# EX-92016
## 16 channel
## Isolated
## Analog input card



Printed Jan. 2003 Rev 1.1

# Table of Contents

# Chapter 1
# Introduction





## 1.1 Introduction

The EX-92016 is a 12-bit 16-channel analog input card for the PCI bus. It provides 16 analog input channels with a sampling rate up to 100k samples/s, 12-bit resolution and isolation protection of 2500 VDC .

**PCI-bus Plug and Play**

The EX-92016 uses a PCI controller to interface the card with the PCI bus. The controller fully implements the PCI bus specification Rev 2.1. All bus relative configurations, such as base address and interrupt assignment, are automatically controlled by software.

**Flexible Input Types and Range Settings**

The EX-92016 features an automatic channel/gain scanning circuit. The circuit, rather than your software, controls multiplier switching during sampling. The on-board SRAM stores different gain values and configurations for each channel. This design lets you perform multi-channel sampling with different gains for each channel and with free combination of single-ended and differential inputs.

**High-speed Data Acquisition**

The EX-92016 provides a sampling rate up to 100k samples/s. It has an on-board FIFO buffer, which can store up to 4K A/D samples and generates an interrupt signal when the FIFO is half full. This feature provides continuous high-speed data transfer and more predictable performance on Windows systems.

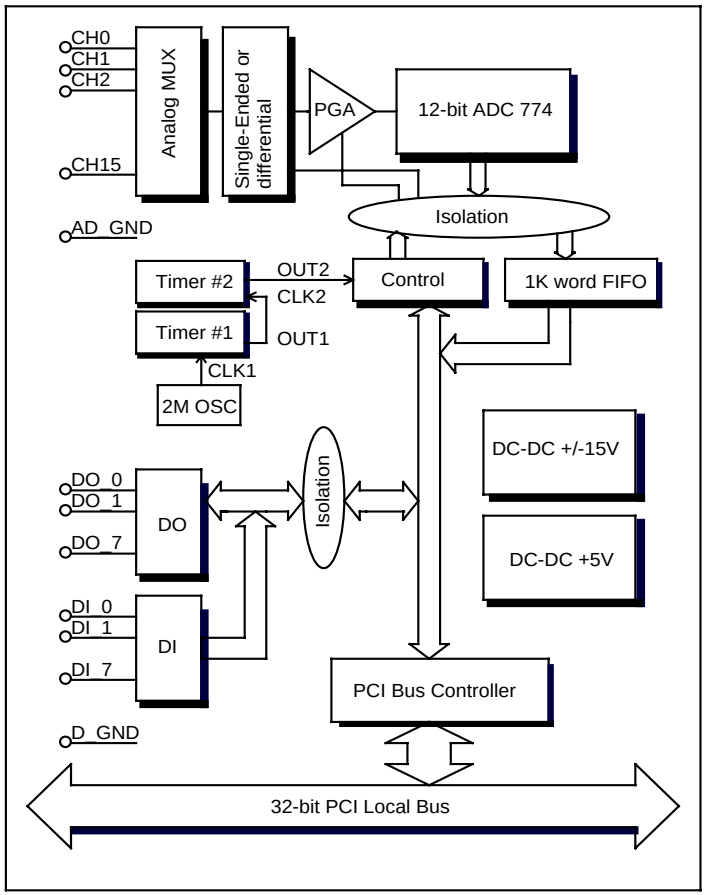
**Supports S/W, Internal and External Pacer Triggering**

The EX-92016 supports three kinds of trigger modes for A/D conversion: software triggering, internal pacer triggering and external pacer triggering. The software trigger allows users to acquire a sample when it is needed; the internal pacer triggers continuous high-speed data acquisition. The EX-92016 also accepts external trigger sources, allowing synchronous sampling with external devices..

**Satisfies the Need for Isolation Protection**

The EX-92016 provides optical isolation protection of 2500 VDC between the inputs and the PC bus to protect the PC and peripherals from damage due to high voltages on the input lines. It is ideal for the situations where budget-conscious users require flexibility, stability and a high level of isolation protection for their data acquisition system

## 1.2 The block diagram of EX-92016




Block diagram of EX-92016





## 1.3 Specifications

**Analog Input:**

- **Channels:** 16 single-ended
- **Resolution:** 12-bit
- **On-board FIFO:** 1K samples
- **Conversion time:** 2.5 μs
- **Input range:**

  Bipolar: ±10 V, ±5 V, ±2.5 V, ±1.25 V

  Unipolar: 0 ~10 V, 0 ~ 5 V, 0 ~ 2.5 V,0~1.25V
- **Maximum Input Overvoltage:** ±30 V
- Common Mode Rejection Ratio (CMRR): 75dB min. at gain=1,2
- **Maximum sampling rate:** 100 kHz
- **Accuracy:** (depending on gain)
- **Gain Accuracy:** 0.01% of FSR±1LSB at gain=1,2
- **Linearity error:** ±1 LSB
- **Drift:** Typical 30 PPM / °C ( 0 ~ 60 °C )
- **Input Impedance:** 10M ohms/10 pf
- **Trigger mode:** Software, on-board programmable pacer or external TTL

**Programmable Timer/Counter**

- **Counter chip:** 82C54 or equivalent
- **Counters:** 3 channels, 16 bits (2 channels are permanently configured as programmable pacers;1 channel is un-used.
- **Time base:**

  Channel #0:un-used

  Channel #1:10 MHz

  Channel #2:Takes input from output of channel 1

**General:**

- **I/O Connector:** 37-pin D-type female connector
- **Power consumption:** +5 V @ 850 mA (Typical), +5 V @ 1.0 A (Max.)
- **Operating temperature:** 0 ~ +60 °C (32 ~ 140 °F)
- **Storage temperature:** -20 ~ +70 °C (-4 ~ 158 °F)
- **Operating humidity:** 5 ~ 95%RH non-condensing
- **Dimension:** 170mm(W) x102mm (H)

## 1.4 Software Supporting

**TOPS CCC** provides versatile software drivers and packages for users' different approach to built-up a system. We not only provide programming library such as DLL for many Windows systems, but also provide drivers for many software package such as LabVIEW™ ,Intouch™ and so on. All the software options are included in the provided CD.

## 1.5 Programming Library

The provided CD includes the function libraries for many different operating systems, including:

- **DOS Library:** Borland**C**/**C**++ and Microsoft C++, the functions descriptions are included in this user's guide.
- **Windows 98/2000/NT/Me/XP DLL:** For VB, VC++, BC5, the functions descriptions are included in this user's guide.
- Windows 98/2000/NT/Me/XP ActiveX: For Windows's applications
- **LabVIEW ® Driver:** Contains the VIs, which are used to interface with NI's LabVIEW ® software package. Supporting Windows 95/98/NT/2000. The LabVIEW ® drivers are free shipped with the board.
- **InTouch Driver:** Contains the InTouch driver which support the Windows 98/2000/NT/XP. The The InTouch ® drivers are free shipped with the board.

# Chapter 2
# Installation





This chapter describes how to install the EX-92016 card. Please follow the follow steps to install the EX-92016 card.

## 2.1 What You Have

In addition to this *User's Manua*l, the package includes the following items:

- EX-92016 board
- Driver/utilities CD
- This user's manual

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

## 2.2 Unpacking

Your EX-92016 card contains sensitive electronic components that can be easily damaged by static electricity. The operator should be wearing an anti-static wristband, grounded at the same point as the anti-static mat. Inspect the card module carton for obvious damage. Shipping and handling may cause damage to your module. Be sure there are no shipping and handing damages on the module before processing.

After opening the card module carton, extract the system module and place it only on a grounded anti-static surface component side up. Again inspect the module for damage. Press down on all the socketed IC's to make sure that they are properly seated. Do this only with the module place on a firm flat surface.

## 2.3 Hardware Installation Outline

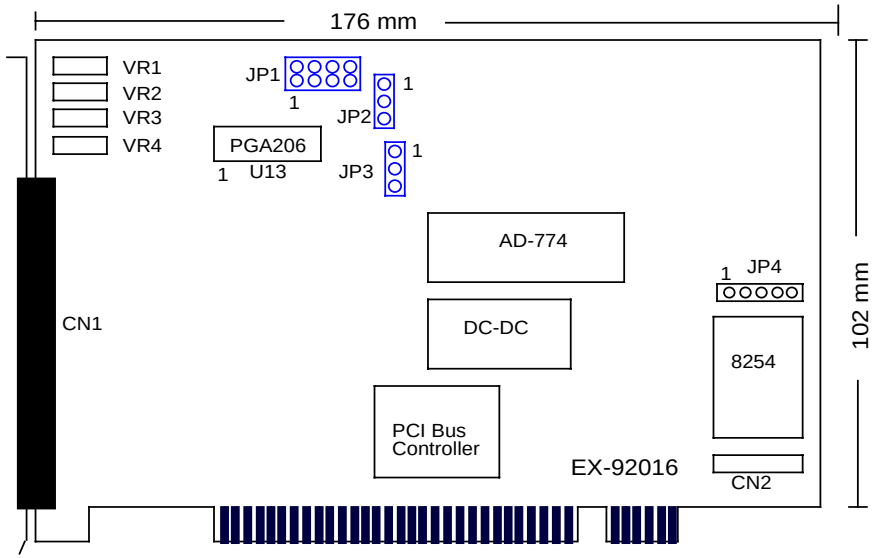
- **PCI configuration**

  The PCI cards are equipped with plug and play PCI controller, it can request base addresses and interrupt according to PCI standard. The system BIOS will install the system resource based on the PCI cards' configuration registers and system parameters (which are set by system BIOS). Interrupt assignment and memory usage (I/O port locations) of the PCI cards can be assigned by system BIOS only. These system resource assignments are done on a board-by-board basis. It is not suggested to assign the system resource by any other methods.

- **PCI slot selection**

  The PCI card can be inserted to any PCI slot without any configuration for system resource.

## 2.4 PCB Layout




Where

- VR1: A/D bi-polar offset adjustment
- VR2: A/D uni-polar offset adjustment
- VR3: A/D full scale adjustment
- VR4: PGA offset adjustment
- U12: PGA operational amplifier
- JP1: Input type selection (Single-Ended or Differential)
- JP2: Un-polar or bipolar section
- JP3: Full scale range selection
- JP4: Timer/counter #0 out/input
- CN1: Analog input and digital I/O





## 2.5 Installation Procedures

1. Turn off your computer.
2. Turn off all accessories (printer, modem, monitor, etc.) connected to your computer.
3. Remove the cover from your computer.
4. Setup jumpers on the card.
5. Before handling the PCI cards, discharge any static buildup on your body by touching the metal case of the computer. Hold the edge and do not touch the components.
6. Position the board into the PCI slot you selected.
7. Secure the card in place at the rear panel of the system.

## 2.6 Device Installation for Windows Systems

If this is the first time to install EX- cards in your Windows system. Once Windows 95/98/2000 has started, the Windows system will find the new EX- cards. You will be informed to input the device information source. Please refer to the ***"Software Installation Guide***" in attached CD, which will guide you the steps of installing the device.

# Chapter 3
# Signal Connections





## 3.1 Digital I/O connector

The I/O connector for the EX-92016 card is a 37-pin D-type connector, which you can connect to 37-pin D-type accessories (see page 76)

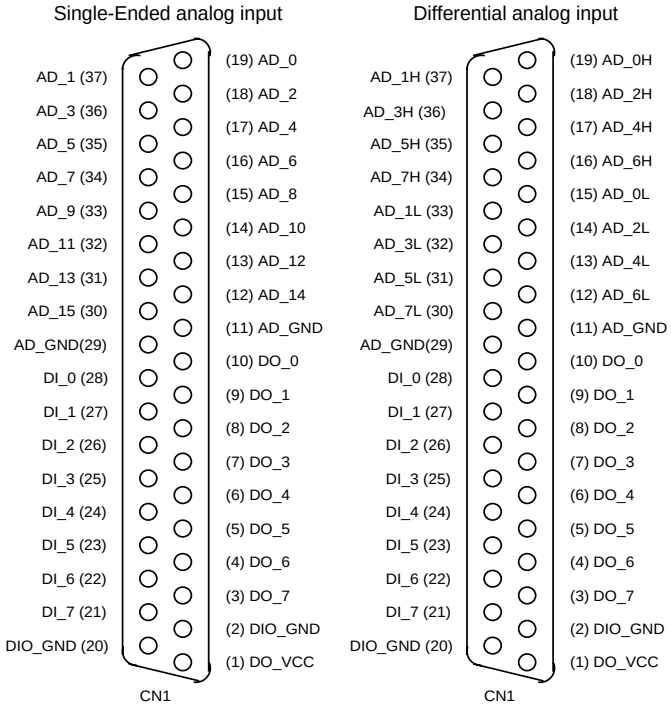



Figure 3-1 Pin assignment of EX-92016 connector CN1

**Legend:**

- **DI_n:** Isolated digital input channel #n
- **DO_n**: Isolated digital output channel #n
- **DO_VCC**: External power supply input for driving digital output transistors, range from +5VDC to +30V
- **DIO_GND**: Ground return path of isolated input and output channels
- **AD_n:** Single-Ended analog input channel #n (n=0~15)
- **AD_nH:** Differential analog high input channel #n (n=0~7)
- **AD_nL:** Differential analog low input channel #n (n=0~7)
- **AD_GND:** Ground return path of analog input channels

## 3.2 Timer/Counter connector

There are total 3 timer/counter channels on the EX-92016 card, channel #1 and channel #2 are both used to generate programmable triggering pulses for A/D converter and only channel #0 is free for user

The timer/counter channel #0 is connected to JP4 as shown in Figure 3-2

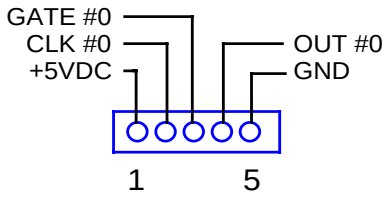



Figure 3-2 Pin assignment of EX-92016 connector JP4

**Legend:**

- **+5VDC:** System +5VDC output
- **CLK #0**: Clock input of timer/counter #0
- **GATE #0**: Gate input of timer/counter #0
- **OUT #0**: Out output of timer/counter #0
- **GND:** Ground return path of timer/counter #0

# Chapter 4
# Jumper settings





## 4.1 Card number setting

Maximum four EX-92016 cards can be installed in system simultaneously with each has a unique card number.

A jumper called "JP5" (see page 15) on the card is used to set the card number starts from 1 to 4

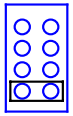
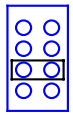
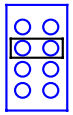
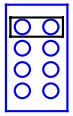
| JP1 | Card number |
|---|---|
|   | 1 (default setting) |
|   | 2 |
|   | 3 |
|   | 4 |

Table 4-1

**Note:** This jumper (JP5) is available only for EX-92016 version 1.3 or later

## 4.2 Analog input type selection

JP1 is the selection jumper of analog signal input type. The following diagram shows the two possible configurations.

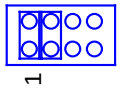
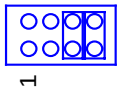
| JP1 | Input type |
|---|---|
|  | Single-Ended input (Default) |
|  | Differential input |

Table 4-2

## 4.3 Input polarity selection

JP2 is the polarity selection jumper. The following diagram shows the two configurations.

| JP2 | Input polarity |
|---|---|
|  | Uni-polar input (Default) |
|  | Bi-polar input |

Table 4-3





## 4.4 Full Range selection

JP3 set the full range of the analog input channels. The following diagram shows the possible configurations.

| JP3 | Input full range |
|---|---|
|  | 10V full range (Default) |
|  | 20V full range (for bi-polar only) |

Table 4-4

## 4.5 Max. Input Range Configurations

The JP2 and JP3 are used to setup the polarity and maximum range of analog input signal range. There are three possible combinations, 0~10V, -5V~+5V and –10V ~+10V.

## 4.6 Analog Input Polarity and Range selection

JP1 is the polarity and maximum input range selection jumper. The following table shows the two configurations.

| JP2 | JP3 | Polarity | Maximum input range | | | |
|---|---|---|---|---|---|---|
| | | | Gain=1 | Gain=2 | Gain=4 | Gain=8 |
|  |  | Unipolar | 0V~+10V | 0 ~+5V | 0~0.25V | 0~1.25V |
|  |  | Bipolar | -5~+5V | -2.5~+2.5V | -1.25~+1.25V | -0.625~+0.625V |
|  |  | Bipolar | -10~+10V | -5~+5V | -2.5~+2.5V | -1.25~+1.25V |

Table 4-5

# Chapter 5
# Registers Format





This information is quite useful for the programmers who wish to handle the card by low-level programming. However, we suggest user have to understand more about the PCI interface then start any low-level programming. In addition, the contents of this chapter can help users understand how to use software driver to manipulate this card.

## 5.1 PCI PnP Registers

There are two types of registers: PCI Configuration Registers (PCR) and Peripheral Interface Bus (PIB). The PCR, which is compliant to the PCI-bus specifications, is initialized and controlled by the plug & play (PnP) PCI BIOS..

The PCI bus controller Tiger 100/320 is provided by Tigerjet Network Inc. (www.tjnet.com). For more detailed information of PIB, please visit Tigerjet technology's web site to download relative information. It is not necessary for users to understand the details of the PIB if you use the software library. The PCI PnP BIOS assigns the base address of the PIB. The assigned address is located at offset 14h of PIB .

EX-92016 board registers are in 32-bit width. But only lowest byte (bit0~bit7) is used. The users can access these registers by only 32-bit I/O or 8-bit I/O instructions. The following sections show the address map, including descriptions and their offset addresses relative to the base address.

## 5.2 PCI controller register address map

- **Reset control register**

  The EX-92016 is in inactive state when the system power on, and should be activated by set bit o of this register to "1" state

  **Address:** Base + 0x00h

  **Attribute:** Write only

  **Value:** 01

- **PCI Internal special control register**

  EX-92016 internal control register, should be written with value 00H before controlling EX-92016 card

  **Address:** Base + 002h

  **Attribute:** Write only

  **Value:** always are 00H

- **Interrupt mask control register**

  Enable or disable PCI interrupt INT #A

  **Address:** Base + 0x05h

  **Attribute:** Write only

  **Value:** 10H =enable PCI INT A#

  00H=disable PCI INT #A

## 5.3 Digital Input Register Address Map

There are 8 isolated digital input channels on EX-92016, each bit of based address is corresponding to a signal on the digital input channel.

**Address:** BASE + 0D8H

**Attribute:** read only

| Address | Port | Bit number | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Base+0D8H | 0 | DI_7 | DI_6 | DI_5 | DI_5 | DI_3 | DI_2 | DI_1 | DI_0 |

## 5.4 Digital Output Register

There are total 8 digital output channels on the EX-92016, each bit of based address is corresponding to a signal on the digital output channel.

**Address:** BASE + 0DCH

**Attribute:** write /read

| Address | Port | Bit number | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Base+0DCH | 0 | DI_7 | DI_6 | DI_5 | DI_5 | DI_3 | DI_2 | DI_1 | DI_0 |

## 5.5 Read FIFO data register

The EX-92016 A/D data is stored in the FIFO after conversion. The data can be transferred to host memory by software only. The register is 12 bits and can be read twice by 8 bits I/O command.

**Address:** Base + 0C0H ~ Base+0C4H

**Attribute:** Read only

| Address | Bit number | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Base+0C0H | AD_7 | AD_6 | AD_5 | AD_5 | AD_3 | AD_2 | AD_1 | AD_0 |
| Base+0C4H | Channel no. #n | | | | AD_11 | AD_10 | AD_9 | AD_8 |





## 5.6 Read FIFO status register

The EX-92016 A/D data is stored in the FIFO after conversion. The FIFO status can be read back from this register..

**Address:** Base + 0D0H

**Attribute:** Read only

| Address | Bit number | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Base+0DCH0 | X | X | X | X | BUSY | FF | HF | EF |

**Where:**

FF: FIFO full flag

HF: FIFO half full flag

EF: FIFO empty

BUSY: '0' means AD is busy, the A/D data has not been latched in FIFO yet. If changes from '0' to '1', A/D data is written into FIFO.

## 5.7 A/D control register

This register is used to set the A/D conversion control modes, such as channel number, gain, trigger, and channel scan

**Address:** Base + 0C0H

**Attribute:** Write only

**Value:**

| Address | Bit number | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Base+0C0H | Trigger mode | Scan mode | Gain | | Channel number | | | |

Where:

Channel Number: set A/D input channel number for conversion or auto scan ending channel

Gain: set gain (0=1, 1=2, 2=4, 3=8)

Scan mode: 0= No auto scan, the channel number will not be auto increased to next channel after current conversion completed

1=Auto scan, the channel number will be auto increased to next channel after current onversion completed

Trigger mode: 0= triggered by software, 1= Triggered by pacer

## 5.8 A/D control status register

This register stores the A/D conversion status such as current channel number, gain, trigger mode, and channel scan mode

**Address:** Base + 0CCH

**Attribute:** Read only

**Value:**

| Address | Bit number | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Base+0C0H | Trigger mode | Scan mode | Gain | | Channel number | | | |

**Where:**

Channel Number: Current A/D input channel

Gain: Current gain (0,1,2,3)

Scan mode: 0= No auto scan, 1= Auto scan

Trigger mode: 0= triggered by software, 1= Triggered by pacer

## 5.9 Interrupt Control and Readback Register

The EX-92016 has a triple interrupt sources,can be generated and be checked by the software. This register is used to select the interrupt sources and readback the interrupt status.

**Address:** Base + 0D4H

**Attribute:** Read / write

**Value: as shown in** Table 5-6

| Interrupt mode | Bit number | | | |
|---|---|---|---|---|
| | 7~3 | 2 | 1 | 0 |
| Disable interrupt | X | 0 | 0 | 0 |
| Interrupt when FIFO is half full | X | 0 | 0 | 1 |
| Interrupt when End of A/D conversion | X | 0 | 1 | 0 |
| Interrupt when start of A/D conversion triggered by timer/counter #2 | X | 1 | 0 | 0 |

Table 5-6





## 5.10 Software Trigger Register

To generate a trigger pulse to the EX-92016 for A/D conversion, you just write any data to this register, and then the A/D converter will be triggered.

**Address:** Base + 0C0H

**Attribute:** Write only

**Value:** any value

## 5.11 Hardware Interrupt Clear Register

Because the PCI interrupt signal is level trigger, the interrupt clear register must be written to clear the flag after processing the interrupt request event; otherwise, that another interrupt request is inserted will cause the software to hang on processing the interrupt event.

**Address:** Base + 0D8H

**Attribute:** write only

**Value:** any value

## 5.12 Clear scan register

This register is used to clear channel scan counter and then restart the counter from channel #0

**Address:** Base + 0D0H

**Attribute:** Write only

**Value:** any value

## 5.13 Timer/Counter Register

The 82C54 chip occupies 4 I/O address locations in the EX-92016 as shown below. Users can refer to 82C54 data sheet for the descriptions about all the features of 82C54. You can download the data sheet on the following web site:

**http://support.intel.com/support/controllers/peripheral/231164.htm**

**Address:** Base + 0E0H~ Base + 0ECH~

**Attribute:** Read/write

**Value:**

| Address | Timer/counter register |
|---|---|
| Base+0E0H | Counter 0 Register (R/W) |
| Base+0E4H | Counter 1 Register (R/W) |
| Base+0E8H | Counter 2 Register (R/W) |
| Base+0ECH | 8254 control mode (W) |

## 5.14 High Level Programming

You may bypass the detailed register structures and use provided libraries to control your EX-92016 card directly. The software libraries, DOS library for Borland C++, and DLL for Windows 95 are included in the TOPS CCC's "Manual & Software Utility" CD.

# Chapter 6
# Operation Theorem

## 6.1 Isolated Digital Input Channels

The isolated digital input is open collector transistor structure. The input voltage range form 0V to 24V and input resister is 4.7K ohms. The connection between outside signal and EX-92016 is shown in Figure 6-3 and Figure 6-4

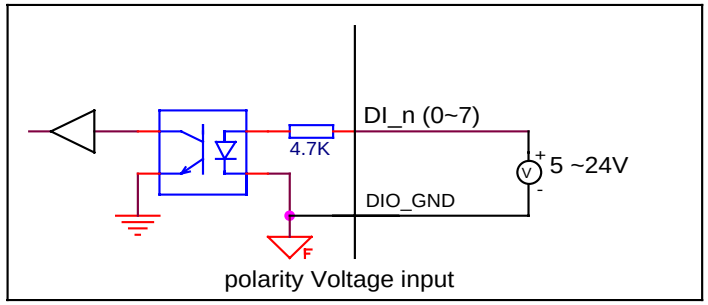



Figure 6-3 isolated digital inputs of EX-92016

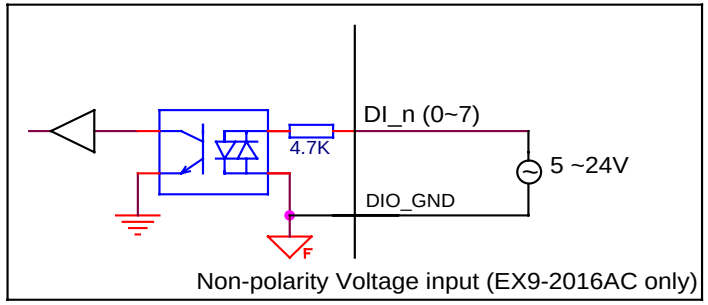



Figure 6-4 non-polarity Isolated digital inputs of EX-92016AC

**Note:** The digital input connections of EX-92016AC are not polarity sensitive whether used on AC or DC voltage.





## 6.2 Isolated Digital Output Channels

On EX-92016, the DO_VCC pin is used to supply the voltage to on-board darlinton transistor and also as "fly-wheel" diode, which can protect the driver if the loading is inductance loading such as relay, motor or solenoid. If the loading is resistance loading such as resistor or LED, the connection to fly-wheel diode is not necessary.

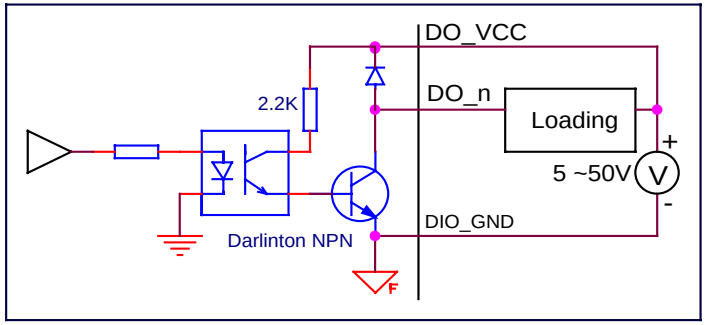



Figure 6-5 isolated digital outputs

**Note:** ***Please note that the DO_VCC pin must be connected to the external power source***.

## 6.3 A/D Signal Source Control

To control the A/D signal source, User should consider the signal type, signal channel and signal range. The EX-92106 provides 16 single-ended or 8 different isolated analog input signals. To avoid ground loops and get more accurate measurement of A/D conversion, it is quite important to understand the signal source type.

- **Single-Ended analog input type**

The single-ended mode has only one input relative to ground and is suitable for connecting with the *floating signal source*. Figure 6-6 shows the single-ended connection.

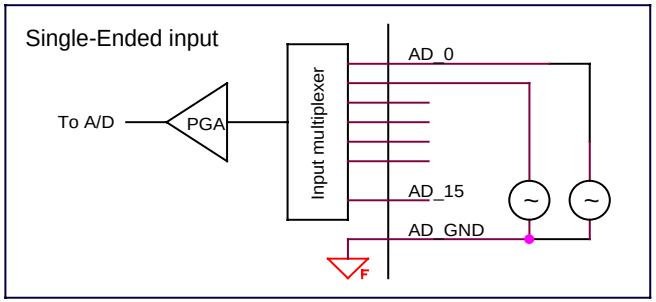



Figure 6-6 Signal sources and single-ended connection

- **Differential analog input type**

The differential input voltage signal is measured is by a pair of signals. The AD circuits measure the voltage difference between the differential pair. The common mode noise can be reduced under this mode. Note that the differential signal pair should be still common ground to the isolation ground plane. Figure 4.2 shows the differential analog signal input connection.

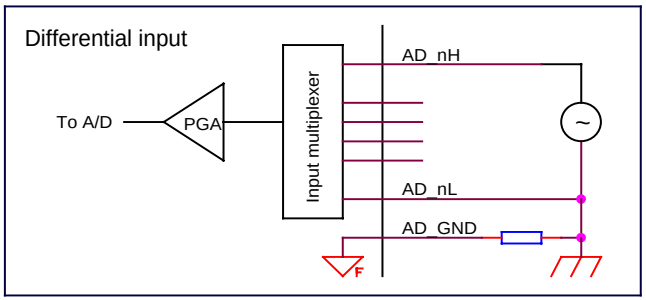



Figure 6-7 Differential analog input connection





- **Signal Conditioning**

Three are 16 SE R/C filters (attenuators) on board for every channel. The RC circuits for each channel are shown in the following diagram, where 'n' is the channel number. User can install the R, C for special purpose such as attenuating the voltage to increase the input voltage range, or used as current sensor

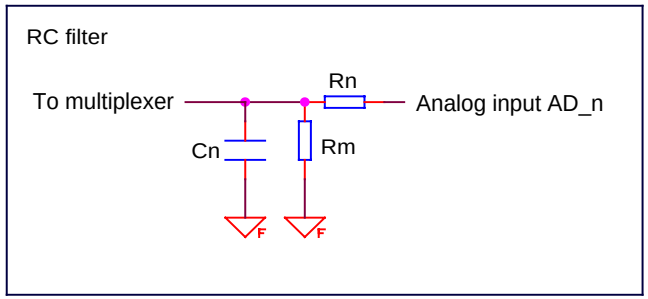



Figure 6-8

The following table shows you the corresponding RC labels for each channel. Please refers to EX-92016 PCB

| Rn | Rm | Cn | Channel number (AD_n) | |
|---|---|---|---|---|
| | | | Single-ended | Differential |
| R3 | R4 | C2 | AD_0 | AD_0H |
| R5 | R6 | C3 | AD_1 | AD_1H |
| R8 | R9 | C5 | AD_2 | AD_2H |
| R12 | R13 | C8 | AD_3 | AD_3H |
| R14 | R15 | C9 | AD_4 | AD_4H |
| R16 | R17 | C10 | AD_5 | AD_5H |
| R18 | R19 | C12 | AD_6 | AD_6H |
| R22 | R23 | C13 | AD_7 | AD_7H |
| R24 | R25 | C15 | AD_8 | AD_0L |
| R26 | R27 | C16 | AD_9 | AD_1L |
| R28 | R29 | C18 | AD_10 | AD_2L |
| R31 | R32 | C21 | AD_11 | AD_3L |
| R34 | R35 | C22 | AD_12 | AD_4L |
| R38 | R39 | C23 | AD_13 | AD_5L |
| R40 | R41 | C24 | AD_14 | AD_6L |
| R43 | R44 | C26 | AD_15 | AD_7L |

Table 6-7

- **Signal channel control**

  There are two ways to control the channel number. The first one is the software programming and the second one is the auto channel scanning, which is controlled by the SCAN mode bit in A/D control register. As SCAN mode is cleared (0), the value of AD control register bit #0~bit #3 defines the channel to be selected.

  As SCAN mode is set 1, the value in control register bit #0~bit #3 defines the ending channel number of auto-scanning operation. Under auto scan mode, the channel is scanning from channel 0 to the ending channel. Whenever a trigger signal is rising, the channel number to be selected will increase automatically. For example, if the ending channel number is 8, the auto channel scanning sequence is 0,1,2,3,4,5,6,7,8,0,1,2,3,4,5,6,7,8, …, until the SCAN mode bit is cleared.

- **Signal input range**

  The maximum A/D signal range of EX-92016 is a ±10 volt when the A/D gain value is 0. The A/D gain control register controls the maximum signal input range. The signal gain is programmable with 4 levels (1,2,4,8). The signal range of the 16 channels will be identical all the time even if the channel number is scanning. The available signal polarity on EX-92016 is bi-polar and uni-polar configuration.

  The following table shows you the maximum input range for each configuration (refer to page 28)

| Polarity (JP2) | Full range (JP3) | Gain | Maximum input range |
|---|---|---|---|
| Uni-polar | 10V | 0 | 0~10V |
| | | 1 | 0 ~ 5 V, |
| | | 2 | 0 ~ 2.5 V, |
| | | 3 | 0~1.25V |
| Bi-polar | 10V | 0 | -5V~+5V |
| | | 1 | -2.5V ~ +2.5 V |
| | | 2 | -1.25V ~ +1.25 V |
| | | 3 | -0.65V~ +0.65V |
| | 20V | 0 | -10V~ +10V |
| | | 1 | -5V ~ +5 V, |
| | | 2 | -2.5V ~ +2.5 V, |
| | | 3 | -1.25V~ +1.25V |

Table 6-8

-





## 6.4 A/D Trigger Source Control

The A/D converter will start to convert the signal to a digital value by a trigger source. In EX-92016 two internal sources can be selected: the software trigger or the timer pacer trigger.

- **Software trigger**

  The trigger source is software controllable in this mode. That is, the A/D conversion is starting when any value is written into the software trigger register. Under this mode, the timing of the A/D conversion is fully controlled by software.

- **Timer Pacer Trigger**

  An on-board timer / counter chip 8254 is used to provide a trigger source for A/D conversion at a fixed rate. Two counters of the 8254 chip are cascaded together to generate trigger pulse with precise period. This mode is ideal for high speed A/D conversion. User can combine this mode with the FIFO half-full interrupt or EOC interrupt to transfer data. It is also possible to use software FIFO polling to transfer data. The A/D trigger, A/D data transfer and Interrupt can be set independently.

## 6.5 A/D Data Transfer Modes

On the EX-92016, the A/D data are buffered in the 1024 (1K) words FIFO memory. The data must be transferred to host memory after the data is ready and before the FIFO is full. EX-92016 provides many data transfer modes that can be used. The different transfer modes are specified as follows:

- **Software Data Polling**

  This mode can be used with software A/D trigger mode. After the A/D conversion is triggered by software, the software should poll the *FF_EF(FIFO Empty)* bit of the A/D control status register. After the A/D conversion is completed, the A/D data is written to FIFO immediately, thus the FF_EF becomes high. You can consider the FF_EF bit as a flag to indicate the converted data ready status

- **FIFO Half-Full Polling**

  The on-board 1 K words FIFO can be stored up to 10.24 ms analog data under 100 KHz sampling rate.

  When the FIFO is half-full and not full, the software can read one "block" (512 words) A/D data without checking the FIFO status. This method is very convenient to read A/D in size of a "block" and it is benefit to software programming.

- **EOC Interrupt Transfer**

  The EX-92016 provides hardware end-of-conversion (EOC) interrupt capability. Under this mode, an interrupt signal is generated when the A/D conversion is ended and the data is ready to be read in the FIFO. After A/D conversion is completed, the hardware interrupt will be inserted and its corresponding Interrupt Service Routine will be invoked and executed. The service program can read the converted data. This method is most suitable for data processing applications under real-time and fixed sampling rate

- **FIFO Half-Full Interrupt Transfer**

The FIFO half-full interrupt transfer mode is useful when the applications do not need real-time processing, but the foreground program is too busy to poll the FIFO data.

Under this mode, an interrupt signal is generated when FIFO becomes half-full. It means there are 512 words data in the FIFO already. The service routine can read a block of data every interrupt occurring. This method is very convenient to read A/D in size of a "block" (512 words) and it is benefit for software programming.

## 6.6 Interrupt Control

- **System Architecture**

The EX-92016's interrupt system is a powerful and flexible system that is suitable for A/D data acquisition The system interrupt can generated by three signals EOC, Half-Full, and Timer Pacer

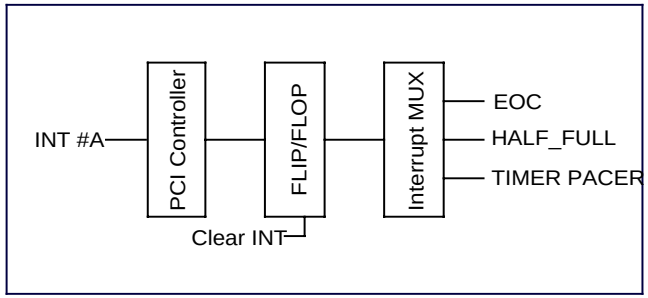



Figure 6-9





## 6.7 Timer/Counter Operation

One 8254 programmable timer/counter chip is installed in EX-92016. Timer #1 and Timer #2 are used for periodically triggering the A/D conversion, and Counter #0 is left free for user applications. The block diagram of the timer/counter system is shown in following diagram.

The timer #1 and timer #2 are cascaded together to generate the timer pacer trigger of A/D conversion. The frequency of the pacer trigger is software controllable. The maximum pacer signal rate is 2MHz/4=500K and the minimum signal rate is 2MHz/65535/ 65535.

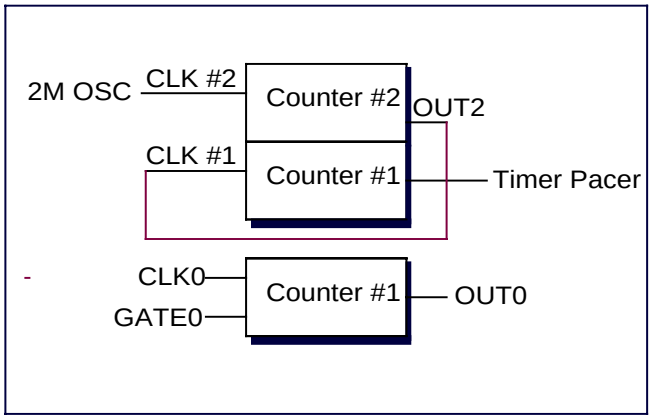



Figure 6-10

# Chapter 7
# Libraries

This chapter describes the software library for operating this card. Only the functions in DOS library and Windows 95 DLL are described. Please refer to the PCIDAQ function reference manual, which included in TOPS CCC CD, for the descriptions of the Windows 98/NT/2000 DLL functions.

## 7.1 Libraries Installation

Please refer to the ***"Software Installation Guid***e" for the detail information about how to install the software libraries for DOS, or Windows 95 DLL, or PCIDAQ for Windows 98/NT/2000.

The device drivers and DLL functions of Windows 98/NT/2000 are included in the PCIDAQ. The TOPS CCC CD also includes the detail examples and readme files

## 7.2 Libraries Installation

This section will show you how to install the software libraries DOSDAQH.LIB for DOS, or Windows 98 DLL, or PCIDAQ for Windows 98/NT/2000.

The device drivers and DLL functions of Windows 98/NT/2000 are included in the PCIDAQ. The TOPS CCC CD also includes the detail examples and readme files

### 7.2.1 How to use the DOSDAQH.LIB in DOS

- **For BC compiler**
  1. Large mode: Add ..\LIB\BC\DOSDAQL.LIB in your project
  1. Huge mode: Add ..\LIB\BC\DOSDAQH.LIB in your project
  2. Include DOSDAQ.H in your source file
- **For MSC compiler**
  1. Large mode: Add ..\LIB\MSC\DOSDAQL.LIB in your project
  2. Huge mode: Add ..\LIB\MSC\DOSDAQH.LIB in your project
  3. Include DOSDAQ.H in your source file
- **For TC compiler**
  4. Large mode: Add ..\LIB\TC\DOSDAQL.LIB in your project
  5. Huge mode: Add ..\LIB\TC\DOSDAQH.LIB in your project
  6. Include DOSDAQ.H in your source file





### 7.2.2 How to use the PCIDAQ.DLL s in Windows

- **VC++6.0:**
  7. Add file '../Include/PCIDAQ.H' in your project
  8. In link page of menu project| setting, add '../LIB/PCIDAQ.LIB' in the blank of Objects/Library Modules
  9. Add this sentence "#include '../Include/PCIDAQ.H' " to the head of your main file.
- **Visual BASIC:**
  2. Add file '../Include/Declare.bas' in your project.
- **Delphi:**
  10. Add file '../Include/Declare.pas' in your project
  11. Add this sentence "uses Declare;" in the head of your unit.pas
- **C++Builder:**
  12. Add file '../Include/PCIDAQ.H' and '../Lib/PCIDAQ_CB.lib' to your project
  13. Add this sentence "#include '../Include/PCIDAQ.H' " to head of your main file.

**Note:** For more information, please refer to program in directory '../Example/'

## 7.3 Summary of function calls

- **PCI common functions**

| | | |
|---|---|---|
| Get Card's number | Get PCI ID code of EX-92016 | 50 |
| Get driver version | Get version number of PCIDAQ.DLL | 51 |
| Get PCI Bus and Slot number | Get PCI bus and slot number occupied by EX-92016 | 52 |
| Close card | Close EX-92016 card before terminating program | 53 |

- **Digital I/O functions**

| Function | Description | page |
|---|---|---|
| Read digital input data | Read digital input port data (8-bit) | 54 |
| Write data to digital output port | Write data to digital output port | 55 |
| Read back digital output data | Read back current value of digital output port | 56 |
| Set bit of digital output port | Activate a bit of digital output port (output transistor ON) | 57 |
| Reset bit of digital output port | De-activate a bit of digital output port (output transistor OFF) | 58 |

- **Timer #0 functions**

| Function | Description | page |
|---|---|---|
| Initial and start timer #0 | Initial and start Timer #0 | 71 |
| Read Timer #0 current value | Read timer #0 counter value | 72 |
| Stop Timer #0 | Stop timer #0 | 73 |





- **AD conversion function**

| Function | Description | page |
|---|---|---|
| Initialize AD triggered by software | Initial AD converter for software polling | 59 |
| Single channel AD acquire | Single channel AD conversion by software trigger | 60 |
| Multiple channel AD acquire | Multiple channels AD conversion by software trigger | 61 |
| Initial AD triggered by timer pacer interrupt | Initial AD converter for timer pacer trigger | 61 |
| Start auto scan triggered by timer pacer | Start auto AD channels conversion periodically | 63 |
| Get auto scan AD data | Get data from scan buffer | 64 |
| Stop AD auto scan | Stop auto scan operation | 65 |
| Start single channel conversion | Start to single AD channel conversion with timer pacer | 66 |
| Start multiple channels conversion | Start to multiple AD channel conversion with timer pacer | 67 |
| Read AD Interrupt status | Read interrupt of AD conversion | 68 |
| Transfer AD Interrupt data | Transfer AD data | 69 |
| Stop timer pacer interrupt of AD conversion | Stop timer pacer conversion operation | 70 |

## 7.4 Open card

**Description:**

Because the EX-92016 is PCI bus architecture and meets the plug and play design, the IRQ and base_address ( pass-through address) are assigned by system BIOS directly. EX-92016 cards have to be initialized by this function before calling other functions.

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_Open (WORD cardNo);
```

**C/C++ (Windows)**

```
WORD D_2016_Open (WORD *ExistCards);
```

**Visual BASIC (Windows)**

```
Function W_2016_Open (ByRef ExitedCards As Long) As Long
```

**Delphi**

```
Function W_2016_Open (var ExistedCards:Integer): Integer;
```

**Argument:**

CardNo: card number (1,2,3,4) ( for DOS only)

existCards: Return a value shows how many EX-92016 cards are installed in your system. (For Windows only)

**Return Code:**

Error code (Please refer to PCIDAQ.H or DOSDAQ.H)





## 7.5 Get Card's number

**Description:**

Get the cards number that is set by jumper on cards.

**Syntax:**

**C/C++(DOS)**

```
void D_2016_GetCardsID (WORD *CardsIDArray);
```

**C/C++(Windows)**

```
WORD W_2016_GetCardsID (WORD *CardsIDArray);
```

**Visual BASIC (Windows)**

```
Function W_2016_GetCardsID (ByRef CardsIDArray As Long)
         As Integer
```

**Delphi**

```
Function W_2016_GetCardsID (var CardsIDArray:Word):Word;
```

**Argument:**

CardsIDArray : This array return card number (1,2,3,4). You should define a 4 elements array, and then pass the array's pointer to this function.

**Return Code:**

Error code (Please refer to PCIDAQ.H or DOSDAQ.H)

## 7.6 Get driver version

**Description:**

This function returns the version of PCIDAQ.DLL driver

**Syntax:**

**C/C++ (DOS)**

```
void D_2016_Version(char *version)
```

**C/C++ (Windows)**

```
WORD D_2016_Version (void)
```

**Visual BASIC (Windows)**

```
Function W_2016_Version () As Long
```

**Delphi**

```
Function W_2016_Version ():Integer;
```

**Argument:**

Version: return the PCIDAQ.DLL driver version string (DOS only)

**Return Code:**

The version of PCIDAQ.DLL (Windows only) in *integer* data format





## 7.7 Get PCI Bus and Slot number

**Description:**

Get the PCI bus and slot number occupied by EX-92016 card

**Syntax:**

**C/C++(DOS)**

```
WORD D_2016_GetBusSlot (WORD cardNo, WORD *bus,WORD *slot);
```

**C/C++ (Windows)**

```
WORD W_2016_GetBusSlot (WORD cardNo, WORD *bus,WORD *slot);
```

**Visual BASIC (Windows)**

```
Function W_2016_GetBusSlot (ByVal cardNo As Long, ByRef bus As
         Long, ByRef slot As Long) As Long
```

**Delphi**

```
Function W_2016_GetBusSlot (cardNo:Integer;var bus:Integer;
         var slot:Integer):Integer;
```

**Argument:**

cardNo: card number (1,2,3,4)

bus: return PCI bus Number

slot: return PCI slot Number of the bus

**Return Code:**

Error code (Please refer to PCIDAQ.H or DOSDAQ.H)

## 7.8 Close card

**Description:**

The IRQ and base_address of EX-92016 (pass-through address) are assigned by system BIOS directly. This function should be called to release all system resource before terminating application program

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_Close (WORD cardNo)
```

**C/C++ (Windows)**

```
W_2016_Close (void)
```

**Visual BASIC (Windows)**

```
Function W_2016_Close ()
```

**Delphi**

```
Function W_2016_Close ();
```

**Argument:**

CardNo: card number (1,2,3,4) (DOS only)

All EX-92016 cards are closed after calling this function (Windows only)

**Return Code:**

None





## 7.9 Read digital input data

**Description:**

This function is used to read data from digital input port. You can get 8-bit input data from EX-92016 by calling this function.

**Syntax:**

**C/C++(DOS)**

```
WORD D_2016_Read_Di (WORD cardNo, WORD *DiData);
```

**C/C++ (Windows)**

```
WORD W_2016_Read_Di (WORD cardNo, WORD *DiData);
```

**Visual BASIC (Windows)**

```
Function W_2016_Read_Di (ByVal cardNo As Long, ByRef DiData As
         Long) As Long
```

**Delphi**

```
Function W_2016_Read_Di (cardNo:Integer; var DiData:Integer):
         Integer;
```

**Argument:**

cardNo: card number (1,2,3,4)

Didata: return digital input data

**Return Code:**

Error code (Please refer to PCIDAQ.H or DOSDAQ.H)

## 7.10 Write data to digital output port

**Description:**

This function is used to write data (byte) to output port. You can send 8-bit output data to EX-92016 by calling this function.

**Syntax:**

**C/C++(DOS)**

```
WORD D_2016_Write_Do (WORD cardNo, WORD Data);
```

**C/C++ (Windows)**

```
WORD W_2016_Write_Do (WORD cardNo, WORD Data);
```

**Visual BASIC(Windows)**

```
Function W_2016_Write_Do (ByVal cardNo As Long, ByVal Data As
         Long) As Long
```

**Delphi**

```
Function W_2016_Write_Do(cardNo:Integer; Data:
         Integer):Integer;
```

**Argument:**

cardNo: card number (1,2,3,4)

Data: Data be written to output port

**Return Code:**

Error code (Please refer to PCIDAQ.H or DOSDAQ.H)





## 7.11 Read back digital output data

**Description:**

This function is used to read current data of output port. You can read back 8-bit output data of EX-92016 by calling this function.

**Syntax:**

**C/C++(DOS)**

```
WORD D_2016_Read_Do (WORD cardNo, WORD *DoData);
```

**C/C++ (Windows)**

```
WORD W_2016_Read_Do (WORD cardNo, WORD *DoData);
```

**Visual BASIC (Windows)**

```
Function W_2016_Read_Do (ByVal cardNo As Long, ByRef DoData As
         Long) As Long
```

**Delphi**

```
Function W_2016_Read_Do (cardNo:Integer; var
         DoData:Integer):Integer;
```

**Argument:**

cardNo: card number (1,2,3,4)

Data : return current output data

**Return Code:**

Error code (Please refer to PCIDAQ.H or DOSDAQ.H)

## 7.12 Set bit of digital output port

**Description:**

Set one digital output bit to ***short*** state (output transistor ON).

**Syntax:**

**C/C++(DOS)**

```
WORD D_2016_Set_Do_Bit (WORD cardNo, WORDbitNo);
```

**C/C++ (Windows)**

```
WORD W_2016_Set_Do_Bit (WORD cardNo, WORDbitNo);
```

**Visual BASIC (Windows)**

```
Function W_2016_Set_Do_Bit (ByVal cardNo As Long,
         ByVal bitNo As Long) As Long
```

**Delphi**

```
Function W_2016_Set_Do_Bit (cardNo:Integer;
         bitNo:Integer):Integer;
```

**Argument:**

cardNo: card number (1,2,3,4)

bitNo : bit number(0 to 7)

**Return Code:**

Error code (Please refer to PCIDAQ.H or DOSDAQ.H)





## 7.13 Reset bit of digital output port

**Description:**

Set one digital output bit to ***open*** state (output transistor OFF)

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_Reset_Do_Bit (WORD cardNo, WORD bitNo);
```

**C/C++ (Windows)**

```
WORD W_2016_Reset_Do_Bit (WORD cardNo, WORD bitNo);
```

**Visual BASIC (Windows)**

```
Function W_2016_Reset_Do_Bit (ByVal cardNo As Long,
         ByVal bitNo As Long) As Long
```

**Delphi**

```
Function W_2016_Reset_Do_Bit (cardNo:Integer; bitNo:
         Integer):Integer;
```

**Argument:**

cardNo: card number (1,2,3,4)

bitNo: bit number(0 to 7)

**Return Code:**

Error code (Please refer to PCIDAQ.H or DOSDAQ.H)

## 7.14 Initialize AD triggered by software

**Description:**

Initialize AD Range of AD conversion by software trigger, it must be called before W_2026_SingleChannel_AD_Acquire or W_2026_MultiChannel_AD_Acquire functions

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_AD_Acquire_Initial (WORD cardNo,WORD AD_Range);
```

**C/C++ (Windows)**

```
WORD W_2026_AD_Acquire_Initial (WORD cardNo,WORD AD_Range);
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Acquire_Initial (ByVal CardNo As Long,
         ByVal AD_Range As Long) As Integer
```

**Delphi**

```
Function W_2026_AD_Acquire_Initial (cardNo:Word; AD_Range:
         Word):Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

AD_Range: Set maximum input range (associated with JP2 and JP3)

| AD_Range value | Gain | Polarity (JP2) | Full Range (JP3) | Max. Input range |
|---|---|---|---|---|
| 0 | 1 |  Bipolar |  10V | -5~+5V |
| 1 | 2 | | | -2.5V~+2.5V |
| 2 | 4 | | | -1.25V~+1.25V |
| 3 | 8 | | | -0.625V~+0.625V |
| 4 | 1 |  Unipolar |  10V | 0V~+10V |
| 5 | 2 | | | 0V~+5V |
| 6 | 4 | | | 0V~+2.5V |
| 7 | 8 | | | 0V~+1.25V |
| 8 | 1 |  Bipolar |  20V | -10~+10V |

**Return Code:**

Error code (Please refer to PCIDAQ.H)





## 7.15 Single channel AD acquire

**Description:**

Single channel AD conversion by software trigger.

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_SingleChannel_AD_Acquire (WORD cardNo,WORD ADChNo ,
     float *ADData);
```

**C/C++ (Windows)**

```
WORD W_2026_SingleChannel_AD_Acquire (WORD cardNo,WORD ADChNo,
     float* ADData);
```

**Visual BASIC (Windows)**

```
Function W_2026_SingleChannel_AD_Acquire (ByVal CardNo As Long,
         ByVal ADChNo As Long, ByRef ADData As Single) As
         Integer
```

**Delphi**

```
Function W_2026_SingleChannel_AD_Acquire (cardNo:Word;
         ADChNo:Word; var ADData : Single) :Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

ADChNo: AD Conversion Channel (0~15).

ADData: Return AD conversion voltage value.

**Return Code:**

Error code (Please refer to PCIDAQ.H)

## 7.16 Multiple channel AD acquire

**Description:**

Multi channel AD conversion by software trigger.

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_MultiChannel_AD_Acquire (WORD cardNo,WORD EndChNo,
     float *AD_Buffer);
```

**C/C++ (Windows)**

```
WORD W_2026_MultiChannel_AD_Acquire (WORD cardNo,WORD
     EndChNo,float* AD_Buffer);
```

**Visual BASIC (Windows)**

```
Function W_2026_MultiChannel_AD_Acquire (ByVal CardNo As Long,
         ByVal EndChNo As Long, ByRef AD_Buffer As Single) As
         Integer
```

**Delphi**

```
Function W_2026_MultiChannel_AD_Acquire (cardNo:Word;
         EndChNo:Word; var AD_Buffer: Single) :Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

EndChNo: AD Conversion end Channel (0~15).

AD_Buffer: The start address of the memory buffer to store the A/D voltage data. The buffer size must large than the number of A/D conversion. Your should use **"float"** data format in AD_Buffer

**Return Code:**

Error code (Please refer to PCIDAQ.H)

## 7.17 Initial AD triggered by timer pacer interrupt

**Description:**

This function is used to set the Timer #1 and Timer#2 for generating constant A/D sampling rate dedicatedly, and range of analog input.

1. *This function must be called before calling following functions* "**W_2026_AD_Int_StartAutoScan ()**", "**W_2026_AD_Int_StartSingleChannel ()**" and "**W_2026_AD_Int_StartMultiChannel ()**"
2. Call function "**W_2026_AD_Int_Stop ()**" to stop timer pacer AD conversion process





**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_AD_Int_Initial (WORD cardNo,WORD AD_Range, float
     Scan_Rate);
```

**C/C++ (Windows)**

```
WORD W_2026_AD_Int_Initial (WORD cardNo,WORD AD_Range,float
     Scan_Rate );
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Int_Initial (ByVal CardNo As Long, ByVal
         AD_Range As Long, ByVal Scan_Rate As Single) As Integer
```

**Delphi**

```
Function W_2026_AD_Int_Initial (cardNo:Word; AD_Range:Word;
         Scan_Rate: Single ) :Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

Scan_Rate: Sampling rate (samples/sec) from 0.0047Hz to 100K Hz.

AD_Range: Set maximum input range (associated with JP2 and JP3)

| AD_Range value | Gain | Polarity (JP2) | Full Range (JP3) | Max. Input range |
|---|---|---|---|---|
| 0 | 1 |  Bipolar |  10V | -5~+5V |
| 1 | 2 | | | -2.5V~+2.5V |
| 2 | 4 | | | -1.25V~+1.25V |
| 3 | 8 | | | -0.625V~+0.625V |
| 4 | 1 |  Unipolar |  10V | 0V~+10V |
| 5 | 2 | | | 0V~+5V |
| 6 | 4 | | | 0V~+2.5V |
| 7 | 8 | | | 0V~+1.25V |
| 8 | 1 |  Bipolar |  20V | -10~+10V |

**Return Code:**

Error code (Please refer to PCIDAQ.H)

## 7.18 Start auto scan triggered by timer pacer

Start AD conversion from channel #0 to End channel, the AD conversion is triggered by timer pacer periodically

The AD input channel will be automatically changed to next channel until End channel reached and then restart from channel #0 again. The AD data will be updated periodically

**Note:** Due to swapping channel, the maximum sample rate=70KHz for this function.

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_AD_Int_StartAutoscan (WORD cardNo,WORD EndChNo);
```

**C/C++ (Windows)**

```
WORD W_2026_AD_Int_StartAutoScan (WORD cardNo,WORD EndChNo);
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Int_StartAutoScan (ByVal CardNo As Long,
         ByVal EndChNo As Long) As Integer
```

**Delphi**

```
Function W_2026_AD_Int_StartAutoScan (cardNo:Word;
         EndChNo:Word):Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

EndChNo: AD Conversion end Channel (0~15).

**Return Code:**

Error code (Please refer to PCIDAQ.H)





## 7.19 Get auto scan AD data

**Description:**

Get AD conversion data from auto scan buffer

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_AD_Int_GetScanData (WORD cardNo,WORD ChannelNo,
     float *ADData);
```

**C/C++ (Windows)**

```
WORD W_2026_AD_Int_GetScanData (WORD cardNo,WORD ChannelNo,
     float *ADData);
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Int_GetScanData (ByVal CardNo As Long,
         ByVal ChannelNo As Long, ByRef ADData As Single)
         As Integer
```

**Delphi**

```
Function W_2026_AD_Int_GetScanData (cardNo:Word;
         ChannelNo:Word; var ADData: Single):Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

ChannelNo: AD input channel number (0~15)

*ADData: return data from auto scan buffer

**Return Code:**

Error code (Please refer to PCIDAQ.H)

## 7.20 Stop AD auto scan

**Description:**

Stop AD auto scan processes

**Syntax:**

**C/C++ (Dos)**

```
WORD D_2016_AD_Int_StopAutoscan (WORD cardNo);
```

**C/C++ (Windows)**

```
WORD W_2026_AD_Int_StopAutoScan (WORD cardNo);
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Int_StopAutoScan (ByVal CardNo As Long)
         As Integer
```

**Delphi**

```
Function W_2026_AD_Int_StopAutoScan (cardNo:Word):Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

**Return Code:**

Error code (Please refer to PCIDAQ.H)





## 7.21 Start single channel conversion

**Description:**

This function is used to converting single channel analog input with fixed counting

You can use the function "*W_2016_AD_Int_Status*" to get how many AD conversions completed and function "*W_2016_AD_Int_DataTransfer*" to read data from buffer

**Syntax:**

**C/C++ (Dos)**

```
WORD D_2016_AD_Int_StartSingleChannel (WORD cardNo,WORD
     ChannelNo,unsigned int Count);
```

**C/C++ (Windows)**

```
WORD W_2026_AD_Int_StartSingleChannel (WORD cardNo,WORD ChNo,
     long Count);
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Int_StartSingleChannel (ByVal CardNo As
         Long, ByVal ChNo As Long, ByVal Count As Long) As
         Integer
```

**Delphi**

```
Function W_2026_AD_Int_StartSingleChannel (cardNo:Word;
         ChNo:Word; Count: Word) :Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

ChNo: Channel number (0~15)

Count: counts of AD conversion

**Return Code:**

Error code (Please refer to PCIDAQ.H)

## 7.22 Start multiple channels conversion

**Description:**

This function is used to converting multiple channel analog inputs starting from channel #0 to end channel with fixed counting

You can use the function "*W_2016_AD_Int_Status*" to get how many AD conversions completed and function "*W_2016_AD_Int_DataTransfer*" to read data from buffer

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_AD_Int_StartMultiChannel (WORD cardNo,WORD
     EndChNo, unsigned int Count);
```

**C/C++ (Windows)**

```
WORD W_2026_AD_Int_StartMultiChannel (WORD cardNo,WORD
     EndChNo, long Count);
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Int_StartMultiChannel (ByVal CardNo As Long,
         ByVal EndChNo As Long, ByVal Count As Long) As Integer
```

**Delphi**

```
Function W_2026_AD_Int_StartMultiChannel (cardNo:Word;
         EndChNo:Word; Count: Word) :Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

EndChNo: AD Conversion end Channel (0~15).

Count: Counts of AD conversion

**Return Code:**

Error code (Please refer to PCIDAQ.H)





## 7.23 Read AD Interrupt status

**Description:**

This function is used to check the status of timer pacer interrupt operation. The W_2026_AD_Int_StartMultiChannel () and W_2026_AD_Int_StartSingleChannel () are executed on background, therefore you can issue this function to check the status of interrupt operation.

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_AD_Int_Status (WORD cardNo,unsigned int *count);
```

**C/C++ (Windows)**

```
WORD W_2026_AD_Int_Status (WORD cardNo,long *count);
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Int_Status (ByVal CardNo As Long, ByRef
         Count As Long) As Integer
```

**Delphi**

```
Function W_2026_AD_Int_Status (cardNo:Word; var
         Count :Integer) :Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

count: The A/D conversion block number performed currently.

**Return Code:**

Error code (Please refer to PCIDAQ.H)

## 7.24 Transfer AD Interrupt data

**Description:**

Use this function to transfer AD data to memory buffer after executing W_2026_AD_Int_StartMultiChannel () and W_2026_AD_Int_StartSingleChannel () functions

**Syntax:**

**C/C++ (Windows)**

```
WORD W_2026_AD_Int_DataTransfer (WORD cardNo,float* AD_Buffer,
     long count);
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Int_DataTransfer (ByVal CardNo As
         Long,ByRef AD_Buffer As Single,ByVal Count As Long)
         As Integer
```

**Delphi**

```
Function W_2026_AD_Int_DataTransfer (cardNo:Word;var D_Buffer:
         Single;Count:Word):Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

AD_Buffer: The start address of the memory buffer to store the A/D voltage data. The buffer size must large than the number of A/D conversion. Your should use **"float"** data format in AD_Buffer

Count: The number of A/D conversion be transferred

**Return Code:**

Error code (Please refer to PCIDAQ.H)





## 7.25 Stop timer pacer interrupt of AD conversion

**Description:**

This function is used to stop the timer pacer interrupt data transfer functions. After executing this function, the internal A/D trigger is disabled and the A/D timer is stopped.

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_AD_Int_Stop (WORD cardNo);
```

**C/C++ (Windows)**

```
WORD W_2026_AD_Int_Stop (WORD cardNo);
```

**Visual BASIC (Windows)**

```
Function W_2026_AD_Int_Stop (ByVal CardNo As Long) As Integer
```

**Delphi**

```
Function W_2026_AD_Int_Stop (cardNo:Word):Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

**Return Code:**

Error code (Please refer to PCIDAQ.H)

## 7.26 Initial and start timer #0

**Description:**

Set 8254's counter #0 with work mode and initial value.

**Syntax:**

**C/C++ (DOS)**

```
WORD D_2016_Timer0_Start (WORD cardNo,WORD TimerMode,unsigned
     int CounterValue);
```

**C/C++ (Windows)**

```
WORD W_2026_Timer0_Start (WORD cardNo,WORD TimerMode,long
     CounterValue);
```

**Visual BASIC (Windows)**

```
Function W_2026_Timer0_Start (ByVal CardNo As Long, ByVal
         TimerMode As Long, ByVal CounterValue As Long)
         As Integer
```

**Delphi**

```
Function W_2026_Timer0_Start (cardNo:Word; TimerMode:Word;
         CounterValue:Word):Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

TimerMode: 8254's work mode, it must be 0 to 5.

CounterValue: 8254's initial value. It must be 0 to 65535.

**Return Code:**

Error code (Please refer to PCIDAQ.H)





## 7.27 Read Timer #0 current value

**Description:**

Read counter #0's current counter value.

**Syntax:**

**C/C++ (Dos)**

```
WORD D_2016_Timer0_Read (WORD cardNo,unsigned int
     *CounterValue);
```

**C/C++ (Windows)**

```
WORD W_2026_Timer0_Read (WORD cardNo,long* CounterValue);
```

**Visual BASIC (Windows)**

```
Function W_2026_Timer0_Read (ByVal CardNo As Long, ByRef
         CounterValue As Long) As Integer
```

**Delphi**

```
Function W_2026_Timer0_Read (cardNo:Word; CounterValue:
         Word):Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

CounterValue: return counter #0's current value.

**Return Code:**

Error code (Please refer to PCIDAQ.H)

## 7.28 Stop Timer #0

**Description:**

Stop counter #0 by setting to work mode 5.

**Syntax:**

**C/C++ (Dos)**

```
WORD D_2016_Timer0_Stop (WORD cardNo, unsigned int
     *CounterValue);
```

**C/C++ (Windows)**

```
WORD W_2026_Timer0_Stop (WORD cardNo, long *CounterValue);
```

**Visual BASIC (Windows)**

```
Function W_2026_Timer0_Stop (ByVal CardNo As Long, ByRef
         CounterValue As Long) As Integer
```

**Delphi**

```
Function W_2026_Timer0_Stop (cardNo:Word; CounterValue:
         Word):Word;
```

**Argument:**

cardNo: card number (1,2,3,4)

CounterValue: return counter #0's current value.

**Return Code:**

Error code (Please refer to PCIDAQ.H)





# Chapter 8
# Calibration & Utilities

Users can calibrate the analog input channels under the users' operating environment for optimizing the accuracy. This chapter will guide you to calibrate your EX-92016 to an accuracy condition.

## 8.1 Calibration

Before calibrating your EX-92016 card, you should prepare some equipments for the calibration:

- A 5 1/2 digit multimeter (6 1/2 is recommended)
- A voltage calibrator or a very stable and noise free DC voltage generator.

## 8.2 VR Assignment

There are four variable resistors (VR) on the EX-92016 board to allow you making accurate adjustment on A/D channels. The function of each VR is specified as shown below.

- VR1 A/D bi-polar offset adjustment
- VR2 A/D uni-polar offset adjustment
- VR3 A/D full scale adjustment
- VR4 PGA offset adjustment

### 6.1.2 A/D Adjustment

- **PGA offset calibration**
  1. Set ***JP1*** as single-ended input
  2. Short the A/D channel #0 (pin 19 of CN1) to ground (AD_GND: pin 11 of CN1).
  3. Use multi-meter to measure the voltage between pin 11 of U13 and ground (AD_GND: pin 11 of CN1)
  4. Adjust ***VR4*** until the read out value approach to zero.
- **Uni-polar input**
  1. Set ***JP1*** as single-ended input
  2. Set ***JP2*** as uni-polar A/D input.
  3. Set ***JP3*** to 10V full range.
  4. Short the A/D channel 0 (pin 19 of CN1) to ground (AD_GND: pin 11 of CN1).
  5. 6. Set the analog gain = 1 and channel number #0 by software.
  6. Adjust ***VR2*** to obtain reading between 0.000~0.001.

7. Applied a +10V reference input signal to A/D channel 0, and trim the ***VR3*** to obtain reading approach to 9.999V

- **Bi-polar input**

1. Set ***JP1*** as single-ended input
2. Set ***JP2*** as bi-polar A/D input.
3. Set ***JP3*** to 20V full range.
4. Set the analog gain = 1 and channel number #0 by software.
5. Short the A/D channel #0 (pin 19 of CN1) to ground (AD_GND: pin 11 of CN1).
6. Adjust ***VR1*** to obtain reading between -0.001V and +0.001V
7. Applied a +10V reference input signal to A/D channel #0 (pin 19 of CN1), and trim the ***VR2*** to obtain reading between +9.999V.





# Chapter 9
# EX-9837 Terminal board

EX-9837 Screw-terminal termination board features one 37-pin D-type connector for easy maintenance, wiring, and installation. It provides 37 channels that are accessed through a 37-pin D-type connector.

## 9.1 Main features

- Low-cost screw-terminal board for the all EX series with 37-pin D-type connector
- Reserved space for signal-conditioning circuits such as low-pass filter, voltage attenuator and current shunt
- Industrial type termination blocks permit heavy-duty and reliable signal connections
- Table-top mounting using nylon standoffs. Screws and washers provided for panel or wall mounting
- Dimensions: 80mm (W) x 181mm (H)