User's Guide

OME-PIO-D96
PCI-Bus Digital I/O Board
Hardware Manual

omega.com®
OMEGA®

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

OME-PIO-D96

User Manual

Table of Contents

1. Introduction

The OME-PIO-D96 provides 96 TTL digital I/O lines. The OME-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 OME-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 OME-ADP-37/PCI or OME-ADP-50/PCI adapter. The adapter can be fixed on the chassis. It can be installed in 5V PCI bus and supports "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 OME-DB-24P, OME-DB-24PR, OME-DB-24PD, OME-DB-24RD, OME-DB-24PRD, OME-DB-16P8R, OME-DB-24POR, OME-DB-24SSR, OME-DB-24C
- Each board = 4 connectors = 4×3 ports = 4×3×8 bits =96 bits
- 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/2000/XP
- 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

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

1.3.1 Options

- OME-DB-24P, OME-DB-24PD : 24 channel isolated D/I board
- OME-DB-24R, OME-DB-24RD : 24 channels relay board
- OME-DB-24PR, OME-DB-24PRD : 24 channels power relay board
- OME-DB-16P8R : 16 channels isolated D/I and 8 channels relay output board
- OME-DB-24POR : 24 channels Photo MOS output board
- OME-DB-24C : 24 channels open-collector output board
- OME-ADP-37/PCI : extender, 50 pin OPTO-22 header to DB-37 for PCI bus I/O boards
- OME-ADP-50/PCI : extender, 50 pin OPTO-22 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 follows:

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/P32A32/P64/C64/A64: D/I/O family
   OME-PISO-P8R8/P8SSR8AC/P8SSR8DC: D/I/O family
   OME-PISO-730/730A: D/I/O card
   OME-PISO-DA2: D/A card

1.5 Product Check List

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

- one piece of OME-PIO-D96 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 follows:

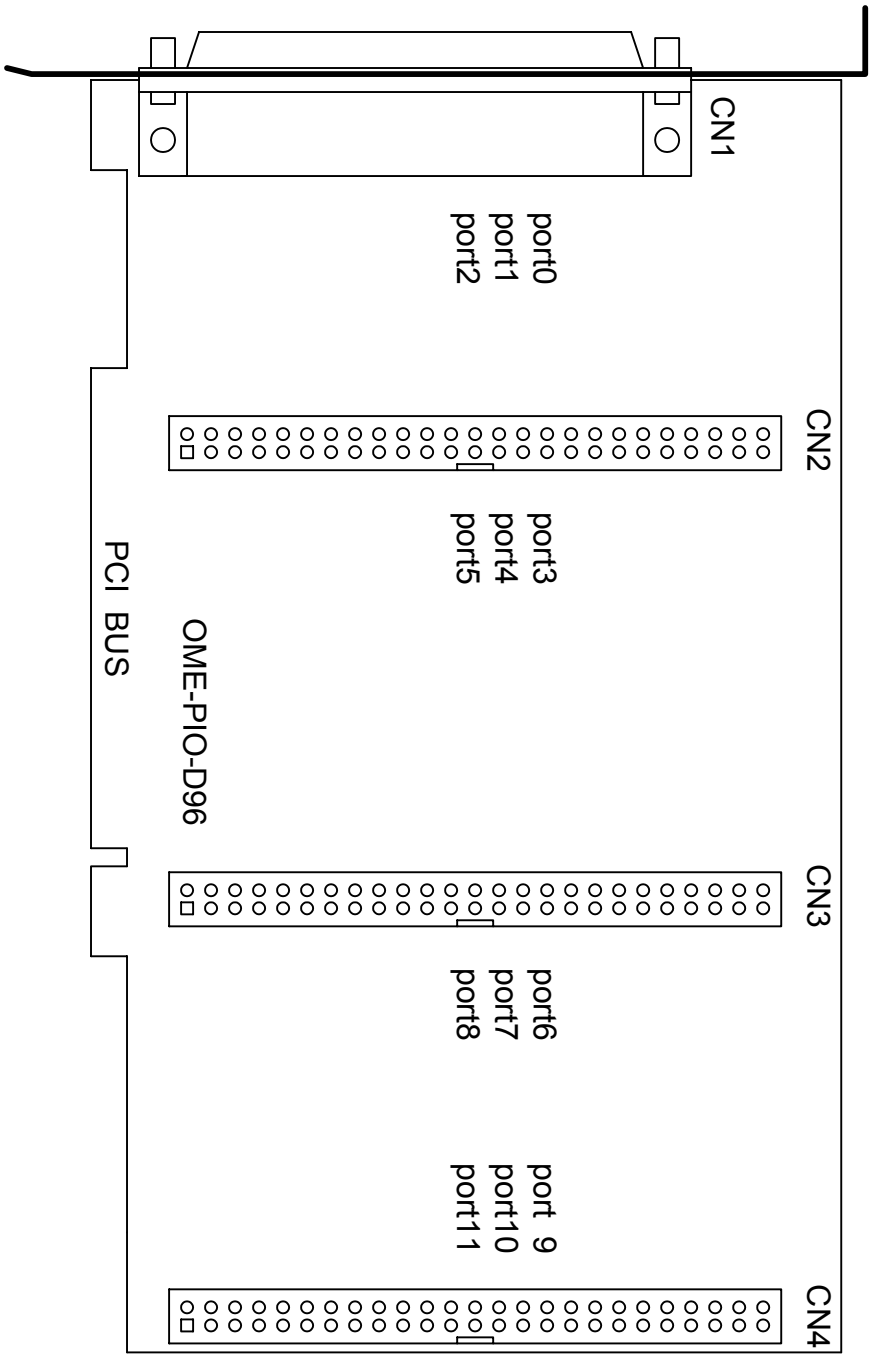
1. Where you can find the software driver & utility?
2. How to install software & utility?
3. Where is the diagnostic program?
4. FAQ

Attention !

If any of these items is missing or damaged, contact 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



2.2 I/O port Location

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

Connector of OME-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 powered up, all D/I/O ports 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 follows:

- All D/I/O operations are disabled
- 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/Os. These recommended steps are given as follows:

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

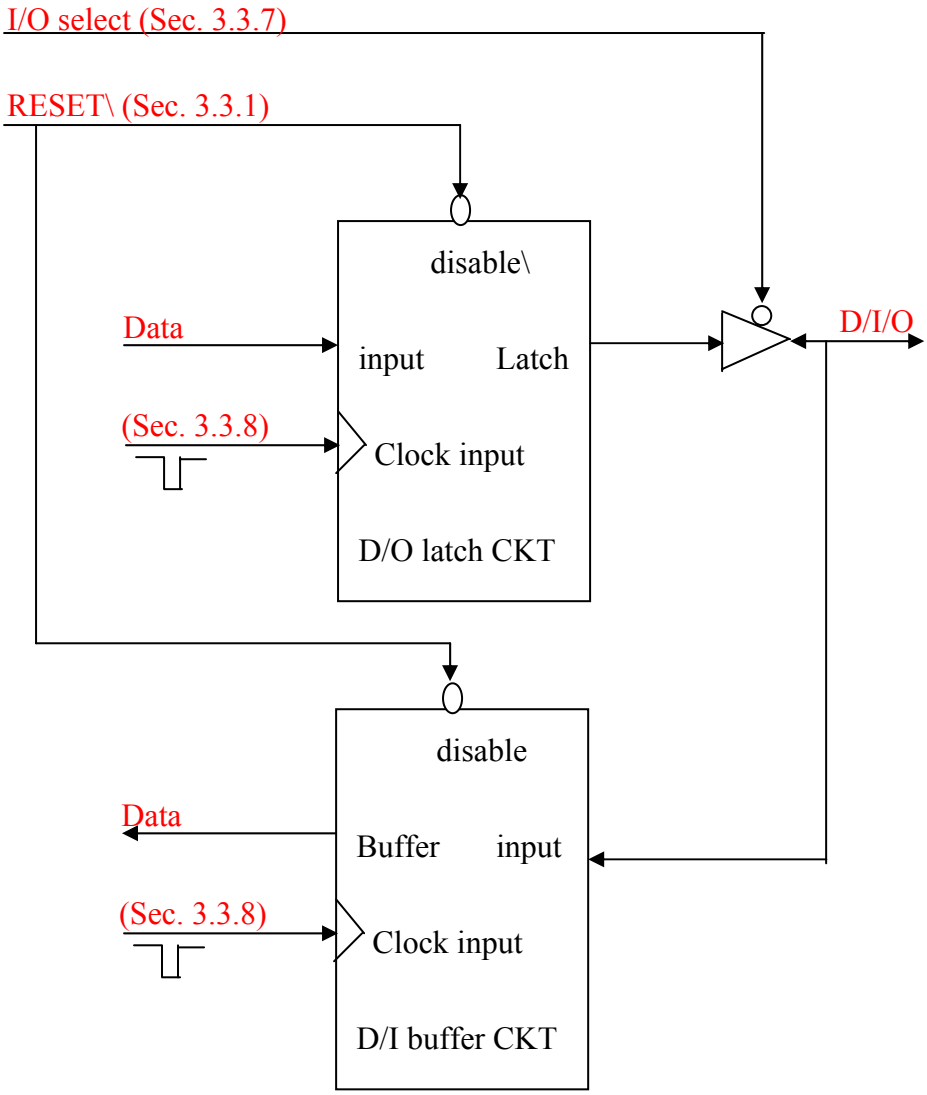
Step 2: enable all D/I/O operation (refer to Sec. 3.3.1)

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



- The RESET\ is in Low-state → all D/I/O are disabled
- The RESET\ is in High-state → all D/I/O are enabled
- 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 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 sources. Refer to Sec. 2.1 & Sec. 2.7 for P2C0/P5C0/P8C0/P11C0 location. The interrupt of OME-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.

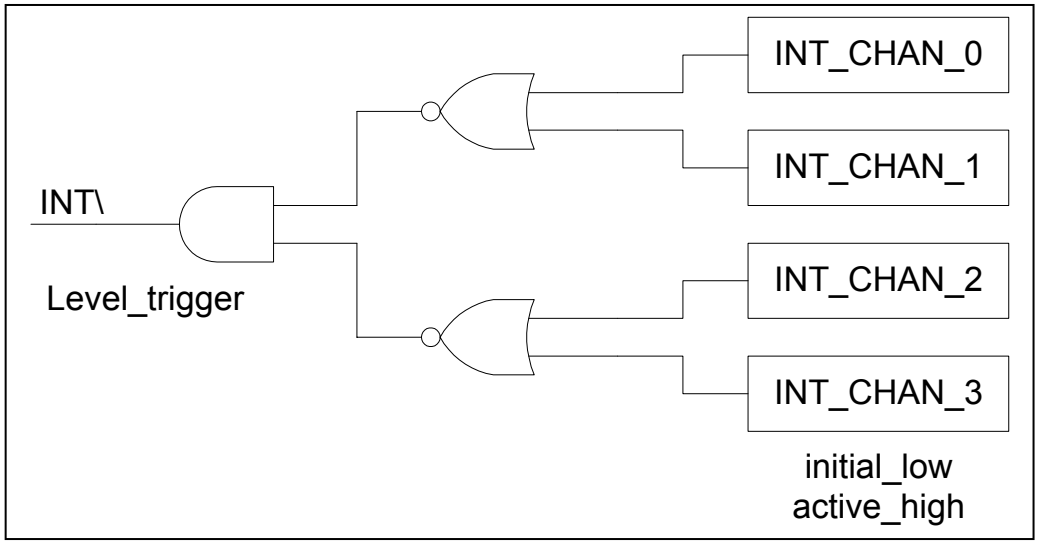
If only one interrupt signal source is used, the interrupt service routine does not have to identify the interrupt source. (Refer to DEMO3.C & DEMO4.C)

If there are more than one interrupt sources, 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

Note: 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 operating systems. The hold time can be as short as micro-second or as long as second. In general, 20ms is enough for most operating systems.

2.5.1 Interrupt Block Diagram of OME-PIO-D96



The interrupt output signal of OME-PIO-D96, INT\, is Level_trigger & Active_Low. If the INT\ generate a low-pulse, the OME-PIO-D96 will interrupt the PC only once. If the INT\ is fixed in low level, the OME-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 mare information.

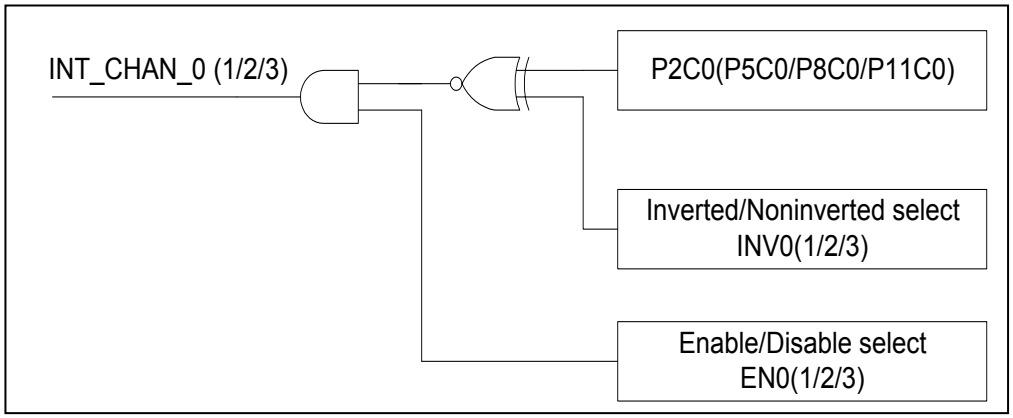
DEMO5.C → for multi-channel interrupt source

If only one interrupt source is used, the interrupt service routine doesn't have to read the status of interrupt source. The demo programs, 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)

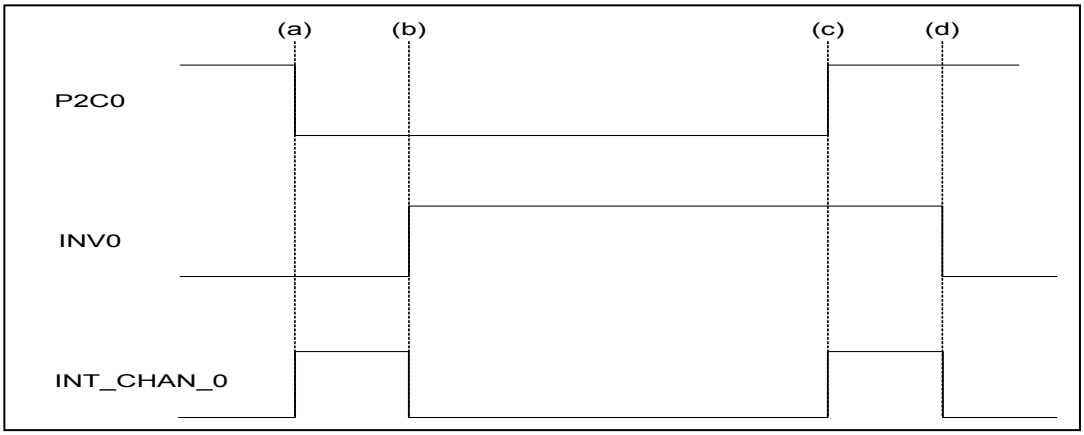
2.5.3 Initial_high, active_low Interrupt source

If the P2C0 is an initial_high, active_low signal, the interrupt service routine should use INV0 to invert or not to invert 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);
}
```



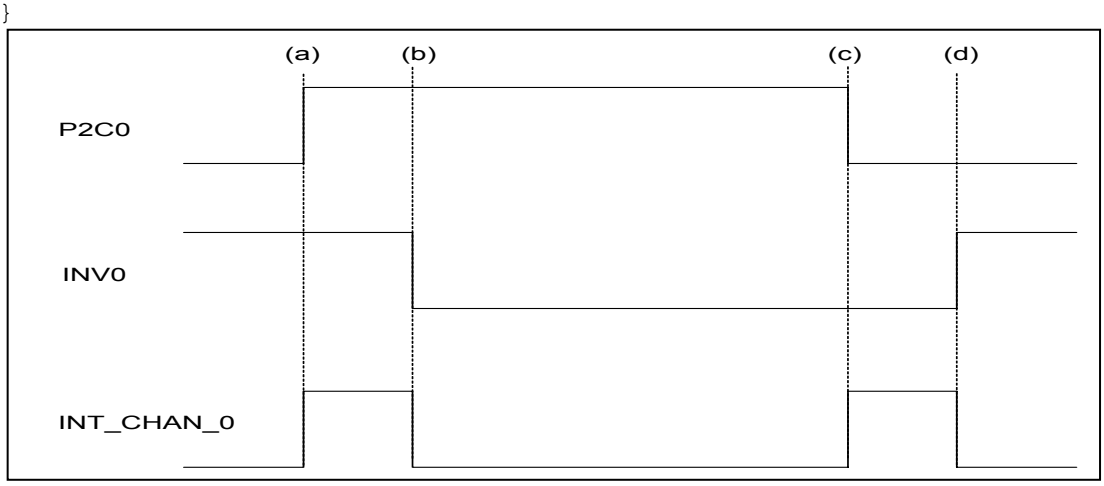
2.5.4 Initial_low, active_high Interrupt source

If the P2C0 is an initial_low, active_high signal, the interrupt service routine should use INV0 to invert or not to invert the P2C0 for high_pulse generation as follows: (Refer to DEMO3.C)

Initial set:

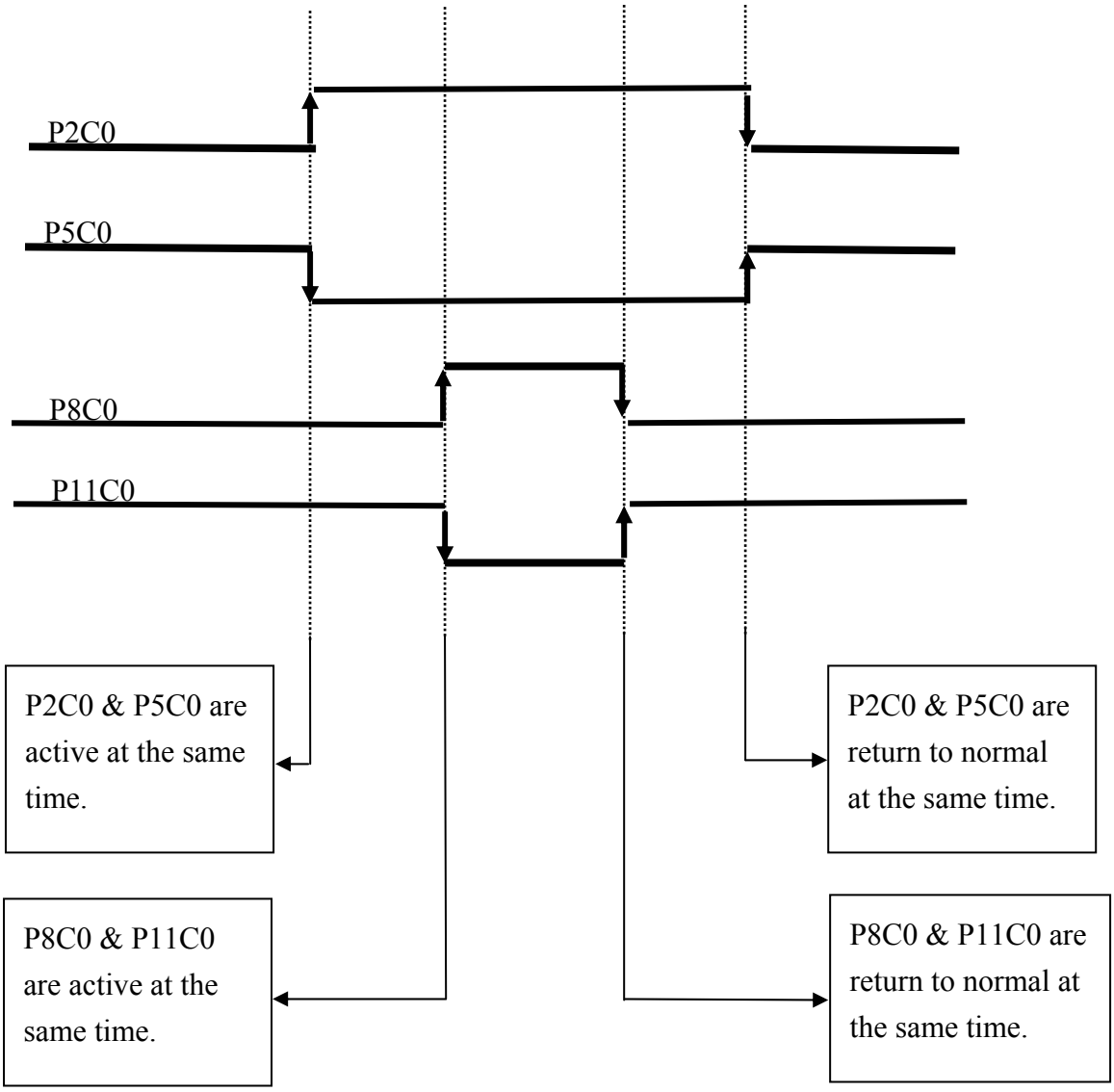
```
now_int_state=0;            /* initial state for P2C0          */
outportb(wBase+0x2a,1);  /* select the non-inverted P2C0 */
```

```
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);
}
```



2.5.5 Muliti 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 the source. **All these four falling edges & rising edges can be detected by DEMO5.C.**

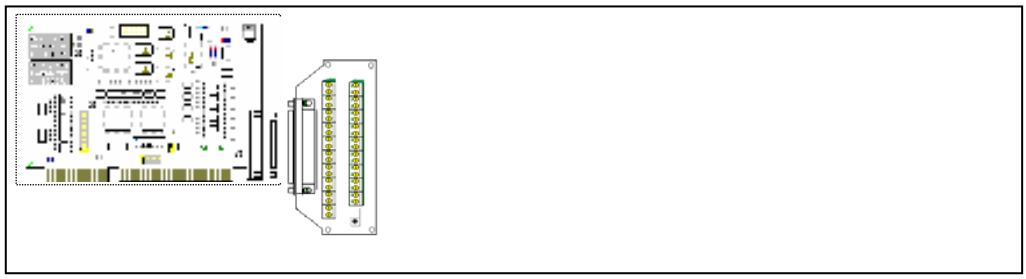
Note: When the interrupt is active, the user program has to identify the active signals. Multiple signals maybe active simultaneously. So the interrupt service routine has to service all active signals at the same time.

```
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

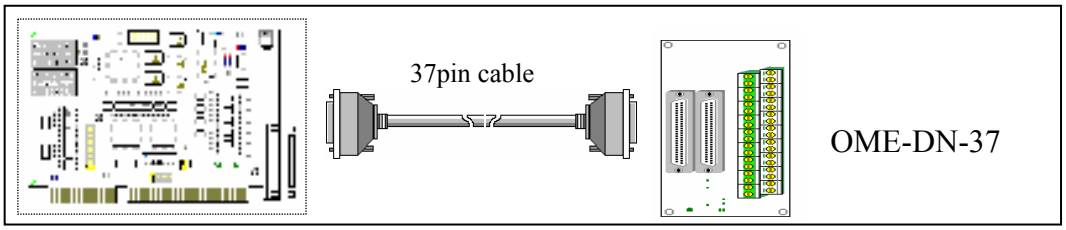
2.6.1 OME-DB-37

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



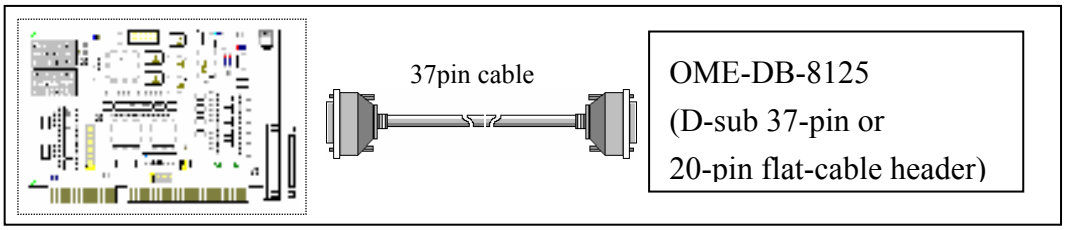
2.6.2 OME-DN-37 & OME-DN-50

The OME-DN-37 is a general purpose daughter board for D-sub 37-pin connector. The OME-DN-50 is designed for 50-pin flat-cable header. They are designed for easy wiring. Both are DIN rail mountable.



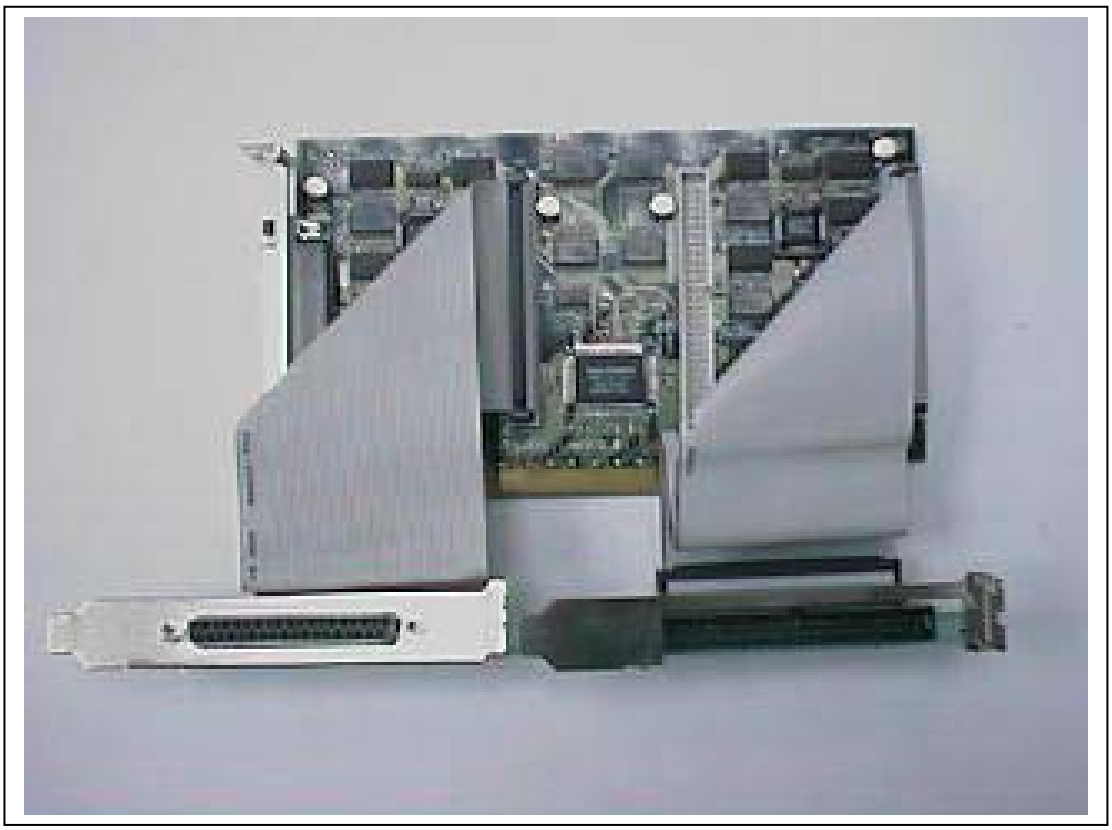
2.6.3 OME-DB-8125

The OME-DB-8125 is a general purpose screw terminal board. It is designed for easy wiring. There is one D-sub 37-pin connector & 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 extenders for 50-pin headers. 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 follows:

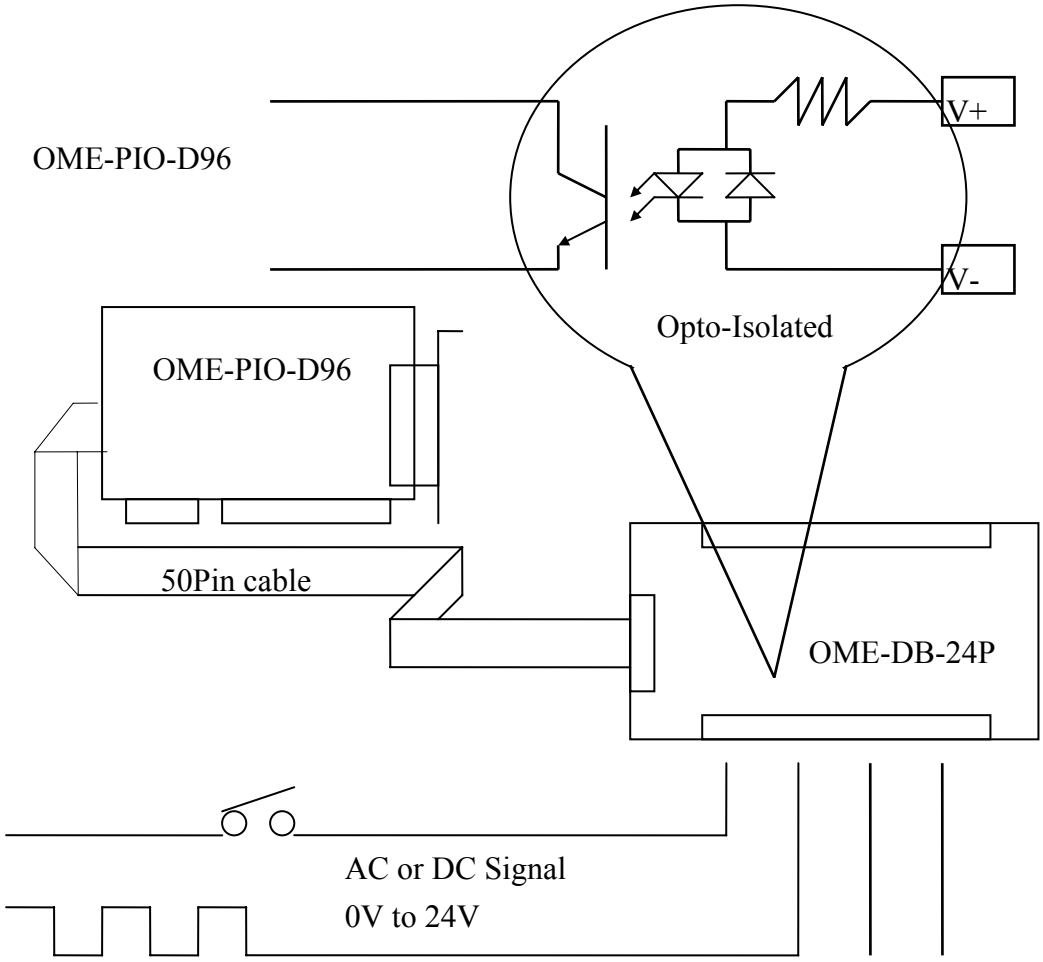


OME-ADP-37/PCI: 50-pin header to D-sub 37 extender.
OME-ADP-50/PCI: 50-pin header to 50-pin header extender.

NOTE: The user has to choose the suitable extender for their boards.

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

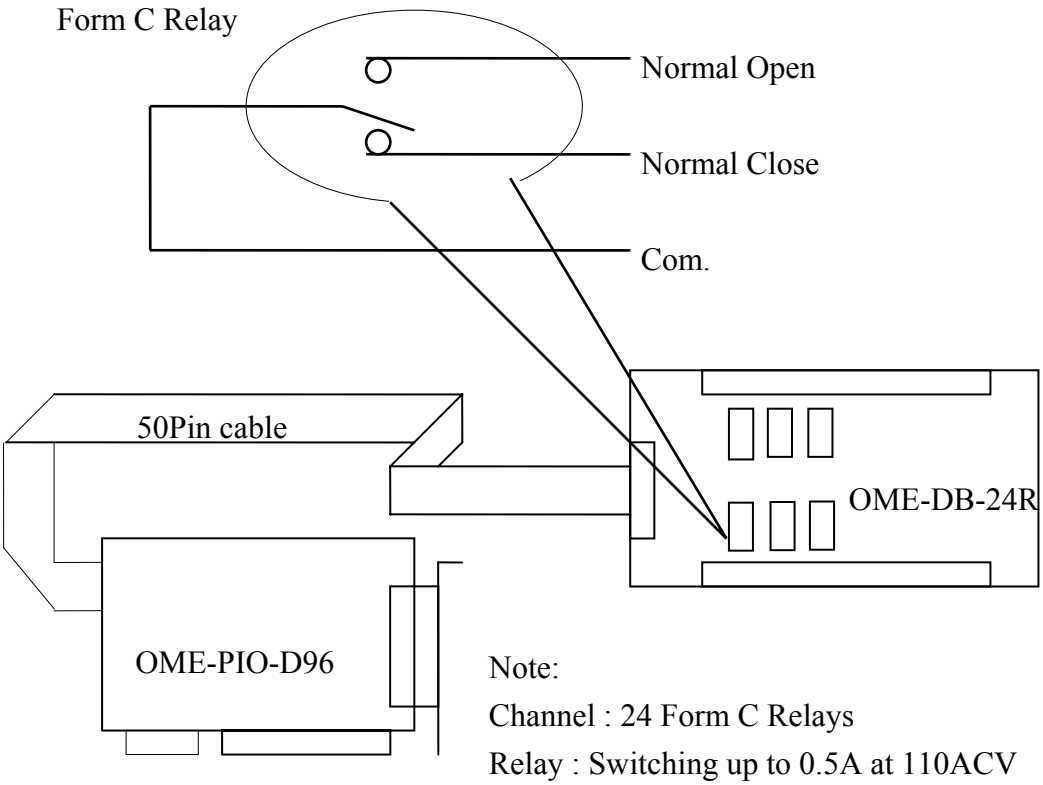
The OME-DB-24P is a 24 channel isolated digital input daughter board. The optically isolated inputs of the OME-DB-24P, consists of a bi-directional opto-coupler with a resistor for current sensing. You can use the OME-DB-24P to sense DC signal from TTL levels up to 24V or use the OME-DB-24P to sense 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 spikes 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/24RD Relay Board

The OME-DB-24R, 24 channel relay output board, consists of 24 form C relays for efficient switch of load by programmed control. The relays 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 and they light when their associated relay is activated.



Note:
Channel : 24 Form C Relays
Relay : Switching up to 0.5A at 110ACV or 1A at 24DCV

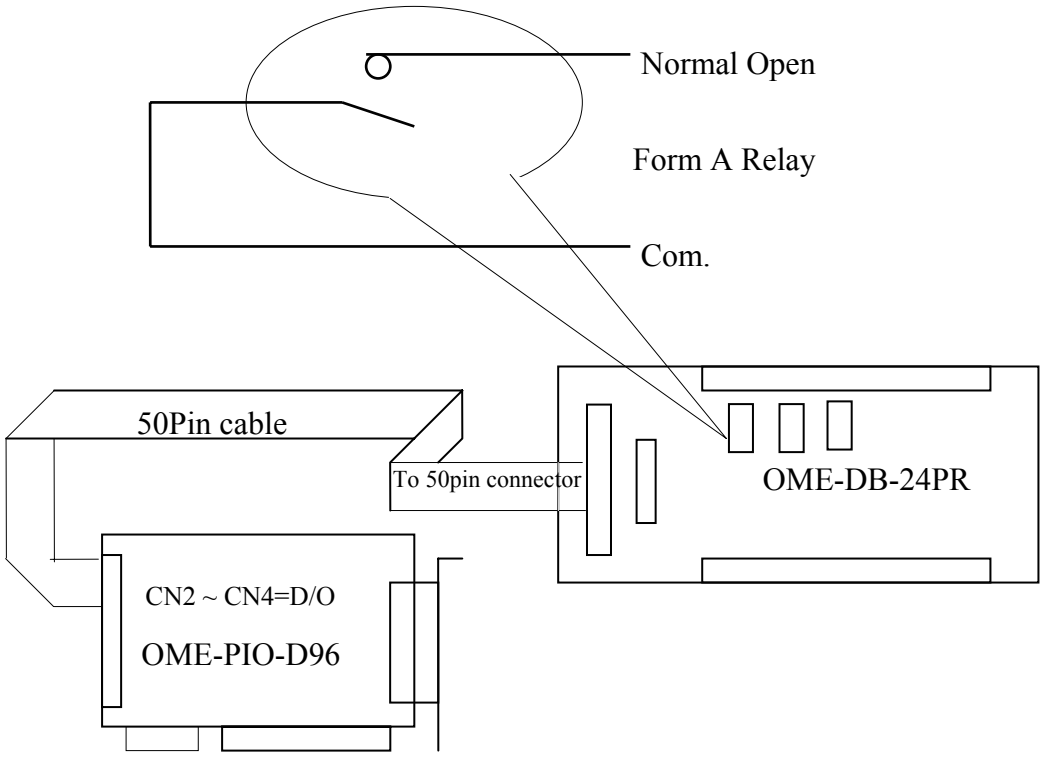
	OME-DB-24R	OME-DB-24RD
50-pin flat-cable header	Yes	Yes
D-sub 37-pin header	No	Yes
Other specifications	Same	

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 inputs

2.6.7 OME-DB-24PR/24POR/24C

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

The OME-DB-24PR, 24 channel power relay output board, consists of 8 form C and 16 form A electromechanical relays for efficient switching of load by programmed control. The contact of each relay can control a 5A load at 250VAC/30VDC. The relay is energized by applying a 5 volt signal to the appropriate relay channel on the 20-pin flat cable connector (only 16 relays) or 50-pin flat cable connector(compatible for OME-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 for OME-DIO-24, OME-DIO-48, OME-DIO-144, OME-PCI-D144, OME-PIO-D144, OME-PIO-D96, OME-PIO-D56, OME-PIO-D48, OME-PIO-D24

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

Channel : 16 Form A Relay , 8 Form C Relay

2.6.8 Daughter Boards Comparison Table

	20-pin flat-cable	50-pin flat-cable	D-sub 37-pin
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 flat-cable header in OME-PIO-D96. The OME-PIO-D96 has one D-sub37-pin 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 OME-PIO/PISO series card in the power-up stage. The IDs of OME-PIO-D96 cards are given as follows:

< REV 1.0 ~ REV 3.0 > :	< REV 4.0 or above > :
• Vendor ID = 0xE159	• Vendor ID = 0xE159
• Device ID = 0x0002	• Device ID = 0x0001
• Sub-vendor ID = 0x80	• Sub-vendor ID = 0x5880
• Sub-device ID = 0x01	• Sub-device ID = 0x01
• Sub-aux ID = 0x10	• Sub-aux ID = 0x10

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. OME-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 OME-PIO/PISO series card in the system. It is implemented based on the PCI plug & play mechanism. It will find all OME-PIO/PISO series cards installed in this system & save all their resources in the library.

Sample program 1: find all OME-PIO-D96 installed in the PC

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

/* step2: save resource of all OME-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 OME-PIO/PISO installed in the 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 follows:

OME-PIO/PISO series card	Description	Sub_vendo Old (New)	Sub_device Old (New)	Sub_AUX
OME-PIO-D144 (Rev4.0)	144 × D/I/O	80(5C80)	01	00
OME-PIO-D96 (Rev4.0)	96 × D/I/O	80(5880)	01	10
OME-PIO-D64 (Rev2.0)	64 × D/I/O	80(4080)	01	20
OME-PIO-D56 (Rev6.0)	24 × D/I/O + 16 × D/I + 16*D/O	80(C080)	01	40
OME-PIO-D48 (Rev2.0)	48 × D/I/O	80(0080)	01	30
OME-PIO-D24 (Rev6.0)	24 × D/I/O	80(C080)	01	40
OME-PIO-821	Multi-function	80	03	10
OME-PIO-DA16(Rev4.0)	16 × D/A	80(4180)	04(00)	00
OME-PIO-DA8 (Rev4.0)	8 × D/A	80(4180)	04(00)	00
OME-PIO-DA4 (Rev4.0)	4 × D/A	80(4180)	04(00)	00
OME-PISO-C64 (Rev4.0)	64 × isolated D/O **(Current sinking)**	80(0280)	08(00)	00
OME-PISO-A64 (Rev3.0)	64 × isolated D/O **(Current sourcing)**	80(8280)	08(00)	50
OME-PISO-P64 (Rev4.0)	64 × isolated D/I	80(0280)	08(00)	10
OME-PISO-P32C32 (Rev5.0)	32*isolated D/O **(Current sinking)** +32*isolated D/I	80(0280)	08(00)	20
OME-PISO-P32A32 (Rev3.0)	32*isolated D/O **(Current sourcing)** +32*isolated D/I	80(8280)	08(00)	70
OME-PISO-P8R8 (Rev2.0)	8 × isolated D/I + 8 × 220V relay	80(4200)	08(00)	30
OME-PISO-P8SSR8AC (Rev2.0)	8 × isolated D/I + 8 × SSR /AC	80(4200)	08(00)	30
OME-PISO-P8SSR8DC (Rev2.0)	8 × isolated D/I + 8 × SSR /DC	80(4200)	08(00)	30
OME-PISO-730 (Rev2.0)	16 × DI +16 ×D/O + 16 × isolated D/I + 16* isolated D/O **(Current sinking)**	80(C2FF)	08(00)	40
OME-PISO-730A (Rev3.0)	16 × DI +16 ×D/O + 16 × isolated D/I + 16* isolated D/O **(Current sourcing)**	80(62FF)	08(00)	80
OME-PISO-813 (Rev2.0)	32 × isolated A/D	80(4280)	0A(02)	00
OME-PISO-DA2 (Rev5.0)	2 × isolated D/A	80(4280)	0B(03)	00

Note: If your board is a different version, it may also have different Sub IDs. However this will present no actual problem. No matter which version of the board you select, we offer the same function calls.

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 resources of all OME-PIO/PISO cards installed in this system. Then the application program can control all functions of OME-PIO/PISO series card directly.

The sample program source is given as follows:

```
/* step1: detect all OME-PIO-D96 cards first */
wSubVendor=0x80; wSubDevice=1; wSubAux=0x10;   /* for PIO_D96 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("Threr are %d OME-PIO-D96 Cards in this PC\n",wBoards);

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

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

3.1.3 Show_PIO_PISO

Show_PIO_PISO(wSubVendor,wSubDevice,wSubAux)

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

This function will output a text string for these 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?

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

1. Remove all OME-PIO-D96 from the PC
2. Install one OME-PIO-D96 into the PC's PCI_slot1, run PIO_PISO.EXE & record the wSlotBus1 & wSlotDevice1
3. Remove all OME-PIO-D96 from the PC
4. Install one OME-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 the PC. These values will be mapped to this PC's physical slot. This mapping will not be changed for any OME-PIO/PISO cards. So it can be used to identify the specified OME-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 OME-PIO/PISO card if they compare the wSlotBus & wSlotDevice in step2 to step1.

3.3 The I/O Address Map

The I/O address of OME-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 OME-PIO/PISO series card very well.** The I/O addresses of OME-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 OME-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.

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

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

outportb(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                     */
outportb(wBase+0xe8,j3);        /* write to Port8                 */
```

NOTE: Make sure the I/O port configuration (DI or DO) before performing read/write to 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

The demo programs are provided on the software 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 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();
}
```

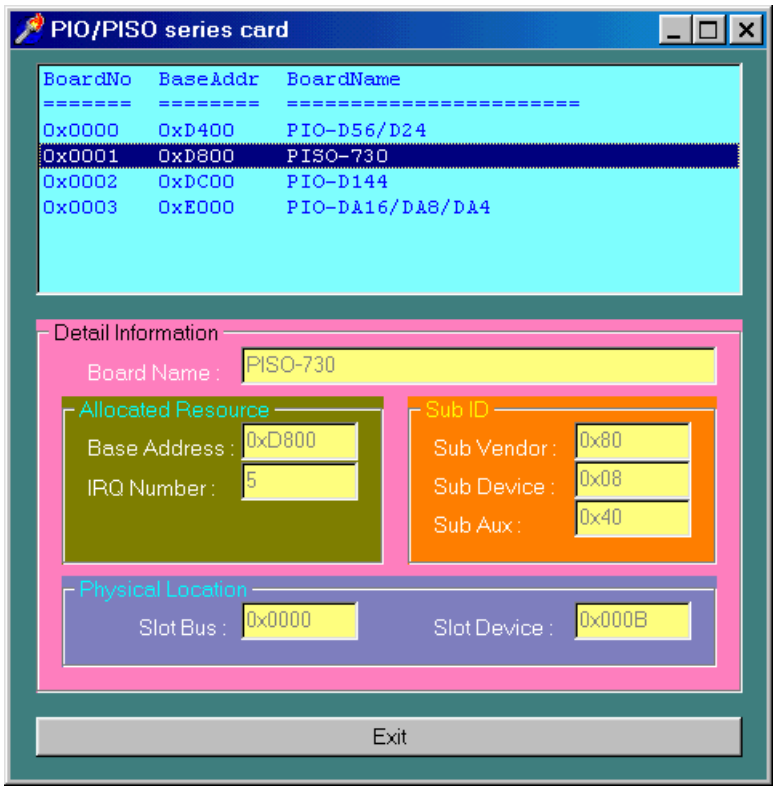
NOTE**: the PIO_PISO.EXE is valid for all PIO/PISO cards**. The user can execute the PIO_PISO.EXE to get the following information:

- 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 CD or floppy disk. It is useful for all OME-PIO/PISO series cards.

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



4.2 DEMO1

```
/* demo 1 : D/O demo of CN1                                              */
/* step 1 : connect a OME-DB-24C to CN1 of OME-PIO-D96                   */
/* step 2 : run DEMO1.EXE                                                */
/* step 3 : check the LEDs of OME-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 OME-PIO-D96
*/
printf("\n(1) Threr are %d OME-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
   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&0x0ffffff;
    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 OME-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                                     */
outportb(wBase,1);                                 /* /RESET -> 1  */
/* step3 : configure I/O direction                                   */
outportb(wBase+0xdc,0x07);                /* set CN2 as D/O ports */
outportb(wBase+0xec,0x00);                /* set CN3 as D/I ports */

for (;;)
    {
    gotoxy(1,6);
    i1=i&0xff;
    i2=(i>>8)&0xff;
    i3=(i>>16)&0xff;
    outportb(wBase+0xd0,i1);
    outportb(wBase+0xd4,i2);
    outportb(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&0x0ffffff;
    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);     /* INV0 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);     /* INV0 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);     /* INV0 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);     /* INV0 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 sources                                    */
/*          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);
}
```

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.

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

M4040/0104