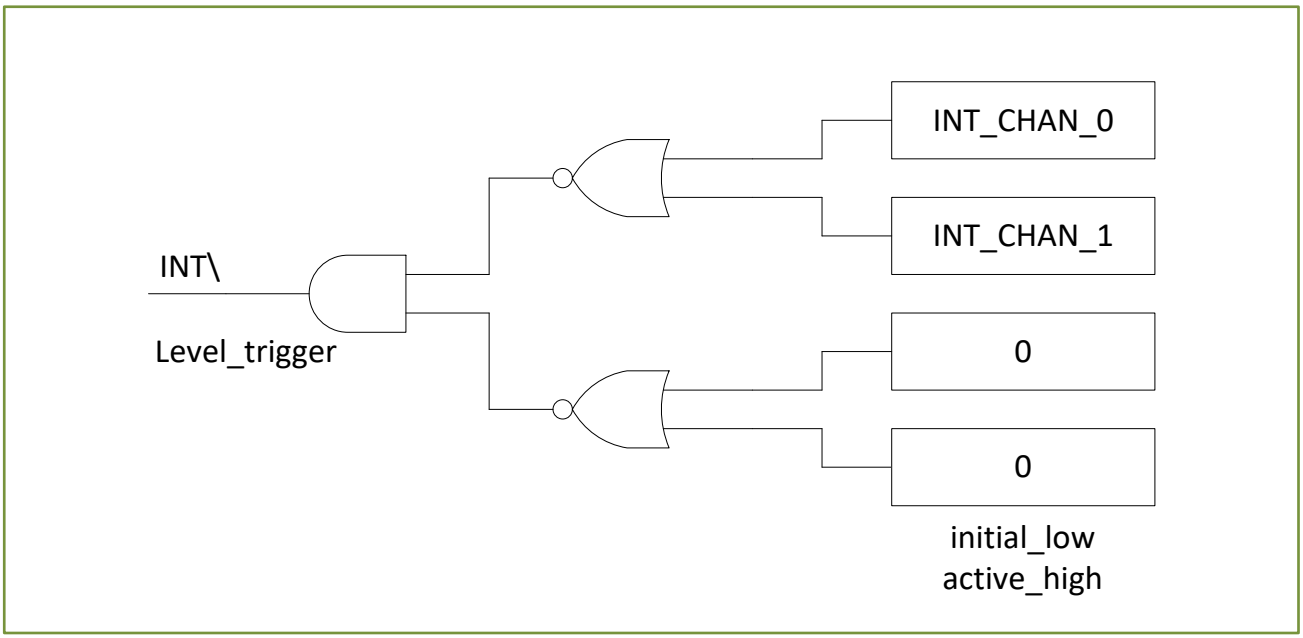
If there is more than one interrupt source, the interrupt service routine will identify the active signals as follows: (refer to Section 7.2.5 “DEMO5.C”)

1. Reads the new status of all interrupt signal sources (refer to Section 6.3.5 “AUX Status Register”)
2. Compares the new status with the old status to identify the active signals
3. If INT_CHAN_0 is active, services it
4. If INT_CHAN_1 is active, services it
5. Updates interrupt status

Note:

If the interrupt signal is too short, the new status may be as same as old status. In that condition, the interrupt service routine cannot identify which interrupt source is active. So the interrupt signal must be hold_active long enough until the interrupt service routine is executed. This hold_time is different for different O.S. The hold_time can be as short as a micro-second or as long as a second. In general, 20 ms is enough for any O. S.

2.3.1 Interrupt Block Diagram



The interrupt output signal of PISO-730 series cards, INT\ is a **level-trigger, Active_Low signal**. If the INT\ generates a low-pulse, the PISO-730 will interrupt the PC once a time. If the INT\ is fixed in low level, the PISO-730 series cards will interrupt the PC continuously. So the INT_CHAN_0/1 must be controlled by **pulse_type** signals. **They must be fixed in low-level state normally and generate a high_pulse to interrupt the PC.**

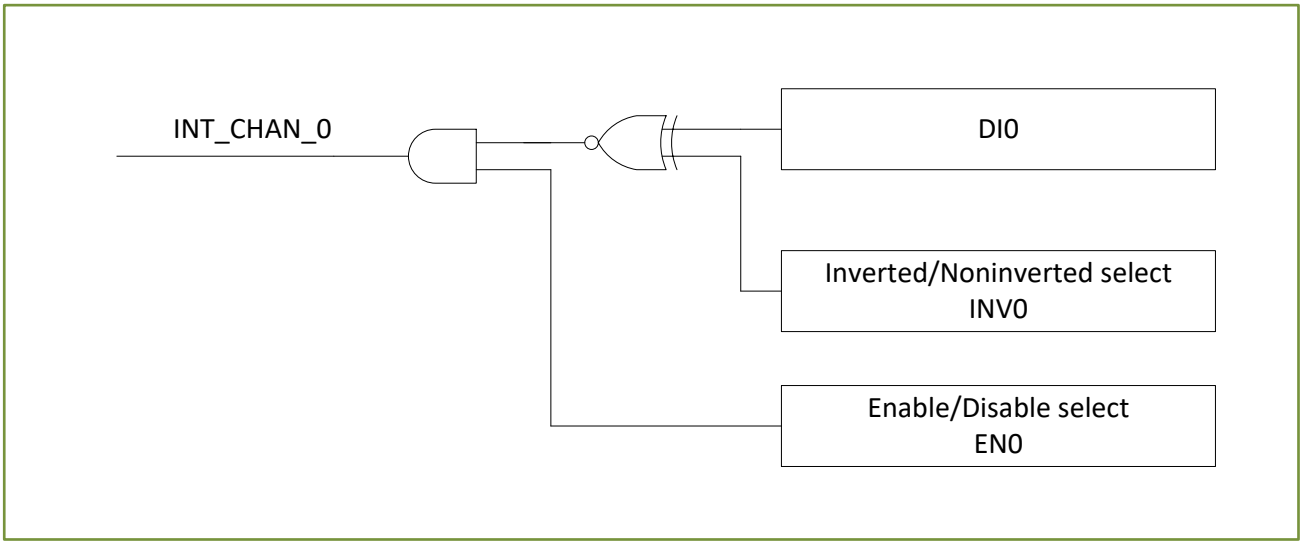
The priority of INT_CHAN_0/INT_CHAN_1 is the same. If these two signals are active at the same time, then INT\ will be activated only once. So the interrupt service routine has to read the status of all interrupt channels for a multi-channel interrupt. Refer to Section 2.3.6 “Multiple Interrupt Source” for more information.

Section 7.2.5 “DEMO5.C” → for multi-channel interrupt source

If only one interrupt source is used, the interrupt service routine doesn’t have to read the status of interrupt source. The demo programs Section 7.2.3 “DEMO3.C” and Section 7.2.4 “DEMO4.C” are designed for single-channel interrupt demo as follows:

Section 7.2.3 “DEMO3.C” and Section 7.2.4 “DEMO4.C” → for INT_CHAN_0 only

2.3.2 INT_CHAN_0



The INT_CHAN_0 must be fixed in a normal, low-level state and generate a high_pulse to interrupt the PC.

The EN0 can be used to enable/disable the INT_CHAN_0 as follows: (Refer to Section 6.3.4 "INT Mask Control Register")

EN0=0 →INT_CHAN_0=disable

EN0=1 →INT_CHAN_0=enable

The INV0 can be used to invert/non-invert the DI0 as follows: (Refer to Section 6.3.6 "Interrupt Polarity Control Register")

INV0=0→INT_CHAN_0=invert state of DI0

INV0=1→INT_CHAN_0=non-invert state of DI0

Refer to the following demo program for more information:

Section 7.2.3 "DEMO3.C" → for INT_CHAN_0 (initial high)

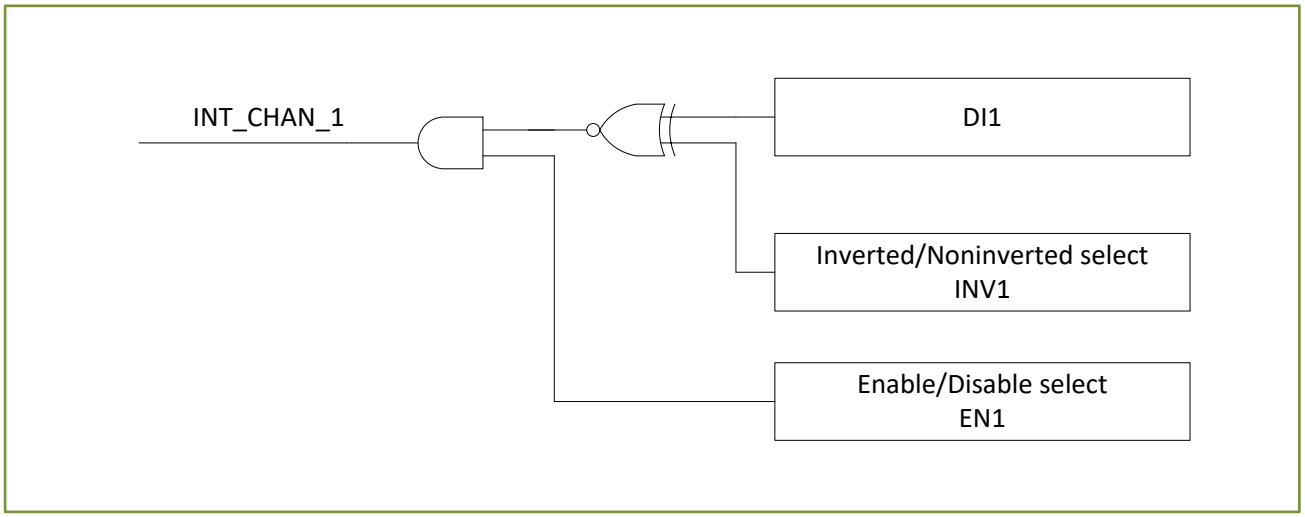
Section 7.2.4 "DEMO4.C" → for INT_CHAN_0 (initial low)

Section 7.2.5 "DEMO5.C" → for multi-channel interrupt source

Note:

Refer to Section 2.3.4 "Initial_high, Active_low Interrupt Source" and Section 2.3.5 "Initial_low, Active_high Interrupt Source" for active high-pulse generation.

2.3.3 INT_CHAN_1



The INT_CHAN_1 must be fixed in a normal low-level state and generated a high_pulse to interrupt the PC.

The EN1 can be used to enable/disable the INT_CHAN_1 as follows: (Refer to Section 6.3.4 "INT Mask Control Register")

EN1=0 →INT_CHAN_1=disable

EN1=1 →INT_CHAN_1=enable

The INV1 can be used to invert/non-invert the DI1 as follows: (Refer to Section 6.3.6 "Interrupt Polarity Control Register")

INV1=0→INT_CHAN_1=invert state of DI1

INV1=1→INT_CHAN_1=non-invert state of DI1

Refer to demo program for more information as follows:

Section 7.2.3 "DEMO3.C" → for INT_CHAN_0 (initial high)

Section 7.2.4 "DEMO4.C" → for INT_CHAN_0 (initial low)

Section 7.2.5 "DEMO5.C" → for multi-channel interrupt source

Note:

Refer to Section 2.3.4 "Initial_high, Active_low Interrupt Source" and Section 2.3.5 "Initial_low, Active_high Interrupt Source" for active high-pulse generation.

2.3.4 Initial_high, Active_low Interrupt Source

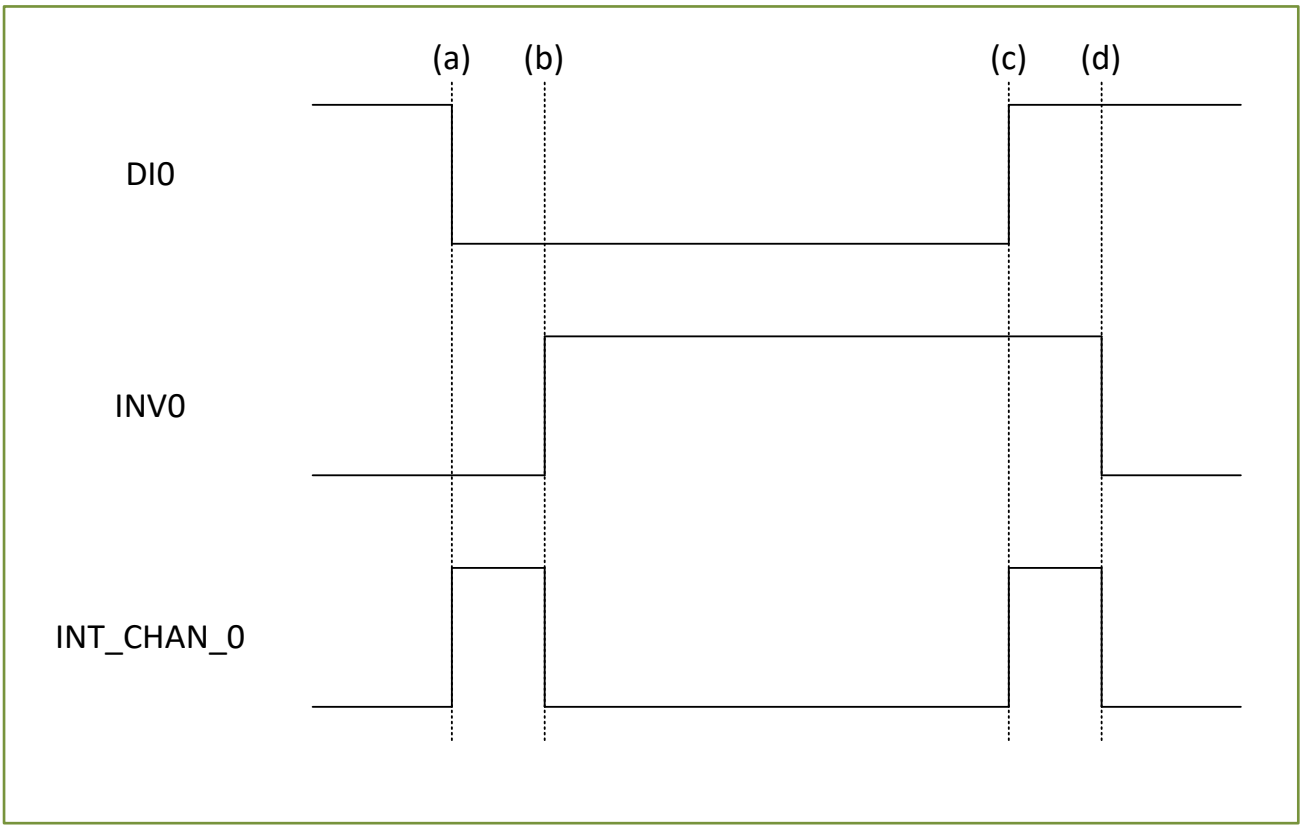
If the DI0 is an initial_high, active_low signal, the interrupt service routine should use INV0 to invert/non-invert the DI0 for high_pulse generation as follows: (Refer to Section 7.2.3 "DEMO3.C" and the DI1 is similarly)

Initial set:

```
now_int_state=1;                    /* initial state for DI0        */
outportb(wBase+0x2a,0);             /* select the inverted DI0      */
```

```
void interrupt irq_service()
{
if (now_int_state==1)                /* now DI0 is changed to LOW, refer to Figure 2-3-4 */(a)
    {                                /* → INT_CHAN_0=!DI0=HIGH now      */
    COUNT_L++;                       /* find a LOW_pulse (DI0)          */
    If((inport(wBase+7)&1)==0)       /* the DI0 is still fixed in LOW   */
      {                              /* → needs to generate a high_pulse */
      outportb(wBase+0x2a,1);        /* INV0 select the non-inverted input, refer to Figure 2-3-4 */(b)
                                     /* INT_CHAN_0=DI0=LOW -->          */
                                     /* INT_CHAN_0 generate a high_pulse */
      now_int_state=0;               /* now DI0=LOW                     */
      }
    else now_int_state=1;            /* now DI0=HIGH                    */
                                     /* doesn't have to generate high_ pulse */
    }
else                                 /* now DI0 is changed to HIGH, refer to Figure2-3-4 */(c)
    {                                /* → INT_CHAN_0=DI0=HIGH now       */
    COUNT_H++;                       /* find a HIGH_pulse (DI0)         */
    If((inport(wBase+7)&1)==1)       /* the DI0 is still fixed in HIGH  */
      {                              /* needs to generate a high_pulse  */
      outportb(wBase+0x2a,0);        /* INV0 select the inverted input,  refer to Figure 2-3-4 */(d)
                                     /* INT_CHAN_0=!DI0=LOW →           */
                                     /* INT_CHAN_0 generate a high_pulse */
      now_int_state=1;               /* now DI0=HIGH                    */
      }
    else now_int_state=0;            /* now DI0=LOW                     */
                                     /* doesn't have to generate high_ pulse */
    }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1  8259,0x20);
```

Figure 2-3-4



2.3.5 Initial_low, Active_high Interrupt Source

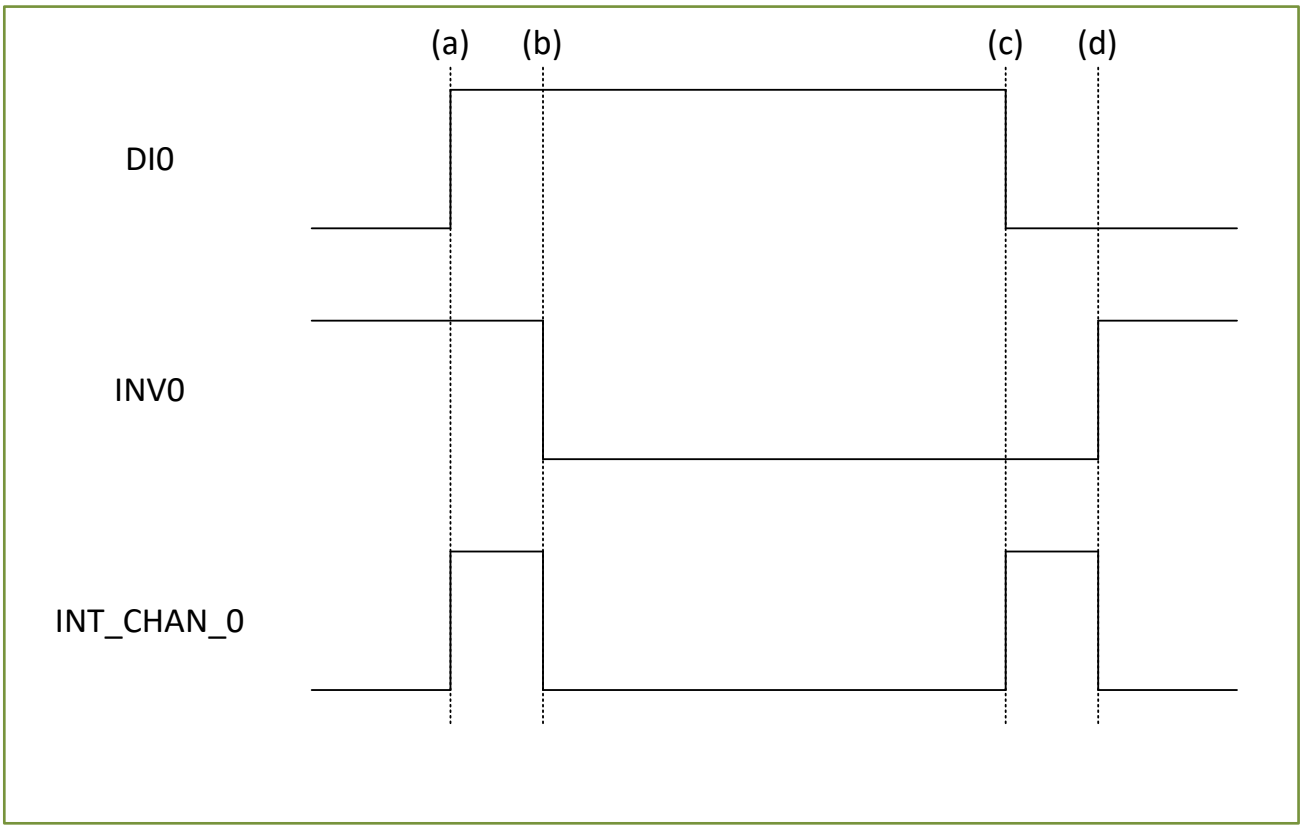
If the DI0 is an initial_low, active_high signal, the interrupt service routine should use INV0 to invert/non-invert the DI0 for high_pulse generation as follows: (Refer to Section 7.2.4 "DEMO4.C" and the DI1 is similarly)

Initial set:

```
now_int_state=0;                    /* initial state for DI0      */
outportb(wBase+0x2a,1);             /* select the non-inverted DI0 */
```

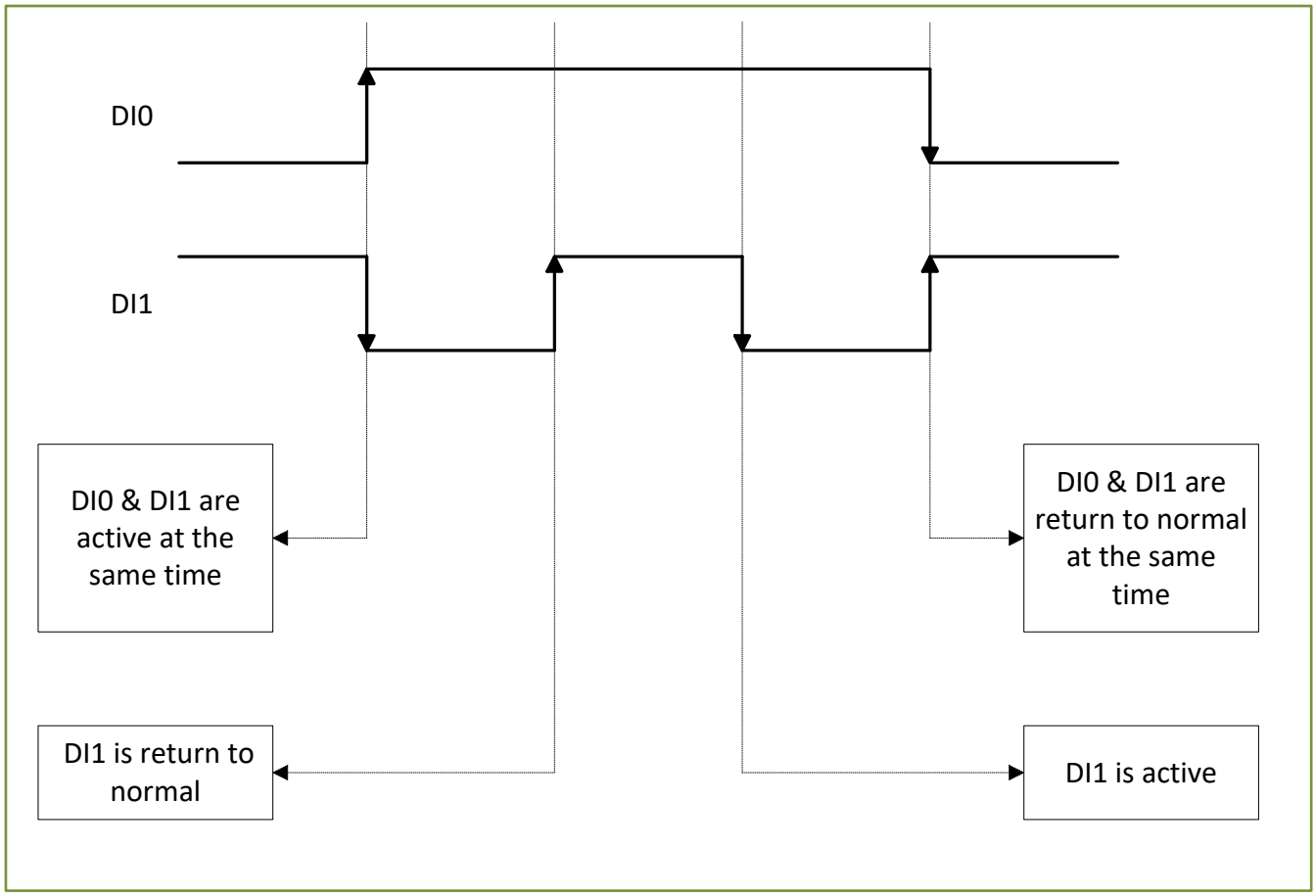
```
void interrupt irq_service()
{
if (now_int_state==1)               /* now DI0 is changed to LOW, refer to Figure 2-3-5   */(c)
    {                               /* → INT_CHAN_0=!DI0=HIGH now        */
    COUNT_L++;                      /* find a LOW_pulse (DI0)            */
    If((inport(wBase+7)&1)==0)      /* the DI0 is still fixed in LOW     */
      {                             /* → needs to generate a high_pulse  */
      outportb(wBase+0x2a,1);       /* INV0 select the non-inverted input, refer to Figure 2-3-5   */(d)
                                    /* INT_CHAN_0=DI0=LOW →              */
                                    /* INT_CHAN_0 generate a high_pulse  */
      now_int_state=0;              /* now DI0=LOW                       */
      }
    else now_int_state=1;           /* now DI0=HIGH                      */
                                    /* doesn☐have to generate high_ pulse */
    }
else                                /* now DI0 is changed to HIGH, refer to Figure 2-3-5   */(a)
    {                               /* → INT_CHAN_0=DI0=HIGH now         */
    COUNT_H++;                      /* find a High_pulse (DI0)           */
    If((inport(wBase+7)&1)==1)      /* the DI0 is still fixed in HIGH    */
      {                             /* needs to generate a high_pulse    */
      outportb(wBase+0x2a,0);       /* INV0 select the inverted input, refer to Figure 2-3-5   */(b)
                                    /* INT_CHAN_0=!DI0=LOW →             */
                                    /* INT_CHAN_0 generate a high_pulse  */
      now_int_state=1;              /* now DI0=HIGH                      */
      }
    else now_int_state=0;           /* now DI0=LOW                       */
                                    /* doesn☐have to generate high_ pulse */
    }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1  8259,0x20);
```

Figure 2-3-5



2.3.6 Multiple Interrupt Source

Assume: DI0 is initial Low, active High
DI1 is initial High, active Low
as follows:



Refer to Section 7.2.5 "DEMO5.C" for source program. **These three falling-edge and rising-edge scenarios can be detected by** Section 7.2.5 "DEMO5.C".

Note:

When the interrupt is active, the user program has to identify the active signals. More than one signal may be simultaneously. The interrupt service routine has to service all active signals at the same time.

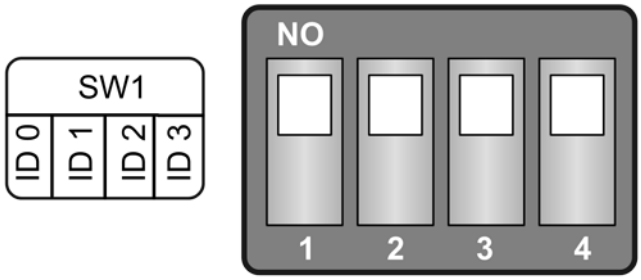
Initial set:

```
now_int_state=0x2;          /* Initial state: DI0 at low level, DI1 at high level    */
invert=0x1;                 /* non-invert DI0 & invert DI1                           */
outportb(wBase+0x2a,invert);
```

```
void interrupt irq_service()
{
new_int_state=inportb(wBase+7)&0x03;      /* read all interrupt state */
int_c=new_int_state^now_int_state;        /* compare which interrupt  */
                                          /* signal has changed       */

if ((int_c&0x1)!=0)                       /* INT_CHAN_0 is active     */
    {
    if ((new_int_state&0x01)!=0)          /* now DI0 changes to high  */
        {
        CNT_H1++;
        }    else                         /* now DI0 changes to low   */
        {
        CNT_L1++;
        }    invert=invert^1;             /* to generate a high pulse */
}
if ((int_c&0x2)!=0)
 {    if ((new_int_state&0x02)!=0)        /* now DI1 change to high   */
        {
        CNT_H2++;
        }    else                         /* now DI1 changes to low   */
        {
        CNT_L2++;
        }    invert=invert^2;             /* to generate a high pulse */
}
now_int_state=new_int_state;
outportb(wBase+0x2a,invert);
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

2.4 Card ID Switch

The PEX-730/730A and PISO-730U(-5V) has a Card ID switch (SW1) with which users can recognize the board by the ID via software when using two or more cards in one computer. The default Card ID is 0x0. For details SW1 (Card ID function) settings, refer to Table 2-1. Note that the Card ID function is only supported by the PEX-730/730A and PISO-730U(-5V).



(Default Settings)

Table 2-1 (*) Default Settings; OFF → 1; ON → 0

Card ID (Hex)	1 ID0	2 ID1	3 ID2	4 ID3
(*) 0x0	ON	ON	ON	ON
0x1	**OFF**	ON	ON	ON
0x2	ON	**OFF**	ON	ON
0x3	**OFF**	**OFF**	ON	ON
0x4	ON	ON	**OFF**	ON
0x5	**OFF**	ON	**OFF**	ON
0x6	ON	**OFF**	**OFF**	ON
0x7	**OFF**	**OFF**	**OFF**	ON
0x8	ON	ON	ON	**OFF**
0x9	**OFF**	ON	ON	**OFF**
0xA	ON	**OFF**	ON	**OFF**
0xB	**OFF**	**OFF**	ON	**OFF**
0xC	ON	ON	**OFF**	**OFF**
0xD	**OFF**	ON	**OFF**	**OFF**
0xE	ON	**OFF**	**OFF**	**OFF**
0xF	**OFF**	**OFF**	**OFF**	**OFF**

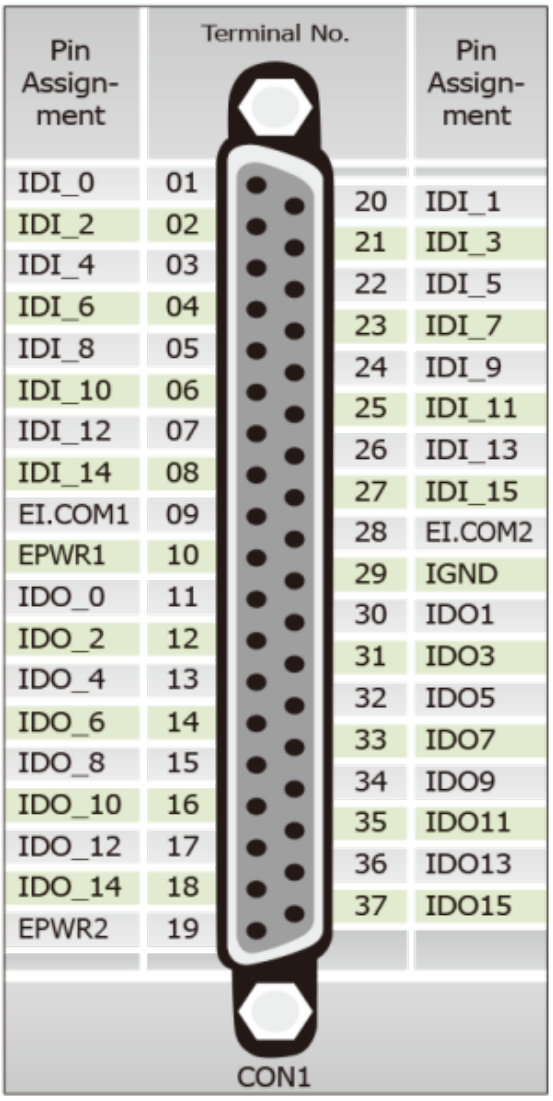
2.5 Pin Assignments

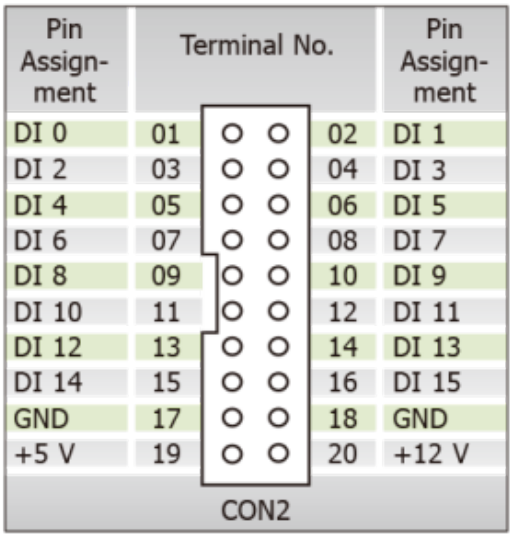
The Pin assignments of CON1, CON2 and CON3 on the PISO-730 series cards are represented in the figure below.

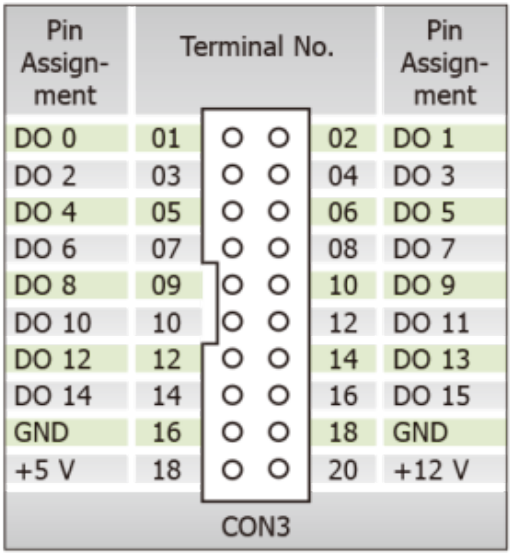
- CON2/CON3: 20-pin flat-cable headers for 5V/TTL Digital Input and Output.

CON2 and COM3 are TTL compatible	
High (1)	2.0 ~ 5.0 V (Voltage over 5.0 V will damage the device)
None Define	2.0 V ~ 0.8 V
Low (0)	Under 0.8 V

- CON1: 37-pin D-type female connector isolation Digital for Input and Output.







3. Hardware Installation

Note:

It is recommended that the driver is installed before installing the hardware as the computer may need to be restarted once the driver is installed in certain operating systems, such as Windows 2000 or Windows XP, etc. Installing the driver first helps reduce the time required for installation and restarting the computer.

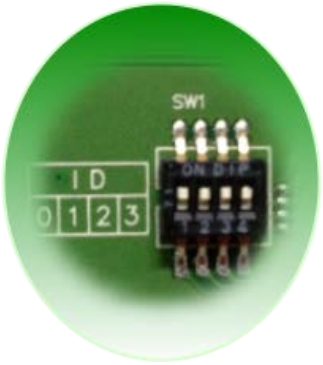
To install your PEX/PISO-730 Series board, follow the procedure described below:

Step 1: Install the driver for your board on Host computer.



For detailed information regarding driver installation, refer to Chapter 4 "Software Installation".

Step 2: Configure the Card ID using the DIP Switch (SW1).



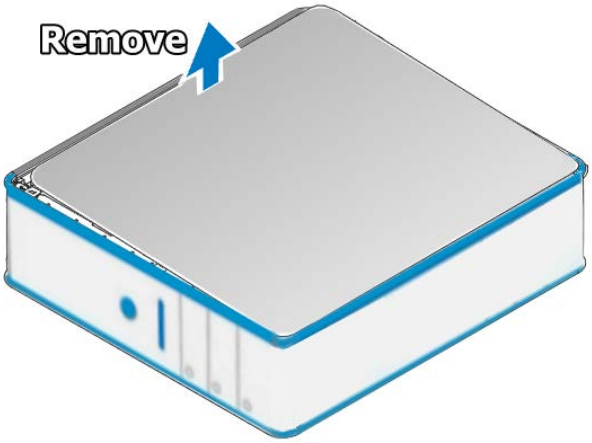
For detailed information about the card ID (SW1), refer to Section 2.4 "Car ID Switch".

Note:

The Card ID function only supports PEX-730, PEX-730A and PISO-730U(-5V).

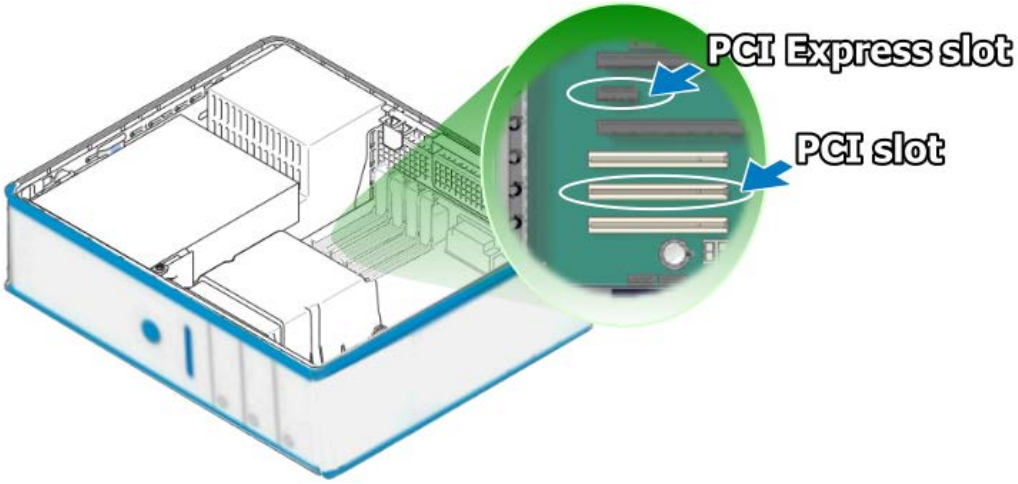
Step 3: Shut down and switch off the power to the computer, and then disconnect the power supply.

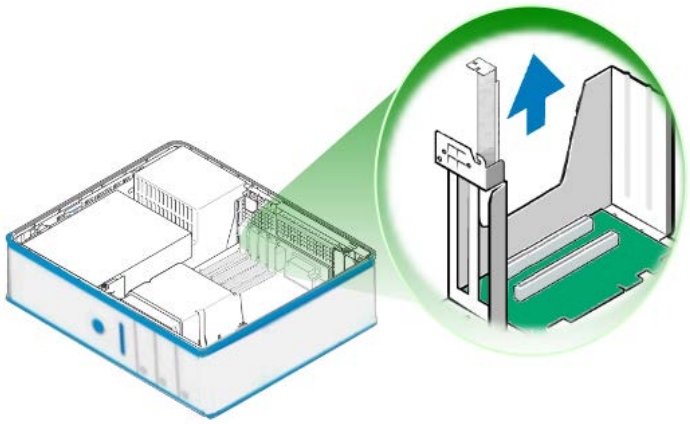




Step 4: Remove the cover from the computer.

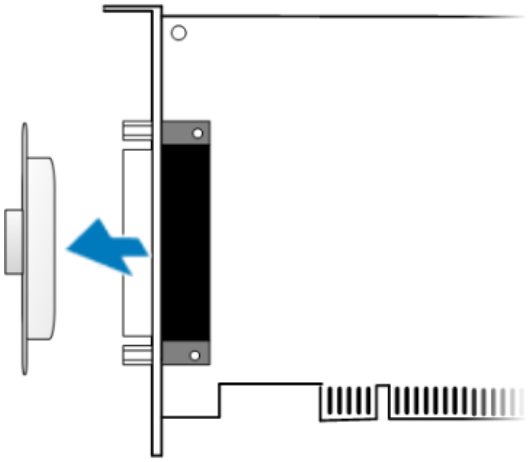
Step 5: Select a vacant PCI/PCI Express slot.



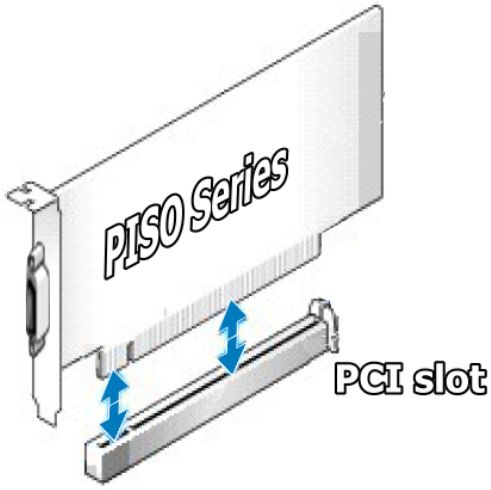


Step 6: Unscrew and remove the PCI slot cover from the computer case.

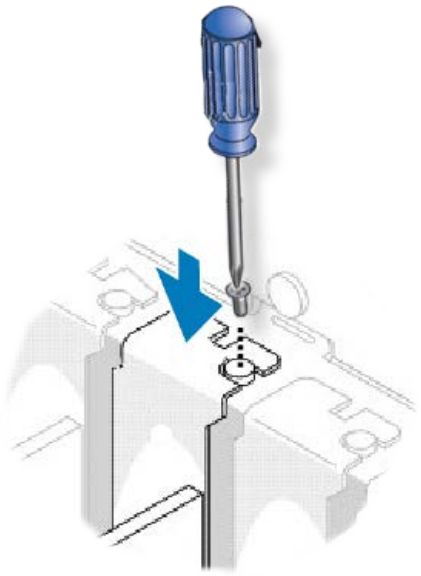
Step 7: Remove the connector cover from your board.



Step 8: Carefully insert your board into the PCI/PCI Express slot by gently pushing down on both sides of the board until it slides into the PCI connector.

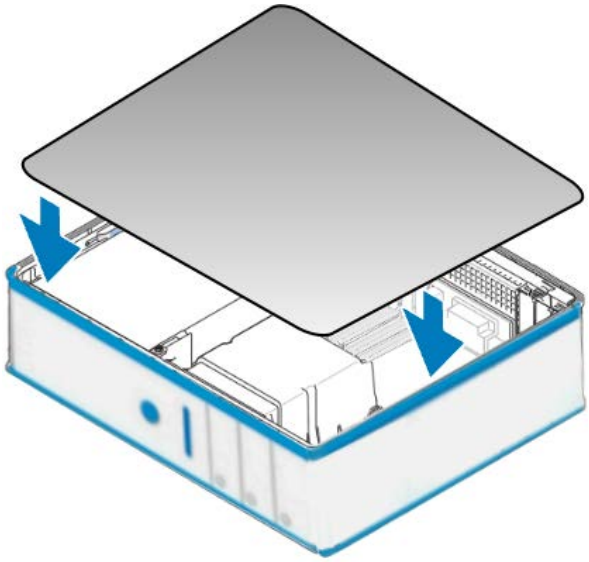






Step 9: Confirm that the board is correctly inserted in the motherboard, and then secure your board in place using the retaining screw that was removed in **Step 6.**

Step 10: Replace the covers on the computer.



Step 11: Re-attach any cables, insert the power cord and then switch on the power to the computer.



Once the computer reboots, follow any message prompts that may be displayed to complete the Plug and Play installation procedure. Refer to Chapter 4 "Software Installation" for more information.

4. Software Installation

This chapter provides a detailed description of the process for installing the PISO-730 series driver and how to verify whether the PISO-730 was properly installed. PISO-730 series card can be used on DOS, Linux and 32/64-bit versions of Windows XP/2003/2008/7/8/10 based systems, and the drivers are fully Plug &Play (PnP) compliant for easy installation.

4.1 Obtaining/Installing the Driver Installer Package

The driver installation package for PISO-730 series card can be found on the companion CD-ROM, or can be obtained from the ICP DAS FTP web site. Install the appropriate driver for your operating system. The location and website addresses for the installation package are indicated below.

- **UniDAQ Driver/SDK** (It is recommended to install this driver for new user.)

Operating System	32/64-bit Windows XP, 32/64-bit Windows 2003, 32/64-bit Windows 7, 32/64-bit Windows 2008, 32/64-bit Windows 8 and 32/64-bit Windows 10
Driver Name	UniDAQ Driver/SDK (unidaq_win_setup_xxxx.exe)
CD-ROM	CD:\\ NAPDOS\PCI\UniDAQ\DLL\Driver\
Web site	http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/dll/driver/
Installing Procedure	To install the UniDAQ driver, follow the procedure described below. Step 1: Double-click the UniDAQ_Win_Setupxxx.exe icon to begin the installation process.

Installation Procedure	Step 2: When the "Welcome to the ICP DAS UniDAQ Driver Setup Wizard" screen is displayed, click the "Next>" button to start the installation. Step 3: On the "Information" screen, verify that the DAQ board is included in the list of supported devices, then click the "Next>" button. Step 4: On the "Select Destination Location" screen, click the "Next>" button to install the software in the default folder, C:\ICPDAS\UniDAQ. Step 5: On the "Select Components" screen, verify that the DAQ board is in the list of device, and then click the "Next>" button to continue. Step 6: On the "Select Additional Tasks" screen, click the "Next>" button to continue. Step 7: On the "Download Information" screen, click the "Next>" button to continue. Step 8: Once the installation has completed, click "No, I will restart my computer later", and then click the "Finish" button. For more detailed information about how to install the UniDAQ driver, refer to Section 2.2 "Install UniDAQ Driver DLL" of the UniDAQ Software Manual, which can be found in the \NAPDOS\PCI\UniDAQ\Manual\ folder on the companion CD, or can be downloaded from: http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/manual/

➢ **PISO-DIO Series Classic Driver** (Recommended to install this driver for have been used PISO-DIO series boards of regular user)

Operating System	Windows 95/98/ME, 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
Driver Name	PISO-DIO Series Classic Driver (PISO-DIO_win_xxxx.exe)
CD-ROM	CD:\\NAPDOS\PCI\PISO-DIO\DLL_OXC\Driver\
Web site	http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/piso-dio/dll_ocx/driver/
Installing Procedure	Please follow the following steps to setup software: Step 1: Double click the PISO-DIO Series Classic Driver to setup it. Step 2: When the Setup Wizard screen is displayed, click the Next> button. Step 3: Select the folder where the drivers are to install. The default path is C:\DAQPro\PISO-DIO. But if you wish to install the drivers to a different location , click the "Browse..." button and select the relevant folder and then click the Next> button. Step 4: Click the Install button to continue. Step 5: Select the item "No, I will restart my computer later", press the Finish button. For detailed information about how to install the PISO-DIO Classic Driver, refer to the PISO-DIO Series Classic Driver DLL Software, which can be found in the \NAPDOS\PCI\PISO-DIO\Manual\ folder on the companion CD, or can be downloaded from: http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/piso-dio/manual/

4.2 PnP Driver Installation

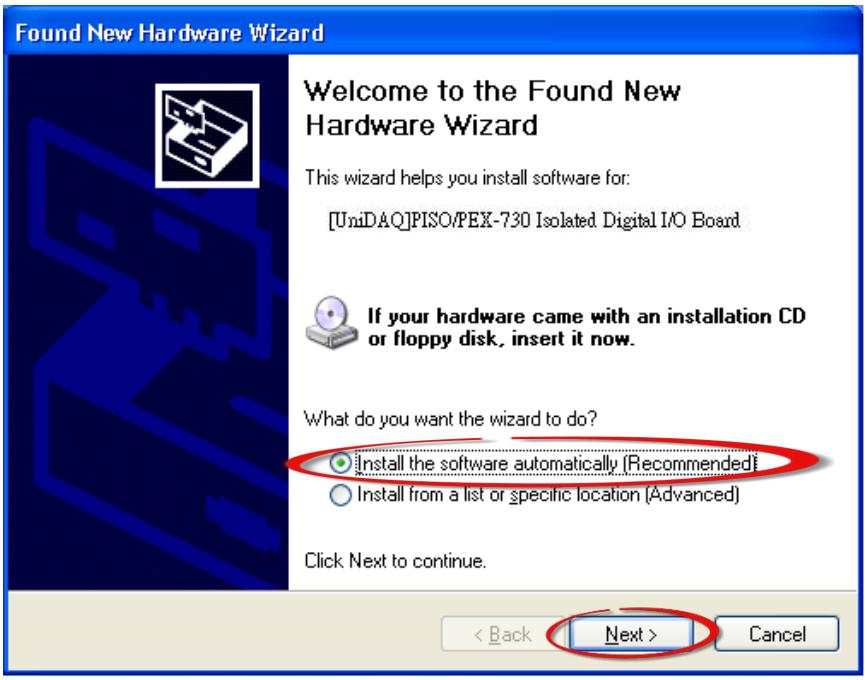
Step 1: Correctly shut down and power off your computer and disconnect the power supply, and then install your board into the computer. For detailed information about the hardware installation of PEX/PISO-730 series board, refer to Chapter 3 "Hardware Installation".

Step 2: Power on the computer and complete the Plug and Play installation**.**

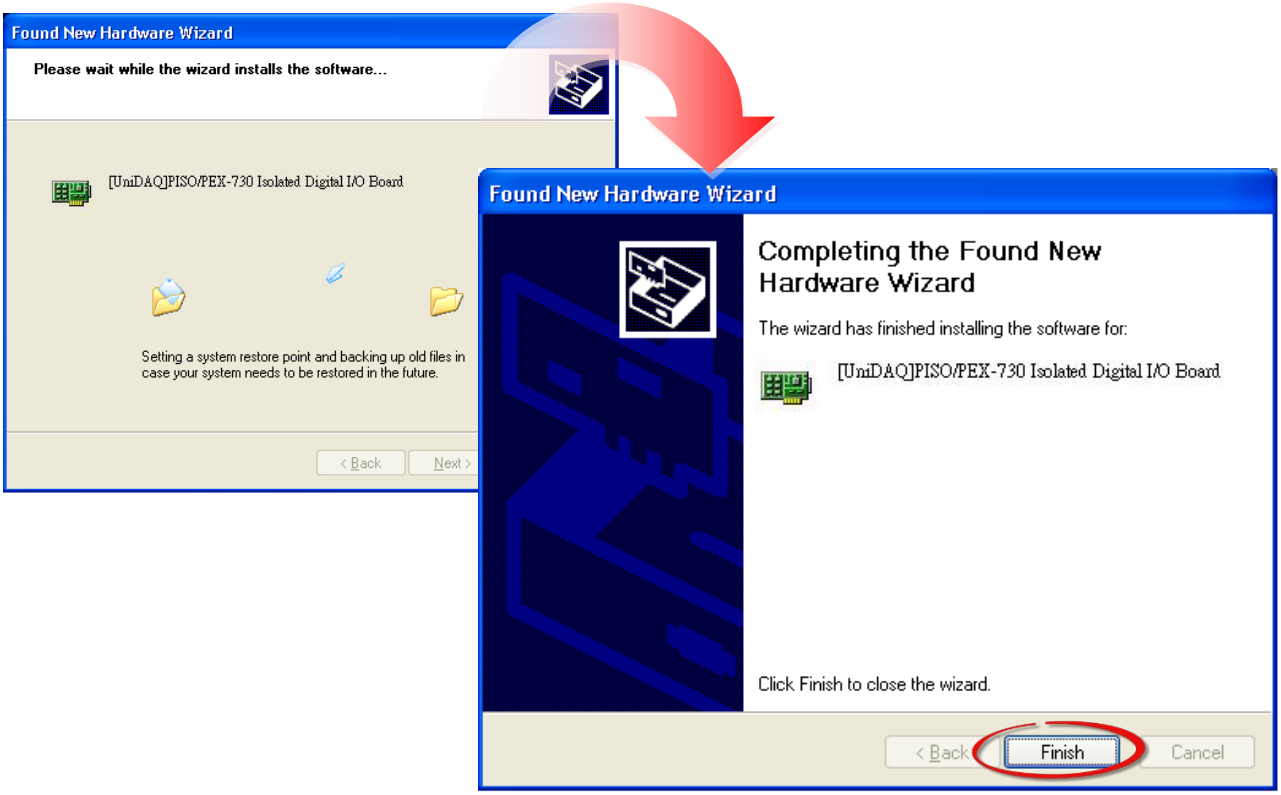
Note:

More recent operating systems, such as Windows 7/8/10 will automatically detect the new hardware and install the necessary drivers etc., so Steps 3 to 5 can be skipped.

Step 3: Select **"Install the software automatically [Recommended]"** and click the **"Next>"** button.



Step 4: Click the **"Finish"** button.



Step 5: Windows pops up **"Found New Hardware"** dialog box again.



4.3 Verifying the Installation

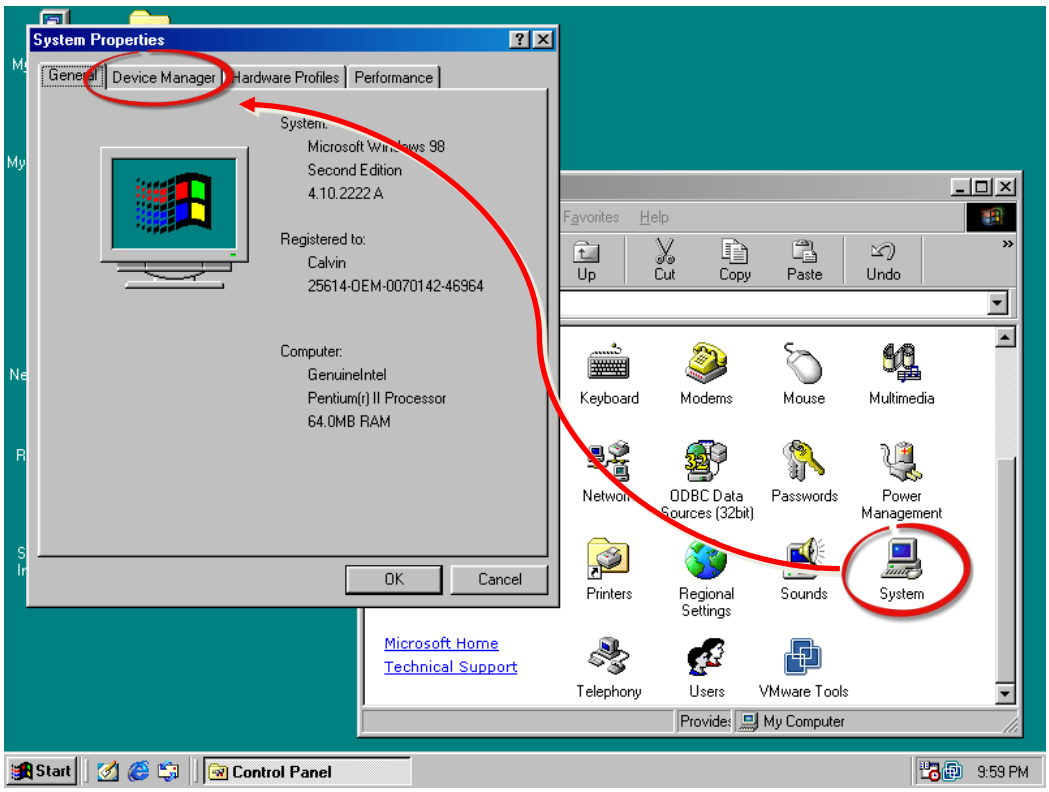
To verify that the driver was correctly installed, use the Windows **Device Manager** to view and update the device drivers installed on the computer, and to ensure that the hardware is operating correctly. The following is a description of how access the Device Manager in each of the major versions of Windows. Refer to the appropriate description for the specific operating system to verify the installation.

4.3.1 Accessing Windows Device Manager

- **Windows 95/98/ME**

Step 1: Either right-click the **"My Computer"** icon on the desktop and then click **"Properties"**, or open the **"Control Panel"** and double-click the **"System"** icon to open the System Properties dialog box.

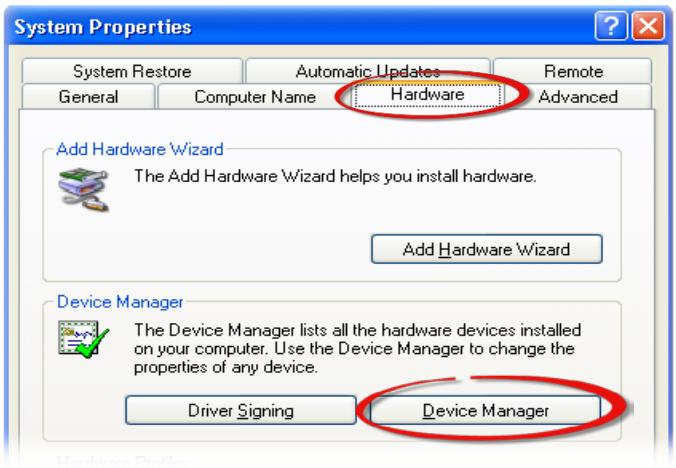
Step 2: In the **System Properties** dialog box, click the "**Device Manager**" tab.



➢ Windows 2000/XP

Step 1: Click the "**Start**" button and then point to "**Settings**" **and click** "**Control Panel**". Double-click the "**System**" icon to open the **"System Properties"** dialog box.

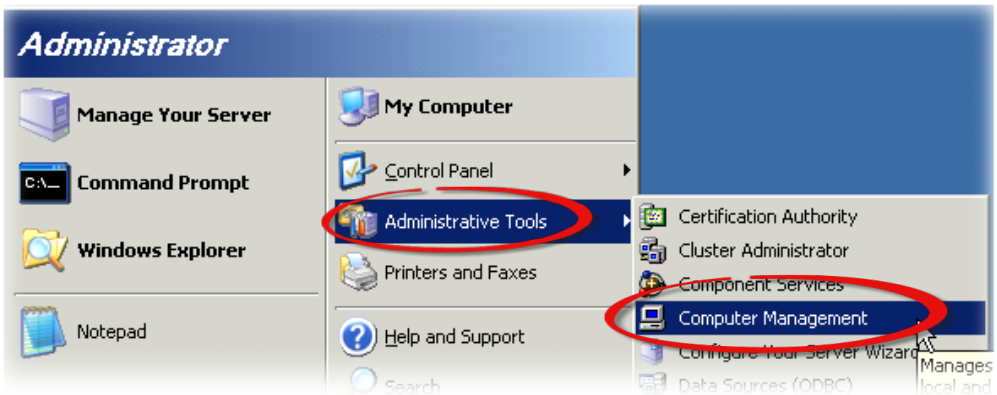
Step 2: Click the "**Hardware**" tab and then click the "**Device Manager**" button.



➢ Windows Server 2003

Step 1: Click the **"Start"** button and point to **"Administrative Tools"**, and then click the **"Computer Management"** option.

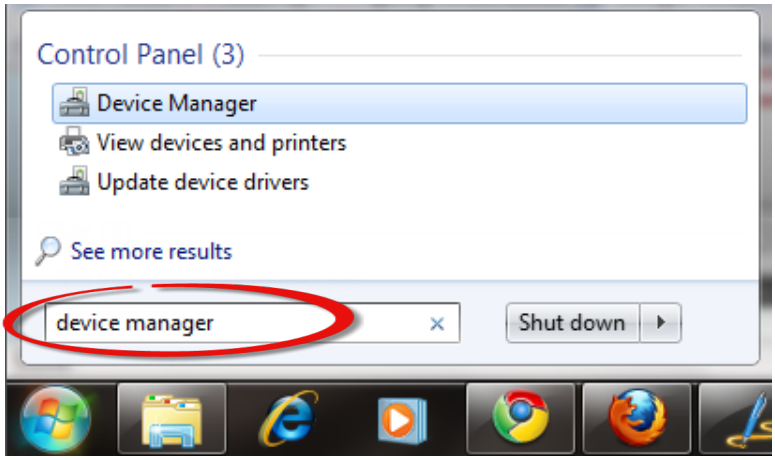
Step 2: Expand the **"System Tools"** item in the console tree, and then click **"Device Manager"**.



➢ Windows 7/10

Step 1: Click the **"Start"** button, and then click **"Control Panel"**.
Step 2: Click **"System and Maintenance"**, and then click **"Device Manager"**.

Alternatively,
Step 1: Click the **"Start"** button.
Step 2: In the **Search field,** type **Device Manager** and then press Enter.



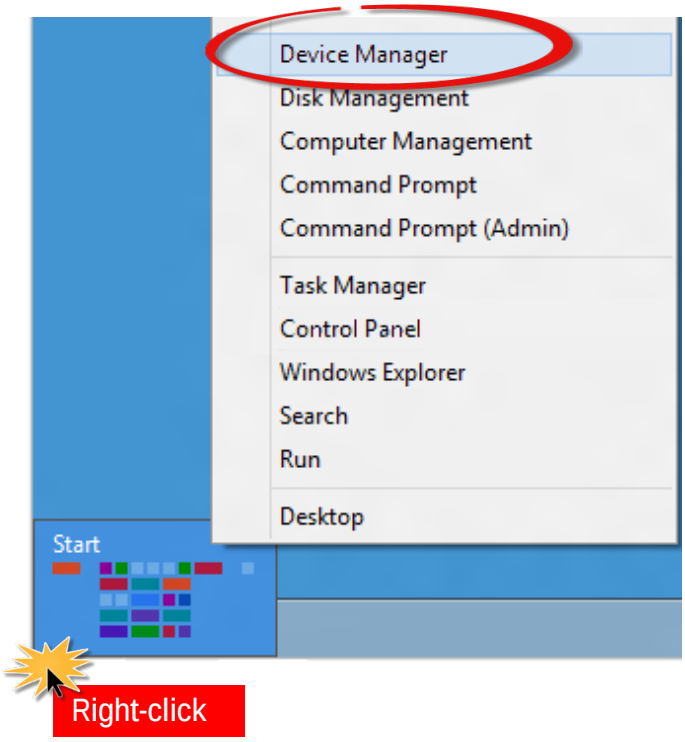
Note:

Administrator privileges are required for this operation. If you are prompted for an administrator password or confirmation, enter the password or provide confirmation by clicking the "Yes" button in the User Account Control message.

➢ Windows 8

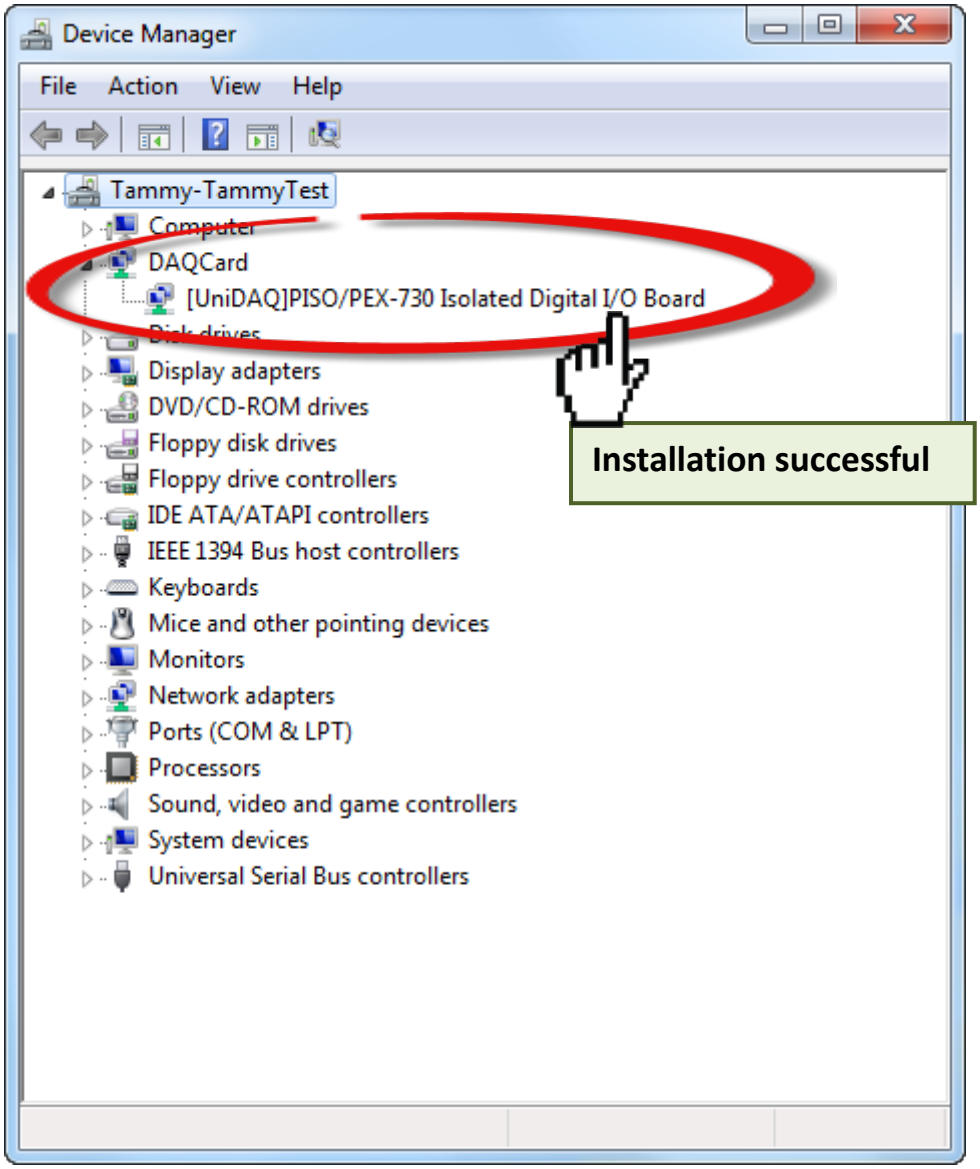
Step 1: To display the **Start screen icon** from the desktop view, hover the mouse cursor over the **bottom-left corner** of screen.
Step 2: **Right-click** the Start screen icon and then click "**Device Manager"**.

Alternatively, press **[Windows Key] +[X]** to open the Start Menu, and then select Device Manager from the options list.



4.3.2 Check that the Installation

Check that the PEX/PISO-730 series card is correctly listed in the **Device Manager** window, as illustrated below.



5. Board Testing

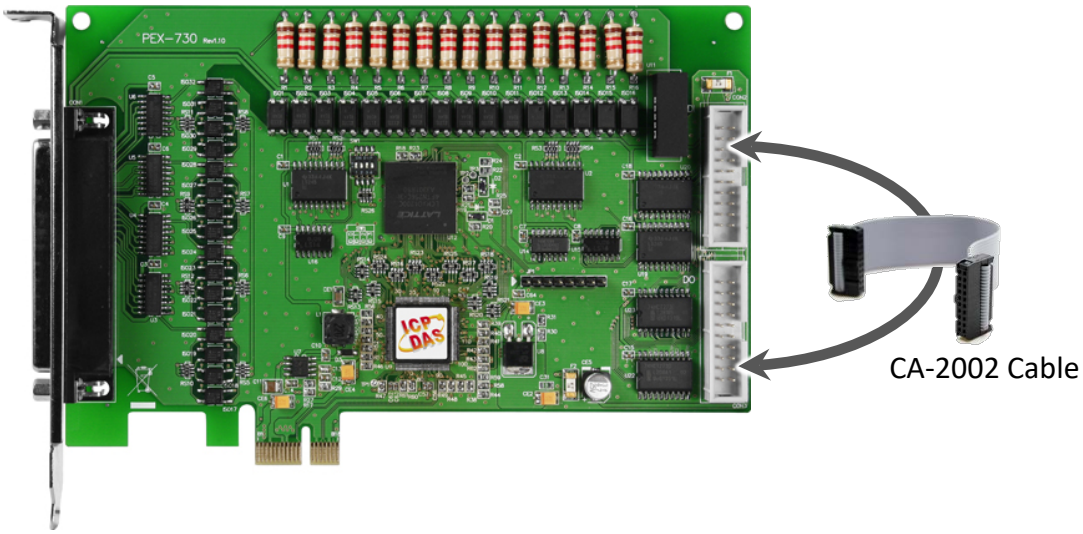
This chapter provides detailed information about the **"Self-Test"** process, which is used to confirm that the PEX/PISO-730 series card is operating correctly. Before beginning the **"Self-Test"** process, ensure that both the hardware and driver installation procedures are fully completed. For detailed information about the hardware and driver installation, refer to Chapter 3 "Hardware Installation" and Chapter 4 "Software Installation".

5.1 Self-test Wiring

5.1.1 Non-isolation DIO Test Wiring

Before beginning the "self-test", ensure that the following items are available:
☑ A CA-2002 (optional) cable
(Optional, Website: http://www.icpdas.com/products/Accessories/cable/cable_selection.htm)

Step 1: Use the CA-2002 cable to connect the CON2 with CON3 on your card.



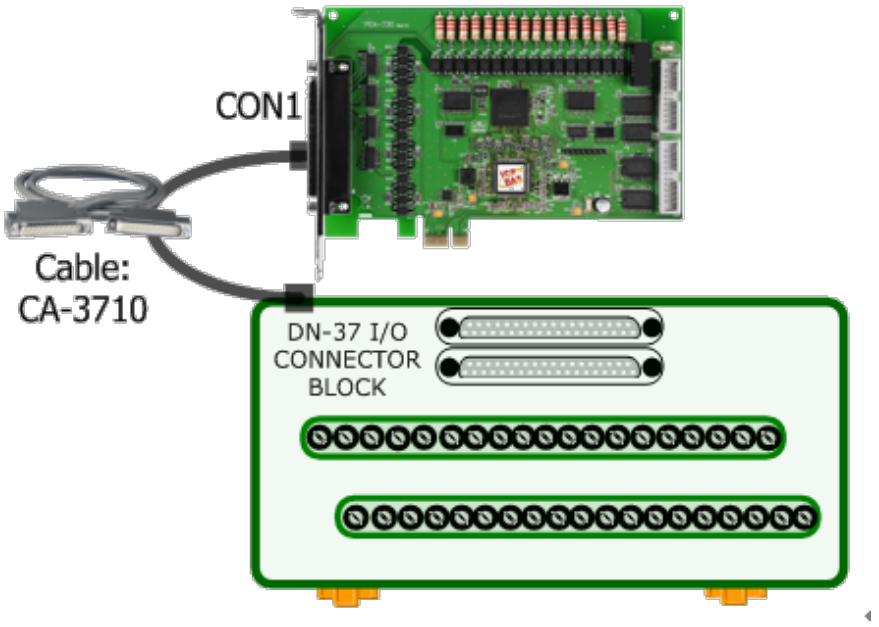
5.1.2 Isolation DIO Test Wiring

Before beginning the “self-test”, ensure that the following items are available:
☑ A DN-37 (optional) Terminal Board
(Optional, Website:
http://www.icpdas.com/root/product/solutions/pc_based_io_board/daughter_boards/dn-37.html)

☑ A CA-3710 (optional) Cable
(Optional, Website: http://www.icpdas.com/products/Accessories/cable/cable_selection.htm)

☑ Exterior power supply device, such as the DIN-KA52F or DP-660
(Optional, Website:
http://www.icpdas.com/root/product/solutions/accessories/power_supply/ka52f.html
http://www.icpdas.com/root/product/solutions/accessories/power_supply/dp-660.html)

Step 1: Connect the DN-37 to the CON1 connector on your card using the CA-3710 cable.



PEX/PISO-730 Series

Note:

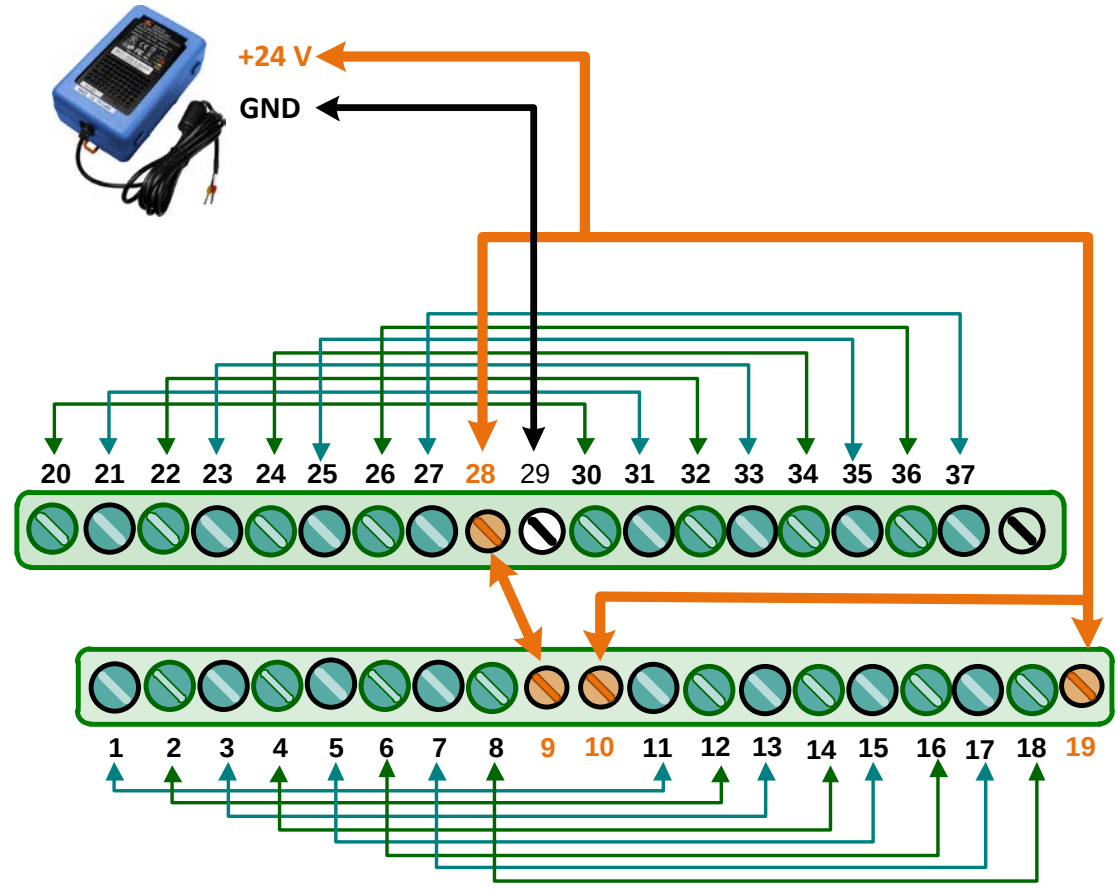
The DI can accept external voltages refer to Section 2.2.4 "Isolation DI Port Architecture (CON1)" for more detail information.

➢ The External Power (+9VDC to +30VDC) Wiring for PEX-730/PISO-730(U):

Step 2: Connect the **IDI (0-15)** with **IDO (0-15)**.
(**Pin1** connects to **Pin11** … … **Pin27** connects to **Pin37**)

Step 3: **External Power +24 V** connect to **EPWR1 (Pin10)** and **EPWR2 (Pin19)**.
External Power +24 V connect to **EI.COM1 (Pin09)** and **EI.COM2 (Pin28)**.
External Power GND connect to **IGND (Pin29)**.

External: DIN-KA52F

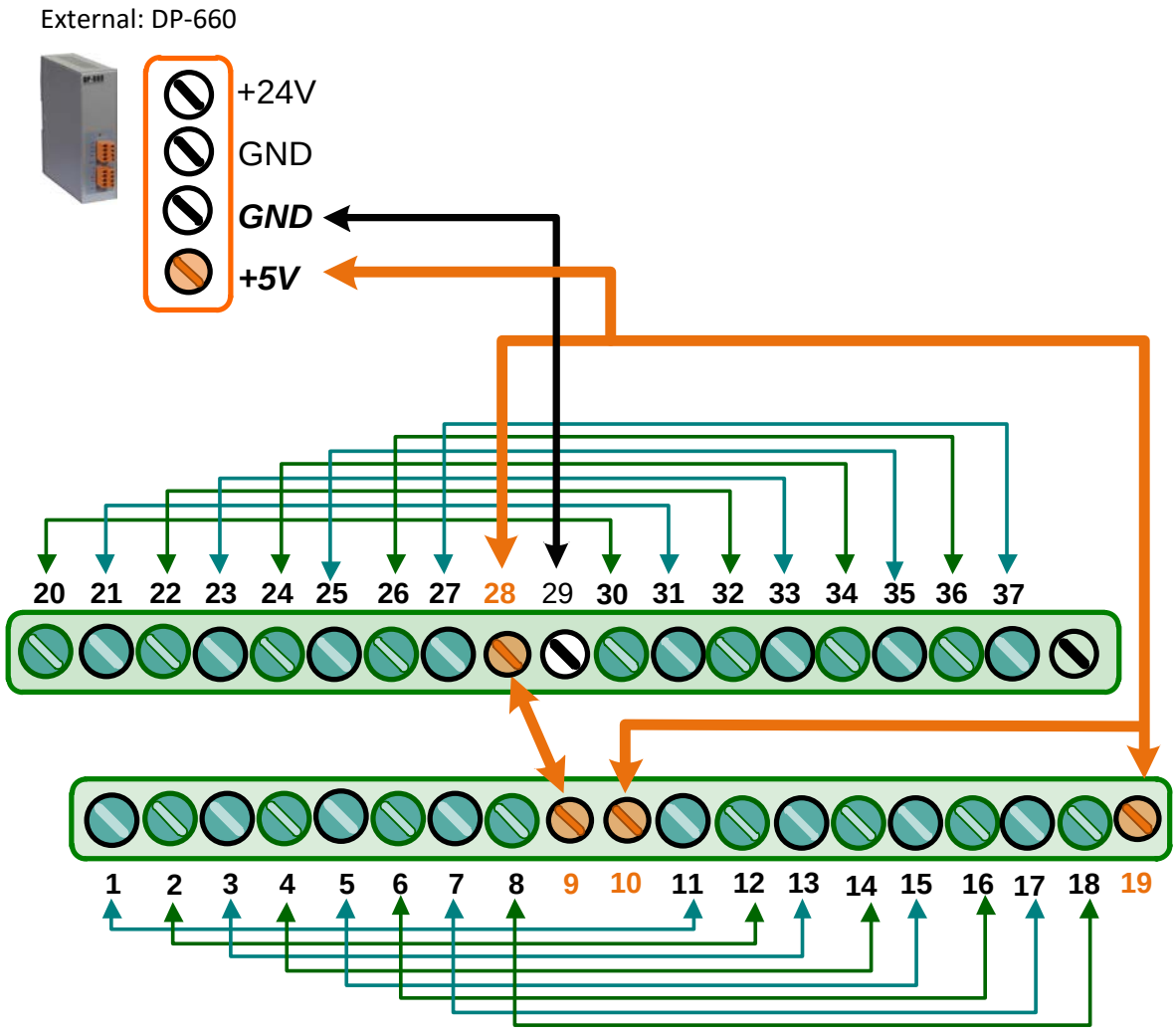


➢ The External Power (+5VDC to +12VDC) Wiring for PISO-730U-5V:

Step 2: Connect the **IDI (0-15)** with **IDO (0-15)**.
(**Pin1** connects to **Pin11** … … **Pin27** connects to **Pin37**)

Step 3: **External Power +5 V** connect to **EPWR1 (Pin10)** and **EPWR2 (Pin19)**.
External Power +5 V connect to **EI.COM1 (Pin09)** and **EI.COM2 (Pin28)**.
External Power GND connect to **IGND (Pin29)**.

External: DP-660



PEX/PISO-730A Series

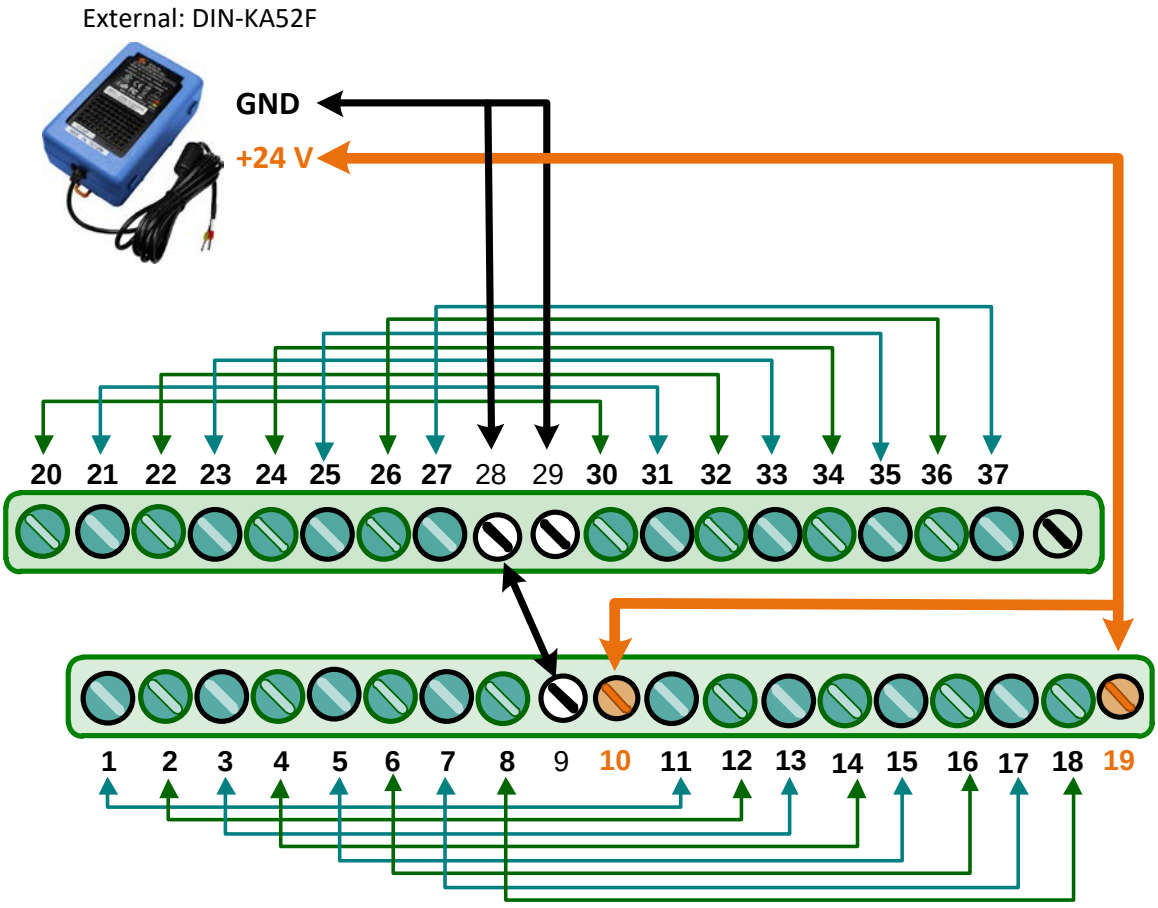
Note:

The DI can accept external voltages refer to Section 2.2.4 "Isolation DI Port Architecture (CON1)" for more detail information.

➢ The External Power (+9VDC to +30VDC) Wiring for PEX-730A/PISO-730A:

Step 2: Connect the **IDI (0-15)** with **IDO (0-15)**.
(**Pin1** connects to **Pin11** … … **Pin27** connects to **Pin37**)

Step 3: **External Power +24 V** connect to **EPWR1 (Pin10)** and **EPWR2 (Pin19)**.
External Power GND connect to **EI.COM1 (Pin09)**, **EI.COM2 (Pin28)** and **IGND (pin29)**.

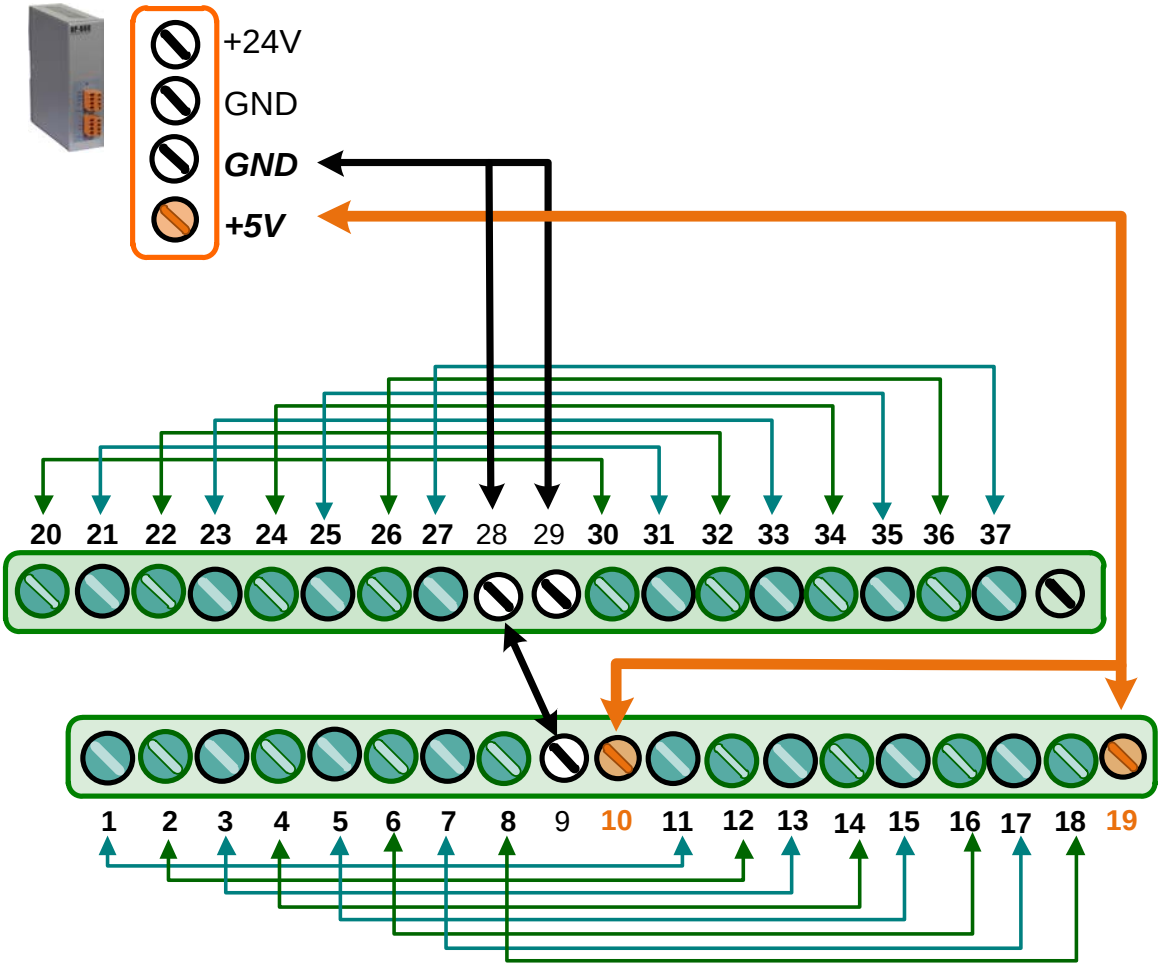


➢ The External Power (+5VDC to +12VDC) Wiring for PISO-730A-5V:

Step 2: Connect the **IDI (0-15)** with **IDO (0-15)**.
(**Pin1** connects to **Pin11** … … **Pin27** connects to **Pin37**)

Step 3: **External Power +5 V** connect to **EPWR1 (Pin10)** and **EPWR2 (Pin19)**.
External Power GND connect to **EI.COM1 (Pin09)**, **EI.COM2 (Pin28)** and **IGND (pin29)**.

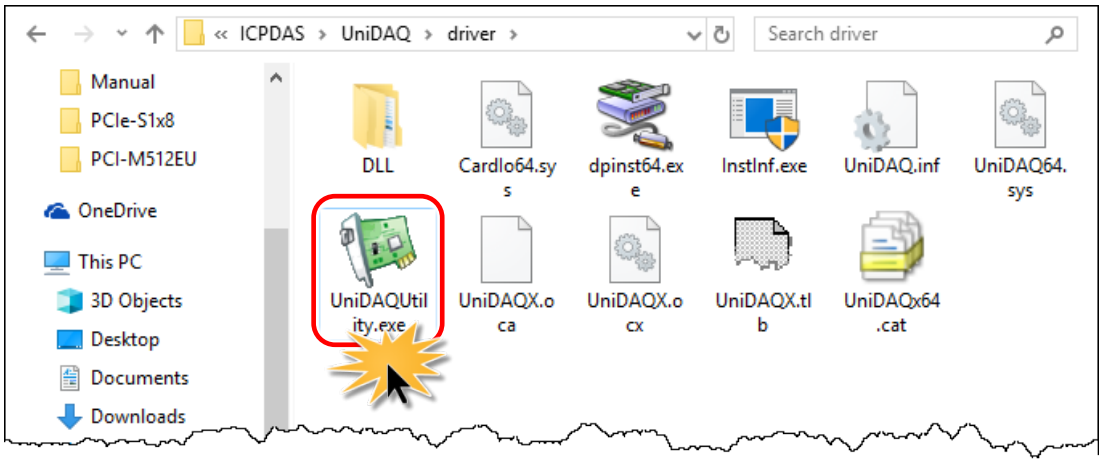
External: DP-660



5.2 Launch the Test Program

The following example use UniDAQ driver to perform self-test. If you install the PISO-DIO series classic driver, refer to Quick Start Guide of the PISO-730 series (http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/piso-dio/manual/quickstart/classic/) to execute the self-test.

Step 1: Double-click the **UniDAQ Utility** software. The UniDAQ Utility will be placed in the **default path "C:\ICPDAS\UniDAQ\Driver"** after completing installation.





Step 2: Confirm that your card has been successfully installed in the Host system. **Note that the device number starts from 0**.

Step 3: Click the **"TEST"** button to start the test.

Note:

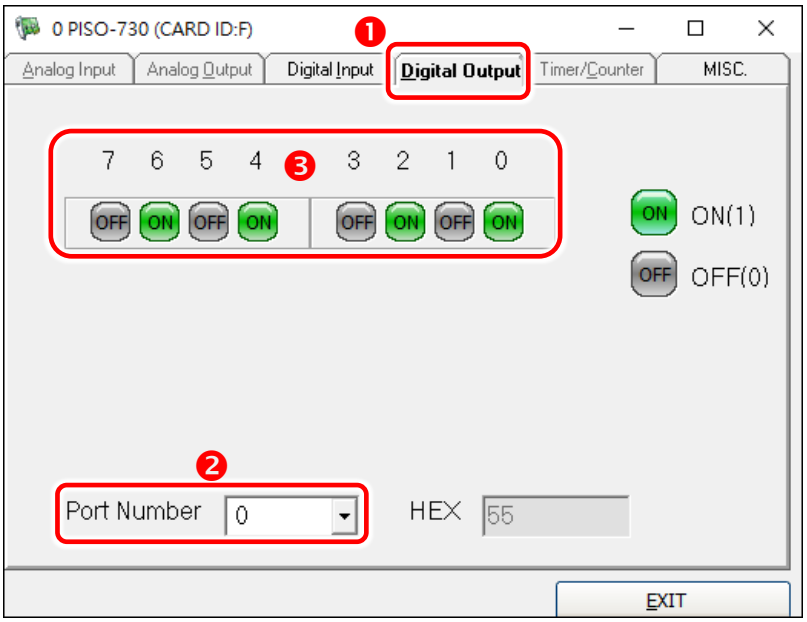
The PEX-730 software is fully compatible with the PISO-730 series software.

Step 4: Get DIO function test result.

1. Click the **"Digital Output"** tab.
2. Select the **"Port0"** from the **"Port Number"** drop-down options.

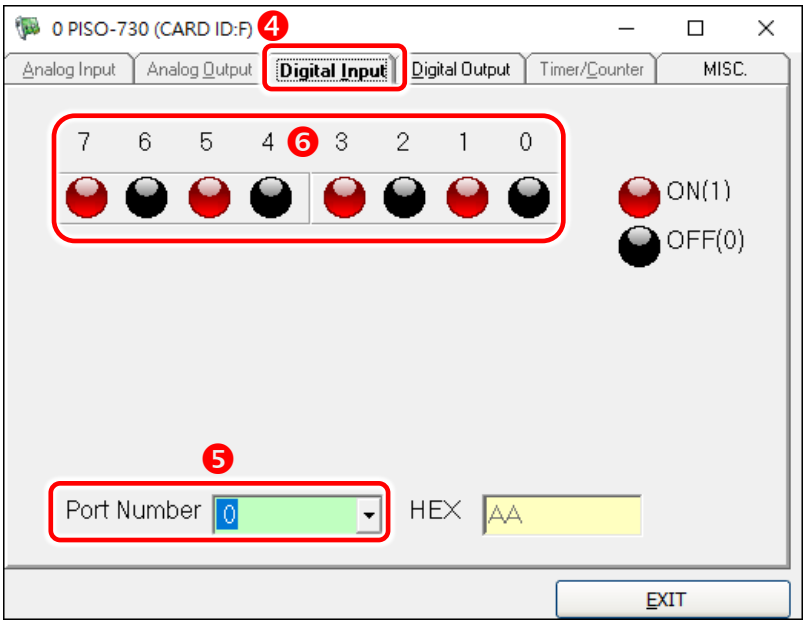
Note: Port0/1 is isolated Digital Input/Output (IDI/IDO), Port2/3 is non-isolated Digital Input/Output (DI/DO).

3. Check the checkboxes for **channels 0, 2, 4 and 6**.



4. Click the **"Digital Input"** tab.
5. Select the **"Port0"** from the **"Port Number"** drop-down options.
6. The DI indicators will turn **black** when the corresponding DO channel 0, 2, 4 and 6 are **ON**.

Note: Port0/1 IDI is the reverse logic, so the red light means low status (Logic 0) and the black light means high status (Logic 1).



6. I/O Control Register

6.1 How to Find the I/O Address

During the power-on stage, the Plug and Play BIOS will assign an appropriate I/O address to each PEX/PISO-730 series card installed in the system. Each board includes four fixed ID numbers that are used to identify the board, and are indicated below:

Table 6-1:

	PISO-730 (Rev 1.0 ~ 2.0)	PISO-730 (Rev 2.3)	PISO-730U (Rev 4.0)	PEX-730
Vendor ID	0xE159	0xE159	0xE159	0xE159
Device ID	0x02	0x01	0x01	0x01
Sub-Vendor ID	0x80	0xCA80	0xCA80	0xCA80
Sub-Device ID	0x08	0x00	0x00	0x00
Sub-Aux ID	0x40	0x40	0x40	0x40

Table 6-2:

	PISO-730A (Rev 2.0)	PISO-730A (Rev 3.3)	PEX-730A (Rev 1.3)
Vendor ID	0xE159	0xE159	0xE159
Device ID	0x02	0x01	0x01
Sub-Vendor ID	0x80	0x62FF	0x62FF
Sub-Device ID	0x08	0x00	0x00
Sub-Aux ID	0x80	0x80	0x80

We provide all necessary functions as follows:

1. **PIO_DriverInit**(&wBoard, wSubVendor, wSubDevice, wSubAux)
2. **PIO_GetConfigAddressSpace**(wBoardNo,*wBase,*wIrq, *wSubVendor,*wSubDevice, *wSubAux, *wSlotBus, *wSlotDevice)
3. **Show_PIO_PISO**(wSubVendor, wSubDevice, wSubAux)

All functions are defined in PISODIO.H. Refer to Section 6.3 "The I/O Address Map" for more information. The important driver information is given as follows:

Allocated resource information:

- **wBase :** BASE address mapping in this PC
- **wIrq:** Allocated IRQ channel number of this board in this PC

PIO/PISO identification information:

- **wSubVendor:** subVendor ID of this board
- **wSubDevice:** subDevice ID of this board
- **wSubAux:** subAux ID of this board

PC's physical slot information:

- **wSlotBus:** The bus number of the slot used by this board.
- **wSlotDevice:** The device number of the slot used by this board.

The PIO_PISO.EXE utility will detect and show all PIO/PISO cards installed in this PC. Refer to Section 6.1.1 "PIO_PISO.EXE Utility for Windows" for more information.

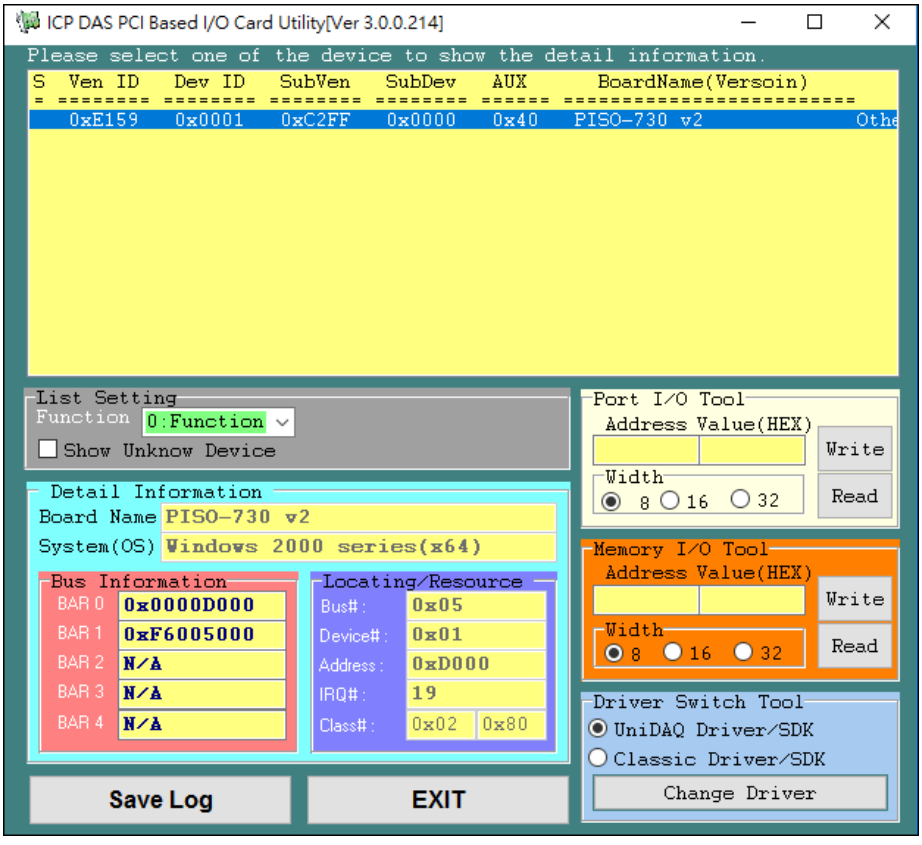
6.1.1 PIO_PISO.EXE Utility for Windows

The PIO_PISO.EXE is valid for all PIO/PISO cards. This program shows all PCI hardware ID regarding the PIO and PISO series DAQ cards. It is useful to test if the card Plug & Play successfully when the computer bootup. If the PIO or PISO series card does not shown in the screen correctly, please try to use another PCI slot and try again.

The user can execute the PIO_PISO.EXE to get the following information:

- List all PIO/PISO cards installed in this PC
- List all resources allocated to every PIO/PISO cards
- List the wSlotBus and wSlotDevice for specified PIO/PISO card identification. (refer to Section 6.2 “The Assignment of I/O Address” for more information about the assignment of I/O Address)

The **PIO_PISO.exe** utility is located on the CD as below and is useful for all PISO-DIO series boards. (CD:\NAPDOS\PCI\Utility\Win32\PIO_PISO\)
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/utility/win32/pio_piso/



6.1.2 PIO_PISO.EXE Utility for DOS

The **PIO_PISO.EXE for DOS** is contained in:



CD:\NAPDOS\PCI\Utility\DOS\



http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/utility/dos/

The PIO_PISO program source is given as follows:

```
/* ------------------------------------------------------------ */
/* Find all PIO_PISO series cards in this PC system */
/* step 1 : plug all PIO_PISO cards into PC */
/* step 2 : run PIO_PISO.EXE */
/* ------------------------------------------------------------ */

#include "PIO.H"

WORD wBase,wIrq;
WORD wBase2,wIrq2;

int main()
{
int    i,j,j1,j2,j3,j4,k,jj,dd,j11,j22,j33,j44;
WORD wBoards,wRetVal;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;
float ok,err;

clrscr();
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);      /*for PIO-PISO */
printf("\nThrer are %d PIO_PISO Cards in this PC",wBoards);
if (wBoards==0 ) exit(0);

printf("\n-----------------------------------------------------");
for(i=0; i<wBoards; i++)
     {
     PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,
                    &wSubDevice,&wSubAux,&wSlotBus,&wSlotDevice);

     printf("\nCard_%d:wBase=%x,wIrq=%x,subID=[%x,%x,%x],
                    SlotID=[%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,
                    wSubAux,wSlotBus,wSlotDevice);
     printf(" --> ");
     ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
     }

PIO_DriverClose();
}
```

6.2 The Assignment of I/O Address

The Plug and Play BIOS will assign the proper I/O address to a PIO/PISO series card. If there is only one PIO/PISO card, the user can identify the card as card_0. If there are two PIO/PISO card s in the system, it is very difficult to identify which board is card_0. The software driver can support a maximum of 16 cards. Therefore, the user can install 16 PIO/PSIO series cards onto one PC system. The methods used to find and identify card_0 and card_1 is demonstrated below.

The simplest way to identify which card is card_0 is to use wSlotBus and wSlotDevice in the following manner:

Step 1: Remove all PEX/PISO-730 series card from the PC.

Step 2: Install one PEX/PISO-730 series card into the PC's PCI_slot1, run **PIO_PISO.EXE**. Then record the **"wSlotBus1"** and **"wSlotDevice1"** information in the **"Locating/Resource"** area.

Step 3: Remove all PEX/PISO-730 series card from the PC.

Step 4: Install one PEX/PISO-730 series card into the PC's PCI_slot2 and run PIO_PISO.EXE. Then record the **"wSlotBus1"** and **"wSlotDevice1"** information in the **"Locating/Resource"** area.

Step 5: Repeat **Steps(3) and (4)** for every PCI_slot and record all information from **"wSlotBus1"** and **"wSlotDevice1"**.

The records may look similar to the table follows:

Table 6-3

PC's PCI Slot	Locating/Resource	
	wSlotBus (Bus#)	wSlotBus (Device#)
Slot_1	0	0x07
Slot_2	0	0x08
Slot_3	0	0x09
Slot_4	0	0x0A
PCI-BRIDGE		
Slot_5	1	0x0A
Slot_6	1	0x08
Slot_7	1	0x09
Slot_8	1	0x07

The above procedure will record all the "wSlotBus" and "wSlotBus" information on a PC. These values will be mapped to this PC's physical slot and this mapping will not be changed for any PIO/PISO cards. Therefore, this information can be used to identify the specified PIO/PISO card by following steps:

Step1: Using the "wSlotBus" and "wSlotDevice" information from Table 6-4.

Step2: Enter the board number into PIO_GetConfigAddressSpace(…) function to get the information for a specific card, especially the "wSlotBus" and "wSlotDevice" details.

Step3: Identify the specific PIO/PISO card by comparing the data of the "wSlotBus" and "wSlotDevice" from Step1 and Step2.

Note:

Normally the card installed in slot 0 is card0 and the card installed in slot1 is card1 for PIO/PISO series cards.

6.3 The I/O Address Map

The I/O address of the PIO/PISO series card is automatically assigned by the main board ROM BIOS. The I/O address can also be re-assigned by the user, but it is strongly recommended that the I/O address is not changed by user. The Plug and Play BIOS will assign an appropriate I/O address to each PIO/PISO series card. The I/O addresses of the PEX/PISO-730 series card are as follows, and are based on the base address of each card.

Address	Read	Write
wBase+0x0	RESET\ Control Register	RESET\ Control Register
wBase+0x2	AUX Control Register	AUX Control Register
wBase+0x3	AUX Data Register	AUX Data Register
wBase+0x5	INT Mask Control Register	INT Mask Control Register
wBase+0x7	AUX Pin Status Register	AUX Pin Status Register
wBase+0x2a	INT Polarity Control Register	INT Polarity Control Register
wBase+0xc0	IDI0 ~ IDI7	IDO0 ~ IDO7
wBase+0xc4	IDI8 ~ IDI15	IDO8 ~ IDO15
wBase+0xc8	DI0 ~ DI7	DO0 ~ DO7
wBase+0xcc	DI8 ~ DI15	DO8 ~ DO15
wBase+0xd0	Read IDO 0 ~ IDO7 Readback	-
wBase+0xd4	Read IDO 8 ~ IDO15 Readback	-
wBase+0xd8	Read DO 0 ~ DO7 Readback	-
wBase+0xdc	Read DO 8 ~ DO15 Readback	-
wBase+0xf0	Read Card ID	-
wBase+0xfc	Read version number	-

Note:

Refer to Section 6.1 “How to Find the I/O Address” for more information about wBase.

6.3.1 RESET\ Control Register

(Read/Write): wBase+0x0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	RESET\

When the PC is first powered-up, the RESET\ signal is in Low state. This disables all DIO operations. Please set the RESET\ signal to High state before issuing any DIO command.

```
outportb(wBase,1);      /* RESET\ = High → all D/I/O are enable now */
outportb(wBase,0);      /* RESET\ = Low  → all D/I/O are disable now */
```

6.3.2 AUX Control Register

(Read/Write): wBase+0x2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

Aux?=0→ this Aux is used as a DI
Aux?=1→ this Aux is used as a DO

When the PC is first powered-on, all Aux? signals are in Low-state. All Aux? are designed as DI for all PIO/PISO series. Please set all Aux? in DI state.

6.3.3 AUX Data Register

(Read/Write): wBase+0x3

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

When the Aux? is used as a DO, this register controls the output state. This register has been designed for future extensions, so please don try to control this register now.

6.3.4 INT Mask Control Register

(Read/Write): wBase+0x5

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	EN1	EN2

EN0/1=0 → Disables INT_CHAN_0/1 as a interrupt signal (default)
EN0/1=1 → Enables INT_CHAN_0/1 as a interrupt signal

For example:

```
outportb(wBase+5,0);        // disable all interrupts
outportb(wBase+5,1);        // enable interrupt of INT_CHAN_0
outportb(wBase+5,2);        // enable interrupt of INT_CHAN_1
outportb(wBase+5,3);        // enable all two channels of interrupt
```

Refer to the following demo program for more information:

Section 7.2.3 “DEMO3.C” → For INT_CHAN_0 only (initial high state)
Section 7.2.4 “DEMO4.C” → For INT_CHAN_0 only (initial low state)
Section 7.2.5 “DEMO5.C” → For multi-channel interrupts sources

6.3.5 AUX Status Register

(Read/Write): wBase+0x7

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

Aux0=INT_CHAN_0, Aux1=INT_CHAN_1, Aux7~4=Aux-ID. Refer to Section 6.1 "How to Find the I/O Address" for more information. The Aux0~1 are used as interrupt sources. The interrupt service routine has to read this register for interrupt source identification. Refer to Section 2.3 "Interrupt Operation" for more information.

6.3.6 Interrupt Polarity Control Register

(Read/Write): wBase+0x2a

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	-	-	INV1	INV0

INV0/1=0 → Selects the inverted signal from INT_CHAN_0/1
INV0/1=1 → Selects the non-inverted signal from INT_CHAN_0/1

For example:

```
outportb(wBase+0x2a,0);  /* select the inverted input from all 2 channels      */
outportb(wBase+0x2a,3);  /* select the non-inverted input from all 2 channels */
outportb(wBase+0x2a,2);  /* select the inverted input of INT_CHAN_0            */
                         /* select the non-inverted input of INT_CHAN_1       */
```

Refer to Section 2.3 "Interrupt Operation" for more information.
Refer to Section 7.2.5 "DEMO5.C" for more information.

6.3.7 I/O Data Register

(Read/Write): wBase+0xc0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IDI7	IDI6	IDI5	IDI4	IDI3	IDI2	IDI1	IDI0

(Read/Write): wBase+0xc4

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IDI15	IDI14	IDI13	IDI12	IDI11	IDI10	IDI9	IDI8

(Read/Write): wBase+0xc8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0

(Read/Write): wBase+0xcc

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8

For example:

```
outportb(wBase+0xc0,0xff);          /* Writes 0xff to IDO0 ~ IDO7    */
DiValue=inportb(wBase+0xc0);        /* Reads states from IDI0 ~ IDI7 */

outportb(wBase+0xc8,0x55);          /* Writes 0x55 to DO0 ~ DO7    */
DiValue=inportb(wBase+0xcc);        /* Reads states from DI8 ~ DI15    */
```

6.3.8 DO Readback Register

(Read): wBase+0xd0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IDO7	IDO6	IDO5	IDO4	IDO3	IDO2	IDO1	IDO0

(Read): wBase+0xd4

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IDO15	IDO14	IDO13	IDO12	IDO11	IDO10	IDO9	IDO8

(Read): wBase+0xd8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

(Read): wBase+0xdc

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8

For example:

```
DiValue=inportb(wBase+0xd0);          /* Reads states from IDO0~IDO7 */
DiValue=inportb(wBase+0xd4);          /* Reads states from IDO8~IDO15  */
DiValue=inportb(wBase+0xd8);          /* Reads states from DO0~DO7  */
DiValue=inportb(wBase+0xdc);          /* Reads states from DO8~DO15  */
```

Note:

The DO Readback function is only supported by the PISO-730U(-5V), PEX-730A and PEX-730.

6.3.9 Card ID Register

(Read): wBase+0xf0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	ID3	ID2	ID1	ID0

For example:

```
wCardID = 0x0F & inportb(wBase+0xf0);          /* read Card ID */
```

Note:

The Card ID function is only supported by the PISO-730U(-5V), PEX-730A and PEX-730.

6.3.10 Version Number Register

(Read): wBase+0xfc

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Ver7	Ver6	Ver5	Ver4	Ver3	Ver2	Ver1	Ver0

For example:

```
Ver_No = inportb(wBase+0xfc);          /* read version number*/
```

Note:

The version number function is only supported by the PISO-730U(-5V), PEX-730A and PEX-730.

7. Demo Programs

7.1 Demo Program for Windows

All demo programs will not work properly if the DLL driver has not been installed correctly. During the DLL driver installation process , the install-shields will register the correct kernel driver to the operation system and copy the DLL driver and demo programs to the correct position based on the driver software package you have selected (32/64-bit Windows XP/2003/2008/7/8/10). Once driver installation is complete, the related demo programs and development library and declaration header files for different development environments will be presented as follows.

➢ Demo Program for PISO-DIO Series Classic Driver

The demo program is contained in:

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

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

For detailed information about the DLL function of the PISO-730 series card, please refer to PISO-DIO DLL Software Manual (CD:\NAPDOS\PCI\PISO-DIO\Manual\)

➢ Demo Program for UniDAQ Driver

The demo program is contained in:



CD:\NAPDOS\PCI\UniDAQ\DLL\Demo\



http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/dll/demo/

- BCB6 → for Borland C++ Builder 6
  UniDAQ.H → Header files
  UniDAQ.LIB → Linkage library for BCB only
- Delphi6 → for Delphi 6
  UniDAQ.PAS → Declaration files
- VB6 → for Visual Basic 6
  UniDAQ.BAS → Declaration files
- CSharp2005 → for C#.NET2005
  UniDAQ.cs → Visual C# Source files
- VC6 → for Visual C++ 6
  UniDAQ.H → Header files
  UniDAQ.LIB → Linkage library for VC only
- VB.NET2005 → for VB.NET2005
  UniDAQ.vb → Visual Basic Source files
- VC.NET2005 → for VC.NET2005 (32-bit)
  UniDAQ.H → Header files
  UniDAQ.LIB → Linkage library for VC only
- VC.NET2005 → for VC.NET2005 (64-bit)
  UniDAQ.H → Header files
  UniDAQ.LIB → Linkage library for VC only

For detailed information about the DLL function and demo program of the UniDAQ, please refer to UniDAQ DLL Software Manual (CD:\NAPDOS\PCI\UniDAQ\Manual\)

7.2 Demo Program for DOS

The demo program is contained in:



- **For PEX-730, PISO-730U and PISO-730(-5V):**
  CD:\NAPDOS\PCI\PISO-DIO\DOS\730\

- **For PISO-730A(-5V):**
  CD:\NAPDOS\PCI\PISO-DIO\DOS\730a\



- **For PEX-730, PISO-730U and PISO-730(-5V):**
  http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/piso-dio/dos/730/

- **For PISO-730A(-5V):**
  http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/piso-dio/dos/730a/

- \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*.* → TC Large Model Library
- \TC\LIB\Huge*.* → TC Huge Model Library File
- \TC\LIB\Large\PIO.H → TC Declaration File
- \TC\LIB\Large\TCPIO_L.LIB → TC Large Model Library File
- \TC\LIB\Huge\PIO.H → TC Declaration 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\PIO.H → MSC Declaration 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\PIO.H → BC Declaration File
- \BC\LIB\Huge\BCPIO_H.LIB → BC Huge Model Library File

For detailed information about the DLL function of the DOS, please refer to PISO-DIO DLL Software Manual (CD:\NAPDOS\PCI\PISO-DIO\Manual\)

7.2.1 Demo1: DO Demo

```
/* ---------------------------------------------------------------------------------------------------------- */
/* DEMO1.C: D/O demo                                                                                          */
/* step 1: connect CON3 to DB-16R                                                                             */
/* step 2: run DEMO1.EXE                                                                                      */
/* ---------------------------------------------------------------------------------------------------------- */


#include "PIO.H"
void piso_730_do(long lDoValue);
void piso_730_ido(long lDoValue);
WORD wBase,wIrq;
int main()
{
int    i,j,k1,k2,l1,l2,jj,dd,j1,i1,j2,i2;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
long lOutPad1,lOutPad2;
char c;

clrscr();
/* step 1: find address-mapping of PIO/PISO cards                                  */
wRetVal=PIO_DriverInit(&wBoards,0x80,0x08,0x40); /* for PEX-730/PISO-730U/PISO-730(-5V) */
                                                 ,0x80); /* for PISO-730A(-5V) */
printf("\nThere are %d PISO-730/730A Cards in this PC",wBoards);
if (wBoards==0) exit(0);
printf("\n---------- The Configuration Space ----------");
for(i=0; i<wBoards; i++)
     {
     PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,&wSubDevice,
     &wSubAux,&wSlotBus,&wSlotDevice);
     printf("\nCard_%d: wBase=%x,wIrq=%x,subID=[%x,%x,%x],SlotID=
               [%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,wSubAux,
               wSlotBus,wSlotDevice);
     printf(" --> ");
     ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
      }
```

```
PIO_GetConfigAddressSpace(0,&wBase,&wIrq,&t1,&t2,&t3,&t4,&t5);
/* step 2: enable all D/I/O port                                    */
outportb(wBase,1);                                    /* enable D/I/O */

printf("\n\n");
lOutPad1=1;
lOutPad2=0x8000;
for(;;)
    {
    gotoxy(1,6);
    piso_730_do(lOutPad1);
    printf("\nOutput DO[0..15] = [%4lx]",lOutPad1);
    piso_730_ido(lOutPad2);
    printf("\nOutput IDO[0..15] = [%4lx]",lOutPad2);
    delay(12000);
    lOutPad1=((lOutPad1<<1)&0xffff);
    lOutPad2=((lOutPad2>>1)&0xffff);

    if (lOutPad1==0) {lOutPad1=1;lOutPad2=0x8000;}
    if (kbhit()!=0) break;
    }
PIO_DriverClose();
}

/* ------------------------------------------------------------ */
void piso_730_do(long lDoValue)
{
outportb(wBase+0xc8,(lDoValue&0xff));
outportb(wBase+0xcc,((lDoValue>>8)&0xff));
}
/* ------------------------------------------------------------ */
void piso_730_ido(long lDoValue)
{
outportb(wBase+0xc0,(lDoValue&0xff));
outportb(wBase+0xc4,((lDoValue>>8)&0xff));
}
```

7.2.2 Demo2: DIO Demo

```
/* ---------------------------------------------------------------------------------------------------- */
/* DEMO2.C: D/I/O demo                                                                                  */
/* step 1: connect DO[0..15] to DI[0..15],                                                              */
/*                      IDO[0..15] to IDI[0..15]                                                        */
/* step 2: run DEMO2.EXE                                                                                */
/* ---------------------------------------------------------------------------------------------------- */

#include "PIO.H"
long piso_730_di(void);
long piso_730_idi(void);
WORD wBase,wIrq;
int main()
{
int    i,j,k,k1,k2,l1,l2,jj,dd,j1,i1,j2,i2;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
long lOutPad1,lOutPad2,lInPad1,lInPad2;
char c;
clrscr();
/* step 1: find address-mapping of PIO/PISO cards                                    */
                .
                .
/* step 2: enable all D/I/O port                                                     */
outportb(wBase,1);              /* enable D/I/O */
lOutPad1=0x0001;
lOutPad2=0x8000;
for(;;)
    {gotoxy(1,8);
    piso_730_do(lOutPad1);
    lInPad1=piso_730_di();
    piso_730_ido(lOutPad2);
    delay(10000);
    lInPad2=piso_730_idi();
    printf("\n DO[0..15]=[%4lx] ,     DI[0..15]=[%4lx]",lOutPad1,lInPad1);
    printf("\nIDO=[%4lx],!IDI=[%4lx]",lOutPad2,(~lInPad2&0xffff));
    lOutPad1=(lOutPad1<<1)&0xffff;
    lOutPad2=(lOutPad2>>1)&0xffff;
    if (lOutPad1==0) lOutPad1=1;
    if (lOutPad2==0) lOutPad2=0x8000;
    if (kbhit()!=0) break;
    }
PIO_DriverClose();
}
```

```
/* ---------------------------------------------------------- */
long piso_730_di(void)
{
long lDiValue;
lDiValue=(inportb(wBase+0xcc)<<8);
lDiValue=(lDiValue|(inportb(wBase+0xc8)))&0xffff;
return(lDiValue);
}

/* ---------------------------------------------------------- */
long piso_730_idi(void)
{
long lDiValue;
lDiValue=(inportb(wBase+0xc4)<<8);
lDiValue=(lDiValue|(inportb(wBase+0xc0)))&0xffff;
return(lDiValue);
}
```

7.2.3 Demo3: Interrupt (DI0 initial high)

```
/* ---------------------------------------------------------------------------------------- */
/* DEMO3.C: interrupt (DI0 initial high)                                                    */
/* step 1: DI0 to function generator                                                        */
/* step 2: run DEMO3.EXE                                                                    */
/* ---------------------------------------------------------------------------------------- */

#include "PIO.H"
#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI         0x20

WORD init_high();
void interrupt (*oldfunc) ();
static void interrupt irq_service();
int COUNT_L,COUNT_H,irqmask,now_int_state;

void piso_730_do(long lDoValue);
long piso_730_di(void);
void piso_730_ido(long lDoValue);
long piso_730_idi(void);

WORD wBase,wIrq;

int main()
{
int    i,j,k,k1,k2,l1,l2,jj,dd,j1,i1,j2,i2;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();
/* step 1: find address-mapping of PIO/PISO cards                                          */
                                .
                                .
/* step 2: enable all D/I/O port                                                    */
outportb(wBase,1);      /* enable D/I/O */
init_high();
printf("\n\n***** show the count of Low_pulse *****\n");
for(;;)
     {
     gotoxy(1,8);
     printf("\nCOUNT_L=[%5d]",COUNT_L);
     if (kbhit()!=0) break;
      }
disable();
```

```
outportb(wBase+5,0);                /* disable all interrupt */
if (wIrq<8)
     {
     setvect(wIrq+8,oldfunc);
     }
else
     {
     setvect(wIrq-8+0x70,oldfunc);
     }
PIO_DriverClose();
}
/* ------------------------------------------------------------ */
WORD init_high()
{
DWORD dwVal;

disable();
outportb(wBase+5,0);                /* disable all interrupt */

if (wIrq<8)
     {
     oldfunc=getvect(wIrq+8);
     irqmask=inportb(A1_8259+1);
     outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
     setvect(wIrq+8, irq_service);
     }
else
     {
     oldfunc=getvect(wIrq-8+0x70);
     irqmask=inportb(A1_8259+1);
     outportb(A1_8259+1,irqmask & 0xfb);     /* IRQ2 */
     irqmask=inportb(A2_8259+1);
     outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
     setvect(wIrq-8+0x70, irq_service);
     }

outportb(wBase+0x2a,0);             /* invert DI0            */

now_int_state=0x1;                   /* now DI0 is high      */
outportb(wBase+5,0x1);              /* enable DI0 interrupt       */
enable();
}
```

7.2.4 Demo4: Interrupt (DI0 initial low)

```
/* ---------------------------------------------------------------------------------------------- */
/* DEMO4.C: Interrupt (DI0 initial low)                                                           */
/* step 1: DI0 to function generator                                                              */
/* step 2: run DEMO4.EXE                                                                          */
/* ---------------------------------------------------------------------------------------------- */

#include "PIO.H"
#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init_low();
void interrupt (*oldfunc) ();
static void interrupt irq_service();
int COUNT_L,COUNT_H,irqmask,now_int_state;

void piso_730_do(long lDoValue);
long piso_730_di(void);
void piso_730_ido(long lDoValue);
long piso_730_idi(void);

WORD wBase,wIrq;

int main()
{
int    i,j,k,k1,k2,l1,l2,jj,dd,j1,i1,j2,i2;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();
/* step 1: find address-mapping of PIO/PISO cards                    */
                .
                .
/* step 2: enable all D/I/O port                    */
outportb(wBase,1);              /* enable D/I/O */

init_Low();

printf("\n\n***** show the count of High_pulse *****\n");
for(;;)
    {
    gotoxy(1,8);
    printf("\nCOUNT_H=[%5d]",COUNT_H);
    if (kbhit()!=0) break;
    }
disable();
```

```
outportb(wBase+5,0);                    /* disable all interrupt */
if (wIrq<8)
     {
     setvect(wIrq+8,oldfunc);
     }
else
     {
     setvect(wIrq-8+0x70,oldfunc);
     }
PIO_DriverClose();
}
/* ------------------------------------------------------------ */
WORD init_low()
{
DWORD dwVal;

disable();
outportb(wBase+5,0);                      /* disable all interrupt */

if (wIrq<8)
     {
     oldfunc=getvect(wIrq+8);
     irqmask=inportb(A1_8259+1);
     outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
     setvect(wIrq+8, irq_service);
     }
else
     {
     oldfunc=getvect(wIrq-8+0x70);
     irqmask=inportb(A1_8259+1);
     outportb(A1_8259+1,irqmask & 0xfb);      /* IRQ2 */
     irqmask=inportb(A2_8259+1);
     outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
     setvect(wIrq-8+0x70, irq_service);
     }
outportb(wBase+0x2a,1);                        /* non-invert DI0                */
now_int_state=0x0;                              /* now DI0 is low                */
outportb(wBase+5,0x1);                         /* enable DI0 interrupt        */
enable();
}
/* ------------------------------------------------------------ */
```

```
void interrupt irq_service()
{
if (now_int_state==1)                  /* now DI0 change to low              */
    {                                    /* INT_CHAN_0 = !DI0                  */
    COUNT_L++;                           /* find a low pulse (DI0)             */
    if ((inportb(wBase+7)&1)==0)    /* DI0 still fixed in low             */
        {                                /* need to generate a high pulse */
        outportb(wBase+0x2a,1);      /* INV0 select noninverted input */
        now_int_state=0;              /* now DI0=low                        */
        }
    else now_int_state=1;             /* now DI0=High                       */
    }
else                                   /* now DI0 change to high             */
    {                                    /* INT_CHAN_0 = DI0                   */
    COUNT_H++;                           /* find a high pulse (DI0)            */
    if ((inportb(wBase+7)&1)==1)    /* DI0 still fixed in high            */
        {                                /* need to generate a high pulse */
        outportb(wBase+0x2a,0);      /* INV0 select inverted input       */
        now_int_state=1;              /* now DI0=high                       */
        }
    else now_int_state=0;             /* now DI0=low                        */
    }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

7.2.5 Demo5: Interrupt (Multi interrupt source)

```
/* ---------------------------------------------------------------------------------------------- */
/* DEMO5.C: Interrupt (Multi interrupt source)                                                    */
/*                DI0 : initial low ,    DI1 : initial high                                       */
/*                                                                                                */
/* step 1: connect DI0 & DI1 to function generator                                                */
/* step 2: run DEMO5.EXE                                                                          */
/* ---------------------------------------------------------------------------------------------*/

#include "PIO.H"
#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI         0x20

WORD init();
void interrupt (*oldfunc) ();
static void interrupt irq_service();
int    irqmask,now_int_state,new_int_state,invert,int_c,int_num;
int    CNT_L1,CNT_L2,CNT_H1,CNT_H2;

WORD wBase,wIrq;

int main()
{
int    i,j,k;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();
/* step 1: find address-mapping of PIO/PISO cards              */
                .
                .
/* step 2: enable all D/I/O port             */
outportb(wBase,1);              /* enable D/I/O */
init();
printf("\n\n***** show the count of High_pulse *****\n");
for(;;)
     {
     gotoxy(1,8);
     printf("\nCNT_L1,CNT_L2=[%5d,%5d]",CNT_L1,CNT_L2);
     printf("\nCNT_H1,CNT_H2=[%5d,%5d]",CNT_H1,CNT_H2);
     if (kbhit()!=0) break;
     }
```

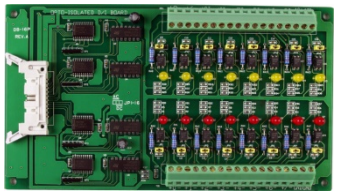
```
disable();
outportb(wBase+5,0);                    /* disable all interrupt */
if (wIrq<8)
    {
    setvect(wIrq+8,oldfunc);
    }
else
    {
    setvect(wIrq-8+0x70,oldfunc);
    }
PIO_DriverClose();
}

/* ------------------------------------------------------------ */
WORD init()
{
DWORD dwVal;
disable();
outportb(wBase+5,0);                    /* disable all interrupt */
if (wIrq<8)
    {
    oldfunc=getvect(wIrq+8);
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
    setvect(wIrq+8, irq_service);
    }
else
    {
    oldfunc=getvect(wIrq-8+0x70);
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & 0xfb);                    /* IRQ2 */
    irqmask=inportb(A2_8259+1);
    outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
    setvect(wIrq-8+0x70, irq_service);
    }
invert=0x1;
outportb(wBase+0x2a,invert);            /* non-invert    DI0              */
                                        /*         invert    DI1              */
now_int_state=0x2;                      /* now    DI0 is low              */
                                        /* now    DI1 is high             */
outportb(wBase+5,0x3);                  /* enable all interrupt       */
enable();
}
/* ------------------------------------------------------------ */
```

```
void interrupt irq_service()
{
int_num++;
new_int_state=inportb(wBase+7)&0x3;
int_c=new_int_state^now_int_state;
if ((int_c&0x1)!=0)                    /* now INT_CHAN_0 change to high */
    {
    if ((new_int_state&0x01)!=0)
        {
        CNT_H1++;
        }
    else                                /* now INT_CHAN_0 change to low    */
        {       CNT_L1++;       }
    invert=invert^1;                    /* generate a high pulse           */
    }
if ((int_c&0x2)!=0)                    /* now INT_CHAN_1 change to high */
    {
    if ((new_int_state&0x02)!=0)
        {       CNT_H2++;       }
    else                                /* now INT_CHAN_1 change to low    */
        {       CNT_L2++;       }
    invert=invert^2;                    /* generate a high pulse           */
    }
now_int_state=new_int_state;
outportb(wBase+0x2a,invert);
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

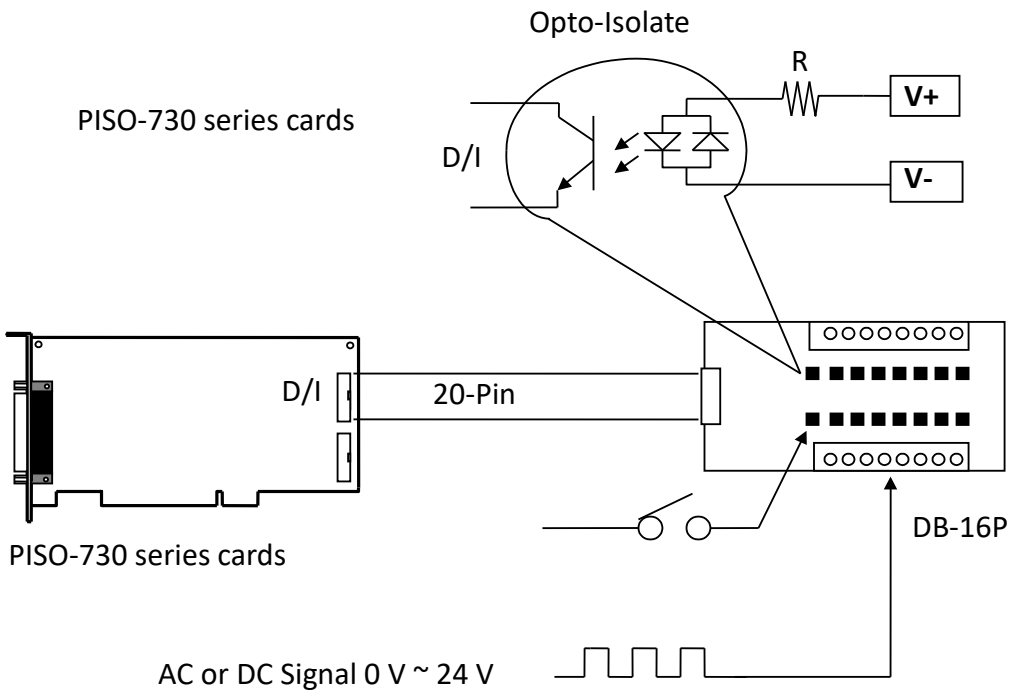
Appendix: Daughter Board

A1. DB-16P Isolated Input Board

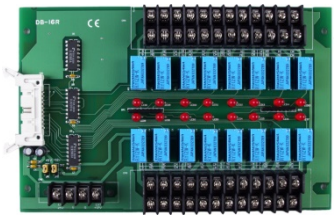
The DB-16P is a 16-channel isolated digital input daughter board. The optically isolated inputs of the DB-16P are consisted of are bi-directional optocoupler with resistor for current sensing. You can use the DB-16P to sense DC signal from TTL levels up to 24 V or use the DB-16P to sense a wide range of AC signals. You can use this board to isolate the computer from large common-mode voltage, ground loops and transient voltage spike that often occur in industrial environments.



DB-16P



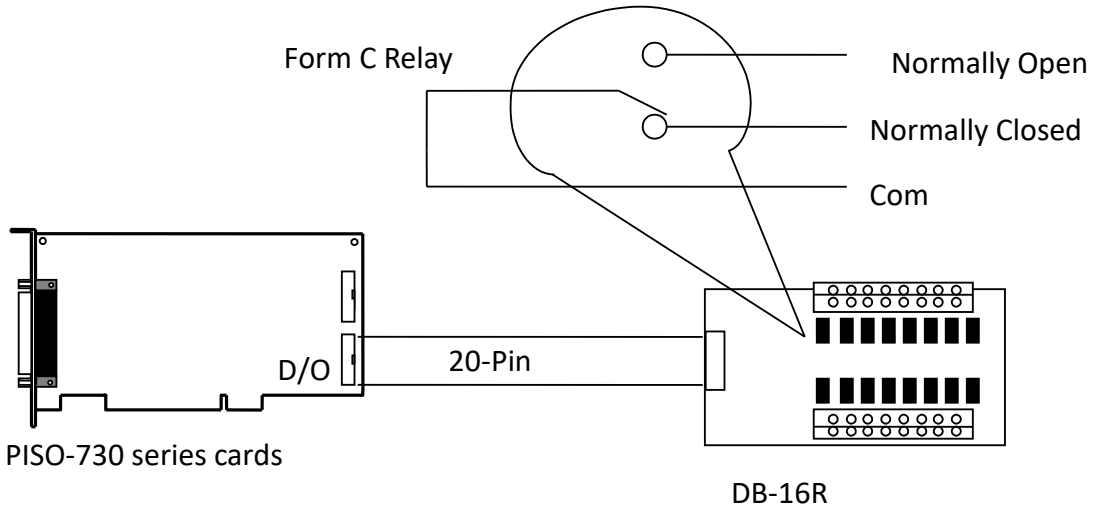
A2. DB-16R Relay Board



DB-16R

The DB-16R is a 16-channel relay output board consisting of 16 Form C relays that enable efficient switching of a load using programmable control. It is both a connector and functionally is compatible with 785 series boards, but with an industrial type terminal block. The relay is powered by applying a 5 V signal to the appropriate relay channel on the 20-pin flat connector. There are 16 LEDs for each relay, which illuminated when their associated relay is activated. This board includes a screw terminal that can be used to connect an external power supply in order to prevent overloading your PC's power supply.

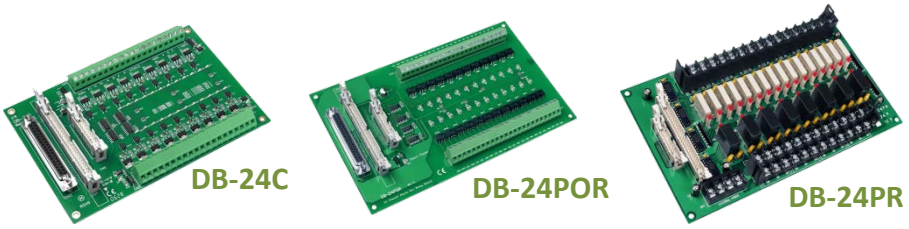
The application example for the DB-16R in the PISO-730 is illustrated in below figure.



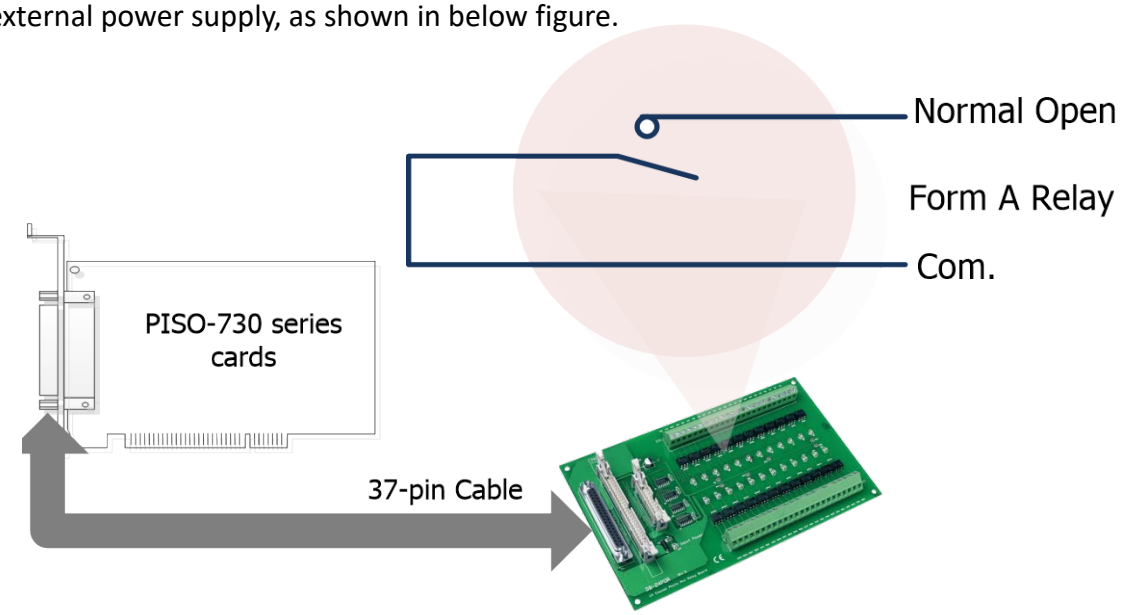
Note:

Channel: 16 Form C Relay
Relay: Switching up to 0.5 A at 110 VAC/ 1 A at 24 VDC

A3. DB-24PR, DB-24POR and DB-24C



The DB-24PR, 24-channel power relay output board, consists of 8 Form-C and 16 Form-A electromechanical relays for efficiently controlling the switch with the use of an appropriately loaded program. The contact of each relay can allow 5 A current load at 250 V_{AC}/30 V_{DC}. The relay is energized by applying a 5 voltage signal to the associate relay channel on the 20-pin flat-cable connector (just used 16 relays) or 50-pin flat-cable connector (OPTO-22 compatible, for DIO-24 series). 24 enunciator LEDs for indicating the status of for each relay and the corresponding LED light will go on when their associated relay has been activated. To avoid overloading your PC's power supply, this board needs a +12 V_{DC} or +24V_{DC} external power supply, as shown in below figure.



DB-24PR	24 * Power Relay, 5A/250 V
DB-24POR	24 * PhotoMOS Relay, 0.1 A/350 V_{AC}
DB-24C	24 Open Collector, 100 mA per channel, 30 V max.

Notes:

1. 50-Pin connector (OPTO-22 compatible) for DIO-24/48/144, PIO-DIO series.
2. Channel: 16 Form A Relay, 8 Form C Relay.
3. Relay: switching up to 5 A at 110 V_{AC}/5 A at 30 V_{DC}.

A4. Daughter Board Comparison Table

	20-pin flat-cable header	50-pin flat-cable header	DB-37 header
DB-37	No	No	Yes
DN-37	No	No	Yes
ADP-37/PCI	No	Yes	Yes
ADP-50/PCI	No	Yes	No
DB-24P	No	Yes	No
DB-24PD	No	Yes	Yes
DB-16P8R	No	Yes	Yes
DB-24R	No	Yes	No
DB-24RD	No	Yes	Yes
DB-24C	Yes	Yes	Yes
DB-24PR	Yes	Yes	No
Db-24PRD	No	Yes	Yes
DB-24POR	Yes	Yes	Yes
DB-24SSR	No	Yes	Yes

Note:

The PISO-730 series card has two 20-pin flat-cable headers and one 37 pin D-type Connector.