PIO-D48

User's Manual

Tables of Contents

1. Introduction

The PIO-D48 provides 48 TTL digital I/O lines. The 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 individually configured to function as either an input or an output. Each group in a 24-bit bi-directional port is configured as digital inputs once the power is turned on or if a reset has been executed. Outputs in the I/O buffers are pulled through 10K resistors up to +5VDC. Outputs can be changed to pull down by changing a jumper selection on the board. This pull-up/pull-down mechanism assures that there are no erroneous outputs once power-on is activated until the board is initialized by the application software.

The 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 either an ADP-37/PCI or an ADP-50/PCI adapter. The adapter can then be fixed onto the chassis. This can then can be installed into a 5 V PCI bus and supports actual “Plug & Play” technology.

1.1 Features

- PCI Bus
- Up to 48 digital I/O channel lines
- All I/O lines buffered on the board
- Two 24-bit ports
- 3 Eight-bit groups independently selectable for either I/O on each 24-bit port
- Input/Output ports can be configured via software control
- SMD, short card, power saving
- Connects directly to DB-24P, DB-24R, DB-24PR, DB-24PD, DB-24RD, DB-24PRD, DB-16P8R, DB-24POR, DB-24SSR, DB-24C or any OPTO-22 compatible daughter boards
- 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
- Emulates two industrial-standard 8255 mode 0
- Buffers output for a higher driving capability than the 8255
- One D-Sub connector, one 50-pin flat cable connector
- Automatically detected by Windows 95/98/2000/XP

1.2 Specifications

- **<u>All inputs are TTL compatible</u>**
  Logic high voltage: 2.4V (Min.)
  Logic low voltage: 0.8V (Max.)
- **<u>All outputs are TTL compatible</u>**
  Sink current: 64 mA (Max.)
  Source current: 32 mA (Max.)
- Environmental :
  Operating Temp. : 0 to 60°C
  Storage Temp. : -20°C to 80 °C
  Humidity: 0 to 90 % non-condensing
- Dimension : 156mm x 105mm
- Power consumption: +5V @ 900mA

1.3 Product Check List

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

- One PIO-D48 card
- One company floppy diskette or CD
- One release note

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

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

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

2. Hardware configuration

2.1 Board Layout & Default Settings

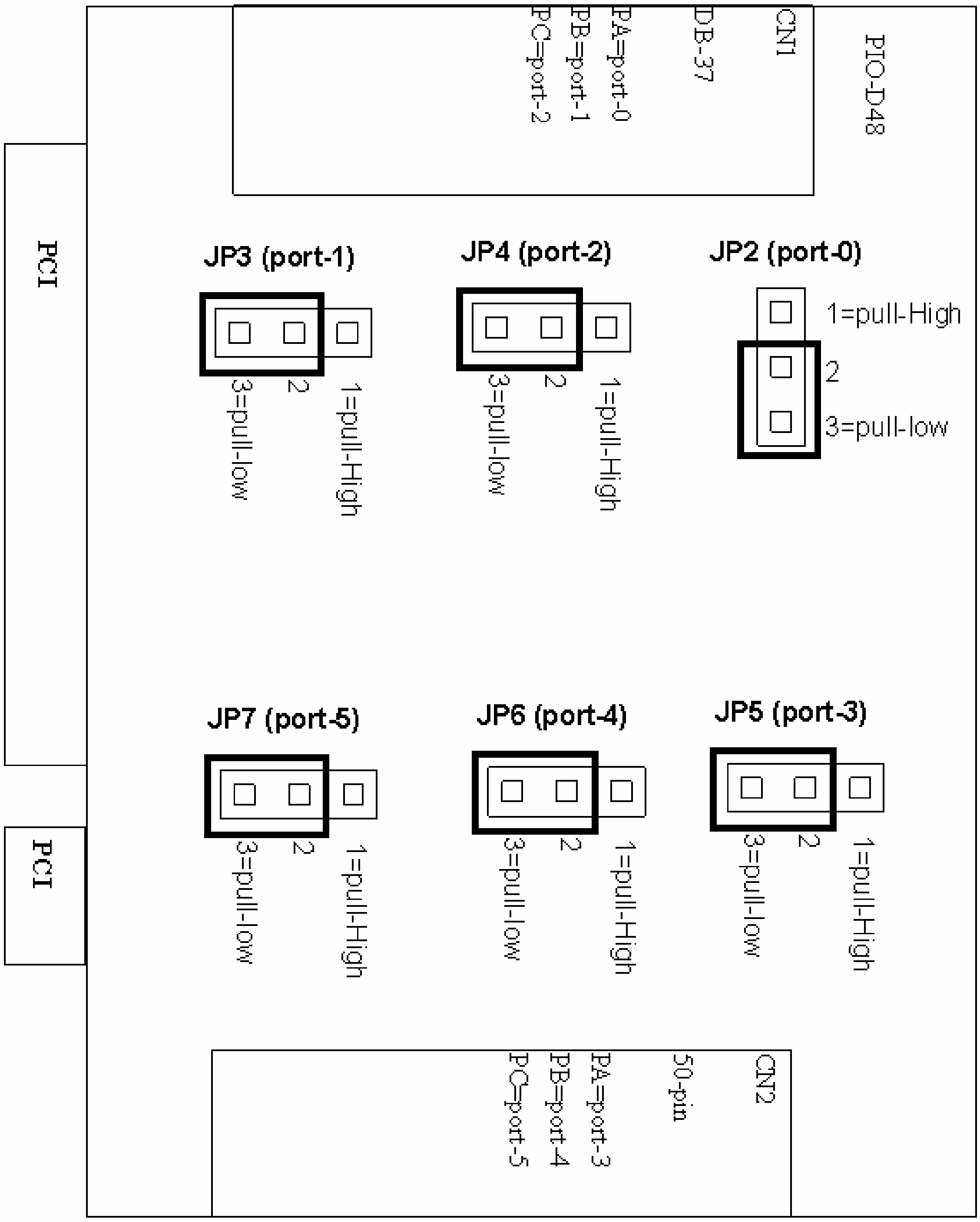


Figure 2.1

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 PIO-D48. Each I/O port can be programmed as a D/I or D/O port. When the PC is first powered-on or reset, all the ports are configured as D/I ports. These D/I ports can be selected to either pull-high or pull-low via placement of the JP2 ~ JP7 jumpers. These I/O port locations are given as follows:

Table 2.1

Connector	**PA0 to PA7**	**PB0 to PB7**	**PC0 to PC7**
CN1 (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 the board layout & I/O port locations.
- Note 2: Refer to Sec. 2.1 for JP 2 ~ 7 pull-high/pull-low placements.

2.3 Pin Assignment

The Pin assignments for all connectors on the PIO-D48 are represented in Figures 2.2 and 2.3. All signal sources for each digital input or output pin (channel) is TTL compatible.

CN1: 37 pin D-type female connector (port-0, port-1, port2)

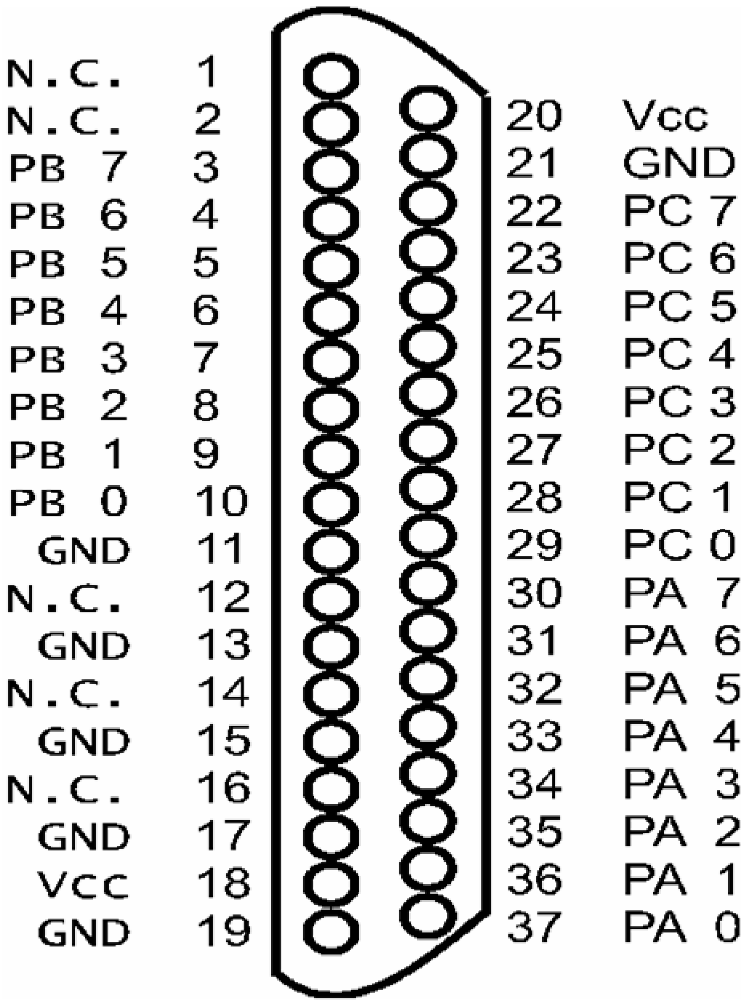


Figure 2.2

PA0 ~ PA7 : port-0

PB0 ~ PB7 : port-1

PC0 ~ PC7 : port-2

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

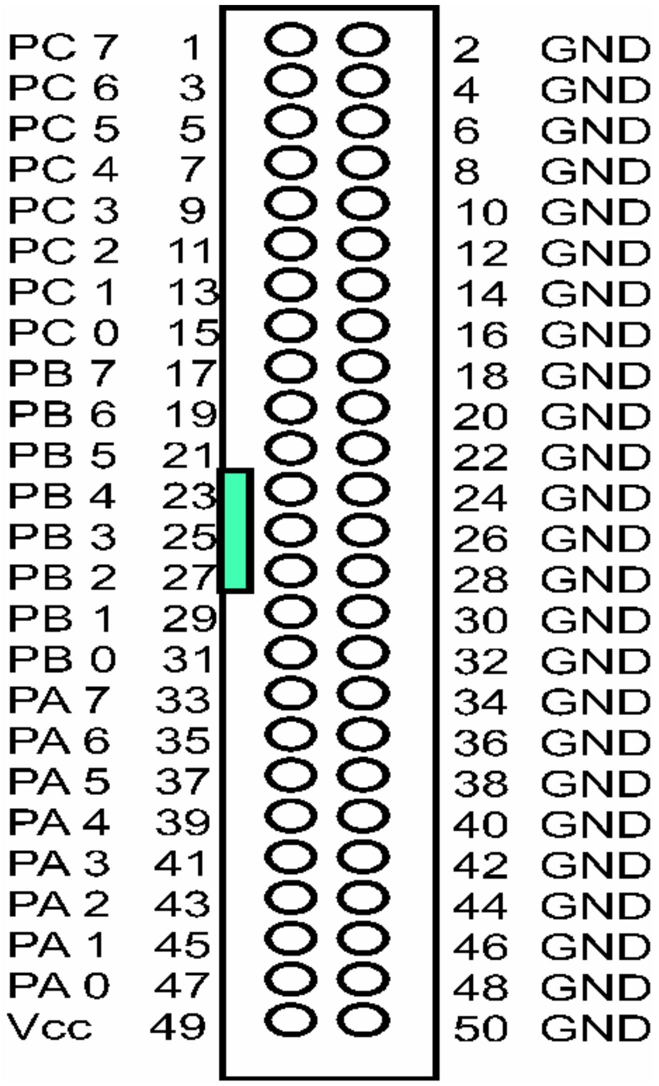


Figure 2.3

PA0 ~ PA7 : port-3

PB0 ~ PB7 : port-4

PC0 ~ PC7 : port-5

2.4 Enable I/O Operation

When the PC is first turned on, all operations involved with digital I/O channels are disabled. Note that the digital I/O channels are enabled or disabled by the RESET\ signal, (refer to Sec. 3.3.1 for more information related to this). The power-on states are given as follows:

- D/I/O operations for each port are disabled.
- D/I/O ports are all configured as Digital input ports.
- D/O latch register outputs are all at high impedance.(refer to Sec. 2.5)

The user has to perform some initialization before using these digital I/O ports. The recommended steps are given below:

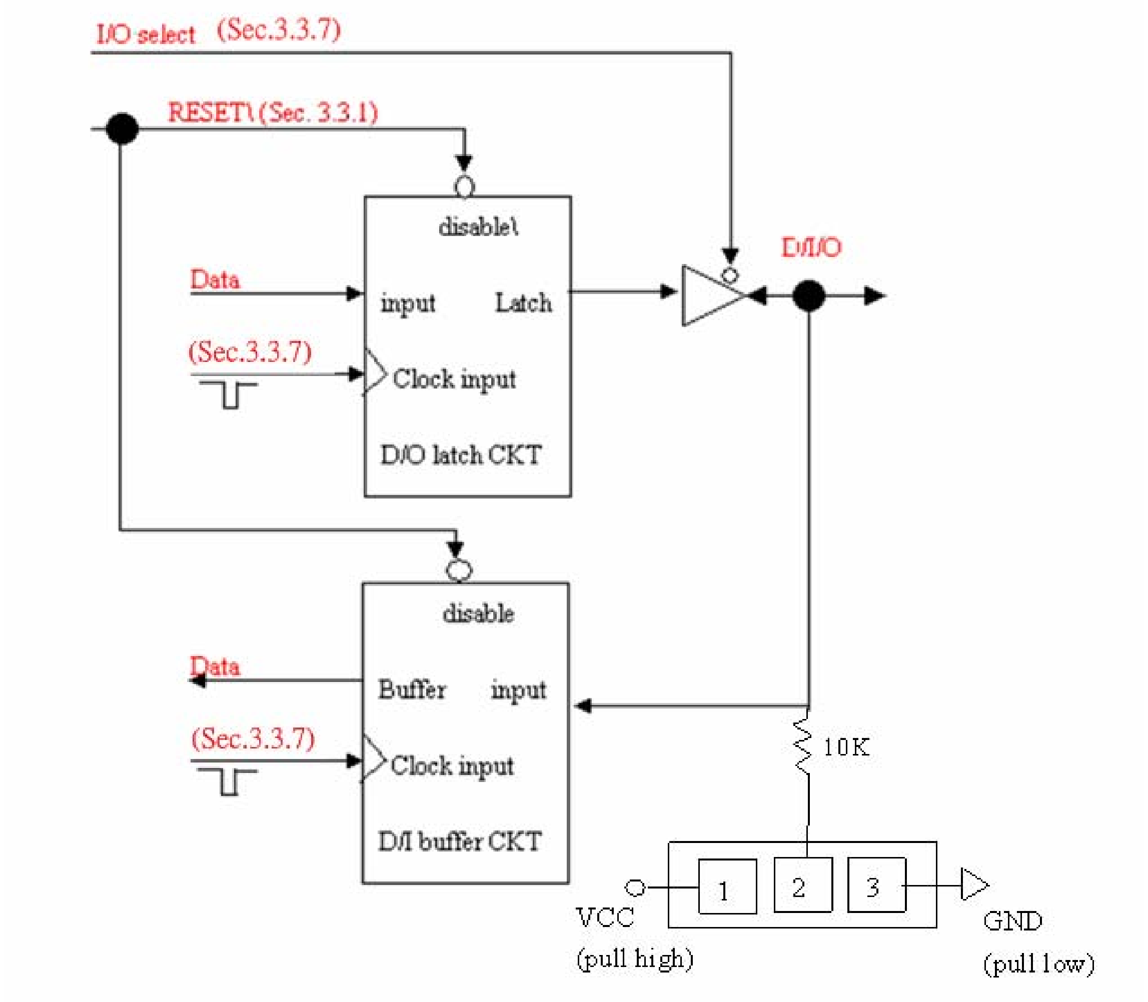
Step 1: Find the address-mapping for PIO/PISO cards. (Refer to Sec.3.1)
Step 2: Enable all Digital I/O operations. (Refer to Sec. 3.3.1).
Step 3: Configure the first three ports to their expected D/I/O states & send their initial values to every D/O port (Refer to Sec. 3.3.7)
Step 4: Configure the other three ports to their expected D/I/O states & send their initial values to every D/O port (Refer to Sec. 3.3.7)

For more information on the initial procedure for digital I/O ports, please refer to the DIO demo program.

2.5 D/I/O Architecture

The digital I/O control architecture for the PIO-D48 is demonstrated in Figure 2.4. The operation method used for the of control signal is presented below.

- RESET\ is in the Low-state → all D/I/O operations are disabled
- RESET\ is in the High-state → all D/I/O operations are enabled.
- If D/I/O is configured as a D/I port → D/I= external input signal.
  → can be selected as either pull-high or pull-low as chosen by placement of the JP2/3/4/5/6/7 jumpers (1-2-ON=pull-high, 2-3-ON=pull-low).
- If D/I/O is configured as a D/O port → D/I = read back D/O
- If D/I/O is configured as a D/I port → sending data to a Digital input port will only change the D/O latch register. The latched data will be output when the port is configured as digital output and is activated right away.



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

Figure 2.4

2.6 Interrupt Operation

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

- INT_CHAN_0: PC3/PC7 from port-2(refer to Sec. 2.6.2)
- INT_CHAN_1: PC3/PC7 from port-5(refer to Sec. 2.6.3)
- INT_CHAN_2: Cout0(refer to Sec. 2.6.4)
- INT_CHAN_3: Cout2(refer to Sec. 2.6.5)

Note that DEMO4.C, DEMO7.C, DEMO8.C, DEMO9.C & DEMO10.C are demo programs for a single interrupt source and DEMO11.C is the demo program for more than one interrupt source in the DOS operating system. If only one interrupt signal source is used, the interrupt service routine does not need to identify the interrupt source. However, if there are more than one interrupt source, the interrupt service routine has to identify the active signals in the following manner:

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 INT_CHAN_0 & non-inverter/inverted the INT_CHAN_0 signal.
4. If INT_CHAN_1 is active, service INT_CHAN_1 & non-inverted/inverted the INT_CHAN_1 signal.
5. If INT_CHAN_2 is active, service INT_CHAN_2 & non-inverted/inverted the INT_CHAN_2 signal.
6. If INT_CHAN_3 is active, service INT_CHAN_3 & non-inverted/inverted the INT_CHAN_3 signal.
7. Update the interrupt status

Limitation: if the interrupt signal is too short, the new status may be the same as the old status. So the interrupt signal must be held active until the interrupt service routine has been executed. This hold time is different for differing operating systems. The hold time can be as short as a micro-second or as long as 1 second. In general, 20ms is enough for all O.S.

2.6.1 *Interrupt Block Diagram for the PIO-D48*

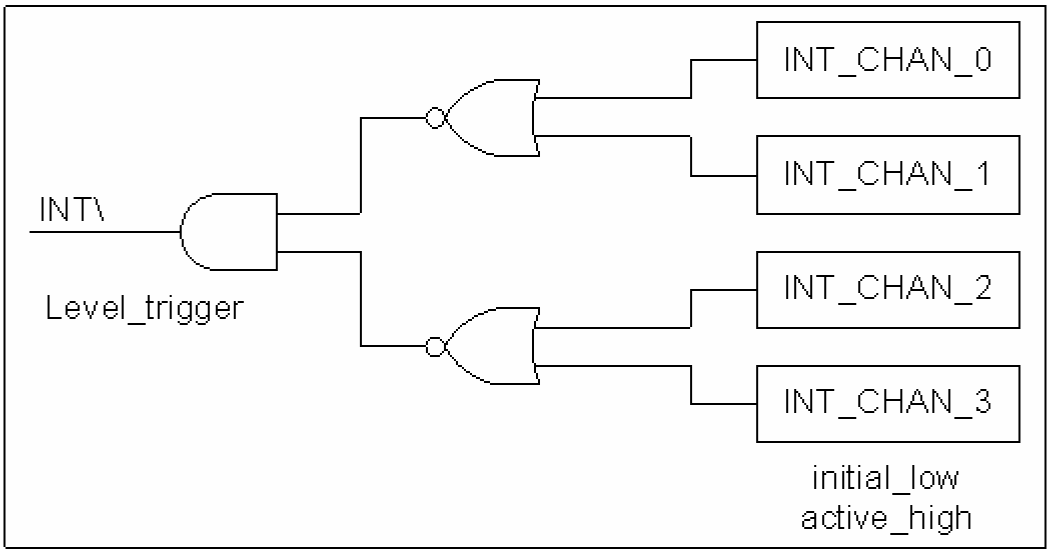


Figure 2.5

The INT\ interrupt output signals are **level-trigger & Active_Low**. If the INT\ generates a low-pulse, the PIO-D48 will interrupt the PC once per occasion. If the INT\ is fixed in low level, the PIO-D48 will interrupt the PC continuously. So the INT_CHAN_0/1/2/3 must be controlled with **pulse_type** signals. **They should normally be fixed in a low level state and 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 per occasion. So the interrupt service routine has to read the status for all interrupt channels for multi-channel interruptions. (Refer to Sec. 2.6 for more information).

DEMO11.C → for both INT_CHAN_0 & INT_CHAN_1

If only one interrupt source is used, the interrupt service routine dosen't have to read the interrupt source status. Note that DEMO4.C to DEMO10.C are demo programs for a single-channel interruption within the DOS operating system.

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

2.6.2 INT_CHAN_0

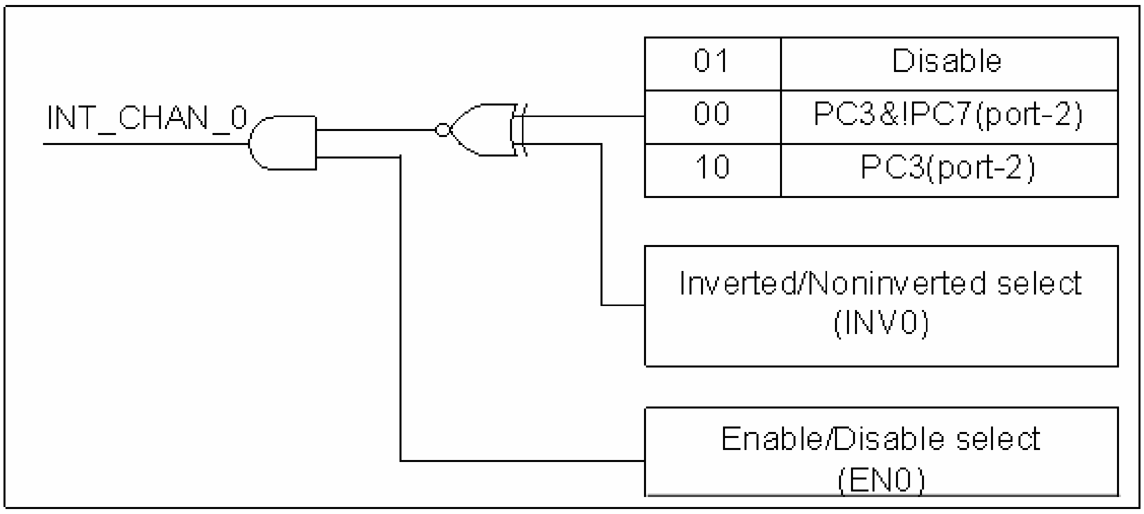

INT_CHAN_0
01
Disable
00
PC3&!PC7(port-2)
10
PC3(port-2)
Inverted/Noninverted select (INV0)
Enable/Disable select (EN0)

Figure 2.6

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

INT_CHAN_0 can be equal to **PC3&!PC7** or be **PC3** programmable as is shown below:(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

EN0 can be used to enable/disable the INT_CHAN_0 as follows: (Refer to Sec. 3.3.4)

EN0=0 → INT_CHAN_0=disabled
EN0=1 → INT_CHAN_0=enabled

INV0 can be used to invert/non-invert the PC3 or PC3&!PC7 as follows: (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 the following demo programs for more information:

DEMO8.C → for INT_CHAN_0 only (PC3 of port-2)
DEMO9.C → for INT_CHAN_0 only (PC3&!PC7 of port-2)

2.6.3 INT_CHAN_1

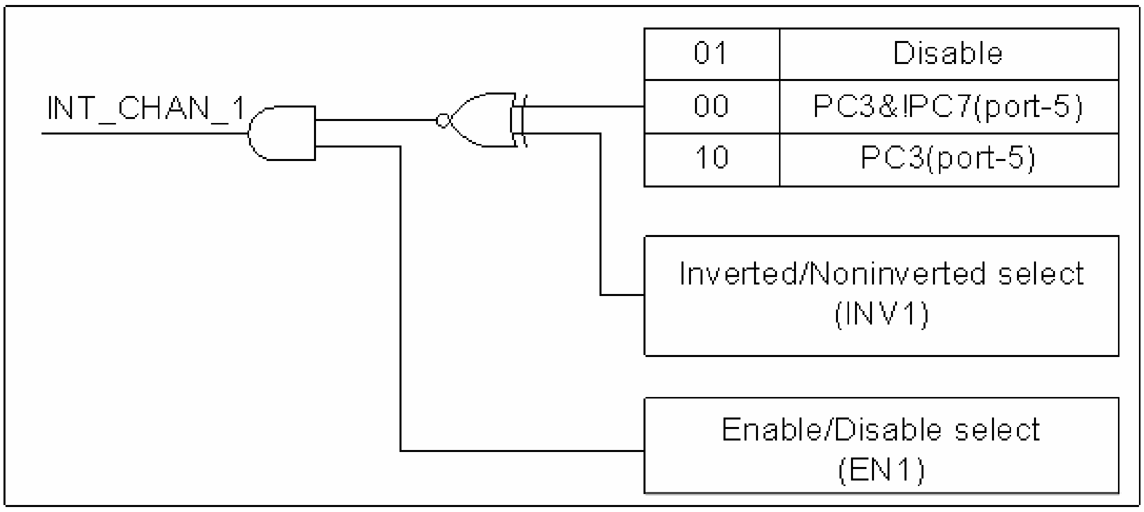

INT_CHAN_1
01
Disable
00
PC3&!PC7(port-5)
10
PC3(port-5)
Inverted/Noninverted select (INV1)
Enable/Disable select (EN1)

Figure 2.7

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

INT_CHAN_1 can be equal to **PC3&!PC7** or be **PC3** programmable as is shown below:(Refer to Sec. 3.3.9)

CTRL_D5=0, CTRL_D4=1 → INT_CHAN_1=disabled
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

EN1 can be used to enable/disable the INT_CHAN_1 as follows: (Refer to Sec. 3.3.4)

EN1=0 → INT_CHAN_1=disabled
EN1=1 → INT_CHAN_1=enabled

INV1 can be used to invert/non-invert the PC3 or PC3&!PC7 as follows: (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 the following demo program for more information:

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

NOTE: Refer to Sec. 2.6.2 for active high-pulse generation.

2.6.4 INT_CHAN_2

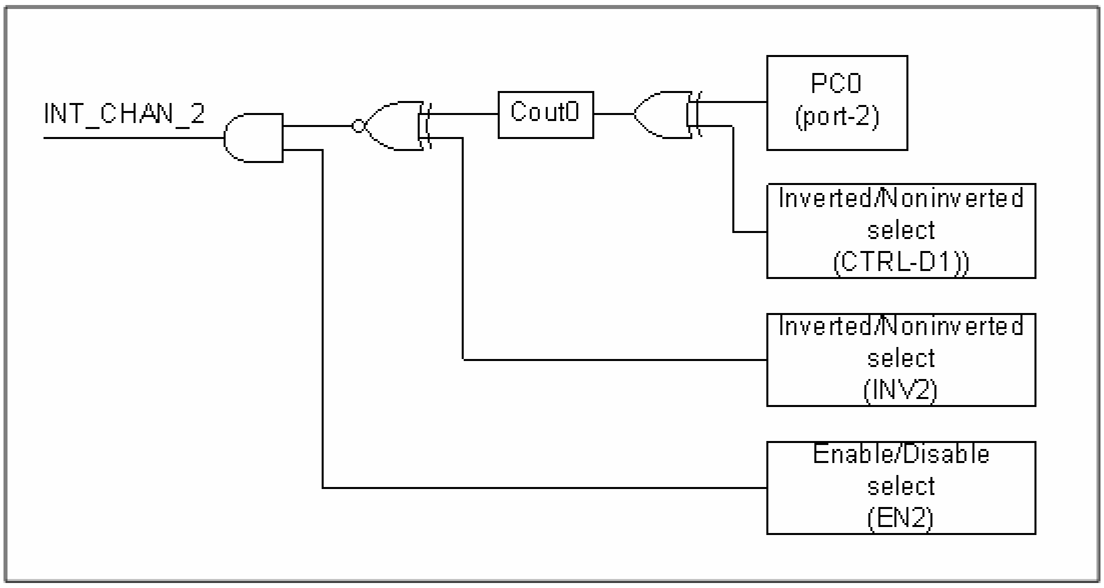

INT_CHAN_2
Cout0
PC0 (port-2)
Inverted/Noninverted select (CTRL-D1))
Inverted/Noninverted select (INV2)
Enable/Disable select (EN2)

Figure 2.8

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

PC0 (port-2) can be inverted/non-inverted programmable as is shown below: (Refer to Sec. 3.3.9)

CTRL_D1=0 → Cin0=PC0 of port-2
CTRL_D1=1 → Cin0=!PC0 of port-2

EN2 can be used to enable/disable the INT_CHAN_2 as follows: (Refer to Sec. 3.3.4)

EN2=0 → INT_CHAN_2=disabled
EN2=1 → INT_CHAN_2=enabled

INV2 can be used to invert/non-invert the Cout0 as follows: (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 the following demo program for more information:

DEMO7.C → for INT_CHAN_2 only (Cout0)

NOTE: Refer to Sec. 2.6.2 for active high-pulse generation.

2.6.5 INT_CHAN_3

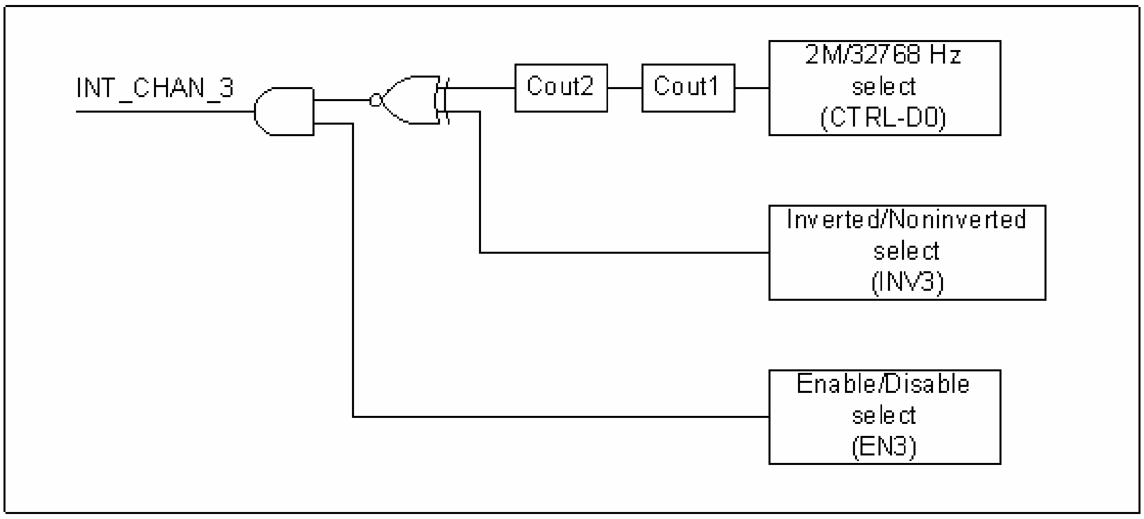

INT_CHAN_3
Cout2
Cout1
2M/32768 Hz select (CTRL-D0)
Inverted/Noninverted select (INV3)
Enable/Disable select (EN3)

Figure 2.9

INT_CHAN_3 should normally be fixed in a low-level state and generate a high_pulse to interrupt the PC.

Cin1 can be 2M/32768Hz programmable as is given below: (Refer to Sec. 3.3.9)
CTRL_D0=0 → Cin1=2M clock source
CTRL_D0=1 → Cin1=32768 Hz clock source

EN3 can be used to enable/disable the INT_CHAN_3 as follows: (Refer to Sec. 3.3.4)
EN3=0 → INT_CHAN_3=disabled
EN3=1 → INT_CHAN_3=enabled

INV3 can be used to invert/non-invert the Cout0 as follows: (Refer to Sec. 3.3.6)
INV2=3 → INT_CHAN_3=invert (Cout2)
INV2=3 → INT_CHAN_3=non-invert (Cout2)

Refer to the following demo program for more information:
DEMO4.C → for INT_CHAN_3 only (Cout2)

NOTE: Refer to Sec. 2.6.2 for active high-pulse generation.

2.7 Daughter Boards

2.7.1 DB-37

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

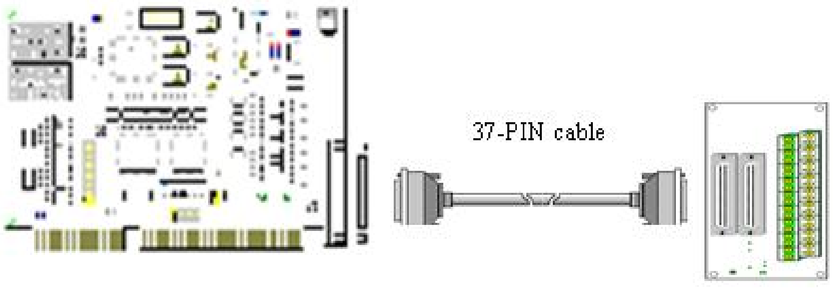


Figure 2.10 DB-37

2.7.2 DN-37 & DN-50

The DN-37 is a general purpose daughter board for DB-37 pins with DIN-Rail Mountings. The DN-50 is designed for 50-pin flat-cable headers win DIN-Rail mountings. They are also designed for easy wire connection via pin-to-pin.

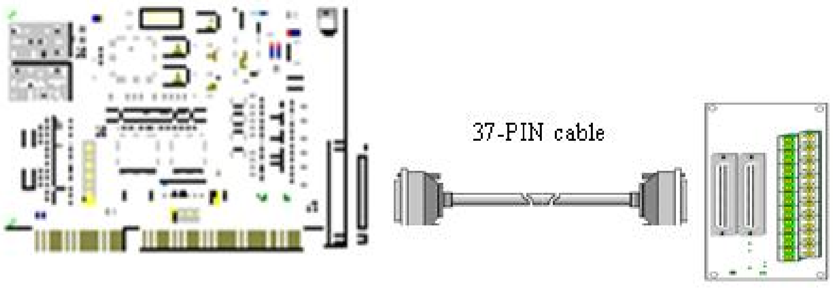


Figure 2.11 DN-37/DN-50

2.7.3 DB-8125

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

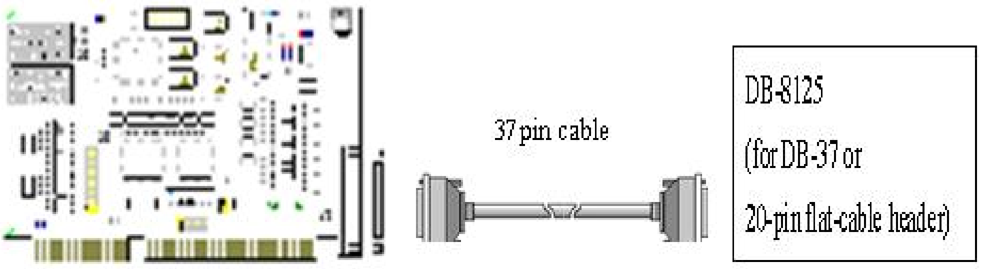


Figure 2.12 DB-1825

2.7.4 ADP-37/PCI & ADP-50/PCI

The ADP-37/PCI & ADP-50/PCI are extenders for the 50-pin header. The one side of the ADP-37/PCI or the ADP-50/PCI can be connected to a 50-pin header. The other side can be mounted onto the PC chassis as is depicted by the following:

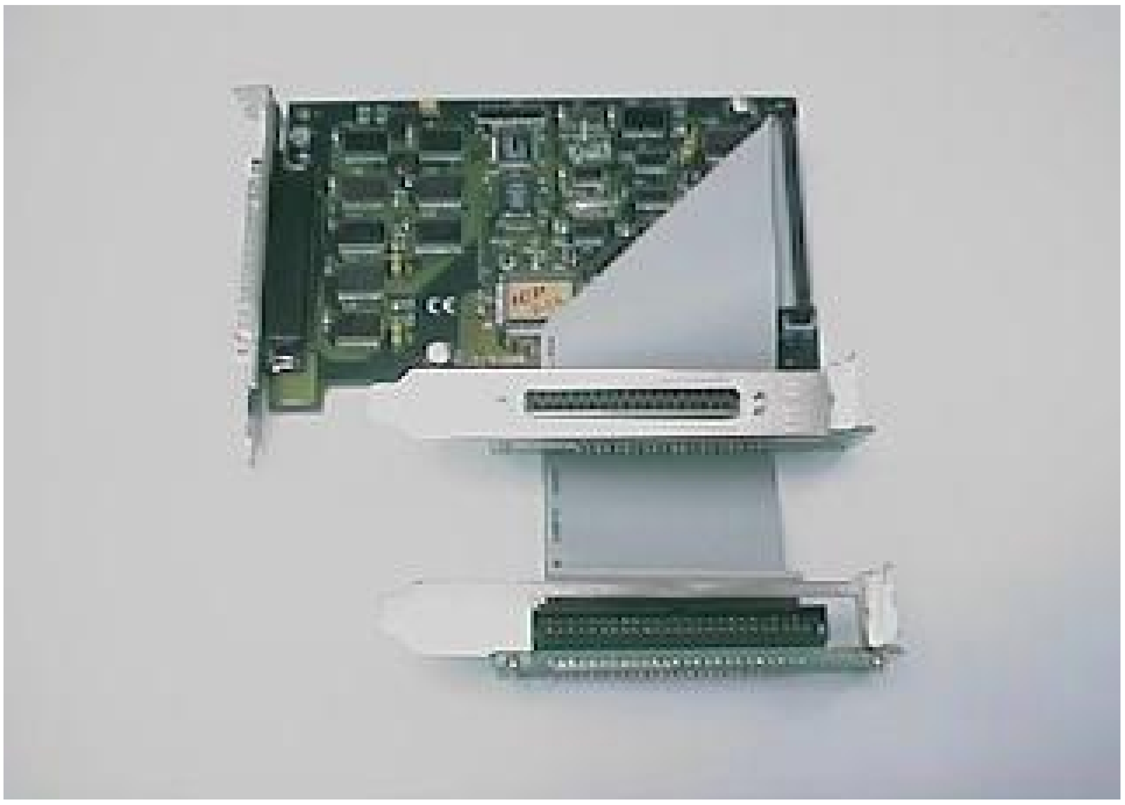


Figure 2.13

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

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

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

The DB-24P is a 24-channel isolated digital input daughter board. The optically isolated inputs of the DB-24P consist of a bi-directional optocoupler with a resistor for current sensing. You can use the DB-24P to sense DC signals from TTL levels up to 24V or use the DB-24P to sense a wide range of AC signals. You can also use this board to isolate the computer from large common-mode voltage, ground loops and transient voltage spikes that often occur in industrial environments, as shown in Figure 2.7. Table 2.4 is the comparison of DB-24P and DB-24PD.

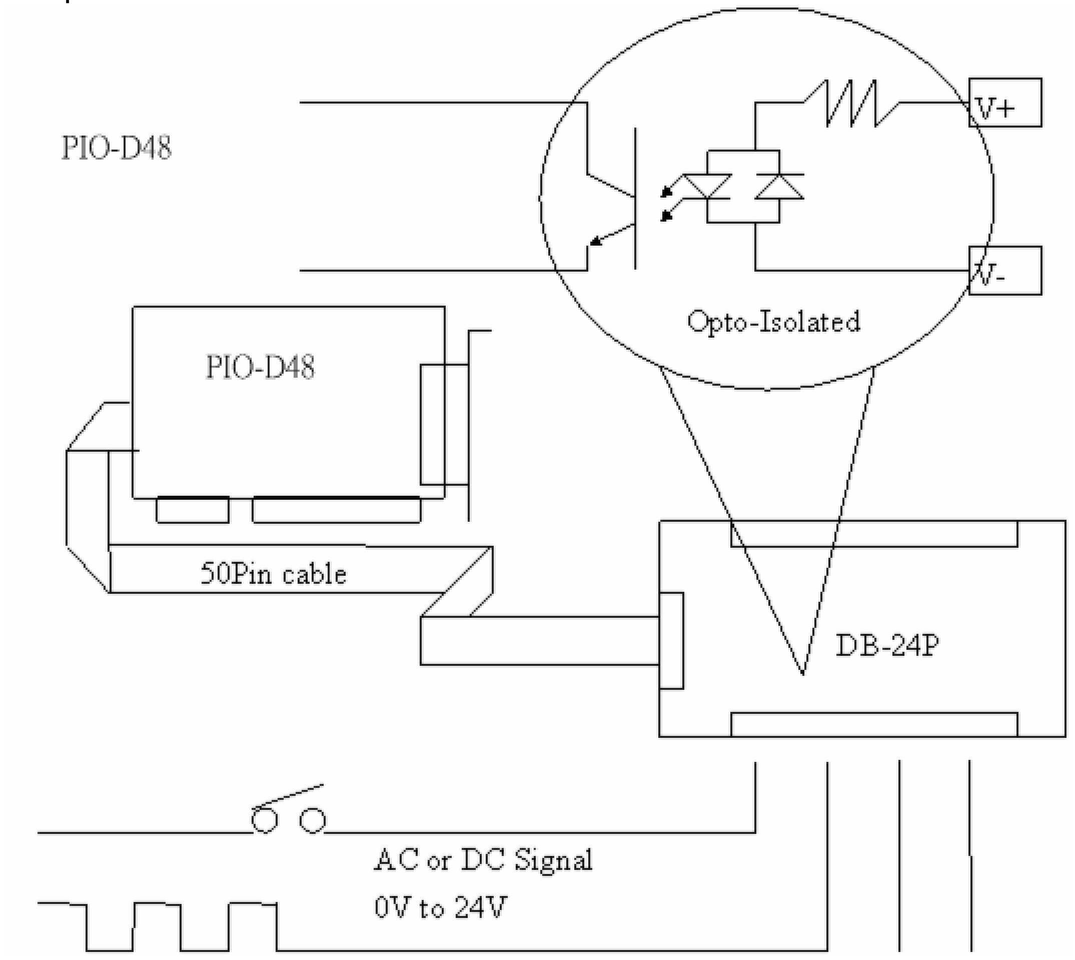


Figure 2.14

Table 2.2

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

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

The DB-24R, 24-channel relay output board, consists of 24 form-C relays for efficiently controlling the switch with the use of an appropriately loaded program. The relays are energized by applying a 12V/24V voltage signal to the appropriate relay channel on the 50-pin flat-cable connector. There are 24 enunciator LEDs for each relay channel and the LED light will go on when their associated relay has been activated. The control scheme is illustrated below.

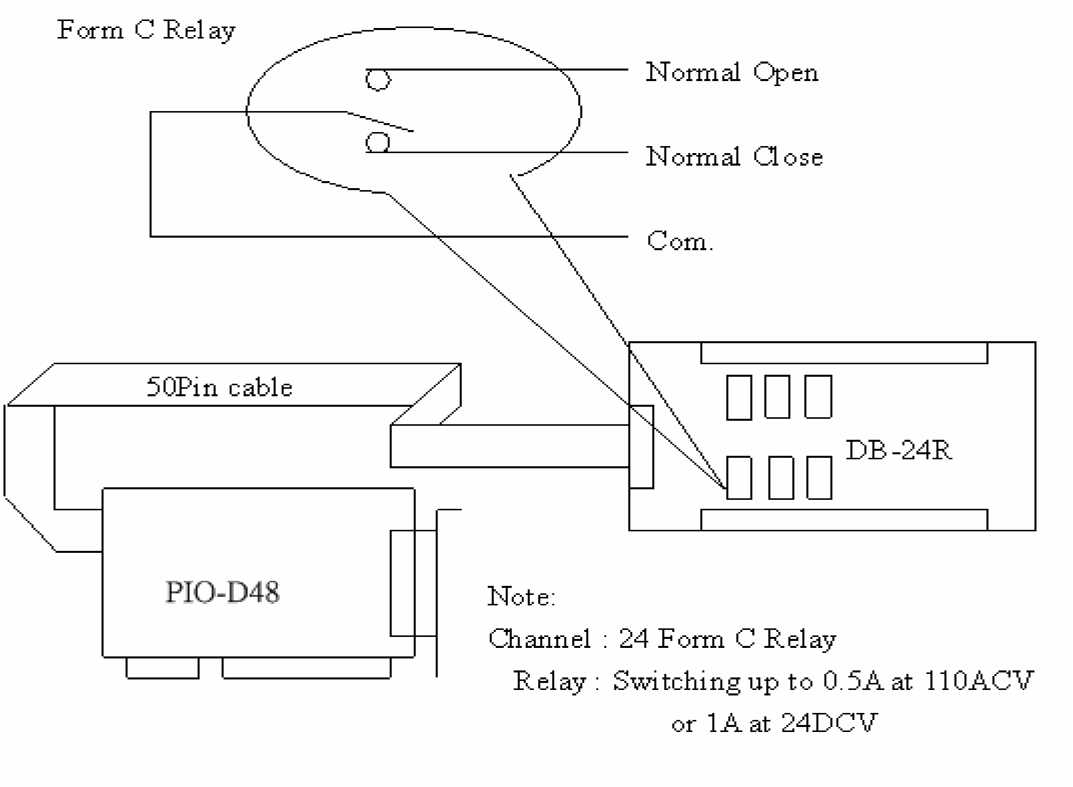


Figure 2.15

Table 2.3

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

Table 2.4

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

2.7.7 Daughter Board Comparison Table

Table 2.5

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

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

3. I/O Control Register

3.1 How to Find the I/O Address

The plug & play BIOS will assign a proper I/O address to every PIO/PISO series card in the power-on stage. The IDs for the PIO-D48 cards are given as follows:

< REV 1.x > :

- **Vendor ID = 0xE159**
- **Device ID = 0x0002**

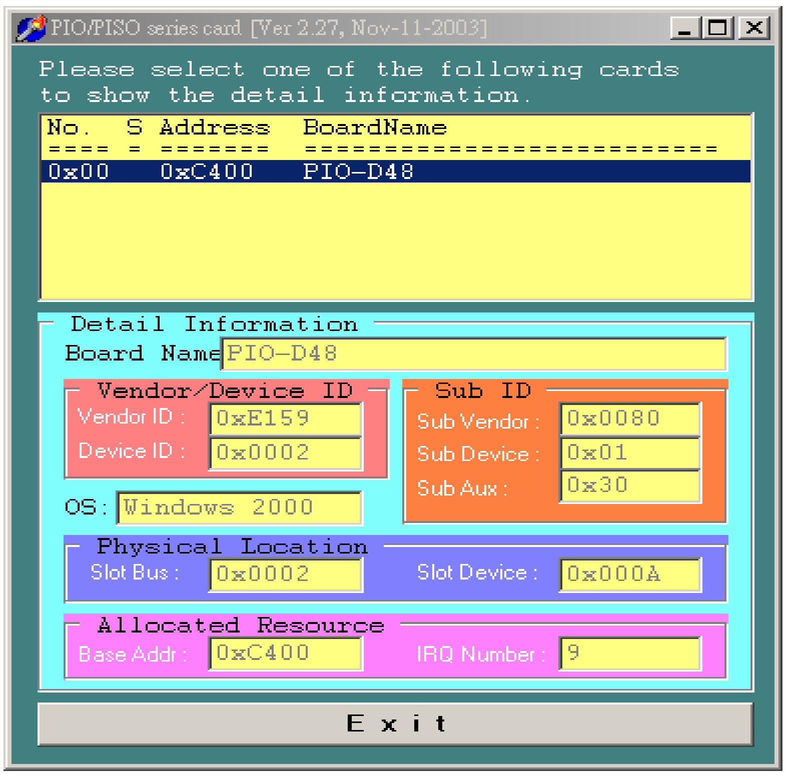
- **Sub-vendor ID = 0x80**
- **Sub-device ID = 0x01**
- **Sub-aux ID = 0x30**

< REV 2.0 or above > :

- **Vendor ID = 0xE159**
- **Device ID = 0x0001**

- **Sub-vendor ID = 0x0080**
- **Sub-device ID = 0x01**
- **Sub-aux ID = 0x30**

The utility program, PIO_PISO.EXE, will detect and present all information from the PIO/PISO cards installed on this PC, as shown in the following figure. Besides, for how to identify the PIO series cards of ICPDAS data acquisition board by the sub-vender, sub-device and sub-Aux ID is given in table 3-1 below the following figure.



PIO/PISO series card [Ver 2.27, Nov-11-2003]

Please select one of the following cards
to show the detail information.

```
No.   S Address   BoardName
==== = =======   =========================
0x00    0xC400    PIO-D48
```

Detail Information

Board Name: PIO-D48

Vendor/Device ID
Vendor ID : 0xE159
Device ID : 0x0002

Sub ID
Sub Vendor : 0x0080
Sub Device : 0x01
Sub Aux : 0x30

OS : Windows 2000

Physical Location
Slot Bus : 0x0002
Slot Device : 0x000A

Allocated Resource
Base Addr : 0xC400
IRQ Number : 9

Figure 3.1

Table 3-1

PIO/PISO series card	Description	Sub_Sendor	Sub_Device	Sub_AUX
PIO-D168	168 * DIO	9880	01	50
PIO-D168A	168 * DIO	80	01	50
PIO-D144(REV4.0)	144 * D/I/O	80 (5C80)	01	00
PIO-D96 (REV4.0)	96 * D/I/O	80 (5880)	01	10
PIO-D64 (REV2.0)	64 * D/I/O	80 (4080)	01	20
PIO-D56 (REV6.1)	24 * D/I/O + 16 * D/I+16*D/O	80 (8080)	01	40
PIO-D48 (REV2.0)	48 * D/I/O	80 (0080)	01	30
PIO-D24 (REV6.1)	24 * D/I/O	80 (8080)	01	40
PIO-821	Multi-function	80	03	10
PIO-DA16	16 * D/A	80	04	00
PIO-DA8	8 * D/A	80	04	00
PIO-DA4	4 * D/A	80	04	00
PISO-C64	64 * isolated D/O (**Current sinking**)	80	08	00
PISO-A64	64 * isolated D/O **(Current sourcing)**	80	08	50
PISO-P64	64 * isolated D/I	80	08	10
PISO-P32C32 (REV5.5)	32* isolated D/O (**Current sinking**) + 32* isolated D/I	80(4280)	08(00)	20(00)
PISO-P32A32 (REV3.3)	32*isolated DO (**Current sourcing**) + 32* isolated D/I	80(C280)	08	70
PISO-P8R8	8* isolated D/I + 8 * 220V relay	80	08	30
PISO-P8SSR8AC	8* isolated D/I + 8 * SSR /AC	80	08	30
PISO-P8SSR8DC	8* isolated D/I + 8 * SSR /DC	80	08	30
PISO-730	16*DI + 16*D/O + 16* isolated D/I + 16*isolated D/O (**Current sinking**)	80(CA80)	08	40
PISO-730A	16*DI + 16*D/O + 16* isolated D/I + 16*isolated D/O (**Current sourcing**)	80	08	80
PISO-813	32 * isolated A/D	80(C280)	0A(00)	00(02)
PISO-DA2	2 * isolated D/A	80	0B	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.

3.2 The Assignment of the I/O Address

The Plug & Play BIOS will assign the proper I/O address to a 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 find it very difficult to identify which board is card_0. The software driver can support a maximum of 16 boards. Therefore, the user can install 16 PIO/PSIO series cards onto one PC system. The methods used to find and identify card_0 and card_1 is demonstrated below:

The simplest way to identify which card is card_0 is to use wSlotBus & wSlotDevice in the following manner:

1. Remove all PIO-D48 boards from the PC
2. Install one PIO- D48 onto the PC's PCI_slot1, run PIO_PISO.EXE. Then record the wSlotBus1 and wSlotDevice1 information.
3. Remove all PIO- D48 boards from the PC
4. Install one PIO- D48 into the PC's PCI_slot2 and run PIO_PISO.EXE. Then record the wSlotBus2 and wSlotDevice2 information.
5. Repeat steps (3) & (4) for every PCI_slot and record the information from wSlotBus and wSlotDevice.
6. The records may look similar to the table below:

Table 3-2

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 the wSlotBus and wSlotDevice information on a PC. These values will be mapped to this PC's physical slot.

This mapping will not be changed for any PIO/PISO card. Therefore, this information can be used to identify the specified PIO/PISO card by following these next 3 steps:

Step1: Using the wSlotBus and wSlotDevice information in table 3-2

Step2: Input the board number into funtion PIO_GetConfigAddressSpace(…) to get the specified card's information, especially the wSlotBus and wSlotDevice information.

Step3: The user can identify a specified PIO/PISO card by comparing it to the data from the wSlotBus & wSlotDevice found in step1 and step2.

Note that normally the card installed in slot 0 is card0 and the card installed in slot1 is card1 for PIO/PISO series cards.

3.3 The I/O Address Map

The I/O address for PIO/PISO series cards are automatically assigned by the main board ROM BIOS. The I/O address can also be re-assigned by the user. It is strongly recommended that users do not change the I/O address. The Plug & Play BIOS will effectively perform the assignment of proper I/O addresses to each PIO/PISO series card. The I/O addresses for the PIO-D48 are given in the table belows, which are based on the base address of each card.

Table 3.3

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\

When the PC's power is first turned on, RESET\ signal is in a Low-state. **This will disable all D/I/O operations.** The user has to set the RESET\ signal to a High-state before any D/I/O command applications are initiated.

```
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

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 turned on, all Aux signals are in a Low-state. All Aux are designed as D/I for all PIO/PISO series.

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

When the Aux is used for D/O, the output state is controlled by this register. This register is designed for feature extension. Therefore, do not use this register.

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

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

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

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

Aux0=INT_CHAN_0, Aux1=INT_CHAN_1, Aux2=INT_CHAN_2, Aux3=INT_CHAN_3, Aux7~4=Aux-ID. The Aux 0~3 are used as interrupt source. The interrupt service routine has to read this register to identify the interrupt source. Refer to Sec. 2.6 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

This register provides a function to control invert or non-invert for the interrupt signal source. A detailed application example is given below.

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

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

outportb(wBase+0x2a,0x0e);  /* select the inverted input of INT_CHAN_0          */
                            /* select the non-inverted input from the others     */

outportb(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. 2.6 and demo5.c for more information.

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

■ **8255 control word (mode-0)**

(Read/Write): wBase+0xcc / 0xdc

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	PA	PC-H	0	PB	PC-L

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

(Read/Write): wBase+0xcc=8255-1
(Read/Write): wBase+0xdc=8255-2

PA/ PB/ PC-H/ PC-L : 1→ import, 0→ outport.
PC-H: high nibble of PC
PC-L: low nibble of PC

■ **Read/Write 8-bit data of 8255**

(Read/Write):wBase+0xc0/0xc4/0xc8/0xd0/0xd4/0xd8

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

(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+0xd0=8255-2-PA → port_3
(Read/Write): wBase+0xd4=8255-2-PB → port_4
(Read/Write): wBase+0xd8=8255-2-PC → port_5

```
outportb(wBase+0xcc,0x80);    /* port-0, port-1, port-2 are D/O port */
outportb(wBase+0xc0,V1);      /* write to port_0 (PA)                */
outportb(wBase+0xc4,V2);      /* write to port_1 (PB)                */
outportb(wBase+0xc8,V3);      /* write to port_2 (PC)                */

outportb(wBase+0xdc,0x9B);    /* port-3, port-4, port-5 are D/I port  */
V1=inportb(wBase+0xd0);       /* read from port_3 (PA)                */
V2=inportb(wBase+0xd4);       /* read from port_4 (PB)                */
V3=inportb(wBase+0xd8);       /* read from port_5 (PC)                */
```

3.3.8 Read/Write 8254

- **8254 control word**

(Read/Write): wBase+0xec

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

SC1,SC0 :	00: counter0	
	01: counter1	
	10: counter2	
	11: read -back command	
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	
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
BCD :	0: binary count	1: BCD count

- **Read/Write 8-bit data of 8254**

(Read/Write):wBase+0xe0/0xe4/0xe8

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

(Read/Write): wBase+0xec=8254 control word
(Read/Write): wBase+0xe0=8254-counter-0
(Read/Write): wBase+0xe4=8254-counter-1
(Read/Write): wBase+0xe8=8254-counter-2

```
outportb(wBase+0xec,0x30);    /* Counter0, mode-0                         */
outportb(wBase+0xe0,0xff);    /*  write to low byte first                 */
outportb(wBase+0xe0,0xff);    /*  write to high byte second               */
                              /*  Then Counter0 will down count from 0xffff  */
```

The configuration of 8254 counter :

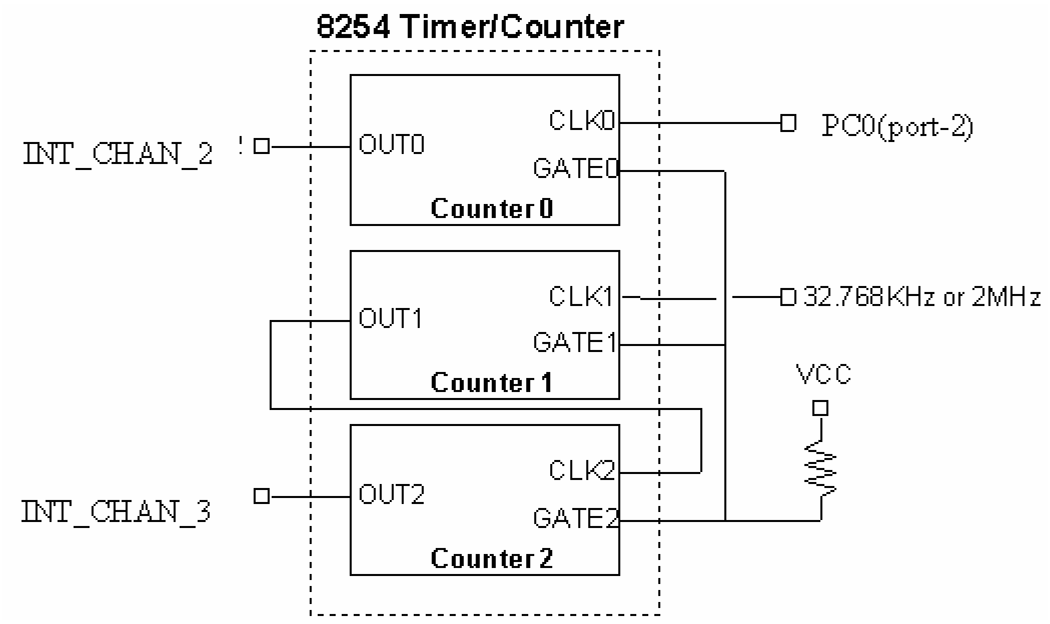


Figure 3.2

Refer to the following demo programs for more related information:

- Int2 demo : counter0 (using interrupt INT_CHAN_2)
- Int3 demo : counter1-counter2 (using interrupt INT_CHAN_3)

3.3.9 Read/Write Clock/Int Control Register

(Read/Write): wBase+0xf0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	CTRL-D5	CTRL-D4	CTRL-D3	CTRL-D2	CTRL-D1	CTRL-D0

CTRL-D0: timer source CLK1 selection (refer to Sec. 3.3.8)
0→ 2MHz , 1→ 32.768KHz

CTRL-D1: invert/non-invert the PC0 of port-2 (refer to Sec. 2.6.4)
0→ non-invert, 1→ invert

CTRL-D3, CTRL-D2: interrupt source select (refer to Sec. 2.6.2)
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

CTRL-D5, CTRL-D4: interrupt source select (refer to Sec. 2.6.3)
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

4. Software Installation

The PIO-D48 can be used in DOS and Windows 98/Me/NT/2000/XP. For Windows O.S, the recommended installation steps are given in Sec 4.1 ~ 4.2

4.1 Software Installing Procedure

Step 1:	Insert the companion CD into the CD-ROM driver and wait a few seconds until the installation program starts automatically. If it does not start automatically for some reason, then please double-click the file \NAPDOS\AUTO32.EXE on the CD.
Step 2:	Click the item: Install Toolkits (Software) / Manuals.
Step 3:	Click the item: PCI Bus DAQ Card.
Step 4:	Click PIO-DIO.
Step 5:	Click "install Toolkit for Windows 98 (Or Me, NT, 2000, XP)".

Then, the InstallShield will start the driver installation process to copy the related material to the indicated directory and register the driver on your computer. The driver target directory is as below for different systems.

Windows NT/2000/XP :
The PIODIO.DLL will be copied onto C:\WINNT\SYSTEM32.
The NAPWNT.SYS and PIO.SYS will be copied into
C:\WINNT\SYSTEM32\DRIVERS.

Windows 95/98/Me :
The PIODIO.DLL,and PIODIO.Vxd will be copied onto C:\Windows\SYSTEM.

4.2 PnP Driver Installation

After installing the hardware (PIO-D48) and you turn the power on for your PC, Windows 98/Me/2000/XP will find a PCI card device and then ask the user to provide a PIODIO.inf to install the hardware driver onto the computer. If the user has trouble preceding through this process, please refer to PnPinstall.pdf for more information.

5. DLL Function Description

The DLL driver is the collection of function calls on the PIO-DIO card for the Windows 98/Me/NT/2000/XP system. The application structure is presented in following figure. The user application program was developed by designated tools such as VB, Delphi and Borland C^{++} Builder which can call on the PIODIO.DLL driver in the user mode. Following that the DLL driver will call up PIO.sys to access the hardware system.

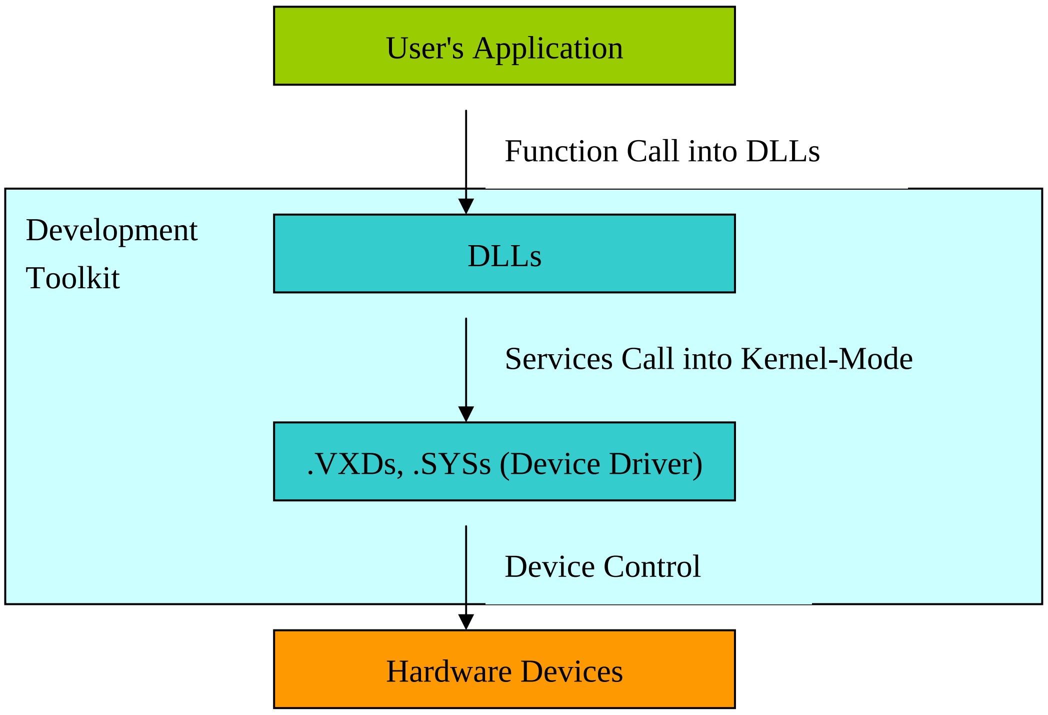


Figure 5.1

5.1 Table of ErrorCodes and ErrorStrings

Table 5.1

Error Code	Error ID	Error String
0	PIODIO_NoError	OK (No error !)
1	PIODIO_DriverOpenError	Device driver can not be opened
2	PIODIO_DriverNoOpen	Users have to call the DriverInit function first
3	PIODIO_GetDriverVersionError	Get driver version error
4	PIODIO_InstallIrqError	Install IRQ Error
5	PIODIO_ClearIntCountError	Clear counter value Error
6	PIODIO_GetIntCountError	Get counter interrupt error
7	PIODIO_RemoveIrqError	Remove IRQ Error
8	PIODIO_FindBoardError	Can not find board
9	PIODIO_ExceedBoardNumber	The Max. board is: 8
10	PIODIO_ResetError	Can't reset interrupt count
11	PIODIO_IrqMaskError	Irq-Mask is 1,2,4,8 or 1 to 0xF
12	PIODIO_ActiveModeError	Active-Mode is 1,2 or 1 to 3
13	PIODIO_GetActiveFlagError	Can’t get interrupt active flag
14	PIODIO_ActiveFlagEndOfQueue	The flag queue is empty

5.2 Function Descriptions

All of the functions provided for the PIO-D48 are listed below with more detailed information for every function presented in the following section. However, in order to make the description more simple and clear, the attributes for the input and output parameters of the function are indicated as [input] and [output] respectively, as shown in following table.

Table 5.2

Keyword	Setting parameter by user before calling this function?	Get the data/value from this parameter after calling this function ?
[Input]	Yes	No
[Output]	No	Yes
[Input, Output]	Yes	Yes

Table 5.3

Return Type	Function Definition
float	PIODIO_FloatSub(float fA, float fB);
short	PIODIO_ShortSub(short nA, short nB);
WORD	PIODIO_GetDllVersion(void);
WORD	PIODIO_DriverInit(void);
void	PIODIO_DriverClose(void);
WORD	PIODIO_SearchCard(WORD *wBoards, DWORD dwPIOCardID);
WORD	PIODIO_GetDriverVersion(WORD *wDriverVersion);
WORD	PIODIO_GetConfigAddressSpace(WORD wBoardNo, DWORD *wAddrBase, WORD *wIrqNo, WORD *wSubVendor, WORD *wSubDevice,WORD *wSubAux, WORD *wSlotBus,WORD *wSlotDevice);
WORD	PIODIO_ActiveBoard(WORD wBoardNo);
WORD	PIODIO_WhichBoardActive(void);
void	PIODIO_OutputWord(DWORD wPortAddress, DWORD wOutData);
void	PIODIO_OutputByte(DWORD wPortAddr, WORD bOutputValue);
DWORD	PIODIO_InputWord(DWORD wPortAddress);
WORD	PIODIO_InputByte(DWORD wPortAddr);
WORD	PIODIO_IntInstall(WORD wBoardNo, HANDLE *hEvent, WORD wInterruptSource, WORD wActiveMode);
WORD	PIODIO_IntRemove(void);
WORD	PIODIO_IntResetCount(void);
WORD	PIODIO_IntGetCount(DWORD *dwIntCount);
WORD	PIOD48_IntInstall(WORD wBoardNo, HANDLE *hEvent, WORD wIrqMask, WORD wActiveMode);
WORD	PIOD48_IntRemove(void);
WORD	PIOD48_IntGetCount(DWORD *dwIntCount);
WORD	PIOD48_IntGetActiveFlag (WORD *bActiveHighFlag, WORD *bActiveLowFlag);
DWORD	PIOD48_Freq(DWORD dwBase);
DWORD	PIOD48_FreqA();
void	PIOD48_SetCounter(DWORD dwBase, WORD wCounterNo, WORD bCounterMode, DWORD wCounterValue);
DWORD	PIOD48_ReadCounter (DWORD dwBase, WORD wCounterNo, WORD bCounterMode);
void	PIOD48_SetCounterA(WORD wCounterNo, WORD bCounterMode, DWORD wCounterValue);
DWORD	PIOD48_ReadCounterA(WORD wCounterNo, WORD bCounterMode);

5.3 FUNCTIONS OF TEST

5.3.1 PIODIO_GetDllVersion

- **Description:**
  To get the version number of PIODIO.DLL driver
- **Syntax:**
  WORD PIODIO_GetDllVersion(Void)
- **Parameter:**
  None
- **Return:**
  200(hex) for version 2.00

5.3.2 PIODIO_ShortSub

- **Description:**
  To perform the subtraction as nA - nB in short data type. This function is provided for testing DLL linkage purpose.
- **Syntax:**
  short PIODIO_ShortSub(short nA, short nB)
- **Parameter:**

nA	:[Input] 2 bytes short data type value
nB	:[Input] 2 bytes short data type value

- **Return:**
  The value of nA – nB

5.3.3 PIODIO_FloatSub

- **Description:**
  To perform the subtraction as fA - fB in float data type. This function is provided for testing DLL linkage purpose.
- **Syntax:**
  float PIODIO_FloatSub(float fA, float fB)
- **Parameter:**

fA	: [Input] 4 bytes floating point value
fB	: [Input] 4 bytes floating point value

- **Return:** The value of fA - fB

5.4 Digital I/O FUNCTIONS

5.4.1 *PIODIO_OutputByte*

- **Description :**

  Send the 8 bits data to the specified I/O port.

- **Syntax :**

  void PIODIO_OutputByte(DWORD wPortAddr, WORD bOutputVal);

- **Parameter :**

WPortAddr	: [Input]	I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace. Only the low WORD is valid.
bOutputVal	: [Input]	8 bit data send to I/O port. Only the low BYTE is valid.

- **Return:**

  None

5.4.2 *PIODIO_InputByte*

- **Description :**

  Read the 8 bits data from the specified I/O port.

- **Syntax :**

  WORD PIODIO_InputByte(DWORD wPortAddr);

- **Parameter :**

  wPortAddr: [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace().
  Only the low WORD is valid.

- **Return:**

  16 bits data with the leading 8 bits are all 0.
  (Only the low BYTE is valid.)

5.4.3 PIODIO_OutputWord

- **Description :**

  Send the 16 bits data to the specified I/O port.
- **Syntax :**

  void PIODIO_OutputWord(DWORD wPortAddr, DWORD wOutputVal);
- **Parameter :**

WPortAddr	: [Input]	I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace(). Only the low WORD is valid.
WOutputVal	: [Input]	16-bit data send to I/O port. Only the low WORD is valid.

- **Return:**

  None

5.4.4 PIODIO_InputWord

- **Description :**

  Obtain the 16 bits data from the specified I/O port.
- **Syntax :**

  DWORD PIODIO_InputWord(DWORD wPortAddr);
- **Parameter :**

wPortAddr	: [Input]	I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace(). Only the low WORD is valid.

- **Return:**

  16-bit data. Only the low WORD is valid.

5.5 Driver Relative Functions

5.5.1 PIODIO_GetDriverVersion

- **Description :**

  Obtain the version number information from the PIODIO driver.
- **Syntax :**

  WORD PIODIO_GetDriverVersion(WORD *wDriverVersion);
- **Parameter :**

  wDriverVersion : [Output] address of wDriverVersion
- **Return:**

  Please refer to "Section 5.1 Error Code".

5.5.2 PIODIO_DriverInit

- **Description :**

  This subroutine opens the PIODIO driver and allocates the computer resource for the device. This function must be called once before applying other PIODIO functions.
- **Syntax :**

  WORD PIODIO_DriverInit();
- **Parameter :**

  None
- **Return:**

  Please refer to "Section 5.1 Error Code".

5.5.3 PIODIO_SearchCard

- **Description :**

  This subroutine will search the card and get the board total. This function must be called once before applying other PIODIO functions.

- **Syntax :**

  WORD PIODIO_SearchCard(WORD *wBoards, DWORD dwPIOCardID);

- **Parameter :**

wBoards	:[Output] Number of boards found in this PC
DwPIOCardID	:[Input] Sub vendor, sub device and sub aux id of the board to find. Please refer to chapter 3.1.

  NOTE :

  Different versions of PIO-D48 boards may have different Sub IDs. This function will find the total amount of PIO-D48 boards which will include all versions, so it does not matter what version Sub ID you input. We have demonstrated an example below:

  wRtn=PIODIO_SearchCard(&wBoards, 0x800130);

  You will get the total number of PIO-D48 boards including all versions of the board in your PC.

- **Return:**

  Please refer to "Section 5.1 Error Code"

5.5.4 PIODIO_DriverClose

- **Description :**

  This subroutine closes the PIODIO Driver and releases this resource from the computers device resources. This function must be executed once before exiting the user's application.

- **Syntax :**

  void PIODIO_DriverClose();

- **Parameter :**

  None

- **Return:**

  None

5.5.5 PIODIO_GetConfigAddressSpace

- **Description :**

  Obtain the I/O address and other information for the PIODIO board.

- **Syntax :**

  WORD PIODIO_GetConfigAddressSpace(WORD wBoardNo, DWORD
  *wAddrBase, WORD *wIrqNo, WORD *wSubVendor, WORD
  *wSubDevice, WORD *wSubAux, WORD *wSlotBus,WORD
  *wSlotDevice);

- **Parameter :**

wBoardNo	: [Input]	PIODIO board number
wAddrBase	: [Output]	The base address of the PIODIO board. Only the low WORD is valid.
wIrqNo	: [Output]	The IRQ number that the board is using.
wSubVendor	: [Output]	Sub Vendor ID.
wSubDevice	: [Output]	Sub Device ID.
wSubAux	: [Output]	Sub Aux ID.
wSlotBus	: [Output]	Slot Bus number.
wSlotDevice	: [Output]	Slot Device ID.

- **Return:**

  Please refer to "Section 5.1 Error Code".

5.6 PIODIO INTERRUPT FUNCTION

5.6.1 PIODIO_IntResetCount

- **Description:**
  This function will clear the counter value on the device driver for the interrupt.
- **Syntax:**
  WORD PIODIO_IntResetCount(void);
- **Parameter:**
  None
- **Return:**
  Please refer to "Section 5.1 Error Code".

5.6.2 PIODIO_IntGetCount

- **Description:**
  This subroutine will read the counter value of the interrupt defined in the device driver.
- **Syntax :**
  WORD PIODIO_IntGetCount(DWORD *dwIntCount);
- **Parameter:**
  dwIntCount : [Output] Address of dwIntCount, which will stores the counter value of interrupt.
- **Return:**
  Please refer to "Section 5.1 Error Code".

5.6.3 *PIODIO_IntInstall*

- **Description:**

  This subroutine installs the IRQ service routine.

- **Syntax:**

  WORD PIODIO_IntInstall(WORD wBoardNo, HANDLE *hEvent,
  WORD wInterruptSource, WORD wActiveMode);

- **Parameter:**

wBoardNo	: [Input] Which board to be used.
hEvent	: [Input] Address of an Event handle. The user's program must call up the Windows API function "CreateEvent()" to create the event-object.
wInterruptSource	: [Input] What the Interrupt-Source to be used ? Please refer to the following table.

Card No.	wInterruptSource	Description
PIO-D48	0	PC3/PC7 from Port-2
	1	PC3/PC7 from Port-5
	2	Cout0
	3	Cout2

wActiveMode	: [Input] When to trigger the interrupt ?
	0 → PIODIO_ActiveLow
	1 → PIODIO_ActiveHigh

- **Return:**

  Please refer to "Section 5.1 Error Code".

5.6.4 *PIODIO_IntRemove*

- **Description:**

  This subroutine removes the IRQ service routine.

- **Syntax:**

  WORD PIODIO_IntRemove(void);

- **Parameter:**

  None

- **Return:**

  Please refer to "Section 5.1 Error Code".

5.7 PIO-D48 INTERRUPT FUNCTION

The largest difference between the PIO-DIO and PIO-D48 interrupt functions is that PIO-DIO supports only one interrupt-source at a time whereas PIO-D48 supports 4 interrupt-sources at a time.

5.7.1 PIOD48_IntInstall

- **Description:**

  This subroutine will install the IRQ service routine. This function can support the multiple interrupt-source and the Active-Mode which can be set to either "Active-Low only", "Active-High only" or "Active-Low or Active-High".

- **Syntax:**

  WORD PIOD48_IntInstall(WORD wBoardNo, HANDLE *hEvent,
  WORD wIrqMask, WORD wActiveMode);

- **Parameter:**

  wBoardNo : [Input] Which board to use.

  hEvent :[Input] Address of the Event handle. The user's program must call the Windows API function "CreateEvent()" to create the event-object.

  wIrqMask :[Input] Which the Interrupt-Source should be used ? Please refer to the hardware manual for more detailed information.

wIrqMask	Description
1	INT_CHAN_0: PC3/PC7 from Port-2
2	INT_CHAN_1: PC3/PC7 from Port-5
4	INT_CHAN_2: Cout0
8	INT_CHAN_3: Cout2

  This function supports 4 interrupt-sources at a time, therefore users can use multiple interrupt-source like 1 +2 +4 +8.

  wActiveMode : [Input] When will the ISR service the interrupt?

wActiveMode	Description
1	PIOD48_ActiveLow (The interrupt is occurred when the Interrupt-Source status is low).
2	PIOD48_ActiveHigh (The interrupt is occurred when the Interrupt-Source status is high).

  This can be 1 (Active- Low), 2(Active- High) or 1 + 2 (Both of the High and Low will active the interrupt).

- **Return:** Please refer to "Section 5.1 Error Code".

5.7.2 PIOD48_IntRemove

- **Description:**

  This subroutine will remove the IRQ service routine.
- **Syntax:**

  WORD PIOD48_IntRemove(void);
- **Parameter:**

  None
- **Return:**

  Please refer to "Section 5.1 Error Code".

5.7.3 PIOD48_IntGetCount

- **Description:**

  This subroutine will read the **Interrupt-Counter** value on the device driver. The Interrupt-Counter will be increased (in the ISR) when the interrupt is triggered. When the interrupt is set to Active-High only or Active-Low only, some of the interrupt signals will be ignored and the Interrupt-Counter will not increase.
- **Syntax :**

  WORD PIOD48_IntGetCount(DWORD *dwIntCount);
- **Parameter:**

dwIntCount	: [Output] Address of dwIntCount, which will store the interrupt counter values.
- **Return:**

  Please refer to "Section 5.1 Error Code".

5.7.4 PIOD48_IntGetActiveFlag

- **Description:**

  This subroutine will read the Active-High and Active-Low flag from the device driver's memory queues (First-in-First-out, Buffer Size: 2000 flags for High/Low).

  The Active-Flag is used to record the Active-State-change for the interrupt-source when an interrupt occurs. The Active-High-Flag records which interrupt-source changed to a high state and the Active-Low-Flag records which interrupt-source changed to a low state. Users can use these flags to indicate which interrupt-source has changed.

  If the Active-Mode is set to Active-Low(/Active-High) only and the state for the Active-Low(/Active-High) is equal to zero, then the ISR will not increase the interrupt-counter, and the Active-Flag for High and Low will not be recorded.

  If users do not utilize this function to retrieve the flags from the device driver's memory queues, these queues will stop recording the flags actions(lost data) when the buffer is full. But the interrupt-counter will still carry on counting while the ISR services the interrupt.

- **Syntax :**

  WORD PIOD48_IntGetActiveFlag(WORD *bActiveHighFlag, WORD *bActiveLowFlag);

- **Parameter:**

  bActiveHighFlag : [Output] Returns a flag that indicates which interrupt-source changed to a High-State.

  bActiveLowFlag : [Output] Returns a flag that indicates which interrupt-source changed to a Low-State.

- **Return:**

  Please refer to "Section 5.1 Error Code".

5.7.5 Architecture of Interrupt mode

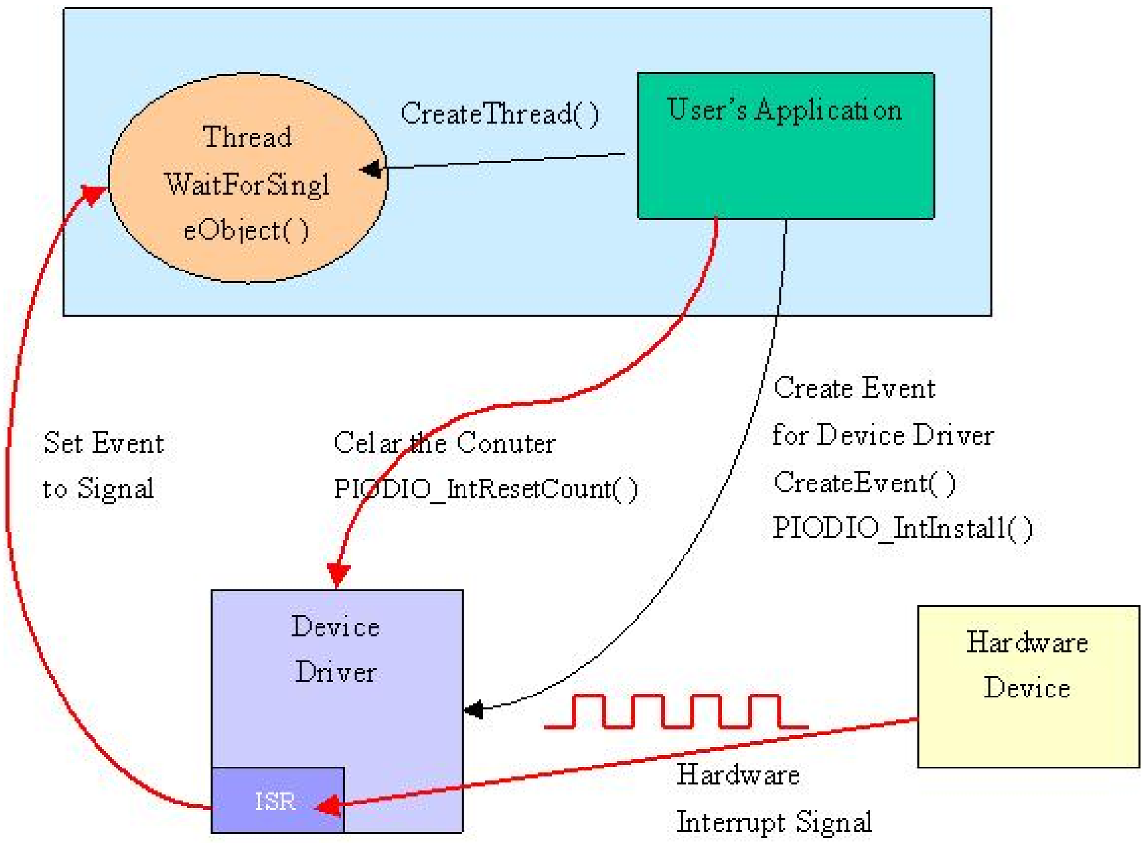


Figure 5.2

5.8 COUNTER FUNCTION

5.8.1 *PIOD48_SetCounter*

- **Description :**
  This subroutine is used to set the 8254 counter's mode and value.
- **Syntax :**
  void PIOD48_SetCounter(DWORD dwBase, WORD wCounterNo,
  WORD bCounterMode, DWORD wCounterValue);
- **Parameter :**
  dwBase : [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace. Only the low WORD is valid.
  wCounterNo : [Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)
  wCounterMode : [Input] The 8254 Counter-Mode: 0 to 5.
  wCounterValue : [Input] The 16 bits value for the counter to count. Only the lower WORD is valid.
- **Return:** None

5.8.2 *PIOD48_ReadCounter*

- **Description :**
  This subroutine is used to obtain the 8254 counter's value.
- **Syntax :**
  DWORD PIOD48_ReadCounter (DWORD dwBase, WORD wCounterNo,
  WORD bCounterMode);
- **Parameter :**
  dwBase : [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace. Only the low WORD is valid.
  wCounterNo : [Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)
  wCounterMode: [Input] The 8254 Counter-Mode: 0 to 5.
- **Return:** 16 bits counter's value. (Only the lower WORD is valid.)

5.8.3 *PIOD48_SetCounterA*

- **Description :**

  This subroutine is used to set the 8254 counter's mode and value. Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax :**

  void PIOD48_SetCounterA(WORD wCounterNo, WORD bCounterMode,
  DWORD wCounterValue);

- **Parameter :**

wCounterNo	: [Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)
wCounterMode	: [Input] The 8254 Counter-Mode: 0 to 5.
wCounterValue	: [Input] The 16 bits value for the counter to count. Only the lower WORD is valid.

- **Return:**

  None

5.8.4 *PIOD48_ReadCounterA*

- **Description :**

  This subroutine is used to obtain the 8254 counter's value. Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax :**

  DWORD PIOD48_ReadCounterA(WORD wCounterNo, WORD
  bCounterMode);

- **Parameter :**

wCounterNo	: [Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)
wCounterMode	: [Input] The 8254 Counter-Mode: 0 to 5.

- **Return:**

  Returns the 16 bits counter's value. (Only the lower WORD is valid.)

5.9 FREQUENCY FUNCTION

5.9.1 PIOD48_Freq

- **Description :**

  This subroutine is used to measure signal frequency. Users have to connect the signal(+) with CN1.Pin29, and connect the signal(-) with CN1.Pin19.

  It will use Counter-0 and Counter-1 to measure frequency, thus users shouldn't use Counter-0 and Counter-1 for any other purposes.

- **Syntax :**

  DWORD PIOD48_Freq(DWORD dwBase);

- **Parameter :**

dwBase	: [Input]	I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace. Only the low WORD is valid.

- **Return:**

  Returns the frequency value. (Only the low WORD is valid.)

5.9.2 PIOD48_FreqA

- **Description :**

  Please refer to the description for the "PIOD48_Freq()" function. Users have to utilize the "PIODIO_ActiveBoard()" function before calling this function.

- **Syntax :**

  DWORD PIOD48_FreqA();

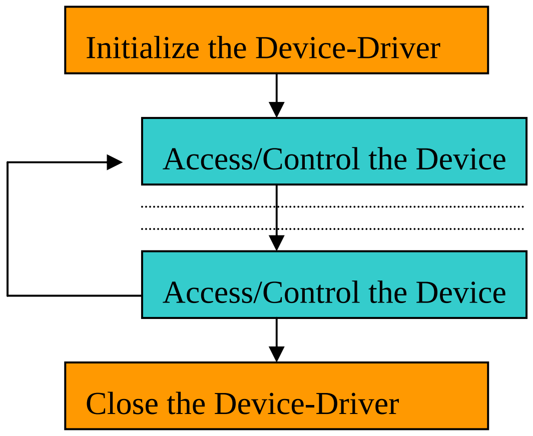
- **Parameter :**

  None

- **Return:**

  Returns the frequency value. (Only the low WORD is valid.)

5.10 Program Architecture



Initialize the Device-Driver
Access/Control the Device
Access/Control the Device
Close the Device-Driver

```
PIODIO_DriverInit()
Enable All DI/DO
….
    PIODIO_InputByte( … )
    ……..
    ……..
    PIODIO_OutputByte(…)
…..
```

Figure 5.3

6. Demo Programs for Windows

All demo programs will not work properly if the DLL driver has not been installed correctly. During the DLL driver installation process , the install-shields will register the correct kernel driver to the operation system and copy the DLL driver and demo programs to the correct position based on the driver software package you have selected (Win98,Me,NT,win2000,XP). Once driver installation is complete, the related demo programs and development library and declaration header files for different development environments will be presented as follows.

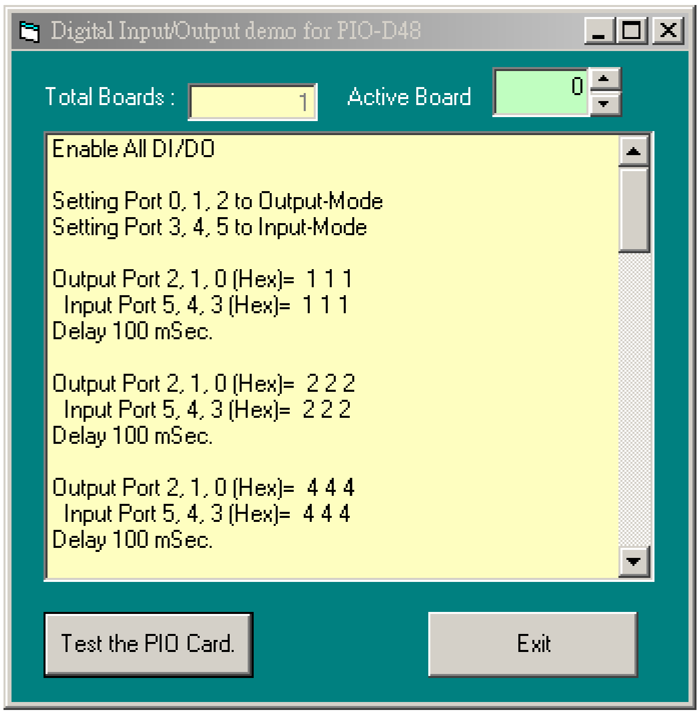
```
|--\Demo                          → demo program
   |--\BCB3                       → for Borland C++ Builder 3
   |    |--\PIODIO.H              → Header file
   |       \ PIODIO.LIB           → Linkage library for BCB only
   |
   |--\Delphi3                    → for Delphi3
   |    |--\ PIODIO.PAS           → Declaration file
   |
   |--\VB6                        → for Visual Basic 6
      |--\ PIODIO.BAS             →Declaration file
```

The list of demo programs :

DIO	: Digital input and output of port0/1/2 and Port3/4/5.
Freq	: Measure frequency of external signal from PC0 of port-2
INT	: Interrupt of P2C3 or P5C3
INT2	: Interrupt of Counter0
INT3	: Interrupt of Counter1 and Counter2
INT4	: Interrupt of four interrupt sources at a time

6.1 Digital intput and output (DIO)

This demo program is used to check the digital output and digital input status for Port0/1/2 and Port3/4/5.

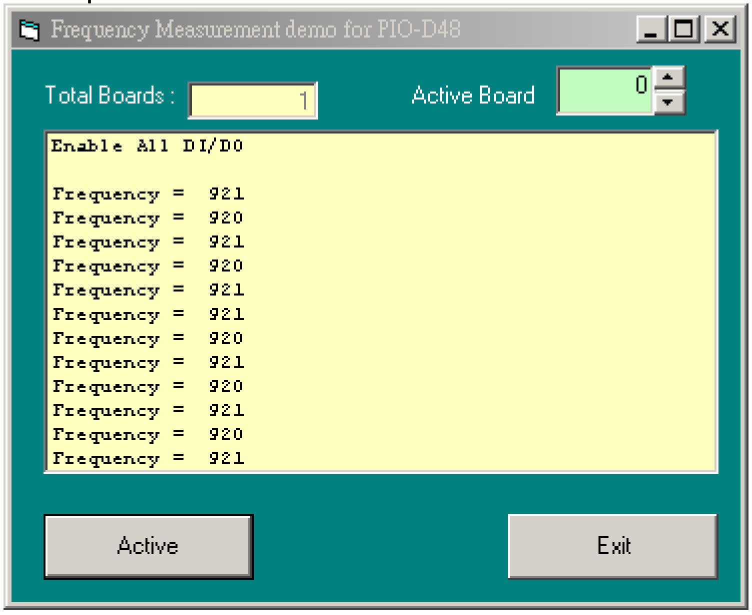


```
Enable All DI/DO

Setting Port 0, 1, 2 to Output-Mode
Setting Port 3, 4, 5 to Input-Mode

Output Port 2, 1, 0 (Hex)=  1 1 1
 Input Port 5, 4, 3 (Hex)=  1 1 1
Delay 100 mSec.

Output Port 2, 1, 0 (Hex)=  2 2 2
 Input Port 5, 4, 3 (Hex)=  2 2 2
Delay 100 mSec.

Output Port 2, 1, 0 (Hex)=  4 4 4
 Input Port 5, 4, 3 (Hex)=  4 4 4
Delay 100 mSec.
```

Figure 6.1 DIO demo

6.2 Frequency measure (Freq)

This demo program is used to measure the frequency of external signals from the PC0 on port2.

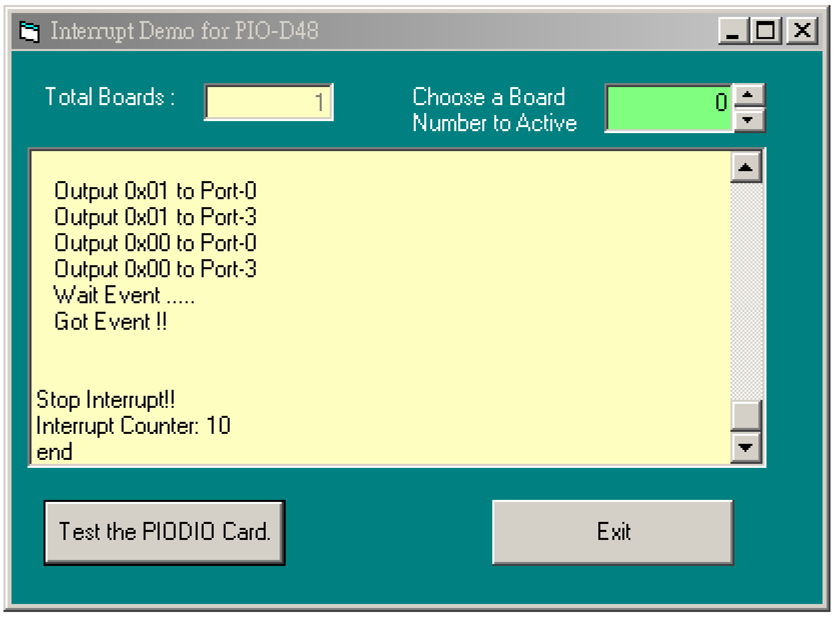


```
Enable All DI/DO

Frequency =  921
Frequency =  920
Frequency =  921
Frequency =  920
Frequency =  921
Frequency =  921
Frequency =  920
Frequency =  921
Frequency =  920
Frequency =  921
Frequency =  920
Frequency =  921
```

Figure 6.2 Frequency measure demo

6.3 Interrupt (Int, Int2, Int3, Int4)

- Int : This demo program uses P2C3 or P5C3 to generate the interrupt-trigger.
- Int2 : This demo program uses the Counter0 to generate the interrupt-trigger. Please apply the external signal into P2C0, and connect the external GND with CN1.GND.
- Int3 : This demo program uses Counter1 and Counter2 to generate the interrupt-trigger.
- Int4 : This demo program can use four interrupt sources at a time to generate the interrupt-trigger.



```
 Output 0x01 to Port-0
 Output 0x01 to Port-3
 Output 0x00 to Port-0
 Output 0x00 to Port-3
 Wait Event .....
 Got Event !!

Stop Interrupt!!
Interrupt Counter: 10
end
```

Figure 6.3 Interrupt demo

Appendix

Appendix A. Related DOS Software

A-1 Where the related software is

The related DOS software and demos in the CD is given as follows:

• \TC*.*	→ for Turbo C 2.xx or above
• \MSC*.*	→ for MSC 5.xx or above
• \BC*.*	→ for BC 3.xx or above
• \TC\LIB\*.*	→ for TC library
• \TC\DEMO\*.*	→ for TC demo program
• \TC\DIAG\*.*	→ for TC diagnostic program
• \TC\LIB\PIO.H	→ TC declaration file
• \TC\\LIB\TCPIO_L.LIB	→ TC large model library file
• \TC\\LIB\TCPIO_H.LIB	→ TC huge model library file
• \MSC\LIB\PIO.H	→ MSC declaration file
• \MSC\LIB\MSCPIO_L.LIB	→ MSC large model library file
• \MSC\\LIB\MSCPIO_H.LIB	→ MSC huge model library file
• \BC\LIB\PIO.H	→ BC declaration file
• \BC\LIB\BCPIO_L.LIB	→ BC large model library file
• \BC\\LIB\BCPIO_H.LIB	→ BC huge model library file

The list of demo programs :

demo1 : D/O demo of CN1 and CN2.
demo2 : D/I demo of CN1 and CN2.
demo3 : D/I/O demo of CN1 and CN2.
demo4 : INT_CHAN_3, timer interrupt.
demo5 :INT_CHAN_2, 16-bit event counter (no interrupt), init_HIGH & active_LOW signal to PC0 of port-2 .

demo6 :INT_CHAN_2, 16-bit event counter (no interrupt), init_LOW & active_HIGH signal to PC0 of port- 2.

demo7 :INT_CHAN_2, 16-bit down-counter (using interrupt), init_HIGH & active_LOW signal to PC0 of port-2 .

demo8 :INT_CHAN_0, interrupt demo, init_HIGH & active_LOW signal to PC3 of port-2 (note: PC7 of port-2 is don't care).

demo9 :INT_CHAN_0, interrupt demo, init_HIGH & active_LOW signal to PC3 of port-2 (note: The PC7 of port_2 is used to enable the interrupt).

demo10 :INT_CHAN_1, interrupt demo, init_HIGH & active_LOW signal to PC3 of port-5 (note:The PC7 of port_5 is used to enable the interrupt)

demo11 :INT_CHAN_0 & INT_CHAN_1 interrupt demo, init_HIGH & active_LOW signal to PC3 of port-2(port-5), (note: PC7 of port-2(port-5) is don't care).

A-2 DOS LIB Function

A-2-1 *Table of ErrorCode and ErrorString*

Table A.1 ErrorCode and ErrorString

Error Code	Error ID	Error String
0	NoError	OK ! No Error!
1	DriverHandleError	Device driver opened error
2	DriverCallError	Got the error while calling the driver functions
3	FindBoardError	Can't find the board on the system
4	TimeOut	Timeout
5	ExceedBoardNumber	Invalidate board number (Valid range: 0 to TotalBoards -1)
6	NotFoundBoard	Can't detect the board on the system

A-2-2 *PIO_DriverInit*

- **Description :**

  This function can detect all PIO/PISO series card in the system. It is implemented based on the PCI Plug & Play mechanism-1. It will find all PIO/PISO series cards installed in this system and save all their resources in the library.

- **Syntax :**

  ```
  WORD  PIO_DriverInit(WORD *wBoards, WORD wSubVendorID,
                                WORD wSubDeviceID,WORD wSubAuxID)
  ```

- **Parameter :**

WBoards	: [Output] Number of boards found in this PC
wSubVendor	: [Input] SubVendor ID of the board
wSubDevice	: [Input] SubDevice ID of the board
wSubAux	: [Input] SubAux ID of the board

- **Return:**

  Please refer to " Table A.1".

A-2-3 *PIO_GetDriverVersion*

- **Description :**

  This subroutine will obtain the version number of PIODIO driver.

- **Syntax :**

  ```
  WORD  PIO_GetDriverVersion(WORD *wDriverVersion)
  ```

- **Parameter :**

  wDriverVersion : [Output] Address of wDriverVersion

- **Return:**

  Please refer to " Table A.1".

A-2-4 *PIO_GetConfigAddressSpace*

- **Description :**

  The user can use this function to save the resources found on all the PIO/PISO cards installed on the system. Then the application program can control all the functions on the PIO/PISO series cards directly.
- **Syntax :**

  WORD PIO_GetConfigAddressSpace(wBoardNo,*wBase,*wIrq,
  wSubVendor, *wSubDevice,*wSubAux,*wSlotBus,*wSlotDevice)
- **Parameter :**

wBoardNo	: [Input]	Board number
wBase	: [Output]	The base address of the board
wIrq	: [Output]	The IRQ number that the board using.
wSubVendor	: [Output]	Sub Vendor ID.
wSubDevice	: [Output]	Sub Device ID.
wSubAux	: [Output]	Sub Aux ID.
wSlotBus	: [Output]	Slot Bus number.
wSlotDevice	: [Output]	Slot Device ID.

- **Return:**

  Please refer to " Table A.1".

A-2-5 *ShowPIOPISO*

- **Description :**

  This function will show a text string for a special Sub_ID. This text string is the same as that defined in PIO.H.
- **Syntax :**

  WORD ShowPIOPISO(wSubVendor, wSubDevice, wSubAux)
- **Parameter :**

wSubVendor	: [Input]	SubVendor ID of the board
wSubDevice	: [Input]	SubDevice ID of the board
wSubAux	: [Input]	SubAux ID of the board.

- **Return:**

  Please refer to " Table A.1".