

PIO-D96

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 Content

1. INTRODUCTION	3
1.1 FEATURES	3
1.2 SPECIFICATIONS	4
1.3 ORDER DESCRIPTION.....	4
1.4 PCI DATA ACQUISITION FAMILY	5
1.5 PRODUCT CHECK LIST	5
2. HARDWARE CONFIGURATION.....	6
2.1 BOARD LAYOUT	6
2.2 I/O PORT LOCATION	7
2.3 ENABLE I/O OPERATION.....	7
2.4 D/I/O ARCHITECTURE	8
2.5 INTERRUPT OPERATION.....	9
2.6 DAUGHTER BOARDS.....	16
2.7 PIN ASSIGNMENT	22
3. I/O CONTROL REGISTER.....	24
3.1 HOW TO FIND THE I/O ADDRESS.....	24
3.2 THE ASSIGNMENT OF I/O ADDRESS	29
3.3 THE I/O ADDRESS MAP.....	30
4. DEMO PROGRAM	36
4.1 PIO_PISO	37
4.2 DEMO1	39
4.3 DEMO2	39
4.4 DEMO3	40
4.5 DEMO4	43
4.6 DEMO5	45

1. Introduction

The PIO-D96 provides 96 TTL digital I/O lines. The PIO-D96 consists of four 24-bit bi-directional ports. Each 24-bit port supports three 8-bit groups PA, PB & PC. Each 8-bit group can be configured to function as either inputs or latched outputs. All groups are configured as inputs upon power-up or reset.

The PIO-D96 has one D-Sub connector and three 50-pin flat cable headers. Each header can connect to a 50-pin flat cable. The flat cable can be connected to ADP-37/PCI or ADP-50/PCI adapter. The adapter can be fixed on the chassis. It can be installed in 5V PCI bus and can support truly "Plug & Play".

1.1 Features

- PCI bus
- Up to 96 channels of digital I/O
- All I/O lines buffered on the board
- Eight-bit groups independently selectable for I/O on each 24-bit port
- Input/ Output programmable I/O ports under software control
- SMD, short card, power saving
- Connects directly to DB-24P, DB-24PR, DB-24PD, DB-24RD, DB-24PRD, DB-16P8R, DB-24POR, DB-24SSR, DB-24C or any OPTO-22 Compatible daughter boards
- Each board = 4 connector = 4×3 port = 4×3×8 bit =96 bit
- 4 interrupt sources: P2C0,P5C0,P8C0,P11C0 (Refer to sec. 2.7)
- One D-sub connector, three 50-pin flat cable connectors
- Automatically detected by windows 95/98/NT
- No base address or IRQ switches to set

1.2 Specifications

- **All inputs are TTL compatible**
Logic high voltage: 2.4V (Min.)
Logic low voltage: 0.8V (Max.)
- **All outputs are TTL compatible**
Sink current: 64mA (Max.)
Source current: 32mA(Max.)
- Environmental :
Operating Temperature: 0°C to 60°C
Storage Temperature: -20°C to 80°C
Humidity: 0 to 90% non-condensing
- Dimension: 180mm X 105mm
- Power Consumption: +5V @ 600mA

1.3 Order Description

- PIO-D96 : PCI bus 96-bit opto-22 board

1.3.1 Options

- DB-24P, DB-24PD : 24 channel isolated D/I board
- DB-24R, DB-24RD : 24 channels relay board
- DB-24PR, DB-24PRD : 24 channels power relay board
- DB-16P8R : 16 channels isolated D/I and 8 channels relay output board
- DB-24POR : 24 channels PhotoMos output board
- DB-24C : 24 channels open-collector output board
- ADP-37/PCI : extender, 50 pin OPTO-22 header to DB-37 for PCI bus I/O boards
- ADP-50/PCI : extender, 50 pin OPTO-22 header to 50-pin header, for PCI bus I/O boards
- 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 PIO-D96 card
- one piece of company floppy diskette or CD
- one piece of release note

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

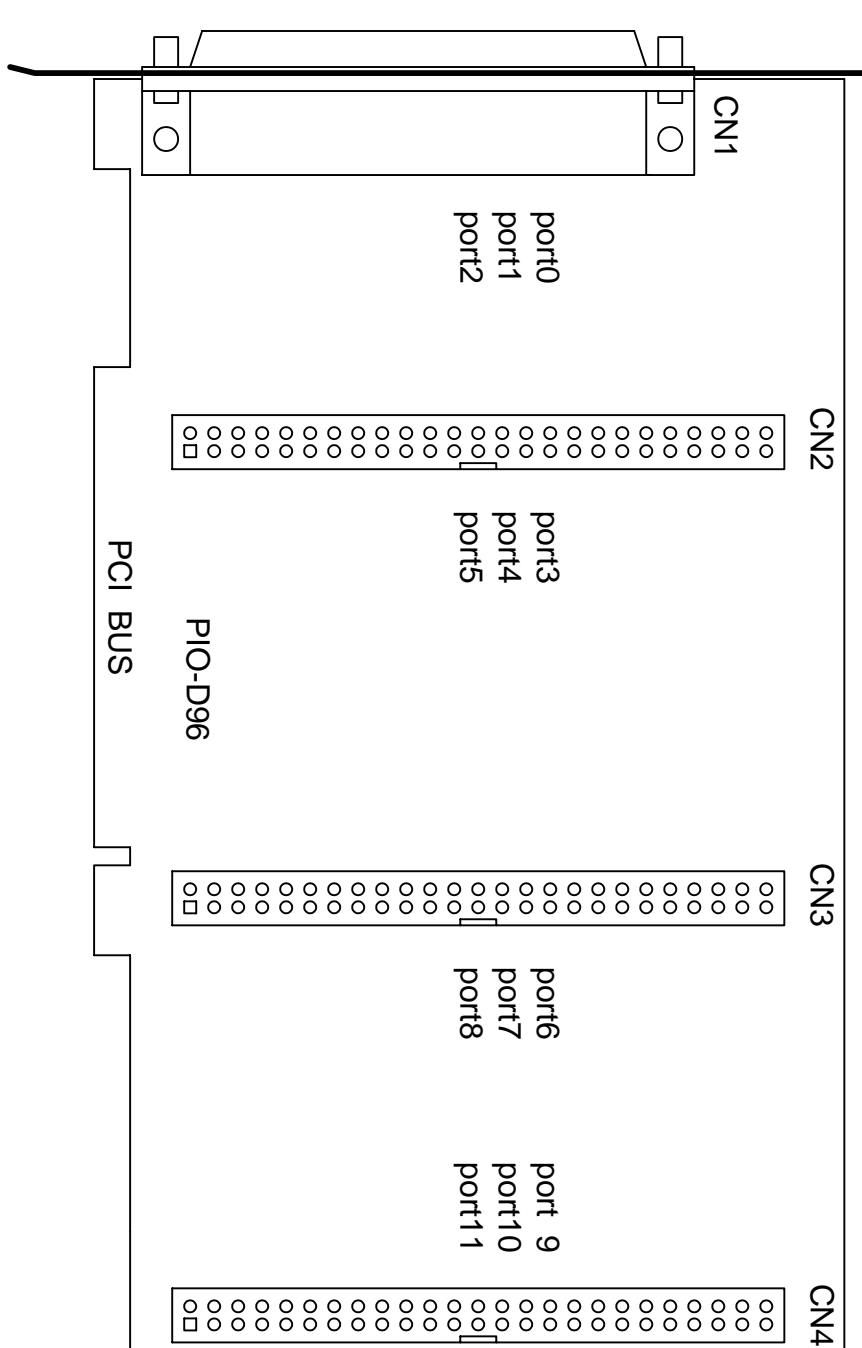
1. where you can find the software driver & utility
2. how to install software & utility
3. where is the diagnostic program
4. FAQ

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



2.2 I/O port Location

There are twelve 8-bit I/O ports in the PIO-D96. Every I/O port can be programmed as D/I or D/O port. When the PC is first power-up, all twelve ports are used as D/I port. The I/O port location is given as follows:

Connector of PIO-D96	PA0 ~ PA7	PB0 ~ PB7	PC0 ~ PC7
CN1	Port0	Port1	Port2
CN2	Port3	Port4	Port5
CN3	Port6	Port7	Port8
CN4	Port9	Port10	Port11

Refer to Sec. 2.1 for board layout & I/O port location.

Note: Each PC0 can be used as interrupt signal source. Refer to Sec. 2.5 for more information.

2.3 Enable I/O Operation

When the PC is power-up, all operation of D/I/O port is disable. The enable/disable of D/I/O is controlled by the RESET\ signal. Refer to Sec. 3.3.1 for more information about RESET\ signal. The power-up states are given as follows:

- All D/I/O operations are disable
- All twelve D/I/O ports are configured as D/I port
- All D/O latch register are undefined.(refer to Sec. 2.4)

The user has to perform some initialization before using these D/I/O. These recommended steps are given as follows:

Step 1: find address-mapping of PIO/PISO cards (refer to Sec. 3.1)

Step 2: enable all D/I/O operation (refer to Sec. 3.3.1)

Step 3: configure the first three ports to their expected D/I/O state & send the initial value to all D/O ports (refer to Sec. 3.3.7)

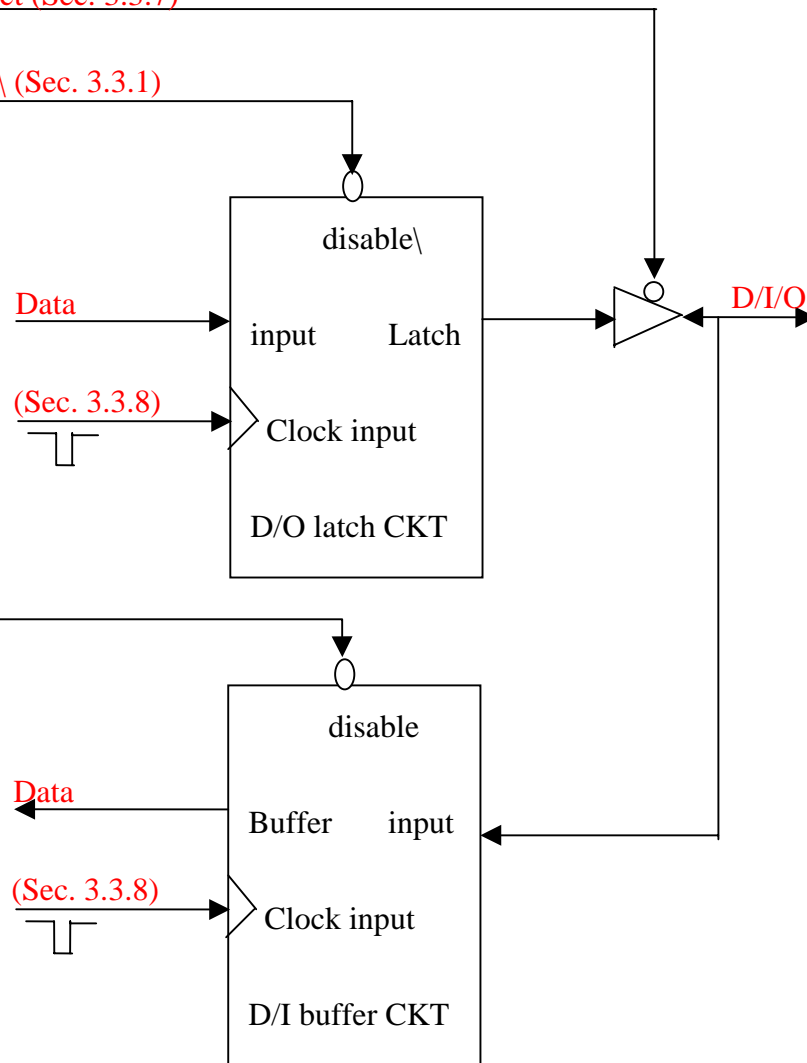
Step 4: configure the other three ports to their expected D/I/O state & send the initial value to all D/O ports(refer to Sec. 3.3.7)

Refer to DEMO1.C for demo program.

2.4 D/I/O Architecture

I/O select (Sec. 3.3.7)

RESET\ (Sec. 3.3.1)



- The RESET\ is in Low-state → all D/I/O operation is disable
- The RESET\ is in High-state → all D/I/O operation is enable.
- If D/I/O is configured as D/I port → D/I= external input signal
- If D/I/O is configured as D/O port → D/I = read back of D/O
- If D/I/O is configured as D/I port → send to D/O will change the D/O latch register only. The D/I & external input signal will not change.

2.5 Interrupt Operation

All P2C0, P5C0, P8C0 and P11C0 can be used as interrupt signal source. Refer to Sec. 2.1 & Sec. 2.7 for P2C0/P5C0/P8C0/P11C0 location. The interrupt of PIO-D96 is **level-trigger & Active_High**. The interrupt signal can be **inverted or non-inverted** programmable. The procedures of programming are given as follows:

1. make sure **the initial level is High or Low**
2. if the initial state is High → select the **inverted** signal (Sec. 3.3.6)
3. if the initial state is Low → select the **non-inverted** signal (Sec. 3.3.6)
4. enable the INT function (Sec. 3.3.4)
5. If the interrupt signal is active → program will transfer into the interrupt service routine → **if INT signal is High now → select the inverted input**
→ **if INT signal is Low now → select the non-inverted input**

Refer to DEMO3.C & DEMO4.C for single interrupt source. Refer to DEMO5.C for four interrupt sources.

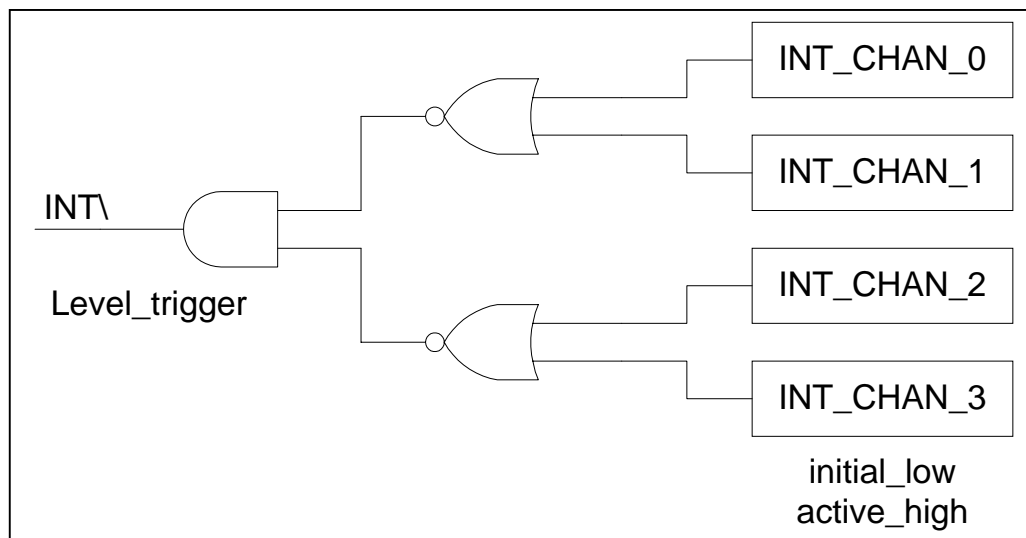
If only one interrupt signal source is used, the interrupt service routine does not have to identify the interrupt source. (refer to DEMO3.C & DEMO4.C)

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 the interrupt signal source
2. Compare the new status with the old status to identify the active signals
3. If P2C0 is active, service P2C0 & non-inverter/inverted the P2C0 signal
4. If P5C0 is active, service P5C0 & non-inverted/inverted the P5C0 signal
5. If P8C0 is active, service P8C0 & non-inverted/inverted the P8C0 signal
6. If P11C0 is active, service P11C0 & non-inverted/inverted the P11C0 signal
7. Save the new status to old status

Limitation: if the interrupt signal is too short, the new status may be as same as old status. So the interrupt signal must be hold-active 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.

2.5.1 Interrupt Block Diagram of PIO-D96



The interrupt output signal of PIO-D96, `INT\`, is `Level_trigger & Active_Low`. If the `INT\` generate a low-pulse, the PIO-D96 will interrupt the PC once a time. If the `INT\` is fixed in low level, the PIO-D96 will interrupt the PC continuously. **So that `INT_CHAN_0/1/2/3` 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` is the same. If all these four 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. 2.5 for more information.

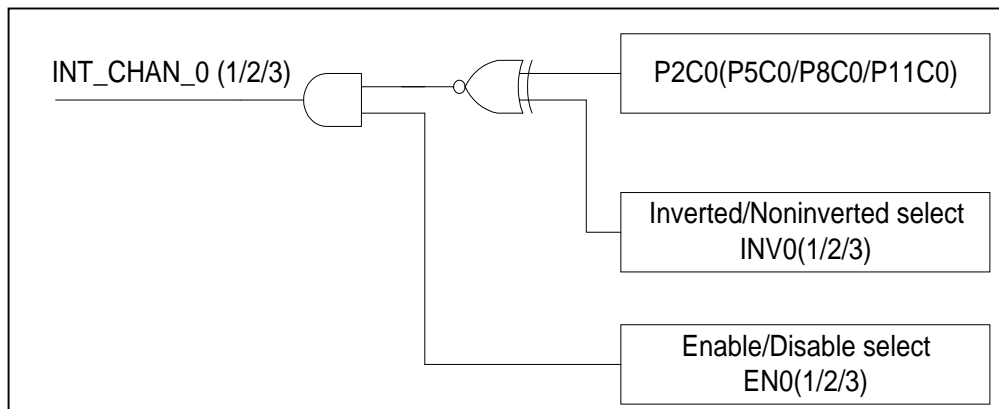
DEMO5.C → for multi-channel interrupt source

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 & DEMO4.C are designed for single-channel interrupt demo as follows:

DEMO3.C → for `INT_CHAN_0` only (P2C0 initial low)

DEMO4.C → for `INT_CHAN_0` only (P2C0 initial high)

2.5.2 INT_CHAN_0/1/2/3



The INT_CHAN_0(1/2/3) must be fixed in low level state normally and generated a high_pulse to interrupt the PC.

The EN0 (EN1/EN2/EN3) can be used to enable/disable the INT_CHAN_0(1/2/3) as follows: (Refer to Sec. 3.3.4)

EN0 (1/2/3) = 0 → INT_CHAN_0(1/2/3) = disable

EN0 (1/2/3) = 1 → INT_CHAN_0(1/2/3) = enable

The INV0 can be used to invert/non-invert the PC0 (1/2/3) as follows: (Refer to Sec.3.3.6)

INV0 (1/2/3) = 0 → INT_CHAN_0(1/2/3) = inverted state of P2C0
(P5C0/P8C0/P11C0)

INV0 (1/2/3) = 1 → INT_CHAN_0(1/2/3) = non-inverted state of P2C0
(P5C0/P8C0/P11C0)

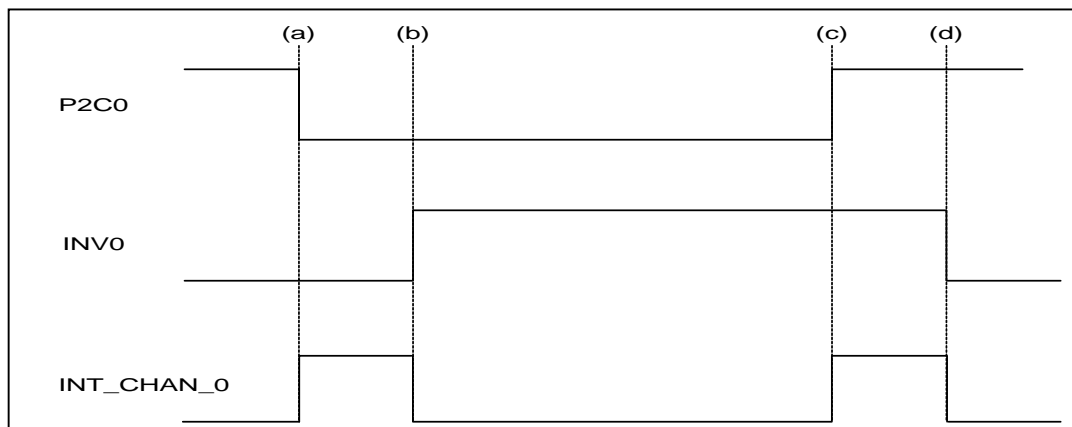
2.5.3 Initial_high, active_low Interrupt source

If the P2C0 is a initial_high, active_low signal, the interrupt service routine should use INV0 to inverted/ non-inverted the P2C0 for high_pulse generation as follows: (Refer to DEMO4.C)

Initial set:

```
now_int_state=1;      /* initial state for P2C0    */
outportb(wBase+0x2a,0); /* select the inverted P2C0 */
```

```
void interrupt irq_service()
{
if (now_int_state==1)      /* now P2C0 is changed to LOW          */(a)
{
    /* --> INT_CHAN_0=!P2C0=HIGH now    */
    COUNT_L++;             /* find a LOW_pulse (P2C0)          */
    If((inport(wBase+7)&1)==0) /* the P2C0 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=P2C0=LOW -->          */
        /* INT_CHAN_0 generate a high_pulse */
        now_int_state=0;     /* now P2C0=LOW                    */
    }
    else now_int_state=1;   /* now P2C0=HIGH                    */
    /* don't have to generate high_pulse */
}
else                        /* now P2C0 is changed to HIGH      */(c)
{
    /* --> INT_CHAN_0=P2C0=HIGH now    */
    COUNT_H++;             /* find a HIGH_pulse (P2C0)        */
    If((inport(wBase+7)&1)==1) /* the P2C0 is still fixed in HIGH */
    {
        /* need to generate a high_pulse */
        outportb(wBase+0x2a,0); /* INV0 select the inverted input  */(d)
        /* INT_CHAN_0=!P2C0=LOW -->          */
        /* INT_CHAN_0 generate a high_pulse */
        now_int_state=1;     /* now P2C0=HIGH                    */
    }
    else now_int_state=0;   /* now P2C0=LOW                    */
    /* don't have to generate high_pulse */
}
}
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



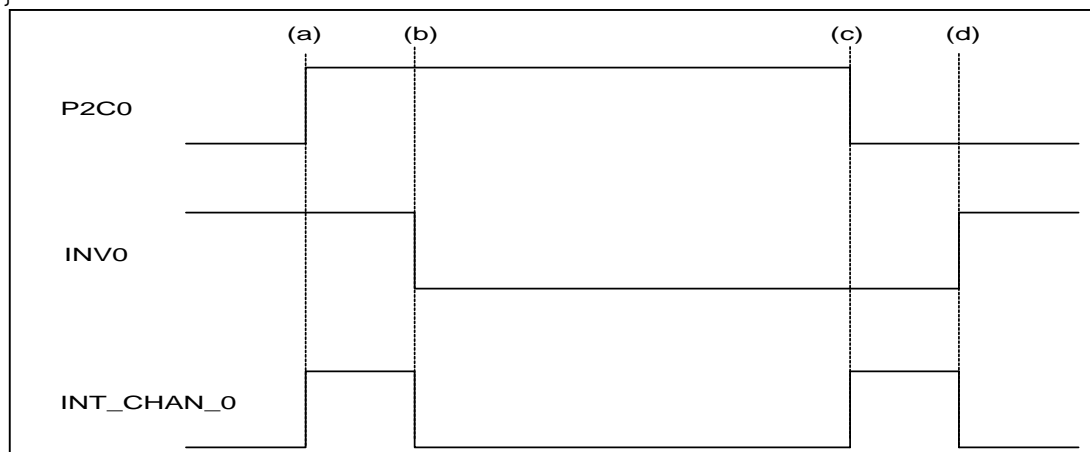
2.5.4 Initial_low, active_high Interrupt source

If the P2C0 is a initial_low, active_high signal, the interrupt service routine should use INV0 to inverted/non-inverted the P2C0 for high_pulse generation as follows: (Refer to DEMO3.C)

Initial set:

```
now_int_state=0;          /* initial state for P2C0          */
outportb(wBase+0x2a,1); /* select the non-inverted P2C0 */
```

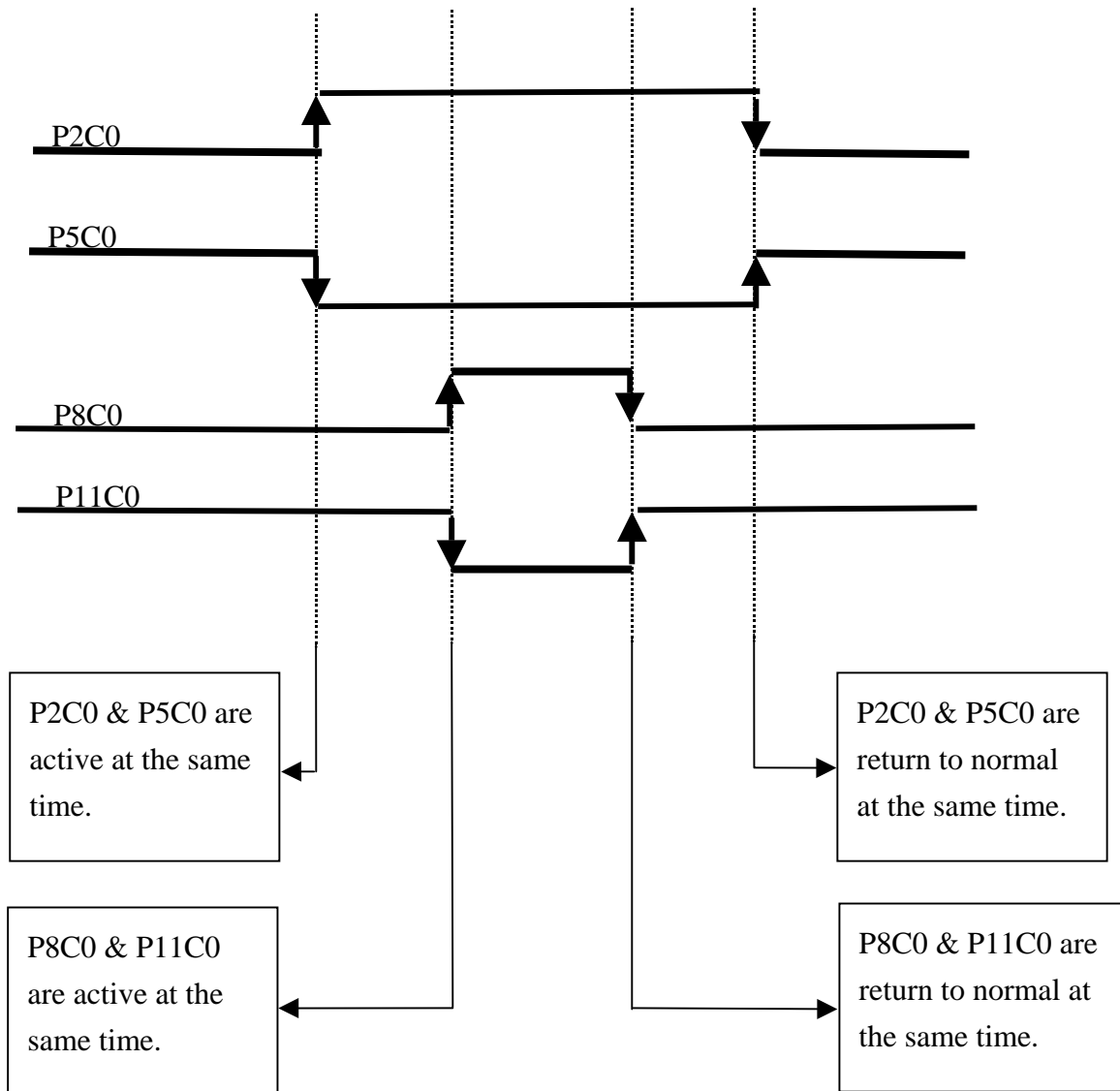
```
void interrupt irq_service()
{
if (now_int_state==1)      /* now P2C0 is changed to LOW          */(c)
{
/* --> INT_CHAN_0=!P2C0=HIGH now      */
COUNT_L++;              /* find a LOW_pulse (P2C0)           */
If((inport(wBase+7)&1)==0)/* the P2C0 is still fixed in LOW    */
{
/* → need to generate a high_pulse  */
outportb(wBase+0x2a,1);/* INV0 select the non-inverted input */ (d)
/* INT_CHAN_0=P2C0=LOW -->          */
/* INT_CHAN_0 generate a high_pulse */
now_int_state=0;        /* now P2C0=LOW                      */
}
else now_int_state=1;   /* now P2C0=HIGH                      */
/* don't have to generate high_pulse */
}
else
/* now P2C0 is changed to HIGH          */(a)
{
/* --> INT_CHAN_0=P2C0=HIGH now      */
COUNT_H++;              /* find a High_pulse (P2C0)         */
If((inport(wBase+7)&1)==1)/* the P2C0 is still fixed in HIGH   */
{
/* need to generate a high_pulse    */
outportb(wBase+0x2a,0);/* INV0 select the inverted input   */ (b)
/* INT_CHAN_0=!P2C0=LOW -->          */
/* INT_CHAN_0 generate a high_pulse */
now_int_state=1;        /* now P2C0=HIGH                      */
}
else now_int_state=0;   /* now P2C0=LOW                      */
/* don't have to generate high_pulse */
}
}
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



2.4.5 Multi Interrupt Source

Assume : P2C0 is initial Low, active High,
P5C0 is initial High, active Low
P8C0 is initial Low, active High
P11C0 is initial High, active Low

as follows:



Refer to DEMO5.C for source program. **All these four 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 maybe active at the same time. So the interrupt service routine has to service all active signals at the same time.

```

void interrupt irq_service()
{
new_int_state=inportb(wBase+7)&0x0f; /* 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 P2C0 change to high */
{
CNT_H1++;
}
else /* now P2C0 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 P5C0 change to high */
{
CNT_H2++;
}
else /* now P5C0 change to low */
{
CNT_L2++;
}
invert=invert^2; /* to generate a high pulse */
}

if ((int_c&0x4)!=0)
{
if ((new_int_state&0x04)!=0) /* now P8C0 change to high */
{
CNT_H3++;
}
else /* now P8C0 change to low */
{
CNT_L3++;
}
invert=invert^4; /* to generate a high pulse */
}

if ((int_c&0x8)!=0)
{
if ((new_int_state&0x08)!=0) /* now P11C0 change to high */
{
CNT_H4++;
}
else /* now P11C0 change to low */
{
CNT_L4++;
}
invert=invert^8; /* 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);
}

```

2.6 Daughter Boards

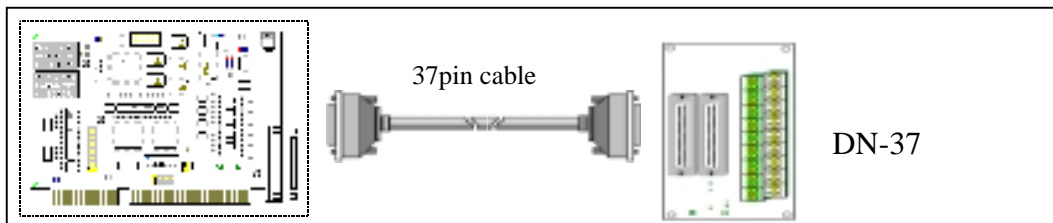
2.6.1 DB-37

The DB-37 is a general purpose daughter board for D-sub 37 pins. It is designed for easy wire connection.



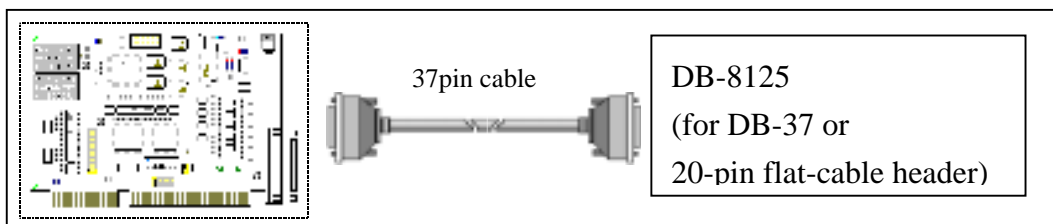
2.6.2 DN-37 & DN-50

The DN-37 is a general purpose daughter board for DB-37 with DIN-Rail Mounting. The DN-50 is designed for 50-pin flat-cable header. They are designed for easy wire connection. Both have Din-Rail mounting.



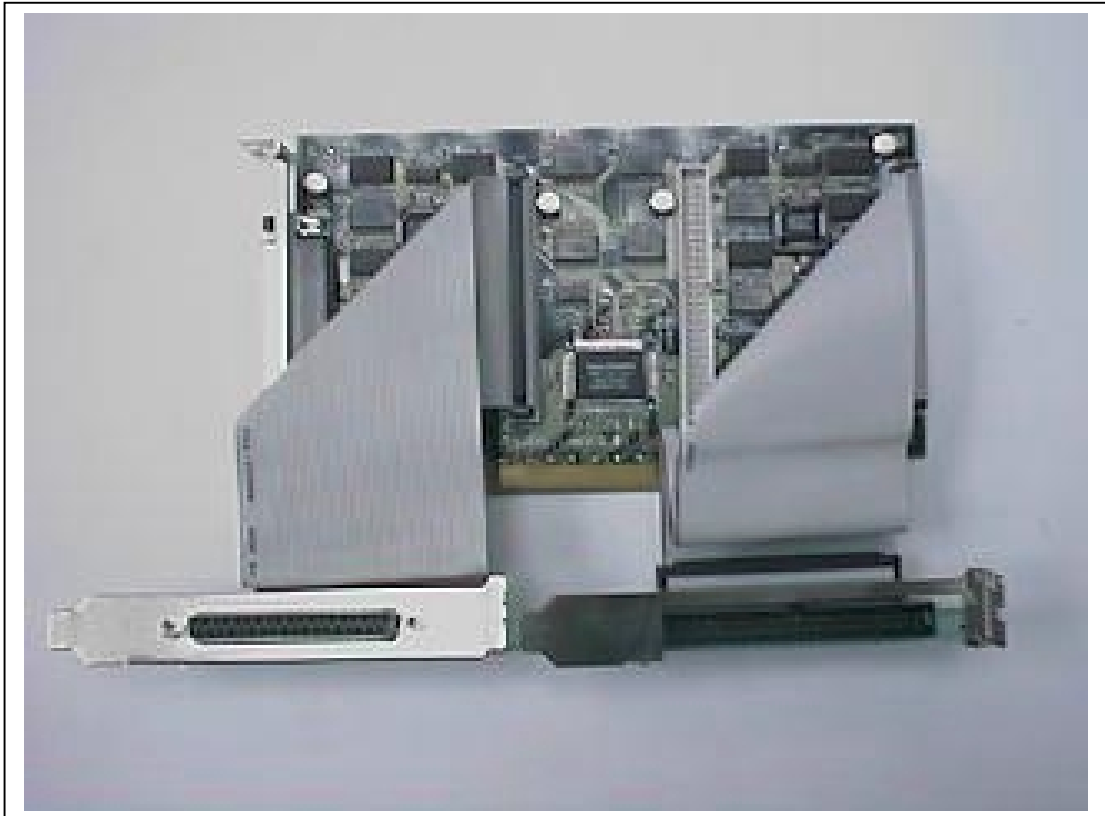
2.6.3 DB-8125

The DB-8125 is a general purpose screw terminal board. It is designed for easy wire connection. There are one DB-37 & two 20-pin flat-cable headers in the DB-8125.



2.6.4 ADP-37/PCI & ADP-50/PCI

The ADP-37/PCI & ADP-50/PCI are extender for 50-pin header. One side of ADP-37/PCI & ADP-50/PCI can be connected to a 50-pin header. The other side can be mounted on the PC chassis as follows:



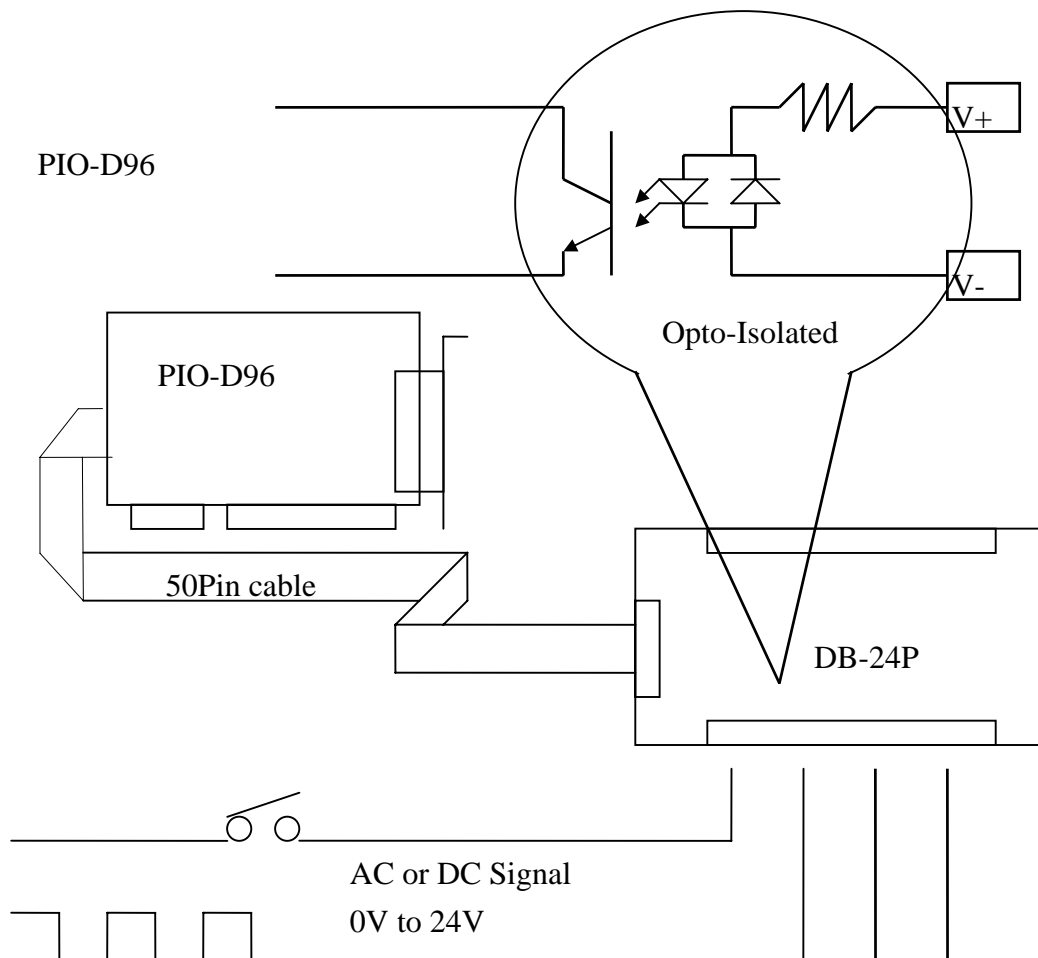
ADP-37/PCI: 50-pin header to DB-37 extender.

ADP-50/PCI: 50-pin header to 50-pin header extender.

NOTE: The user can choose the suitable extender for his own application

2.6.5 DB-24P, DB-24PD Isolated Input Board

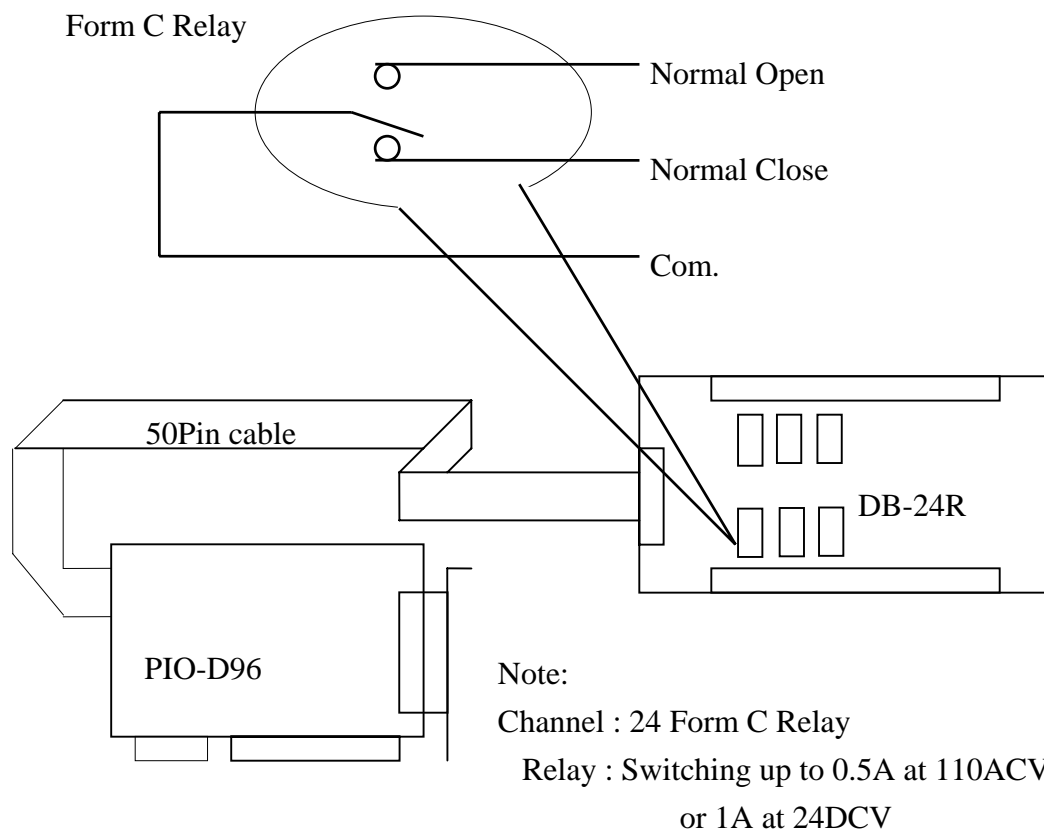
The DB-24P is a 24-channel isolated digital input daughter board. The optically isolated inputs of the DB-24P consists of a bi-directional optocoupler with a resistor for current sensing. You can use the DB-24P to sense DC signal from TTL levels up to 24V or use the DB-24P to sense a wide range of AC 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.



	DB-24P	DB-24PD
50-pin flat-cable header	Yes	Yes
D-sub 37-pin header	No	Yes
Other specifications	Same	

2.6.6 DB-24R, DB-24RD Relay Board

The DB-24R, 24-channel relay output board, consists of 24 form C relays for efficient switch of load by programmed control. The relay are energized by apply 12V/24V voltage signal to the appropriated relay channel on the 50-pin flat connector. There are 24 enunciator LEDs for each relay, light when their associated relay is activated.



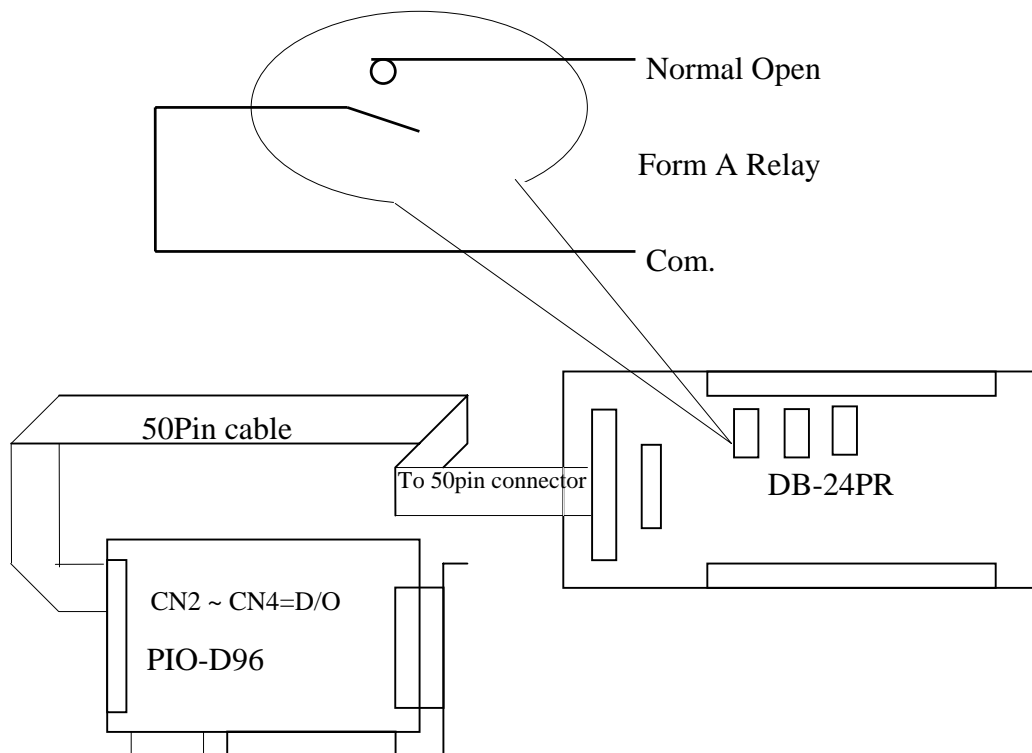
	DB-24R	DB-24RD
50-pin flat-cable header	Yes	Yes
D-sub 37-pin header	No	Yes
Other specifications	Same	

DB-24R, DB-24RD	24*Relay (120V, 0.5A)
DB-24PR,DB-24PRD	24* Power Relay (250V, 5A)
DB-24POR	24*PhotoMOS Relay (350V, 01.A)
DB-24SSR	24*SSR (250VAC, 4A)
DB-24C	24*O.C. (30V, 100 mA)
DB-16P8R	16*Relay (120V, 0.5A) + 8*isolated input

2.6.7 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, 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). Twenty-four 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,
PCI-D144, PIO-D144, PIO-D96, PIO-D56, PIO-D48, PIO-D24

20-Pin connector for 16 channel digital output, A-82X, A-62X, DIO-64, ISO-
DA16/DA8

Channel : 16 Form A Relay , 8 Form C Relay

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

2.6.8 Daughter Boards Comparison Table

	20-pin flat-cable	50-pin flat-cable	D-sub 37-pin
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: There is no 20 pin flat-cable header in PIO-D96. The PIO-D96 has one DB-37 connector and three 50 pin flat-cable headers.

2.7 Pin Assignment

CN1: 37 pin of D-type female connector.(for Port0, Port1, Port2)

Pin Number	Description	Pin Number	Description
1	N. C.	20	VCC
2	N. C.	21	GND
3	P1B7	22	P2C7
4	P1B6	23	P2C6
5	P1B5	24	P2C5
6	P1B4	25	P2C4
7	P1B3	26	P2C3
8	P1B2	27	P2C2
9	P1B1	28	P2C1
10	P1B0	29	P2C0
11	GND	30	P0A7
12	N.C.	31	P0A6
13	GND	32	P0A5
14	N.C.	33	P0A4
15	GND	34	P0A3
16	N.C.	35	P0A2
17	GND	36	P0A1
18	VCC	37	P0A0
19	GND	XXXXXXX	This pin not available

All signals are TTL compatible.

CN2/CN3/CN4: 50-pin of flat-cable connector (for Port3 ~ Port11)

Pin Number	Description	Pin Number	Description
1	P5C7/P8C7/P11C7	2	GND
3	P5C6/P8C6/P11C6	4	GND
5	P5C5/P8C5/P11C5	6	GND
7	P5C4/P8C4/P11C4	8	GND
9	P5C3/P8C3/P11C3	10	GND
11	P5C2/P8C2/P11C2	12	GND
13	P5C1/P8C1/P11C1	14	GND
15	P5C0/P8C0/P11C0	16	GND
17	P4B7/P7B7/P10B7	18	GND
19	P4B6/P7B6/P10B6	20	GND
21	P4B5/P7B5/P10B5	22	GND
23	P4B4/P7B4/P10B4	24	GND
25	P4B3/P7B3/P10B3	26	GND
27	P4B2/P7B2/P10B2	28	GND
29	P4B1/P7B1/P10B1	30	GND
31	P4B0/P7B0/P10B0	32	GND
33	P3A7/P6A7/P9A7	34	GND
35	P3A6/P6A6/P9A6	36	GND
37	P3A5/P6A5/P9A5	38	GND
39	P3A4/P6A4/P9A4	40	GND
41	P3A3/P6A3/P9A3	42	GND
43	P3A2/P6A2/P9A2	44	GND
45	P3A1/P6A1/P9A1	46	GND
47	P3A0/P6A0/P9A0	48	GND
49	VCC	50	GND

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 **PIO-D96** are given as follows:

- **Sub-vendor ID = 80**
- **Sub-device ID = 01**
- **Sub-aux ID = 10**

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 PIO-D96 in this PC

```
wSubVendor=0x80; wSubDevice=1; wSubAux=0x10; /* for PIO_D96 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor, wSubDevice, wSubAux);
printf("Threr are %d PIO-D96 Cards in this PC\n", wBoards);

/* step2: save resource of all PIO-D96 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 and 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 PIO-D96 cards first */
wSubVendor=0x80; wSubDevice=1; wSubAux=0x10; /* for PIO_D96 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("Threr are %d PIO-D96 Cards in this PC\n",wBoards);

/* step2: save resource of all PIO-D96 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 PIO-D96 directly */
wBase=wConfigSpace[0][0];/* get base address the card_0 */
output(wBase,1); /* enable all D/I/O operation of card_0 */

wBase=wConfigSpace[1][0];/* get base address the card_1 */
output(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 PIO-D96 from this PC
2. Install one PIO-D96 into the PC's PCI_slot1, run PIO_PISO.EXE & record the wSlotBus1 & wSlotDevice1
3. Remove all PIO-D96 from this PC
4. Install one PIO-D96 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:

Step 1: 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 PIO-D96 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	read Port0	write Port0
Wbase+0xc4	read Port1	write Port1
Wbase+0xc8	read Port2	write Port2
Wbase+0xcc	×	Port0~Port2 configuration
Wbase+0xd0	read Port3	write Port3
Wbase+0xd4	read Port4	write Port4
Wbase+0xd8	read Port5	write Port5
Wbase+0xdc	×	Port3~Port5 configuration
Wbase+0xe0	read Port6	write Port6
Wbase+0xe4	read Port7	write Port7
Wbase+0xe8	read Port8	write Port8
Wbase+0xec	×	Port6~Port8 configuration
Wbase+0xf0	read Port9	write Port9
Wbase+0xf4	read Port10	write Port10
Wbase+0xf8	read Port11	write Port11
Wbase+0xfc	×	Port9~Port11 configuration

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-on, 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-up, 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	EN3	EN2	EN1	EN0

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

EN0=0 → disable P2C0 as a interrupt signal (default)

EN0=1 → enable P2C0 as a interrupt signal

```
outportb(wBase+5,0);      /* disable interrupt          */
outportb(wBase+5,1);      /* enable interrupt P2C0       */
outportb(wBase+5,0x0f);   /* enable interrupt P2C0, P5C0,P8C0,P11C0 */
```

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=P2C0, Aux1=P5C0, Aux2=P8C0, Aux3=P11C0, Aux7~4=Aux-ID. Refer to DEMO5.C for more information. The Aux 0~3 are used as interrupt source. 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	INV3	INV2	INV1	INV0

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

INV0=1 → select the non-inverted signal from P2C0

INV0=0 → select the inverted signal from P2C0

outportb(wBase+0x2a,0x0f); /* select the non-inverted input P2/5/8/11C0 */

outportb(wBase+0x2a,0x00); /* select the inverted input of P2/5/8/11C0 */

outportb(wBase+0x2a,0x0e); /* select the inverted input of P2C0 */

/* select the non-inverted input P5/8/11C0 */

outportb(wBase+0x2a,0x0c); /* select the inverted input of P2/5C0 */

/* select the non-inverted input P8/11C0 */

Refer to Sec. 2.5 for more information.

Refer to DEMO5.C for more information.

3.3.7 I/O Selection Control Register

(Write): wBase+0xcc

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Port2	Port1	Port0

(Write): wBase+0xdc

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Port5	Port4	Port3

(Write): wBase+0xec

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Port8	Port7	Port6

(Write): wBase+0xfc

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Port11	Port10	Port9

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

Before using these D/I/O ports, user has to configure the state of ports as desired.

port?=1 → this port is used as a D/O port

port?=0 → this port is used as a D/I port

```
outportb(wBase+0xcc,0x03);      /* set port0 as D/O ports */
                                /* set port1 as D/O ports */
                                /* set port2 as D/I ports */

outportb(wBase+0xdc,0x07);      /* set port3 ~ port5 as D/O ports */
outportb(wBase+0xec,0x00);      /* set port6 ~ port8 as D/I ports */
```

3.3.8 Read/Write 8-bit data Register

(Read/Write):wBase+0xc0/0xc4/0xc8/0xd0/0xd4/0xd8/
0xe0/0xe4/0xe8/0xf0/0xf4/0xf8/

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
D7	D6	D5	D4	D3	D2	D1	D0

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

There are twelve 8-bit I/O port in the PIO-D96. Every I/O port can be programmed as D/I or D/O port. Refer to Sec. 3.3.8 for D/I or D/O selection. When the PC is power-up, all twelve ports are used as D/I port.

```

outputb(wBase+0xc0,Val);      /* write to Port0          */
Val=inportb(wBase+0xc0);     /* read from Port0         */

outputb(wBase+0xcc,0x07);    /* set port0~port2 as D/O ports */
outputb(wBase+0xc0,i1);      /* write to Port0          */
outputb(wBase+0xc4,i2);      /* write to Port1          */
outputb(wBase+0xc8,i3);      /* write to Port2          */

outputb(wBase+0xec,0x04);    /* set Port6 & Port7 as D/I ports */
/* set Port8 as D/O port      */
j1=inportb(wBase+0xe0);      /* read Port6              */
j2=inportb(wBase+0xe4);      /* read Port7              */
outputb(wBase+0xe8,j3);      /* write to Port8          */

```

NOTE : Make sure the I/O port configuration (DI or DO) before read/write the data register. (refer to sec. 3.3.7)

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 program 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\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 validate 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. 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

User can find this utility in the company CD or floppy disk. It is useful for all PIO/PISO series card.

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



4.2 DEMO1

```
/* demo 1 : D/O demo of CN1 */
/* step 1 : connect a DB-24C to CN1 of PIO-D96 */
/* step 2 : run DEMO1.EXE */
/* step 3 : check the LEDs of DB-24C will turn on sequentially */
/* ----- */
#include "PIO.H"
WORD wBase,wIrq;
main()
{
int i1,i2,i3;
long i=1;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();

/* step1 : find address-mapping of PIO/PISO cards */
wRetVal=PIO_DriverInit(&wBoards,0x80,0x01,0x10); /* for PIO-D96 */
printf("\n(1) Threr are %d PIO-D96 Cards in this PC",wBoards);
if ( wBoards==0 ) exit(0);

printf("\n\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);
/* select card_0 */
/* step2 : enable all D/I/O port */
outportb(wBase,1); /* /RESET -> 1 */
/* step3 : configure I/O direction */
outportb(wBase+0xcc,0x07); /* set CN1 as D/O ports */

for (;;)
{
i1=i&0xff;
i2=(i>>8)&0xff;
i3=(i>>16)&0xff;
outportb(wBase+0xc0,i1);
outportb(wBase+0xc4,i2);
outportb(wBase+0xc8,i3);
delay(10000);
i=i<<1;
i=i&0xffffffff;
if (i==0) i=1;
if (kbhit()!=0) return;
}
PIO_DriverClose();
}
```

4.3 DEMO2

```
/* demo 2 : DI/O demo of CN2 - CN3 */
/* step 1 : connect CN2 t0 CN3 of PIO-D96 */
```

```

/* step 2 : run DEMO2.EXE */
/* step 3 : check the information on screen D/I will same as D/O */
/* ----- */
#include "PIO.H"
WORD wBase,wIrq;

main()
{
int i1,i2,i3,j1,j2,j3;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;
long i=1;

clrscr();

/* step1 : find address-mapping of PIO/PISO cards */
.
.
/* step2 : enable all D/I/O port */
outputb(wBase,1); /* /RESET -> 1 */
/* step3 : configure I/O direction */
outputb(wBase+0xdc,0x07); /* set CN2 as D/O ports */
outputb(wBase+0xec,0x00); /* set CN3 as D/I ports */

for (;;)
{
gotoxy(1,6);
i1=i&0xff;
i2=(i>>8)&0xff;
i3=(i>>16)&0xff;
outputb(wBase+0xd0,i1);
outputb(wBase+0xd4,i2);
outputb(wBase+0xd8,i3);
j1=inportb(wBase+0xe0);
j2=inportb(wBase+0xe4);
j3=inportb(wBase+0xe8);

printf("\nD/O = [%2x,%2x,%2x] , D/I = [%2x,%2x,%2x]
",i1,i2,i3,j1,j2,j3);

if ((j1!=i1)|| (j2!=i2)|| (j3!=i3))
{
printf("\n\nError .....");
}
else printf("\n\nO.K. ....");

i=i<<1;
i=i&0xffffffff;
if (i==0) i=1;

if (kbhit()!=0) return;
}
PIO_DriverClose();
}

```

4.4 DEMO3

```

/* demo 3 : Count high pulse of P2C0 */
/* (initial Low & active High) */
/* step 1 : run DEMO3.EXE */
/* ----- */

#include "PIO.H"

```

```

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init_low();
WORD wBase,wIrq;

static void interrupt irq_service();
int COUNT_L,COUNT_H,irqmask,now_int_state;

int main()
{
int i,j;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();

/* step1 : find address-mapping of PIO/PISO cards */
:
:
/* select card_0 */
/* step2 : enable all D/I/O port */
outportb(wBase,1); /* /RESET -> 1 */

/* step3 : configure I/O direction */
outportb(wBase+0xcc,0x00); /* set CN1 as D/I ports */

init_low();
printf("\n\n***** show the count of High_pulse *****\n");

for (;;)
{
gotoxy(1,8);
printf("\nCOUNT_H=%d",COUNT_H);
if (kbhit()!=0) break;
}
outportb(wBase+5,0); /* disable all interrupt */

PIO_DriverClose();
}

/* Use P2C0 as external interrupt signal */
WORD init_low()
{
disable();
outportb(wBase+5,0); /* disable all interrupt */
if (wIrq<8)
{
irqmask=inportb(A1_8259+1);
outportb(A1_8259+1,irqmask & 0xff ^ (1<<wIrq));
setvect(wIrq+8,irq_service);
}
else
{
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+5,1);          /* enable interrupt (P2C0) */
now_int_state=0;             /* now ini_signal is low */
outportb(wBase+0x2a,1);      /* select the non-inverte */
enable();
}

void interrupt irq_service()
{
if (now_int_state==1)        /* now P2C0 change to low */
{                             /* INT_CHAN_0 = !P2C0 */
COUNT_L++;                 /* find a low pulse (P2C0) */
if ((inportb(wBase+7)&1)==0) /* P2C0 still fixed in low */
{                             /* need to generate a high pulse */
outportb(wBase+0x2a,1);      /* INVO select noninverted input */
now_int_state=0;            /* now P2C0=low */
}
else now_int_state=1;        /* now P2C0=High */
}
else                          /* now P2C0 change to high */
{                             /* INT_CHAN_0 = P2C0 */
COUNT_H++;                 /* find a high pulse (P2C0) */
if ((inportb(wBase+7)&1)==1) /* P2C0 still fixed in high */
{                             /* need to generate a high pulse */
outportb(wBase+0x2a,0);      /* INVO select inverted input */
now_int_state=1;            /* now P2C0=high */
}
else now_int_state=0;        /* now P2C0=low */
}
}

if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}

```

4.5 DEMO4

```
/* demo 4 : Count high pulse of P2C0 */
/*          (initial High & active Low) */
/* step 1 : run DEMO4.EXE */
/* ----- */

#include "PIO.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init_high();
WORD wBase,wIrq;

static void interrupt irq_service();
int COUNT_L,COUNT_H,irqmask,now_int_state;

int main()
{
  int i,j;
  WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
  WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
  char c;

  clrscr();

  /* step1 : find address-mapping of PIO/PISO cards */
  :
  :
  /* select card_0 */
  /* step2 : enable all D/I/O port */
  outportb(wBase,1);
  /* /RESET -> 1 */

  /* step3 : configure I/O direction */
  outportb(wBase+0xcc,0x00);
  /* set CN1 as D/I ports */

  init_high();
  printf("\n\n***** show the count of Low_pulse *****\n");

  for (;;)
  {
    gotoxy(1,7);
    printf("\nCOUNT_L=%d",COUNT_L);
    if (kbhit()!=0) break;
  }
  outportb(wBase+5,0);
  /* disable all interrupt */

  PIO_DriverClose();
}

/* Use P2C0 as external interrupt signal */
WORD init_high()
{
  disable();
  outportb(wBase+5,0);
  /* disable all interrupt */
  if (wIrq<8)
  {
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & 0xff ^ (1<<wIrq));
    setvect(wIrq+8,irq_service);
  }
}
```

```

else
{
    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+5,1);          /* enable interrupt (P2C0) */
now_int_state=1;             /* now ini_signal is high */
outportb(wBase+0x2a,0);      /* select the inverte */
enable();
}

void interrupt irq_service()
{
    if (now_int_state==1)     /* now P2C0 change to low */
    {                          /* INT_CHAN_0 = !P2C0 */
        COUNT_L++;           /* find a low pulse (P2C0) */
        if ((inportb(wBase+7)&1)==0) /* P2C0 still fixed in low */
        {                    /* need to generate a high pulse */
            outportb(wBase+0x2a,1); /* INVO select noninverted input */
            now_int_state=0;      /* now P2C0=low */
        }
        else now_int_state=1;    /* now P2C0=High */
    }
    else                       /* now P2C0 change to high */
    {                          /* INT_CHAN_0 = P2C0 */
        COUNT_H++;           /* find a high pulse (P2C0) */
        if ((inportb(wBase+7)&1)==1) /* P2C0 still fixed in high */
        {                    /* need to generate a high pulse */
            outportb(wBase+0x2a,0); /* INVO select inverted input */
            now_int_state=1;      /* now P2C0=high */
        }
        else now_int_state=0;    /* now P2C0=low */
    }
    if (wIrq>=8) outportb(A2_8259,0x20);
    outportb(A1_8259,0x20);
}

```

4.6 DEMO5

```
/* demo 5 : Four interrupt source */
/*      P2C0 : initial Low , active High */
/*      P5C0 : initial High , active Low */
/*      P8C0 : initial Low , active High */
/*      P11C0 : initial High , active Low */
/* step 1 : run DEMO5.EXE */
/* ----- */

#include "PIO.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init();
WORD wBase,wIrq;

static void interrupt irq_service();
int irqmask,now_int_state,new_int_state,invert,int_c,int_num;
int CNT_L1,CNT_L2,CNT_L3,CNT_L4;
int CNT_H1,CNT_H2,CNT_H3,CNT_H4;

int main()
{
  int i,j;
  WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
  WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
  char c;

  clrscr();

  /* step1 : find address-mapping of PIO/PISO cards */
  :
  /* select card_0 */
  /* step2 : enable all D/I/O port */
  outportb(wBase,1); /* /RESET -> 1 */

  /* step3 : configure I/O direction */
  outportb(wBase+0xcc,0x00); /* set CN1 as D/I ports */
  outportb(wBase+0xdc,0x00); /* set CN2 as D/I ports */
  outportb(wBase+0xec,0x00); /* set CN3 as D/I ports */
  outportb(wBase+0xfc,0x00); /* set CN4 as D/I ports */

  init();
  printf("\n***** show the count of pulse *****\n");

  for (;;)
  {
    gotoxy(1,7);
    printf("\n(CNT_L,CNT_H)=(%d,%d) (%d,%d) (%d,%d) (%d,%d)
%x",CNT_L1,CNT_H1,CNT_L2,CNT_H2,CNT_L3,CNT_H3,CNT_L4,CNT_H4,int_num);
    if (kbhit()!=0) break;
  }

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

/* Use P2C0, P5C0, P8C0 & P11C0 as external interrupt signal */
WORD init()
```

```

{
disable();
outportb(wBase+5,0);          /* disable all interrupt */
if (wIrq<8)
{
irqmask=inportb(A1_8259+1);
outportb(A1_8259+1,irqmask & 0xff ^ (1<<wIrq));
setvect(wIrq+8,irq_service);
}
else
{
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=0x05;
outportb(wBase+0x2a,invert);          /* P2C0 = non-inverte input */
/* P5C0 = inverte input */
/* P8C0 = non-inverte input */
/* P11C0 = inverte input */

now_int_state=0x0a;          /* P2C0 = Low */
/* P5C0 = High */
/* P8C0 = Low */
/* P11C0 = High */

CNT_L1=CNT_L2=CNT_L3=CNT_L4=0;          /* Low_pulse counter */
CNT_H1=CNT_H2=CNT_H3=CNT_H4=0;          /* High_pulse counter */
int_num=0;
outportb(wBase+5,0x0f);          /* enable interrupt P2C0 */
enable();          /* P5C0, P8C0, P11C0 */
}

/* ----- */
/* NOTE:1.The hold-time of INT_CHAN_0/1/2/3 must long enough */
/* 2.The ISR must read the interrupt status again to the */
/* active interrupt sources. */
/* 3.The INT_CHAN_0&INT_CHAN_1 can be active at the same time*/
/* ----- */
void interrupt irq_service()
{
int_num++;
new_int_state=inportb(wBase+7)&0x0f;
int_c=new_int_state^now_int_state;

if ((int_c&0x1)!=0)
{
if ((new_int_state&0x01)!=0) /* now P2C0 change to high */
{
CNT_H1++;
}
else /* now P2C0 change to low */
{
CNT_L1++;
}
invert=invert^1;          /* generate a high pulse */
}

if ((int_c&0x2)!=0)
{
if ((new_int_state&0x02)!=0) /* now P5C0 change to high */

```

```

        {
            CNT_H2++;
        }
    else /* now P5C0 change to low */
        {
            CNT_L2++;
        }
    invert=invert^2; /* generate a high pulse */
}

if ((int_c&0x4)!=0)
{
    if ((new_int_state&0x04)!=0) /* now P8C0 change to high */
        {
            CNT_H3++;
        }
    else /* now P8C0 change to low */
        {
            CNT_L3++;
        }
    invert=invert^4; /* generate a high pulse */
}

if ((int_c&0x8)!=0)
{
    if ((new_int_state&0x08)!=0) /* now P11C0 change to high */
        {
            CNT_H4++;
        }
    else /* now P11C0 change to low */
        {
            CNT_L4++;
        }
    invert=invert^8; /* 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);
}

```