PISO-725

User Manual

Version 1.6
Oct. 2011

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.

Tables of Contents

1. Introduction

The PISO-725 is a 16-channel digital input/output interface board for PCI bus computers. It can be installed in a 5 V PCI slot and supports true "Plug & Play" to automatically obtain I/O resources. The PISO-725 has a single DB-37 connector, and provides 8 electromechanical relay outputs and 8 isolated/non-isolated digital inputs. The digital inputs can be set to either isolated or non-isolated via a hardware jumper. Each of the digital inputs will generate an interrupt signal if the state is changed, which is very useful when monitoring for contact closures/openings as it is not necessary to continuously poll the inputs. The isolated DI channels use a short optical transmission path to transfer an electronic signal between elements of a circuit and keep them electrically isolated. With 3750 Vrms isolation protection, these DI channels allow the input signals to be completely floated so as to prevent ground loops and isolate the host computer from damaging voltages. Relays are used where it is necessary to control a circuit using a low-power signal (with complete electrical isolation between the control and controlled circuits), or where several circuits must be controlled by one signal. All relays are de-energized (off) while powering-on, and support On/Off status read back. The PISO-725 can be used in various applications, including contact closure, external voltage sensing, load sensing and I/O control.

The PISO-725 supports various OS versions, such as Linux, DOS, Windows 98/NT/2000 and 32/64-bit Windows 7/Vista/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.

1.1 Features and Applications

1.1.1 Features

- PCI Bus
- State-changed interrupt for all digital inputs
- Jumper selectable isolated or non-isolated digital inputs
- 8 electromechanical relay outputs (Form C x 4, Form A x 4)
- 3750 Vrms Photo-isolation protection
- Supports relay output status readback
- LED to indicate output state
- One DB-37 connector for input and output
- Supports Plug & Play to obtain I/O resources
- No more manually setting of I/O address and IRQ

1.1.2 Applications

- Factory automation
- Laboratory automation
- Communication switching
- Product testing

1.2 Specifications

Model Name	PISO-725
Digital Input	
Isolation Voltage	3750 Vrms (Using external power)
Channels	8
Compatibility	Photo coupler isolated
Input Voltage	Logic 0: 0 ~ 1 V Logic 1: 9 ~ 24 V
Input Impedance	1.2 KΩ, 1 W
Response Speed	4 kHz (Typical)
Relay Output	
Channels	8
Relay Type	Form C
Contact Rating	AC:0.3 A / 120 V DC: 1 A / 30 V
Operating Time	5 ms (typical)
Release Time	10 ms (typical)
Life	Mechanical: 100,000 ops.(30 V / 1 A)
General	
Bus Type	5 V PCI, 32-bit, 33 MHz
Data Bus	8-bit
Card ID	No
I/O Connector	Female DB37 x 1
Dimensions (L x W x D)	150 mm x 110 mm x 22 mm
Power Consumption	300 mA @ +5 V
Operating Temperature	0 ~ 60 °C
Storage Temperature	-20 ~ 70 °C
Humidity	5 ~ 85% RH, non-condensing

1.3 Order Description

PISO-725: 8 channels isolated digital input and 8 channels relay output board. Includes one CA-4002 D-Sub connector

Options

- DN-37: I/O connector block with DIN-Rail mounting and 37-pin D-type connector.
- DB-37: 37-pin D-type connector pin to pin screw terminal for any 37 pin D-type connector of I/O board.

1.4 PCI Data Acquisition Family

We provide a family of PCI-BUS data acquisition cards. These cards can be divided into three groups as following:

1. **PCI-series: first generation, isolated or non-isolated cards**
   PCI-1002/1202/1800/1802/1602: multi-function family, non-isolated
   PCI-P16R16/P16C16/P16POR16/P8R8: D/I/O family, isolated
   PCI-TMC12: timer/counter card, non-isolated

2. **PIO-series: cost-effective generation, non-isolated cards**
   PIO-821: multi-function family
   PIO-D168/D144/D96/D64/D56/D48/D24: D/I/O family
   PIO-DA16/DA8/DA4: D/A family

3. **PISO-series: cost-effective generation, isolated cards**
   PISO-813: A/D card
   PISO-P32C32/P32A32/P64/C64/A64: D/I/O family
   PISO-P8R8/P8SSR8AC/P8SSR8DC: D/I/O family
   PISO-730: D/I/O card
   PISO-725: D/I/O card
   PISO-DA2: D/A card

1.5 Product Check List

The shipping package includes the following items:

- One PISO-725 series card
- One software utility PCI CD.
- One Quick Start Guide.

It is recommended that you read the Quick Start Guide first. All the necessary and essential information is given in the Quick Start Guide, including:

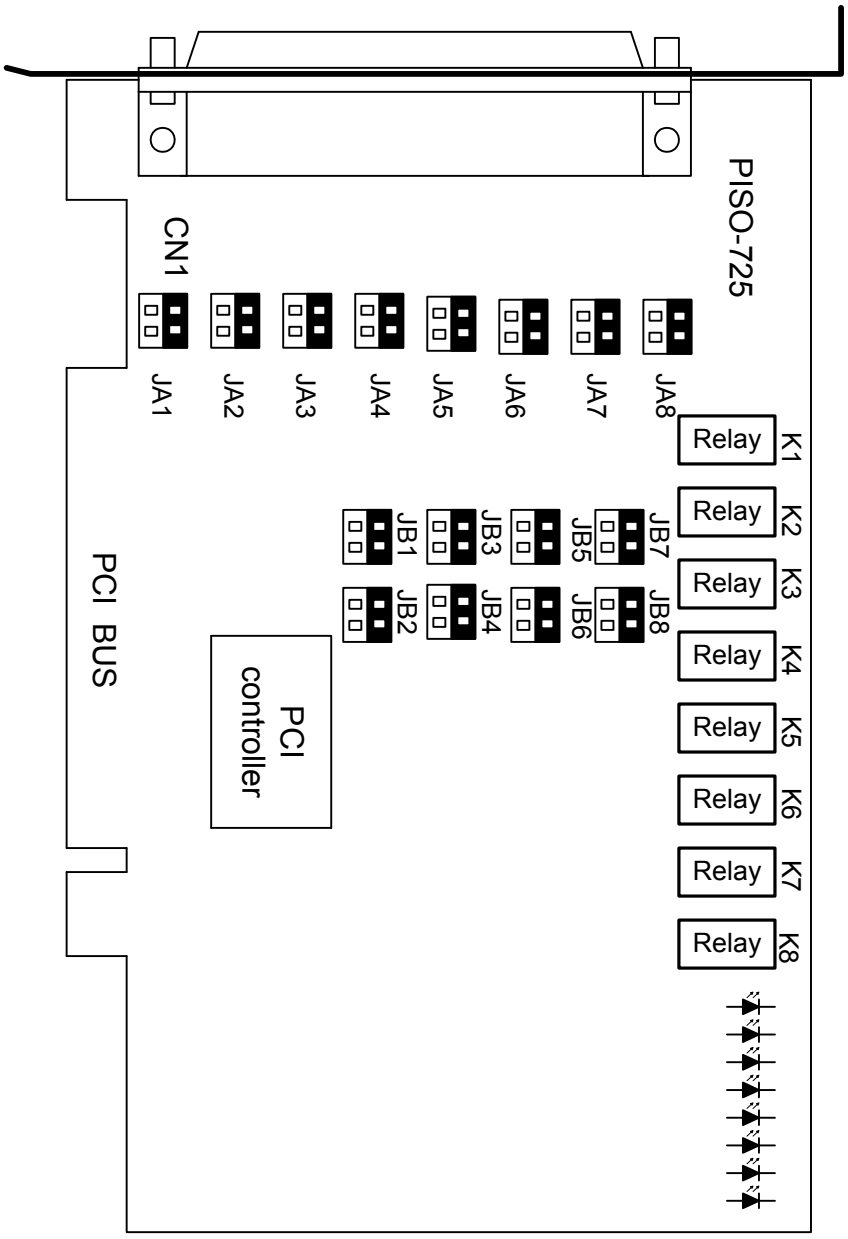
- Where to get the software driver, demo programs and other resources.
- How to install the software.
- How to test the card.

Attention!

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

2. Hardware configuration

2.1 Board Layout



- (JA?、JB?) is used to select the isolated/non-isolated digital input.
- The default setting of all (JA?、 JB?) = isolated input

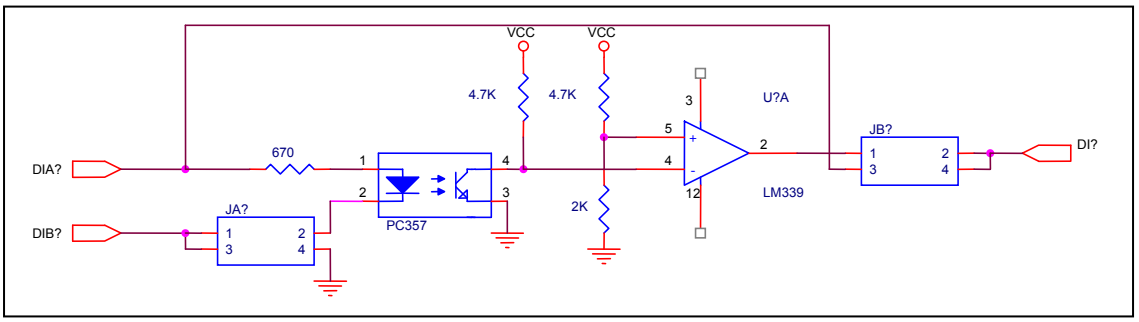
2.2 I/O Operation

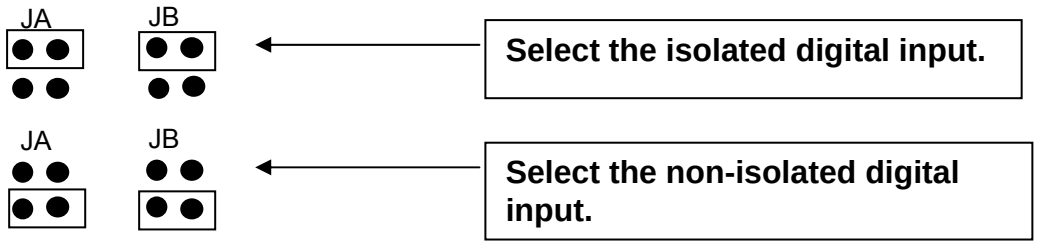
2.2.1 Isolated/Non-isolated Digital Input

The PISO-725 provides 8-channel of digital input. These inputs can be used as isolated or non-isolated based on different jumper setting. The jumper setting for isolated input is given as following:

Channel	Signal Name	Pin Assignment	Jumper
(0+, 0-)	(DIA0,DIB0)	(12,30)	JA1 & JB1
(1+, 1-)	(DIA1,DIB1)	(13,31)	JA2 & JB2
(2+, 2-)	(DIA2,DIB2)	(14,32)	JA3 & JB3
(3+, 3-)	(DIA3,DIB3)	(15,33)	JA4 & JB4
(4+, 4-)	(DIA4,DIB4)	(16,34)	JA5 & JB5
(5+, 5-)	(DIA5,DIB5)	(17,35)	JA6 & JB6
(6+, 6-)	(DIA6,DIB6)	(18,36)	JA7 & JB7
(7+, 7-)	(DIA7,DIB7)	(19,37)	JA8 & JB8

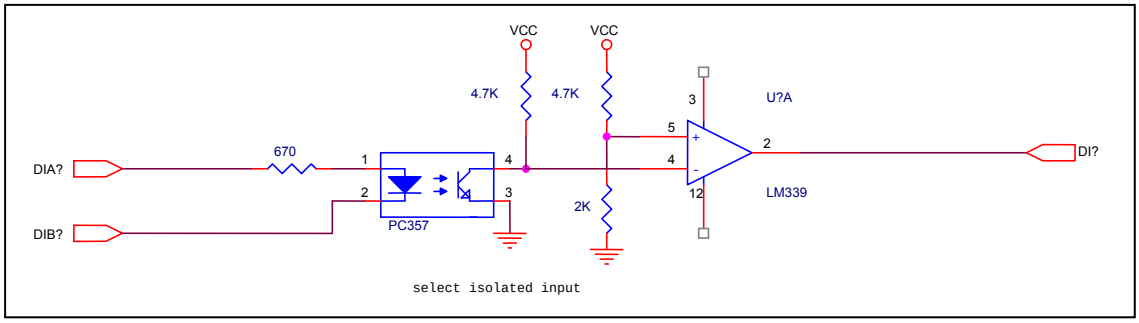
The block diagram of JA, JB & digital input circuit:





2.2.2 Isolated Digital Input

Refer to Sec. 2.2.1 for more information.



The features of isolated input are given as following:

- Photo-coupler for isolated input: PC-357
- Input_high voltage for isolated input: 3.5 ~ 30 V
- Input_low voltage for isolated input: 0 ~ 1 V
- Input impedance for isolated input: 3 k, 1/2 W
- Isolation voltage for isolated input: 3750 V
- Response time for isolated input: 20 μs

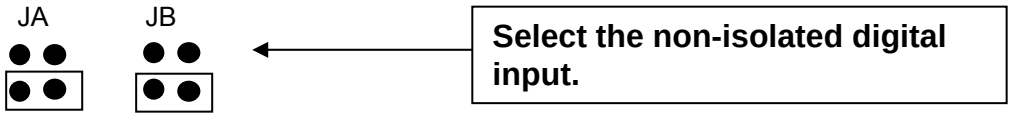
The (DIA? and DIB?) is used as a differential input (In+, In-) as following:

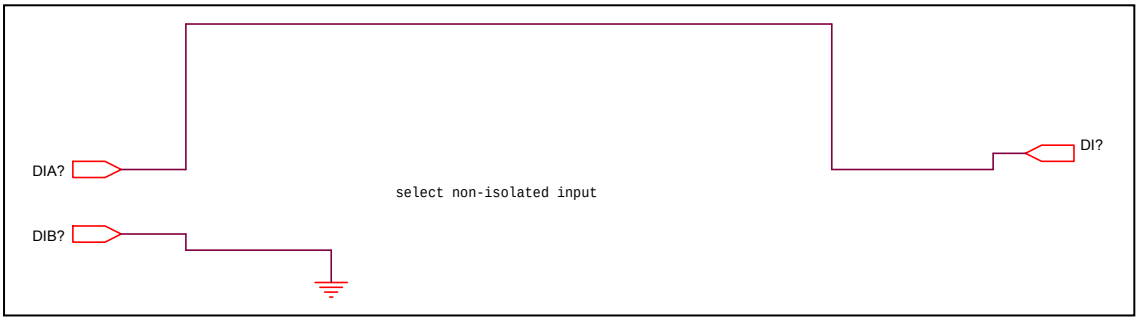
Channel	Signal Name	Pin Assignment	Jumper
(0+, 0-)	(DIA0,DIB0)	(12,30)	JA1 & JB1
(1+, 1-)	(DIA1,DIB1)	(13,31)	JA2 & JB2
(2+, 2-)	(DIA2,DIB2)	(14,32)	JA3 & JB3
(3+, 3-)	(DIA3,DIB3)	(15,33)	JA4 & JB4
(4+, 4-)	(DIA4,DIB4)	(16,34)	JA5 & JB5
(5+, 5-)	(DIA5,DIB5)	(17,35)	JA6 & JB6
(6+, 6-)	(DIA6,DIB6)	(18,36)	JA7 & JB7
(7+, 7-)	(DIA7,DIB7)	(19,37)	JA8 & JB8

If all input pins are floating, all DI? will be equal to 1.

2.2.3 Non-isolated Digital Input

Refer to Sec. 2.2.1 for more information.





The non-isolated input is TTL compatible.

All DIB? are connected to GND. All DIA? Are used as a single-ended input as following:

Channel	Signal Name	Pin Assignment	Jumper
0+	DIA0	12	JA1 & JB1
1+	DIA1	13	JA2 & JB2
2+	DIA2	14	JA3 & JB3
3+	DIA3	15	JA4 & JB4
4+	DIA4	16	JA5 & JB5
5+	DIA5	17	JA6 & JB6
6+	DIA6	18	JA7 & JB7
7+	DIA7	19	JA8 & JB8
GND	DIB0 to DIB 7	30 to 37	All DB?=GND

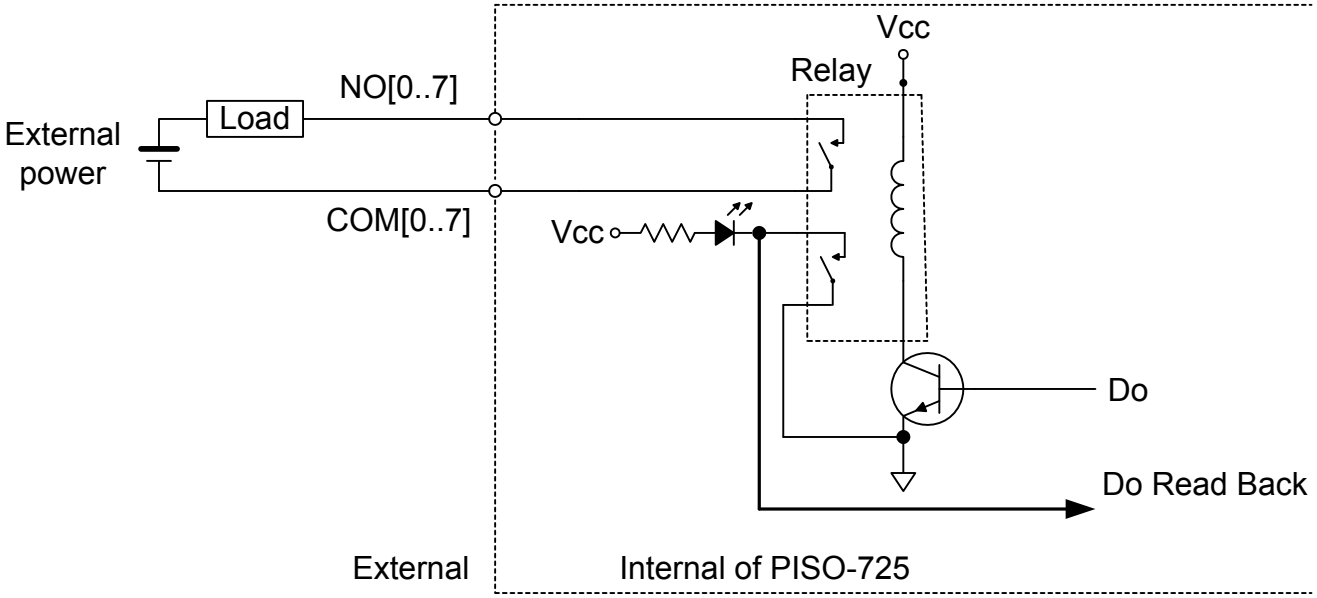
If all input pins are float, all DI? will be equal to 1.

2.2.4 Digital Output Architecture

When the PC is power-up, all states of output relay are “open”. The enable/disable of output operation is controlled by the RESET\ signal. Refer to Sec. 3.3.1 for more information about RESET\ signal.

- The RESET\ is in Low-state → all output operation are disable
- The RESET\ is in High-state → all output operation are enable

The block diagram of isolated output is given as following:



The relay of PISO-725 is 2-Form-C type as following:
One Form-C for user’s external device:

- COM: common input of relay
- NO: Normally Open Output (this pin will OPEN from COM after power-up)
- NC: Normally Closed Output (this pin will CLOSE to COM after power-up)

The other form-C for read back (refer to Sec. 4.3.7 for more information)

2.3 Interrupt Operation

There are 8 interrupt sources in PISO-725. These 8 signals are named as INT_CHAN_0, INT_CHAN_1, ……., and INT_CHAN_7 as following:

INT_CHAN_0: (DIA0, DIB0)
INT_CHAN_1: (DIA1, DIB1)
INT_CHAN_2: (DIA2, DIB2)
INT_CHAN_3: (DIA3, DIB3)
INT_CHAN_4: (DIA4, DIB4)
INT_CHAN_5: (DIA5, DIB5)
INT_CHAN_6: (DIA6, DIB6)
INT_CHAN_7: (DIA7, DIB7)

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

If there are more than one interrupt source, the interrupt service routine has to identify and service all active signals as following: (refer to DEMO5.C and DEMO6.C)

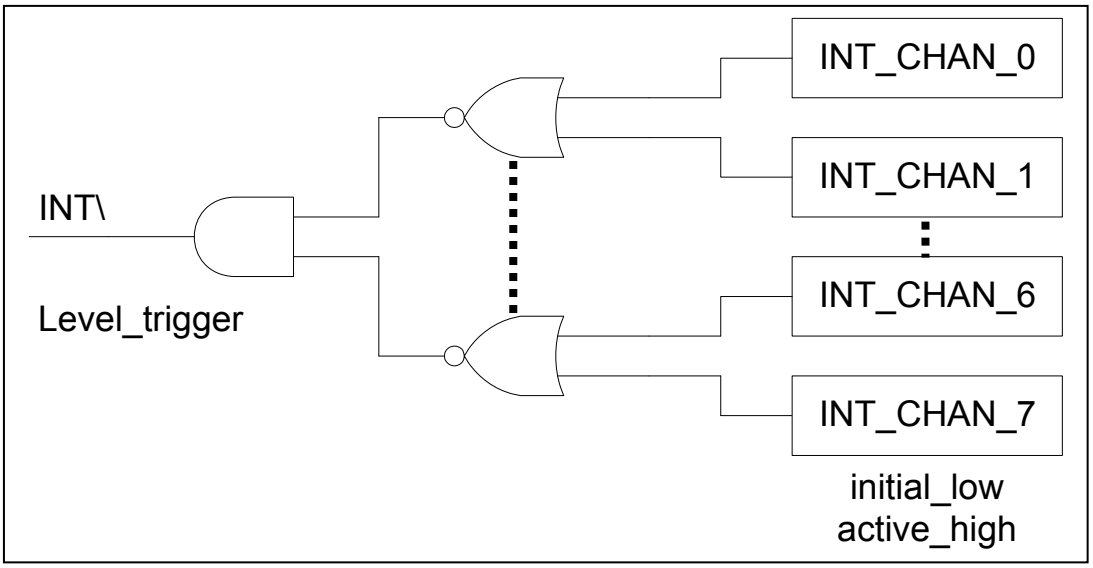
1. **Read the new status of all interrupt signal sources (refer to Sec 4.3.5)**
2. **Compare the new status with the old status to identify the active signals**
3. **If INT_CHAN_0 is active, service it**
4. **If INT_CHAN_1 is active, service it**
5. **If INT_CHAN_2 is active, service it**
6. **If INT_CHAN_3 is active, service it**
7. **If INT_CHAN_4 is active, service it**
8. **If INT_CHAN_5 is active, service it**
9. **If INT_CHAN_6 is active, service it**
10. **If INT_CHAN_7 is active, service it**
11. **Update 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 can not 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 micro-second or as long as second. In general, 20ms is enough for all O. S.

Refer to the following sections for more information:

Sec. 5.5.3 DEMO3.C
Sec. 5.5.4 DEMO4.C
Sec. 5.5.5 DEMO5.C
Sec. 5.5.6 DEMO6.C

2.3.1 Interrupt Block Diagram of PISO-725



The interrupt output signal of PISO-725, INT\ is **level-trigger & Active_Low**. If the INT\ generate a low-pulse, the PISO-725 will interrupt the PC once a time. If the INT\ is fixed in low level, the PISO-725 will interrupt the PC continuously. So the INT_CHAN_0/1/2/3/4/5/6/7 must be controlled in a **pulse_type** signals. **They must be fixed in low level state normally and generated a high_pulse to interrupt the PC.**

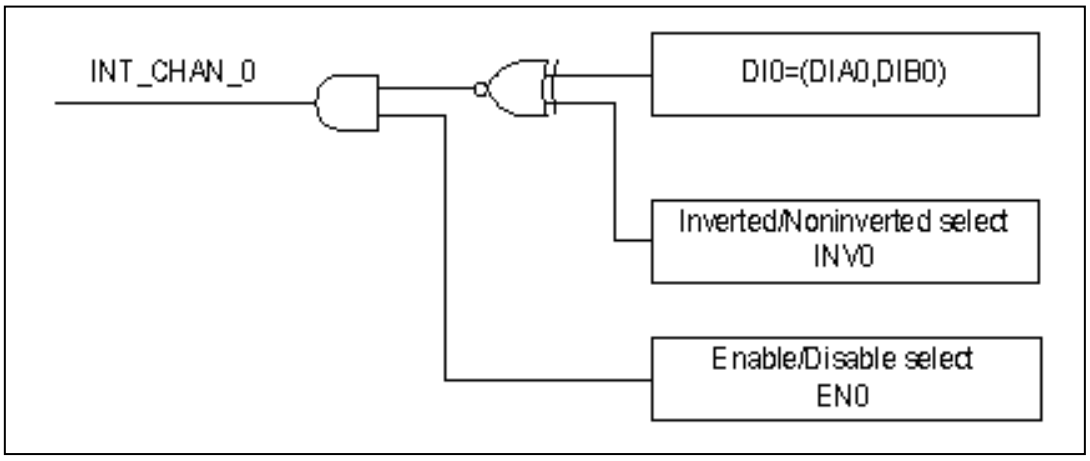
The priority of INT_CHAN_0/1/2/3/4/5/6/7 is the same. If all these 8 signals are active at the same time, then INT\ will be active only once a time. So the interrupt service routine has to read the status of all interrupt channels for multi-channel interrupt. Refer to DEMO5.C and DEMO6.C for more information.

DEMO5.C → for 2-channel interrupt source
DEMO6.C → for 8-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 program DEMO3.C and DEMO4.C are designed for single-channel interrupt demo.

DEMO3.C and DEMO4.C → for INT_CHAN_0 only

2.3.2 INT_CHAN_0



The INT_CHAN_0 must be fixed in low level state normally and generated a high_pulse to interrupt the PC.

The EN0 can be used to enable/disable the INT_CHAN_0 as following: (refer to Sec. 4.3.4)

EN0=0→INT_CHAN_0=disable
EN0=1→INT_CHAN_0=enable

The INV0 can be used to invert/non-invert the DI0 as following: (Refer to Sec. 4.3.6)

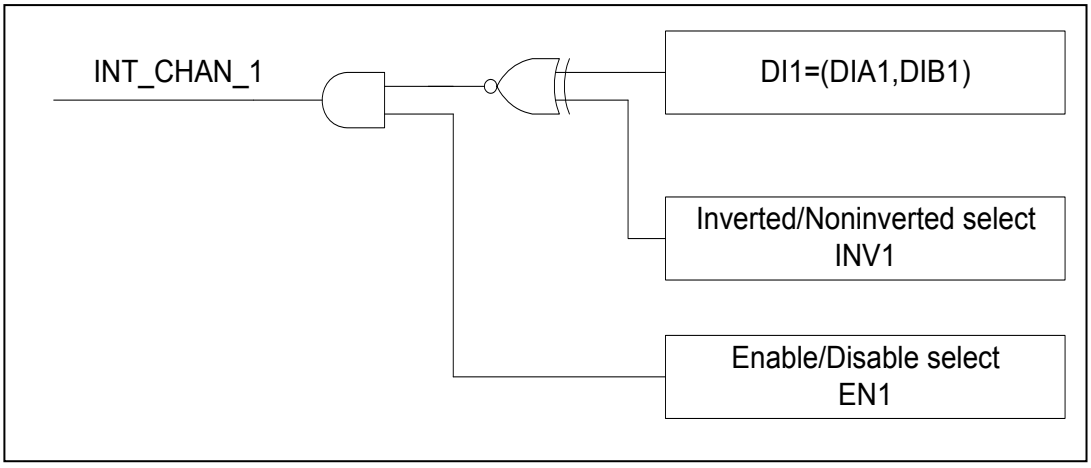
INV0=0→INT_CHAN_0=invert state of DI0
INV0=1→INT_CHAN_0=non-invert state of DI0

Refer to demo program for more information as following:

DEMO3.C → for INT_CHAN_0 (initial high)
DEMO4.C → for INT_CHAN_0 (initial low)
DEMO5.C → for 2-channel interrupt source
DEMO6.C → for 8-channel interrupt source

NOTE: Refer to Sec. 2.3.5 and Sec. 2.3.6 for active high-pulse generation.

2.3.3 INT_CHAN_1



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

The EN1 can be used to enable/disable the INT_CHAN_1 as following: (refer to Sec. 4.3.4)

EN1=0→INT_CHAN_1=disable

EN1=1→INT_CHAN_1=enable

The INV1 can be used to invert/non-invert the DI1 as following: (Refer to Sec. 4.3.6)

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 following:

DEMO3.C → for INT_CHAN_0 (initial high)

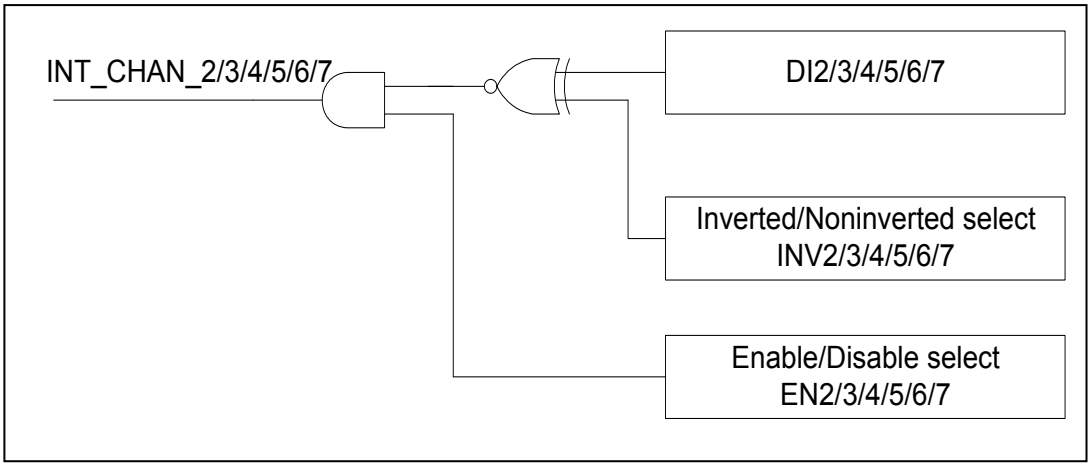
DEMO4.C → for INT_CHAN_0 (initial low)

DEMO5.C → for 2-channel interrupt source

DEMO6.C → for 8-channel interrupt source

NOTE: Refer to Sec. 2.3.5 and Sec. 2.3.6 for active high-pulse generation.

2.3.4 INT_CHAN_2 to INT_CHAN_7



The INT_CHAN_2/3/4/5/6/7 must be fixed in low level state normally and generated a high_pulse to interrupt the PC.

The EN2 to EN7 can be used to enable/disable the INT_CHAN_2 to INT_CHAN_7 as following: (refer to Sec. 4.3.4)
EN2 to EN7=0→INT_CHAN_2 to INT_CHAN_7=disable
EN2 to EN7=1→INT_CHAN_2 to INT_CHAN_7=enable

The INV2 to INV7 can be used to invert/non-invert the DI2 to DI7 as following: (Refer to Sec. 4.3.6)
INV2 to INV7=0→INT_CHAN_2 to INT_CHAN_7=invert state of DI2 to DI7
INV2 to INV7=1→INT_CHAN_2 to INT_CHAN_7=non-invert state of DI2 to DI7

Refer to demo program for more information as following:
DEMO3.C → for INT_CHAN_0 (initial high)
DEMO4.C → for INT_CHAN_0 (initial low)
DEMO5.C → for 2-channel interrupt source
DEMO6.C → for 8-channel interrupt source

NOTE:
1. **DI2=(DIA2, DIB2)**
2. **DI3=(DIA3, DIB3)**
3. **DI4=(DIA4, DIB4)**
4. **DI5=(DIA5, DIB5)**
5. **DI6=(DIA6, DIB6)**
6. **DI7=(DIA7, DIB7)**
7. **Refer to Sec. 2.3.5 and Sec. 2.3.6 for active high-pulse generation.**

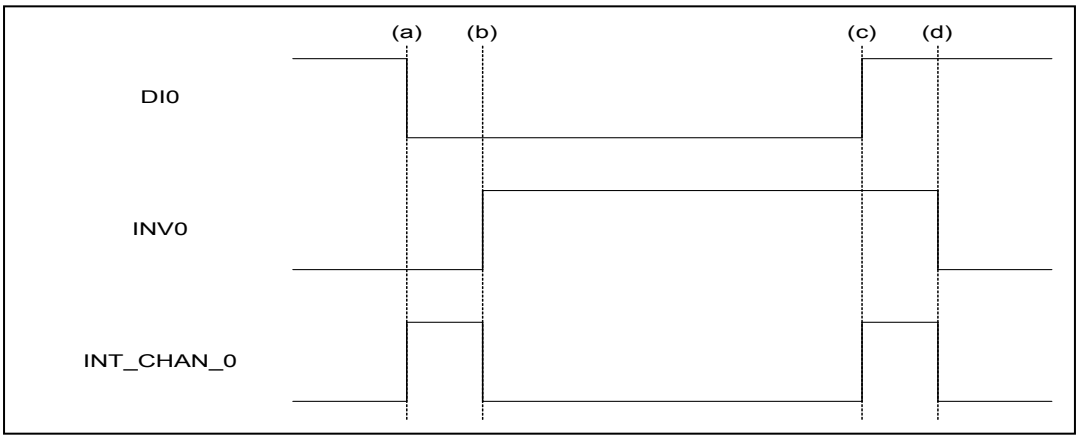
2.3.5 Initial_high, active_low Interrupt source

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 following: (Refer to DEMO3.C and the DI1/2/3/4/5/6/7 are similar.)

Initial setting:

```
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          */(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     */
     {                      /* → need to generate a high_pulse   */
     outportb(wBase+0x2a,1);/* INV0 select the non-inverted input */(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                       */
                            /* don't have to generate high_pulse  */
   }
else                        /* now DI0 is changed to HIGH         */(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    */
     {                      /* need to generate a high_pulse      */
     outportb(wBase+0x2a,0);/* INV0 select the inverted input     */(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                        */
                            /* don't have to generate high_pulse  */
   }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



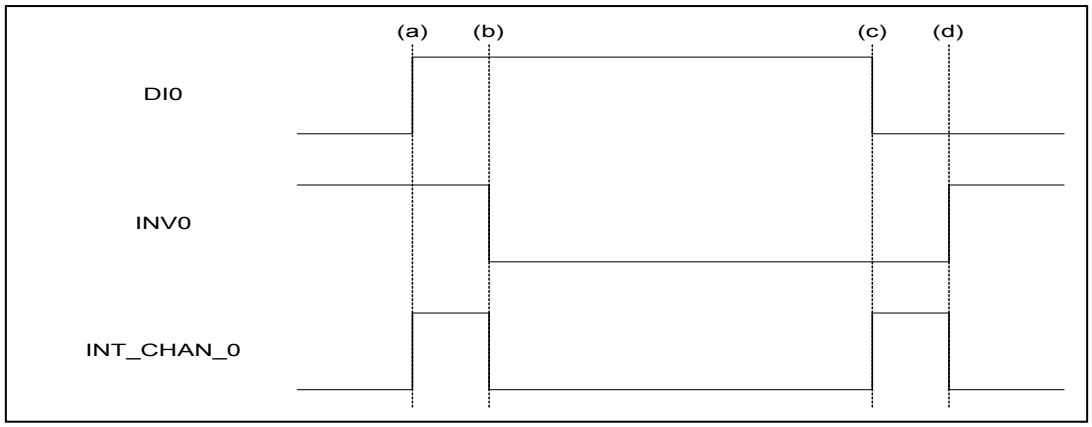
2.3.6 Initial_low, active_high Interrupt source

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 following: (Refer to DEMO4.C and the DI1/2/3/4/5/6/7 are similar.)

Initial setting:

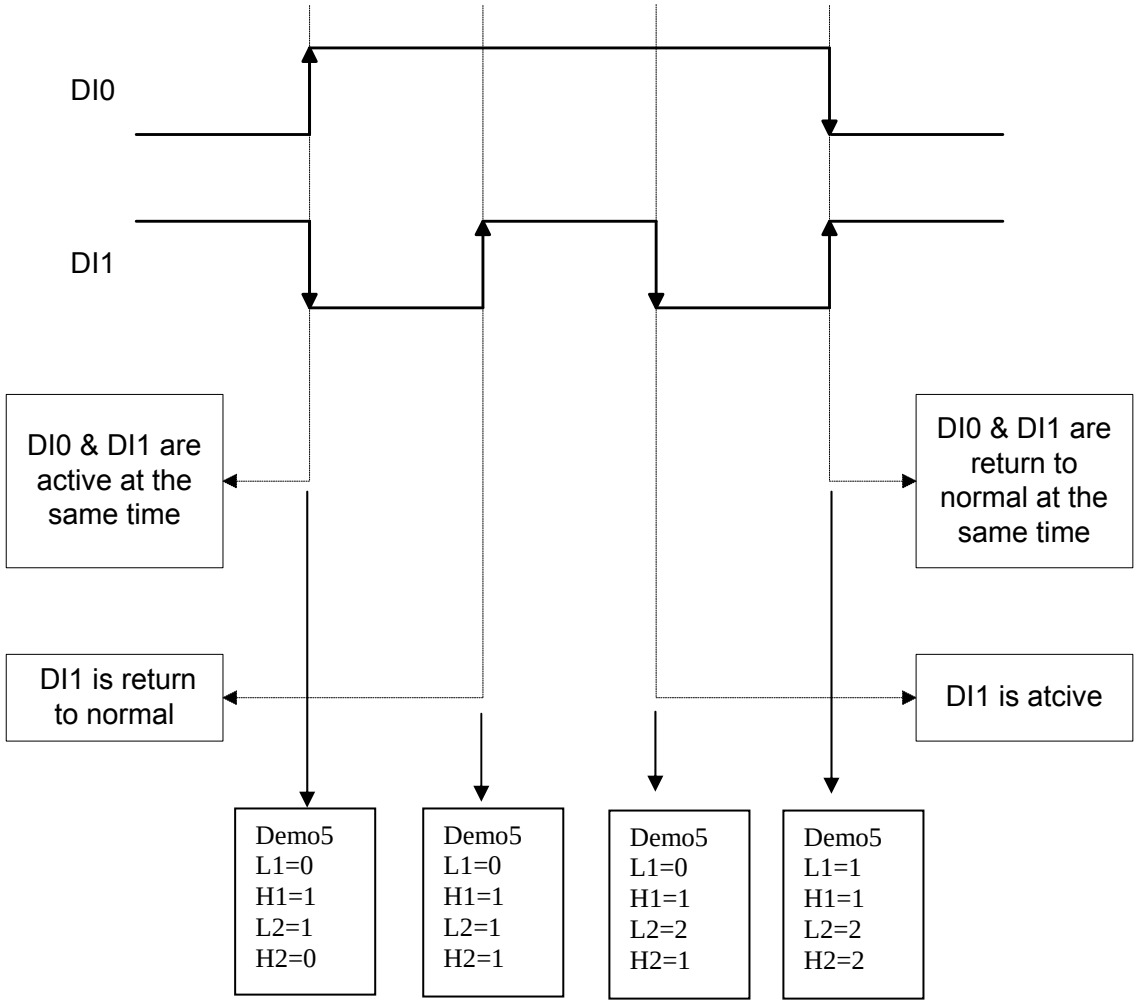
```
now_int_state=0;                    /* initial state for DI0          */
outportb(wBase+0x2a,1);             /* select the non-inverted DI0 */
```

```
void interrupt irq_service()
{
if (now_int_state==1)          /* now DI0 is changed to LOW          */(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      */
     {                         /* → need to generate a high_pulse   */
     outportb(wBase+0x2a,1);/* INV0 select the non-inverted input */(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                       */
                               /* don't have to generate high_pulse  */
   }
else                           /* now DI0 is changed to HIGH         */(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     */
     {                         /* need to generate a high_pulse      */
     outportb(wBase+0x2a,0);/* INV0 select the inverted input     */(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                        */
                               /* don't have to generate high_pulse  */
   }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



2.3.7 Multiple Interrupt Source 1

Assume: DI0=(DIA0,DIB0) is initial Low, active High
DI1=(DIA1,DIB1) is initial High, active Low
as following:



Refer to DEMO5.C for source program. **All these three falling-edge and rising-edge can be detected by DEMO5.C.**

NOTE:
When the interrupt is active, the user program has to identify the active signals. These signals may be active at the same time. So the interrupt service routine has to service all active signals at the same time.

Initial setting:

```
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 be change         */
if ((int_c&0x1)!=0)                   /* INT_CHAN_0 is active     */
   {
   if ((new_int_state&0x01)!=0)       /* now DI0 change to high   */
      {
      CNT_H1++;
      }
   else                               /* now DI0 change 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 change 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);
}
```

2.3.8 Multiple Interrupt Source 2

Assume: DI0/2/4/5 are initial Low, active High
DI1/3/6/7 are initial High, active Low

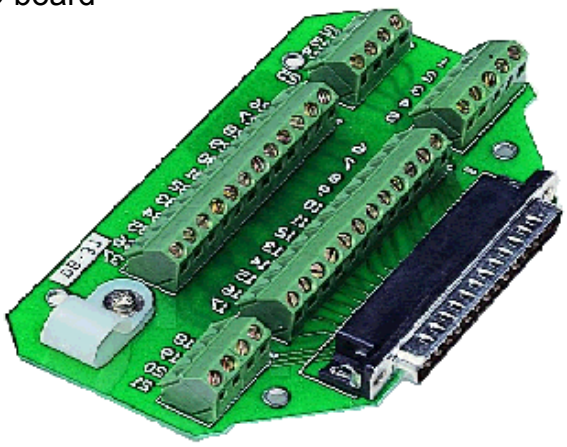
Refer to DEMO6.C for state-changed interrupt for all 8 digital inputs.

2.4 Daughter Boards

2.4.1 DB-37

Direct connection board

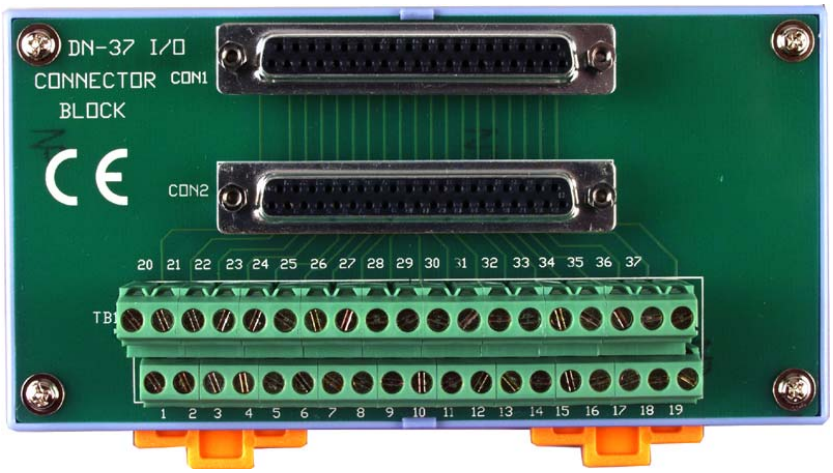
- 37-pin D-type connector pin to pin screw terminal for any 37-pin D-type connector of I/O board



2.4.2 DN-37

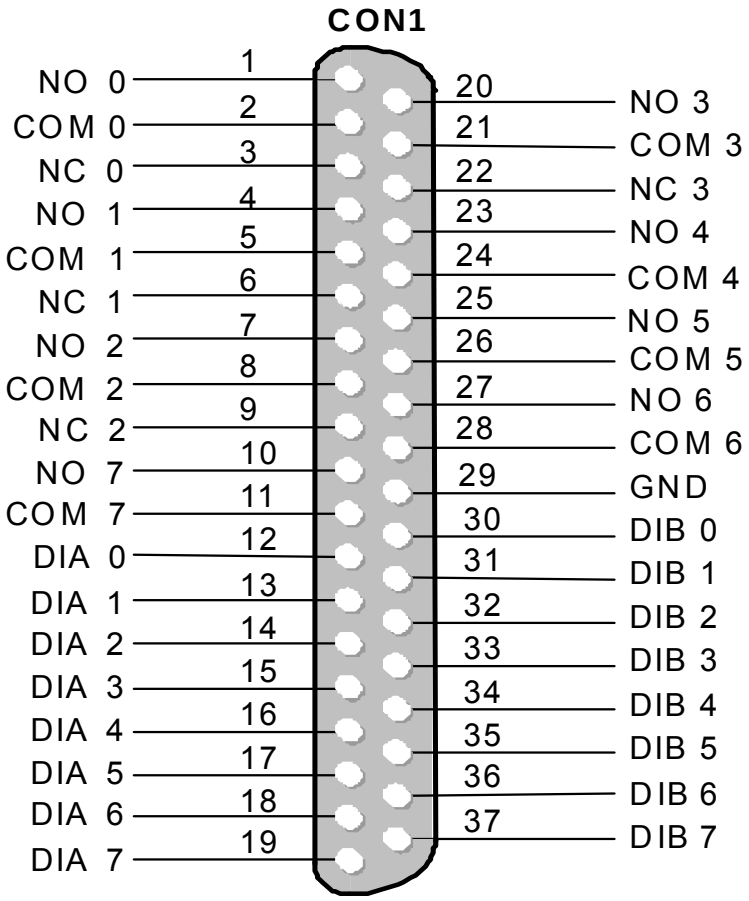
I/O connector block with DN-Rail mounting

- Two 37-pin D-type connector (one for extension)
- Pin to pin screw terminal for I/O connector



2.5 Pin Assignment

■ CON1: 37 pin of D-type female connector



NOTE:

1: COM?=Common Point, NO?=Normally Open, NC?=Normally Closed

2: DI0=(DIA0, DIB0)

DI1=(DIA1, DIB1)

DI2=(DIA2, DIB2)

DI3=(DIA3, DIB3)

DI4=(DIA4, DIB4)

DI5=(DIA5, DIB5)

DI6=(DIA6, DIB6)

DI7=(DIA7, DIB7)

3: The DI0 ~ DI7 can be isolated/non-isolated input based on (JA?,JB?) setting. Refer to Sec. 2.1 ~ Sec. 2.3 for more information.

3. Software Installation Guide

The PISO-725 can be used in DOS and Windows 98/NT/2K and 32-bit/64-bit Windows XP/2003/Vista/7. The recommended installation procedure for windows is given in Sec. 3.1 ~ 3.2. Or refer to Quick Start Guide (CD:\NAPDOS\PCI\PISO-725\ Manual\).
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/piso-725/manual/

3.1 Software Installing Procedure

- UniDAQ SDK driver (32-bit/64-bit Windows XP/2003/Vista/7):

Step 1: Insert the companion CD into the CD-ROM drive and after a few seconds the installation program should start automatically. If it doesn't start automatically for some reason, double-click the **AUTO32.EXE** file in the **NAPDOS** folder on this CD.

Step 2: Click the item: "**PCI Bus DAQ Card**".

Step 3: Click the item: "**UniDAQ**".

Step 4: Click the item: "**DLL for Windows 2000 and XP/2003/Vista 32-bit**".

Step 5: Double-Click "**UniDAQ_Win_Setup_x.x.x.x_xxxx.exe**" file in the **Driver** folder.

- Windows driver (Windows 98/NT/2K and 32-bit Windows XP/2003/Vista/7):

Step 1: Insert the companion CD into the CD-ROM drive and after a few seconds the installation program should start automatically. If it doesn't start automatically for some reason, double-click the **AUTO32.EXE** file in the **NAPDOS** folder on this CD.

Step 2: Click the item: "**PCI Bus DAQ Card**".

Step 3: Click the item: "**PISO-725**".

Step 4: Click the item "**DLL and OCX for Windows 98/NT/2K/XP/2003**".

Step 5: Choose the Win2K_XP, Win98 or WinNT folders for setup according to your PC platform and then double-Click "**.exe**" to install driver.

The setup program will then start the driver installation and copy the relevant files to the specified directory and register the driver on your computer. The directory where the drive is stoned is different for different windows versions, as shown below.

■ **Windows 64-bit Windows XP/2003/Vista/7:**

The UniDAQ.DLL file will be copied into the C:\WINNT\SYSTEM32 folder
The NAPWNT.SYS and UniDAQ.SYS files will be copied into the C:\WINNT\SYSTEM32\DRIVERS folder

For more detailed UniDAQ.DLL function information, please refer to UniDAQ SDK user manual (CD:\NAPDOS\PCI\UniDAQ\Manual\)**.**
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/maunal/

■ **Windows NT/2K and 32-bit Windows XP/2003/Vista/7:**

The PISO725.DLL file will be copied into the C:\WINNT\SYSTEM32 folder
The NAPWNT.SYS and PISO725.SYS files will be copied into the C:\WINNT\SYSTEM32\DRIVERS folder

■ **Windows 95/98/ME:**

The PISO725.DLL and PISO725.Vxd files will be copied into the C:\Windows\SYSTEM folder

For more detailed PISO725.DLL function information, please refer to PISO-725_dll_software_Manual.pdf(CD:\NAPDOS\PCI\PISO-725\Manual\)**".**
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/piso-725/manual/

3.2 PnP Driver Installation

Power off the computer and install the PISO-P16R16U and PEX-P8R8i/P16R16i cards. Turn on the computer and Windows 98/ME/2K and 32-bit/64-bit Windows XP/2003/Vista/7 should automatically defect the new PCI device(s) and then ask for the location of the driver files for the hardware. If a problem is encountered during installation, refer to the PnPinstall.pdf file for more information.

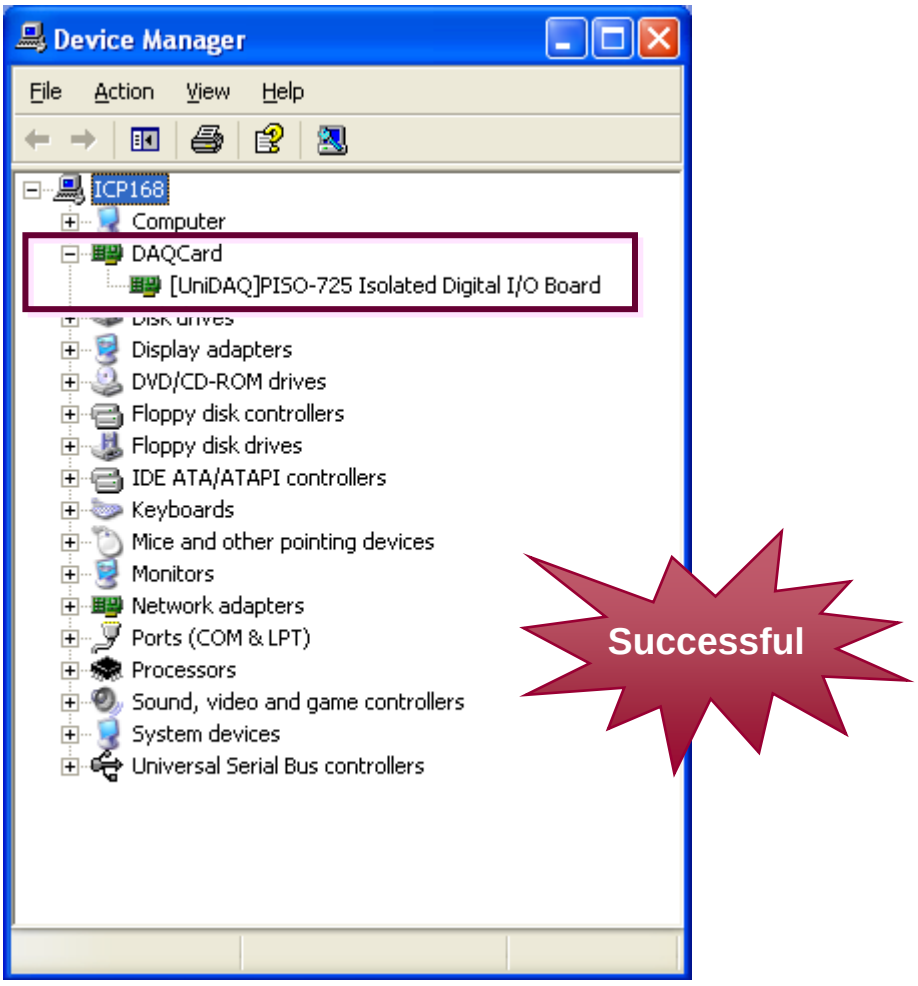
3.3 Confirm the Successful Installation

Make sure the PISO-725 card installed is correct on the computer as follows:

Step 1: Select **"Start"** → **"Control Panel"** and then double click the **"System"** icon on Windows.

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

Step 3: Check the PISO-725 card which listed correctly or not, as illustrated below.



4. I/O Control Register

4.1 How to Find the I/O Address

The plug & play BIOS will assign a proper I/O address to every PIO/PISO series card in the power-up stage. The fixed IDs of PIO/PISO series cards are given as following:

Model Name	PISO-725 <Rev 1.0 ~ 2.0>	PISO-725 <Rev 2.3>
Vendor ID	0xE159	0xE159
Device ID	0x02	0x01
Sub Vendor ID	0x80	0xC380
Sub Device ID	0x0C	0x00
Sub-Aux ID	0x00	0x00

We provide all necessary functions as following:

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 PIO.H. Refer to Chapter 5 for more information. The important driver information is given as following:

1. Resource-allocated information:

- wBase : BASE address mapping in this PC
- wIrq: IRQ channel number allocated in this PC

2. PIO/PISO identification information:

- wSubVendor: subVendor ID of this board
- wSubDevice: subDevice ID of this board
- **wSubAux: set this variable to 0xff for PISO-725**

3. PC's physical slot information:

- wSlotBus: hardware slot ID1 in this PC's slot position
- wSlotDevice: hardware slot ID2 in this PC's slot position

The utility program, **PIO_PISO.EXE**, will detect and show all PIO/PISO cards installed in this PC. Refer to Sec. 5.3 or Sec. 5.4 for more information.

4.1.1 PIO_DriverInit

PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux)

- wBoards=0 to N → number of boards found in this PC
- wSubVendor → subVendor ID of board to find
- wSubDevice → subDevice ID of board to find
- **wSubAux → set to 0xff for PISO-725**

This function can detect all PIO/PISO series card 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 & save all their resource in the library.

Sample program 1: find all PISO-725 in this PC

```
wSubVendor=0x80; wSubDevice=0x0C; wSubAux=0xff;/*for PISO-725 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("Threr are %d PISO-725 Cards in this PC\n",wBoards);
/* step2: save resource of all PISO-725 cards installed in this PC */
for (i=0; i<wBoards; i++)
   {
    PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wID1,&wID2,&wID3,
                                        &wID4,&wID5);
    printf("\nCard_%d: wBase=%x, wIrq=%x", i,wBase,wIrq);
    wConfigSpace[i][0]=wBaseAddress;      /*save all resource of this card */
    wConfigSpace[i][1]=wIrq;              /* save all resource of this card  */
   }
```

Sample program 2: find all PIO/PISO in this PC (refer to Sec. 5.3 for more information)

```
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);  /*find all 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);
   }
```

The sub-IDs of PIO/PISO series card are given as following:

PIO/PISO series card	Description	Sub_vendor	Sub_device	Sub_AUX
PIO-D144	144 * D/I/O	80	01	00
PIO-D96	96 * D/I/O	80	01	10
PIO-D64	64 * D/I/O	80	01	20
PIO-D56	24* D/I/O + 16*D/I + 16*D/O	80	01	40
PIO-D48	48*D/I/O	80	01	30
PIO-D24	24*D/I/O	80	01	40
PIO-823	Multi-function	80	03	00
PIO-821	Multi-function	80	03	10
PIO-DA16	16*D/A	80	04	00
PIO-DA8	8*D/A	80	04	00
PIO-DA4	4*D/A	80	04	00
PISO-C64	64 * isolated D/O	80	08	00
PISO-P64	64 * isolated D/I	80	08	10
PISO-P32C32	32 D/I + 32 D/O	80	08	20
PISO-P8R8	8* isolated D/I + 8 * 220 V relay	80	08	30
PISO-P8SSR8AC	8* isolated D/I + 8 * SSR /AC	80	08	30
PISO-P8SSR8DC	8* isolated D/I + 8 * SSR /DC	80	08	30
PISO-730	16*DI + 16*D/O + 16* isolated D/I + 16* isolated D/O	80	08	40
PISO-813	32 * isolated A/D	80	0A	00
PISO-DA2	2 * isolated D/A	80	0B	00
PISO-725	**8 * D/I + 8 * D/O**	**80(C380)**	**0C(0000)**	**0xff**
PISO-PS300	**3 axis stepping+ 3 axis encoder**	**81**	**04**	**0xff**

Note:

The sub-IDs will be added more and more without notice. The user can refer to PIO.H for the newest information.

4.1.2 PIO_GetConfigAddressSpace

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

- wBoardNo=0 to N → totally N+1 boards found by PIO_DriveInit(….)
- wBase → base address of the board control word
- wIrq → allocated IRQ channel number of this board
- wSubVendor → subVendor ID of this board
- wSubDevice → subDevice ID of this board
- **wSubAux → don't care this variable for PISO-725**
- wSlotBus → hardware slot ID1 of this board
- wSlotDevice → hardware slot ID2 of this board

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

The sample program source is given as following:

```
/* step1: detect all PISO-725 cards first                    */
wSubVendor=0x80; wSubDevice=0x0C; wSubAux=0xff;/*for PISO-725 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("Threr are %d PISO-725 Cards in this PC\n",wBoards);

/* step2: save resource of all PISO-725 cards installed in this PC */
for (i=0; i<wBoards; i++)
  {
  PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&t1,&t2,&t3,&t4,&t5);
  printf("\nCard_%d: wBase=%x, wIrq=%x", i,wBase,wIrq);
 wConfigSpace[i][0]=wBaseAddress;     /* save all resource of this card     */
 wConfigSpace[i][1]=wIrq;             /* save all resource of this card     */
 }
/* step3: control the PISO-725 directly */
wBase=wConfigSpace[0][0];            /* get base address the card_0  */
outport(wBase,1);                 /* enable all D/I/O operation of card_0   */

wBase=wConfigSpace[1][0];                /* get base address the card_1 */
outport(wBase,1);                 /* enable all D/I/O operation of card_1 */
```

4.1.3 Show_PIO_PISO

Show_PIO_PISO(wSubVendor,wSubDevice,wSubAux)

- wSubVendor → subVendor ID of board to find
- wSubDevice → subDevice ID of board to find
- **wSubAux → set this variable to 0xff for PISO-725**

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

The demo program is given as following:

```
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);  /*find all 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);
   }
```

4.2 The Assignment I/O Address

The plug & play BIOS will assign the appropriate I/O address to the PIO/PISO series card. If there is only one PIO/PISO board, the board will be identified as card_0. If there are two PIO/PISO boards in the system, it is very difficult to identify which board is card_0. The software driver can support a maximum of 16 boards. Therefore, the user can install 16 PIO/PSIO series cards in one PC system. Details of how to locate and identify card_0 and card_1 are provided below:

The simplest way to identify which card is card_0 is to use wSlotBus and wSlotDevice functions as follows:

Step 1: Remove all PISO-725 from this PC

Step 2: Install one PISO-725 into the PC's PCI_slot1,
run PIO_PISO.EXE and record the wSlotBus1 and wSlotDevice1

Step 3: Remove all PISO-725 from this PC

Step 4: Install one PISO-725 into the PC's PCI_slot2,
run PIO_PISO.EXE and record the wSlotBus2 and wSlotDevice2

Step 5: Repeat Step3 and Step 4 for all PCI_slot, record all wSlotBus? and wSlotDevice?

The records may be as following:

PC's PCI slot	WslotBus	wSlotDevice
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 is used to record all the wSlotBus and wSlotDevice information for the PC. These values will be mapped to this PC's physical slots and this mapping will not be changed for any PIO/PISO cards. Therefore, this information can be used to identify the specific PIO/PISO card using the following steps:

Step1: Record all wSlotBus? & wSlotDevice?

Step2: Use PIO_GetConfigAddressSpace(…) to get the specified card's wSlotBus & wSlotDevice

Step3: The user can identify the specified PIO/PISO card if he compares the wSlotBus & wSlotDevice in step2 to step1.

4.3 The I/O Address Map

The I/O address of PIO / PISO series card is automatically assigned by the main board ROM BIOS. The I/O address can also be re-assigned by user. **It is strongly recommended not to change the I/O address by user. The plug&play BIOS will assign proper I/O address to each PIO/PISO series card very well.**

The I/O address table of PISO-725 is given as following:

Address	Read	Write
wBase+0	RESET\ control register	Same
wBase+2	Aux control register	Same
wBase+3	Aux data register	Same
wBase+5	INT mask control register	Same
wBase+7	Aux pin status register	Same
wBase+0x2a	INT polarity control register	Same
wBase+0xc0	Read Back of DO0 ~ DO7 (inverse of DO0 ~ DO7)	DO0~DO7
wBase+0xc4	DI0 ~ DI7	N/A

Note. Refer to Sec. 4.1 for more information about wBase.

4.3.1 RESET\ Control Register

(Read/Write): wBase+0

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

Note. Refer to Sec. 4.1 for more information about wBase.

When the PC is first power-up, the RESET\ signal is in Low-state. **This will disable all D/I/O operations.** The user has to set the RESET\ signal to High-state before any D/I/O 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 */
```

4.3.2 AUX Control Register

(Read/Write): wBase+2

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

Note. Refer to Sec. 4.1 for more information about wBase.

Aux?=0→ this Aux is used as a D/I
Aux?=1→ this Aux is used as a D/O

When the PC is first power-on, All Aux? signal are in Low-state. All Aux? are designed as D/I for all PIO/PISO series. **Don't change this register.**

4.3.3 AUX data Register

(Read/Write): wBase+3

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

Note. Refer to Sec. 4.1 for more information about wBase.

When the Aux? is used as D/O, the output state is controlled by this register. This register is designed for feature extension, so **don't change this register.**

4.3.4 INT Mask Control Register

(Read/Write): wBase+5

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
EN7	EN6	EN5	EN4	EN3	EN2	EN1	EN0

Note. Refer to Sec. 4.1 for more information about wBase.

EN0/1/2/3/4/5/6/7=0 → **disable** INT_CHAN_0/1/2/3/4/5/6/7 as a interrupt signal (default)

EN0/1/2/3/4/5/6/7=1 → **enable** INT_CHAN_0/1/2/3/4/5/6/7 as a interrupt signal

```
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+5,4);          /* enable interrupt of INT_CHAN_2        */
outportb(wBase+5,8);          /* enable interrupt of INT_CHAN_3        */
outportb(wBase+5,0x10);       /* enable interrupt of INT_CHAN_4        */
outportb(wBase+5,0x20);       /* enable interrupt of INT_CHAN_5        */
outportb(wBase+5,0x40);       /* enable interrupt of INT_CHAN_6        */
outportb(wBase+5,0x80);       /* enable interrupt of INT_CHAN_7        */
outportb(wBase+5,0xff);       /* enable all two channels of interrupt  */
```

Refer to the following demo program for more information:

DEMO3.C → for INT_CHAN_0 only (initial high state)

DEMO4.C → for INT_CHAN_0 only (initial low state)

DEMO5.C → for multi-channel interrupt source

4.3.5 Aux Status Register

(Read/Write): wBase+7

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

Note. Refer to Sec.4.1 for more information about wBase.

Aux0=INT_CHAN_0, Aux1=INT_CHAN_1, ….., Aux7=INT_CHAN_7. The Aux0~7 are used as interrupt sources. The interrupt service routine has to read this register for interrupt source identification. Refer to Sec. 2.3 for more information.

4.3.6 Interrupt Polarity Control Register

(Read/Write): wBase+0x2A

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INV7	INV6	INV5	INV4	INV3	INV2	INV1	INV0

Note. Refer to Sec. 4.1 for more information about wBase.

INV0/1/2/3/4/5/6/7=0→ select the **invert signal** from INT_CHAN_0/1/2/3/4/5/6/7
INV0/1/2/3/4/5/6/7=1→ select the **non-invert signal** from
INT_CHAN_0/1/2/3/4/5/6/7

```
outportb(wBase+0x2a,0);      /*select the invert input from all 8 channels */
outportb(wBase+0x2a,0x3);    /* select the non-invert input from all 8 channels */

outportb(wBase+0x2a,0x2);    /* select the inverted input of
                                    INT_CHAN_0/2/3/4/5/6/7 */
                             /* select the non-inverted input of INT_CHAN_1 */
```

Refer to Sec. 2.3 for more information.
Refer to DEMO3.C, DEMO4.C, DEMO5.C and DEMO6.C for more information.

4.3.7 I/O Data Register

(Write): wBase+0xC0 → write to DO0 ~ DO7

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

Note. Refer to Sec. 4.1 for more information about wBase.

```
outportb(wBase+0xc0,0xff);              /* write 0xff to DO0~DO7 */
```

(Read): wBase+0xC0 → DO0 ~ DO7 read back **(inverse)**

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

Note. Refer to Sec. 4.1 for more information about wBase.

```
DoReadBack=inportb(wBase+0xc0) ^ 0xff;      /* DO0~DO7 read back */
```

NOTE: the read back data is inversed of DO0 ~ DO7. So the software driver has to inverse the data again to get the original DO0 ~ DO7.

(Read): wBase+0xC4 → read DI0 ~ DI7

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

Note. Refer to Sec. 4.1 for more information about wBase.

```
DI=inportb(wBase+0xc4);                 /* read DI0~DI7 */
```

5. Demo Program

5.1 Demo Programs for Windows

Please note that none of the demo programs will work normally if the DLL driver has not been installed correctly. During the DLL driver installation process, the install shield will register the correct kernel driver to the operating system and copy the DLL driver and demo programs to the correct location depending on the driver software package you have selected (Win98/Me/NT/2000 and 32-bit Win XP/2003/Visa/7). After installing the driver, the related demo programs, development library and declaration header files for the different development environments will be available in the following folders.

The demo program is contained in:
CD:\NAPDOS\PCI\PISO-725\DLL_OCX\Demo\
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/piso-725/dll_ocx/demo/

- BCB 4 → For Borland C++ Builder 4
  PISO725.H → Header files
  PISO725.LIB → Linkage library for BCB
- Delphi4 → For Delphi 4
  PISO725.PAS → Declaration files
- VB6 → For Visual Basic 6
  PISO725.BAS → Declaration files
- VC6 → For Visual C++ 6
  PISO725.H → Header files
  PISO725.LIB → Linkage library for VC6
- VB.NET2005 → For VB.NET2005
  PISO725.vb → Declaration files
- CSharp2005 → For C#.NET2005
  PISO725.cs → Declaration files

A list of available demo programs is as follows:

- DIO demo
- Interrupt demo

5.2 Demo Programs for DOS

The related DOS software and demos are located on the CD as below:
CD:\NAPDOS\PCI\PISO-725\dos\PISO725\
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/piso-725/dos/piso725/

After installing the software, the following drivers will be installed onto your hard disk:

- \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
- \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

NOTE: The library is available for all PIO/PISO series cards.

5.3 PIO_PISO for DOS

```
/* ------------------------------------------------------------ */
/* 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();
printf("\n*** PIO_PISO.EXE Rev. 2.0 ***\n\n");

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);

if ((wSubVendor==0x80)&&(wSubDevice==0x0C))/* for PISO-725 */
   printf("\nCard_%d:wBase=%x,wIrq=%x,subID=[%x,%x], SlotID=[%x,%x]",
   i,wBase,wIrq,wSubVendor,wSubDevice,wSlotBus,wSlotDevice);
else /* for the other PIO_PISO series cards */
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();
}
```

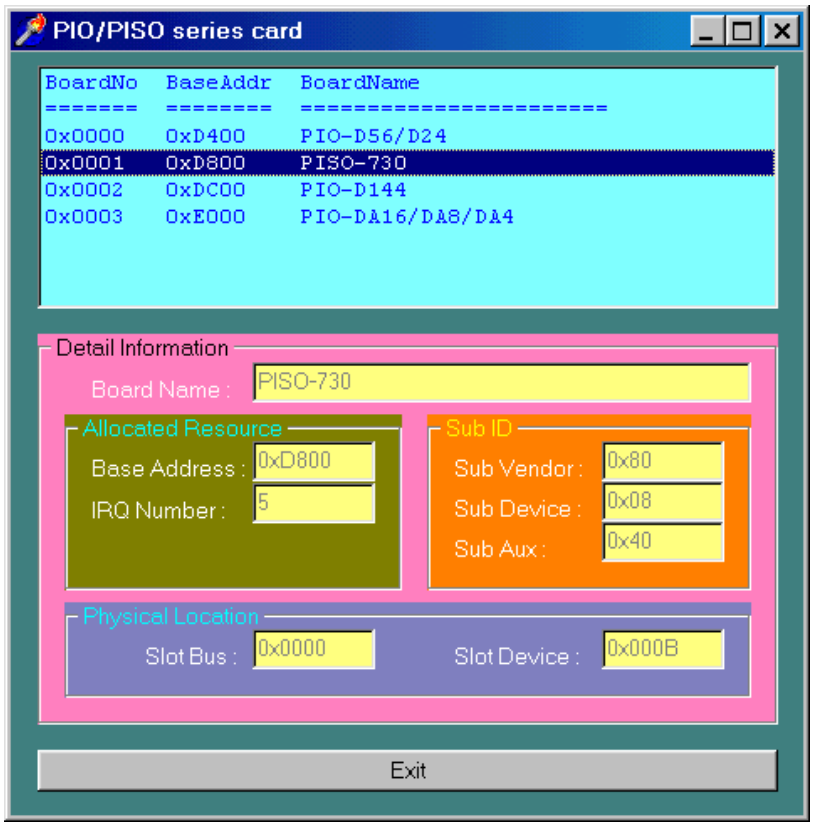
NOTE: the PIO_PISO.EXE is valid for all PIO/PISO cards. It can be find in the \TC\DIAG\ directory. The user can execute the PIO_PISO.EXE to get the following information:

- 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 Sec. 4.2 for more information)

5.4 PIO_PISO for Windows

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

After executing the utility, detailed information for all PIO/PISO cards that are installed in the PC will be shown, as illustrated below:



5.5 Demo Program for Demo

5.5.1 DEMO1: D/O Demo

```
/* ------------------------------------------------------------*/
/* DEMO1.C : PISO-725 D/O demo                                 */
/* step 1  : run DEMO1.EXE                                     */
/* ------------------------------------------------------------ */

#include "PIO.H"
void piso_725_do(char DO);
char piso_725_do_readback(void);
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;
char c,DO,ReadBack;

clrscr();
/* step 1: find address-mapping of PIO/PISO cards                  */
wRetVal=PIO_DriverInit(&wBoards,0x80,0x0C,0xff);  /* for PISO-725  */
printf("\nThrer are %d PISO-725 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],SlotID=[%x,%x]"
   ,i,wBase,wIrq,wSubVendor,wSubDevice,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");
DO=1;
for(;;)
   {
   gotoxy(1,6);
   piso_725_do(DO);
   printf("\nOutput   DO[0..7] = [%2x]",DO&0xff);

   delay(12000);
   gotoxy(1,7);
   ReadBack=piso_725_do_readback();
   printf("\nReadBack DO[0..7] = [%2x]",ReadBack&0xff);

   delay(12000);
   DO=DO<<1; if (DO==0) DO=1;

   if (kbhit()!=0) break;
   }
```

```
PIO_DriverClose();
}

/* ------------------------------------------------------------ */
void piso_725_do(char DoValue)
{
outportb(wBase+0xc0,DoValue);
}
/* ------------------------------------------------------------ */
char piso_725_do_readback(void)
{
return(inportb(wBase+0xc0) ^ 0xff);
}
```

5.5.2 DEMO2: D/I/O Demo

```
/* ------------------------------------------------------------ */
/* DEMO2.C : PISO-725 D/I/O demo                                */
/* step 1  : run DEMO2.EXE                                      */
/* ------------------------------------------------------------ */

#include "PIO.H"
void piso_725_do(char DoValue);
char piso_725_do_readback(void);
char piso_725_di(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,DI,DO,ReadBack;

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     */

printf("\n\n");
DO=1;
for(;;)
  {
  gotoxy(1,6);
  piso_725_do(DO);
  printf("\nOutput   DO[0..7] = [%2x]",DO&0xff);

  delay(12000);
  gotoxy(1,7);
  ReadBack=piso_725_do_readback();
  printf("\nReadBack DO[0..7] = [%2x]",ReadBack&0xff);

  gotoxy(1,9);
  DI=piso_725_di();
  printf("\nInput    DI[0..7] = [%2x]",DI&0xff);

  delay(12000);
  DO=DO<<1; if (DO==0) DO=1;
  if (kbhit()!=0) break;
  }
PIO_DriverClose();
}

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

char piso_725_di(void)
{
return(inportb(wBase+0xc4));
}
```

5.5.3 DEMO3: Init_High, Active_Low

```
/* ------------------------------------------------------------ */
/* DEMO3.C : PISO-725 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;

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();
}

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

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);
}
```

5.5.4 DEMO4: Init_Low, Active_High

```
/* ------------------------------------------------------------ */
/* DEMO4.C : PISO-725 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;

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);
}
```

5.5.5 DEMO5: 2-Channel Interrupt

```
/* ------------------------------------------------------------ */
/* DEMO5.C : PISO-725 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);
}
```

5.5.6 DEMO6: 8-Channel Interrupt

```
/* ------------------------------------------------------------------------ */
/* DEMO6.C : PISO-725 Interrupt (Multi interrupt source)                    */
/*           DI0/2/4/5: initial low ,  DI1/3/6/7 : initial high             */
/* step 1  : connect DI0/1/2/3/4/5/6/7     to external signals              */
/* step 2  : run DEMO6.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_H1,CNT_L2,CNT_H2;
int  CNT_L3,CNT_H3,CNT_L4,CNT_H4;
int  CNT_L5,CNT_H5,CNT_L6,CNT_H6;
int  CNT_L7,CNT_H7,CNT_L8,CNT_H8;

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     */

CNT_L1=CNT_L2=CNT_L3=CNT_L4=CNT_L5=CNT_L6=CNT_L7=CNT_L8=0;
CNT_H1=CNT_H2=CNT_H3=CNT_H4=CNT_H5=CNT_H6=CNT_H7=CNT_H8=0;
init();

printf("\n\n***** show the count of High_pulse & Low_pulse *****\n");
for(;;)
  {
  gotoxy(1,8);
  printf("\nCNT_L1/2/3/4/5/6/7/8=[%5d,%5d,%5d,%5d,%5d,%5d,%5d,%5d]",
         CNT_L1,CNT_L2,CNT_L3,CNT_L4,CNT_L5,CNT_L6,CNT_L7,CNT_L8);
  printf("\nCNT_H1/2/3/4/5/6/7/8=[%5d,%5d,%5d,%5d,%5d,%5d,%5d,%5d]",
         CNT_H1,CNT_H2,CNT_H3,CNT_H4,CNT_H5,CNT_H6,CNT_H7,CNT_H8);
  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=0x35;
outportb(wBase+0x2a,invert);      /* non-invert  DI0/2/4/5    */
                                  /*    invert  DI1/3/6/7     */
                                  /*     00110101                 */
now_int_state=0xCA;               /* now  DI0/2/4/5 = low     */
                                /* now  DI1/3/6/7 = high      */
                                /*     11001010               */
outportb(wBase+5,0xFF);    /* enable all interrupt            */
enable();
}
```

```
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       */
   }

if ((int_c&0x4)!=0)          /* now INT_CHAN_2 change to high */
   {
   if ((new_int_state&0x04)!=0)
      {
      CNT_H3++;
      }
   else                      /* now INT_CHAN_2 change to low  */
      {
      CNT_L3++;
      }
   invert=invert^4;          /* generate a high pulse       */
   }

if ((int_c&0x8)!=0)          /* now INT_CHAN_3 change to high */
   {
   if ((new_int_state&0x08)!=0)
      {
      CNT_H4++;
      }
   else                      /* now INT_CHAN_3 change to low  */
      {
      CNT_L4++;
      }
   invert=invert^8;          /* generate a high pulse       */
   }
```

```
if ((int_c&0x10)!=0)          /* now INT_CHAN_4 change to high */
   {
   if ((new_int_state&0x10)!=0)
      {
      CNT_H5++;
      }
   else                       /* now INT_CHAN_4 change to low  */
      {
      CNT_L5++;
      }
   invert=invert^0x10;              /* generate a high pulse       */
   }

if ((int_c&0x20)!=0)          /* now INT_CHAN_5 change to high */
   {
   if ((new_int_state&0x20)!=0)
      {
      CNT_H6++;
      }
   else                       /* now INT_CHAN_5 change to low  */
      {
      CNT_L6++;
      }
   invert=invert^0x20;              /* generate a high pulse       */
   }

if ((int_c&0x40)!=0)          /* now INT_CHAN_6 change to high */
   {
   if ((new_int_state&0x40)!=0)
      {
      CNT_H7++;
      }
   else                       /* now INT_CHAN_6 change to low   */
      {
      CNT_L7++;
      }
   invert=invert^0x40;              /* generate a high pulse        */
   }

if ((int_c&0x80)!=0)          /* now INT_CHAN_7 change to high  */
   {
   if ((new_int_state&0x80)!=0)
      {
      CNT_H8++;
      }
   else                       /* now INT_CHAN_7 change to low    */
      {
      CNT_L8++;
      }
   invert=invert^0x80;              /* 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);
}
```