

User's Guide

omega.com[®]

Ω OMEGA[®]

Shop online at

www.omega.com

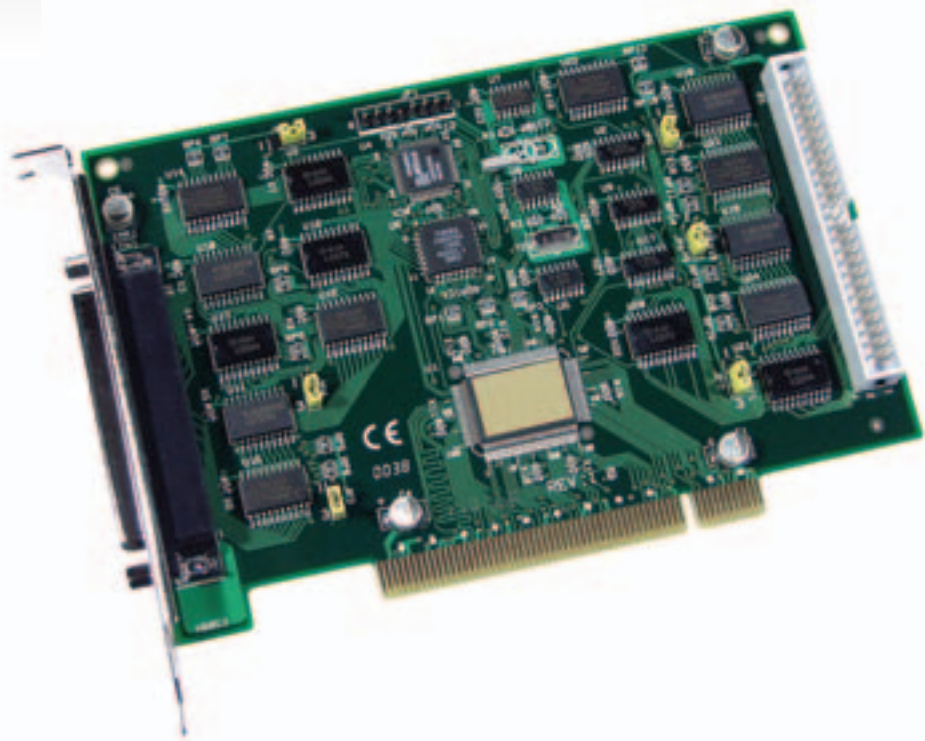
e-mail: info@omega.com

ISO 9001
CERTIFIED
CORPORATE QUALITY

STAMFORD, CT

ISO 9002
CERTIFIED
CORPORATE QUALITY

MANCHESTER, UK



OME-PIO-D48 **PCI-Bus Digital I/O Board** **Hardware Manual**



OMEGAnet® Online Service
www.omega.com

Internet e-mail
info@omega.com

Servicing North America:

USA:
ISO 9001 Certified

One Omega Drive, P.O. Box 4047
Stamford CT 06907-0047
TEL: (203) 359-1660 FAX: (203) 359-7700
e-mail: info@omega.com

Canada:

976 Bergar
Laval (Quebec) H7L 5A1, Canada
TEL: (514) 856-6928 FAX: (514) 856-6886
e-mail: info@omega.ca

For immediate technical or application assistance:

USA and Canada: Sales Service: 1-800-826-6342 / 1-800-TC-OMEGA®
Customer Service: 1-800-622-2378 / 1-800-622-BEST®
Engineering Service: 1-800-872-9436 / 1-800-USA-WHEN®
TELEX: 996404 EASYLINK: 62968934 CABLE: OMEGA

Mexico:

En Español: (001) 203-359-7803 e-mail: espanol@omega.com
FAX: (001) 203-359-7807 info@omega.com.mx

Servicing Europe:

Benelux:

Postbus 8034, 1180 LA Amstelveen, The Netherlands
TEL: +31 (0)20 3472121 FAX: +31 (0)20 6434643
Toll Free in Benelux: 0800 0993344
e-mail: sales@omegaeng.nl

Czech Republic:

Frystatska 184, 733 01 Karviná, Czech Republic
TEL: +420 (0)59 6311899 FAX: +420 (0)59 6311114
Toll Free: 0800-1-66342 e-mail: info@omegashop.cz

France:

11, rue Jacques Cartier, 78280 Guyancourt, France
TEL: +33 (0)1 61 37 29 00 FAX: +33 (0)1 30 57 54 27
Toll Free in France: 0800 466 342
e-mail: sales@omega.fr

Germany/Austria:

Daimlerstrasse 26, D-75392 Deckenpfronn, Germany
TEL: +49 (0)7056 9398-0 FAX: +49 (0)7056 9398-29
Toll Free in Germany: 0800 639 7678
e-mail: info@omega.de

United Kingdom:

ISO 9002 Certified

One Omega Drive, River Bend Technology Centre
Northbank, Irlam, Manchester
M44 5BD United Kingdom
TEL: +44 (0)161 777 6611 FAX: +44 (0)161 777 6622
Toll Free in United Kingdom: 0800-488-488
e-mail: sales@omega.co.uk

It is the policy of OMEGA to comply with all worldwide safety and EMC/EMI regulations that apply. OMEGA is constantly pursuing certification of its products to the European New Approach Directives. OMEGA will add the CE mark to every appropriate device upon certification.

The information contained in this document is believed to be correct, but OMEGA Engineering, Inc. accepts no liability for any errors it contains, and reserves the right to alter specifications without notice.

WARNING: These products are not designed for use in, and should not be used for, patient-connected applications.

OME-PIO-D48

User Manual

Table of Contents

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 & DEFAULT SETTING.....	6
2.2	I/O PORT LOCATION.....	7
2.3	I/O PORT INITIALIZATION OPERATION	7
2.4	D/I/O ARCHITECTURE	9
2.5	INTERRUPT OPERATION.....	10
2.6	DAUGHTER BOARDS.....	17
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.....	38
4.1	PIO_PISO.....	39
4.2	DEMO1	41
4.3	DEMO2	43
4.4	DEMO3	44

1. Introduction

The OME-PIO-D48 provides 48 TTL digital I/O lines. The OME-PIO-D48 consists of two 24-bit bi-direction ports. Each 24-bit port supports three 8-bit groups A, B, C. 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. Outputs of the I/O buffers are pulled up through 10K resistors to +5VDC. Outputs can be changed to pull-down by jumper selection on the board. This pull-up/pull-down mechanism assures that there are no erroneous outputs at power-up until the board is initialized by application software.

The OME-PIO-D48 has one D-Sub connector and one 50-pin flat-cable header. The header can be connected to a 50-pin flat-cable. The flat-cable can be connected to OME-ADP-37/PCI or OME-ADP-50/PCI adapters. The adapter can be fixed on the chassis. It can be installed in a 5 V PCI bus and supports "Plug & Play".

1.1 Features

- PCI Bus
- Up to 48 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 OME-DB-24P, OME-DB-24R, OME-DB-24PR, OME-DB-24PD, OME-DB-24RD, OME-DB-24PRD, OME-DB-16P8R, OME-DB-24POR, OME-DB-24SSR or OME-DB-
- One 32-bit programmable internal timer
- One 16-bit event counter
- Interrupt source: 4 channels
- Pull-up or pull-down resistors on I/O lines
- Emulate two industrial-standard 8255 mode 0
- Buffer output for higher driving capability than 8255
- One D-Sub connector, one 50-pin flat cable connector
- 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: 64 mA (Max.)
Source current: 32 mA (Max.)
- Power consumption: +5V @ 900mA
- Environmental :
Operating Temp. : 0°C to 60°C
Storage Temp. : -20°C to 80 °C
Humidity: 0 to 90 % non-condensing
Dimension : 156mm x 105mm

1.3 Order Description

- OME-PIO-D48 : PCI bus 48-bit opto-22 DIO board

1.3.1 Options

- OME-DB-24P, DB-24PD: 24 channel isolated D/I board
- OME-DB-24R, DB-24RD: 24 channel relay board
- OME-DB-24PR, DB-24PRD: 24 channel power relay board
- OME-DB-16P8R: 16 channel isolated D/I and 8 channel relay output board
- OME-DB-24POR: 24 channel Photo Mos output board
- OME-DB-24SSR: 24 channel Solid State output board
- OME-DB-24C: 24-channel open-collector output board
- OME-ADP-37/PCI: extender, 50-pin header to OME-DB-37 for PCI Bus I/O boards
- OME-ADP-50/PCI: extender, 50-pin header to 50-pin header, for PCI Bus I/O boards

1.4 PCI Data Acquisition Family

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

1. OME-PCI-series: first generation, isolated or non-isolated cards

OME-PCI-1002/1202/1800/1802/1602: multi-function family, non-isolated

OME-PCI-P16R16/P16C16/P16POR16/P8R8: D/I/O family, isolated

OME-PCI-TMC12: timer/counter card, non-isolated

2. OME-PIO-series: cost-effective generation, non-isolated cards

OME-PIO-823/821: multi-function family

OME-PIO-D144/D96/D64/D56/D48/D24: D/I/O family

OME-PIO-DA16/DA8/DA4: D/A family

3. OME-PISO-series: cost-effective generation, isolated cards

OME-PISO-813: A/D card

OME-PISO-P32C32/P64/C64: D/I/O family

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

OME-PISO-730: D/I/O card

1.5 Product Check List

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

- one piece of OME-PIO-D48 card
- one piece of software floppy diskette or CD
- one piece of release note

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

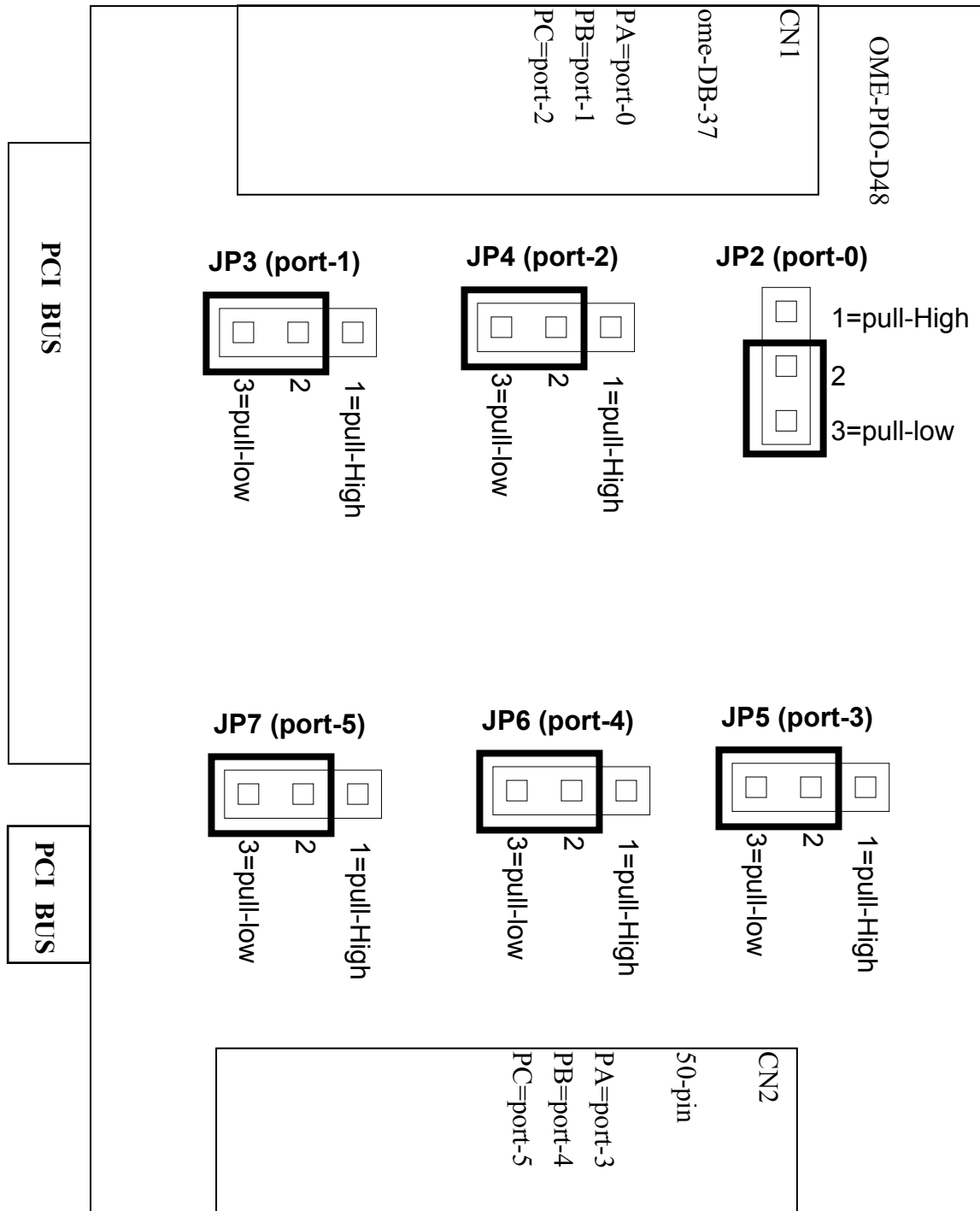
1. Where you can find the software driver & utility?
2. How to install software & utility?
3. Where is the diagnostic program?
4. FAQ

Attention !

If any of these items are missing or damaged, contact Omega Engineering immediately. Save the shipping materials and the box in case you want to ship or store the product in the future.

2. Hardware configuration

2.1 Board Layout & Default Setting



Default Setting: JP2/3/4/5/6/7=2-3 short=pull-low

2.2 I/O Port Location

There are six 8-bit I/O ports in the OME-PIO-D48. Every I/O port can be programmed as D/I or D/O port. When the PC is first power-up, all six ports are configured as D/I port. These D/I ports can be pull-high or pull-low selected by JP2 ~ JP7. These I/O port locations are given as following:

Connector of OME-PIO-D48	PA0 to PA7	PB0 to PB7	PC0 to PC7
CN1 (OME-DB-37)	port-0 (pull-high/low by JP2)	port-1 (pull-high/low by JP3)	port-2 (pull-high/low by JP4)
CN2 (50-pin head)	port-3 (pull-high/low by JP5)	port-4 (pull-high/low by JP6)	port-5 (pull-high/low by JP7)

- Note 1: Refer to Sec. 2.1 for board layout & I/O port location.
- Note 2: Refer to Sec. 2.1 for JP 2 ~ 7 pull-high/pull-low.

2.3 I/O Port Initialization Operation

When the PC is powered-up, all operations of D/I/O port are disabled. 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 following:

- All D/I/O are disabled
- All six D/I/O ports are configured as D/I port
- All D/O latch register outputs are at high impedance.(refer to Sec. 2.4)

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

- | |
|---|
| <p>Step 1: find address-mapping of PIO/PISO cards (refer to Sec. 3.1)</p> <p>Step 2: enable all D/I/O (refer to Sec. 3.3.1)</p> <p>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)</p> <p>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)</p> |
|---|

The sample program is given as following:

```
/* step 1: find address-mapping of PIO/PISO cards */
clrscr();
wRetVal=PIO_DriverInit(&wBoards,0x80,0x01,0x30); /* for OME-PIO-D48 */
printf("\nThrer are %d PIO_D48 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);
}

/* select card_0 */
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 */

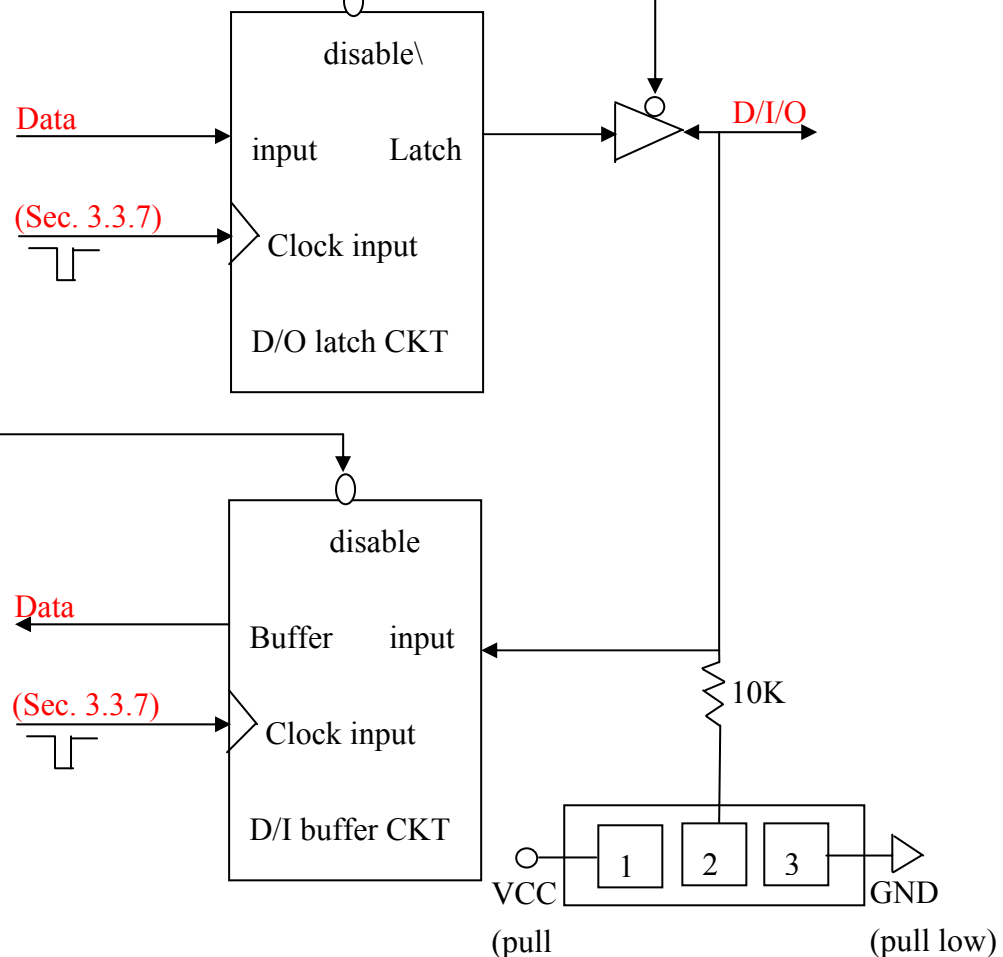
/* step 3: program 8255-1-PA-PB-PC as output port(first 3 port) */
outportb(wBase+0xcc,0x80); /* 8255-1-PA,PB,PC are all output port */
outportb(wBase+0xc0,0); /* 8255-1-PA initial value=0 */
outportb(wBase+0xc4,0); /* 8255-1-PB initial value=0 */
outportb(wBase+0xc8,0); /* 8255-1-PB initial value=0 */

/* step 4: program 8255-2-PA-PB-PC as output port(the other port)*/
outportb(wBase+0xdc,0x80); /* 8255-2-PA,PB,PC are all output port */
outportb(wBase+0xd0,0); /* 8255-2-PA initial value=0 */
outportb(wBase+0xd4,0); /* 8255-2-PB initial value=0 */
outportb(wBase+0xd8,0); /* 8255-2-PB initial value=0 */
```

2.4 D/I/O Architecture

I/O select (Sec. 3.3.7)

RESET\ (Sec. 3.3.1)



**JP2/JP3/JP4/JP5/JP6/JP7 → pull-high/pull-low select
(default: all JPs are in 2-3-short → select pull-low)**

- 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
→ can be pulled high or low selected by JP2/3/4/5/6/7
(1-2-ON=pull-high, 2-3-ON=pull-low)
- 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

There are four interrupt sources in OME-PIO-D48. These four signals are named as INT_CHAN_0, INT_CHAN_1, INT_CHAN_2 and INT_CHAN_3. Their signal sources are given as following:

INT_CHAN_0: PC3/PC7 from port-2(refer to Sec. 2.5.2)

INT_CHAN_1: PC3/PC7 from port-5(refer to Sec. 2.5.3)

INT_CHAN_2: Cout0(refer to Sec. 2.5.4)

INT_CHAN_3: Cout2(refer to Sec. 2.5.5)

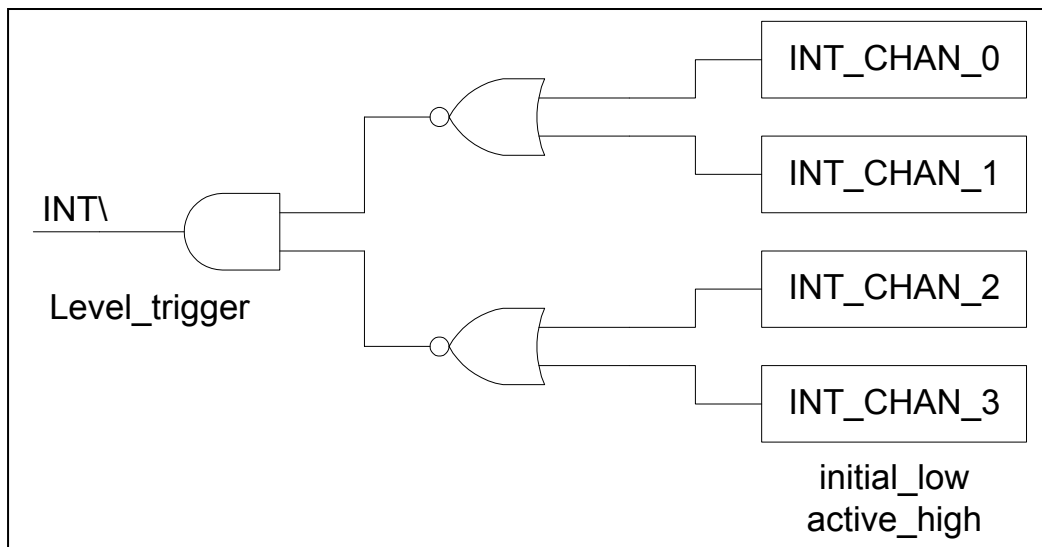
If only one interrupt signal source is used, the interrupt service routine does not have to identify the interrupt source. Refer to DEMO4.C, DEMO7.C, DEMO8.C, DEMO9.C and DEMO10.C for more information.

If there are more than one interrupt source, the interrupt service routine has to identify the active signals as following: (refer to DEMO11.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. If INT_CHAN_2 is active, service it
6. If INT_CHAN_3 is active, service it
7. 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 have 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.

2.5.1 Interrupt Block Diagram of OME-PIO-D48



The interrupt output signal of OME-PIO-D48, INT\, is **level-trigger & Active_Low**. If the INT\ generate a low-pulse, the OME-PIO-D48 will interrupt the PC once a time. If the INT\ is fixed in low level, the OME-PIO-D48 will interrupt the PC continuously. So the INT_CHAN_0/1/2/3 must be controlled in a **pulse_type** signals. **They must be fixed in low level state normally and must generate 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.

DEMO11.C → for both INT_CHAN_0 & INT_CHAN_1

If only one interrupt source is used, the interrupt service routine does not have to read the status of interrupt source. The demo programs; DEMO4.C to DEMO10.C; are designed for single-channel interrupt demo as following:

DEMO4.C → for INT_CHAN_3 only

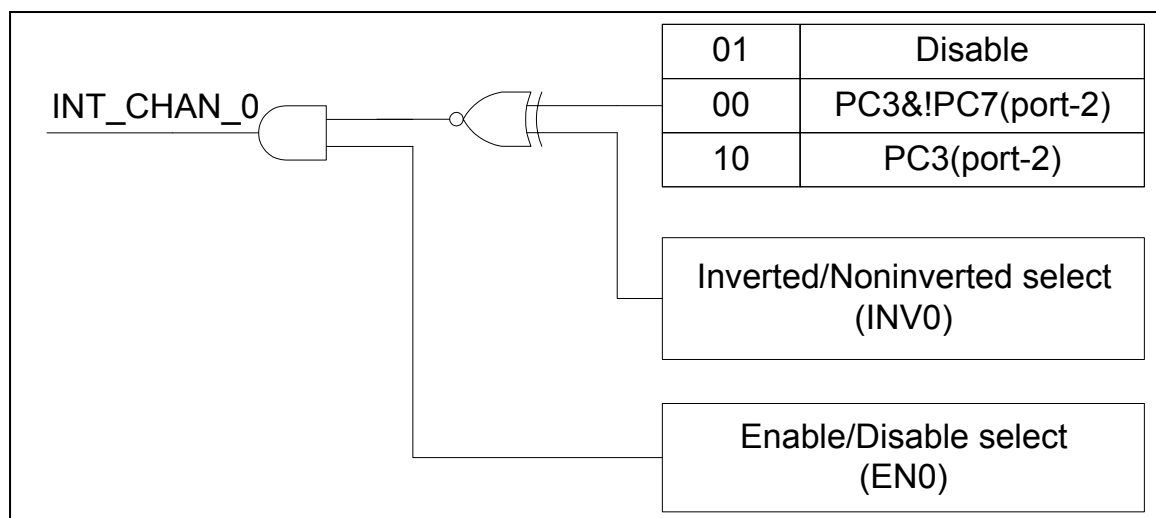
DEMO7.C → for INT_CHAN_2 only

DEMO8.C → for INT_CHAN_0 only

DEMO9.C → for INT_CHAN_0 only

DEMO10.C → for INT_CHAN_1 only

2.5.2 INT_CHAN_0



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

The INT_CHAN_0 can be equal to **PC3&!PC7** or **PC3** programmable as following:(refer to Sec. 3.3.9)

CTRL_D3=0, CTRL_D2=1 → INT_CHAN_0=disable

CTRL_D3=1, CTRL_D2=0 → INT_CHAN_0=PC3 of port-2

CTRL_D3=0, CTRL_D2=0 → INT_CHAN_0=PC3&!PC7 of port-2

The EN0 can be used to enable/disable the INT_CHAN_0 as following: (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 PC3 or PC3&!PC7 as following:
(Refer to Sec. 3.3.6)

INV0=0 → INT_CHAN_0=inverted state of (PC3 or PC3&!PC7 of port-2)

INV0=1 → INT_CHAN_0=non-inverted state of (PC3 or PC3&!PC7 of port-2)

Refer to demo program for more information as following:

DEMO8.C → for INT_CHAN_0 only (PC3 of port-2)

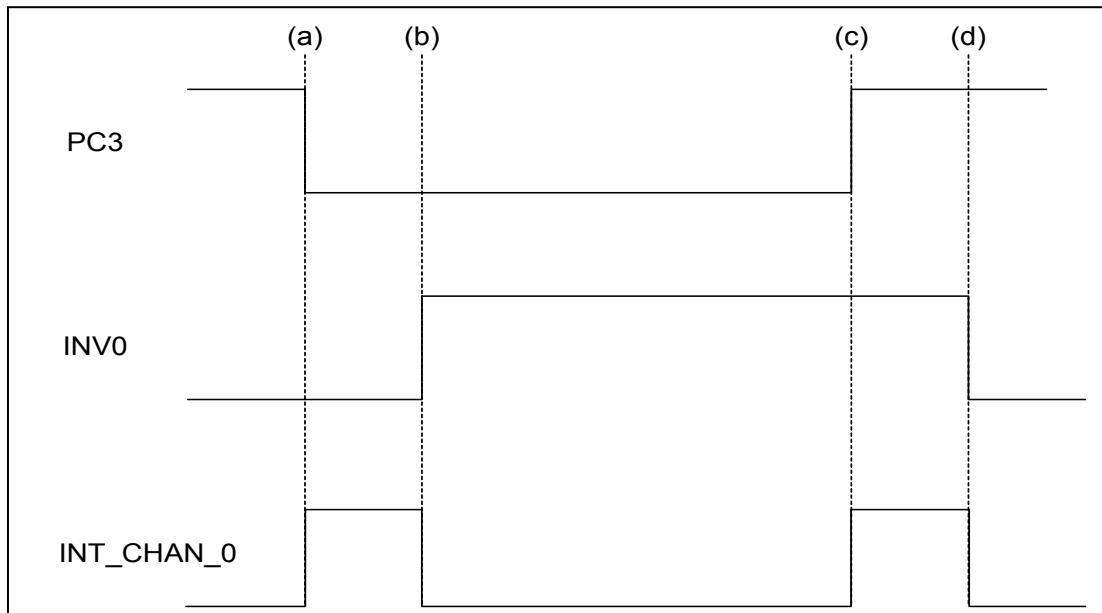
DEMO9.C → for INT_CHAN_0 only (PC3&!PC7 of port-2)

If the PC3 is a level signal, the interrupt service routine should use INVO to inverted/non-inverted the PC3 for high_pulse generation as following:

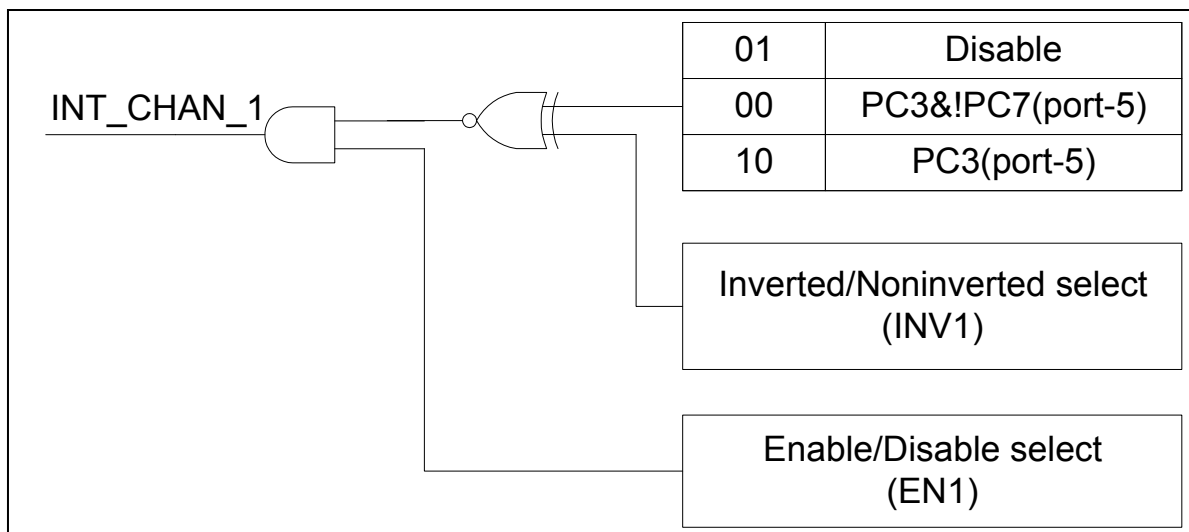
```

void interrupt irq_service() /* this ISR is in DEMO8.C          */
{
if (now_int_state==1)      /* now PC3 is changed to LOW    */(a)
{
/* --> INT_CHAN_3=!PC3=HIGH now    */
COUNT++;                /* find a LOW_pulse (PC3)    */
If((inport(wBase+7)&1)==0)/* the PC3 is still fixed in LOW */
{
/* → need to generate a high_pulse */
outportb(wBase+0x2a,1);/* INVO select the non-inverted input */(b)
/* INT_CHAN_0=PC3=LOW -->          */
/* INT_CHAN_0 generate a high_pulse */
now_int_state=0;      /* now PC3=LOW          */
}
else now_int_state=1; /* now PC3=HIGH          */
/* (no need to generate high_pulse) */
}
else
/* now PC3 is changed to HIGH    */(c)
{
/* --> INT_CHAN_0=PC3=HIGH now    */
/* find a HIGH_pulse (PC3)        */
If((inport(wBase+7)&1)==1)/* the PC3 is still fixed in HIGH */
{
/* need to generate a high_pulse */
outportb(wBase+0x2a,0);/* INVO select the inverted input */(d)
/* INT_CHAN_0=!PC3=LOW -->        */
/* INT_CHAN_0 generate a high_pulse */
now_int_state=1;      /* now PC3=HIGH          */
}
else now_int_state=0; /* now PC3=LOW          */
/* (no need to generate high_pulse) */
}
}
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}

```



2.5.3 INT_CHAN_1



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

The INT_CHAN_1 can be equal to **PC3&!PC7** or **PC3** programmable as following:(refer to Sec. 3.3.9)

CTRL_D5=0, CTRL_D4=1 → INT_CHAN_1=disable

CTRL_D5=1, CTRL_D4=0 → INT_CHAN_1=PC3 of port-5

CTRL_D5=0, CTRL_D4=0 → INT_CHAN_1=PC3&!PC7 of port-5

The EN1 can be used to enable/disable the INT_CHAN_1 as following: (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 PC3 or PC3&!PC7 as following: (Refer to Sec. 3.3.6)

INV1=0 → INT_CHAN_1=inverted state of (PC3 or PC3&!PC7 of port-5)

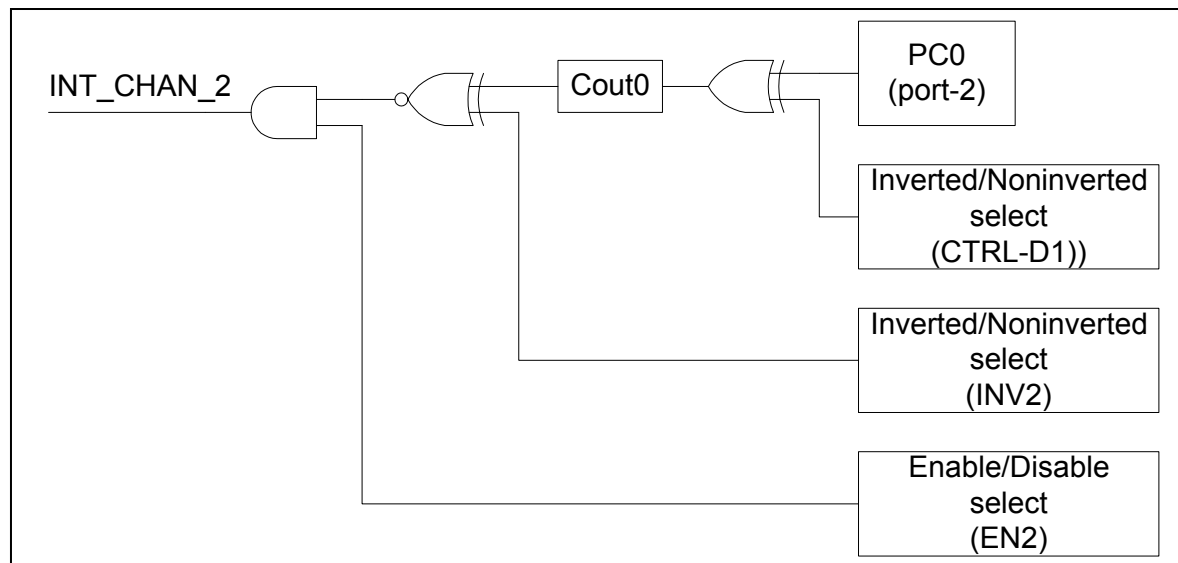
INV1=1 → INT_CHAN_1=non-inverted state of (PC3 or PC3&!PC7 of port-5)

Refer to demo program for more information as following:

DEMO10.C → for INT_CHAN_1 only (PC3&!PC7 of port-5)

NOTE: refer to Sec. 2.5.2 for active high-pulse generation.

2.5.4 INT_CHAN_2



The INT_CHAN_2 must be fixed in low level state normally and must generate a high_pulse to interrupt the PC.

The PC0 (port-2) can be inverted/non-inverted programmable as following:
(refer to Sec. 3.3.9)

CTRL_D1=0 → Cin0=PC0 of port-2

CTRL_D1=1 → Cin0=!PC0 of port-2

The EN2 can be used to enable/disable the INT_CHAN_2 as following: (refer to Sec. 3.3.4)

EN2=0 → INT_CHAN_2=disable

EN2=1 → INT_CHAN_2=enable

The INV2 can be used to invert/non-invert the Cout0 as following: (Refer to Sec. 3.3.6)

INV2=0 → INT_CHAN_2=inverted state of (Cout0)

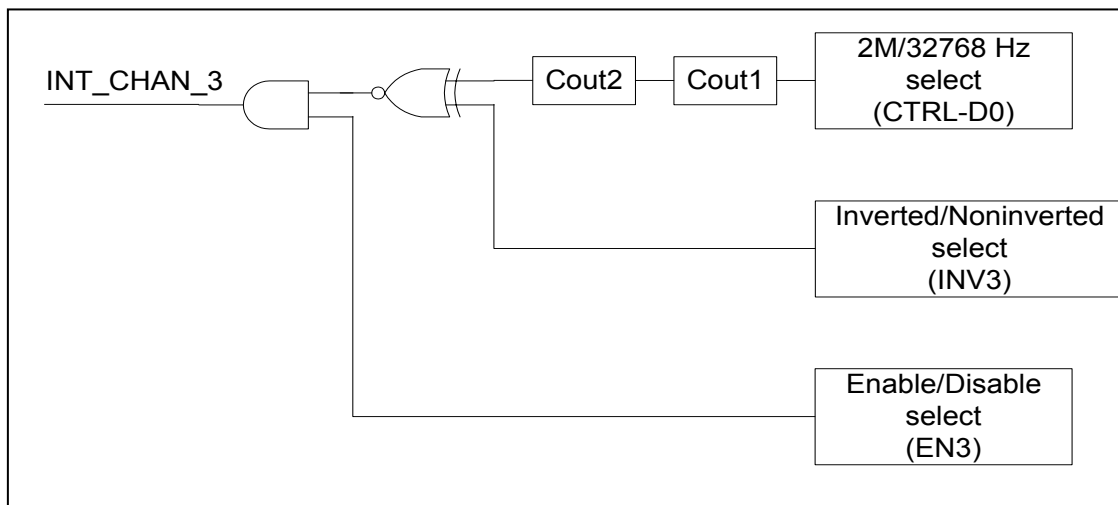
INV2=1 → INT_CHAN_2=non-inverted state of (Cout0)

Refer to demo program for more information as following:

DEMO7.C → for INT_CHAN_2 only (Cout0)

NOTE: refer to Sec. 2.5.2 for active high-pulse generation.

2.5.5 INT_CHAN_3



The INT_CHAN_3 must be fixed in low level state normally and generated a high_pulse to interrupt the PC.

The Cin1 can be 2M/32768Hz programmable as following: (refer to Sec. 3.3.9)

CTRL_D0=0 → Cin1=2M clock source

CTRL_D0=1 → Cin1=32768 Hz clock source

The EN3 can be used to enable/disable the INT_CHAN_3 as following: (refer to Sec. 3.3.4)

EN3=0 → INT_CHAN_3=disable

EN3=1 → INT_CHAN_3=enable

The INV3 can be used to invert/non-invert the Cout0 as following: (Refer to Sec. 3.3.6)

INV2=3 → INT_CHAN_3=inverte of (Cout2)

INV2=3 → INT_CHAN_3=non-inverte of (Cout2)

Refer to demo program for more information as following:

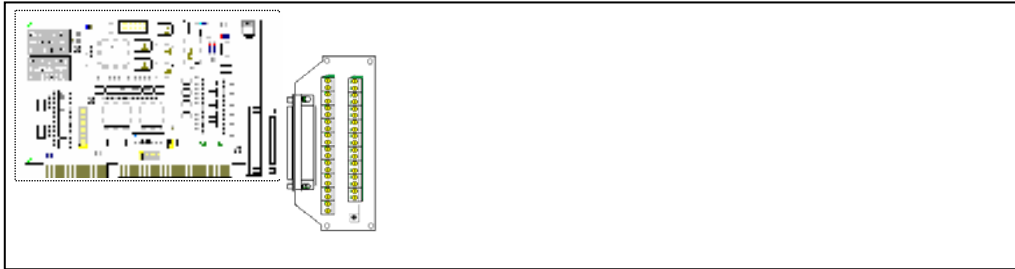
DEMO4.C → for INT_CHAN_3 only (Cout2)

NOTE: refer to Sec. 2.5.2 for active high-pulse generation.

2.6 Daughter Boards

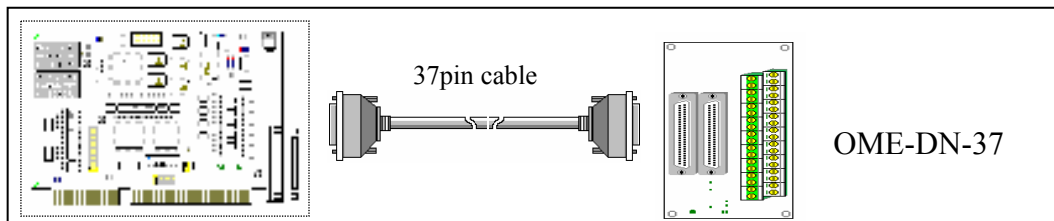
2.6.1 OME-DB-37

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



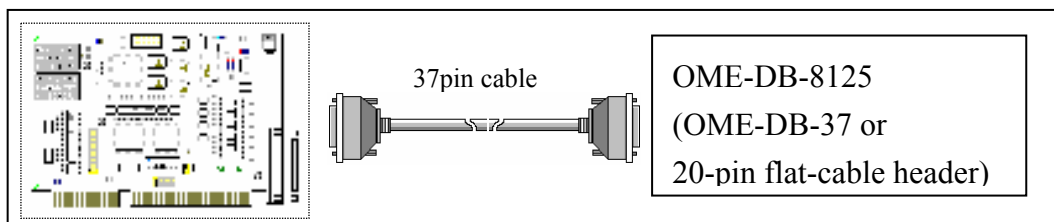
2.6.2 OME-DN-37 & OME-DN-50

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



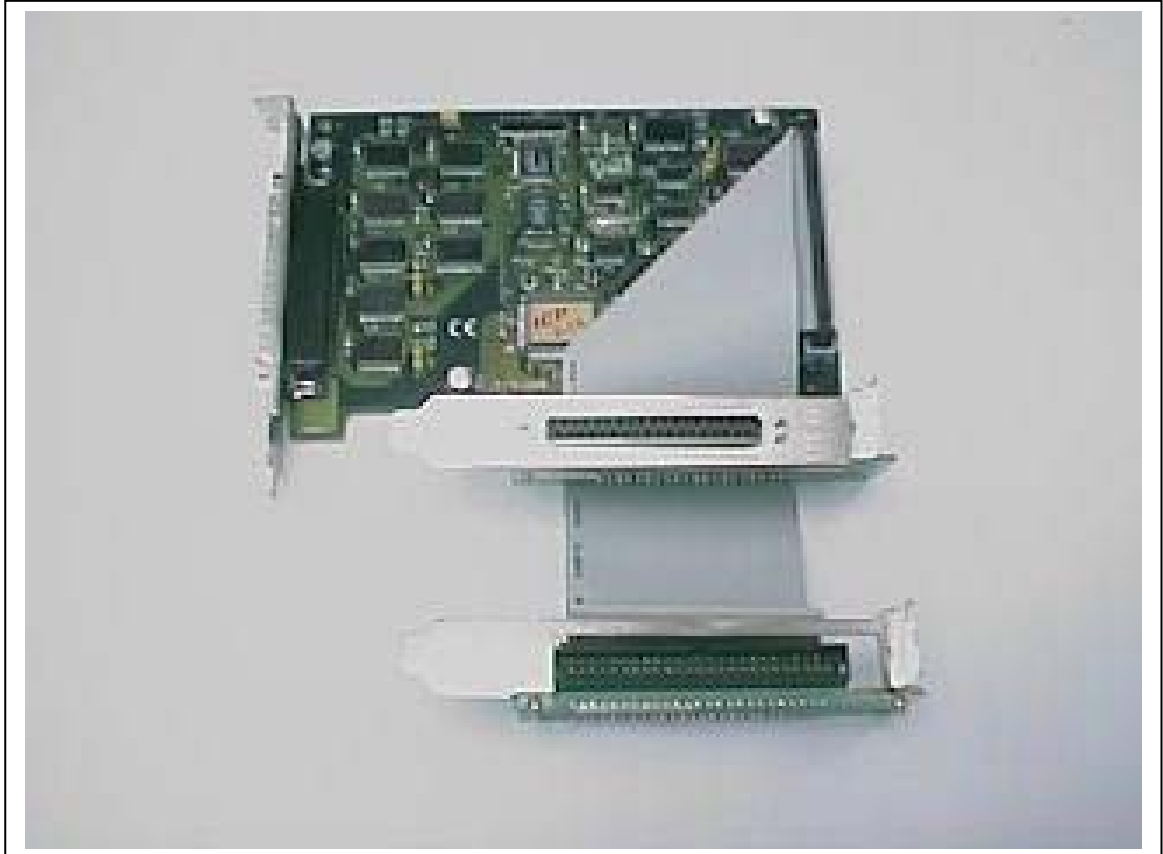
2.6.3 OME-DB-8125

The OME-DB-8125 is a general purpose screw terminal board designed for ease of wiring. There are one DB-37 & two 20-pin flat-cable headers in the OME-DB-8125.



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

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



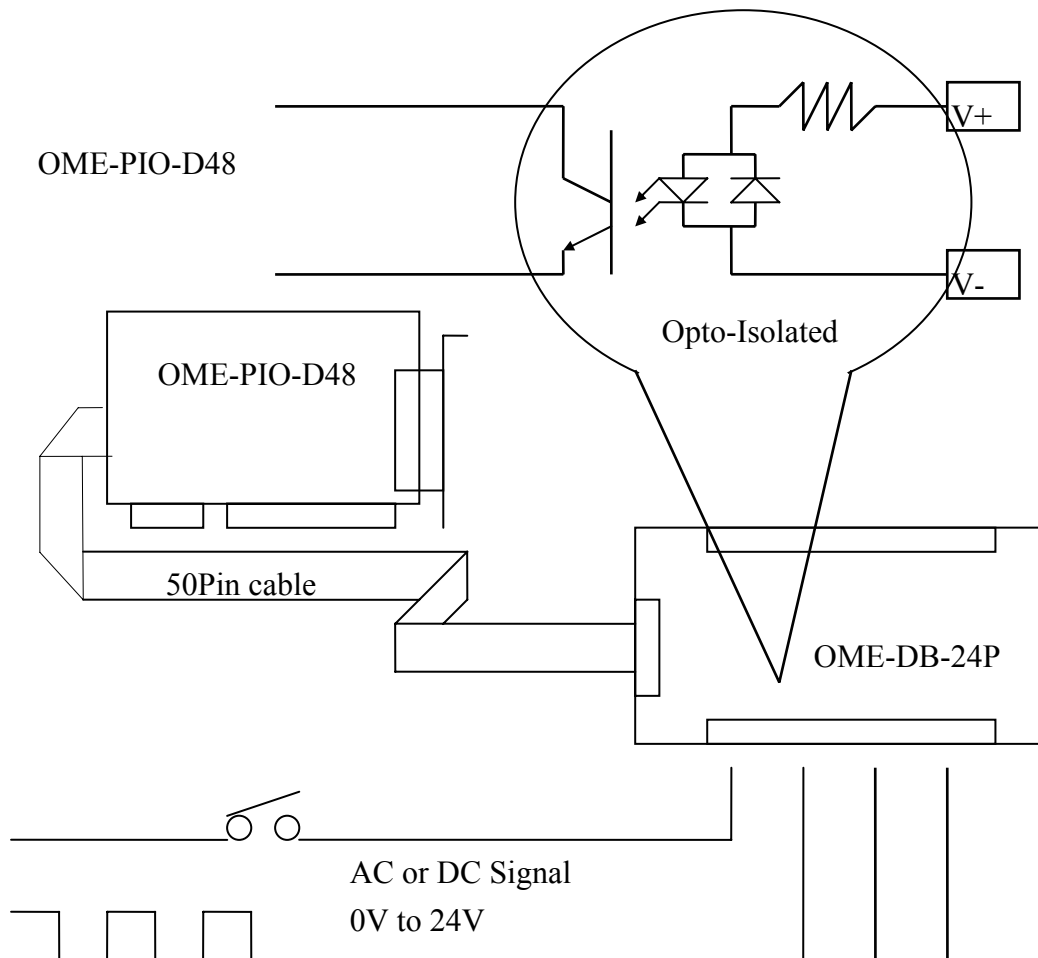
OME-ADP-37/PCI: 50-pin header to OME-DB-37 extender.

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

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

2.6.5 OME-DB-24P/PD Isolated Input Board

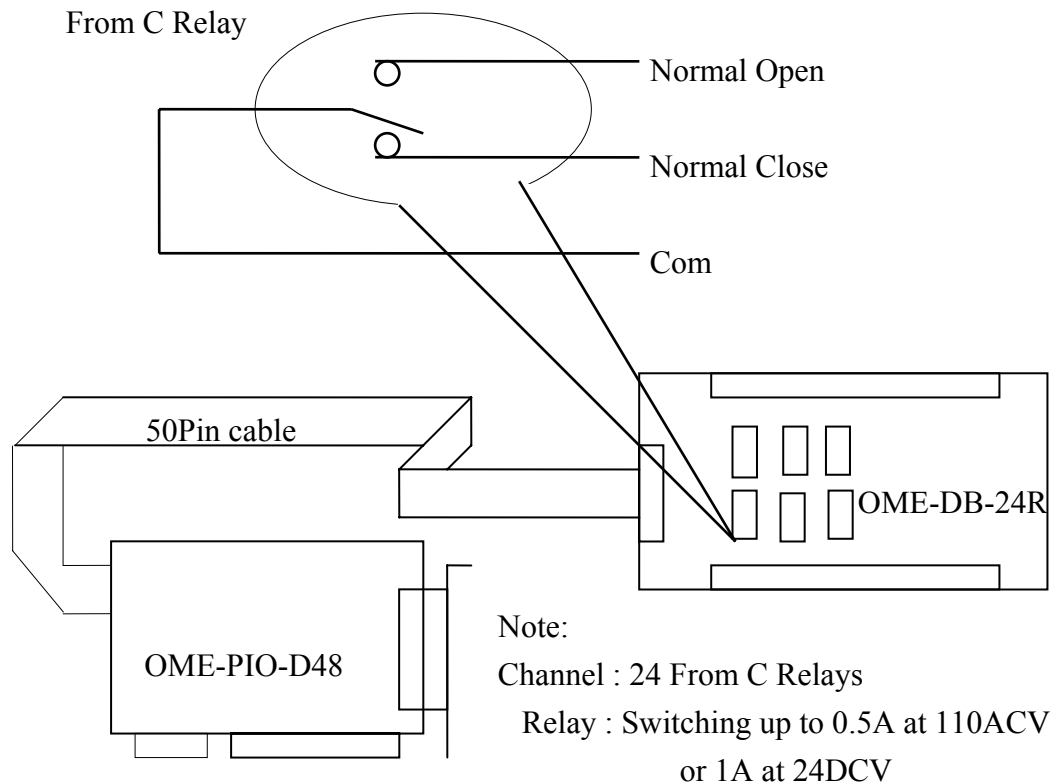
The OME-DB-24P/DB-24PD is a 24-channel isolated digital input daughter board. The optically isolated inputs of the OME-DB-24P/DB-24PD, consists of a bi-directional opto-coupler with a resistor for current sensing. You can use the OME-DB-24P/DB-24PD to sense DC signal from TTL levels up to 24V or use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



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

2.6.6 OME-DB-24R/RD Relay Board

The OME-DB-24R/DB-24RD, 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 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.



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

The other output daughter boards are given as following:

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

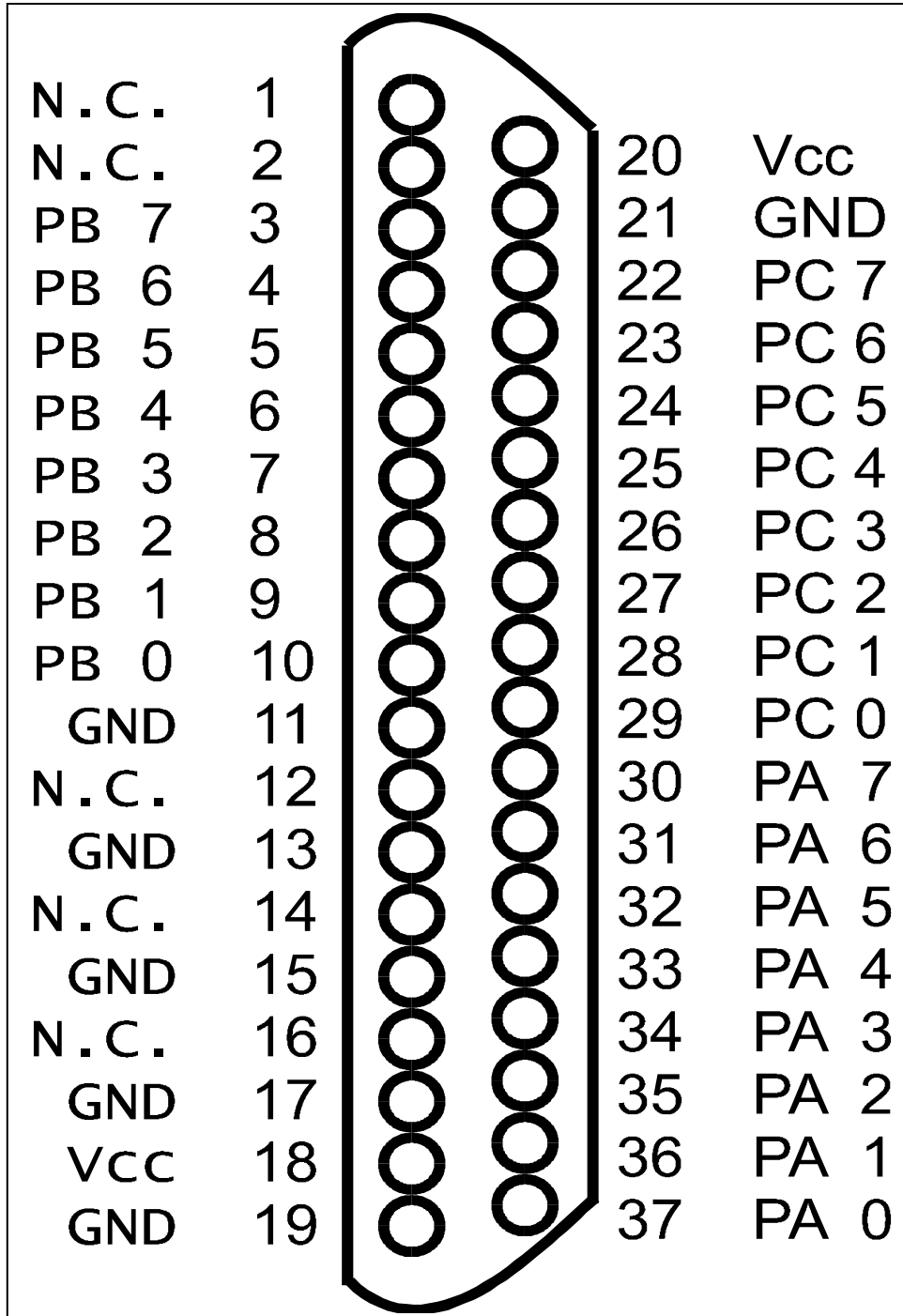
2.6.7 Daughter Board Comparison Table

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

NOTE: There is no 20-pin header in OME-PIO-D48. The OME-PIO-D48 has one D-Sub 37 connector and one 50-pin flat-cable header.

2.7 Pin Assignment

CN1: 37 pin of D-type female connector. (port-0, port-1, port-2)

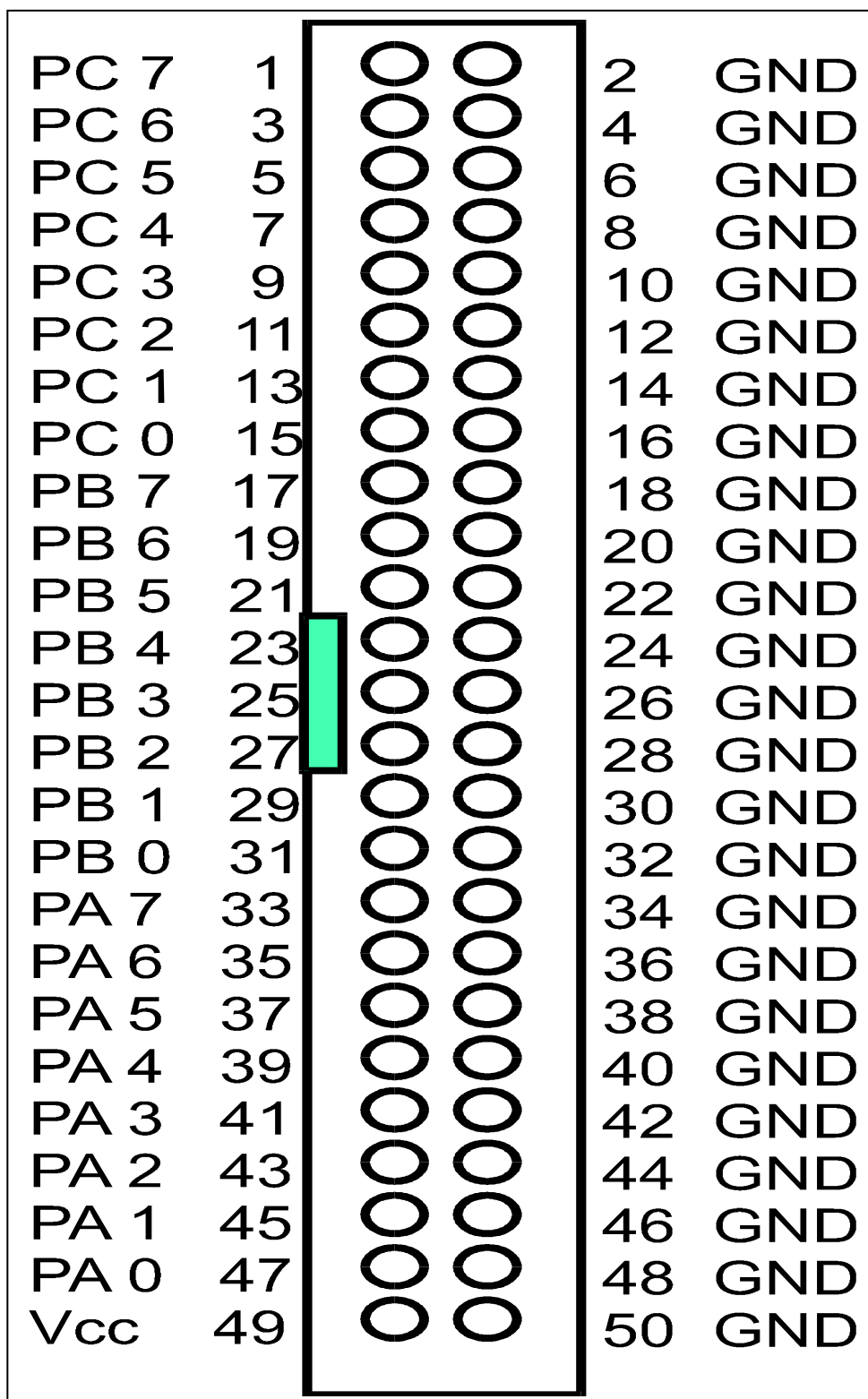


PA0 ~ PA7 : port-0

PB0 ~ PB7 : port-1

PC0 ~ PC7 : port-2

CN2: 50-pin of flat-cable connector (port-3, port-4, port-5)



PA0 ~ PA7 : port-3

PB0 ~ PB7 : port-4

PC0 ~ PC7 : port-5

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 OME-PIO/PISO series card in the power-up stage. The IDs of OME-PIO-D48 cards are given as follows:

- | | |
|-------------------------|----------------------------|
| < REV 1.0 ~ REV 5.0 > : | < REV 6.0 or above > : □ |
| • Vendor ID = 0xE159 | • Vendor ID = 0xE159 □ |
| • Device ID = 0x0002 | • Device ID = 0x0001 □ |
| • Sub-vendor ID = 0x80 | • Sub-vendor ID = 0x0080 □ |
| • Sub-device ID = 0x01 | • Sub-device ID = 0x01 □ |
| • Sub-aux ID = 0x30 | • Sub-aux ID = 0x30 □ |

We provide all necessary functions as following:

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

All functions are defined in PIO.H. Refer to Chapter 4 for more information. The important driver information is given as following:

1. Resource-allocated information:

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

2. PIO/PISO identification information:

- wSubVendor: subVendor ID of this board
- wSubDevice: subDevice ID of this board
- wSubAux: 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 OME-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 OME-PIO/PISO series cards installed in this system & save all their resource in the library.

Sample program 1: find all OME-PIO-D48 in this PC

```
wSubVendor=0x80; wSubDevice=1; wSubAux=0x30; /* for PIO_D48 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("Threr are %d OME-PIO-D48 Cards in this PC\n",wBoards);

/* step2: save resource of all OME-PIO-D48 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 OME-PIO/PISO series card are given as following:

OME-PIO/PISO series card	Description	Sub_vendor	Sub_device	Sub_AUX
OME-PIO-D144 (Rev 4.0)	144 * D/I/O	5C80	01	00
OME-PIO-D96 (Rev 4.0)	96 * D/I/O	5880	01	10
OME-PIO-D64 (Rev 2.0)	64 * D/I/O	4080	01	20
OME-PIO-D56 (Rev 6.0)	24* D/I/O + 16*D/I + 16*D/O	C080	01	40
OME-PIO-D48 (Rev 2.0)	48*D/I/O	0080	01	30
OME-PIO-D24 (Rev 6.0)	24*D/I/O	C080	01	40
OME-PIO-821	Multi-function	80	03	10
OME-PIO-DA16 (Rev 4.0)	16*D/A	4180	00	00
OME-PIO-DA8 (Rev 4.0)	8*D/A	4180	00	00
OME-PIO-DA4 (Rev 4.0)	4*D/A	4180	00	00
OME-PISO-C64 (Rev 4.0)	64 * isolated D/O (Current Sinking)	0280	00	00
OME-PISO-A64 (Rev 3.0)	64 * isolated D/O (Current Sourcing)	0280	00	50
OME-PISO-P64 (Rev 4.0)	64 * isolated D/I	0280	00	10
OME-PISO-P32C32 (Rev 5.0)	32 * isolated D/O (Current Sinking) +32 * isolated D/I	80	08	20
OME-PISO-P32A32 (Rev 3.0)	32 * isolated D/O (Current Sourcing) +32 * isolated D/I	8280	00	70
OME-PISO-P8R8 (Rev 2.0)	8* isolated D/I + 8 * 220V relay	4200	00	30
OME-PISO-P8SSR8AC (Rev 2.0)	8* isolated D/I + 8 * SSR /AC	4200	00	30
OME-PISO-P8SSR8DC (Rev 2.0)	8* isolated D/I + 8 * SSR /DC	4200	00	30
OME-PISO-730 (Rev 2.0)	16*DI + 16*D/O + 16* isolated D/I + 16* isolated D/O (Current Sinking)	C2FF	00	40
OME-PISO-730A (Rev 3.0)	16*DI + 16*D/O + 16* isolated D/I + 16* isolated D/O (Current Sourcing)	62FF	00	80
OME-PISO-813 (Rev 2.0)	32 * isolated A/D	4280	02	00
OME-PISO-DA2 (Rev 5.0)	2 * isolated D/A	4280	03	00

Note: If your board is a different version, it may also have different sub IDs. We offer the same function calls irrespective of the board version.

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 following:

```
/* step1: detect all OME-PIO-D48 cards first */
wSubVendor=0x80; wSubDevice=1; wSubAux=0x30; /* for PIO_D48 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("Threr are %d OME-PIO-D48 Cards in this PC\n",wBoards);

/* step2: save resource of all OME-PIO-D48 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=%0x, wIrq=%0x", 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 OME-PIO-D48 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 following:

```
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff); /*find all PIO_PISO*/
printf("\nThrer are %d PIO_PISO Cards in this PC",wBoards);
if (wBoards==0 ) exit(0);

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

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

3.2 The Assignment of I/O Address

The plug & play BIOS will assign the proper I/O address to OME-PIO/PISO series card. If there is only one OME-PIO/PISO board, the user can identify the board as card_0. If there are two OME-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 following:

1. Remove all OME-PIO-D48 from this PC
2. Install one OME-PIO-D48 into the PC's PCI_slot1, run PIO_PISO.EXE & record the wSlotBus1 & wSlotDevice1
3. Remove all OME-PIO-D48 from this PC
4. Install one OME-PIO-D48 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 following:

PC's PCI slot	wSlotBus	wSlotDevice
Slot_1	0	0x07
Slot_2	0	0x08
Slot_3	0	0x09
Slot_4	0	0x0A
PCI-BRIDGE		
Slot_5	1	0x0A
Slot_6	1	0x08
Slot_7	1	0x09
Slot_8	1	0x07

The above procedure 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 following:

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 addresses of OME-PIO-D48 are given as following:

Address	Read	Write
wBase+0	RESET\ control register	Same
wBase+2	Aux control register	Same
wBase+3	Aux data register	Same
wBase+5	INT mask control register	Same
wBase+7	Aux pin status register	Same
wBase+0x2a	INT polarity control register	Same
wBase+0xc0	Read 8255-1-PA(port-0)	Write 8255-1-PA(port-0)
wBase+0xc4	Read 8255-1-PB(port-1)	Write 8255-1-PB(port-1)
wBase+0xc8	Read 8255-1-PC(port-2)	Write 8255-1-PC(port-2)
wBase+0xcc	Read 8255-1 control word	Write 8255-1 control word
wBase+0xd0	Read 8255-2-PA(port-3)	Write 8255-2-PA(port-3)
wBase+0xd4	Read 8255-2-PB(port-4)	Write 8255-2-PB(port-4)
wBase+0xd8	Read 8255-2-PC(port-5)	Write 8255-2-PC(port-5)
wBase+0xdc	Read 8255-2 control word	Write 8255-2 control word
wBase+0xe0	Read 8254-counter0	Write 8254-counter0
wBase+0xe4	Read 8254-counter1	Write 8254-counter1
wBase+0xe8	Read 8254-counter2	Write 8254-counter2
wBase+0xec	Read 8254 control word	Write 8254 control word
wBase+0xf0	Read clock/int control word	Write clock/int control word

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	EN3	EN2	EN1	EN0

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

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

EN0=1 → enable INT_CHAN_0 as a interrupt signal

EN1=0 → disable INT_CHAN_1 as a interrupt signal (default)

EN1=1 → enable INT_CHAN_1 as a interrupt signal

EN2=0 → disable INT_CHAN_2 as a interrupt signal (default)

EN2=1 → enable INT_CHAN_2 as a interrupt signal

EN3=0 → disable INT_CHAN_3 as a interrupt signal (default)

EN3=1 → enable INT_CHAN_3 as a interrupt signal

```
outputb(wBase+5,0);      /* disable all interrupts      */
outputb(wBase+5,1);      /* enable interrupt of INT_CHAN_0 */
outputb(wBase+5,2);      /* enable interrupt of INT_CHAN_1 */
outputb(wBase+5,4);      /* enable interrupt of INT_CHAN_2 */
outputb(wBase+5,8);      /* enable interrupt of INT_CHAN_3 */
outputb(wBase+5,0x0f);   /* enable all four channels of interrupt */
```

Refer to the following demo program for more information:

DEMO4.C → for INT_CHAN_3 only

DEMO7.C → for INT_CHAN_2 only

DEMO8.C → for INT_CHAN_0 only

DEMO9.C → for INT_CHAN_0 only

DEMO10.C → for INT_CHAN_1 only

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, Aux2=INT_CHAN_2,
Aux3=INT_CHAN_3, Aux7~4=Aux-ID. Refer to Sec. 4.1 for more information. The
Aux 0~3 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	INV3	INV2	INV1	INV0

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

INV0=0 → select the inverted signal from interrupt_channel_0

INV0=1 → select the non-inverted signal from interrupt_channel_0

INV1=control interrupt channel_1, INV2=control interrupt channel_2

INV3=control interrupt channel_3

```
outputb(wBase+0x2a,0); /* select the inverted input from all 4 channels */
```

```
outputb(wBase+0x2a,0x0f); /* select the non-inverted input from all 4 channels */
```

```
outputb(wBase+0x2a,0x0e); /* select the inverted input of INT_CHAN_0 */
```

```
/* select the non-inverted input from the others */
```

```
outputb(wBase+0x2a,0x0c); /* select the inverted input of INT_CHAN_0 & */
```

```
/* INT_CHAN_1 */
```

```
/* select the non-inverted input from the others */
```

Refer to Sec. 2.5 for more information.

Refer to DEMO5.C for more information.

3.3.7 Read/Write 8255-1 & 8255-2 (I/O port)

(Read/Write): wBase+0xc0=8255-1-PA → port_0

(Read/Write): wBase+0xc4=8255-1-PB → port_1

(Read/Write): wBase+0xc8=8255-1-PC → port_2

(Read/Write): wBase+0xcc=8255-1 control word → control D/I or D/O of port_0/1/2

(Read/Write): wBase+0xd0=8255-2-PA → port_3

(Read/Write): wBase+0xd4=8255-2-PB → port_4

(Read/Write): wBase+0xd8=8255-2-PC → port_5

(Read/Write): wBase+0xdc=8255-2 control word → control D/I or D/O of port_3/4/5

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

8255 control word (mode-0)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	D4	D3	0	D1	D0

D4: 1→PA=input, 0→PA=output

D3: 1→PC(high nipple)=input, 0→PC(high nipple)=output

D1: 1→PB=input, 0→PB=output

D0: 1→PC(low nipple)=input, 0→PC(low nipple)=output

There are six 8-bit I/O port in the OME-PIO-D48. Every I/O port can be programmed as D/I or D/O port based on control word setting. All six ports are configured as D/I ports when the first power on.

```
outputb(wBase+0xcc,0x80); /* port-0, port-1, port-2 are D/O port */
```

```
outputb(wBase+0xc0,V1); /* write to port_0 (PA) */
```

```
outputb(wBase+0xc4,V2); /* write to port_1 (PB) */
```

```
outputb(wBase+0xc8,V3); /* write to port_2 (PC) */
```

```
outputb(wBase+0xdc,0x9B); /* port-3, port-4, port-5 are D/I port */
```

```
V1=inputb(wBase+0xd0); /* read from port_3 (PA) */
```

```
V2=inputb(wBase+0xd4); /* read from port_4 (PB) */
```

```
V3=inputb(wBase+0xd8); /* read from port_5 (PC) */
```

3.3.8 Read/Write 8254

(Read/Write): wBase+0xe0=8254-counter-0

(Read/Write): wBase+0xe4=8254-counter-1

(Read/Write): wBase+0xe8=8254-counter-2

(Read/Write): wBase+0xec=8254 control word

8254 control word

SC1	SC0	RL1	RL0	M2	M1	M0	BCD
-----	-----	-----	-----	----	----	----	-----

BCD: 0: binary count 1: BCD count

M2,M1,M0: 000:mode0 interrupt on terminal count

001:mode1 programmable one-shot

010:mode2 rate generator

011:mode3 square-wave generator

100:mode4 software triggered pulse

101:mode5 hardware triggered pulse

RL1,RL0: 00: counter latch instruction

01: read/write low counter byte only

10: read/write high counter byte only

11: read/write low counter byte first, then high counter byte

SC1,SC0: 00: counter0

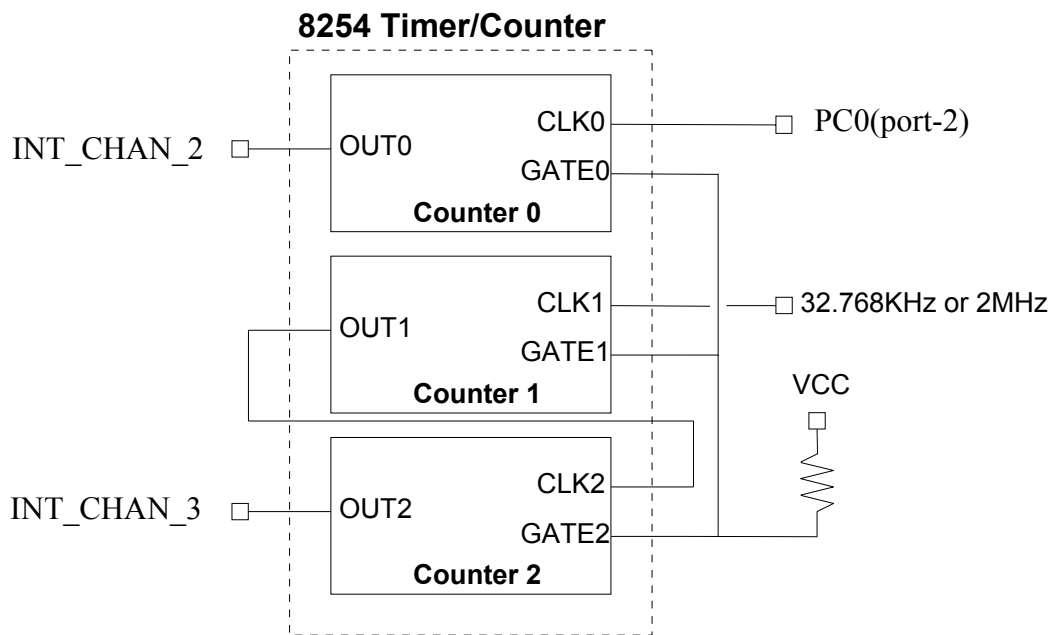
01: counter1

10: counter2

11: read -back command

```
WORD pio_d48_c0(char cConfig, char cLow, char cHigh)/*COUNTER_0 */
{
    outportb(wBase+0xec, cConfig);
    outportb(wBase+0xe0, cLow);
    outportb(wBase+0xe0, cHigh);
    return (NoError);
}
WORD pio_d48_c1(char cConfig, char cLow, char cHigh)/*COUNTER_1 */
{
    outportb(wBase+0xec, cConfig);
    outportb(wBase+0xe4, cLow);
    outportb(wBase+0xe4, cHigh);
    return (NoError);
}
WORD pio_d48_c2(char cConfig, char cLow, char cHigh)/*COUNTER_2 */
{
    outportb(wBase+0xec, cConfig);
    outportb(wBase+0xe8, cLow);
    outportb(wBase+0xe8, cHigh);
    return (NoError);
}
```

The configuration of 8254 counter



Refer to the following demo program for more information:

- DEMO4.C: counter0 demo (using interrupt INT_CHAN_3)
- DEMO5.C: counter1-counter2 demo (no interrupt)
- DEMO6.C: counter1-counter2 demo (no interrupt)
- DEMO7.C: counter1-counter2 demo (using interrupt INT_CHAN_2)

3.3.9 Read/Write Clock/Int Control Register

(Read/Write): wBase+0xf0=clock/int control register

0	0	CTRL-D5	CTRL-D4	CTRL-D3	CTRL-D2	CTRL-D1	CTRL-D0
---	---	---------	---------	---------	---------	---------	---------

CTRL-D0: timer source CLK1selection (refer to Sec. 3.3.8)

0:2MHz, 1:32.768KHz

(refer to Sec. 4.5: DEMO4.C)

CTRL-D1: invert/non-invert the PC0 of port-2 (refer to Sec. 2.5.4)

0: non-invert, 1: invert

(refer to Sec. 4.6: DEMO5.C)

CTRL-D3, CTRL-D2: interrupt source select (refer to Sec. 2.5.2)

01 : disable PC3 & !PC7 (of port-2) as interrupt source

10 : INT_CHAN_0=PC3 of port-2

00 : INT_CHAN_0=PC3&!PC7 of port-2

(refer to Sec. 4.9: DEMO8.C)

CTRL-D5, CTRL-D4: interrupt source select (refer to Sec. 2.5.3)

01 : disable PC3 & !PC7 (of port-5) as interrupt source

10 : INT_CHAN_1=PC3 of port-5

00 : INT_CHAN_1=PC3&!PC7 of port-5

(refer to Sec. 4.11: DEMO10.C)

4. Demo Program

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

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 following:

- \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 validated for all OME-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();
}
```

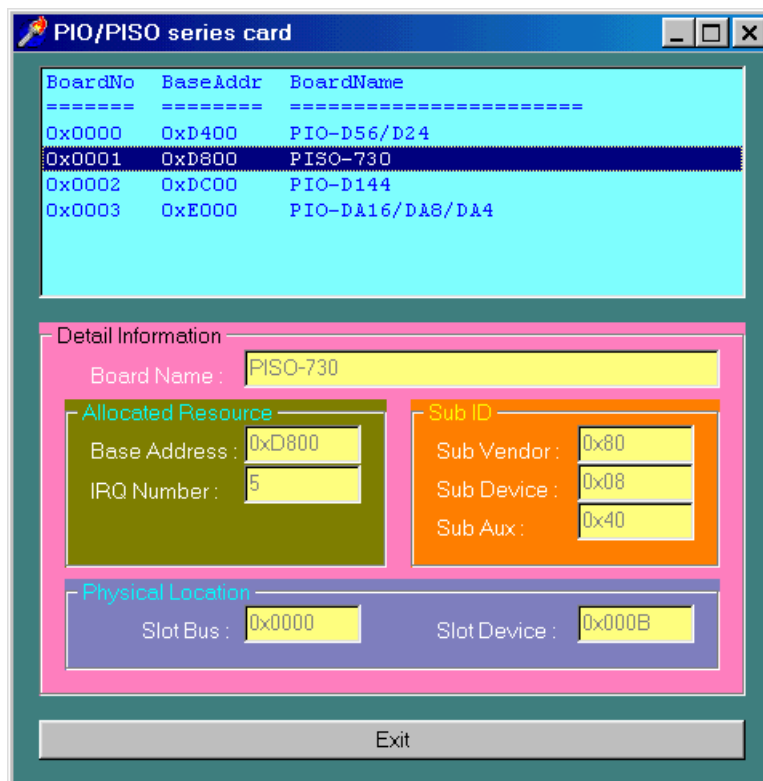
NOTE: the PIO_PISO.EXE is valid for all PIO/PISO cards. The user can execute the PIO_PISO.EXE to get the following information:

- 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 software floppy disk or CD. It is useful for all PIO/PISO series card.

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



4.2 DEMO1

```
/* ----- */
/* demo1 : D/O demo for OME-PIO-D48 */
/* step 1 : connect CN1 of OME-PIO-D48 to OME-DB-24C */
/* step 2 : connect CN2 of OME-PIO-D48 to another OME-DB-24C */
/* step 3 : run DEMO1.EXE */
/* ----- */

#include "PIO.H"

WORD wBase,wIrq;
WORD wBase2,wIrq2;

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

/* step 1: find address-mapping of PIO/PISO cards */
clrscr();
wRetVal=PIO_DriverInit(&wBoards,0x80,0x01,0x30); /* for OME-PIO-D48 */
printf("\nThrer are %d PIO_D48 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);
}

/* select card 0 */
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 */

/* step 3: program 8255-1-PA-PB-PC as output port */
outportb(wBase+0xcc,0x80); /* 8255-1-PA,PB,PC are all output port*/
outportb(wBase+0xc0,0); /* 8255-1-PA initial=0 */
outportb(wBase+0xc4,0); /* 8255-1-PB initial=0 */
outportb(wBase+0xc8,0); /* 8255-1-PB initial=0 */

/* step 4: program 8255-2-PA-PB-PC as output port */
outportb(wBase+0xdc,0x80); /* 8255-2-PA,PB,PC are all output port*/
outportb(wBase+0xd0,0); /* 8255-2-PA initial=0 */
outportb(wBase+0xd4,0); /* 8255-2-PB initial=0 */
outportb(wBase+0xd8,0); /* 8255-2-PB initial=0 */
```

```

printf("\n----- Test D/O (DB-24C)-----");
i=1;
for (;;)
{
/* send test pattern to CN1 of PIO_D48 */
outportb(wBase+0xc0,i); printf("\nOutput=%x",i);
outportb(wBase+0xc4,i);
outportb(wBase+0xc8,i);

/* send test pattern to CN2 of PIO_D48 */
outportb(wBase+0xd0,i);
outportb(wBase+0xd4,i);
outportb(wBase+0xd8,i);

delay(1000);

if (kbhit() !=0)
{
c=getch(); if ((c=='q') || (c=='Q')) return;
}

i=i<<1;          /* next bit */
if (i>0x80)
{
i=1;
c=getch();
if ((c=='Q') || (c=='q')) return;
printf("\n----- Test D/O (DB-24C)-----");
}

}

PIO_DriverClose();
}

```

4.3 DEMO2

```
/* ----- */
/* dem01 : D/O demo for OME-PIO-D48 */
/* step 1 : connect CN1 of OME-PIO-D48 to OME-DB-24P */
/* step 2 : connect CN2 of OME-PIO-D48 to another OME-DB-24P */
/* step 3 : run DEMO2.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,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;
float ok,err;

/* step 1: find address-mapping of PIO/PISO cards */
:
/* select card_0 */
:
/* step 2: enable all D/I/O port */
:
/* step 3: program 8255-1-PA-PB-PC as input port */
outportb(wBase+0xcc,0x9B);/* 8255-1-PA,PB,PC are all input port */
/* step 4: program 8255-2-PA-PB-PC as input port */
outportb(wBase+0xdc,0x9B);/* 8255-2-PA,PB,PC are all input port */

printf("\n----- Test D/I (DB-24P)-----");
i=1; ok=err=0.0;
for (;;)
{
j1=inportb(wBase+0xc0); /*read D/I data from CN1 of PIO_D48*/
j2=inportb(wBase+0xc4);
j3=inportb(wBase+0xc8);

j11=inportb(wBase+0xd0); /*read D/I data from CN2 of PIO_D48*/
j22=inportb(wBase+0xd4);
j33=inportb(wBase+0xd8);

printf("\n[%2x %2x %2x] [%2x %2x %2x]",j1,j2,j3,j11,j22,j33);
delay(1000);

if (kbhit()!=0)
{
c=getch(); if ((c=='q') || (c=='Q')) return;
}
}
PIO_DriverClose();
}
```

4.4 DEMO3

```
/* ----- */
/* demo1 : D/O demo for OME-PIO-D48 */
/* step 1 : connect a ADP-37/PCI to CN2 of OME-PIO-D48 */

/* step 2 : connect CN1 of OME-PIO-D48 to ADP-37/PCI in step 1 */ □
/* (by one DB-37 to DB-37 cable) */ □
/* step 3 : run DEMO3.EXE */ □
/* ----- */ □
#include "PIO.H" □
WORD wBase,wIrq; WORD wBase2,wIrq2; □
int main() □
{ □
    int i,j,k,jj,dd; □
    WORD wBoards,wRetVal,j1,j2,j3,t1,t2,t3,t4,t5; □
    WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice; □
    char c; □
□
    /* step 1: find address-mapping of PIO/PISO cards */ □
    . □
    /* step 2: enable all D/I/O port */ □
    . □
    /* step 3: program 8255-1-PA-PB-PC as output port */ □
    . □
    /* step 4: program 8255-2-PA-PB-PC as input port */ □
    . □
    /* step 5: read/write test pattern */ □
□
    printf("\n----- Test1: D/I/O CN1-CN2 -----");
```

```

for (;;)
{
    /* send test pattern to CN1 of PIO_D48 */
    outportb(wBase+0xc0,i); outportb(wBase+0xc4,i);
    outportb(wBase+0xc8,i);
    printf("\nCN1=output, CN2=input, Output=%x",i);
    delay(1000);      /* delay for D/O settle time */
    /* read test pattern to CN2 of PIO_D48 */
    j1=inportb(wBase+0xd0);
    j2=inportb(wBase+0xd4);
    j3=inportb(wBase+0xd8);
    printf(" , DI=[%2x,%2x,%2x]",j1,j2,j3);
    delay(1000);
    if (kbhit()!=0)
    {
        c=getch();
        if ((c=='q') || (c=='Q')) break;
    }
}

```

```

i=i<<1;      /* next bit */   □
if (i>0x80)  □
  { □
    i=1;

    c=getch();
    if ((c=='Q') || (c=='q')) break;
    printf("\n----- Test1: D/I/O CN1-CN2 -----");
  }

}

printf("\n----- Test2: D/I/O CN1-CN2 -----");

/* step 6: program 8255-1-PA-PB-PC as input port          */
outportb(wBase+0xcc,0x9B); /*8255-1-PA,PB,PC are all output port */

/* step 7: program 8255-2-PA-PB-PC as output port       */
outportb(wBase+0xdc,0x80); /* 8255-2-PA,PB,PC are all input port */
outportb(wBase+0xd0,0);   /* 8255-1-PA initial=0          */
outportb(wBase+0xd4,0);   /* 8255-1-PB initial=0          */
outportb(wBase+0xd8,0);   /* 8255-1-PB initial=0          */

/* step 8: read/write test pattern                       */
i=1;
for (;;)
{
  /* send test pattern to CN2 of PIO_D48 */
  outportb(wBase+0xd0,i);
  outportb(wBase+0xd4,i);
  printf("\nCN1=input, CN2=output, Output=%x",i);
  outportb(wBase+0xd8,i);
  delay(1000);          /* delay for D/O settle time          */

  /* read test pattern to CN1 of PIO_D48 */
  j1=inportb(wBase+0xc0);
  j2=inportb(wBase+0xc4);
  j3=inportb(wBase+0xc8);
  printf(" , DI=[%2x,%2x,%2x]",j1,j2,j3);

  delay(1000);

  if (kbhit() !=0)
  {
    c=getch(); if ((c=='q') || (c=='Q')) break;
  }

  i=i<<1;          /* next bit */
  if (i>0x80)
  {
    i=1;
    c=getch();
    if ((c=='Q') || (c=='q')) return;
    printf("\n----- Test2: D/I/O CN1-CN2 -----");
  }
}
PIO_DriverClose();
}

```

DEMO4

```
/* ----- */
/* demo 4 : INT_CHAN_3, timer interrupt, demo */
/* (it is designed to be a machine independent timer) */
/* 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;
WORD pio_d48_c0(char cConfig, char cLow, char cHigh);
WORD pio_d48_c1(char cConfig, char cLow, char cHigh);
WORD pio_d48_c2(char cConfig, char cLow, char cHigh);

static void interrupt irq_service();
int COUNT,irqmask,now_int_state;

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

/* step 1: find address-mapping of PIO/PISO cards */
.
.
/* step 2: enable all D/I/O port */
.
.
init_high(); /* interrupt initialize, INIT_CHAN_3 is HIGH now */
COUNT=0;
printf("\n*** show the count of Low_pulse ***\n");

for (;;)
{
printf("\nLow Pulse Count=%d (one low pulse every second)",
COUNT);
if (kbhit()!=0) {getch(); break;}
}

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

```

/* ----- */
/* Use INT_CHAN_3 as external interrupt signal */

WORD init_high()
{
DWORD dwVal;

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 */
outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
irqmask=inportb(A2_8259+1);
outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
setvect(wIrq-8+0x70, irq_service);
}

/* select CLK1=32768 Hz */
outportb(wBase+0xf0,0x15); /* CTRL_D0=1-->timer clock=32768 Hz */

/* program OUT1 to 2 Hz */
pio_d48_c1(0x76,0,0x40); /* COUNTER1,MODE-3,32768/0x4000=2 Hz */

/* program OUT2 to 1 Hz */
/* note: the 8254 need extra 2-clock for initialization */
pio_d48_c2(0xb6,2,0); /* COUNTER2,MODE-3,2/2=1Hz */

for (;;) /* wait COUT2=HIGH */
{
if ((inportb(wBase+7)&8)!=0) break;
}

/* Note: now the COUT2 is HIGH */
/* --> INV3 must select the inverted COUT2 */
/* --> INT_CHAN_3=!COUT2=init_LOW, active_HIGH */

outportb(wBase+0x2a,0); /* select the inverted COUT2 */
/* INT_CHAN_3=!COUT2 */
now_int_state=1; /* now COUT2 is high */
outportb(wBase+5,8); /* enable INT_CHAN_3 interrupt */
enable();
}

```

```

void interrupt irq_service()
{
if (now_int_state==1)          /* now COUT2 is changed to LOW      */
{
COUNT++;                    /* --> INT_CHAN_3=!COUT2=HIGH now */
if((inportb(wBase+7)&8)==0)   /* find a LOW_pulse (COUT2)      */
{
outportb(wBase+0x2a,8); /* INV3 select the non-inverted input*/
/* INT_CHAN_3=COUT2=LOW --> */
/* INT_CHAN_3 generate a high_pulse */
now_int_state=0;          /* now COUT2=LOW */
}
else now_int_state=1;      /* now COUT2=HIGH */
/* (no need to generate high_pulse) */
}
else                          /* now COUT2 is changed to HIGH */
{
/* --> INT_CHAN_3=COUT2=HIGH now */
/* find a HIGH_pulse (COUT2) */
if((inportb(wBase+7)&8)==8)   /* the COUT2 is still fixed in HIGH */
{
outportb(wBase+0x2a,0); /* INV3 select the inverted input */
/* INT_CHAN_3=!COUT2=LOW --> */
/* INT_CHAN_3 generate a high_pulse */
now_int_state=1;          /* now COUT2=HIGH */
}
else now_int_state=0;      /* now COUT2=LOW */
/* (no need to generate high_pulse) */
}

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

/* ----- */

WORD pio_d48_c0(char cConfig, char cLow, char cHigh)/*COUNTER_0 */
{
outportb(wBase+0xec,cConfig);
outportb(wBase+0xe0,cLow);
outportb(wBase+0xe0,cHigh);
return(NoError);
}

WORD pio_d48_c1(char cConfig, char cLow, char cHigh)/*COUNTER_1 */
{
outportb(wBase+0xec,cConfig);
outportb(wBase+0xe4,cLow);
outportb(wBase+0xe4,cHigh);
return(NoError);
}

WORD pio_d48_c2(char cConfig, char cLow, char cHigh)/*COUNTER_2 */
{
outportb(wBase+0xec,cConfig);
outportb(wBase+0xe8,cLow);
outportb(wBase+0xe8,cHigh);
return(NoError);
}

```

```

/* ----- */
/* demo 5:INT_CHAN_2, 16-bit event counter (no interrupt) */
/* step 1:apply a init_HIGH & active_LOW signal to PC0 of port-2*/
/* step 2:run demo5.exe */
/* ----- */
#include "PIO.H"

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

WORD init_high();
WORD wBase,wIrq;

WORD pio_d48_c0(char cConfig, char cLow, char cHigh);
WORD pio_d48_c1(char cConfig, char cLow, char cHigh);
WORD pio_d48_c2(char cConfig, char cLow, char cHigh);

int main()
{
    int i,j;
    WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
    WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
    char c;
    DWORD dwVal;
    unsigned int high,low,count;

    /* step 1: find address-mapping of PIO/PISO cards */
        .
        .
    /* step 2: enable all D/I/O port */
        .
        .
    /* step 3: select PC0 of port2 as init_HIGH & active_LOW sinal */
    outportb(wBase+0xf0,0); /* CTRL_D1=0 -> init_HIGH, active_LOW */

    /* step 4: latch&read COUNTER-0 to compute low-pulse-count */
    printf("\n*** 16-bit event down counter ***\n");

    /* NOTE : The 8254 need the extra starting two event_clock to */
    /*         initialize So the counter value before the starting */
    /*         two clock is error */
    pio_d48_c0(0x30,0xff,0xff);/* COUNTER0,mode-0 down count 0xffff */
    for (;;)
    {
        outportb(wBase+0xec,0x00); /* latch counter_0 */
        low=inportb(wBase+0xe0); /* read low-count */
        high=inportb(wBase+0xe0); /* read high-count */
        count=(0xff-high)*256+(0xff-low)+2; /* add the starting */
                                           /* two clock */
        printf("\nhigh=%x,low=%x,LOW_pulse_count=%u",high,low,count);
        if (kbhit()!=0) {getch(); break;}
    }

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

```

```

/* ----- */
/* demo 6:INT_CHAN_2, 16-bit event counter (no interrupt) */
/* step 1:apply a init_LOW & active_HIGH signal to PC0 of port-2*/
/* step 2:run demo6.exe */
/* ----- */
#include "PIO.H"

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

WORD init_high();
WORD wBase,wIrq;
WORD pio_d48_c0(char cConfig, char cLow, char cHigh);
WORD pio_d48_c1(char cConfig, char cLow, char cHigh);
WORD pio_d48_c2(char cConfig, char cLow, char cHigh);

int main()
{
int i,j;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;
DWORD dwVal;
unsigned int high,low,count;

/* step 1: find address-mapping of PIO/PISO cards */
.
.
/* step 2: enable all D/I/O port */
.
.
/* step 3: select PC0 of port_2 as init_LOW & active_HIGH signal */
outportb(wBase+0xf0,2); /* CTRL_D1=1 --> init_LOW, active_HIGH */

/* step 4: latch&read COUNTER=0 to compute HIGH_pulse_count */
printf("\n*** 16-bit event down counter ***\n");

/* NOTE : The 8254 need the extra starting two event_clock to */
/* initialize So the counter value before the starting */
/* two clock is error */
pio_d48_c0(0x30,0xff,0xff);/* COUNTER0,mode=0 down count 0xffff */

for (;;)
{
outportb(wBase+0xec,0x00); /* latch counter_0 */
low=inportb(wBase+0xe0); /* read low-count */
high=inportb(wBase+0xe0); /* read high-count */
count=(0xff-high)*256+(0xff-low)+2; /* add the starting two */
/* clock */
printf("\nhigh=%x,low=%x,HIGH_pulse_count=%u",high,low,count);
if (kbhit()!=0) {getch(); break;}
}

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

DEMO7

/* ----- */
/* demo 7:INT_CHAN_2, 16-bit down-counter (using interrupt) */
/* step 1:apply a init_HIGH & active_LOW signal to PC0 of port-2*/

```

```

/* step 2:run demo7.exe */
/* ----- */
#include "PIO.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOĪ 0x20

WORD init_low();
WORD wBase,wIrq;

WORD pio_d48_c0(char cConfig, char cLow, char cHigh);
WORD pio_d48_c1(char cConfig, char cLow, char cHigh);
WORD pio_d48_c2(char cConfig, char cLow, char cHigh);

static void interrupt irq_service();
int COUNT,irqmask,now_int_state;

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

/* step 1: find address-mapping of PIO/PISO cards */
.
.
/* step 2: enable all D/I/O port */
.
.
init_low(); /*interrupt initialize,INT_CHAN_2 now is initial LOW*/
COUNT=0;
printf("\n*** show the count of Low_pulse **\n");

for (;;)
{
printf("\nInterrupt Count=%d (one interrupt=5 low_pulse)"
,COUNT);
if (kbhit()!=0) {getch(); break;}
}

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

```

```

/* Use INT_CHAN_2 as external interrupt signal */
WORD init_low()
{
    DWORD dwVal;

    disable();

    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 */
        outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
        irqmask=inportb(A2_8259+1);
        outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
        setvect(wIrq-8+0x70, irq_service);
    }

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

    /* select PC0 of port_2 as init_HIGH & active_LOW signal */
    outportb(wBase+0xf0,0); /* CTRL_D1=0 -> init_HIGH, active_LOW */

    /* NOTE : The 8254 need the extra starting two event_clock to */
    /* initialize So the counter value before the starting */
    /* two clock is error */

    pio_d48_c0(0x30,3,0); /* COUNTER0,mode-0 down count 3+2=5 */

    /* Note: now the COUT0 is LOW */
    /* --> INV2 must select the non-inverted COUT0 */
    /* --> INT_CHAN_2=COUT0=init_LOW, active_HIGH */

    outportb(wBase+0x2a,4); /* select the non-inverted COUT0 */
    /* INT_CHAN_2=COUT0=LOW now */
    now_int_state=0; /* now COUT0 is LOW */
    outportb(wBase+5,4); /* enable INT_CHAN_2 interrupt */
    enable();
}

/* ----- */

void interrupt irq_service()
{
    if (now_int_state==0) /* now COUT0 is changed to HIGH */
    { /* --> INT_CHAN_2=COUT0=HIGH now */
        COUNT++; /* find a HIGH_pulse (COUT0) */
        pio_d48_c0(0x30,3,0); /* COUNTER0,mode-0 down count 3+2=5 */
        /* now INT_CHAN_2=COUT0=LOW now --> */
        /* INT_CHAN_2 generate a HIGH_pulse */
        now_int_state=0; /* now COUT0=LOW */
    }

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

```

DEMO8

```

/* ----- */
/* demo 8:INT_CHAN_0, interrupt demo */
/* step 1:apply a init_HIGH & active_LOW signal to PC3 of port-2*/
/* (note: PC7 of port-2 is don't care */
/* step 2: run demo8.exe */
/* ----- */
#include "PIO.H"

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

WORD init_high();
WORD wBase,wIrq;
WORD pio_d48_c0(char cConfig, char cLow, char cHigh);
WORD pio_d48_c1(char cConfig, char cLow, char cHigh);
WORD pio_d48_c2(char cConfig, char cLow, char cHigh);
static void interrupt irq_service();
int COUNT,irqmask,now_int_state;

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

/* step 1: find address-mapping of PIO/PISO cards */
/* step 2: enable all D/I/O port */

init_high(); /* interrupt initialize, INIT_CHAN_0 is HIGH now */
COUNT=0;
printf("\n*** show the count of Low_pulse **\n");
for (;;)
{
printf("\nLow Pulse Count=%d (initial is HIGH, active is LOW)"
,COUNT);
if (kbhit()!=0) {getch(); break;}
}

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

/* Use INT_CHAN_0 as external interrupt signal */
WORD init_high()
{
DWORD dwVal;
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 */
outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
}
}

```



```

    irqmask=inportb(A2_8259+1);
    outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
    setvect(wIrq-8+0x70, irq_service);
}
/* select PC3 of port_2 as interrupt signal */
outportb(wBase+0xf0,0x18); /* CTRL_D3=1,CTRL_D2=0 --> */
/* INT_CHAN_0=PC3 of port_2 */
for (;;) /* wait PC3=HIGH */
{
    if ((inportb(wBase+7)&1)!=0) break;
    printf("\nWait PC3=HIGH");
}
/* Note: now the PC3 of port-2 is HIGH */
/* --> INV0 must select the inverted PC3 */
/* --> INT_CHAN_0=!PC3=init_LOW, active_HIGH */
outportb(wBase+0x2a,0); /* select the inverted PC3 */
/* INT_CHAN_0=!PC3 */
now_int_state=1; /* now PC3 is high */
outportb(wBase+5,1); /* enable INT_CHAN_0 interrupt */
enable();
}
void interrupt irq_service()
{
    if (now_int_state==1) /* now PC3 is changed to LOW */
    /*
    {
        /* --> INT_CHAN_3=!PC3=HIGH now */
        COUNT++; /* find a LOW_pulse (PC3) */
    /*
    if((inportb(wBase+7)&1)==0) /* the PC3 is still fixed in LOW */
    {
        /* --> need to generate a high_pulse */
        outportb(wBase+0x2a,1); /* INV0 select the non-inverted input */
        /* INT_CHAN_0=PC3=LOW --> */
    /*
        /* INT_CHAN_0 generate a high_pulse */
        now_int_state=0; /* now PC3=LOW */
    /*
    }
    else now_int_state=1; /* now PC3=HIGH */
    /* (no need to generate high_pulse ) */
}
else /* now PC3 is changed to HIGH */
{
    /* --> INT_CHAN_0=PC3=HIGH now */
    /* find a HIGH_pulse (PC3) */
}
/*
if((inportb(wBase+7)&1)==1) /* the PC3 is still fixed in HIGH */
{
    /* --> need to generate a high_pulse */
    outportb(wBase+0x2a,0); /* INV0 select the inverted input */
    /* INT_CHAN_0=!PC3=LOW --> */
    /* INT_CHAN_0 generate a high_pulse */
    now_int_state=1; /* now PC3=HIGH */
}
else now_int_state=0; /* now PC3=LOW */
/* (no need to generate high_pulse ) */
}
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}

```

```

/*-----*/
/*demo 9:INT_CHAN_0, interrupt demo */
/*step 1:apply a init_HIGH & active_LOW signal to PC3 of port-2 */
/* note: The PC7 of port_2 is used to enable the interrupt*/
/* operation, if PC7=GND --> PC3_interrupt is enable */
/* if PC7=VCC --> PC3_interrupt is disable */
/*----- or -----*/
/*step 1:apply a init_LOW & active_HIGH signal to PC7 of port-2 */
/* note: The PC3 of port_2 is used to enable the interrupt*/
/* operation, if PC3=VCC --> PC7_interrupt is enable */
/* if PC3=GND --> PC7_interrupt is disable) */
/*-----*/
/*step 2:run demo9.exe */
/*-----*/
#include "PIO.H"

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

WORD init_high();
WORD wBase,wIrq;
WORD pio_d48_c0(char cConfig, char cLow, char cHigh);
WORD pio_d48_c1(char cConfig, char cLow, char cHigh);
WORD pio_d48_c2(char cConfig, char cLow, char cHigh);

static void interrupt irq_service();
int COUNT,irqmask,now_int_state;

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

/* step 1: find address-mapping of PIO/PISO cards */
.
.
/* step 2: enable all D/I/O port */
.
.
init_high(); /* interrupt initialize, INIT_CHAN_0 is HIGH now */
COUNT=0;
printf("\n*** show the count of Low_pulse **\n");

for (;;)
{
printf("\nLow Pulse Count=%d (initial is HIGH, active is LOW)"
,COUNT);
if (kbhit()!=0) {getch(); break;}
}

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

```

```

/* Use INT_CHAN_0 as external interrupt signal */
WORD init_high()
{
    DWORD dwVal;

    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 */
        outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
        irqmask=inportb(A2_8259+1);
        outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
        setvect(wIrq-8+0x70, irq_service);
    }

    /* select PC3 & PC7 of port_2 as interrupt signal */
    outportb(wBase+0xf0,0x10); /* CTRL_D3=0,CTRL_D2=0 --> */
    /* INT_CHAN_0=PC3&!PC7 -> */
    /* PC7 can enable/disable PC3 */
    /* PC3 can enable/disable PC7 */

    /* Note: In this demo, assume the PC3 is init_HIGH & active_LOW */
    /* the PC7 is used to enable/disable PC3 --> */
    /* PC7=VCC --> PC3-interrupt is disable */
    /* PC7=GND --> PC3-interrupt is enable */
    /* ----- */
    /* now the PC3 of port-2 is HIGH */
    /* --> INV0 must select the inverted PC3 */
    /* --> INT_CHAN_0=!PC3=init_LOW, active_HIGH */

    outportb(wBase+0x2a,0); /* select the inverted PC3 */
    /* INT_CHAN_0=!PC3 */
    now_int_state=1; /* now PC3 is high */
    outportb(wBase+5,1); /* enable INT_CHAN_0 interrupt */

    for (;;) /* wait PC3&!PC7=HIGH */
    {
        if ((inportb(wBase+7)&1)!=0) break;
        printf("\nWait PC3&!PC7=HIGH");
    }

    enable();
}

```

```

void interrupt irq_service()
{
if (now_int_state==1)          /* now PC3   is changed to LOW
*/
{
COUNT++;                      /* --> INT_CHAN_3=!PC3=HIGH now      */
/* find a LOW_pulse (PC3)
*/
if((inportb(wBase+7)&1)==0)    /* the PC3 is still fixed in LOW      */
{
outportb(wBase+0x2a,1); /* INVO select the non-inverted input*/
/* INT_CHAN_0=PC3=LOW -->
*/
/* INT_CHAN_0 generate a high_pulse */
now_int_state=0;          /* now PC3=LOW
*/
}
else now_int_state=1;      /* now PC3=HIGH
*/
/* (no need to generate high_pulse ) */
}
else
{
/* now PC3 is changed to HIGH
*/
/* --> INT_CHAN_0=PC3=HIGH now
*/
/* find a HIGH_pulse (PC3)
*/
if((inportb(wBase+7)&1)==1)    /* the PC3 is still fixed in HIGH      */
{
outportb(wBase+0x2a,0); /* INVO select the inverted input
*/
/* INT_CHAN_0=!PC3=LOW -->
*/
/* INT_CHAN_0 generate a high_pulse */
now_int_state=1;          /* now PC3=HIGH
*/
}
else now_int_state=0;      /* now PC3=LOW
*/
/* (no need to generate high_pulse ) */
}
}
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}

/* ----- */

WORD pio_d48_c0(char cConfig, char cLow, char cHigh)/*COUNTER_0 */
{
outportb(wBase+0xec,cConfig);
outportb(wBase+0xe0,cLow);
outportb(wBase+0xe0,cHigh);
return(NoError);
}

WORD pio_d48_c1(char cConfig, char cLow, char cHigh)/*COUNTER_1 */
{
outportb(wBase+0xec,cConfig);
outportb(wBase+0xe4,cLow);
outportb(wBase+0xe4,cHigh);
return(NoError);
}

WORD pio_d48_c2(char cConfig, char cLow, char cHigh)/*COUNTER_2 */
{
outportb(wBase+0xec,cConfig);
outportb(wBase+0xe8,cLow);
outportb(wBase+0xe8,cHigh);
return(NoError);
}

```

```

/*-----*/
/*demo 10:INT_CHAN_1, interrupt demo */
/*step 1 :apply a init_HIGH & active_LOW signal to PC3 of port-5*/
/*      note:The PC7 of port_5 is used to enable the interrupt*/
/*      operation, if PC7=GND --> PC3_interrupt is enable */
/*      if PC7=VCC --> PC3_interrupt is disable) */
/*step 2 :run demo10.exe */
/*-----*/
#include "PIO.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOĪ    0x20

WORD init_high();
WORD wBase,wIrq;
WORD pio_d48_c0(char cConfig, char cLow, char cHigh);
WORD pio_d48_c1(char cConfig, char cLow, char cHigh);
WORD pio_d48_c2(char cConfig, char cLow, char cHigh);

static void interrupt irq_service();
int COUNT,irqmask,now_int_state;

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

/* step 1: find address-mapping of PIO/PISO cards */
.
.
/* step 2: enable all D/I/O port */
.
.
init_high(); /* interrupt initialize, INIT_CHAN_1 is HIGH now */
COUNT=0;
printf("\n*** show the count of Low_pulse **\n");

for (;;)
{
printf("\nLow Pulse Count=%d (initial is HIGH, active is LOW)"
,COUNT);
if (kbhit()!=0) {getch(); break;}
}

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

```

```

/* Use INT_CHAN_1 as external interrupt signal */

WORD init_high()
{
DWORD dwVal;

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 */
outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
irqmask=inportb(A2_8259+1);
outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
setvect(wIrq-8+0x70, irq_service);
}

/* select PC3 & !PC7 of port 5 as interrupt signal */
outportb(wBase+0xf0,0x04); /* CTRL_D5=0,CTRL_D4=0 --> */
/* INT_CHAN_1=PC3&!PC7 -> */
/* PC7 can enable/disable PC3 */

/* Note: In this demo, assume the PC3 is init HIGH & active_LOW */
/* the PC7 is used to enable/disable PC3 --> */
/* PC7=VCC --> PC3-interrupt is disable */
/* PC7=GND --> PC3-interrupt is enable */
/* ----- */
/* now the PC3 of port-5 is HIGH */
/* --> INV1 must select the inverted PC3 */
/* --> INT_CHAN_1=!PC3=init_LOW, active_HIGH */

outportb(wBase+0x2a,0); /* select the inverted PC3 */
/* INT_CHAN_1=!PC3 */
now_int_state=1; /* now PC3 is high */
outportb(wBase+5,2); /* enable INT_CHAN_1 interrupt */

for (;;) /* wait PC3&!PC7=HIGH */
{
if ((inportb(wBase+7)&2)!=0) break;
printf("\nWait PC3&!PC7=HIGH");
}

enable();
}

```

```

void interrupt irq_service()
{
if (now_int_state==1)          /* now PC3 is changed to LOW
*/
{
COUNT++;                    /* --> INT_CHAN_1=!PC3=HIGH now */
/* find a LOW_pulse (PC3)
*/
if((inportb(wBase+7)&2)==0) /* the PC3 is still fixed in LOW */
{
/* --> need to generate a high_pulse */
outportb(wBase+0x2a,2); /* INV1 select the non-inverted input*/
/* INT_CHAN_1=PC3=LOW --> */
/* INT_CHAN_1 generate a high_pulse */
now_int_state=0;          /* now PC3=LOW
*/
}
else now_int_state=1;      /* now PC3=HIGH */
/* (no need to generate high_pulse ) */
}
else
/* now PC3 is changed to HIGH */
{
/* --> INT_CHAN_1=PC3=HIGH now */
/* find a HIGH_pulse (PC3) */
if((inportb(wBase+7)&2)==2) /* the PC3 is still fixed in LOW */
{
/* --> need to generate a high_pulse */
outportb(wBase+0x2a,0); /* INV1 select the inverted input */
/* INT_CHAN_1=!PC3=LOW --> */
/* INT_CHAN_1 generate a high_pulse */
now_int_state=1;          /* now PC3=HIGH */
}
else now_int_state=0;      /* now PC3=HIGH */
/* (no need to generate high_pulse ) */
}
}

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

/* ----- */

WORD pio_d48_c0(char cConfig, char cLow, char cHigh)/*COUNTER_0 */
{
outportb(wBase+0xec,cConfig);
outportb(wBase+0xe0,cLow);
outportb(wBase+0xe0,cHigh);
return(NoError);
}

WORD pio_d48_c1(char cConfig, char cLow, char cHigh)/*COUNTER_1 */
{
outportb(wBase+0xec,cConfig);
outportb(wBase+0xe4,cLow);
outportb(wBase+0xe4,cHigh);
return(NoError);
}

WORD pio_d48_c2(char cConfig, char cLow, char cHigh)/*COUNTER_2 */
{
outportb(wBase+0xec,cConfig);
outportb(wBase+0xe8,cLow);
outportb(wBase+0xe8,cHigh);
return(NoError);
}

```

```

DEMO11

/* ----- */
/* demo 11 : INT_CHAN_0 & INT_CHAN_1 interrupt demo */
/* step 1 : apply a init_HIGH & active_LOW signal to PC3 of port-2 */
/*          (note: PC7 of port-2 is don't care) */
/* step 2 : apply a init_HIGH & active_LOW signal to PC3 of port-5 */
/*          (note: PC7 of port-5 is don't care) */
/* step 3 : run demoll.exe */
/* ----- */

#include "PIO.H"

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

WORD init_high();
WORD wBase, wIrq;

WORD pio_d48_c0(char cConfig, char cLow, char cHigh);
WORD pio_d48_c1(char cConfig, char cLow, char cHigh);
WORD pio_d48_c2(char cConfig, char cLow, char cHigh);

static void interrupt irq_service();
int irqmask, now_int_state, invert, new_int_state, int_c;
int INT0_H, INT0_L, INT1_H, INT1_L;

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

    /*          step 1: find address-mapping of PIO/PISO cards */
    clrscr();
    wRetVal=PIO_DriverInit(&wBoards, 0x80, 0x01, 0x30); /* for OME-PIO-D48 */
    printf("\nThrer are %d PIO_D48 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);
    }

    /* select card_0 */
    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 */

    init_high(); /* interrupt initialize, INIT_CHAN_0/1 is HIGH now */
    printf("\n*** show the count of Low_pulse **\n");
    for (;;)

```



```

    {
        printf("\nINT0=[%x,%x],INT1=[%x,%x]"
            ,INT0_L,INT0_H,INT1_L,INT1_H);
        if (kbhit()!=0) {getch(); break;}
    }

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

/* ----- */
/* Use INT_CHAN_0 & INT_CHAN_1 as external interrupt signal */

WORD init_high()
{
    DWORD dwVal;

    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 */
        outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
        irqmask=inportb(A2_8259+1);
        outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
        setvect(wIrq-8+0x70, irq_service);
    }

    /* select PC3 of port_2 & PC3 of port_5 as interrupt signal */
    outportb(wBase+0xf0,0x28);                /* CTRL_D5/4/3/2=1010 --> */
                                                /* INT_CHAN_0=PC3 of port_2 */
                                                /* INT_CHAN_1=PC3 of port_5 */

    /* Note: now both PC3 are HIGH */
    /* --> INV0, INV1 must select the inverted PC3 */
    /* --> INT_CHAN_0=!PC3 of port_2=init_LOW, active_HIGH */
    /* --> INT_CHAN_1=!PC3 of port_5=init_LOW, active_HIGH */

    invert=0;
    outportb(wBase+0x2a,invert);                /* select the inverted PC3 */
                                                /* INT_CHAN_0=!PC3 of port_2 */
                                                /* INT_CHAN_1=!PC3 of port_5 */
    now_int_state=3;                /* now both PC3 are high */
    outportb(wBase+5,3);                /* enable INT_CHAN_0 & INT_CHAN_1 */

    for (;;)                /* wait both PC3 are HIGH */
    {
        if ((inportb(wBase+7)&3)!=0) break;
        printf("\nWait PC3 of port-2 & PC3 of port-5 are HIGH");
    }

    INT0_H=INT0_L=INT1_H=INT1_L=0;
    enable();
}

/* ----- */

```

```

/* NOTE:1.The hold-time of INT_CHAN_0&INT_CHAN_1 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()
{
/* now ISR can not know which interrupt is active */

new_int_state=inportb(wBase+7)&0x03; /* read all interrupt */
/* signal states */
int_c=new_int_state ^ now_int_state; /* compare new_state to */
/* old_state */

/* INT_CHAN_0 & INT_CHAN_1 can be active at the same time */

if ((int_c&0x01) !=0 ) /* INT_CHAN_0 is active */
{
if ((new_int_state&1)==0)/* now INT_CHAN_0 is changed to LOW */
{
INT0_L++;
}
else /* now INT_CHAN_0 is changed to HIGH */
{
INT0_H++;
}

invert=invert^1; /* to generate a HIGH_pulse */
}

if ((int_c&0x02) !=0 ) /* INT_CHAN_1 is active */
{
if ((new_int_state&2)==0)/* now INT_CHAN_1 is changed to LOW */
{
INT1_L++;
}
else /* now INT_CHAN_1 is changed to HIGH */
{
INT1_H++;
}

invert=invert^2; /* to generate a HIGH_pulse */
}

now_int_state=new_int_state; /* update interrupt status */
outportb(wBase+0x2a,invert); /* generate a HIGH_pulse */

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

```

WARRANTY/DISCLAIMER

OMEGA ENGINEERING, INC. warrants this unit to be free of defects in materials and workmanship for a period of **13 months** from date of purchase. OMEGA's WARRANTY adds an additional one (1) month grace period to the normal **one (1) year product warranty** to cover handling and shipping time. This ensures that OMEGA's customers receive maximum coverage on each product.

If the unit malfunctions, it must be returned to the factory for evaluation. OMEGA's Customer Service Department will issue an Authorized Return (AR) number immediately upon phone or written request. Upon examination by OMEGA, if the unit is found to be defective, it will be repaired or replaced at no charge. OMEGA's WARRANTY does not apply to defects resulting from any action of the purchaser, including but not limited to mishandling, improper interfacing, operation outside of design limits, improper repair, or unauthorized modification. This WARRANTY is VOID if the unit shows evidence of having been tampered with or shows evidence of having been damaged as a result of excessive corrosion; or current, heat, moisture or vibration; improper specification; misapplication; misuse or other operating conditions outside of OMEGA's control. Components which wear are not warranted, including but not limited to contact points, fuses, and triacs.

OMEGA is pleased to offer suggestions on the use of its various products. However, OMEGA neither assumes responsibility for any omissions or errors nor assumes liability for any damages that result from the use of its products in accordance with information provided by OMEGA, either verbal or written. OMEGA warrants only that the parts manufactured by it will be as specified and free of defects. OMEGA MAKES NO OTHER WARRANTIES OR REPRESENTATIONS OF ANY KIND WHATSOEVER, EXPRESS OR IMPLIED, EXCEPT THAT OF TITLE, AND ALL IMPLIED WARRANTIES INCLUDING ANY WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE HEREBY DISCLAIMED. LIMITATION OF LIABILITY: The remedies of purchaser set forth herein are exclusive, and the total liability of OMEGA with respect to this order, whether based on contract, warranty, negligence, indemnification, strict liability or otherwise, shall not exceed the purchase price of the component upon which liability is based. In no event shall OMEGA be liable for consequential, incidental or special damages.

CONDITIONS: Equipment sold by OMEGA is not intended to be used, nor shall it be used: (1) as a "Basic Component" under 10 CFR 21 (NRC), used in or with any nuclear installation or activity; or (2) in medical applications or used on humans. Should any Product(s) be used in or with any nuclear installation or activity, medical application, used on humans, or misused in any way, OMEGA assumes no responsibility as set forth in our basic WARRANTY/DISCLAIMER language, and, additionally, purchaser will indemnify OMEGA and hold OMEGA harmless from any liability or damage whatsoever arising out of the use of the Product(s) in such a manner.

RETURN REQUESTS/INQUIRIES

Direct all warranty and repair requests/inquiries to the OMEGA Customer Service Department. BEFORE RETURNING ANY PRODUCT(S) TO OMEGA, PURCHASER MUST OBTAIN AN AUTHORIZED RETURN (AR) NUMBER FROM OMEGA'S CUSTOMER SERVICE DEPARTMENT (IN ORDER TO AVOID PROCESSING DELAYS). The assigned AR number should then be marked on the outside of the return package and on any correspondence.

The purchaser is responsible for shipping charges, freight, insurance and proper packaging to prevent breakage in transit.

FOR **WARRANTY** RETURNS, please have the following information available BEFORE contacting OMEGA:

1. Purchase Order number under which the product was PURCHASED,
2. Model and serial number of the product under warranty, and
3. Repair instructions and/or specific problems relative to the product.

FOR **NON-WARRANTY** REPAIRS, consult OMEGA for current repair charges. Have the following information available BEFORE contacting OMEGA:

1. Purchase Order number to cover the COST of the repair,
2. Model and serial number of the product, and
3. Repair instructions and/or specific problems relative to the product.

OMEGA's policy is to make running changes, not model changes, whenever an improvement is possible. This affords our customers the latest in technology and engineering.

OMEGA is a registered trademark of OMEGA ENGINEERING, INC.

© Copyright 2002 OMEGA ENGINEERING, INC. All rights reserved. This document may not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of OMEGA ENGINEERING, INC.

Where Do I Find Everything I Need for Process Measurement and Control? OMEGA...Of Course!

Shop online at www.omega.com

TEMPERATURE

- Thermocouple, RTD & Thermistor Probes, Connectors, Panels & Assemblies
- Wire: Thermocouple, RTD & Thermistor
- Calibrators & Ice Point References
- Recorders, Controllers & Process Monitors
- Infrared Pyrometers

PRESSURE, STRAIN AND FORCE

- Transducers & Strain Gages
- Load Cells & Pressure Gages
- Displacement Transducers
- Instrumentation & Accessories

FLOW/LEVEL

- Rotameters, Gas Mass Flowmeters & Flow Computers
- Air Velocity Indicators
- Turbine/Paddlewheel Systems
- Totalizers & Batch Controllers

pH/CONDUCTIVITY

- pH Electrodes, Testers & Accessories
- Benchtop/Laboratory Meters
- Controllers, Calibrators, Simulators & Pumps
- Industrial pH & Conductivity Equipment

DATA ACQUISITION

- Data Acquisition & Engineering Software
- Communications-Based Acquisition Systems
- Plug-in Cards for Apple, IBM & Compatibles
- Datalogging Systems
- Recorders, Printers & Plotters

HEATERS

- Heating Cable
- Cartridge & Strip Heaters
- Immersion & Band Heaters
- Flexible Heaters
- Laboratory Heaters

ENVIRONMENTAL MONITORING AND CONTROL

- Metering & Control Instrumentation
- Refractometers
- Pumps & Tubing
- Air, Soil & Water Monitors
- Industrial Water & Wastewater Treatment
- pH, Conductivity & Dissolved Oxygen Instruments