PIO-DA4/DA8/DA16
PIO-DA4U/DA8U/DA16U
PISO-DA4U/DA8U/DA16U

User Manual

Version 3.0
Oct. 2011

Warranty

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for damages consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

Names are used for identification only and may be registered trademarks of their respective companies.

Tables of Contents

1. INTRODUCTION 3

1.1 FEATURES 4
1.2 SPECIFICATIONS 5
1.3 ORDER INFORMATION 6
1.4 PRODUCT CHECKLIST 6

2. HARDWARE CONFIGURATION 7

2.1 BOARD LAYOUT 7
2.2 COUNTER ARCHITECTURE 9
2.3 INTERRUPT OPERATION 10
2.4 D/I/O BLOCK DIAGRAM 17
2.5 D/A ARCHITECTURE 20
2.6 D/A CONVERSION OPERATIONS 21
2.7 PIN ASSIGNMENTS 31
2.8 DAUGHTER BOARDS 34

3. I/O CONTROL REGISTER 39

3.1 HOW TO FIND THE I/O ADDRESS 39
3.2 THE ASSIGNMENT OF THE I/O ADDRESSES 45
3.3 THE I/O ADDRESS MAP 47

4. SOTFWARE INSTALLATION 55

4.1 SOFTWARE INSTALLING PROCEDURE 55
4.2 PNP DRIVER INSTALLATION 56
4.3 CONFIRM THE SUCCESSFUL INSTALLATION 57

5. DEMO PROGRAMS 58

5.1 DEMO PROGRAMS FOR WINDOWS 58
5.2 DEMO PROGRAMS FOR DOS 59
5.3 PIO_PISO.EXE FOR WINDOWS 61
5.4 DEMO1 62
5.5 DEMO2 63
5.6 DEMO3 64
5.7 DEMO5 66
5.8 DEMO8 68
5.9 DEMO9 69

1. Introduction

The PISO-DA4U/DA8U/DA16U and PIO-DA4U/DA8U/DA16U analog output boards support the universal PCI interface (3.3 V/5 V PCI). These cards are equipped with 4/8/16 14-bit analog output channels. Each of the D/A channels features a double-buffered latch. The PISO-DAxU and the PIO-DAxU series cards (universal PCI versions) are fully compatible with the PIO-DAx cards (PCI versions) and users can directly replace the PIO-DAx with either the PISO-DAxU or the PIO-DAxU without any need for software/driver modification.

Adds high-voltage isolation design has been added to the PISO-DA series which provides the ability to protect the user's computer from unexpected voltage surges via the built-in high-quality isolation components that feature a 2500 V_{DC} bus-type isolation. For both the PIO-DA and the PISO-DA series, the voltage output range is from -10 V to +10 V, with a current output range from 0 to 20 mA. In addition, the PISO-DA and PIO-DA series also provide the following advantages:

1. Accurate and easy-to-use calibration

   ICP DAS provides software-based calibration instead of the previous manual calibration so that jumpers and trim-pots are no longer required. The calibration information can be saved in EEPROM for long-term use.

2. Individual channel configuration

   Every channel can be individually configured as either voltage or current output.

3. Card ID

   ICP DAS provides a Card ID function for both the PISO-DA series and the PIO-DA4U/DA8U/DA16U (version 1.1 or above series). Users can set a unique card ID for each card, which can then be used to individually identify then if more than two boards are used in a single PC.

Note: This card needs a ±12 V power supply, which can be found in either a regular PC or an Industrial PC.

1.1 Features

- Supports the +5 V PCI bus for PIO-DA series
- supports the +5 V and +3.3 V PCI bus for PIO-DAxU/PISO-DAxU series
- Digital input port can be set to either pull-high or pull-low for PIO-DAxU/ PISO-DAxU series
- Card ID function for PIO-DAxU/PISO-DAxU series
- Built-in DC/Dc converter with 3000 V_{DC} isolation for PISO-DAxU series
- 2500 V_{DC} bus-type and power isolation protection for PISO-DAxU series
- 4/8/16 analog output channels, 14-bit analog output
- Voltage output range ±10 V
- Current output range: 0 ~ 20 mA (sink)
- Two pacer timer interrupt sources
- Double-buffered D/A latch
- Software calibration
- 16 D/I channels, 16 D/O channels
- One D-Sub connector, two 20-pin flat cable connectors
- Connects directly to DB-16P, DB-16R, DB-24C, DB-24PR and DB-24POR daughter boards

Comparison Table of the Different Version Information:

	D/I Register	Pin Assignment	Card ID
PIO-DA	0xE0/E4/E8/EC 0xF0/F4/F8/FC	A. GND (CN3.5/10/15/24/29)	**N/A**
PIO-DAxU V1.0	0xE0/E4/E8/EC 0xF0/F4/F8/FC	A. GND (CN3.5/10/15/24/29)	**N/A**
PIO-DAxU V1.1	0xE0/E4/E8/EC 0xF0/F4/F8/FC	A. GND (CN3.5/10/15/24/29)	**Yes**
PIO-DAxU V1.2 or above	0xE0/E4	A. GND (CN3.5/10/15)	**Yes**
PISO-DAxU V1.3 or above	0xE0/E4	A. GND (CN3.5/10/15/24/29)	**Yes**

1.2 Specifications

Model Name	PISO-DA4U/DA8U/DA16U	PIO-DA4U/DA8U/DA16U
Analog Output		
Isolation Voltage	2500 V (Bus Type)	N/A
Channels	4/8/16 independent	4/8/16 independent
Resolution	14-bit	
Accuracy	0.01% of FSR ± 2 LSB @ 25 °C, ± 10 V	
Output Range	Voltage: +/- 10 V Current: 0 ~ 20 mA	
Output Driving	+/- 5 mA	
Slew Rate	0.71 V/μs	
Output Impedance	0.1 Ω max.	
Operating Mode	Software	
Digital Input		
Channels	16	
Compatibility	5 V/TTL	
Input Voltage	Logic 0: 0.8 V max. Logic 1: 2.0 V min.	
Response Speed	1.0 MHz (Typical)	
Digital Output		
Channels	16	
Compatibility	5 V/TTL	
Output Voltage	Logic 0: 0.4 V max. Logic 1: 2.4 V min.	
Output Capability	Sink: 2.4 mA @ 0.8 V Source: 0.8 mA @ 2.0 V	
Response Speed	1.0 MHz (Typical)	
Timer/Counter		
Channels	3	
Resolution	16-bit	
Compatibility	5 V/TTL	
Reference Clock	Internal: 4 MHz	
General		
Bus Type	3.3 V / 5 V Universal PCI, 32-bit, 33 MHz	
Data Bus	8-bit	
Card ID	Yes (4-bit)	Yes (4-bit) for Version 1.1 or above
I/O Connector	Female DB37 x 1 20-pin box header x 2	
Dimensions (L x W x D)	180 mm x 97 mm x 22 mm	188 mm x 97 mm x 22 mm (Version 1.1 or above)
Power Consumption	2200 mA @ +5 V (PISO-DA4U) 2400 mA @ +5 V (PISO-DA8U) 3000 mA @ +5 V (PISO-DA16U)	600 mA@ +5 V (PIO-DA4U) 800 mA@ +5 V (PIO-DA8U) 1400 mA @ +5 V (PIO-DA16U)
Operating Temperature	0 ~ 60 °C	
Storage Temperature	-20 ~ 70 °C	
Humidity	5 ~ 85% RH, non-condensing	

1.3 Order Information

- PIO-DA4: PCI bus 4-channel D/A board
- PIO-DA8: PCI bus 8-channel D/A board
- PIO-DA16: PCI bus 16-channel D/A board
- PIO-DA4U: Universal PCI, 4-channel D/A board
- PIO-DA8U: Universal PCI, 8-channel D/A board
- PIO-DA16U: Universal PCI, 16-channel D/A board
- PISO-DA4U: Universal PCI, 4-channel isolated D/A board
- PISO-DA8U: Universal PCI, 8-channel isolated D/A board
- PISO-DA16U: Universal PCI, 16-channel isolated D/A board

1.3.1 Options

- DB-16P: 16 channel isolated D/I board
- DB-16R: 16 channel relay board
- DB-24PR: 24 channel power relay board
- DB-24POR: 24 channel PhotoMos output board
- DB-24C: 24-channel open-collector output board
- ADP-20/PCI : extender, 20-pin header to 20-pin header for PCI Bus I/O

1.4 Product Checklist

The shipping package includes the following items:

- One PIO-DA or PISO-DA series card
- One CD-ROM
- One Quick Start Guide.

It is recommended that you read the Quick Start Guide first. All necessary and essential information is given in the Quick Start Guide, including:

- Where to get the software driver, demo programs and other resources.
- How to install the software.
- How to test the card.

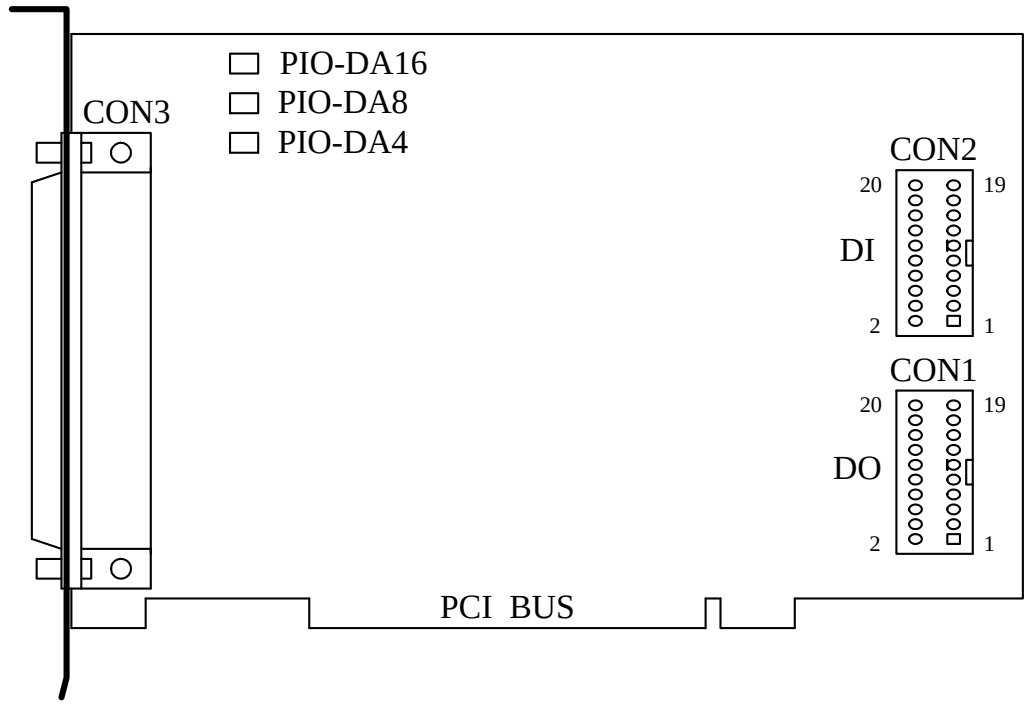
Attention!

If any of these items are missing or damaged, contact the dealer from whom you purchased the product. Please save the shipping materials and carton in case you need to ship or store the product in the future.

2. Hardware configuration

2.1 Board Layout

➢ PIO-DA4/DA8/DA16 Board Layout

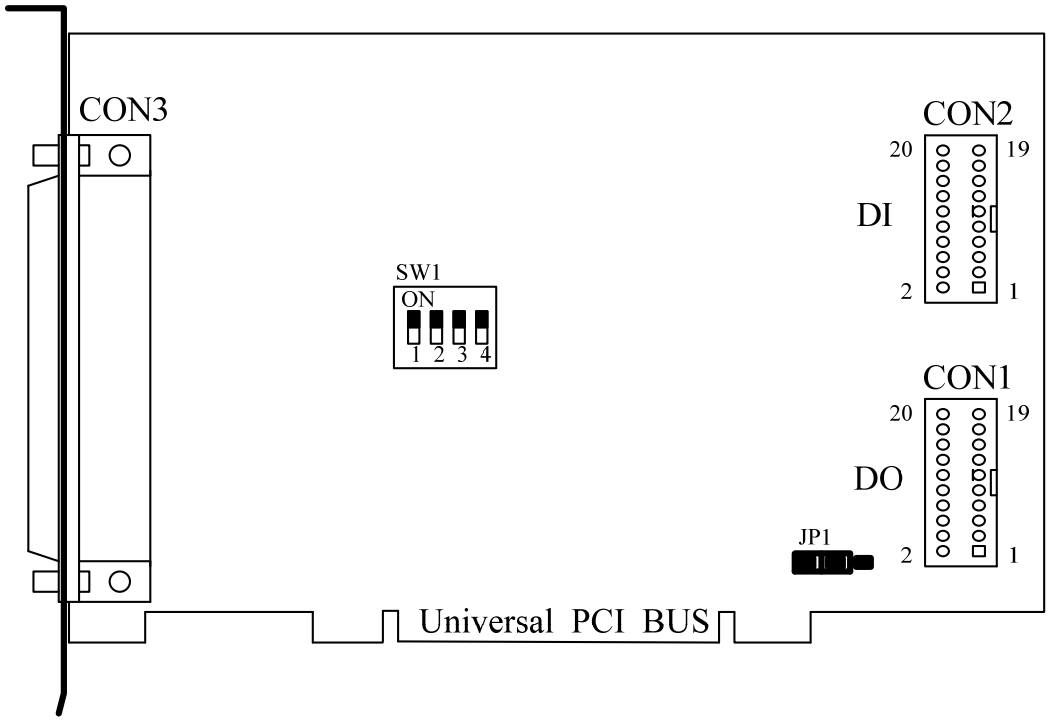


CON1: 16-channel D/O

CON2: 16-channel D/I

CON3: 4/8/16-channel D/A converter voltage/current output

➢ PIO-DA4U/DA8U/DA16U and PISO-DA4U/DA8U/DA16U Board Layout



CON1: 16-channel D/O

CON2: 16-channel D/I

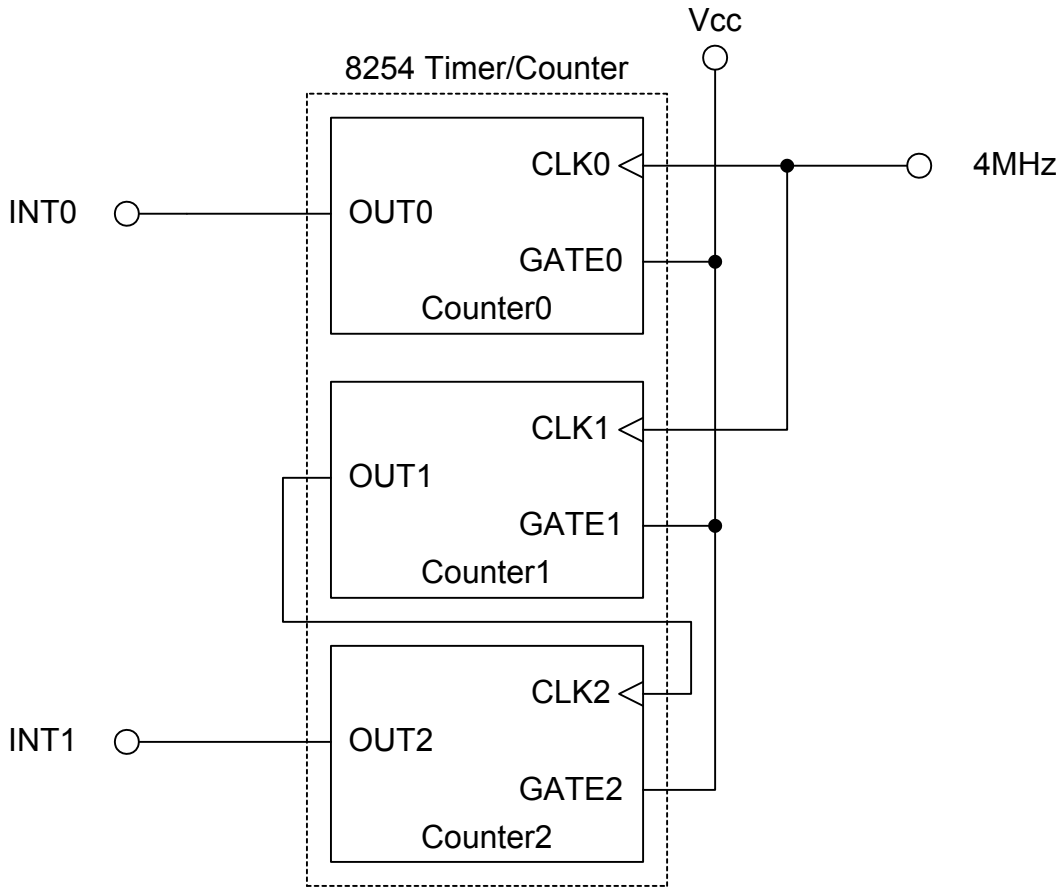
CON3: 4/8/16 channel D/A converter voltage/current output

SW1: Card ID

JP1: Pull-high/pull-low resisters for DI

2.2 Counter Architecture

There is a single 8254(Timer/Counter) chip on the PIO-DA/PISO-DA series board and provides two interrupt sources. The first is a 16-bit timer output (INT0) and the other one is a 32-bit timer output (INT1). The block diagram is shown below:



2.3 Interrupt Operation

There are two interrupt sources included in the PIO-DA/PISO-DA series. These two signals are named as INT0 and INT1, and their signal sources are as follows:

INT0: 8254 counter0 output (Refer to Sec. 2.2)
INT1: 8254 counter2 output (Refer to Sec. 2.2)

If only one interrupt signal source is used, the interrupt service routine doesn't have to identify the interrupt source. Refer to DEMO3.C and DEMO4.C for more information.

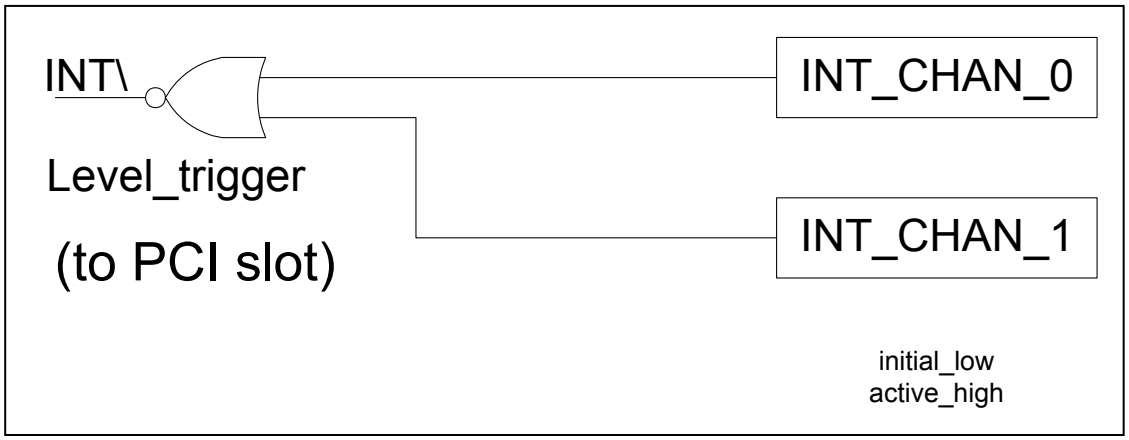
If there is more than one interrupt source, the interrupt service routine has to identify the active signals in the following manner: (Refer to DEMO5.C and DEMO6.C)

1. Read the new status of all interrupt signal sources
2. Compare the new status with the old status to identify the active signals
3. If INT0 is active, service it
4. If INT1 is active, service it
5. Save the new status to replace the old status

Note:

If the interrupt signal is too short, the new status may be the same as the old status. In that situation, the interrupt service routine will not be able to identify which interrupt source is active, so the interrupt signal must be hold_active for long enough until the interrupt service routine is executed. This hold_time is different for different OS versions. The hold_time can be as short as a micro-second or as long as second. In general, 20 mS should be long enough for all OS version.

2.3.1 Interrupt Block Diagram



The interrupt output signal of PIO-DA/PISO-DA series cards, **INT\, is set to Level-Trigger and Active_Low.** If INT\ generates a low_pulse, the PIO-DA4/8/16 will interrupt the PC once each time. If INT\ is fixed at low_level, the PIO-DA4/8/16 will interrupt the PC continuously. So for **the signal pulse_type for INT_CHAN_0/1 must be controlled and must be fixed at a low_level state normally and a high_pulse generated to interrupt the PC.**

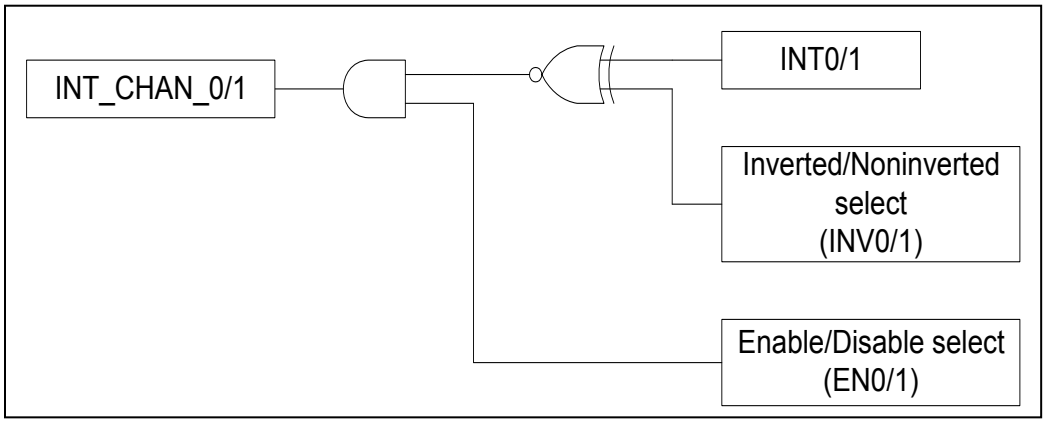
The priority of INT_CHAN_0/1 is the same. If both of these signals are active at the same time, then INT\ will only be active once at a time. So the interrupt service routine has to read the status of both interrupt channels to per form a multiple-channel interrupt. Refer to Sec. 2.3 for more information.

DEMO5.C → for INT_CHAN_0 & INT_CHAN_1

If only one interrupt source is used, the interrupt service routine doesn't have to read the status of the interrupt source. The demo programs, DEMO3.C and DEMO4.C, are designed to demons rate a single channel interrupt. See:

DEMO3.C → for INT_CHAN_1 only (initial high)
DEMO4.C → for INT_CHAN_1 only (initial low)

2.3.2 INT_CHAN_0/1



The architecture for INT_CHAN_0 and INT_CHAN_1 is shown in the above figure. The only difference between INT0 and INT1 is that the INT_CHAN_0 signal source is from the 8254 counter0 output and the INT_CHAN_1 signal source is from the 8254 counter2 output.

INT_CHAN_0/1 must be fixed at a low level state normally and a high_pulse generated to interrupt the PC.

EN0/1 can be used to enable/disable the INT_CHAN_0/1 in the following manner: (Refer to Sec.3.3.4)
EN0/1 = 0 → INT_CHAN_0/1 = disabled
EN0/1 = 1 → INT_CHAN_0/1 = enabled

INV0/1 can be used to invert/non-invert INT0/1 in the following manner: (Refer to Sec.3.3.6)
INV0/1 = 0 → INT_CHAN_0/1 = inverted state for INT0/1
INV0/1 = 1 → INT_CHAN_0/1 = non-inverted state for INT0/1

As noted above, **if INT\ is fixed at a low level state, the PIO-DA4/8/16 will interrupt the PC continuously, so the interrupt service routine should use INV0/1 to invert/non-invert the INT0/1 in order to generate a high_pulse** (Refer to the next section)

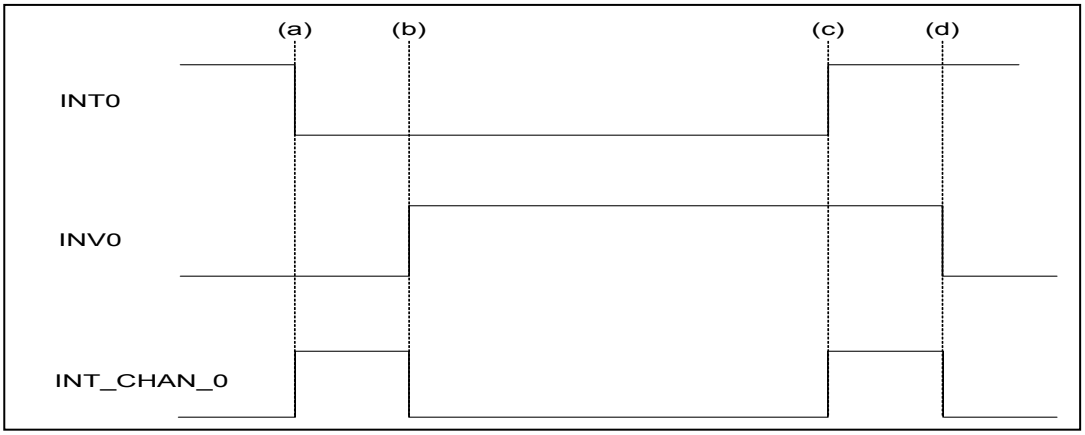
2.3.3 Initial_high, active_low Interrupt source

If INT0 (8254 counter0 output) is an initial_high, active_low signal (depending on 8254 counter mode), the interrupt service routine should use INV0 to invert/non-invert INT0 to generate a high_pulse in the following manner: (Refer to DEMO3.C)

Initial settings:

```
now_int_state=1;                    /* initial state for INT0      */
outportb(wBase+0x2a,0);             /* select the inverted INT0  */
```

```
void interrupt irq_service()
{
if (now_int_state==1)           /* now INT0 is changed to LOW          */(a)
   {                            /* --> INT_CHAN_0=!INT0=HIGH now       */
   COUNT_L++;                   /* find a LOW_pulse (INT0)             */
   If((inport(wBase+7)&1)==0)   /* the INT0 is still fixed in LOW      */
     {                          /* → need to generate a high_pulse   */
     outportb(wBase+0x2a,1);    /* INV0 select the non-inverted input */(b)
                                /* INT_CHAN_0=INT0=LOW -->             */
                                /* INT_CHAN_0 generate a high_pulse    */
     now_int_state=0;           /* now INT0=LOW                        */
     }
   else now_int_state=1;        /* now INT0=HIGH                       */
                                /* don't have to generate high_pulse   */
   }
else                            /* now INT0 is changed to HIGH         */(c)
   {                            /* --> INT_CHAN_0=INT0=HIGH now        */
   COUNT_H++;                   /* find a HIGH_pulse (INT0)            */
   If((inport(wBase+7)&1)==1)   /* the INT0 is still fixed in HIGH     */
     {                          /* need to generate a high_pulse       */
     outportb(wBase+0x2a,0);    /* INV0 select the inverted input     */(d)
                                /* INT_CHAN_0=!INT0=LOW -->            */
                                /* INT_CHAN_0 generate a high_pulse    */
     now_int_state=1;           /* now INT0=HIGH                       */
     }
   else now_int_state=0;        /* now INT0=LOW                        */
                                /* don't have to generate high_pulse   */
   }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



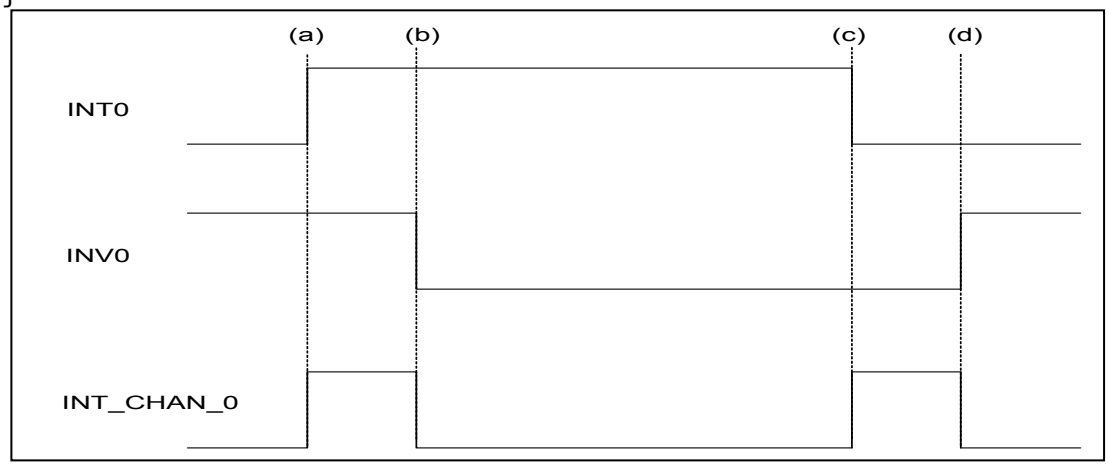
2.3.4 Initial_low, active_high Interrupt source

If INT0 (8254 counter0 output) is an initial_low, active_high signal (depending on the 8254 counter mode), the interrupt service routine should use INV0 to invert/non-invert INT0 to generate a high_pulse in the following manner: (Refer to DEMO4.C)

Initial setting:

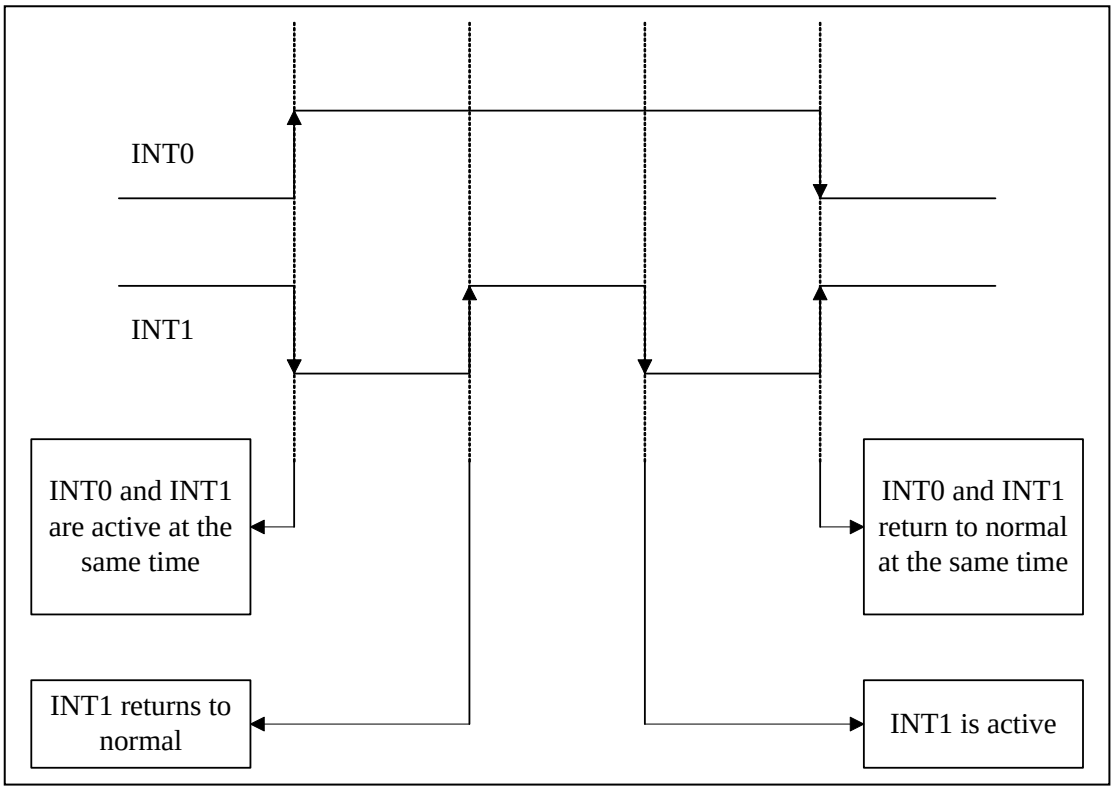
```
now_int_state=0;                    /* initial state for INT0          */
outportb(wBase+0x2a,1);            /* select the non-inverted INT0 */
```

```
void interrupt irq_service()
{
if (now_int_state==1)          /* now INT0 is changed to LOW          */(c)
   {                           /* --> INT_CHAN_0=!INT0=HIGH now       */
   COUNT_L++;                  /* find a LOW_pulse (INT0)             */
   If((inport(wBase+7)&1)==0)/* the INT0 is still fixed in LOW      */
     {                         /* → need to generate a high_pulse    */
     outportb(wBase+0x2a,1); /* INV0 select the non-inverted input */(d)
                               /* INT_CHAN_0=INT0=LOW -->             */
                               /* INT_CHAN_0 generate a high_pulse    */
     now_int_state=0;          /* now INT0=LOW                        */
     }
   else now_int_state=1;       /* now INT0=HIGH                       */
                               /* don't have to generate high_pulse   */
   }
else                           /* now INT0 is changed to HIGH         */(a)
   {                           /* --> INT_CHAN_0=INT0=HIGH now        */
   COUNT_H++;                  /* find a High_pulse (INT0)            */
   If((inport(wBase+7)&1)==1)/* the INT0 is still fixed in HIGH     */
     {                         /* need to generate a high_pulse       */
     outportb(wBase+0x2a,0); /* INV0 select the inverted input      */(b)
                               /* INT_CHAN_0=!INT0=LOW -->            */
                               /* INT_CHAN_0 generate a high_pulse    */
     now_int_state=1;          /* now INT0=HIGH                       */
     }
   else now_int_state=0;       /* now INT0=LOW                        */
                               /* don't have to generate high_pulse   */
   }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



2.3.5 Multiple Interrupt Sources

Assume: INT0 is initial Low and active High,
INT1 is initial High and active Low
as below:



Refer to DEMO5.C for the source program. **All of these falling-edge and rising-edge can be detected using DEMO5.C.**

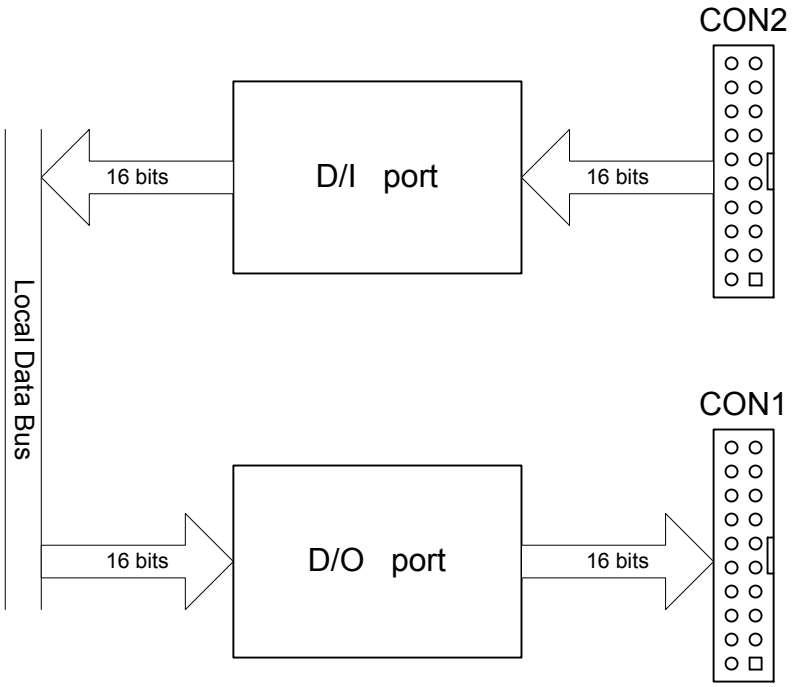
Note:

When the interrupt is active, the user program has to identify the active signals. These signals may all be active at the same time, so the interrupt service routine has to service all active signals at the same time.

```
/* ------------------------------------------------------------ */
/* Note : 1.The hold_time of INT_CHAN_0 & INT_CHAN_1 must long   */
/*          enoug.                                              */
/*        2.The ISR must read the interrupt status again to     */
/*          identify the active interrupt source.               */
/*        3.The INT_CHAN_0 & INT_CHAN_1 can be active at the same */
/*          time.                                               */
/* ------------------------------------------------------------ */

void interrupt irq_service()
{

/* now ISR can not know which interrupt is active               */
new_int_state=inportb(wBase+7)&0x03;      /* read all interrupt   */
                                          /* signal state         */
int_c=new_int_state^now_int_state;        /* compare new_state to */
                                          /* old_state            */

if ((int_c&0x01)==1)             /* INT_CHAN_0 is active         */
   {
   if ((new_int_state&1)==0)     /* INT0 change to low now       */
      {
      INT0_L++;
      }
   else                          /* INT0 change to high now      */
      {
      INT0_H++;
      }
   invert=invert^1;              /* generate high_pulse          */
   }

if ((int_c&0x02)==2)             /* INT_CHAN_1 is active         */
   {
   if ((new_int_state&2)==0)     /* INT1 change to low now       */
      {
      INT1_L++;
      }
   else                          /* INT1 change to high now      */
      {
      INT1_H++;
      }
   invert=invert^2;              /* generate high_pulse          */
   }

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

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

}
```

2.4 D/I/O Block Diagram

The PISO-DA/PIO-DA series provides 16 digital input channels and 16 digital output channels, and all signal levels are TTL compatible. The connection diagram and block diagram are as follows:

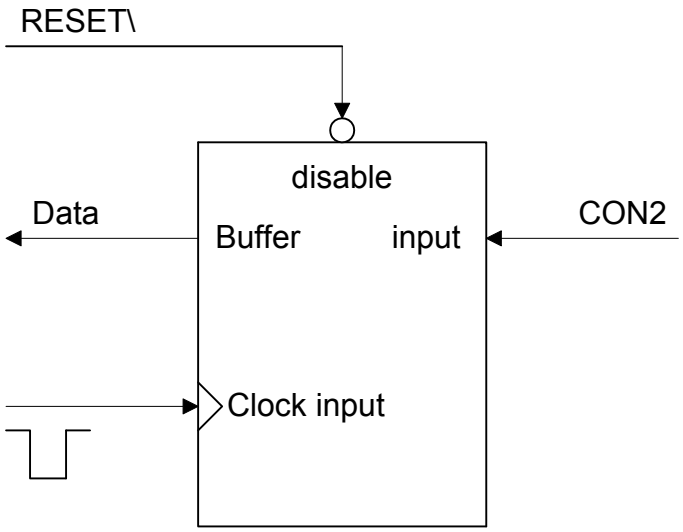


The D/I Port can be connected to a DB-16P, which is a 16-channel isolated digital input daughter board. The D/O Port can be connected to either a DB-16R or a DB-24PR. The DB-16R is a 16 channels relay output board. The DB-24PR is a 24 channels power relay output board.

2.4.1 DI Port Architecture (CON2)

When the PC is powered up, all DI port (CON2) operation are disabled. The enabled/disabled status of a DI port is controlled by the RESET\ signal. Refer to Sec. 3.3.1 for more information about the RESET\ signal.

- The RESET\ signal is in the Low-state → all DI operations are disabled
- The RESET\ signal is in the High-state → all DI operations are enabled



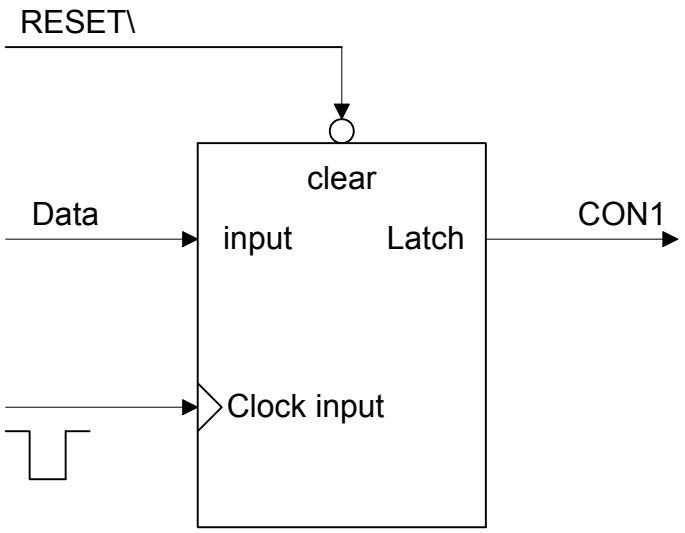
D/I buffer CKT

2.4.2 DO Port Architecture (CON1)

When the PC is powered up, the states of all DO channels are cleared low. The RESET\ signal is used to clear the DO states. Refer to Sec. 3.3.1 for more information about the RESET\ signal.

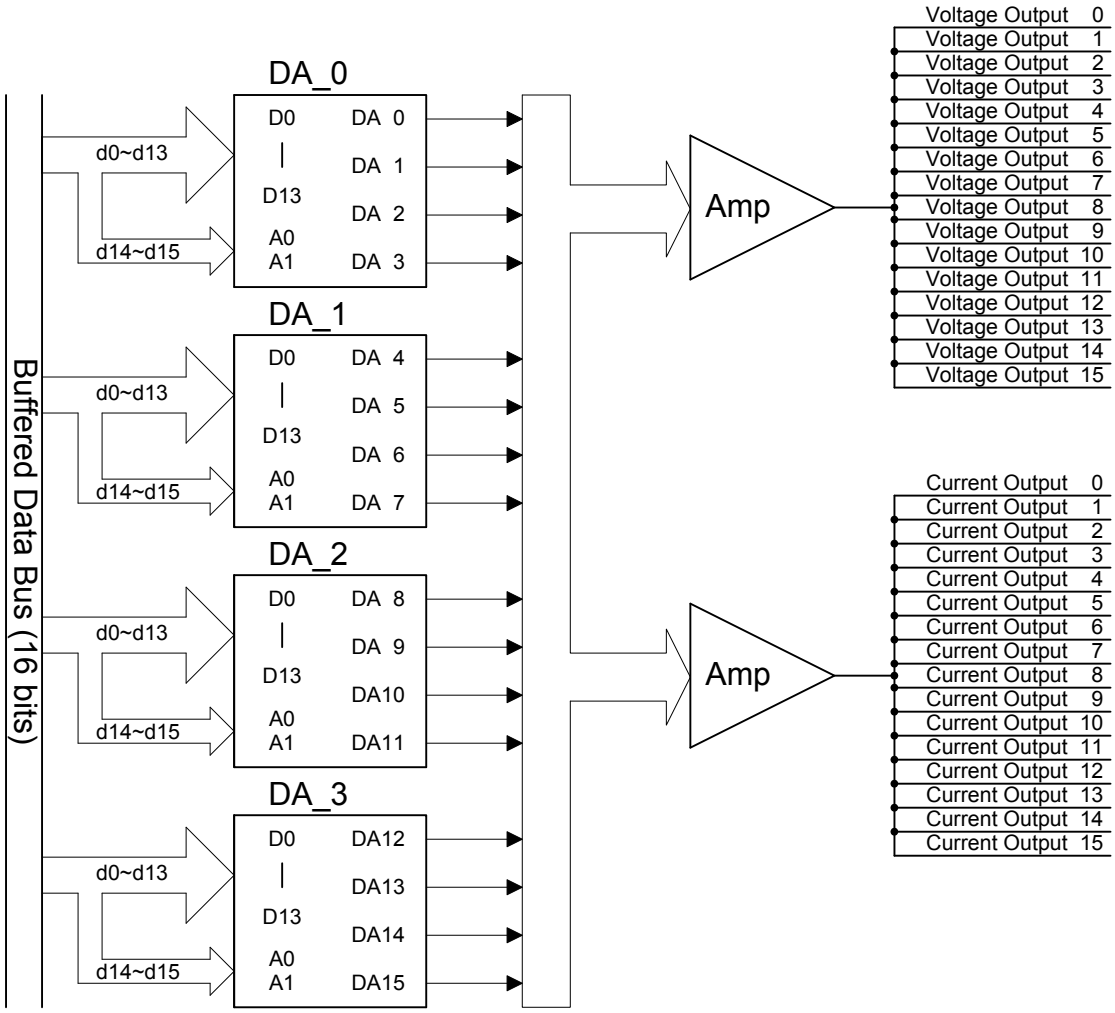
- The RESET\ signal is in the Low-state → all DO channels are cleared to the low state

The block diagram of DO is as follows:



D/O buffer CKT

2.5 D/A Architecture



The PIO-DA4/8/16 provides 4/8/16 channels of double-buffered digital to analog output and provides voltage output and current output simultaneously.

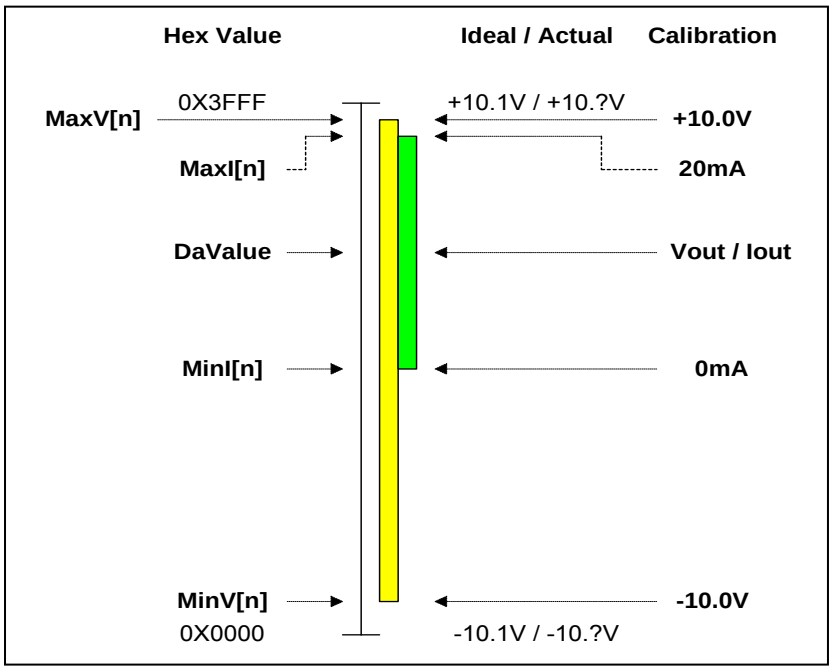
2.6 D/A Conversion Operations

The D/A converters on PISO-DA/PIO-DA series cards use 14-bit resolution, so the digital data values range from 0x0000 to 0x3fff. The hardware is designed to output voltage in a range from -10.1 ~ +10.1 volts, as follows:

0x0000 → about –10.1 volts
0x3FFF → about +10.1 volts

In a conventional design, there will be some VRs that need to be adjusted so that the voltage output for 0x0000 = -10.0 V and 0x3fff = +10.0 V. In addition, these VRs also have to be adjusted so that the current output for 0x1fff = 0 mA and 0x3fff = 20 mA. In conventional designs, these VRs are commonly used for voltage/current output, so the user has to perform some calibration when changing from voltage to current. Also, if these VRs are changed, the user has to re-perform the calibration. This procedure is complex and creates a heavy workload. The PISO-DA/PIO-DA series uses software calibration to replace this complex procedure in the following manner:

- For each voltage output channel, find two hex values MaxV[n] and MinV[n] (stored in the onboard EEPROM). MaxV[n] is mapped to exactly +10 V and MinV[n] is mapped to exactly –10 V.
- For each current output channel, also find two hex values MaxI[n] and MinI[n] (stored in the onboard EEPROM). MaxI[n] is mapped to exactly 20 mA and MinI[n] is mapped to exactly 0 mA.



Consequently, the software can be used to calibrate the analog output without the need for any hardware Trim-pot adjustment. For example,

Channel n	MinV[n]	MaxV[n]	MinI[n]	MaxI[n]
0	134	16297	8180	15943
1	137	16293	8172	15976
2	132	16296	8199	15949
3	134	16391	8177	15963
4	135	16298	8165	15955
5	131	16292	8150	15947
6	136	16295	8172	15968
7	134	16297	8163	15961
8	134	16294	8188	15959
9	132	16295	8169	15948
10	135	16298	8172	15946
11	133	16296	8177	15975
12	131	16292	8159	15942
13	134	16297	8173	15973
14	132	16293	8168	15949
15	133	16295	8175	15965

If the user wants to send Vout(volts) to Channel n, the calibrated hex value, DaValue, sent to D/A converter can be calculated in the following way:

```
DeltaV[n]=20.0/(MaxV[n]–MinV[n]);          /* DeltaV[n]=volts per count at channel_n */
DaValue=(Vout+10.0)/DeltaV[n]+MinV[n];  /* DaValue=Hex value to send to the D/A */
pio_da16_da(n,DaValue);                          /* send the DaValue to Channel n  */
```

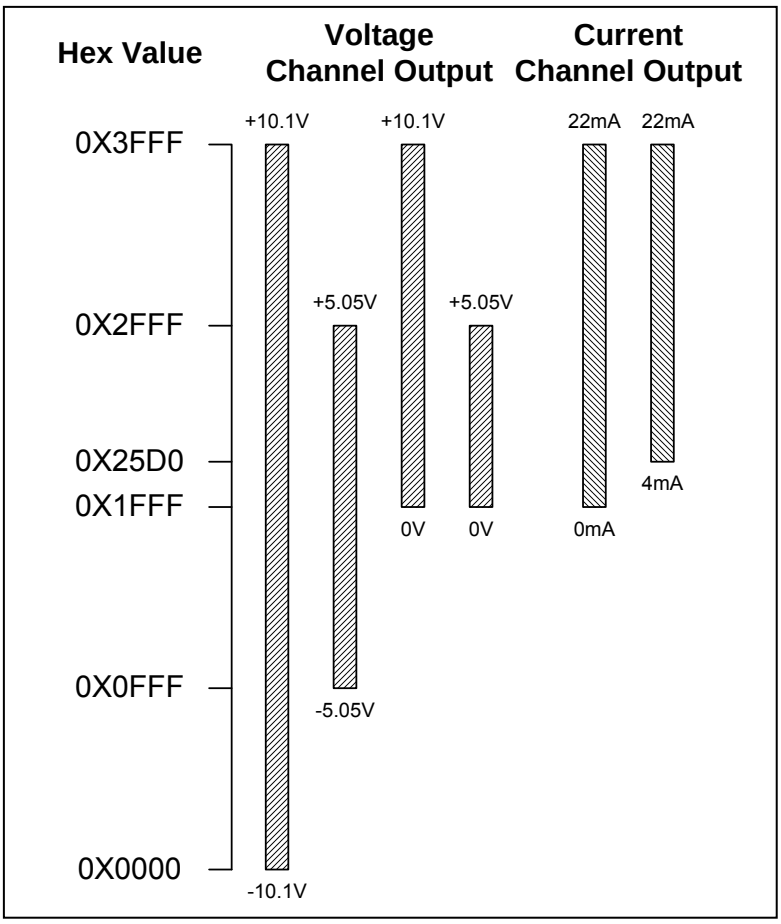
If the user wants to send Iout(mA) to Channel n, the calibrated hex value, DaValue, sent to the D/A converter can be calculated in the following way: (Refer to DEMO9.C)

```
DeltaI[n]=20.0/(MaxI[n]–MinI[n]);          /* DeltaI[n]=mA per count at channel_n */
DaValue=Iout/DeltaI[n]+MinI[n];            /* DaValue=Hex value send to D/A   */
pio_da16_da(n,DaValue);                          /* send the DaValue to Channel n  */
```

Refer to DEMO7.C and DEMO9.C for more information.

2.6.1 Output Range and Resolution

The voltage output range for PISO-DA/PIO-DA series cards is always ±10.1 V, and the current output range is always 0~22 mA, as illustrated below:



The resolution for each range is as follows:

Configuration	Equivalent Bits	Resolution
-10 V ~ +10 V	14-bit	1.22 mV
0 V ~ 10 V	13-bit	1.22 mV
-5 V ~ +5 V	13-bit	1.22 mV
0 V ~ +5 V	12-bit	1.22 mV
0 mA ~ 20 mA	13-bit	2.70 μA
4 mA ~ 20 mA	13-bit	2.70 μA

2.6.2 ±10 V Voltage Output

The voltage output for PISO-DA/PIO-DA series cards is always in the range of ±10.1 V. If the user needs to output a voltage in the range of ±10 V, the software calibration is the same as that described in Sec. 2.6. Consequently, Vout will be in the range of ±10 V, so the DaValue will approximately be from 0x0000 to 0x3fff, which means that the resolution is about 14 bits.

2.6.3 ±5 V Voltage Output

The voltage output for PISO-DA/PIO-DA series cards is always in the range of ±10.1 V. If the user needs to output a voltage in the range of ±5 V, the software calibration is same as that described in Sec. 2.6. Consequently, Vout will be in range of +5 V, so the DaValue will approximately be from 0x0fff to 0x2fff, which means that the resolution is about 13 bits.

2.6.4 0~10 V Voltage Output

The voltage output for PISO-DA/PIO-DA series cards is always in the range of ±10 V.1. If the user needs to output a voltage in the range of 0~10 V, the software calibration is the same as that described in Sec.2.6. Consequently, Vout will be in the range of 0~10 V, so the DaValue will approximately be from 0x1fff to 0x3fff, which means the resolution is about 13 bits.

2.6.5 0~5 V Voltage Output

The voltage output for PISO-DA/PIO-DA series cards is always in the range of ±10.1 V. If the user needs to output a voltage in the range of 0~5 V, the software calibration is the same as that described in Sec. 2.6. Consequently, Vout will be in the range 0~5 V, so the DaValue will approximately be from 0x1fff to 0x2fff, which means that the resolution is about 12 bits.

2.6.6 0~20 mA Current Output

The current output for PISO-DA/PIO-DA series cards is always in the range of 0~22 mA. If the user needs to output a current in the range of 0~20 mA, the software calibration is the same as that described in Sec. 2.6. Iout will be in the range of 0~20 mA, so the DaValue will approximately be from 0x1fff to 0x3fff, which means that the resolution is about 13 bits.

2.6.7 4~20 mA Current Output

The current output for PISO-DA/PIO-DA series cards is always in the range of 0~22 mA. If the user needs to output a current in the range of 4~20 mA, the software calibration is the same as that described in Sec. 2.6. Iout will be in the range of 4~20 mA, so the DaValue will approximately be from 0x2600 to 0x3fff, which means that the resolution is about 13 bits.

2.6.8 No VR & No Jumper Design

In a conventional 12-bit D/A board, for example the A-626/A-628, there are many jumpers that allow the following functions to be performed:

(1) Selecting the reference voltage (internal -10/-5/or external)
(2) Selecting unipolar/bipolar (0-10 V or ±10 V)
(3) Selecting different output ranges (0-10 V or 0-5 V)

There are also many VRs that allow the following functions to be performed:

(1) Adjustment of the output voltage offset
(2) Full-scale adjustment of the output voltage
(3) Adjustment of the output current offset
(4) Full-scale adjustment of the output current

There are so many VRs and jumpers that if makes QC and re-calibration very difficult. Every step must be handled manually, meaning that calibrating these D/A boards is not an enjoyable task.

When we designed the PISO-DA/PIO-DA series, we tried to remove many/the these majorities of VRs and jumpers, but still retain the same precision and performance. In the long run, we selected a 14-bit D/A converter and adapted the software calibration to provide at least the same performance and precision as the A-626/A-628:

Configuration	Equivalent Bits	Resolution
-10 V ~ +10 V	14-bit	1.22 mV
0 V ~ 10 V	13-bit	1.22 mV
-5 V ~ +5 V	13-bit	1.22 mV
0 V ~ +5 V	12-bit	1.22 mV
0 mA ~ 20 mA	13-bit	2.70 μA
4 mA ~ 20 mA	13-bit	2.70 μA

- All these VRs and jumpers have been removed.
- All calibrations can be performed using software.
- All channel configurations can be selected using software, meaning that there is no need to change any hardware.
- Precision is at least the same as the A-626/A628.
- All 16 channels can be configured and used in different configurations at the same time. (For example, channel_0=±10 V, channel_1=4~20 mA, channel_2=0~5 V, etc)
- All these features can be implemented on a small, compact and reliable half-size PCB.

2.6.9 Factory Software Calibration

It is recommended that a 16-bit A/D card is used to calibrate the PISO-DA/PIO-DA series cards. The I-7000 series is a set of precise remote control modules and the I-7017 is an 8-channel 16-bit precision A/D module (24-bit sigma-delta A/D converter). Two I-7017 modules are used for voltage output calibration and another two for current output calibration.

The steps required to calibrate the voltage for channel_n are as follows:

Step 1: DaValue=0
Step 2: Send the DaValue to channel_n on the PIO/PISO card
Step 3: Measure the voltage of channel_n on the I-7017
If this value is >= -10 V, then go to Step 5
Step 4: Increase the DaValue, then return to Step 2
Step 5: MinV[n]=DaValue-1
Step 6: DaValue=0x3fff
Step 7: Send the DaValue to channel_n on the PIO/PISO card
Step 8: Measure the voltage of channel_n on the I-7017
If this value is >= +10 V, then go to Step 10
Step 9: Increase the DaValue, then return to Step 7
Step 10: MaxV[n]=DaValue

Note: MinV[n] and MaxV[n] are described in Sec. 2.6

The steps required to calibrate the current for channel_n are as follows:

Step 1: DaValue=0x1fff
Step 2: Send the DaValue to hannel_n on the PIO/PISO card
Step 3: Measure the current of channel_n on the I-7017
If this value is >= 0 mA, then go to Step 5
Step 4: Increase the DaValue, the return to Step 2
Step 5: MinI[n]=DaValue-1
Step 6: DaValue=0x3fff
Step 7: Send the DaValue to channel_n on the PIO/PISO card
Step 8: Measure the current of channel_n on the I-7017
If this value is >= 20 mA, than go to Step 10
Step 9: Increase the DaValue, the return to Step 7
Step 10: MaxI[n]=DaValue

Note: MinI[n] and MaxI[n] are described in Sec. 2.6

2.6.10 User Software Calibration

The users can perform calibration themselves using a voltage meter and a current meter.

Step 1: Run DEMO12.EXE
Step 2: Select the card number for the PIO/PISO card that you want to calibrate
Step 3: Select the item (MinV[n]/MaxV[n]/MinI[n]/MaxI[n]) that you want to calibrate
Step 4: Measure the analog output using the voltage meter or the current meter and decide whether to increase or decrease the DaValue. The DaValue will immediately be sent to the D/A converter. The user can then determine the correct value for DaValue that is mapped to the accurate output value.
Step 5: Repeat Step 4 for each channel

After this procedure, the new data for MinV[n]/MaxV[n]/MinI[n]/MaxI[n] will be stored in the onboard EEPROM.

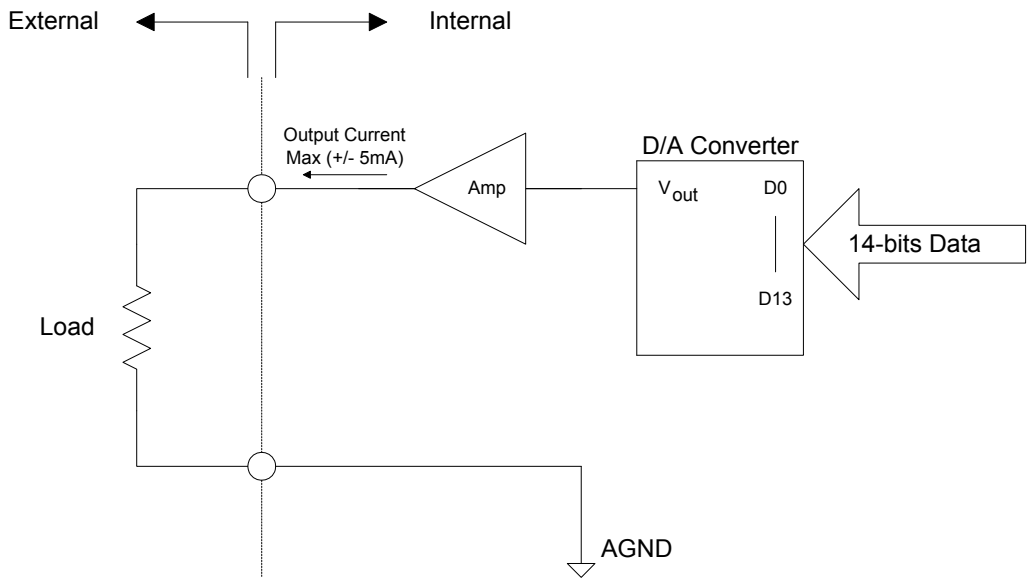
DEMO10.EXE can be executed to back up the old calibration data to "A:\DA16.DAT" before a new calibration is performed.

If an error occurs while the new calibration is being performed, DEMO11.EXE can be executed to download the data from "A:\DA16.DAT" to the EEPROM.

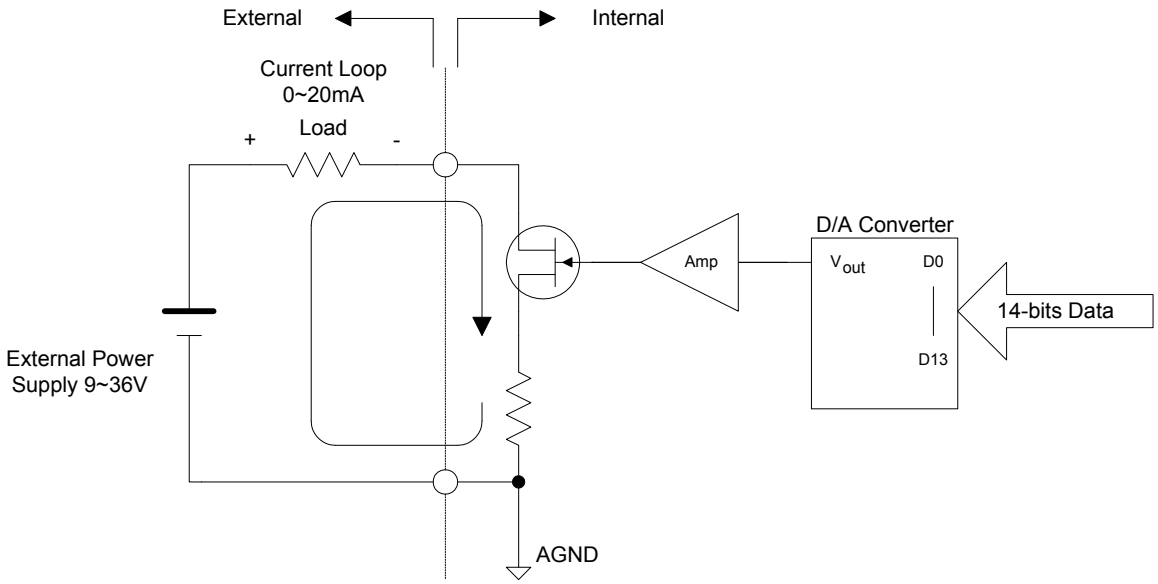
DEMO10.EXE → Save the old calibration data
DEMO11.EXE → Download the old calibration data
DEMO12.EXE → Perform a new calibration

Note:
Demo10.exe, Demo11.exe and Demo12.exe are DOS programs that can run on either a pure DOS or a FreeDOS (http://www.freedos.org/) system. These DOS programs do not work on the DOS command prompt within Windows.

2.6.11 Voltage Output Connection



2.6.12 Current Output Connection



2.7 Pin Assignments

■ **CON1: Digital Output Connector**

Pin	Name	Pin	Name
1	Digital Output 0	2	Digital Output 1
3	Digital Output 2	4	Digital Output 3
5	Digital Output 4	6	Digital Output 5
7	Digital Output 6	8	Digital Output 7
9	Digital Output 8	10	Digital Output 9
11	Digital Output 10	12	Digital Output 11
13	Digital Output 12	14	Digital Output 13
15	Digital Output 14	16	Digital Output 15
17	PCB ground	18	PCB ground
19	PCB +5 V	20	PCB +12 V

■ **CON2: Digital input connector**

Pin	Name	Pin	Name
1	Digital Input 0	2	Digital Input 1
3	Digital Input 2	4	Digital Input 3
5	Digital Input 4	6	Digital Input 5
7	Digital Input 6	8	Digital Input 7
9	Digital Input 8	10	Digital Input 9
11	Digital Input 10	12	Digital Input 11
13	Digital Input 12	14	Digital Input 13
15	Digital Input 14	16	Digital Input 15
17	PCB ground	18	PCB ground
19	PCB +5 V	20	PCB +12 V

Note: All Signals are TTL Compatible

High (1)	2.0 ~ 5.0 V(Voltage over 5.0V will damage the device)
None Define	2.0 V ~ 0.8 V
Low(0)	Under 0.8 V

■ **CON3: Analog Output Connector for the PIO-DA4/DA8/DA16 and PISO-DA4U/DA8U/DA16U.**

Pin	Name	Pin	Name
1	Voltage Output 0	20	Current Output 0
2	Voltage Output 1	21	Current Output 1
3	Voltage Output 2	22	Current Output 2
4	Voltage Output 3	23	Current Output 3
5	Analog ground	24	Analog ground
6	Voltage Output 4	25	Current Output 4
7	Voltage Output 5	26	Current Output 5
8	Voltage Output 6	27	Current Output 6
9	Voltage Output 7	28	Current Output 7
10	Analog ground	29	Analog ground
11	Voltage Output 8	30	Current Output 8
12	Voltage Output 9	31	Current Output 9
13	Voltage Output 10	32	Current Output 10
14	Voltage Output 11	33	Current Output 11
15	Analog ground	34	Current Output 12
16	Voltage Output 12	35	Current Output 13
17	Voltage Output 13	36	Current Output 14
18	Voltage Output 14	37	Current Output 15
19	Voltage Output 15		

■ CON3: Analog Output Connector for the PIO-DA4U/DA8U/DA16U

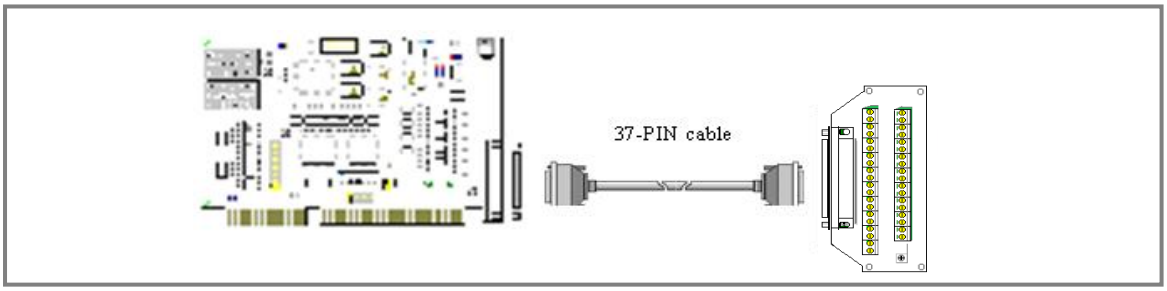
Pin	Name	Pin	Name
1	Voltage Output 0	20	Current Output 0
2	Voltage Output 1	21	Current Output 1
3	Voltage Output 2	22	Current Output 2
4	Voltage Output 3	23	Current Output 3
5	Analog ground	**24**	**N/A**
6	Voltage Output 4	25	Current Output 4
7	Voltage Output 5	26	Current Output 5
8	Voltage Output 6	27	Current Output 6
9	Voltage Output 7	28	Current Output 7
10	Analog ground	**29**	**N/A**
11	Voltage Output 8	30	Current Output 8
12	Voltage Output 9	31	Current Output 9
13	Voltage Output 10	32	Current Output 10
14	Voltage Output 11	33	Current Output 11
15	Analog ground	34	Current Output 12
16	Voltage Output 12	35	Current Output 13
17	Voltage Output 13	36	Current Output 14
18	Voltage Output 14	37	Current Output 15
19	Voltage Output 15		

Note: Pin 24 and Pin 29 are not Analog Ground on PIO-DA Universal board. Wiring of applications transferred from PIO-DA to PIO-DA Universal board need be rearranged.

2.8 Daughter Boards

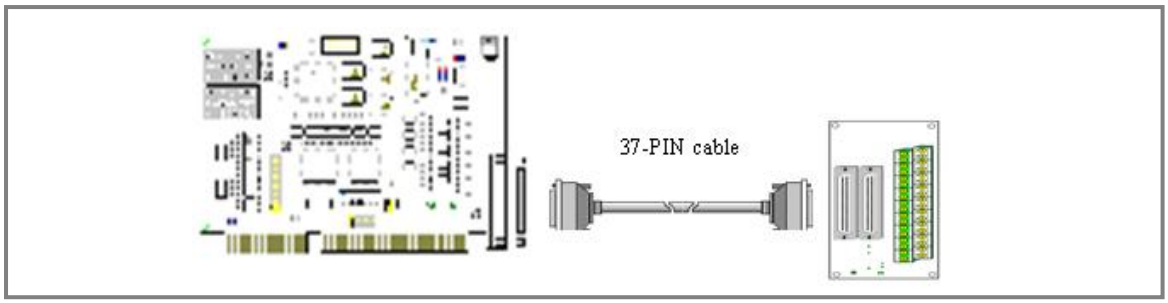
2.8.1 DB-37

The DB-37 is a general-purpose daughter board for D-sub 37-pin devices, and is designed for easy wiring.



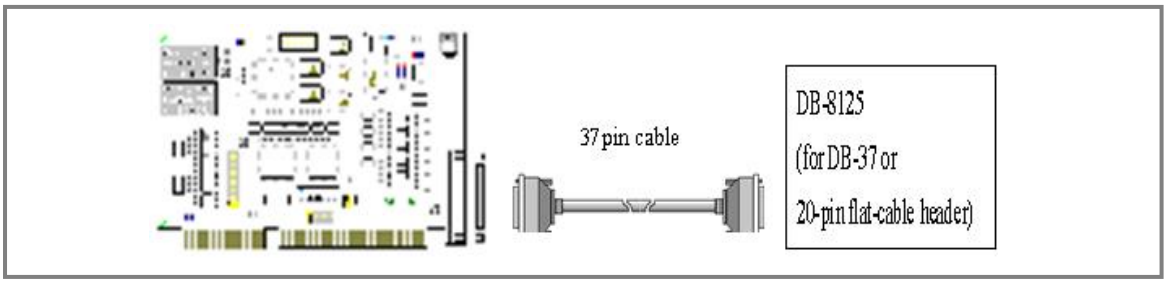
2.8.2 DN-37

The DN-37 is a general-purpose daughter board for the DB-37 using DIN-Rail Mounting, and is designed for easy wiring.



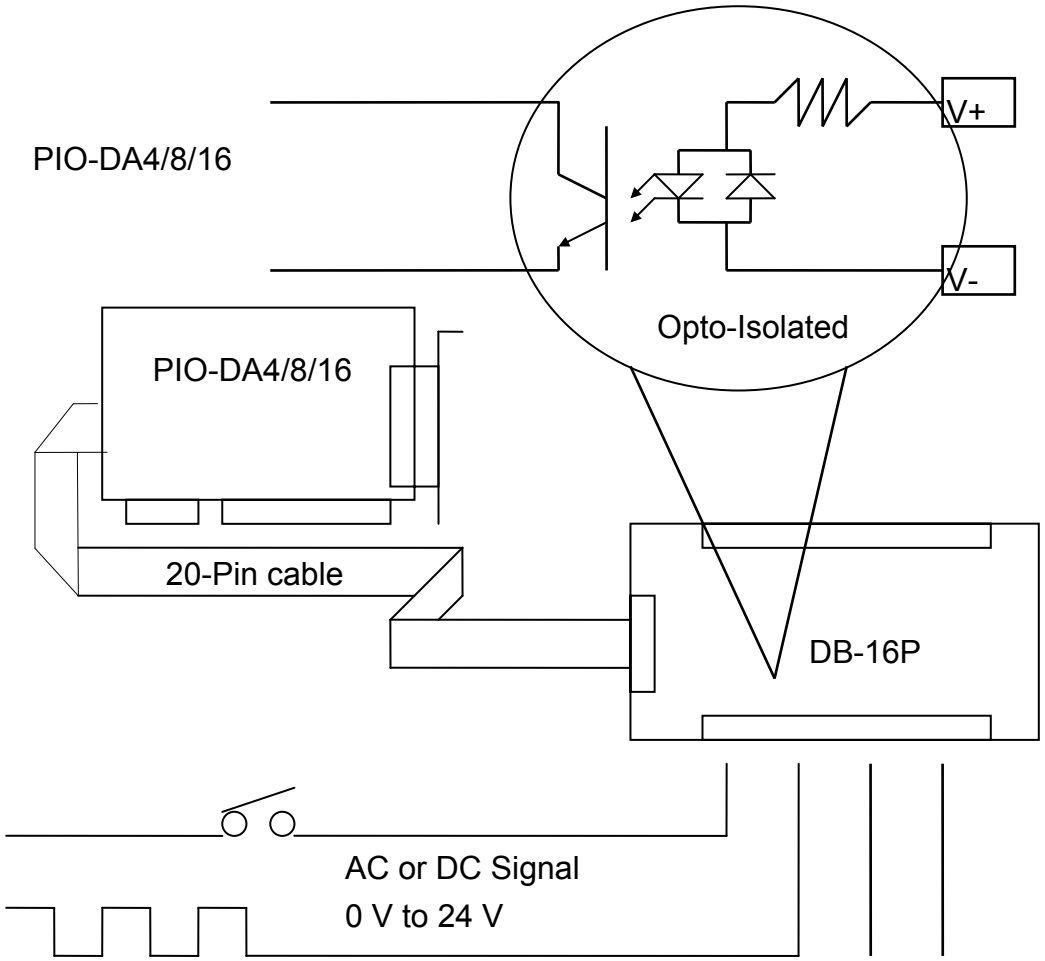
2.8.3 DB-8125

The DB-8125 is a general-purpose screw terminal board, and is designed for easy wiring. The DB-8128 uses one DB-37 and two 20-pin flat-cable headers.



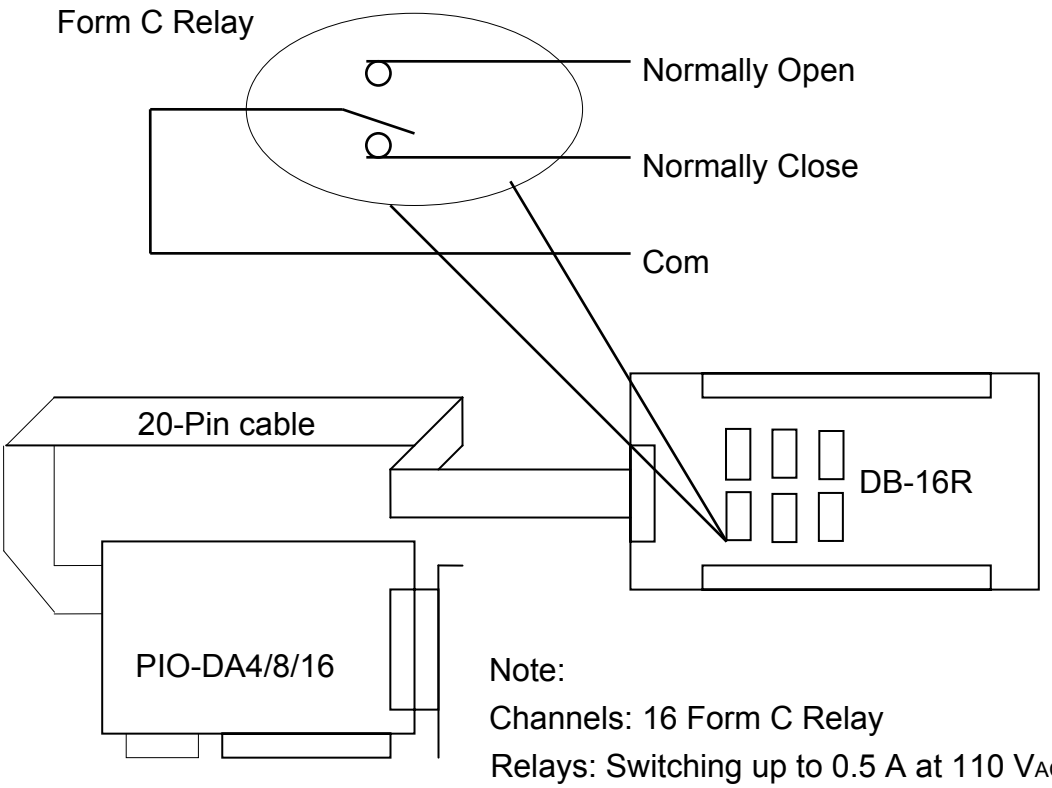
2.8.4 DB-16P Isolated Input Board

The DB-16P is a 16-channel isolated digital input daughter board with optically isolated inputs that consist of a bi-directional opto-coupler with a resistor to allow current sensing. The DB-16P can be used to sense DC signals from TTL levels up to 24 V or to sense a wide range of AC signals. This board can also be used to isolate the host computer from large common-mode voltages, ground loops and transient voltage spikes that often occur in industrial environments.



2.8.5 DB-16R Relay Board

The DB-16R is a 16-channel relay output board that consists of 16 Form C relays that enable efficient switching of loads through a programmable control. The connectors and functionally is compatible with 785 series boards, but contains an industrial-type terminal block. The relays are powered by applying a 5 V signal to the appropriate relay channel via the 20-pin flat cable connector. There are 16 LEDs, one for each relay, which are illuminated when their associated relay is activated. To avoid overloading the power supply of your PC, this board includes a screw terminal to allow an external power supply to be connected.

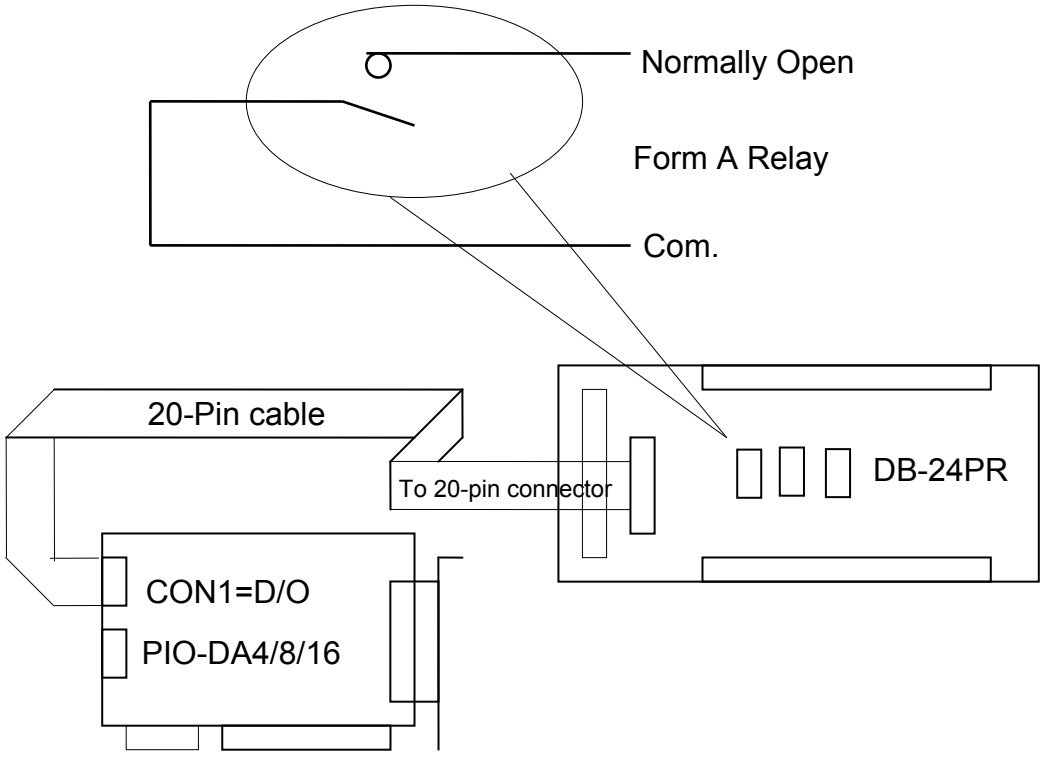


Note:
Channels: 16 Form C Relay
Relays: Switching up to 0.5 A at 110 V_{AC}
or 1 A at 24 V_{DC}

2.8.6. DB-24PR, DB-24POR, DB-24C

DB-24PR	24 x power relays, 5 A/250 V
DB-24POR	24 x PhotoMOS relays, 0.1 A/350 V_{AC}
DB-24C	24 x open collectors, 100 mA per channel, 30 V max.

The DB-24PR is a 24-channel power relay output board that consists of 8 Form C and 16 Form A electromechanical relays that enable efficient switching of loads though a programmable control. The contact of each relay can control a 5 A load at 250 V_{AC}/30 V_{DC}. The relay is powered by applying a 5 V signal to the appropriate relay channel via the 20-pin flat cable connector, which only uses 16 relays or 50-pin flat cable connector. (OPTO-22 compatible, for the DIO-24 series). There are 24 LEDs, one for each relay, which are illuminated when their associated relay is activated. To avoid overloading the power supply of your PC, this board requires a +12 V_{DC} or +24 V_{DC} external power supply.



Note:

1. **50-Pin connector (OPTO-22 compatible), for DIO-24, DIO-48, DIO-144, PIO-D144, PIO-D96, PIO-D56, PIO-D48, PIO-D24**
2. **20-Pin connector for 16 channel digital output, A-82X, A-62X, DIO-64, ISO-DA16/DA8, PIO-D56, PIO-DA4/8/16**
3. **Channel: 16 Form A Relay , 8 Form C Relay**
4. **Relay: Switching up to 5 A at 110 V_{AC}/5 A at 30 V_{DC}**

2.8.7. 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:
There is no 50-pin flat-cable header on the PIO-DA4/8/16. The PIO-DA4/8/16 has one DB-37 connector and two 20-pin flat-cable headers.

3. I/O Control Register

3.1 How to Find the I/O Address

The Plug & Play BIOS will assign an appropriate I/O address for each PIO/PISO series card during the power-on stage. The fixed IDs for the PIO/PISO series cards are shown in the tables below:

	PIO-DA4 PIO-DA8 PIO-DA16	PIO-DA4 PIO-DA8 PIO-DA16	PIO-DA4U PIO-DA8U PIO-DA16U	PISO-DA4U PISO-DA8U PISO-DA16U
Version	Rev1.0 - Rev3.0	Rev4.0 ~ above	Rev1.0 ~ above	Rev1.0 ~ above
Vendor ID	0xE159	0xE159	0xE159	0xE159
Device ID	0x02	0x01	0x01	0x01
Sub Vendor ID	0x80	0x4180	0x4180	0x4180
Sub Device ID	0x04	0x00	0x00	0x00
Sub-Aux ID	0x00	0x00	0x00	0x00

ICP DAS provides the following necessary functions:

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

All functions are defined in the PIO.H include file. Refer to Chapter 4 for more information. The relevant driver information is as follows:

1. Allocated resource information:

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

2. PIO/PISO identification information:

- wSubVendor: The subVendor ID of this board
- wSubDevice: The subDevice ID of this board
- wSubAux: The subAux ID of this board

3. Physical slot information:

- wSlotBus: The bus number of the slot used by this board
- wSlotDevice: The device number of the slot used by this board

The **PIO_PISO.EXE utility program** can be used to detect and display the details of all PIO/PISO cards installed in this PC. Refer to Sec. 5.3 for more information.

3.1.1 PIO_DriverInit

PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux)

- wBoards=0 to N → The number of boards found in this PC
- wSubVendor → The subVendor ID of the board you are seeking
- wSubDevice → The subDevice ID of the board your are seeking
- wSubAux → The subAux ID of the board you are seeking

This function can be used to detect all PIO/PISO series cards within your system. Implementation is based on the PCI Plug & Play mechanism. The function locates all PIO/PISO series cards installed in this system and save the relevant resource information in the library.

■ Sample program 1: Detect all PIO-DA4/8/16 series cards installed in this PC

```
/* Step 1: Detect all PIO-DA16/8/4 series cards installed in this PC */
wSubVendor=0x80; wSubDevice=4; wSubAux=0x00; /* For PIO-DA4/8/16
series cards*/

wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("There are %d PIO-DA16 Cards in this PC\n",wBoards);

/* Step 2: Save the resource information for all PIO-DA4/8/16 series cards
installed in this PC */
for (i=0; i<wBoards; i++)
{
 PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wID1,&wID2,&wID3,&wID4,
                           &wID5);
 printf("\nCard_%d: wBase=%x, wIrq=%x", i,wBase,wIrq);
 wConfigSpace[i][0]=wBaseAddress; /*Save the resource information for this card */
 wConfigSpace[i][1]=wIrq;         /*Save the resource information for this card */
 }
```

- Sample program 2: Detect all PIO/PISO cards installed in this PC. (Refer to Sec. 4.1 for more information)

```
/* Step 1: Detect all PIO/PISO series cards installed in this PC */
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);   /* Detect all PIO_PISO series
cards */

printf("\nThere are %d PIO_PISO Cards in this PC",wBoards);

if (wBoards==0 ) exit(0);

/* Step 2: Save the resource information for all PIO/PISO cards installed in this PC */
printf("\n----------------------------------------------------");

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

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

  ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
  }
```

3.1.2 PIO_GetConfigAddressSpace

PIO_GetConfigAddressSpace(wBoardNo,*wBase,*wIrq, *wSubVendor,*wSubDevice,*wSubAux,*wSlotBus, *wSlotDevice)

- wBoardNo=0 to N → The total number of boards using the PIO_DriverInit(...) function
- wBase → The base address of the board control word
- wIrq → The allocated IRQ channel number for this board
- wSubVendor → The subVendor ID of this board
- wSubDevice → The subDevice ID of this board
- wSubAux → The subAux ID of this board
- wSlotBus → The bus number of the slot used by this board
- wSlotDevice → The device number of the slot used by this board

The function can be used to save the resource information for all PIO/PISO cards installed in this system. The application program can then directly control all functions of the PIO/PISO series card.

Detect the configuration address space for your PIO-DA4/8/16 series cards

```
/* Step 1: Detect all PIO-DA4/8/16 series cards */
wSubVendor=0x80; wSubDevice=4; wSubAux=0x00; /*For PIO_DA4/8/16
series cards */

wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("There are %d PIO-DA16/8/4 Cards in this PC\n",wBoards);

/* Step 2: Save the resource information for all PIO-DA4/8/16 series cards
installed in this PC */
for (i=0; i<wBoards; i++)
 {
  PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&t1,&t2,&t3,&t4,&t5);
  printf("\nCard_%d: wBase=%x, wIrq=%x", i,wBase,wIrq);
  wConfigSpace[i][0]=wBaseAddress;/*Save the resource information for this card*/
  wConfigSpace[i][1]=wIrq;          /*Save the resource information for this card*/
 }
```

```
/* Step 3: Control the PIO-DA4/8/16 cards directly */
wBase=wConfigSpace[0][0];      /* get the base address for card_0 */
outport(wBase,1);              /* enable all D/I/O operations of card_0 */

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

3.1.3 Show_PIO_PISO

Show_PIO_PISO(wSubVendor,wSubDevice,wSubAux)

- wSubVendor → The subVendor ID of the board you are seeking
- wSubDevice → The subDevice ID of the board you are seeking
- wSubAux → The subAux ID of the board you are seeking

This function will display a text string showing these special subIDs, which are the same as those defined in the PIO.H include file.

The code for the demo program is as follows:

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

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

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

  }
```

3.2 The Assignment of the I/O Addresses

The Plug & Play BIOS will assign an appropriate I/O address for each PIO/PISO series card. If there is only one I/O board, the board will be identified as card_0. However, if there are two or more I/O boards in the system, identifying which board is card_0 becomes more difficult. The software driver can support a maximum of 16 boards.

Sometimes, it is difficult to find the card number. The easiest way to identify which card is card_0 is to use the wSlotBus and wSlotDevice functions in the following manner:

Step 1: Remove all PIO-DA4/8/16 series cards from the PC.

Step 2: Install a PIO-DA4/8/16 series card into PCI_slot1 on the PC and then run PIO_PISO.EXE. Record the results shown for wSlotBus1 and wSlotDevice1.

Step 3: Remove all PIO-DA4/8/16 series cards from the PC.

Step 4: Install a PIO-DA4/8/16 series card into PCI_slot2 on the PC and then run PIO_PISO.EXE again. Record the result shown for wSlotBus2 and wSlotDevice2.

Step 5: Repeat (3) and (4) for all PCI_slots and record the results shown for each wSlotBus and wSlotDevice.

A possible sample record:

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 procedure outlined above can be used to record all wSlotBus and wSlotDevice information for all slots in the PC. This mapping is fixed for each PC, and can then be used to identify a specific PIO-PISO card in the following manner:

Step 1: Record all wSlotBus and wSlotDevice information.

Step 2: Use the PIO_GetConfigAddressSpace(…) function to retrieve the wSlotBus and wSlotDevice information for the specified card.

Step 3: The specified PIO-PISO card can be identified from the two results.

3.3 The I/O Address Map

The I/O address for PIO-DA/PISO-DA series cards is automatically assigned by the ROM BIOS of the PC and provides Plug & Play capabilities for PIO/PISO series cards.

The PIO-DA/PISO-DA series I/O addresses are mapped as follows:

Address	Read	Write
wBase+0	Reserved	RESET\ control register
wBase+2	Reserved	Aux control register
wBase+3	Aux data register	Aux data register
wBase+5	Reserved	INT mask control register
wBase+7	Aux pin status register	Reserved
wBase+0x2a	Reserved	INT polarity control register
wBase+0xc0	Read 8254-counter0	Write 8254-counter0
wBase+0xc4	Read 8254-counter1	Write 8254-counter1
wBase+0xc8	Read 8254-counter2	Write 8254-counter2
wBase+0xcc	Read 8254 control word	Write 8254 control word
wBase+0xd4	Read the Card ID	Reserved
wBase+0xe0	Read the low byte of the D/I	DA_0 chip select
wBase+0xe4	Read the high byte of the D/I	DA_1 chip select
wBase+0xe8	Read the low byte of the D/I **(Only support the PIO-DA)**	DA_2 chip select
wBase+0xec	Read the high byte of the D/I **(Only support the PIO-DA)**	DA_3 chip select
wBase+0xf0	Read the low byte of the D/I **(Only support the PIO-DA)**	Write the low byte of the D/A
wBase+0xf4	Read the high byte of the D/I **(Only support the PIO-DA)**	Write the high byte of the D/A
wBase+0xf8	Read the low byte of the D/I **(Only support the PIO-DA)**	Write the low byte of the D/O
wBase+0xfc	Read high byte of the D/I **(Only support the PIO-DA)**	Write the high byte of the D/O

Note: Refer to Sec. 3.1 for more information regarding wBase.

3.3.1 RESET\ Control Register

(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 regarding wBase.

When the PC is first powered on, the RESET\ signal is in the Low-state. **This will disable all DI/DO operations.** The user has to set the RESET\ signal to the High-state before any DI/DO commands are sent.

```
outportb(wBase,1);    /* RESET\=High → All DIO operations are now enabled */
outportb(wBase,0);    /* RESET\=Low → All DIO operations are now disabled */
```

3.3.2 AUX Control Register

(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 regarding wBase.

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

Aux? = 0 → This Aux is used as a D/I
Aux? = 1 → This Aux is used as a D/O

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 regarding wBase.

When the Aux? is used as D/O, the output state is controlled by this register. This register is designed for future expansion, so it is not possible to control this register.

3.3.4 INT Mask Control Register

(Write): wBase+5

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

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

EN0 = 0→ Disable INT0 as an interrupt signal (default)
EN0 = 1→ Enable INT0 as an interrupt signal

EN1 = 0→ Disable INT1 as an interrupt signal (default)
EN1 = 1→ Enable INT1 as an interrupt signal

```
outportb(wBase+5,0);        /* Disable all interrupts          */
outportb(wBase+5,1);        /* Enable the interrupt of INT0    */
outportb(wBase+5,2);        /* Enable the interrupt of INT1    */
outportb(wBase+5,3);        /* Enable both interrupt channels  */
```

Refer to the following demo programs for more information:
DEMO3.C and DEMO4.C → single interrupt source
DEMO5.C and DEMO6.C → multiple interrupt source

3.3.5 Read Card ID Register

(Read): wBase+0xD4

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	ID3	ID2	ID1	ID0

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

```
wCardID = inportb(wBase+0xD4);        /* Read Card ID(0x0~0x15) */
```

Note: The Card ID function is supported by the following models:
1. **PIO-DA4U/DA8U/DA16U (Ver. 1.1 or above)**
2. **PISO-DA4U/DA8U/DA16U**

3.3.6 Aux Status Register

(Read): 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 regarding wBase.

Aux0=INT0, Aux1=INT1, Aux2~3= EEPROM control, Aux4~7=Aux-ID. Refer to Sec. 4.1 for more information. Aux0~1 are used as interrupt sources. The interrupt service routine needs to read this register to identify the interrupt sources. Refer to Sec. 2.3 for more information.

3.3.7 Interrupt Polarity Control Register

(Write): wBase+0x2A

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

Note: Refer to Sec. 3.1 for more information regarding wBase.

INV0/1=0→ select the inverted signal from INT0/1
INV0/1=1→ select the non-inverted signal from INT0/1

```
outportb(wBase+0x2a,0);    /*Select the inverted input from both channels */
outportb(wBase+0x2a,3);  /*Select the non-inverted input from both channels */

outportb(wBase+0x2a,2);  /*Select the inverted input from INT0 */
                         /*Select the non-inverted input from the others */
```

Refer to Sec. 2.3 and the DEMO3/4/5/6.C files for more information.

3.3.8 Digital Input

(Read): wBase+0xe0 → Low byte of the D/I port

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0

(Read): wBase+0xe4 → High byte of the D/I port

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8

Note: Refer to Sec. 3.1 for more information regarding wBase.

```
wDiLoByte = inportb(wBase+0xe0);          /* Read the D/I state (DI7~DI0) */
wDiHiByte = inportb(wBase+0xe4);          /* Read the D/I state (DI15~DI8) */
wDiValue = (wDiHiByte<<8)|wDiLoByte;
```

Refer to the DEMO2.C file for more information.

3.3.9 Digital Output

(Write): wBase+0xf8 → Low byte of the D/O port

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

(Write): wBase+0xfc → High byte of the D/O port

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8

Note: Refer to Sec. 3.1 for more information regarding wBase.

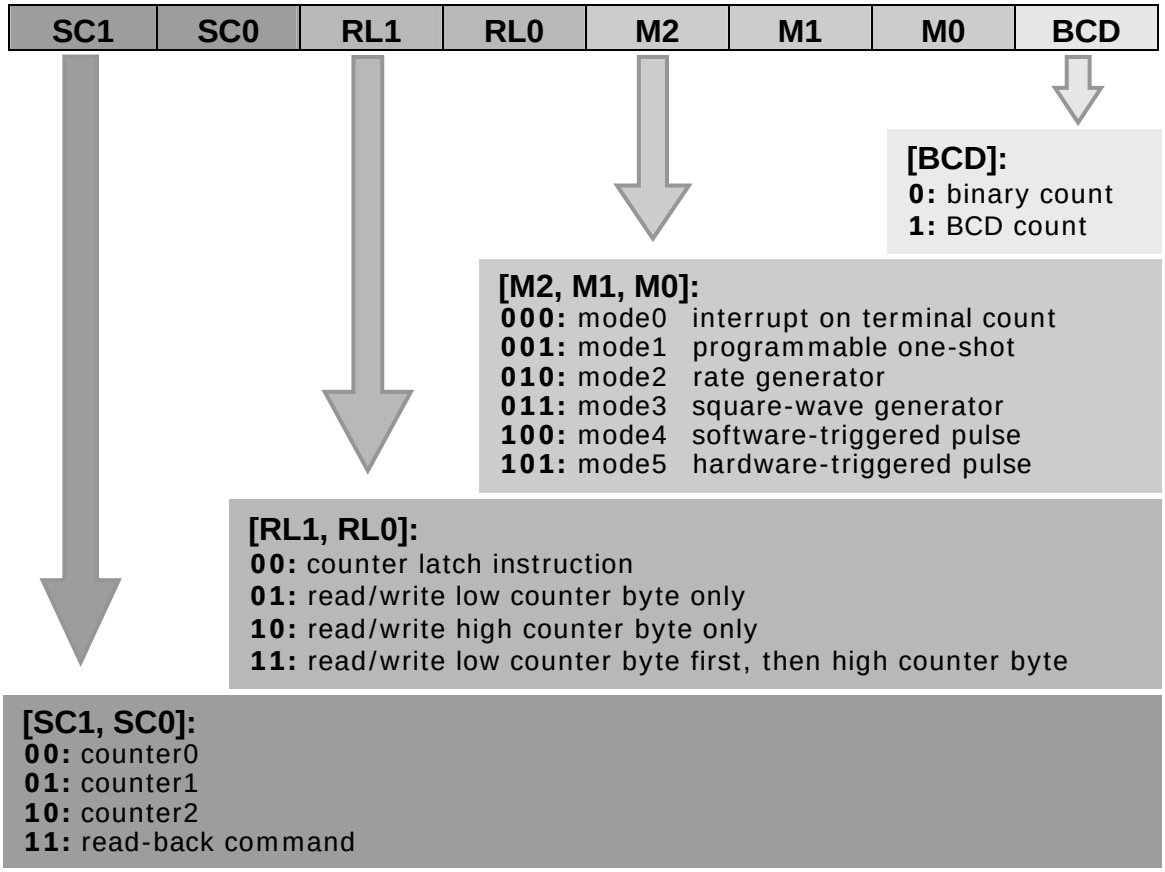
```
outportb(wBase+0xf8,wDoValue);        /*Controls the DO state (DO7~DO0) */
outportb(wBase+0xfc,wDoValue>>8); /*Controls the DO state (DO15~DO8) */
```

Refer to the DEMO1/2.C files for more information.

3.3.10 Read/Write 8254

(Read/Write): wBase+0xc0=8254-counter-0
(Read/Write): wBase+0xc4=8254-counter-1
(Read/Write): wBase+0xc8=8254-counter-2
(Read/Write): wBase+0xcc=8254 control word

8254 Control Word




```
WORD pio_da16_c0(char cConfig, char cLow, char cHigh) /*COUNTER_0*/
{
outportb(wBase+0xcc,cConfig);
outportb(wBase+0xc0,cLow);
outportb(wBase+0xc0,cHigh);
return(NoError);
}
WORD pio_da16_c1(char cConfig, char cLow, char cHigh) /*COUNTER_1*/
{
outportb(wBase+0xcc,cConfig);
outportb(wBase+0xc4,cLow);
outportb(wBase+0xc4,cHigh);
return(NoError);
}
WORD pio_da16_c2(char cConfig, char cLow, char cHigh) /*COUNTER_2*/
{
outportb(wBase+0xcc,cConfig);
outportb(wBase+0xc8,cLow);
outportb(wBase+0xc8,cHigh);
return(NoError);
}
```

3.3.11 D/A Select

There are 1/2/4 D/A converters in PIO-DA4/8/16 cards. It is necessary to select which D/A converter is desired after the D/A data has be sent. D/A channels are allocated as follows:

Write	A1	A0	Description
WBase+0xe0 DA_0	0	0	D/A output channel 0
	0	1	D/A output channel 1
	1	0	D/A output channel 2
	1	1	D/A output channel 3
Wbase+0xe4 DA_1	0	0	D/A output channel 4
	0	1	D/A output channel 5
	1	0	D/A output channel 6
	1	1	D/A output channel 7
Wbase+0xe8 DA_2	0	0	D/A output channel 8
	0	1	D/A output channel 9
	1	0	D/A output channel10
	1	1	D/A output channel11
Wbase+0xec DA_3	0	0	D/A output channel12
	0	1	D/A output channel13
	1	0	D/A output channel14
	1	1	D/A output channel15

Note: Refer to Sec.3.3.11 for more information regarding A1 and A0

```
outportb(wBase+0xf0,wDaValue);  /* output the low byte for D/A data      */
outportb(wBase+0xf4,(wDaValue>>8)|0x02);
                                /*output the high byte for D/A data and  */
                                /*select channel 2 on this converter     */
outportb(wBase+0xe0,0);         /*select DA_0                            */
                                /* after this procedure, the wDaValue will */
                                /* be sent to channel_2                   */
```

Refer to the DEMO6.C, DEMO7.C, DEMO8.C and DEMO9.C files for more information.

3.3.12 D/A Data Output

(write):wBase+0xf0

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
D7	D6	D5	D4	D3	D2	D1	D0

(write):wBase+0xf4

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
A1	A0	D13	D12	D11	D10	D9	D8

Note: Refer to Sec.3.3.10 For more information regarding A1 and A2

Each D/A converter have four analog output channels. When writing data to the D/A converter, the relevant channel to be used is indicated by A1 and A0.

D/A programming sequence:
1. Send data to the D/A converter. (This data will be buffered)
2. Select the D/A converter. (Start the conversion)

```
outportb(wBase+0xf0,wDaValue);  /* output low byte of D/A data       */
outportb(wBase+0xf4,(wDaValue>>8)|0x02);
                                 /* output high byte of D/A data and */
                                 /* select channel 2 on this converter */

outportb(wBase+0xe0,0);          /* select DA_0                      */
                                /* after this procedure wDaValue will */
                                /* be sent to channel_2             */

pio_da16_da(2,wDaValue);        /* send wDaValue to channel_2       */

void pio_da16_da(char cChannel_no,int iVal)
{
    iVal=iVal+(cChannel_no%4)*0x4000;  /* cChannel_no: 0 - 15   */
    outportb(wBase+0xf0,iVal);         /* iVal: 0x0000 - 0x3fff  */
    outportb(wBase+0xf4,(iVal>>8));
    outportb(wBase+0xe0+4*(cChannel_no/4),0xff);
}
```

Refer to the DEMO6.C, DEMO7.C, DEMO8.C and DEMO9.C files for more information.

4. Sotfware Installation

The PIO-DA and PISO-DA series cards can be used in DOS and Windows 98/ME/NT/2K and 32-bit/64-bit Windows XP/2003/Vista/7. The recommended installation procedure for windows is given in Sec. 4.1 ~ 4.2. Or refer to Quick Start Guide (CD:\NAPDOS\PCI\PIO-DA\Manual\QuickStart\).
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pio-da/manual/quickstart/

4.1 Software Installing Procedure

- UniDAQ SDK driver (32-bit/64-bit Windows XP/2003/Vista/7):

Step 1: Insert the companion CD into the CD-ROM drive and after a few seconds the installation program should start automatically. If it doesn't start automatically for some reason, double-click the **AUTO32.EXE** file in the **NAPDOS** folder on this CD.

Step 2: Click the item: "**PCI Bus DAQ Card**".

Step 3: Click the item: "**UniDAQ**".

Step 4: Click the item: "**DLL for Windows 2000 and XP/2003/Vista 32-bit**".

Step 5: Double-Click "**UniDAQ_Win_Setup_x.x.x.x_xxxx.exe**" file in the **Driver** folder.

- Windows driver (Windows 98/NT/2K and 32-bit Windows XP/2003/Vista/7):

Step 1: Insert the companion CD into the CD-ROM drive and after a few seconds the installation program should start automatically. If it doesn't start automatically for some reason, double-click the **AUTO32.EXE** file in the **NAPDOS** folder on this CD.

Step 2: Click the item: "**PCI Bus DAQ Card**".

Step 3: Click the item: "**PIO-DA**".

Step 4: Click the item "**DLL and OCX for Windows 98/NT/2K/XP/2003**".

Step 5: Double-Click "**PIO_DA_Setup_vxxx.exe**" file in the **Driver** folder.

The setup program will then start the driver installation and copy the relevant files to the specified directory and register the driver on your computer. The directory where the drive is stoned is different for different windows versions, as shown below.

- **Windows 64-bit Windows XP/2003/Vista/7:**

The UniDAQ.DLL file will be copied into the C:\WINNT\SYSTEM32 folder
The NAPWNT.SYS and UniDAQ.SYS files will be copied into the C:\WINNT\SYSTEM32\DRIVERS folder

For more detailed UniDAQ.DLL function information, please refer to UniDAQ SDK user manual (CD:\NAPDOS\PCI\UniDAQ\Manual\).
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/maunal/

- **Windows NT/2K and 32-bit Windows XP/2003/Vista/7:**

The PIODA.DLL file will be copied into the C:\WINNT\SYSTEM32 folder
The NAPWNT.SYS and PIODA.SYS files will be copied into the C:\WINNT\SYSTEM32\DRIVERS folder

- **Windows 95/98/ME:**

The PIODA.DLL and PIODA.Vxd files will be copied into the C:\Windows\SYSTEM folder

For more detailed PIODIO.DLL function information, please refer to "PIO-DA DLL Software Manual.pdf" (CD:\NAPDOS\PCI\PIO-DA\Manual\).
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pio-da/manual/

4.2 PnP Driver Installation

Power off the computer and install the PIO-DA and PISO-DA series cards. Turn on the computer and Windows 98/Me/2K and 32-bit/64-bit Windows XP/2003/Vista/7 should automatically defect the new PCI device(s) and then ask for the location of the driver files for the hardware. If a problem is encountered during installation, refer to the PnPinstall.pdf file for more information.

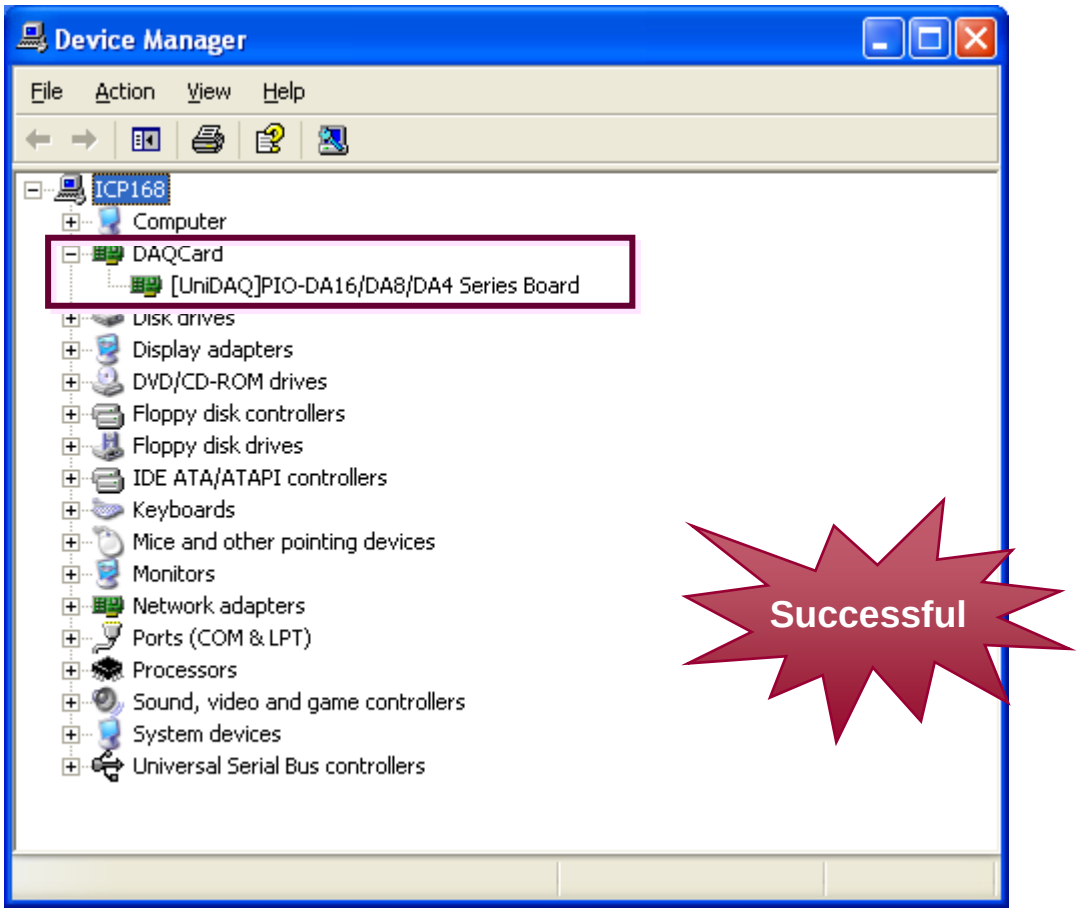
4.3 Confirm the Successful Installation

Make sure the PIO-DA and PISO-DA series card installed are correct on the computer as follows:

Step 1: Select **"Start"** → **"Control Panel"** and then double click the **"System"** icon on Windows.

Step 2: Click the **"Hardware"** tab and then click the **"Device Manager"** button.

Step 3: Check the PIO-DA or PISO-DA series card which listed correctly or not, as illustrated below.



5. Demo Programs

5.1 Demo Programs for Windows

Please note that none of the demo programs will work normally if the DLL driver has not been installed correctly. During the DLL driver installation process, the install shield will register the correct kernel driver to the operating system and copy the DLL driver and demo programs to the correct location depending on the driver software package you have selected (Win98/Me/NT/2K and 32-bit Win XP/2003/Visa/7). After installing the driver, the related demo programs, development library and declaration header files for the different development environments will be available in the following folders.

The demo program is contained in:

CD:\NAPDOS\PCI\PIO-DA\DLL_OCX\Demo\
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pio-da/dll_ocx/demo/

- BCB 6 → For Borland C++ Builder 6
  PIODA.H → Header files
  PIODA.LIB → Linkage library for BCB
- Delphi6 → For Delphi 6
  PIODA.PAS → Declaration files
- VB6 → For Visual Basic 6
  PIODA.BAS → Declaration files
- VC6 → For Visual C++ 6
  PIODA.H → Header files
  PIODA.LIB → Linkage library for VC6
- VB.NET2005 → For VB.NET2005
  PIODA.vb → Declaration files
- CSharp2005 → For C#.NET2005
  PIODA.cs → Declaration files

A list of available demo programs is as follows:

- DA: D/A Output demo
- DIO: D/I/O demo
- DIO2: D/I/O LED interface
- Interrupt: Single interrupt

5.2 Demo Programs for DOS

The related DOS software and demos are located on the CD as below:
CD:\NAPDOS\PCI\PIO-DA\dos\
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pio-da/dos/

After installing the software, the following drivers will be installed onto your hard disk:

- \TC*.* → for Turbo C 2.xx or above
  \TC\LIB*.* → for TC library
  \TC\DEMO*.* → for TC demo programs

  \TC\LIB\Large*.* → TC large model library
  \TC\LIB\Huge*.* → TC huge model library
  \TC\LIB\Large\PIO.H → TC declaration file
  \TC\\LIB\Large\TCPIO_L.LIB → TC large model library file
  \TC\LIB\Huge\PIO.H → TC declaration file
  \TC\\LIB\Huge\TCPIO_H.LIB → TC huge model library file

- \MSC*.* → for MSC 5.xx or above
  \MSC\LIB\Large\PIO.H → MSC declaration file
  \MSC\LIB\Large\MSCPIO_L.LIB → MSC large model library file
  \MSC\LIB\Huge\PIO.H → MSC declaration file
  \MSC\\LIB\Huge\MSCPIO_H.LIB → MSC huge model library file

- \BC*.* → for BC 3.xx or above
  \BC\LIB\Large\PIO.H → BC declaration file
  \BC\LIB\Large\BCPIO_L.LIB → BC large model library file
  \BC\LIB\Huge\PIO.H → BC declaration file

Note: The library is valid for all PIO/PISO series cards.

A list of available demo programs is as follows:

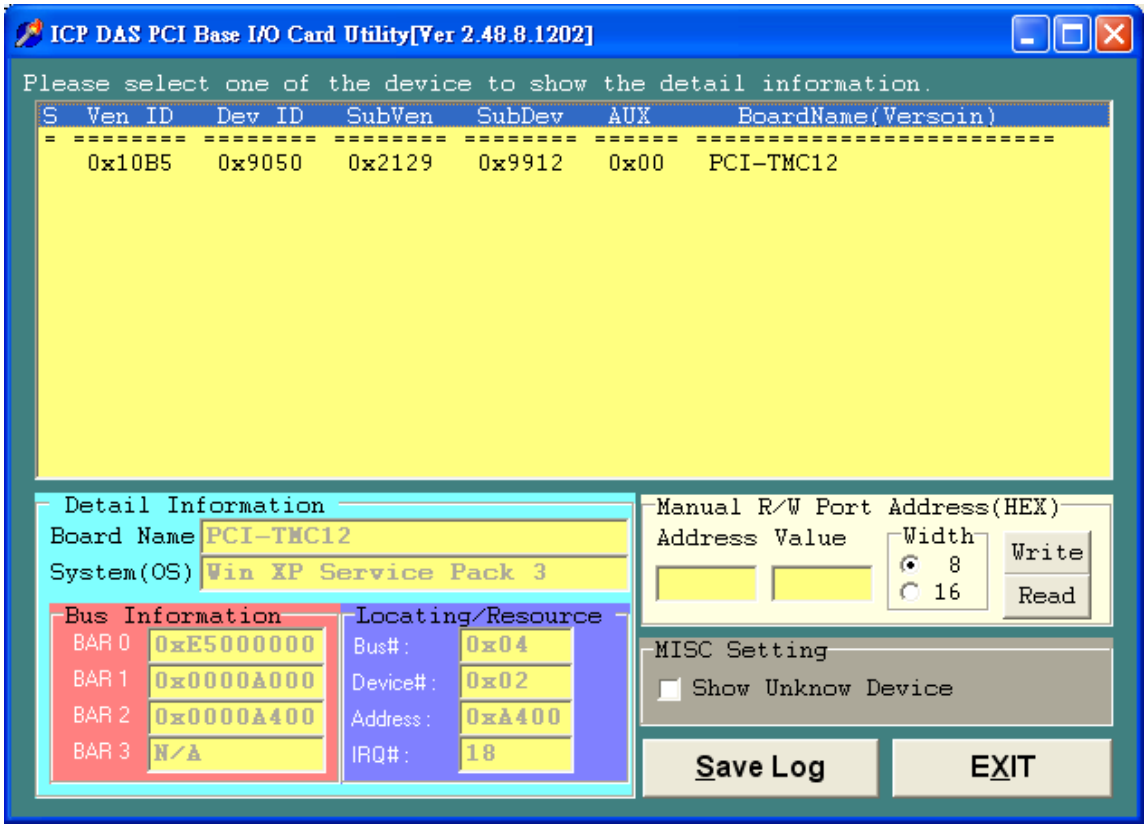
- DEMO1.EXE: D/O demo program
- DEMO2.EXE: D/I/O demo program
- DEMO3.EXE: Single interrupt source (initial high)
- DEMO4.EXE: Single interrupt source (initial low)
- DEMO5.EXE: Two interrupt source
- DEMO6.EXE: Waveform generator without calibration
- DEMO7.EXE: Waveform generator with calibration
- DEMO8.EXE: D/A hex value output without calibration
- DEMO9.EXE: D/A hex value output with calibration
- DEMO10.EXE: Save EEPROM data to file
- DEMO11.EXE: Download EEPROM data from file
- DEMO12.EXE: User software calibration
- DEMO13.EXE: Factory calibration

Note: The calibration demos programs can only be used in a DOS system.

5.3 PIO_PISO.EXE for Windows

The PIO_PISO.exe utility is located on the CD as below and is useful for all PIO/PISO series cards.
CD:\NAPDOS\PCI\Utility\Win32\PIO_PISO\
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/utility/win32/pio_piso/

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



Note: The PIO_PISO.EXE application is valid for all PIO/PISO cards. The user can execute the PIO_PISO.EXE file to retrieve the following information:

- List all PIO/PISO cards installed in the PC
- List the resources allocated to each PIO/PISO card
- List the wSlotBus and wSlotDevice details for identification of specific PIO/PISO cards. (Refer to Sec. 3.2 for more information)

5.4 DEMO1

```
/* ---------------------------------------------------------------------- */
/* DEMO1: D/O demo for PIO-DA16/8/4                                       */
/* Step 1: Run DEMO1.EXE under DOS                                        */
/* Step 2: Check the LEDs of DB-24C will turn on sequentially             */
/* ---------------------------------------------------------------------- */

#include "PIO.H"
void pio_da16_do(WORD wDo);
WORD wBase,wIrq;

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

/* Step 1: find address-mapping of PIO/PISO cards  */
wRetVal=PIO_DriverInit(&wBoards,0x80,0x04,0x00); /*for PIO-DA16/8/4*/
printf("\n(1) There are %d PIO-DA16/8/4 Cards in this PC",wBoards);
if ( wBoards==0 )  exit(0);

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

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

   printf(" --> ");
   ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
   }

PIO_GetConfigAddressSpace(0,&wBase,&wIrq,&t1,&t2,&t3,&t4,&t5);
                                    /* select card_0 */
/* Step 2: enable all D/I/O port                      */
outportb(wBase,1);                        /* /RESET -> 1  */
printf("\n\n(2) DEMO1 D/O test");
j=1;
for(;;)
   {
   gotoxy(1,8);
   pio_da16_do(j);
   printf("\nDO ==> %4x",j);
   delay(10000);
   if (kbhit()!=0) break;
   j=j<<1; j=j&0x0ffff;if (j==0) j=1;
   }
PIO_DriverClose();
}
/* ---------------------------------------------------------------------------- */
void pio_da16_do(WORD wDo)
{
outportb(wBase+0xf8,wDo);           /* 0xf8 : low  byte of DO port */
outportb(wBase+0xfc,(wDo>>8));     /* 0xfc : high byte of DO port */
}
```

5.5 DEMO2

```
/* ---------------------------------------------------------------------------- */
/* DEMO2: D/I/O demo for PIO-DA16/8/4                                           */
/* Step 1: Connect CON1 & CON2 with a 20-pin 1 to 1 flot cable                  */
/* Step 2: Run DEMO2.EXE under DOS                                              */
/* ---------------------------------------------------------------------------- */
#include "PIO.H"

void pio_da16_di(WORD *wDi);
void pio_da16_do(WORD wDo);
WORD wBase,wIrq;

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

clrscr();

/* Step 1: find address-mapping of PIO/PISO cards      */
        .
        .
/* Step 2: enable all D/I/O port                       */
outportb(wBase,1);                          /* /RESET -> 1  */

printf("\n\n(2) DEMO2  D/I/O test");
j=1;
for(;;)
  {
  pio_da16_do(j);
  pio_da16_di(&k);
  gotoxy(1,9);
  printf("DO = %4x , DI = %4x",j,k);
  if (k!=j) printf(" <-- Test Error      ");
  else      printf(" <-- Test Ok         ");
  j++; j=j&0x0ffff;if (j==0) j=1;
  if (kbhit()!=0) break;
  }
PIO_DriverClose();
}
/* ---------------------------------------------------------------------------- */
void pio_da16_di(WORD *wDi)
{
int  in_l,in_h;
in_l=inportb(wBase+0xe0)&0x0ff;
in_h=inportb(wBase+0xe4)&0x0ff;
(*wDi)=(in_h<<8)+in_l;
}
/* ---------------------------------------------------------------------------- */
void pio_da16_do(WORD wDo)
{
outportb(wBase+0xf8,wDo);           /* 0xf8 : low  byte of DO port */
outportb(wBase+0xfc,(wDo>>8));      /* 0xfc : high byte of DO port */
}
```

5.6 DEMO3

```
/* ---------------------------------------------------------------------------- */
/* DEMO3: INT_CHAN_1, timer interrupt demo (initial high)                       */
/*        (It is designed to be a machine independent timer)                    */
/* Step1: Run DEMO3.EXE under DOS                                               */
/* ---------------------------------------------------------------------------- */

#include "PIO.H"
#define A1_8259 0x20
#define A2_8259 0xA0
static void interrupt irq_service();
void pio_da16_c0(char cConfig, char cLow, char cHigh);
void pio_da16_c1(char cConfig, char cLow, char cHigh);
void pio_da16_c2(char cConfig, char cLow, char cHigh);
void init_int1_high();
WORD wBase,wIrq;
int  COUNT_L,COUNT_H,irqmask,now_int_state;

int main()
{
int   i,j;
WORD  wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD  wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
clrscr();
/* Step 1: find address-mapping of PIO/PISO cards              */
        .
        .
/* Step 2: enable all D/I/O port                               */
outportb(wBase,1);                      /* /RESET -> 1  */
printf("\n\n(2) DEMO3 Interrupt (1Hz) test");
init_int1_high();       /* interrupt initialize, INT1 is high now */
COUNT_L=0;COUNT_H=0;
printf("\n\n*** Show the count of Low_pulse ***\n");
for (;;)
   {
   gotoxy(1,10);
   printf("\nINT count = %d",COUNT_L);
   if (kbhit()!=0) break;
   }
outportb(wBase+5,0);                /* disable all interrupt */
PIO_DriverClose();
}

/* Use INT_CHAN_1 as internal interrupt signal                 */
void init_int1_high()
{
DWORD dwVal;
disable();
outportb(wBase+5,0);                /* disable all interrupt */
if (wIrq<8)
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   setvect(wIrq+8, irq_service);
   }
else
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);              /* IRQ2 */
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
   setvect(wIrq-8+0x70, irq_service);
   }
```

```
/* CLK source = 4 MHz */
pio_da16_c1(0x76,0x90,0x01);      /* COUNTER1, mode3, div   400  */
pio_da16_c2(0xb6,0x10,0x27);      /* COUNTER2, mode3, div  10000 */
                                  /* program Cout2 1Hz           */

/* note : the 8254 need extra 2-clock for initialization       */
for (;;)
   {
   if ((inportb(wBase+7)&2)==2) break;      /* wait Cout2 = high    */
   }

/* note : Cout2 = high, INV1 must select the inverted Cout2          */
/*      --> INT_CHAN_1 = !Cout2 = init_low, active_high              */
outportb(wBase+0x2a,0);           /* INV1 = 0, inverted Cout2       */

now_int_state=1;                  /* now Cout2 is high              */
outportb(wBase+5,2);              /* EN1 = 1, enable INT_CHAN_1     */
                                  /* as interrupt source            */
enable();
}

/* ---------------------------------------------------------------------*/
void interrupt irq_service()
{
if (now_int_state==1)             /* now INT1(Cout2) changed to low    */
  {                               /* --> INT_CHAN_1=!INT1=high now     */
  COUNT_L++;                      /* find a low pulse (INT1)           */
  if((inportb(wBase+7)&2)==0)          /* INT1 is still fixed in low ->  */
    {                                  /* need to generate a high pulse  */
    outportb(wBase+0x2a,2);     /* INV1 select non-inverted input  */
                                /* INT_CHAN_1=INT1=low -->         */
                                /* INT_CHAN_1 generate high pulse  */
    now_int_state=0;                /* now INT1=low                */
    }
  else now_int_state=1;             /* now INT1=high               */
                                    /* don't have to gen. high pulse  */
  }
else                              /* now INT1(Cout2) changed to high */
  {                             /* --> INT_CHAN_1=INT1=high now    */
  COUNT_H++;                        /* find a low pulse (INT1)       */
  if((inportb(wBase+7)&2)==2)     /* INT1 is still fixed in high ->  */
    {                             /* need to generate a high pulse   */
    outportb(wBase+0x2a,0);       /* INV1 select inverted input      */
                                  /* INT_CHAN_1=!INT1=low -->        */
                                /* INT_CHAN_1 generate high_pulse */
    now_int_state=1;                /* now INT1=high               */
    }
  else now_int_state=0;             /* now INT1=low                */
                                    /* don't have to gen. high pulse   */
  }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}

/* ------------------------------------------------------------ */
void pio_da16_c0(char cConfig, char cLow, char cHigh) /* COUNTER0 */
{
outportb(wBase+0xcc,cConfig);
outportb(wBase+0xc0,cLow);
outportb(wBase+0xc0,cHigh);
}
```

5.7 DEMO5

```
/* ---------------------------------------------------------------------- */
/* DEMO5 : INT_CHAN_0 & INT_CHAN_1 timer interrupt demo                   */
/*          (It is designed to be a machine independent timer)            */
/* Step 1: Run DEMO5.EXE under DOS                                        */
/* ---------------------------------------------------------------------- */

#include "PIO.H"
#define A1_8259 0x20
#define A2_8259 0xA0
static void interrupt irq_service();
WORD wBase,wIrq;
int  irqmask,now_int_state,new_int_state,int_c;
int  INT0_L,INT0_H,INT1_L,INT1_H;
int  b0,b1,invert;

int main()
{
int   i,j;
WORD  wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD  wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
clrscr();
/* Step 1: find address-mapping of PIO/PISO cards                         */
        .
        .
/* Step 2: enable all D/I/O port                                          */
outportb(wBase,1);                                  /* /RESET -> 1  */
printf("\n\n(2) DEMO5 Interrupt test");
init_high();          /* interrupt initialize, INT_CHAN_0/1 is high now */
printf("\n\n*** Show the count of Low_pulse ***\n");
INT0_L=INT0_H=INT1_L=INT1_H=0;
for (;;)
   {
   gotoxy(1,10);
   printf("\nINT0[%x,%x],INT1[%x,%x]",INT0_H,INT0_L,INT1_H,INT1_L);
   if (kbhit()!=0) break;
   }
outportb(wBase+5,0);                          /* disable all interrupt */
PIO_DriverClose();
}
/* Use INT_CHAN_0 & INT_CHAN_1 as internal interrupt signal               */
void init_high()
{
DWORD dwVal;
disable();
outportb(wBase+5,0);                    /* disable all interrupt     */
if (wIrq<8)
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   setvect(wIrq+8, irq_service);
   }
else
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);                  /* IRQ2 */
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
   setvect(wIrq-8+0x70, irq_service);
   }
```

```
/* CLK source = 4 MHz */
pio_da16_c0(0x36,0x20,0x4e);      /* COUNTER0, mode3, div 20000  */
                                  /* program Cout0 200Hz         */
pio_da16_c1(0x76,0x90,0x01);      /* COUNTER1, mode3, div  400   */
pio_da16_c2(0xb6,0x64,0x00);      /* COUNTER2, mode3, div  100   */
                                  /* program Cout2 100Hz         */

/* note : the 8254 need extra 2-clock for initialization        */
for (;;)
   {
   if ((inportb(wBase+7)&3)==3) break; /* wait Cout0&Cout2 = high */
   }

/* note : Cout0/2 = high, INV0/1 must select the inverted Cout0/2       */
/*      --> INT_CHAN_0 = !Cout0 = init_low, active_high                 */
/*      --> INT_CHAN_1 = !Cout2 = init_low, active_high                 */
outportb(wBase+0x2a,0);           /* INV0=0, INV1=0 inverted      */
now_int_state=3;                  /* now Cout0 & Cout2 is high    */
outportb(wBase+5,3);             /* enable INT_CHAN_0/1 interrupt */
enable();
}

/* ------------------------------------------------------------------------- */
/* Note : 1.The hold_time of INT_CHAN_0 & INT_CHAN_1 must long enoug */
/*        2.The ISR must read the interrupt status again to identify the     */
/*          active interrupt source.                                         */
/*        3.The INT_CHAN_0 & INT_CHAN_1 can be active at the same time*/
/* ------------------------------------------------------------------------- */

void interrupt irq_service()
{

/* now ISR can not know which interrupt is active                       */
new_int_state=inportb(wBase+7)&0x03;          /* read all interrupt     */
                                               /* signal state          */
int_c=new_int_state^now_int_state;  /* compare new_state to old_state  */

if ((int_c&0x01)==1)                 /* INT_CHAN_0 is active          */
   {
   if ((new_int_state&1)==0)         /* INT0 change to low now        */
      {
      INT0_L++;
      }
   else                              /* INT0 change to high now       */
      {
      INT0_H++;
      }
   invert=invert^1;                 /* generate high_pulse           */
   }
if ((int_c&0x02)==2)                /* INT_CHAN_1 is active          */
   {
   if ((new_int_state&2)==0)       /* INT1 change to low now         */
      {
      INT1_L++;
      }
   else                             /* INT1 change to high now       */
      {
      INT1_H++;
      }
   invert=invert^2;                  /* generate high_pulse          */
   }
now_int_state=new_int_state;        /* update interrupt status      */
outportb(wBase+0x2a,invert);         /* generate a high pulse       */
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

5.8 DEMO8

```
/* ---------------------------------------------------------------- */
/* DEMO8: D/A Output without calibration                            */
/* Step1: Run DEMO8.EXE under DOS                                   */
/* ---------------------------------------------------------------- */

#include "PIO.H"

void pio_da16_da(int cChannel_no,int iVal);

WORD  wBase,wIrq;

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

clrscr();

/* Step 1: find address-mapping of PIO/PISO cards                   */
        .
        .
/* Step 2: enable all D/I/O port                                    */
outportb(wBase,0x11);                       /* /RESET -> 1   */

printf("\n\n(2) A/D Output without calibration test");

printf("\n\n    (a)  1.23V Voltage output to each channel");
for (i=0; i<16; i++)
   {
   j=1.23*16383/20.0+8192;
   pio_da16_da(i,j);
   }
getch();
printf("\n\n    (b) 1.23mA Current output to each channel");
for (i=0; i<16; i++)
   {
   j=1.23*8192/20+8191;
   pio_da16_da(i,j);
   }
getch();

outportb(wBase+5,0);                /* disable all interrupt */
outportb(wBase+3,0);                /* all D/O are Low       */
outportb(wBase+2,0);                /* all AUX as D/I        */
PIO_DriverClose();
}

/* ------------------------------------------------------------ */
void pio_da16_da(int iChannel_no,int iVal)
{
iVal=iVal+(iChannel_no%4)*0x4000;      /* iChannel_no : 0 - 15   */
outportb(wBase+0xf0,iVal);             /* iVal : 0x0000 - 0x3fff */
outportb(wBase+0xf4,(iVal>>8));
outportb(wBase+0xe0+4*(iChannel_no/4),0xff);
}
```

5.9 DEMO9

```
/* ------------------------------------------------------------- */
/* DEMO9 : D/A Output with calibration                           */
/* step1 : Run DEMO9.EXE under DOS                               */
/* ------------------------------------------------------------- */

#include "PIO.H"
void pio_da16_da(int cChannel_no,int iVal);
WORD  wBase,wIrq;
WORD  wN10V[16],wP10V[16],w00mA[16],w20mA[16],EEP;
float fDeltaV[16],fDeltaI[16];

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

clrscr();

/* Step 1: find address-mapping of PIO/PISO cards                */
        .
        .
/* Step 2: enable all D/I/O port                                 */
outportb(wBase,0x11);              /* /RESET -> 1                */
outportb(wBase+2,0x1c);           /* AUX 4/3/2 are D/O, othes D/I */
outportb(wBase+3,0);              /* all D/O are Low             */

printf("\n\n(2) A/D Output with calibration test");

for (i=0; i<64;i++)
   {
   if (i<16)
     {
     EEP_READ(i,&j,&k);
     wN10V[i]=(j<<8)+k;
     }
   if ((i>=16)&&(i<32))
     {
     EEP_READ(i,&j,&k);
     wP10V[i-16]=(j<<8)+k;
     }
   if ((i>=32)&&(i<48))
     {
     EEP_READ(i,&j,&k);
     w00mA[i-32]=(j<<8)+k;
     }
   if (i>=48)
     {
     EEP_READ(i,&j,&k);
     w20mA[i-48]=(j<<8)+k;
     }
   }

for (i=0; i<16; i++)
   {
   fDeltaV[i]=20.0/(wP10V[i]-wN10V[i]);
   fDeltaI[i]=20.0/(w20mA[i]-w00mA[i]);
   }
```

```
printf("\n\n    (a)   1.23V Voltage output to each channel");
for (i=0; i<16; i++)
    {
    j=(1.23+10.0)/fDeltaV[i]+wN10V[i];
    pio_da16_da(i,j);
    }
getch();
printf("\n\n    (b) 1.23mA Current output to each channel");
for (i=0; i<16; i++)
    {
    j=1.23/fDeltaI[i]+w00mA[i];
    pio_da16_da(i,j);
    }
getch();

outportb(wBase+5,0);                    /* disable all interrupt       */
outportb(wBase+3,0);                    /* all D/O are Low             */
outportb(wBase+2,0);                    /* all AUX as D/I              */
PIO_DriverClose();
```