

PISO-730

User Manual

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 assume 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 1999 by ICP DAS. All rights are reserved.

Trademark

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

Tables of Contents

1. INTRODUCTION	3
1.1 FEATURES	3
1.2 SPECIFICATIONS	4
1.3 ORDER DESCRIPTION.....	5
1.4 PCI DATA ACQUISITION FAMILY	5
1.5 PRODUCT CHECK LIST	6
2. HARDWARE CONFIGURATION.....	7
2.1 BOARD LAYOUT	7
2.2 I/O OPERATION	8
2.3 INTERRUPT OPERATION.....	12
2.4 DAUGHTER BOARDS.....	20
2.5 PIN ASSIGNMENT	24
3. I/O CONTROL REGISTER.....	26
3.1 HOW TO FIND THE I/O ADDRESS	26
3.2 THE ASSIGNMENT OF I/O ADDRESS	31
3.3 THE I/O ADDRESS MAP.....	32
4. DEMO PROGRAM.....	37
4.1 PIO_PISO	38
4.2 DEMO1	40
4.3 DEMO2	42
4.4 DEMO3	43
4.5 DEMO4	45
4.6 DEMO5	47

1. Introduction

The PISO-730 provides 32 channels isolated digital I/O (16×DI and 16×DO) and 32 channels TTL-level digital I/O (16×DI and 16×DO). The board interface to field logic signals, eliminating ground-loop problems and isolating the host PC from damaging voltages. Its isolated I/O channels provide up to 2,500Vdc of protection.

The PISO-730 has one 37-pin D-type connector and two 20-pin flat-cable connectors. The flat-cable can be connected to ADP-20/PCI adapter. The adapter can be fixed on the chassis. It can be installed in a 5V PCI slot and can support truly “Plug and Play”.

1.1 Features

- PCI Bus
- 32 isolated DIO channels (16×DI and 16×DO)
- 32 TTL-level DIO channels (16×DI and 16×DO)
- DC/DC converter build-in
- One DB-37 D-type connector for isolated input and output
- Two separate 20-pin connectors for non-isolated input and output
- Up to 2500Vdc isolated voltage
- Interrupt source: 2 channels
- Connects directly to DB-24PR, 24POR, DB-24C, DB-16P, DB-16R
- SMD, short card, power saving
- Automatically detected by Windows 95/98/NT
- No base address or IRQ switches to set

1.2 Specifications

Isolated DIO channels

- **Optical isolated input channel**
Channel NO.: 16 digital inputs
Type: Isolated current input
Isolation voltage: 2,500Vdc
Input voltage: 3.5-30Vdc
Input impedance: 1.2K Ω /1W
Response time: 10KHz (Max.)
- **Optical isolated output channel**
Channel NO.: 16 digital outputs
Output voltage: open-collector 5-30Vdc
Isolation voltage: 2,500Vdc
Sink current: 200mA (Max.)
Response time: 10KHz (Max.)

TTL-level DIO channels

- **TTL-level input channel**
Channel NO.: 16 digital input
Input voltage level: (TTL compatible)
 V_{IL} : 0.8V (Max.)
 V_{IH} : 2.4V (Min.)
- **TTL-level output channel**
Channel NO.: 16 digital output
Driver capacity: (TTL compatible)
 I_{OL} : 8 mA (sink)
 I_{OH} : 0.4 mA (source)

General specifications

- Operation Temp.: 0-50°C
- Storage Temp.: -20°C to 70°C
- Humidity: 0-90%, non-condensing
- Dimensions: 180mm×105mm
- Power consumption: 640mA

1.3 Order Description

- PISO-730: PCI bus 32 channel isolated digital I/O board

1.3.1 Options

- DB-24PR, DB-24PR: 24 channels power relay board
- DB-24POR: 24 channel PhotoMos output board
- DB-24C: 24 channel open-collector output board
- DB-16P: 16 channel isolated D/I board
- DB-16R: 16 channel relay board
- ADP-20/PCI: Extender, 20-pin header to 20-pin header for PCI bus I/O boards
- 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
- NAPPCI/win: DLLs for Windows 95/98
- NAPPCI/wnt: DLLs for Windows NT 4.0
- NAPVIEW/1: LabVIEW driver for Windows 95/98
- NAPVIEW/2: LabVIEW driver for Windows NT

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 follows:

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-823/821: multi-function family

PIO-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/P64/C64: D/I/O family

PISO-P8R8/P8SSR8AC/P8SSR8DC: D/I/O family

PISO-730: D/I/O card

PISO-DA2: D/A card

1.5 Product Check List

In addition to this manual, the package includes the following items:

- one piece of PISO-730 card
- one piece of company floppy diskette or CD
- one piece of release note

It is recommended to read the release note firstly. All importance information will be given in release note as follows:

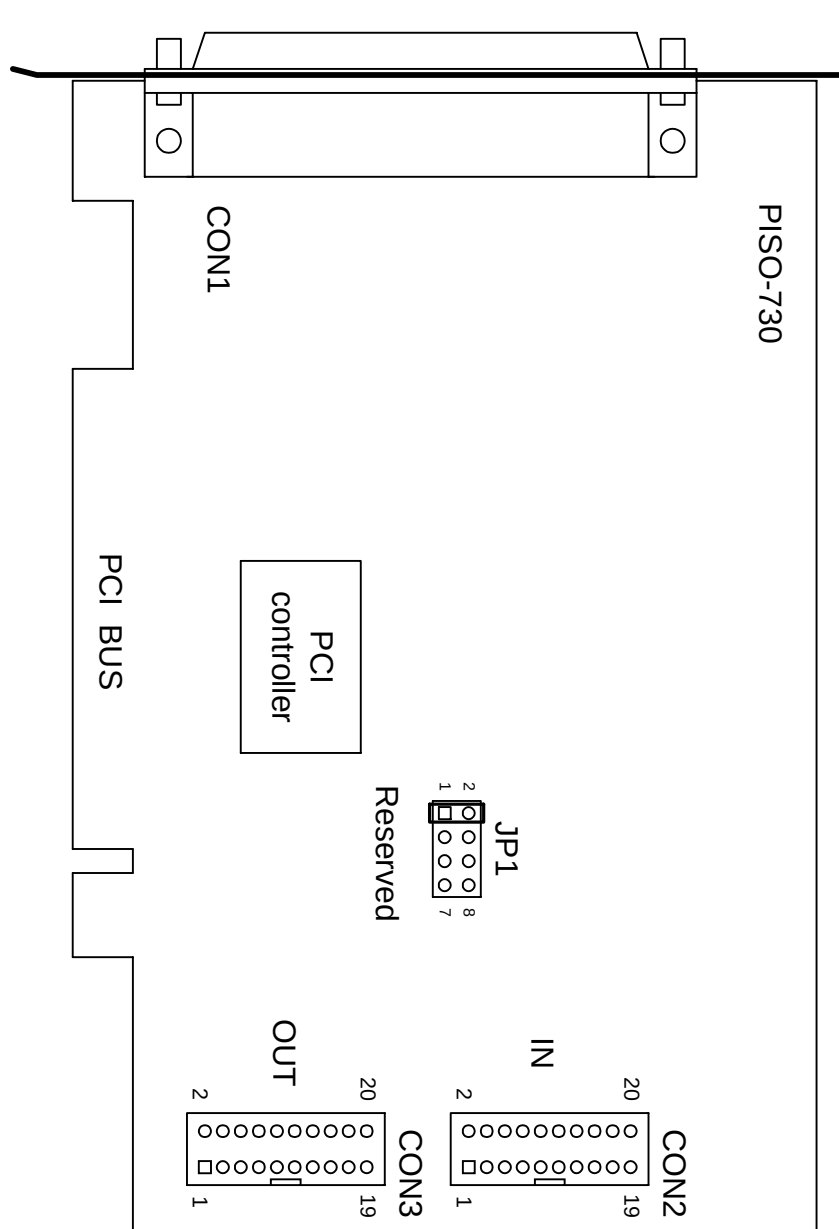
1. where you can find the software driver & utility
2. how to install software & utility
3. where is the diagnostic program
4. FAQ

Attention!

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 want to ship or store the product in the future.

2. Hardware configuration

2.1 Board Layout



CON1: 16 channels isolated D/I and 16 channels isolated D/O

CON2: 16 channels TTL-level (non-isolated) D/I

CON3: 16 channels TTL-level (non-isolated) D/O

JP1: Reserved

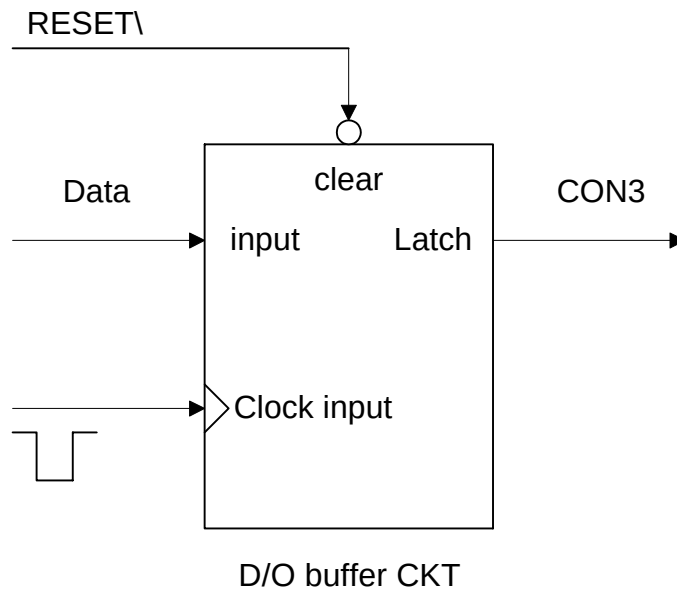
2.2 I/O Operation

2.2.1 Non-isolated DO Port Architecture (CON3)

When the PC is power-up, all operations of non-isolated DO states are clear to low-state. The RESET\ signal is used to clear non-isolated DO states. Refer to Sec. 3.3.1 for more information about RESET\ signal.

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

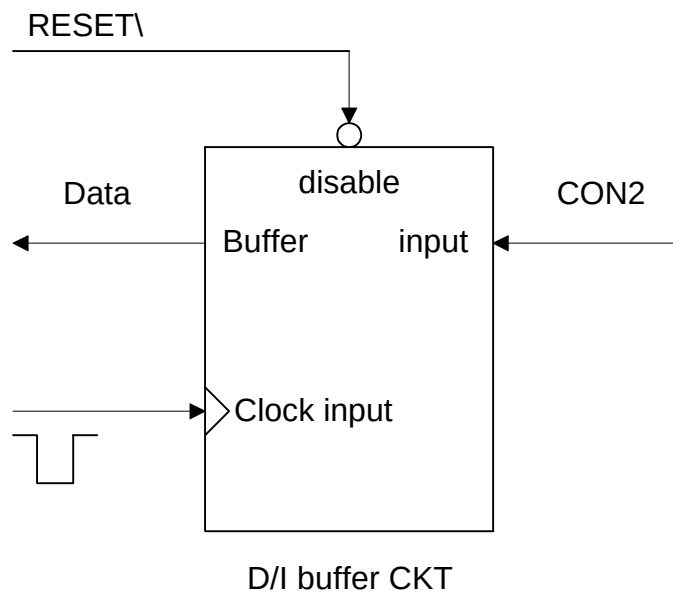
The block diagram of non-isolated DO is given as follows:



2.2.2 Non-isolated DI Port Architecture (CON2)

When the PC is power-up, all operations of non-isolated DI port are disable. The enable/disable of non-isolated DI port 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 non-isolated DI operation is disable
- The RESET\ is in High-state → all non-isolated DI operation is enable



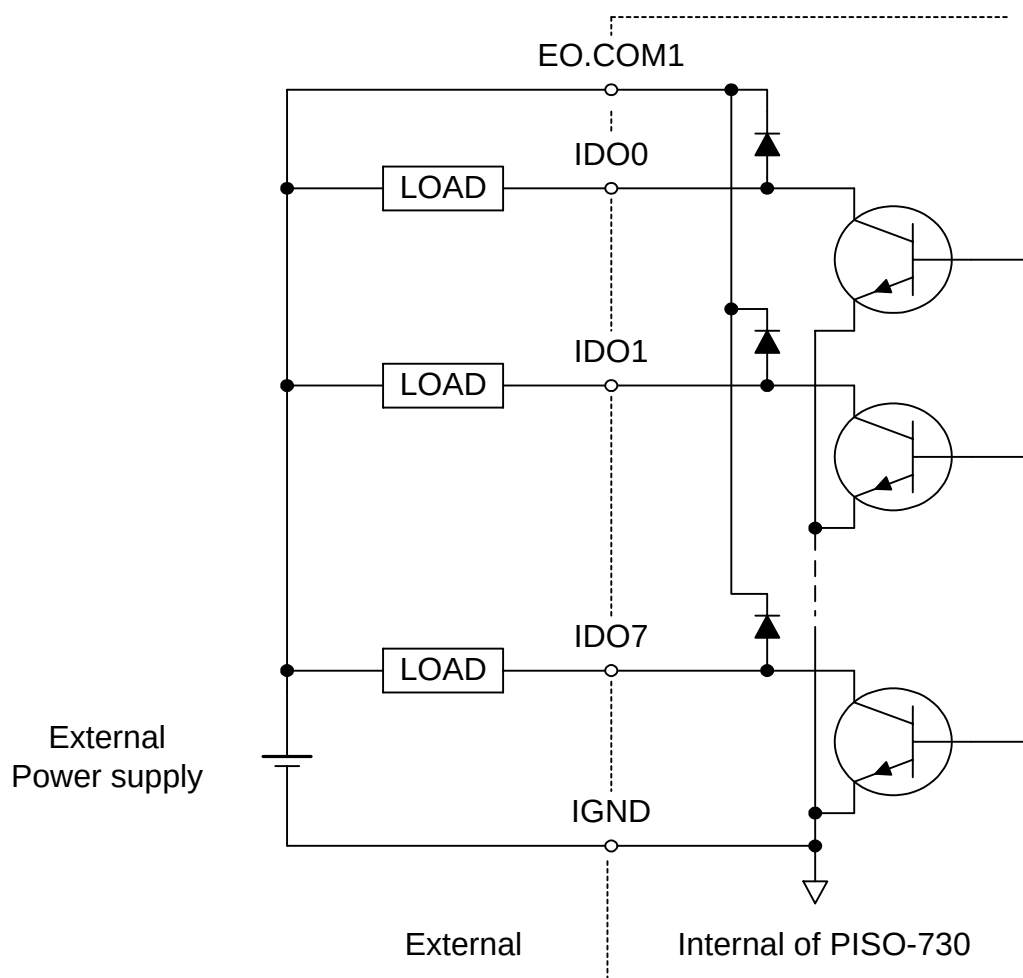
2.2.3 Isolated DO Port Architecture (CON1)

When the PC is power-up, all operations of isolated DO states are clear to low-state. The RESET\ signal is used to clear isolated DO states. Refer to Sec. 3.3.1 for more information about RESET\ signal.

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

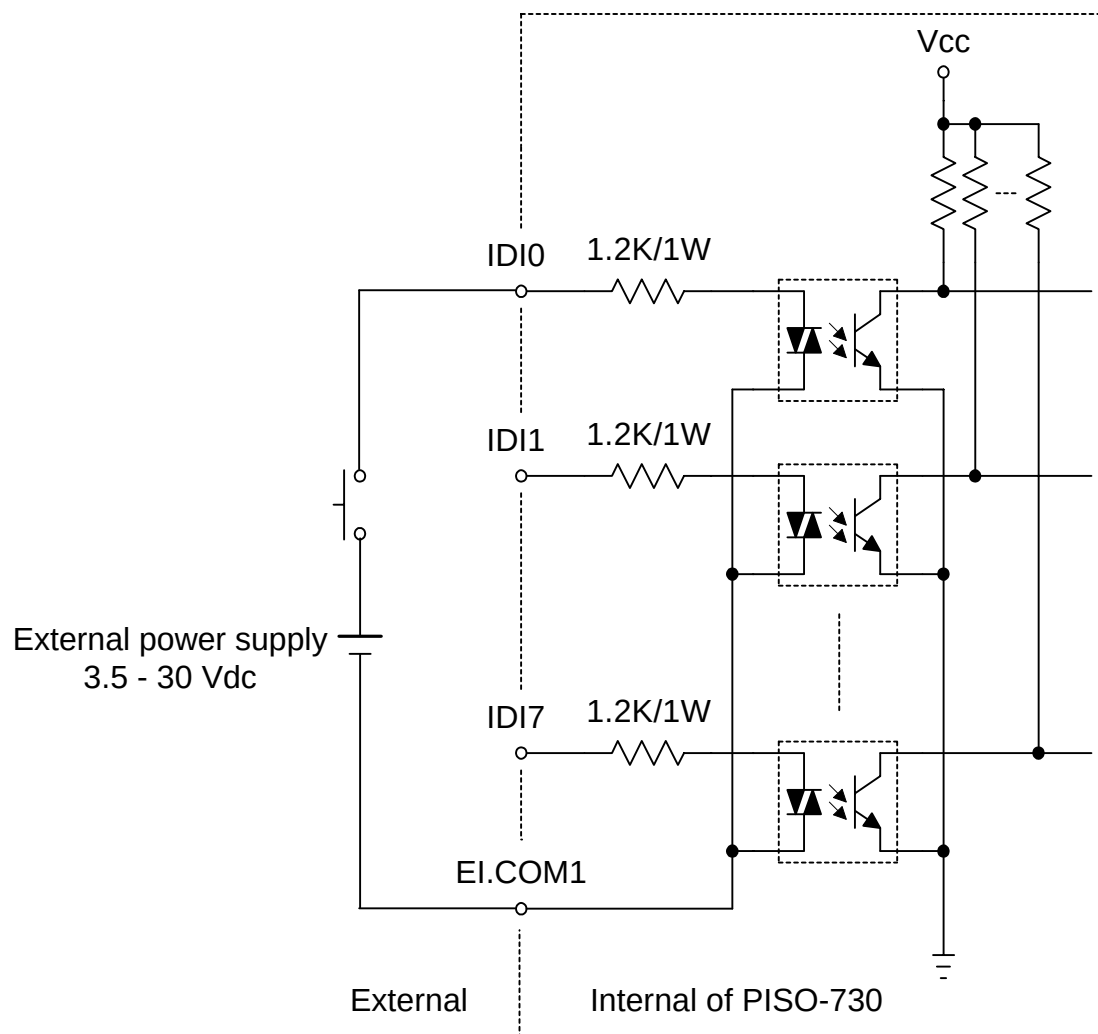
Each eight open-collector output channels share EO.COM(IDO0~IDO7 use EO.COM1 and IDO8~IDO15 use EO.COM2)

The block diagram of isolated DO is given as follows:



2.2.4 Isolated DI Port Architecture (CON1)

The PISO-730 provides 16 channels isolated digital input. Each of the isolated digital input accepts voltages from 3.5-30Vdc. Each eight input channels share one external common end point. (IDI0~IDI7 use EI.COM1 and IDI8~IDI15 use EI.COM2)



2.3 Interrupt Operation

There are two interrupt sources in PISO-730. 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

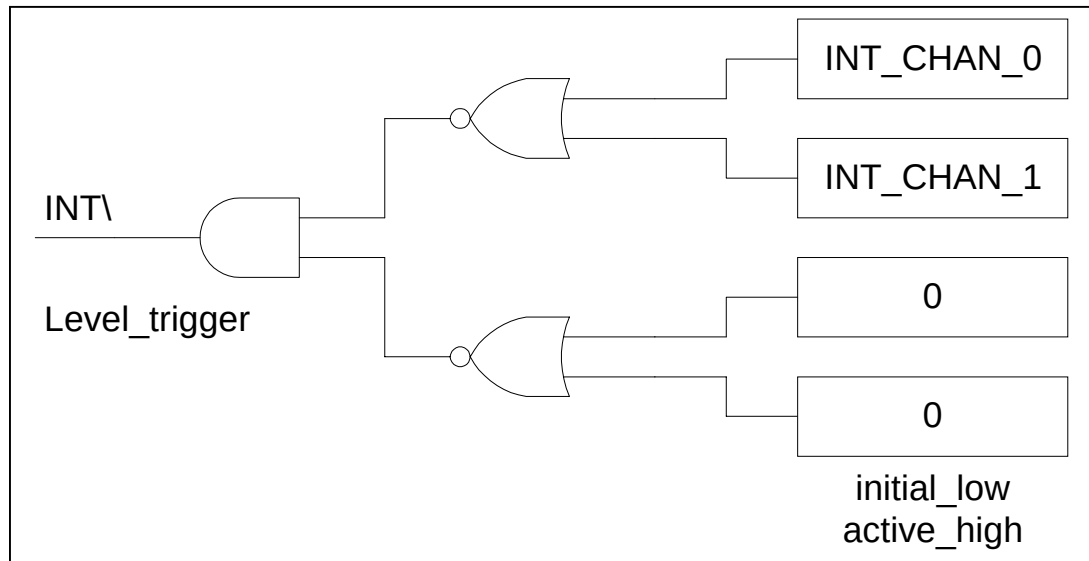
If only one interrupt signal source is used, the interrupt service routine does not have to identify the interrupt source. Refer to DEMO3.C and DEMO4.C for more information.

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

1. Read the new status of all interrupt signal sources(refer to Sec 3.3.5)
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. 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.

2.3.1 Interrupt Block Diagram of PISO-730



The interrupt output signal of PISO-730, `INT\` is **level-trigger & Active_Low**. If the `INT\` generate a low-pulse, the PISO-730 will interrupt the PC once a time. If the `INT\` is fixed in low level, the PISO-730 will interrupt the PC continuously. So the `INT_CHAN_0/1` 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.**

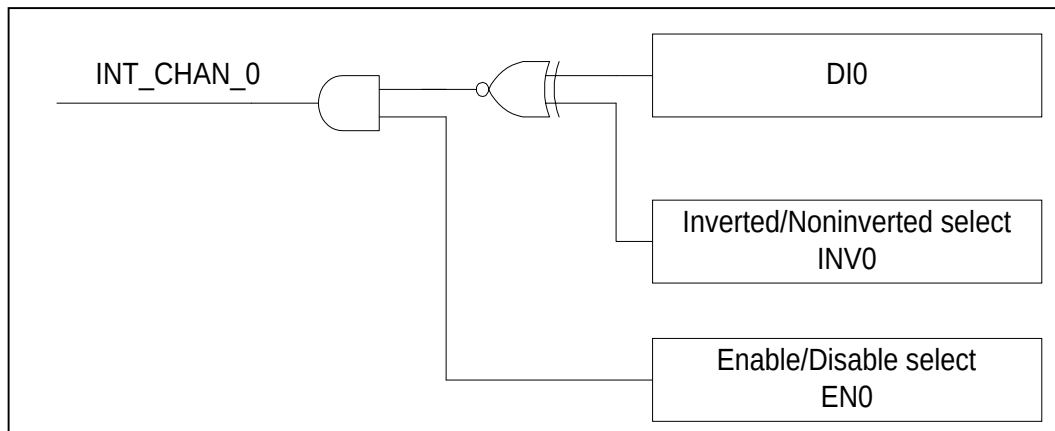
The priority of `INT_CHAN_0/1` is the same. If all these two signals are active at the same time, then `INT\` will be active only once a time. So the interrupt service routine has to read the status of all interrupt channels for multi-channel interrupt. Refer to Sec. 2.6.7 for more information.

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 program DEMO3.C and DEMO4.C are designed for single-channel interrupt demo as follows:

DEMO3.C & 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 follows: (refer to Sec. 3.3.4)

EN0=0 → INT_CHAN_0=disable

EN0=1 → INT_CHAN_0=enable

The INV0 can be used to invert/non-invert the DIO as follows: (Refer to Sec. 3.3.6)

INV0=0 → INT_CHAN_0=invert state of DIO

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

Refer to demo program for more information as follows:

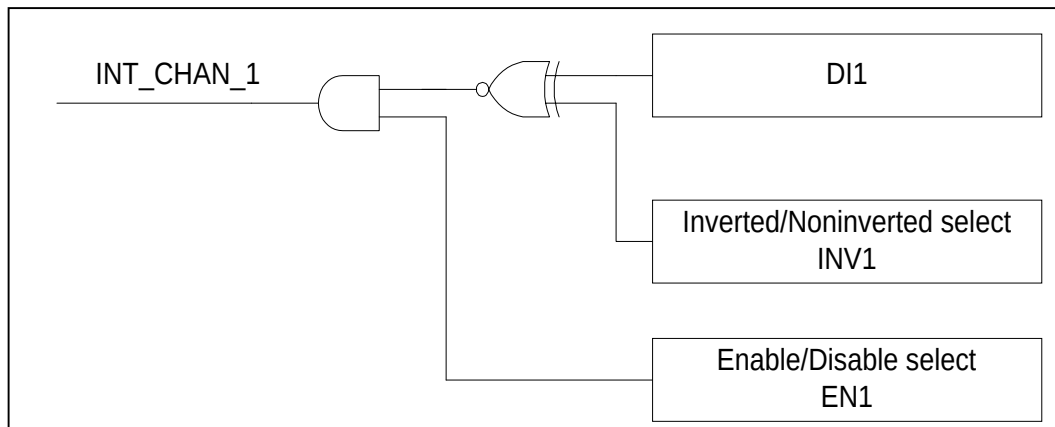
DEMO3.C → for INT_CHAN_0 (initial high)

DEMO4.C → for INT_CHAN_0 (initial low)

DEMO5.C → for multi-channel interrupt source

NOTE: Refer to Sec. 2.3.4 & Sec. 2.3.5 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 follows: (refer to Sec. 3.3.4)

EN1=0 → INT_CHAN_1=disable

EN1=1 → INT_CHAN_1=enable

The INV1 can be used to invert/non-invert the DI1 as follows: (Refer to Sec. 3.3.6)

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 follows:

DEMO3.C → for INT_CHAN_0 (initial high)

DEMO4.C → for INT_CHAN_0 (initial low)

DEMO5.C → for multi-channel interrupt source

NOTE: Refer to Sec. 2.3.4 & Sec. 2.3.5 for active high-pulse generation.

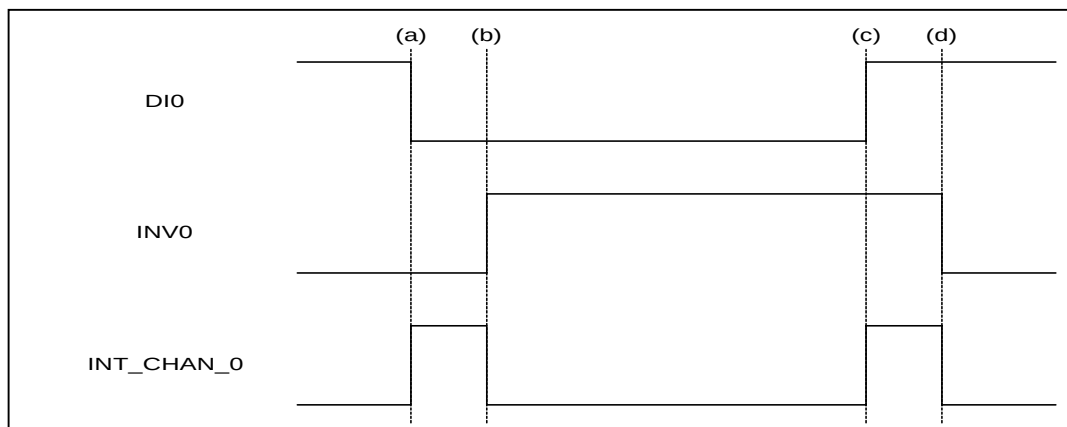
2.3.4 Initial_high, active_low Interrupt source

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 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      */(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);
}
```



2.3.5 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 DEMO4.C and the DI1 is similarly)

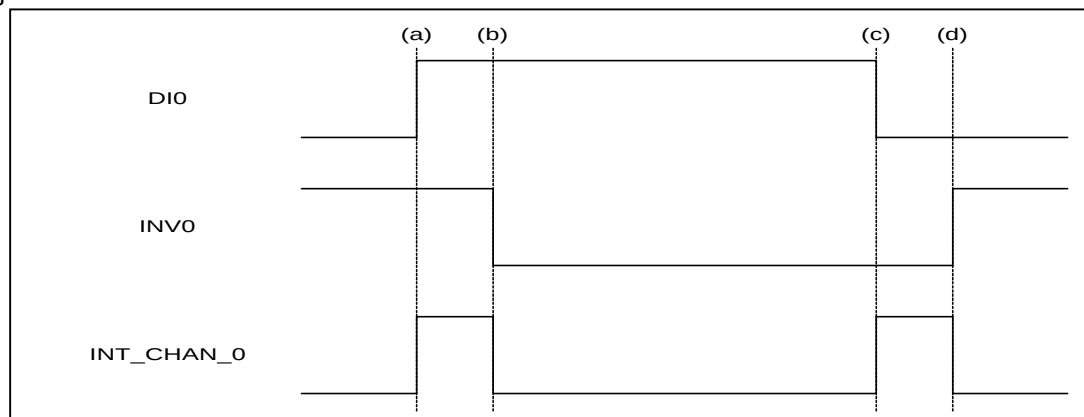
Initial set:

```

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      */(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);
}

```

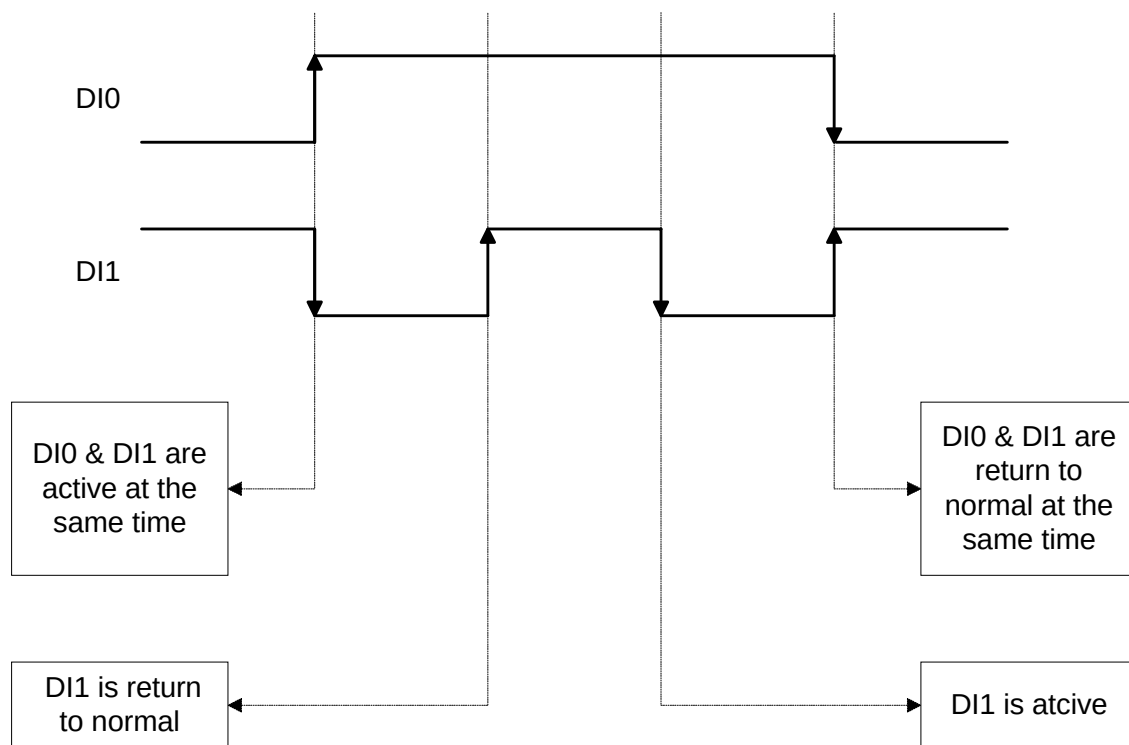


2.3.6 Multiple Interrupt Source

Assume: DI0 is initial Low, active High

DI1 is initial High, active Low

as follows:



Refer to DEMO5.C for source program. **All these three falling-edge & 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);
```

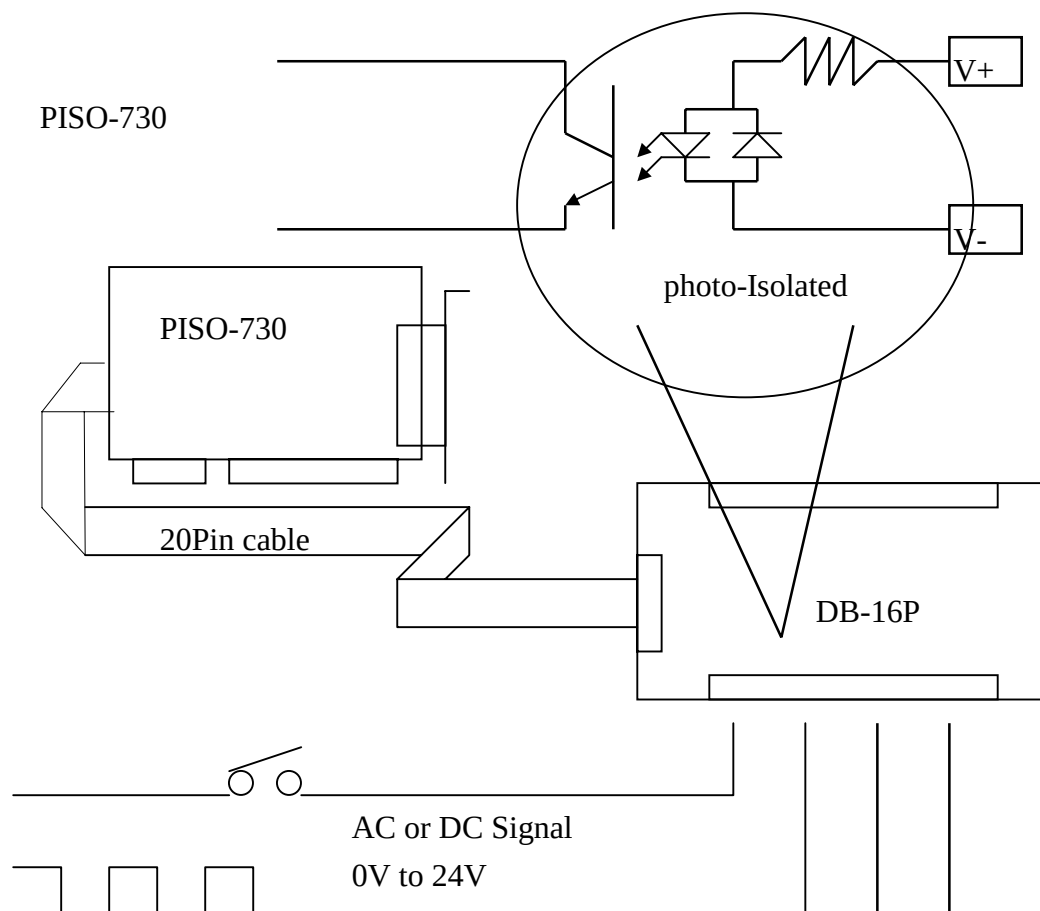
```
void interrupt irq_service()
{
    new_int_state=inportb(wBase+7)&0x03; /* read all interrupt state */
    int_c=new_int_state^now_int_state;    /* compare which interrupt */
                                        /* signal be change */
    if ((int_c&0x1)!=0)                  /* INT_CHAN_0 is active */
    {
        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);
}
```

2.4 Daughter Boards

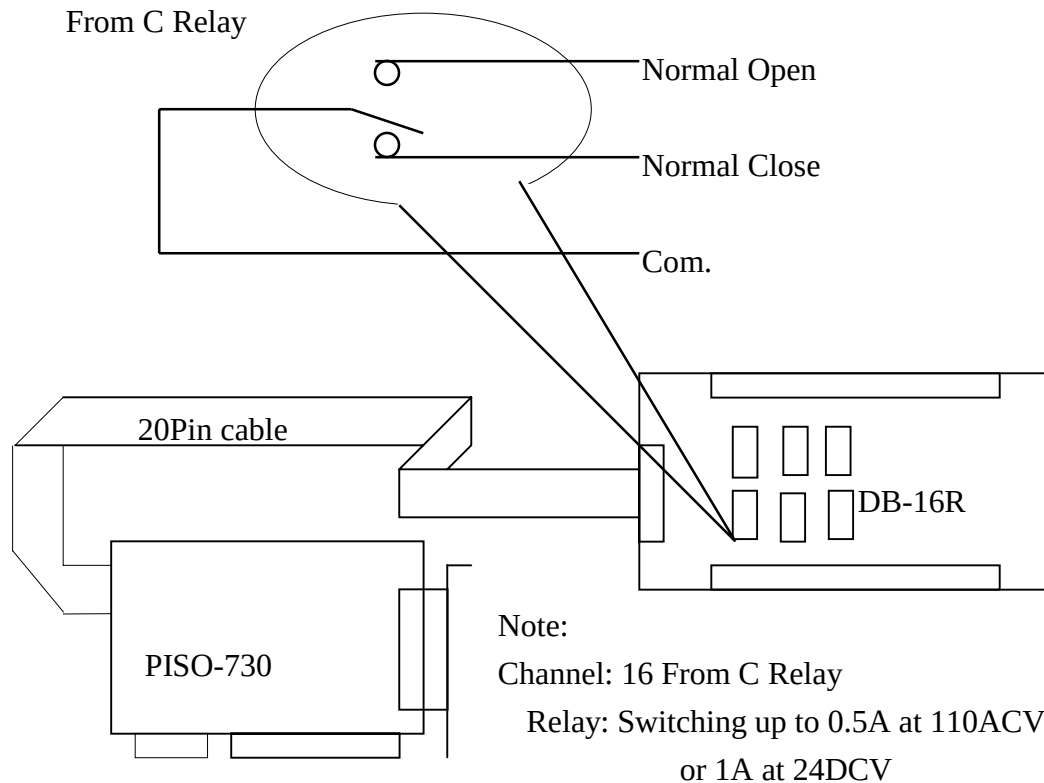
2.4.1 DB-16P Isolated Input Board

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



2.4.2 DB-16R Relay Board

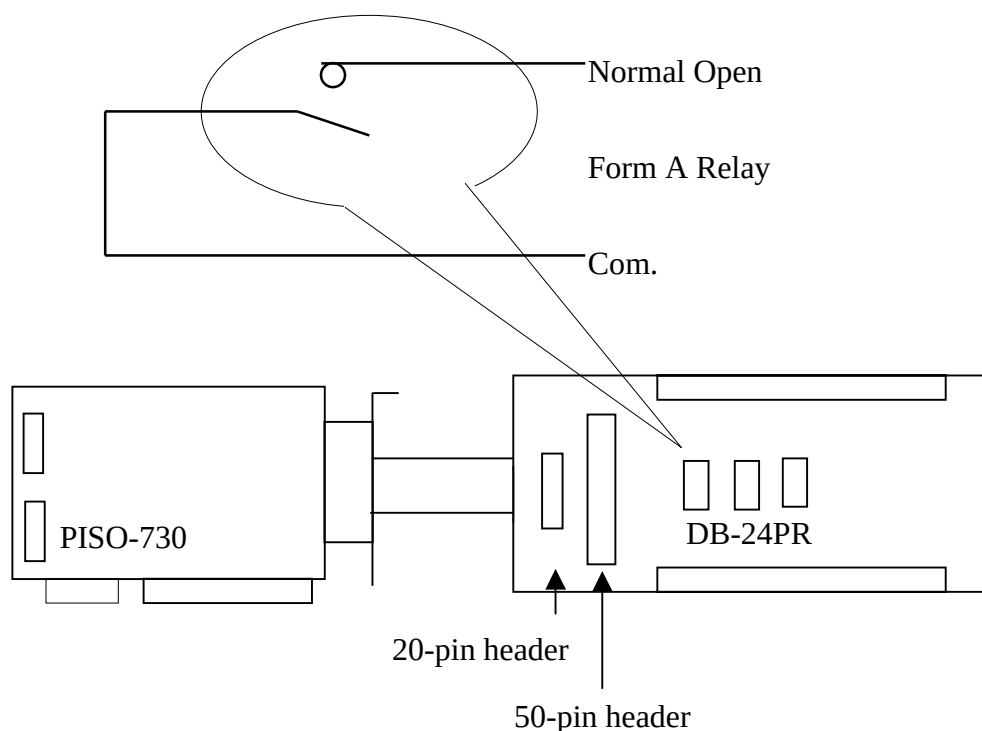
The DB-16R is a 16-channels relay output board, consists of 16 form C relays for efficient switch of load by programmed control. The relay are energized by applying 12V/24V voltage signal to the appropriated relay channel on the 20-pin flat connector. There are 16 enunciator LEDs for each relay, light when their associated relay is activated.



2.4.3 DB-24PR, DB-24POR, DB-24C

DB-24PR	24*power relay, 5A/250V
DB-24POR	24*photoMOS relay, 0.1A/350VAC
DB-24C	24*open collector, 100mA per channel, 30V max.

The DB-24PR, is a 24-channel power relay output board, consists of 8 form C and 16 form A electromechanical relays for efficient switching of load programmed control. The contact of each relay can control a 5A load at 250ACV/30VDCV. The relay is energized by applying a 5 voltage signal to the appropriate 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, one for each relay, light when their associated relay is activated. To avoid overloading your PC's power supply, this board needs a +12VDC or +24VDC external power supply.



Note:

50-Pin connector (OPTO-22 compatible), for DIO-24, DIO-48, DIO-144,
PIO-D144, PIO-D96, PIO-D56, PIO-D48, PIO-D24

Channel: 16 Form A Relays, 8 Form C Relay

Relay: switching up to 5A at 110ACV / 5A at 30DCV

2.4.4 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 has two 20-pin flat-cable header, and one 37 pin D-type connector

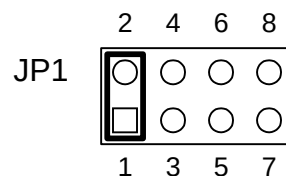
2.5 Pin Assignment

2.5.1 Isolated I/O connector

CON1: 37 pin of D-type female connector

Pin No.	Description	Pin No	Description
1	IDI0	20	IDI1
2	IDI2	21	IDI3
3	IDI4	22	IDI5
4	IDI6	23	IDI7
5	IDI8	24	IDI9
6	IDI10	25	IDI11
7	IDI12	26	IDI13
8	IDI14	27	IDI15
9	EI.COM1	28	EI.COM2
10	EO.COM1	29	IGND
11	IDO0	30	IDO1
12	IDO2	31	IDO3
13	IDO4	32	IDO5
14	IDO6	33	IDO7
15	IDO8	34	IDO9
16	IDO10	35	IDO11
17	IDO12	36	IDO13
18	IDO14	37	IDO15
19	EO.COM2		

2.5.2 JP1



Note: Reserved

2.5.3 TTL-level I/O connector

CON2: 20-pin of flat-cable connector

Pin no.	Description	Pin no.	Description
1	DI0	2	DI1
3	DI2	4	DI3
5	DI4	6	DI5
7	DI6	8	DI7
9	DI8	10	DI9
11	DI10	12	DI11
13	DI12	14	DI13
15	DI14	16	DI15
17	D.GND	18	D.GND
19	+5V	20	+12V

CON3: 20-pin of flat-cable connector

Pin no.	Description	Pin no.	Description
1	DO0	2	DO1
3	DO2	4	DO3
5	DO4	6	DO5
7	DO6	8	DO7
9	DO8	10	DO9
11	DO10	12	DO11
13	DO12	14	DO13
15	DO14	16	DO15
17	D.GND	18	D.GND
19	+5V	20	+12V

All signals are TTL compatible

3. I/O Control Register

3.1 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 card are given as follows:

- **Vendor ID = E159**
- **Device ID = 0002**

The sub IDs of **PISO-730** are given as follows:

- **Sub-vendor ID= 80**
- **Sub-device ID = 08**
- **Sub-aux ID = 40**

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 PIO.H. Refer to Chapter 4 for more information. The important driver information is given as follows:

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: subAux ID of this board

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 & show all PIO/PISO cards installed in this PC. Refer to Sec. 4.1 for more information.

3.1.1 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 → subAux ID of board to find

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-730 in this PC

```
wSubVendor=0x80; wSubDevice=8; wSubAux=0x40; /* for PISO-730 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("Threr are %d PISO-730 Cards in this PC\n",wBoards);

/* step2: save resource of all PISO-730 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. 4.1 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 follows:

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 + 32	80	08	20
PISO-P8R8	8* isolated D/I + 8 * 220V 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

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

3.1.2 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 → subAux ID of this board
- 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 follows:

```
/* step1: detect all PISO-730 cards first */
wSubVendor=0x80; wSubDevice=8; wSubAux=0x40; /* for PISO-730 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("Threr are %d PISO-730 Cards in this PC\n",wBoards);

/* step2: save resource of all PISO-730 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-730 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 */
```

3.1.3 Show_PIO_PISO

Show_PIO_PISO(wSubVendor,wSubDevice,wSubAux)

- wSubVendor → subVendor ID of board to find
- wSubDevice → subDevice ID of board to find
- wSubAux → subAux ID of board to find

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 follows:

```
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);
}
```

3.2 The Assignment of I/O Address

The plug & play BIOS will assign the proper I/O address to PIO/PISO series card. If there is only one PIO/PISO board, the user can identify the board as card_0. If there are two PIO/PISO boards in the system, the user will be very difficult to identify which board is card_0 ? The software driver can support 16 boards max. Therefore the user can install 16 boards of PIO/PSIO series in one PC system. How to find the card_0 & card_1 ?

It is difficult to find the card NO. The simplest way to identify which card is card_0 is to use wSlotBus & wSlotDevice as follows:

1. Remove all PISO-730 from this PC
2. Install one PISO-730 into the PC's PCI_slot1, run PIO_PISO.EXE & record the wSlotBus1 & wSlotDevice1
3. Remove all PISO-730 from this PC
4. Install one PISO-730 into the PC's PCI_slot2, run PIO_PISO.EXE & record the wSlotBus2 & wSlotDevice2
5. repeat (3) & (4) for all PCI_slot?, record all wSlotBus? & wSlotDevice?

The records may be as follows:

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 will record all wSlotBus? & wSlotDevice? in this PC. These values will be mapped to this PC's physical slot. This mapping will not be changed for any PIO/PISO cards. So it can be used to identify the specified PIO/PISO card as follows:

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 compare the wSlotBus & wSlotDevice in step2 to step1.

3.3 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 of PISO-730 are given as follows:

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	IDI0~IDI7	IDO0~IDO7
wBase+0xc4	IDI8~IDI15	IDO8~IDO15
wBase+0xc8	DI0~DI7	DO0~DO7
wBase+0xcc	DI8~DI15	DO8~DO15

Note. Refer to Sec. 3.1 for more information about wBase.

3.3.1 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. 3.1 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 */
```

3.3.2 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. 3.1 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. Please set all Aux? in D/I state.

3.3.3 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. 3.1 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 control this register now.

3.3.4 INT Mask Control Register

(Read/Write): wBase+5

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	EN1	EN0

Note. Refer to Sec. 3.1 for more information about wBase.

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

EN0/1=1 → enable INT_CHAN_0/1 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+5,3);      /* 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

3.3.5 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. 3.1 for more information about wBase.

Aux0=INT_CHAN_0, Aux1=INT_CHAN_1, Aux7~4=Aux-ID. Refer to Sec. 4.1 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 Sec. 2.5 for more information.

3.3.6 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	0	0	INV1	INV0

Note. Refer to Sec. 3.1 for more information about wBase.

INV0/1=0→ select the invert signal from INT_CHAN_0/1

INV0/1=1→ select the non-invert signal from INT_CHAN_0/1

```
outportb(wBase+0x2a,0);    /* select the invert input from all 2 channels    */
```

```
outportb(wBase+0x2a,0x3); /* select the non-invert input from all 2 channels */
```

```
outportb(wBase+0x2a,0x2); /* select the inverted input of INT_CHAN_0    */
```

```
/* select the non-inverted input of INT_CHAN_1 */
```

Refer to Sec. 2.6.7 for more information.

Refer to DEMO5.C for more information.

3.3.7 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

Note. Refer to Sec. 3.1 for more information about wBase.

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

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

4. Demo Program

It is recommended to read the release note first. All important information will be given in release note as follows:

1. where you can find the software driver & utility
2. **how to install software & utility**
3. where is the diagnostic program
4. FAQ

There are many demo programs given in the company floppy disk or CD. After the software installation, the driver will be installed into disk as follows:

- \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.

4.1 PIO_PISO

```
/* ----- */
/* 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();
}
```

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 & wSlotDevice for specified PIO/PISO card identification.
(refer to Sec. 3.2 for more information)

4.1.1 PIO_PISO.EXE for Windows

There has an software utility “PIO_PISO.EXE” for Windows95/98 for the detailed information about this file, please refer to the “Readme.txt” of development toolkit for Windows95/98. It is useful for all PIO/PIS series card.

The setup steps from the CD-ROM are given as follows:

- Step1: Toolkit(Software)/Manuals
- Step2: T Agree
- Step3: PCI Bus DAQ Card
- Step4: PIO_PISO
- Step5: Install Toolkits for Windows95/98
- Step6: After installation, this program will be extracted in user define directory.

After executing the utility, every detail information for all PIO/PISO cards that installed in the PC will be shown as follows:

The screenshot shows a window titled "PIO/PISO series card". It contains a table of installed cards and a section for detailed information.

BoardNo	BaseAddr	BoardName
0x0000	0xD400	PIO-D56/D24
0x0001	0xD800	PISO-730
0x0002	0xDC00	PIO-D144
0x0003	0xE000	PIO-DA16/DAB/DA4

Below the table is a section titled "Detail Information" for the selected "PISO-730" card.

Board Name : PISO-730

Allocated Resource

Base Address : 0xD800
IRQ Number : 5

Sub ID

Sub Vendor : 0x00
Sub Device : 0x08
Sub Aux : 0x40

Physical Location

Slot Bus : 0x0000
Slot Device : 0x000B

At the bottom of the window is an "Exit" button.

4.2 DEMO1

```
/* DEMO1.C : PISO-730 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 PISO-730 */
    printf("\nThrre are %d PISO-730 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 ID0[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));
}

```

4.3 DEMO2

```
/* DEMO2.C : PISO-730 D/I/O demo */
/* step 1 : connect D0[0..15] to DI[0..15], */
/* ID0[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 D0[0..15]=[%4lx] , DI[0..15]=[%4lx]", lOutPad1, lInPad1);
        printf("\n ID0=[%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);
}
```

4.4 DEMO3

```
/* DEMO3.C : PISO-730 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();
}

/* ----- */
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);
}

```

4.5 DEMO4

```
/* DEMO4.C : PISO-730 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);
}

```

4.6 DEMO5

```
/* DEMO5.C : PISO-730 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);
    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);
    if (wIrq<8)
    {
        setvect(wIrq+8,oldfunc);
    }
    else
    {
        setvect(wIrq-8+0x70,oldfunc);
    }
    PIO_DriverClose();
}
/* ----- */
WORD init()
{
    DWORD dwVal;
    disable();
    outportb(wBase+5,0);
    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);
}

```