PIO-D64

User’s Manual

Tables of Contents

1. Introduction

The PIO-D64 provides 32 digital input channels, 32 digital output channels and 6 counter/ timer channels. According to the digital input/output function, we also provide two daughter boards DB-16P and DB-16R for the integration application of two 16-bit input ports and two 16-bit output ports of the PIO-D64 board. DB-16P daughter board offers the digital input isolation purpose by connecting the input ports (CN2, CN4) through it. And DB-16R functions the digital output relay control by interfacing the output ports (CN1, CN3) with it. Besides, there are also four on board clock source 2MHz, 1MHz, 500kHz and 250kHz, which can be selected by jumper setting. Note that the user can also use the clock source from the soldering pad. One chip, 8254, provides 3 channels for frequency measurement, event counting and pulse generation. And the other chip, 8254, provides 3 channels interrupt trigger source. Furthermore, this board also provides a breadboard area for user add-on circuit. It can be installed in a 5V 32-bit PCI slot and is supported with actual “Plug & Play” technology.

1.1 Features

- PCI bus interface;
- 32 digital input channels (strobe control selectable);
- 32 digital output channels;
- Four independent programmable 16-bit timers/counters;
- One 32-bit timer with a 4MHz clock base;
- Provide clock source: 2MHz, 1MHz, 500KHz, 250KHz;
- Interrupt source: 3 channels;
- Breadboard area for add-on circuit;
- Five 20-pin flat cable connectors;

- Connect directly to DB-24PR, 24POR, DB-24C, DB-16P, DB-16R;
- SMD, short card, power saving;
- Automatically detected by Windows 95/98/2000/XP/NT.

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: 24 mA (Max.);

  Source current: 15 mA (Max.);
- Power consumption: +5V @ 580mA ;
- Environment :

  Operating Temperature : 0 to 60°C

  Storage Temperature : -20°C to 80 °C

  Humidity: 0 to 90 % non-condensing
- Dimensions: 156mm x 110mm

1.3 Product Check List

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

- one piece of PIO-D64 card
- one piece of company floppy diskette or CD
- one piece of release note

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

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

Attention:
If any of these items is missed 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.

1.3.1 Product options

- DB-24PR, DB-24PRD: 24 channels power relay board;
- DB-24POR: 24 channels PhotoMos output board;
- DB-24C: 24 channels open-collector output board;
- DB-16P: 16 channels isolated D/I board;
- DB-16R: 16 channels relay board.

2. Hardware configuration

2.1 Board Layout

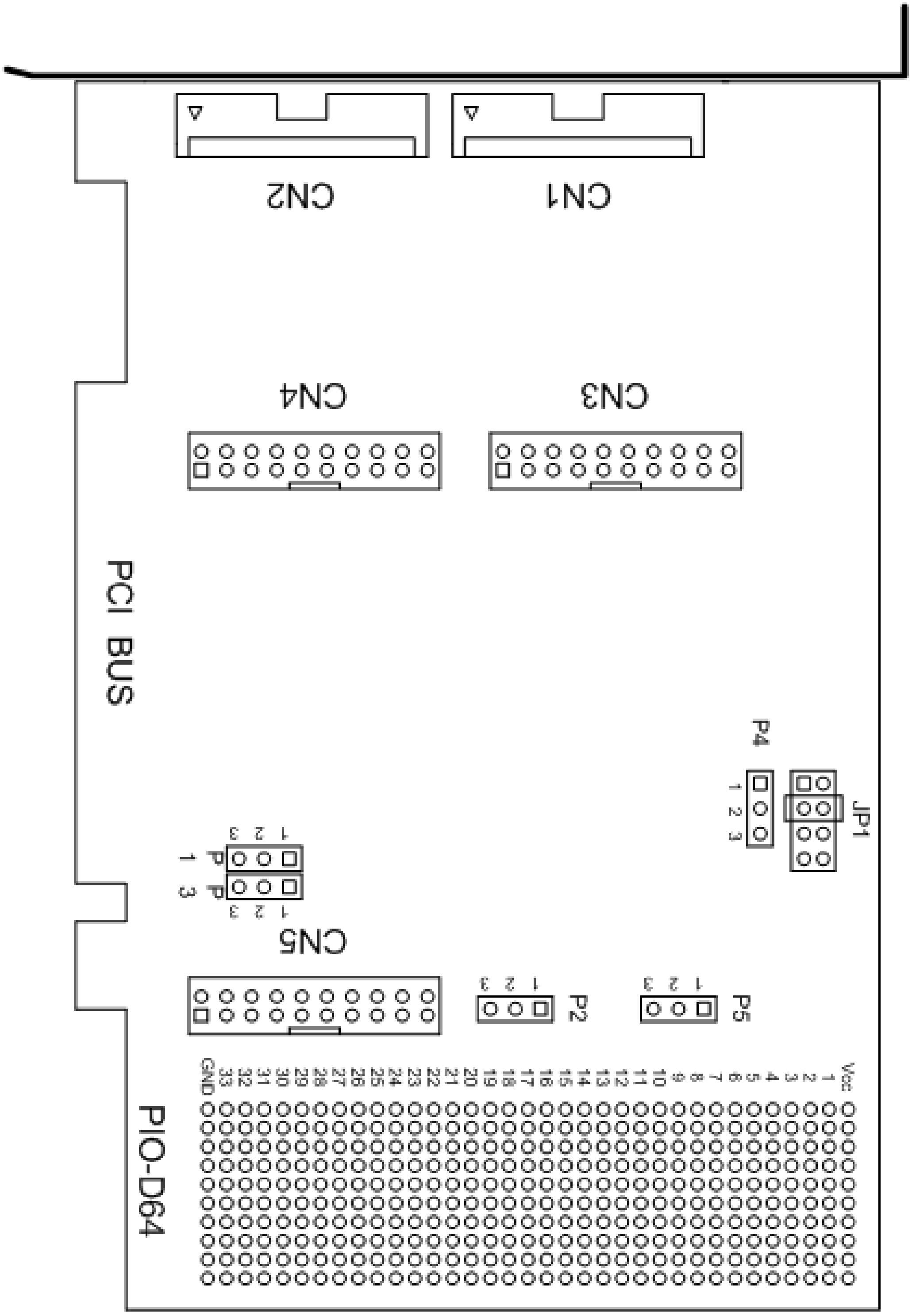


Figure 2.1

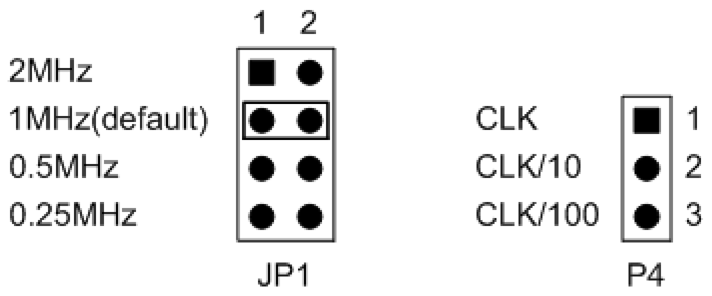
2.2 I/O Port Location

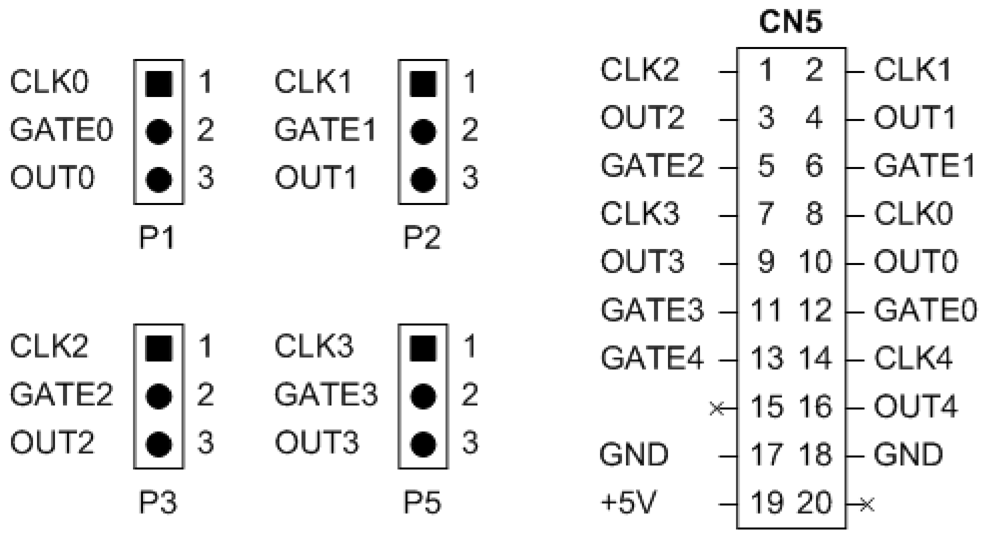
There are two 16-bit digital input ports and two 16-bit digital output ports on the PIO-D64.These I/O port locations are given as below and illustrated in Figure 2.1.

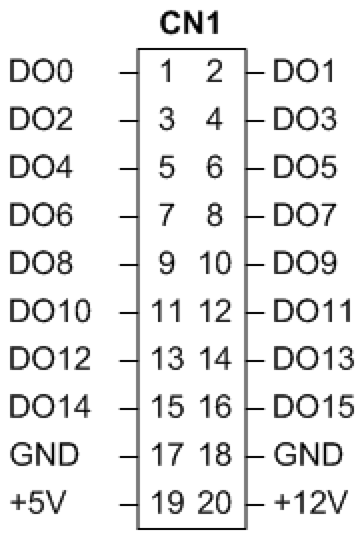
Connector of PIO-D64	Description
CN1	DO0~DO15
CN2	DI0~DI15
CN3	DO16~DO31
CN4	DI16~DI31

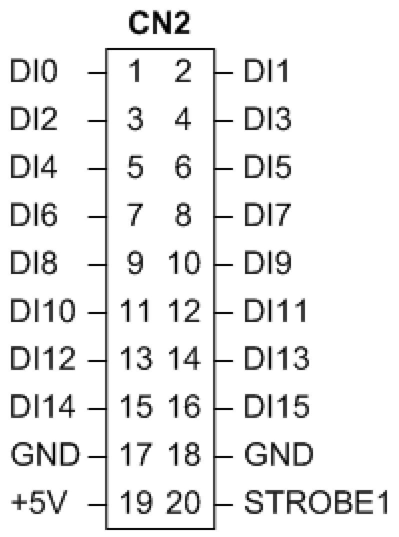
Besides, there is also a connector interface (CN5) for timer and counter function, as shown in Figure 2.1.

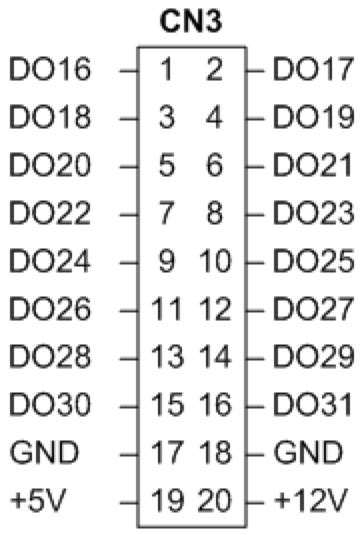
2.3 Pin Assignment

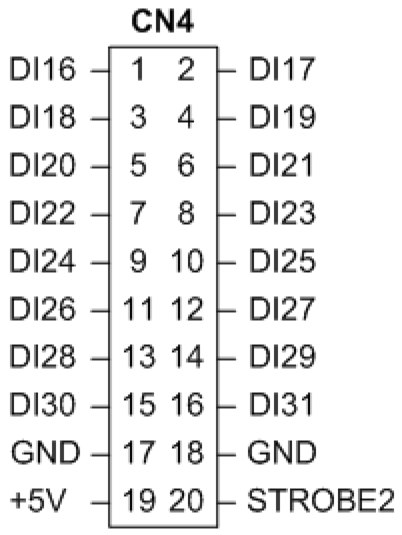












All signals are TTL compatible.

2.4 I/O Operation

2.4.1 DO Port Architecture (CN1 & CN3)

When the PC is power-up, all of DO states are clear to low-state by the RESET\ signal. Fore more information about RESET\ signal, please refer to Sec. 3.1.1. Note that the RESET\ is in Low-state in order to clear all DO states to low level signal. The detail block diagram of DO function is represented as Figure 2.2.

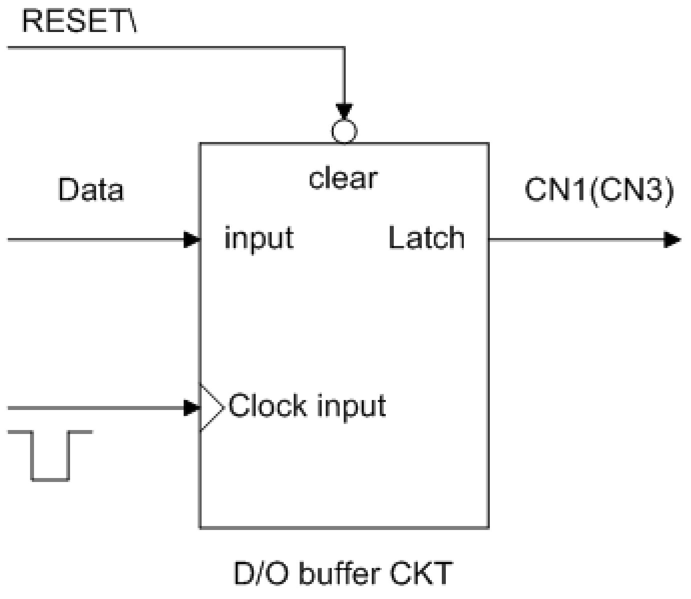


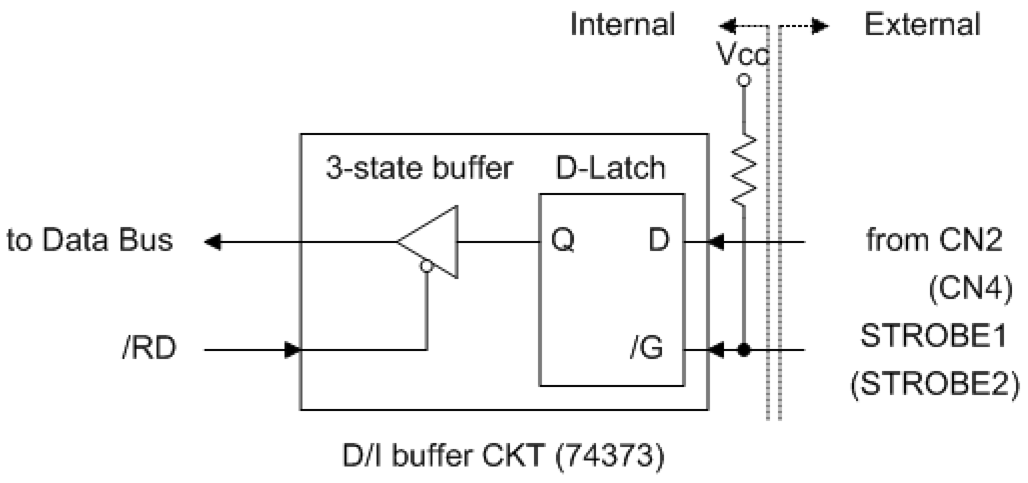
Figure 2.2: Block diagram of DO function

2.4.2 DI Port Architecture (CN2 & CN4)

The enable/disable of DI port is controlled by the RESET\ signal, as depicted as below:

- The RESET\ is in Low-state → all DI operation is disable
- The RESET\ is in High-state → all DI operation is enable

Note that when the PC is power-up, all operation of DI port is disable because RESET\ is in low level. Besides, user may need to latch input data by external strobe single in some application. We provide the following architecture, as shown in Figure 2.3, to allow user to apply the STROBE pin to latch D/I input signal. **<u>If no signal is connected to strobe pin, the input data is transparent.</u>**



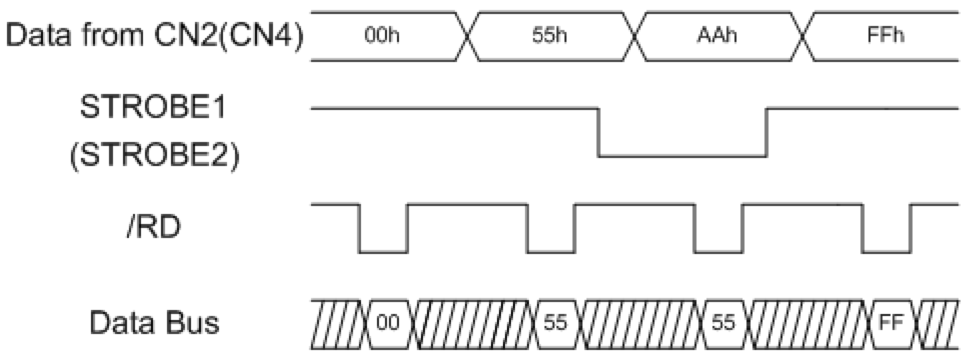


Figure 2.3

2.5 Timer/ Counter Architecture

PIO-D64 has two timer/counter chips, 8254. The first 8254 chip is used as general purpose timer/counter, as shown in Figure 2.4. The pin assignment is presented in Sec.2.3.

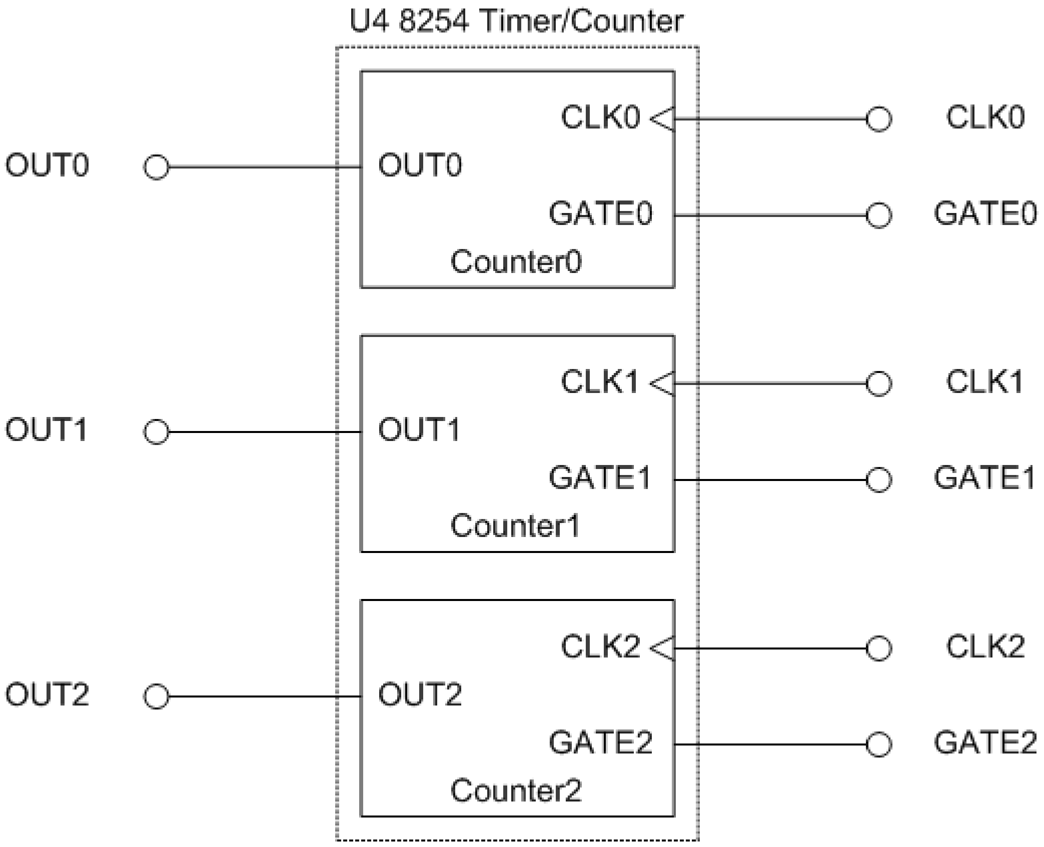


Figure 2.4

The second 8254 chip is used to generate interrupt trigger signals, as shown in Figure 2.5. The Counter3 accept event signal and will generate trigger signal of the interrupt. And the Counter4 and Counter5 are cascaded together, which has clock source 4MHz. It is used to generate pacer timer trigger of the interrupt.

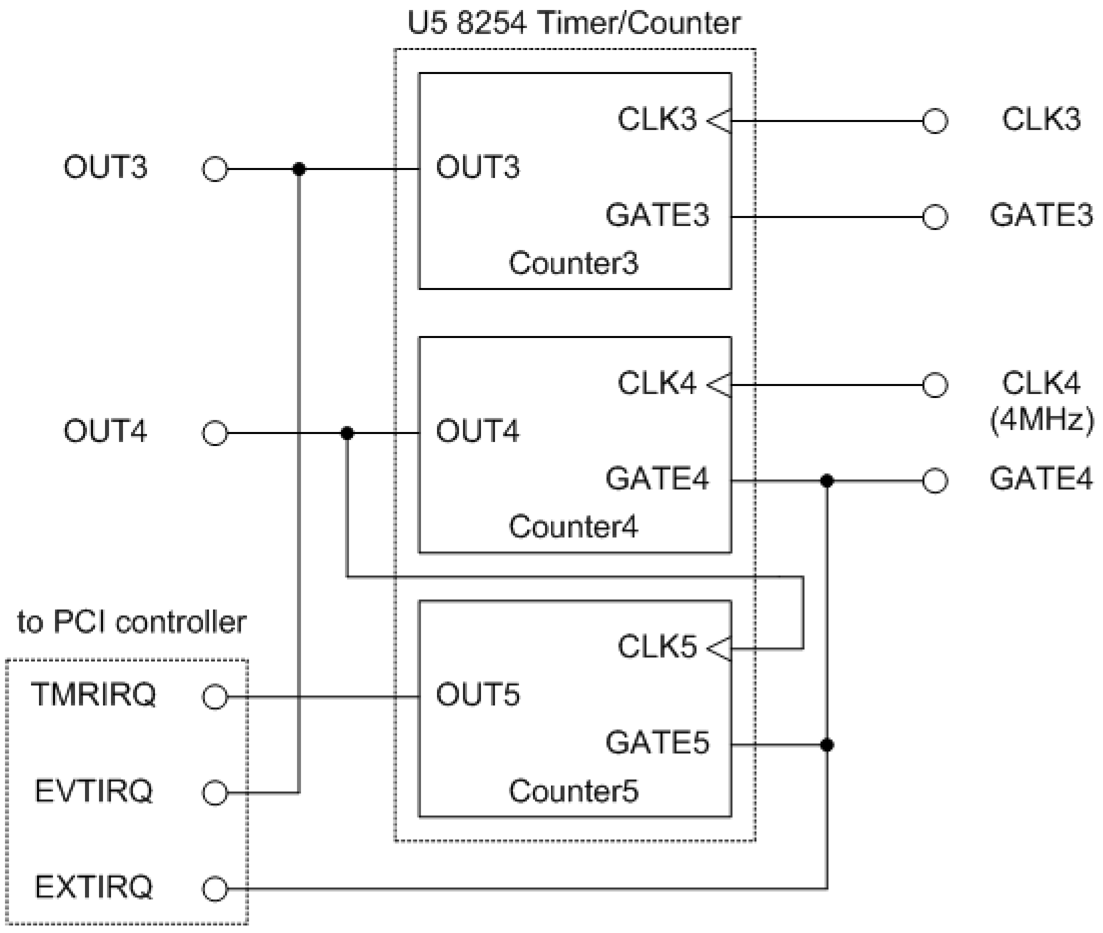
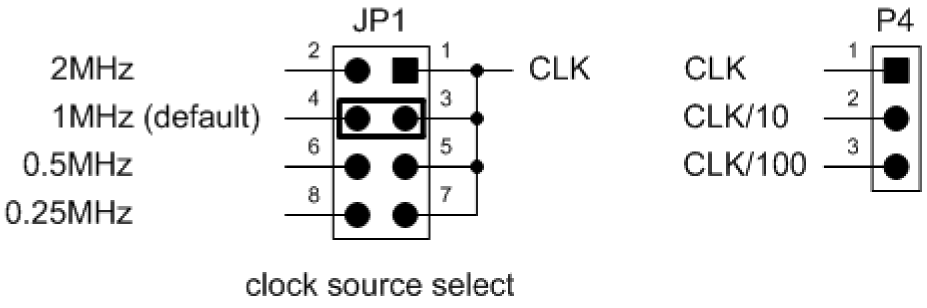


Figure 2.5

Note: Refer to Sec.2.3 for more information about pin assignment.
Refer to Sec.2.7 for more information about operation of interrupt.

2.6 Clock source

The PIO-D64 provides wide range clock source as below table. By jumper setting of JP1, user can select suitable clock output from the corresponding P4 soldering pad.



JP1 setting	P4 soldering pad clock output		
	P4.1	P4.2	P4.3
1-2	2MHz	200KHz	20KHz
3-4 (default)	1MHz	100KHz	10KHz
5-6	500KHz	50KHz	5KHz
7-8	250KHz	25KHz	2.5KHz

2.7 Interrupt Operation

There are three interrupt sources in PIO-D64. These three signals are named as INT_CHAN_0, INT_CHAN_1 and INT_CHAN_2. Their signal sources are given as follows: **(Refer to Sec. 2.5 for the source of interrupt signal)**

INT_CHAN_0: EXTIRQ
INT_CHAN_1: EVTIRQ
INT_CHAN_2: TMRIRQ

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 and DEMO5.C of DOS operating system for more information.

If there are more than one interrupt source, the interrupt service routine has to identify the active signals as follows: (refer to DEMO6.C of DOS operation system)

1. Read the new status of all interrupt signal sources(refer to Sec 3.1.5)
2. Compare the new status with the old status to identify the active signals
3. If INT_CHAN_0 is active, service it
4. If INT_CHAN_1 is active, service it
5. If INT_CHAN_2 is active, service it
6. Update interrupt status

Note that if the interrupt signal is too short, the new status may be as same as old status. In that condition the interrupt service routine can not identify which interrupt source is active. So the interrupt signal must be hold_active long enough until the interrupt service routine is executed. This hold_time is different for different operating system. The hold_time can be as short as micro-second or as long as second. In general, 20ms is enough for all operating system.

2.7.1 Interrupt Block Diagram of PIO-D64

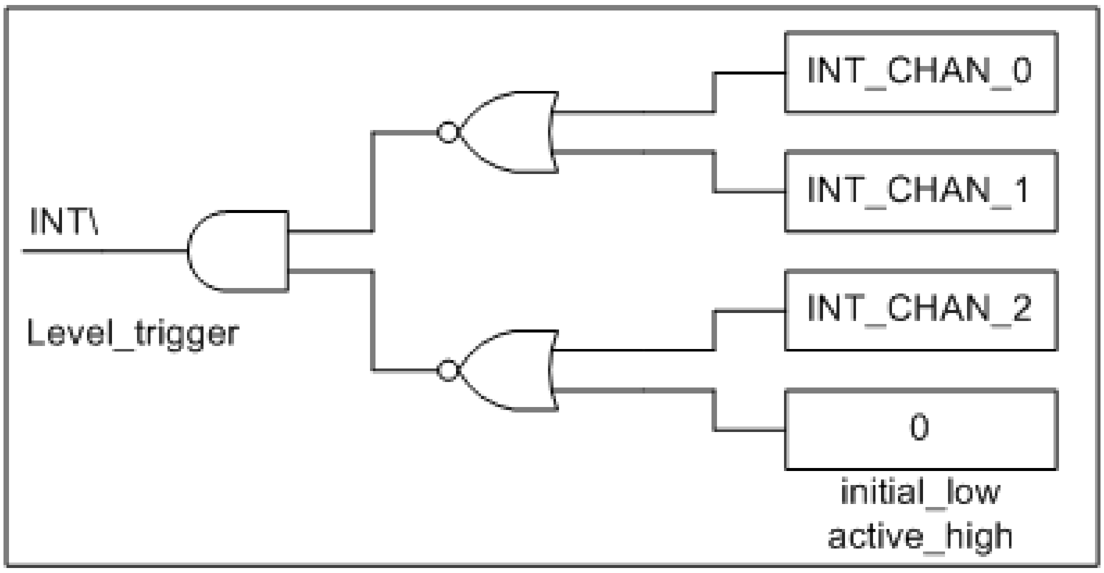


Figure 2.6

The interrupt output signal of PIO-D64, INT\, is **level-trigger & Active_Low**. If the INT\ generates a low-pulse, the PIO-D64 will interrupt the PC once a time. If the INT\ is fixed in low level, the PIO-D64 will interrupt the PC continuously. Therefore, for the normal application, the INT_CHAN_0/1/2 must be controlled in a **pulse_type** signals. That is, **they must be fixed in low level state normally and generate a high_pulse to interrupt the PC.**

The priority of INT_CHAN_0/1/2 is the same. If all these three 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 DEMO6.C in DOS operating system for demonstrate the application under the condition of both INT_CHAN_1 and INT_CHAN_2.

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 and DEMO5.C in DOS operating system, are designed for single-channel interrupt application as follows:

DEMO3.C → for INT_CHAN_0 only
DEMO4.C → for INT_CHAN_1 only
DEMO5.C → for INT_CHAN_2 only

2.7.2 INT_CHAN_0

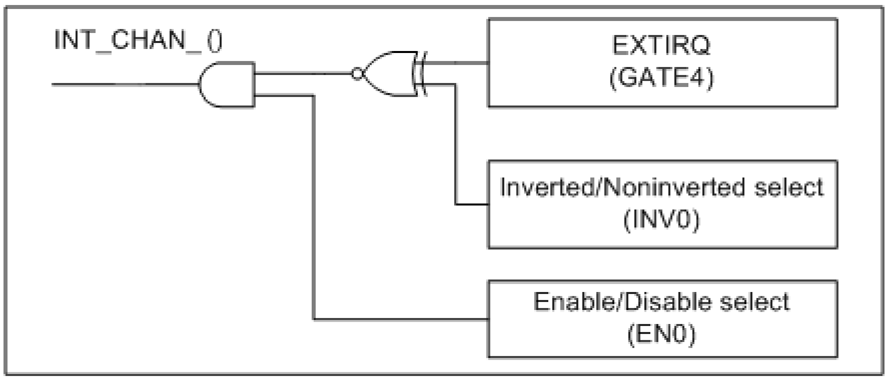


Figure 2.7

Figure 2.7 illustrates the control method of external interrupt. Note that the signal source come from GATE4. The INV0 is used to invert/non-invert the trigger signal source and EN0 is used to disable/enable the timer interrupt (Pin13 of CN5) (Refer to Sec. 2.5 for the source of interrupt signal)**. The INT_CHAN_0 must be fixed in low level state normally and generated a high_pulse to interrupt the PC.**

1. The EN0 can be used to enable/disable the INT_CHAN_0 as follows: (refer to Figure 2.7 and Sec. 3.1.4)
    - EN0=0 → INT_CHAN_0=disable
    - EN0=1 → INT_CHAN_0=enable

2. The INV0 can be used to invert/non-invert the EXTIRQ as follows: (Refer to Figure 2.7 and Sec. 3.1.6)
    - INV0=0 → INT_CHAN_0=inverted state of EXTIRQ
    - INV0=1 → INT_CHAN_0=non-inverted state of EXTIRQ

NOTE: Refer to DEMO3.C in DOS operating system for more information.

2.7.3 *INT_CHAN_1*

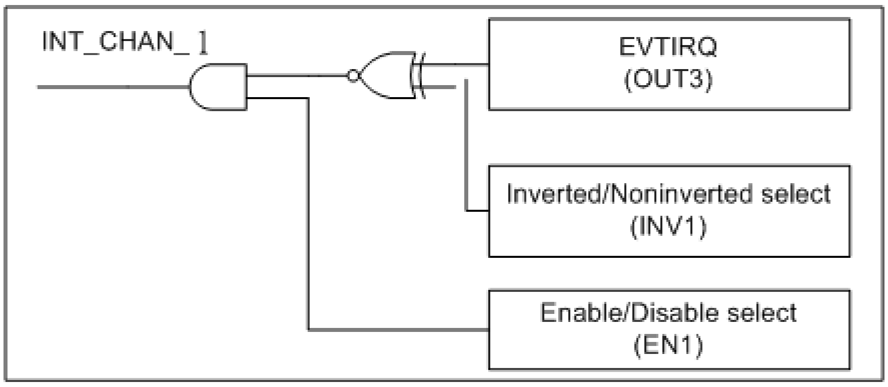


Figure 2.8

Figure 2.8 illustrates the control method of event interrupt. Note that the signal source come from OUT3. The INV1 is used to invert/non-invert the trigger signal source and EN1 is used to disable/enable the timer interrupt (Refer to Sec. 2.5 for the source of interrupt signal). User can use Counter3 as event counter to count the event signal that comes from Pin7 of CN5. When the amount of event is the same as counter3 setting, the interrupt of INT_CHAN_1 will be trigger. **The INT_CHAN_1 must be fixed in low level state normally and generated a high_pulse to interrupt the PC.**

1. The EN1 can be used to enable/disable the INT_CHAN_1 as follows: (refer to Figure 2.8 and Sec. 3.1.4)
    - EN1=0 → INT_CHAN_1=disable
    - EN1=1 → INT_CHAN_1=enable

2. The INV1 can be used to invert/non-invert the EVTIRQ as follows: (Refer to Figure 2.8 and Sec. 3.1.6)
    - INV1=0 → INT_CHAN_1=inverted state of EVTIRQ
    - INV1=1 → INT_CHAN_1=non-inverted state of EVTIRQ

NOTE: Refer to DEMO4.C in DOS operating system for more information.

2.7.4 INT_CHAN_2

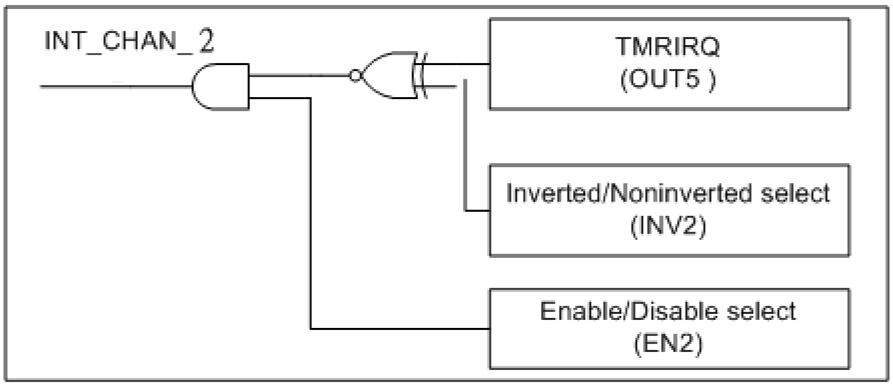


Figure 2.9

Figure 2.9 illustrates the control method of timer interrupt. Note that the signal source come from OUT5. The INV2 can be used to invert/non-invert the Trigger signal source and EN2 is used to disable/enable the timer interrupt (Refer to Sec.2.5 for the source of interrupt signal). **Note that the INT_CHAN_2 must be fixed in low level state normally and generated a high_pulse to interrupt the PC**. Because Counter4 and Counter5 are cascaded together, it can be used as 32-bit timer base on 4MHz clock source.

1. The EN2 can be used to enable/disable the INT_CHAN_2 as follows: (refer to Figure 2.9 and Sec. 3.1.4)
   - EN2=0 → INT_CHAN_2=disable
   - EN2=1 → INT_CHAN_2=enable

2. The INV2 can be used to invert/non-invert the TMRIRQ as follow2: (Refer to Figure 2.9 and Sec. 3.1.6)
   - INV2=0 → INT_CHAN_2=inverted state of TMRIRQ
   - INV2=1 → INT_CHAN_2=non-inverted state of TMRIRQ

NOTE: Refer to DEMO5.C in DOS operating system for more information.

2.8 Daughter Boards

2.8.1 DB-16P Isolated Input Board

The DB-16P is a 16-channel isolated digital input daughter board. The optically isolated inputs of the DB-16P consist of a bi-directional photo-coupler with a resistor for current limiting. You can use the DB-16P to sense DC signal from TTL levels up to 24V or use the DB-16P to sense a wide range of AC signals. You can use this board to isolated the computer from large common-mode voltage, ground loops and transient voltage spike that often occur in industrial environments. The detail function block diagram is shown as Figure 2.10.

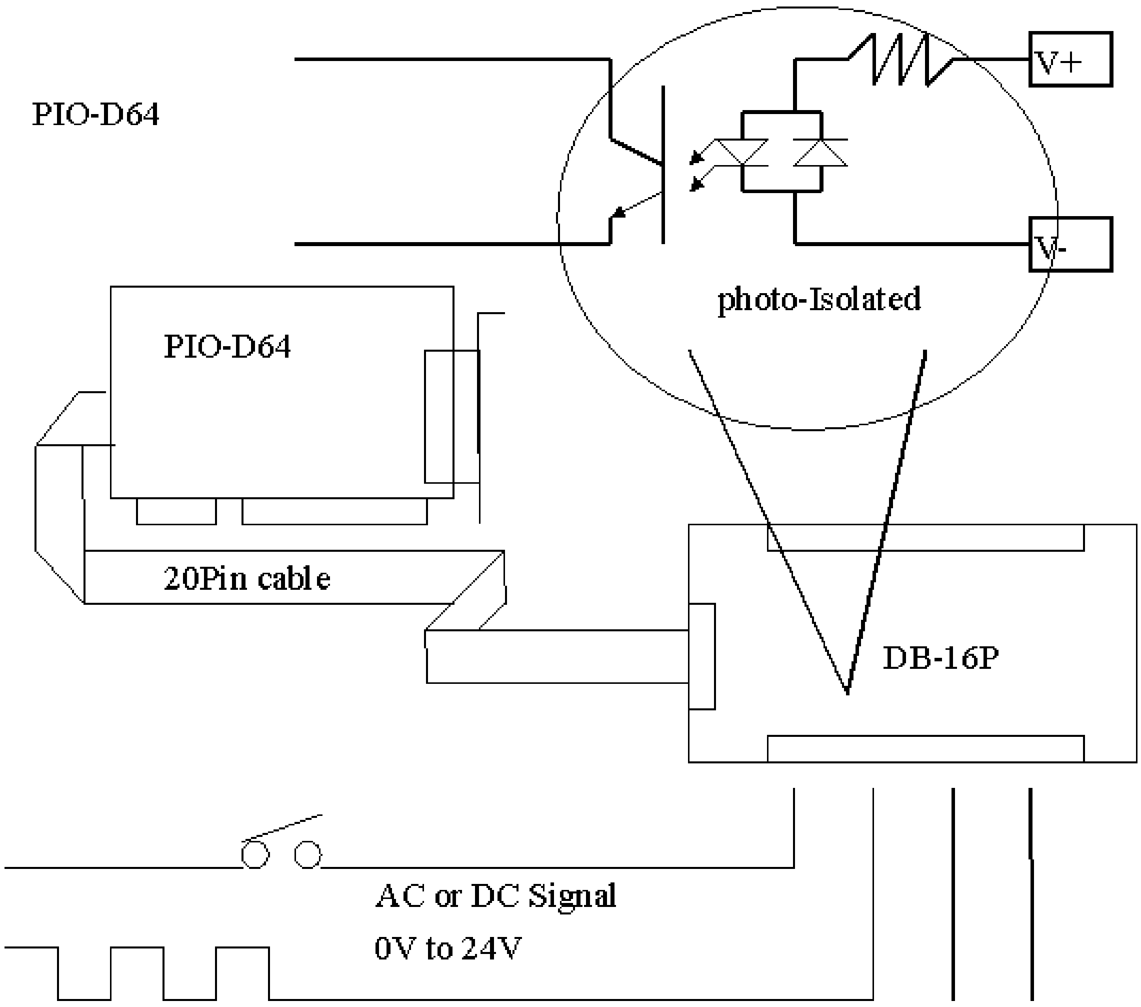


Figure 2.10

2.8.2 DB-16R Relay Board

The DB-16R is a 16-channels relay output board, which consists of 16 form C relays for efficient switch of load by programmable control. The relay are energized by applying 12V/24V voltage signal to the appropriated relay channel on the 20-pin flat connector. There are 16 enunciator LEDs on the relay daughter board. The LED light when their associated relay is activated. The detail function block diagram is shown as Figure 2.11.

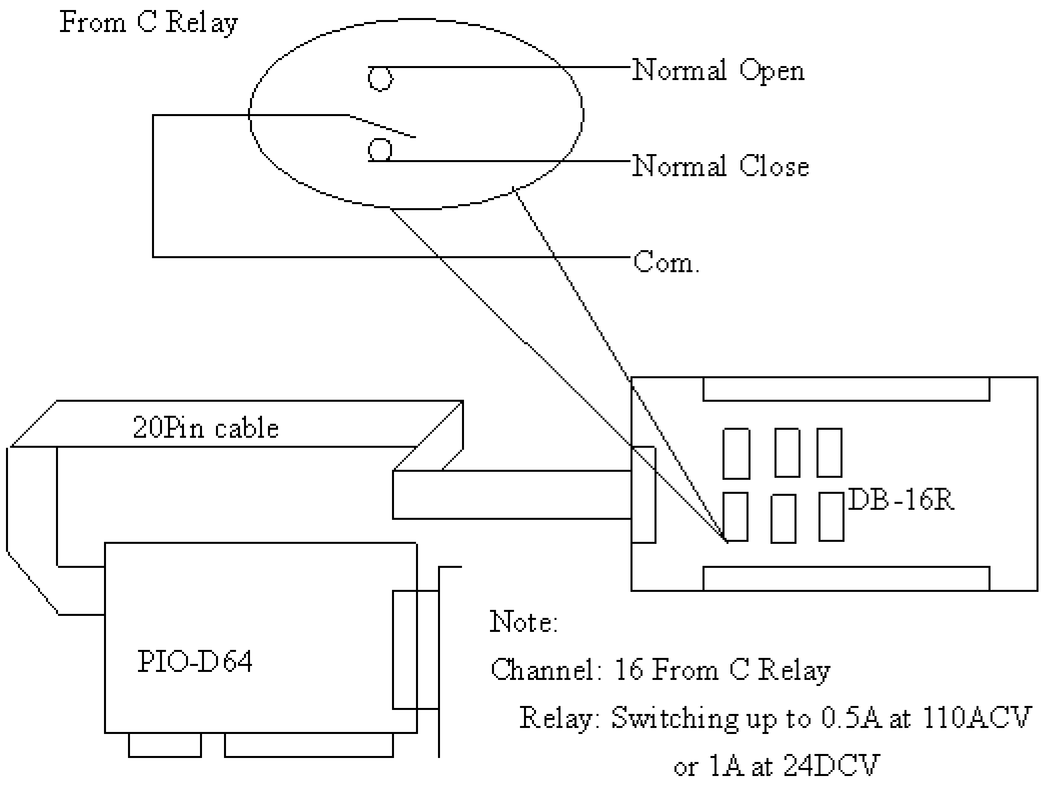


Figure 2.11

2.8.3 DB-24PRD, DB-24POR, DB-24C

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

The DB-24PR, is a 24-channel power relay output board, which consists of 8 form C and 16 form A electromechanical relays for efficient switching of load programmable control. The contact of each relay can control a 5A load at 250ACV/30VDCV. The relay is energized by applying a 5 voltage signal to the appropriate relay channel on the 20-pin flat cable connector (just used 16 relays) or 50-pin flat cable connector (OPTO-22 compatible, for DIO-24 series). There are 24 enunciator LEDs on the relay daughter board. The LED light when their associated relay are activated. To avoid overloading your PC’ s power supply, this board needs a +12VDC or +24VDC external power supply. The detail function block diagram is shown as Figure 2.12.

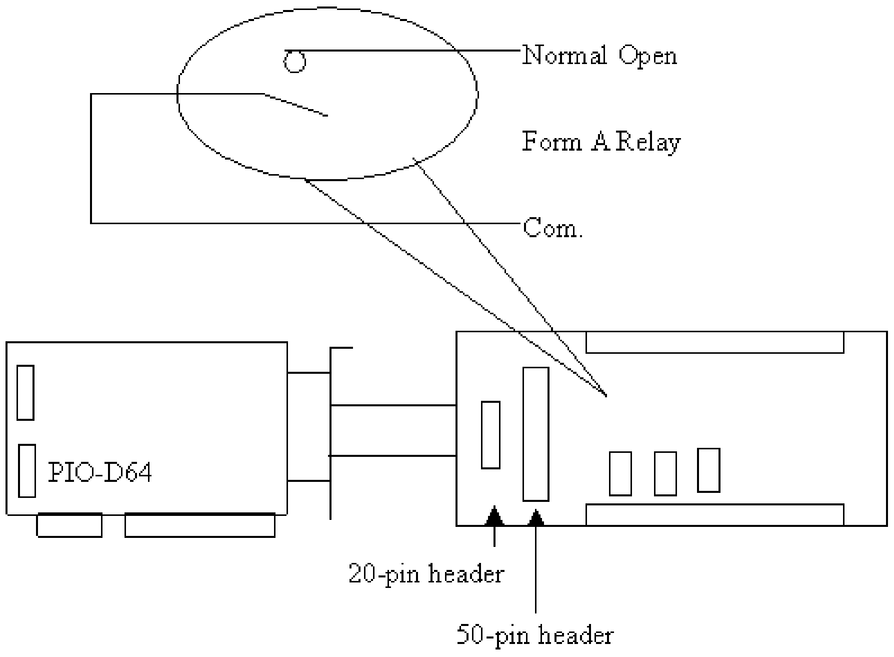


Figure 2.12

Note:
50-Pin connector (OPTO-22 compatible), for DIO-24, DIO-48, DIO-144,
PIO-D144, PIO-D96, PIO-D56, PIO-D48, PIO-D24,PIO-D168(A)
Channel: 16 Form A Relays, 8 Form C Relay
Relay: switching up to 5A at 110ACV / 5A at 30DCV

2.8.4 Daughter Board Comparison Table

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

NOTE: The PIO-D64 only has 20-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-up stage. The IDs of PIO-D64 card are given as follows:

< REV 1.0 > :

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

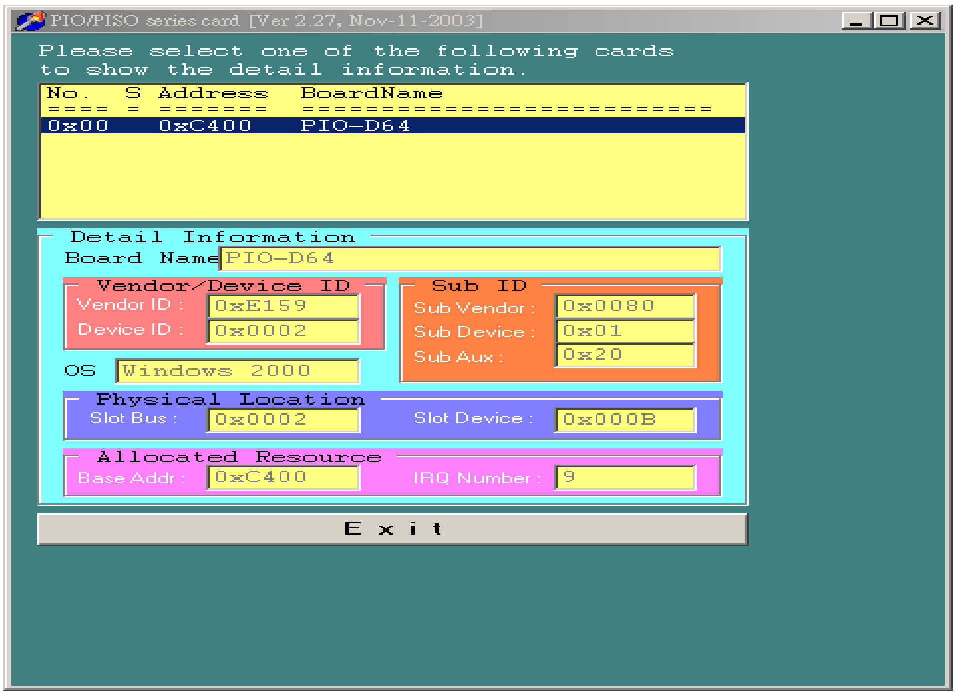
- **Sub-vendor ID = 0x80**
- **Sub-device ID = 0x01**
- **Sub-aux ID = 0x20**

< REV 2.0 > :

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

- **Sub-vendor ID = 0x4080**
- **Sub-device ID = 0x01**
- **Sub-aux ID = 0x20**

The utility program, PIO_PISO.EXE, will detect and present all information of PIO/PISO cards installed in this PC, as shown in following figure. Besides, 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.



```
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-D64

Detail Information
Board Name PIO-D64
Vendor/Device ID
  Vendor ID : 0xE159
  Device ID : 0x0002
Sub ID
  Sub Vendor : 0x0080
  Sub Device : 0x01
  Sub Aux : 0x20
OS  Windows 2000
Physical Location
  Slot Bus : 0x0002        Slot Device : 0x000B
Allocated Resource
  Base Addr : 0xC400       IRQ Number : 9
                    Exit
```

Figure 3.1

The Sub-IDs of PIO/PISO series card are given as follows:

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	96 * D/I/O	80	01	10
PIO-D64(REV 2.0)	64 * D/I/O	80 (4080)	01	20
PIO-D56	24 * D/I/O + 16 * D/I+16*D/O	80	01	40
PIO-D48	48 * D/I/O	80	01	30
PIO-D24	24 * D/I/O	80	01	40
PIO-823	Multi-function	80	03	00
PIO-821	Multi-function	80	03	10
PIO-DA16	16 * D/A	80	04	00
PIO-DA8	8 * D/A	80	04	00
PIO-DA4	4 * D/A	80	04	00
PISO-C64	64 * isolated D/O (**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	32* isolated D/O (**Current sinking**) + 32* isolated D/I	80	08	20
PISO-P32A32	32*isolated DO (**Current sourcing**) + 32* isolated D/I	80	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	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	0A	00
PISO-DA2	2 * isolated D/A	80	0B	00

Note: If the board has different version, it may has different Sub IDs. But no matter which version of the board you select, we offer the same function calls.

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 the maximum 16 boards. Therefore, the user can install 16 boards of PIO/PSIO series cards in one PC system. For how to find and identify the card_0, card_1 and the others is demonstrated as below:

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

1. Remove all PIO-D64 from this PC
2. Install one PIO-D64 into the PC's PCI_slot1, run PIO_PISO.EXE and record the wSlotBus1 & wSlotDevice1
3. Remove all PIO-D64 from this PC
4. Install one PIO-D64 into the PC's PCI_slot2, run PIO_PISO.EXE and record the wSlotBus2 & wSlotDevice2
5. repeat (3) & (4) for all PCI_slot?, record all wSlotBus? & wSlotDevice?

The records may be as follows:

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 records all information of wSlotBus and wSlotDevice in this PC. These values will be mapped to this PC's physical slot. And this mapping will not be changed for any PIO/PISO cards. Therefore, this information can be used to identify the specified PIO/PISO card by following steps:

Step1: Using the information of wSlotBus and wSlotDevice in table 3-2
Step2: Input board number into funtion PIO_GetConfigAddressSpace(…) to get the specified card's information, especially wSlotBus and wSlotDevice
Step3: The user can identify the specified PIO/PISO card by comparing the data of the wSlotBus & wSlotDevice in step1 and step2.

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

3.3 The I/O Address Map

The I/O address of PIO/PISO series card is automatically assigned by the main board ROM BIOS. The I/O address can also be re-assigned by user. **It is strongly recommended not to change the I/O address by user. The Plug & Play BIOS will assign proper I/O address to each PIO/PISO series card very well**. The I/O address of PIO-D64 are given as follows, 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	DI0~DI7	DO0~DO7
wBase+0xc4	DI8~DI15	DO8~DO15
wBase+0xc8	DI16~DI23	DO16~DO23
wBase+0xcc	DI24~DI31	DO24~DO31
wBase+0xd0	Read U4 8254-counter0	Write U4 8254-counter0
wBase+0xd4	Read U4 8254-counter1	Write U4 8254-counter1
wBase+0xd8	Read U4 8254-counter2	Write U4 8254-counter2
wBase+0xdc	Read U4 8254 control word	Write U4 8254 control word
wBase+0xe0	Read U5 8254-counter3	Write U5 8254-counter3
wBase+0xe4	Read U5 8254-counter4	Write U5 8254-counter4
wBase+0xe8	Read U5 8254-counter5	Write U5 8254-counter5
wBase+0xec	Read U5 8254 control word	Write U5 8254 control word

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

3.3.1 RESET\ Control Register

(Read/Write): wBase+0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	RESET\

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

When the PC is first power-up, the RESET\ signal is in Low-state. **This will disable all D/I/O operations.** The user has to set the RESET\ signal to High-state before any D/I/O command.

```
outportb(wBase,1);          /* RESET\ = High → all D/I/O are enable now */
outportb(wBase,0);          /* RESET\ = Low  → all D/I/O are disable now */
```

3.3.2 AUX Control Register

(Read/Write): wBase+2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

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

Aux?=0→ this Aux is used as a D/I

Aux?=1→ this Aux is used as a D/O

When the PC is first power-on, All Aux? signal are in Low-state. All Aux? are designed as D/I for all PIO/PISO series. Please set all Aux? in D/I state.

3.3.3 AUX data Register

(Read/Write): wBase+3

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

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

When the Aux? is used as D/O, the output state is controlled by this register. This register is designed for feature extension, so don't control this register now.

3.3.4 INT Mask Control Register

(Read/Write): wBase+5

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	EN3	EN2	EN1	EN0

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

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

EN1=0→ disable INT_CHAN_1 as a interrupt signal (default)
EN1=1→ enable INT_CHAN_1 as a interrupt signal

EN2=0→ disable INT_CHAN_2 as a interrupt signal (default)
EN2=1→ enable INT_CHAN_2 as a interrupt signal

EN3=0→ disable INT_CHAN_3 as a interrupt signal (default)
EN3=1→ enable INT_CHAN_3 as a interrupt signal

```
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+5,7);        /* enable all four channels of interrupt  */
```

Refer to the following demo program for more information:

DEMO3.C of DOS	→ for INT_CHAN_0 only
DEMO4.C of DOS	→ for INT_CHAN_1 only
DEMO5.C of DOS	→ for INT_CHAN_2 only
DEMO6.C of DOS	→ for INT_CHAN_1 and INT_CHAN_2

3.3.5 Aux Status Register

(Read/Write): wBase+7

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

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

Aux0=INT_CHAN_0, Aux1=INT_CHAN_1, Aux2=INT_CHAN_2, Aux3=INT_CHAN_3, Aux7~4=Aux-ID. The Aux 0~3 are used as interrupt sources. The interrupt service routine has to read this register for interrupt source identification. Refer to Sec. 2.7 for more information.

3.3.6 Interrupt Polarity Control Register

(Read/Write): wBase+0x2A

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	INV2	INV1	INV0

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

INV0/1/2=0→ select the invert signal from INT_CHAN_0/1/2

INV0/1/2=1→ select the non-invert signal from INT_CHAN_0/1/2

```
outportb(wBase+0x2a,0);       /* select the invert input from all 3 channels      */
outportb(wBase+0x2a,0x0f);    /* select the non-invert input from all 3 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 DEMO6.C of DOS for more information.

3.3.7 Read/Write 8254

8254 control word

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

4. Software Installation

The PIO-D64 can be used in DOS and Windows 98/Me/NT/2000/XP. For Windows operating system, 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 cannot be started automatically for some reasons, please double-click the file \NAPDOS\AUTO32.EXE in this 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 system.

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

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

4.2 PnP Driver Installation

After installing the hardware (PIO-D64) and power on your PC, Windows 98/Me/2000/XP can find a PCI card device and ask user to provide a PIODIO.inf to install hardware driver on the computer. If user has trouble to proceed this process, please refer to PnPinstall.pdf for more information.

5. DLL Function Description

The DLL driver is the collection of function calls of the PIO-DIO cards for Windows 98/Me/NT/2000/XP system. The application structure is presented as following figure. The user application program developed by designate tools like VB, Delphi and Borland C^{++} Builder can call PIODIO.DLL driver in user mode. And then DLL driver will call the PIO.sys to access the hardware system.

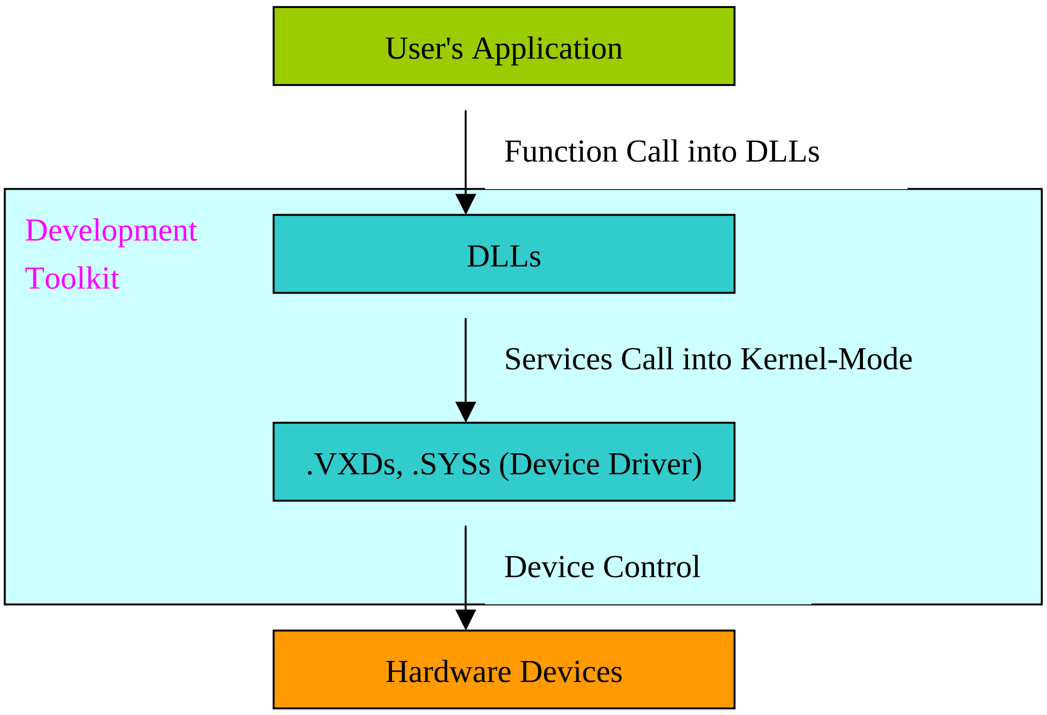


Figure 5.1

5.1 Table of ErrorCode and ErrorString

Table 5.1

Error Code	Error ID	Error String
0	PIODIO_NoError	OK (No error !)
1	PIODIO_DriverOpenError	Device driver can't be opened
2	PIODIO_DriverNoOpen	Users have to call the DriverInit function firstly
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 of interrupt error
7	PIODIO_RemoveIrqError	Remove IRQ Error
8	PIODIO_FindBoardError	Can not find board
9	PIODIO_ExceedBoardNumber	The Max. boards 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 PIO-D64 are listed as below and the detail information for every function will be presented in the following section. However, in order to make the description simplify and clearly, the attribute of the input and output parameter 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

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 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 total boards. 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 version PIO-D64 boards may have different Sub IDs. This function will find the total board of PIO-D64 including all version, no matter what version Sub ID you input. Following is the example demonstration:

  wRtn=PIODIO_SearchCard(&wBoards, 0x800120);

  you will get the total numbers of PIO-D64 boards including REV 1.0 and REV 2.0 in PC.

- **Return:**

  Please refer to "Section 5.1 Error Code"

5.5.4 PIODIO_DriverClose

- **Description :**

  This subroutine closes the PIODIO Driver and releases the resource from computer device resource. This function must be called 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 of 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 PIODIO board. Only the low WORD is valid.
wIrqNo	: [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 "Section 5.1 Error Code".

5.6 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 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 a Event handle. The user's program must call 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 .

Table 5.3

wInterruptSource	Description
0	EXTIRQ
1	EVTIRQ
2	TMRIRQ

  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".

5.6.5 Architecture of Interrupt mode

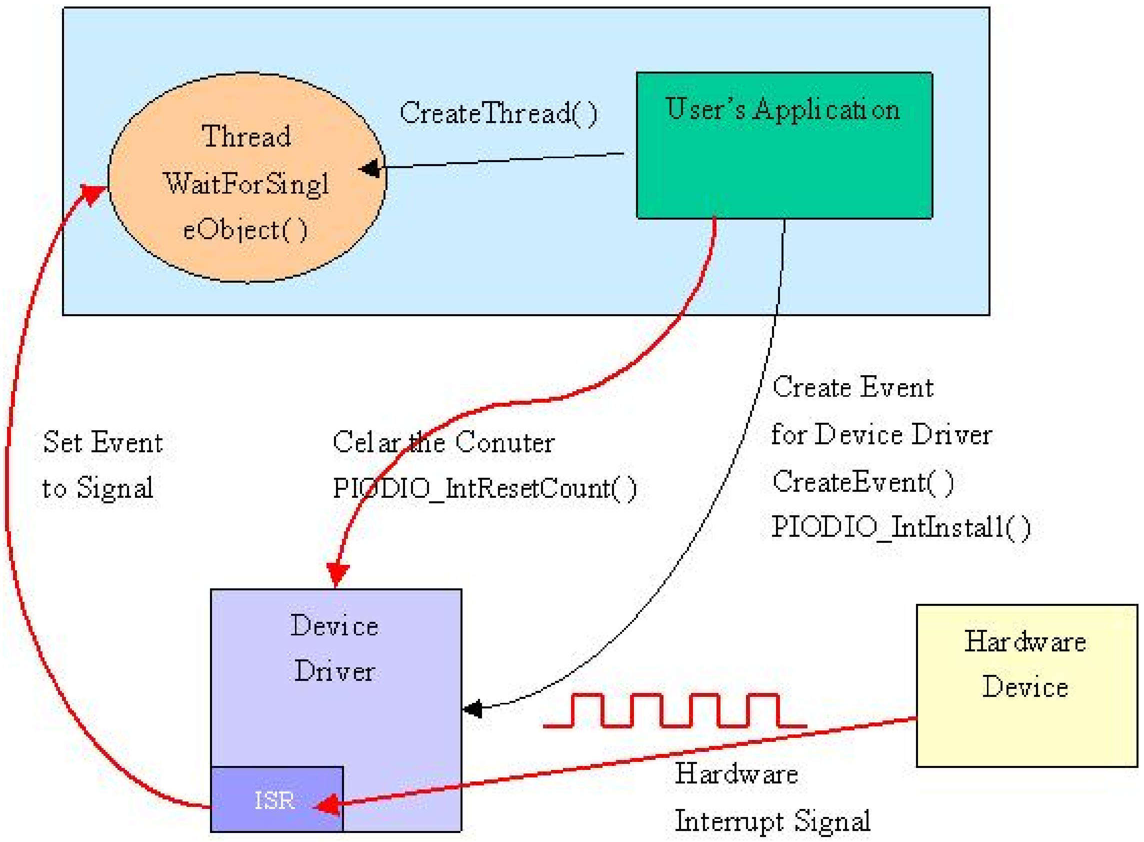


Figure 5.2

5.7 COUNTER FUNCTION

5.7.1 PIOD64_SetCounter

- **Description :**

  This subroutine is used to set the 8254 counter's mode and value.

- **Syntax :**

  void PIOD64_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

5.7.2 PIOD64_ReadCounter

- **Description :**

  This subroutine is used to obtain the 8254 counter's value.

- **Syntax :**

  DWORD PIOD64_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.)

5.7.3 PIOD64_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 PIOD64_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

5.7.4 PIOD64_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 PIOD64_ReadCounterA(WORD wCounterNo,
  WORD bCounterMode);

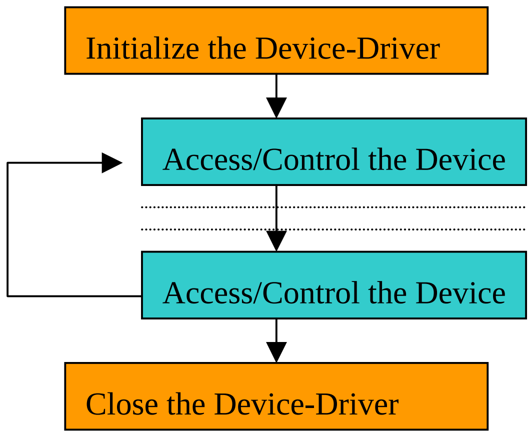
- **Parameter :**

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.)

5.7.5 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 of demo programs will not work normally if DLL driver would not be installed correctly. During the installation process of DLL driver, 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). After driver installation, the related demo programs and development library and declaration header files for different development environments are presented as follows.

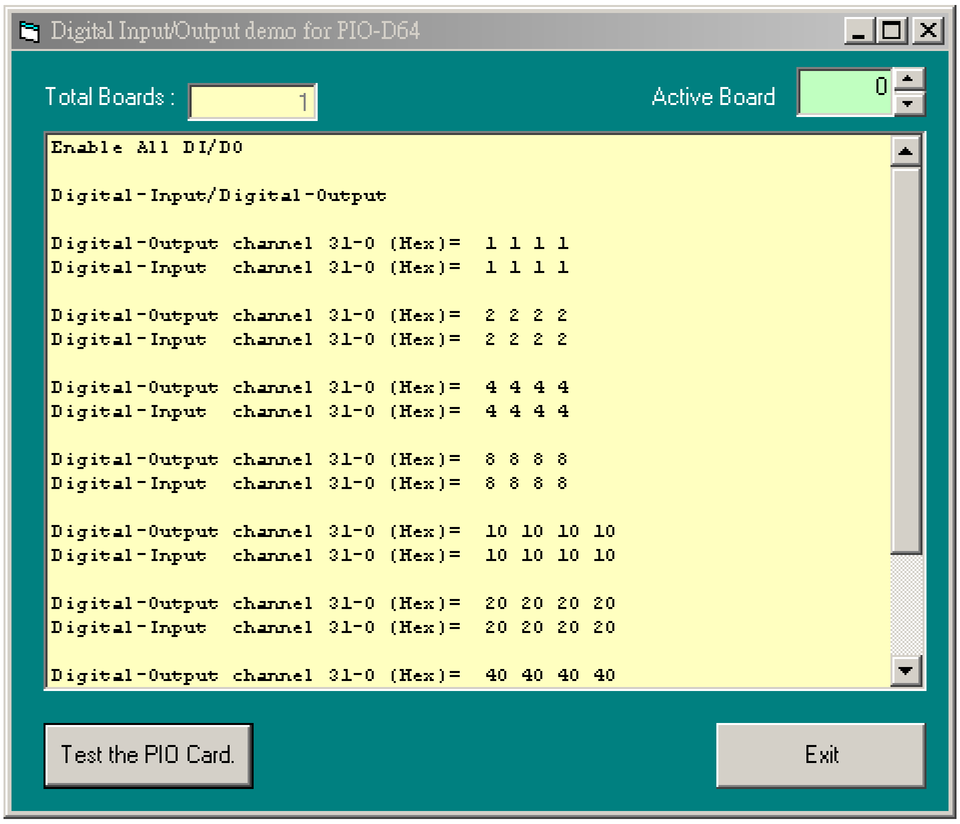
```
|--\Demo                          → demo program
   |--\BCB3                       → for Borland C++ Builder 3
   |   |--\PIODIO.H               → Header file
   |      \ PIODIO.LIB            → Linkage library for BCB only
   |
   |--\Delphi3                    → for Delphi3
   |   |--\ PIODIO.PAS            → Declaration file
   |
   |--\VB5                        → for Visual Basic 6
   |  |--\ PIODIO.BAS             →Declaration file
   |
   |--\VC6                        → for Visual C++ 6
   |   |--\PIODIO.H               → Header file
   |      \ PIODIO.LIB            → Linkage library for VC
```

The list of demo programs :

Dio	: Digital Input / Output .
INT	: Interrupt of EXTIRQ.
Counter	: Counter0.

6.1 Digital Intput/Output

This demo program is used to check the digital input and output status of CN2/CN4 and CN1/CN3.

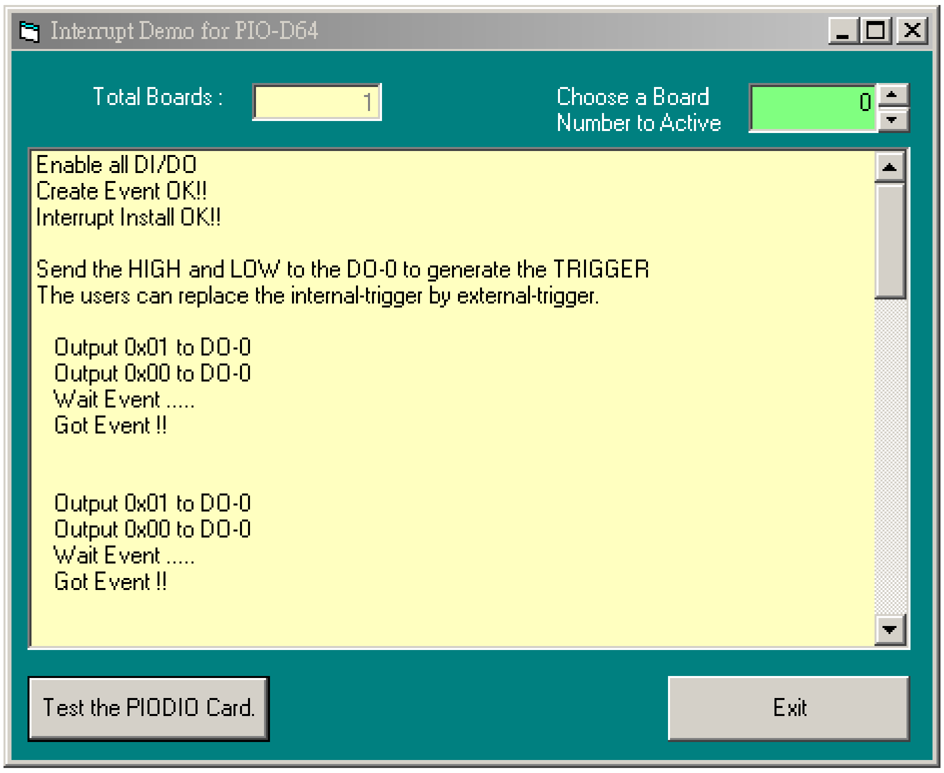


```
Enable All DI/DO

Digital-Input/Digital-Output

Digital-Output channel 31-0 (Hex)=  1 1 1 1
Digital-Input  channel 31-0 (Hex)=  1 1 1 1

Digital-Output channel 31-0 (Hex)=  2 2 2 2
Digital-Input  channel 31-0 (Hex)=  2 2 2 2

Digital-Output channel 31-0 (Hex)=  4 4 4 4
Digital-Input  channel 31-0 (Hex)=  4 4 4 4

Digital-Output channel 31-0 (Hex)=  8 8 8 8
Digital-Input  channel 31-0 (Hex)=  8 8 8 8

Digital-Output channel 31-0 (Hex)=  10 10 10 10
Digital-Input  channel 31-0 (Hex)=  10 10 10 10

Digital-Output channel 31-0 (Hex)=  20 20 20 20
Digital-Input  channel 31-0 (Hex)=  20 20 20 20

Digital-Output channel 31-0 (Hex)=  40 40 40 40
```

Figure 6.1

6.2 Interrupt of EXTIRQ

This demo program uses EXTIRQ as interrupt source. Then DO0 output a high and low signal repeatedly to trigger the interrupt source .

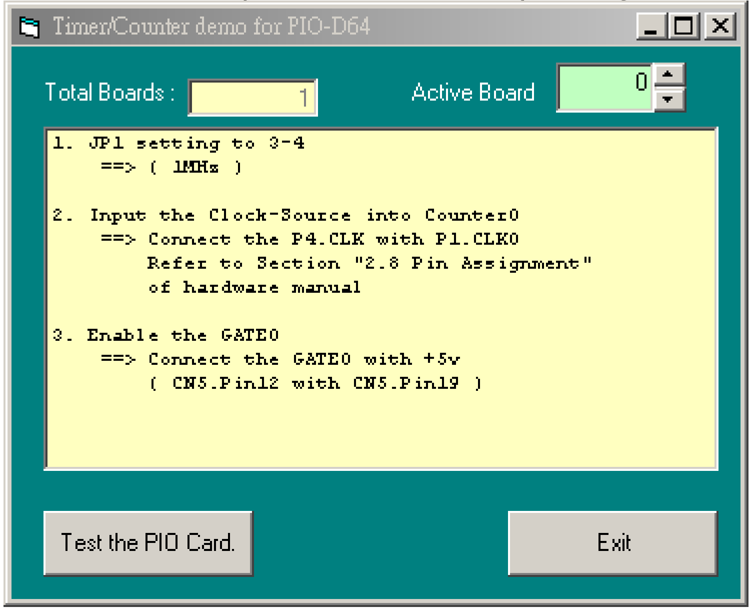


```
Enable all DI/DO
Create Event OK!!
Interrupt Install OK!!

Send the HIGH and LOW to the DO-0 to generate the TRIGGER
The users can replace the internal-trigger by external-trigger.

  Output 0x01 to DO-0
  Output 0x00 to DO-0
  Wait Event .....
  Got Event !!


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

Figure 6.2

6.3 Counter Function of counter0

This demo program uses internal clock to test counter0 function. user can select clock suitable clock output from the corresponding P4 soldering pad.



```
1. JP1 setting to 3-4
    ==> ( 1MHz )

2. Input the Clock-Source into Counter0
    ==> Connect the P4.CLK with P1.CLK0
        Refer to Section "2.8 Pin Assignment"
        of hardware manual

3. Enable the GATE0
    ==> Connect the GATE0 with +5v
        ( CN5.Pin12 with CN5.Pin19 )
```

Figure 6.3

Appendix

Appendix A. *Related DOS Software*

A-1 Where is the related software

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

- \TC*.* → for Turbo C 2.xx or above
- \MSC*.* → for MSC 5.xx or above
- \BC*.* → for BC 3.xx or above

- \TC\LIB\*.* → for TC library
- \TC\DEMO\*.* → for TC demo program
- \TC\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.C : D/O demo
DEMO2.C : D/I/O demo
DEMO3.C : Use external int. to measure pulse width(high level)
DEMO4.C : Use EVTIRQ to count event
DEMO5.C : Use TMRIRQ to generate 0.5Hz squa.
DEMO6.C : Use TMRIRQ to generate 0.5Hz squa. EVTIRQ to count

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 resource 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_DriverClose*

- **Description :**

  This subroutine closes the PIODIO Driver and releases the resource from computer device resource. This function must be called once before exiting the user's application.

- **Syntax :**

  WORD PIO_DriverClose ()

- **Parameter :**

  None

- **Return:**

  Please refer to " Table A.1".

A-2-4 *PIO_GetConfigAddressSpace*

- **Description :**

  The user can use this function to save resource of all PIO/PISO cards installed in this system. Then the application program can control all functions of PIO/PISO series card directly.

- **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 *PIO_GetDriverVersion*

- **Description :**

  This subroutine obtains 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-2-6 *ShowPIOPISO*

- **Description :**

  This function will show a text string for this 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".