PCIS-DASK

Data Acquisition Software Development Kit
for NuDAQ® PCI Bus Cards

User's Manual

Manual Rev. 2.01

Revision Date: September 27, 2008

Part No: 50-11223-2020

Recycled Paper

Advance Technologies; Automate the World.

Copyright 2008 ADLINK TECHNOLOGY INC.

All Rights Reserved.

Disclaimer

The information in this document is subject to change without prior notice in order to improve reliability, design, and function and does not represent a commitment on the part of the manufacturer.

In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

This document contains proprietary information protected by copyright. All rights are reserved. No part of this manual may be reproduced by any mechanical, electronic, or other means in any form without prior written permission of ADLINK.

Trademark Information

NuDAQ is a registered trademark of ADLINK Technology Inc.

Product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

Getting service

Customer satisfaction is our top priority. Contact us should you require any service or assistance.

ADLINK TECHNOLOGY INC.

Web Site	http://www.adlinktech.com
Sales & Service	service@adlinktech.com
Telephone No.	+886-2-8226-5877
Fax No.	+886-2-8226-5717
Mailing Address	9F No. 166 Jian Yi Road, Chungho City, Taipei Hsien 235, Taiwan

ADLINK TECHNOLOGY AMERICA, INC.

Sales & Service	info@adlinktech.com
Toll-Free	+1-866-4-ADLINK (235465)
Fax No.	+1-949-727-2099
Mailing Address	8900 Research Drive, Irvine, CA 92618, USA

ADLINK TECHNOLOGY EUROPEAN SALES OFFICE

Sales & Service	emea@adlinktech.com
Toll-Free	+49-211-4955552
Fax No.	+49-211-4955557
Mailing Address	Nord Carree 3, 40477 Düsseldorf, Germany

ADLINK TECHNOLOGY SINGAPORE PTE LTD

Sales & Service	singapore@adlinktech.com
Telephone No.	+65-6844-2261
Fax No.	+65-6844-2263
Mailing Address	84 Genting Lane #07-02A, Cityneon Design Center, Singapore 349584

ADLINK TECHNOLOGY INDIA LIAISON OFFICE

Sales & Service	india@adlinktech.com
Telephone No.	+91-80-57605817
Fax No.	+91-80-26671806
Mailing Address	No. 1357, Ground Floor, "Anupama", Aurobindo Marg JP Nagar (Ph-1) Bangalore - 560 078

ADLINK TECHNOLOGY BEIJING

Sales & Service	market@adlinkchina.com.cn
Telephone No.	+82-2-20570565
Fax No.	+82-2-20570563
Mailing Address	4F, Kostech Building, 262-2, Yangjae-Dong, Seocho-Gu, Seoul, 137-130, Korea

ADLINK TECHNOLOGY BEIJING

Sales & Service	market@adlinkchina.com.cn
Telephone No.	+86-10-5885-8666
Fax No.	+86-10-5885-8625
Mailing Address	Room 801, Building E, Yingchuangdongli Plaza, No.1 Shangdidonglu, Haidian District, Beijing, China

ADLINK TECHNOLOGY SHANGHAI

Sales & Service	market@adlinkchina.com.cn
Telephone No.	+86-21-6495-5210
Fax No.	+86-21-5450-0414
Mailing Address	Floor 4, Bldg. 39, Caoheting Science and Technology Park, No.333 Qinjiang Road, Shanghai , China

ADLINK TECHNOLOGY SHENZHEN

Sales & Service	market@adlinkchina.com.cn
Telephone No.	+86-755-2643-4858
Fax No.	+86-755-2664-6353
Mailing Address	C Block, 2nd Floor, Building A1, Cyber-tech Zone, Gaoxin Ave. 7.S, High-tech Industrial Park S., Nanshan District, Shenzhen, Guangdong Province, China

Using this manual

Audience and scope

This manual guides you when using the PCIS-DASK software driver for NuDAQ PCI bus data acquisition cards. This manual also describes how to install and use the software library and meet your requirements when creating programs for your software applications.

How this manual is organized

This manual is organized as follows:

Chapter 1 Introduction: This chapter intoduces the PCIS-DASK and lists all DAQ modules and language environments which the program supports.

Chapter 2 Function Classes: This chapter describes the classes of functions which the PCIS-DASK supports.

Chapter 3 Building Applications: This section describes the fundamentals of building PCIS-DASK applications in Windows and Linux.

Chapter 4 Application Hints: This chapter provides the PCIS-DASK programming schemes for various DAQ operations.

Chapter 5 Continuous Data Transfer: This section illustrates the mechanism and techniques that PCIS-DASK uses for continuous data transfer.

Chapter 6 Utilities: This chapter describes the Win32 and PCIS-DASK/X utilities.

Chapter 7 Sample Programs: This chapter provides some PCIS-DASK sample programs for supported module

Chapter 8 Distribution of Applications: This section lists the files, installers, and manual installation procedures needed when distributing your PCIS-DASK-based applications.

Conventions

Take note of the following conventions used throughout the manual to make sure that you perform certain tasks and instructions properly.

NOTE	Additional information, aids, and tips that help you perform particular tasks.
IMPORTANT	Critical information and instructions that you MUST perform to complete a task.
WARNING	Information that prevents physical injury, data loss, module damage, program corruption etc. when trying to complete a particular task.

1 Introduction

The PCIS-DASK is a software development kit for NuDAQ data acquisition cards utilizing the PCI bus. With high performance data acquisition driver, the PCIS-DASK lets you develop custom applications under Windows® NT/98/2000/XP/Server 2003 and Linux environments.

With memory and data buffer management capabilities, the PCIS-DASK gives you freedom from dealing with complex issues and focus more on developing your applications. The PCIS-DASK also implements simple communication with NuDAQ PCI-bus cards, while the easy-to-use functions allow you to utilize all the card's features in a high-level way.

The PCIS-DASK also delivers you the advantage of all the power features of Microsoft® Win32 System and Linux for your data acquisition applications, including running multiple applications and using extended memory. The PCIS-DASK's flawless support for Visual Basic environment makes it easy to create custom user interfaces and graphics.

In addition to the software drivers, the PCIS-DASK comes with sample programs for your reference. These sample programs help you develop your applications quickly and conveniently.

1.1 Hardware Support

The PCIS-DASK currently supports the following NuDAQ data acquisition and NuIPC CompactPCI cards:

- PCI-6202
- PCI-6208A/cPCI-6208A
- PCI-6208V/16V/cPCI-6208V
- PCI-6308A
- PCI-6308V
- PCI-7200/cPCI-7200
- PCI-7230/cPCI-7230
- PCI-7233/PCI-7233H
- PCI-7234
- PCI-7224
- PCI-7248/cPCI-7248
- cPCI-7249R
- PCI-7250
- cPCI-7252
- PCI-7256
- PCI-7258
- PCI-7260
- PCI-7296
- PCI-7300A/cPCI-7300A
- PCI-7348
- PCI-7396
- PCI-7432/cPCI-7432
- PCI-7433/cPCI-7433
- PCI-7434/cPCI-7434
- cPCI-7432R
- cPCI-7433R
- cPCI-7434R
- PCI-7442
- PCI-7443
- PCI-7444
- cPCI-7452
- PCI-8554
- PCI-9111
- PCI-9112/cPCI-9112
- PCI-9113
- PCI-9114
- cPCI-9116
- PCI-9118
- PCI-9221
- PCI-9222/9223
- PCI-9812/10

NOTE ADLINK periodically upgrades the PCIS-DASK for new cards/modules. Check the card/modules's Release Notes to know if PCIS-DASK supports it.

1.2 Language Support

The PCIS-DASK is a DLL (Dynamic-Link Library) version for use-with Windows® and Linux environments. It works with any Windows programming language that allows calls to a DLL. These include Microsoft® Visual C/C++ (4.0 or higher versions), Borland C++ (5.0 or higher versions), or Microsoft® Visual Basic (4.0 or higher version). In Linux, it works with any 32-bit compiler, such as gcc.

The PCIS-DASK also comes with a prototype function that supports Borland Delphi 2.x (32-bit) or higher versions.

2 Function Classes

This chapter describes the classes of functions that the PCIS-DASK supports.

All PCIS-DASK functions are grouped into different classes:

- General Configuration Function Group
- Actual Sampling Rate Function Group
- Analog Input Function Group
  - Analog Input Configuration Functions
  - One-Shot Analog Input Functions
  - Continuous Analog Input Functions
  - Asynchronous Analog Input Monitoring Functions
- Analog Output Function Group
  - Analog Output Configuration Functions
  - One-Shot Analog Output Functions
  - Continuous Analog Output Functions
  - Asynchronous Analog Output Monitoring Functions
- Digital Input Function Group
  - Digital Input Configuration Functions
  - One-Shot Digital Input Functions
  - Continuous Digital Input Functions
  - Asynchronous Digital Input Monitoring Functions
- Digital Output Function Group
  - Digital Output Configuration Functions
  - One-Shot Digital Output Functions
  - Continuous Digital Output Functions
  - Asynchronous Digital Output Monitoring Functions
- Timer/Counter Function Group
- DIO Function Group
  - Digital Input/Output Configuration Functions
  - Dual-Interrupt System Setting Functions
  - Local Interrupt Setting Functions

- Emergency Shutdown Function Group
- Watchdog Timer Function Group
- Hot-system Reset Hold Function Group
- Calibration Function Group
- SSI Function Group
- PWM Function Group

2.1 General Configuration Function Group

These functions initialize and configure data acquisition cards.

Register_Card	Initializes the hardware and software states of a NuDAQ PCI-bus data acquisition card. This function must be called before any other DASK library functions.
Release_Card	Tells the DASK library that the registered card is not in use and can be released. This function makes room for a new card to register.
GetCardType	Gets the card type of the device with a specified card index.
GetCardIndexFromID	Gets the card type and the sequence number of the device with a specified card ID.
GetBaseAddr	Gets the I/O base addresses of the device with a specified card index.
GetLCRAddr	Gets the LCR base address (defined by the PCI controller on board) of the device with a specified card index.
SetInitPattern	Sets the state of the initial or safety-out pattern.
GetInitPattern	Gets the state of relays set by the onboard switches.
IdentifyLED_Control	Controls identification LED.

2.2 Actual Sampling Rate Function Group

GetActualRate	Returns the actual sampling rate the device will perform for the defined sampling rate value.
GetActualRate_9524	Obtains the actual sampling rate the PCI-9524 will perform according to the board type and the specified rate.

2.3 Analog Input Function Group

Analog Input Configuration Functions

AI_9111_Config Informs PCIS-DASK library of the trigger source and trigger mode selected for the analog input operation of PCI9-111. You must call this function before calling function to perform continuous analog input operation of PCI-9111.

AI_9112_Config Informs PCIS-DASK library of the trigger source selected for the analog input operation of PCI-9112. You must call this function before calling function to perform continuous analog input operation of PCI-9112.

AI_9113_Config Informs PCIS-DASK library of the trigger source selected for the analog input operation of PCI-9113. You must call this function before calling function to perform continuous analog input operation of PCI-9113.

AI_9114_Config Informs PCIS-DASK library of the trigger source selected for the analog input operation of PCI-9114. You must call this function before calling function to perform continuous analog input operation of PCI-9114.

AI_9116_Config Informs PCIS-DASK library of the trigger source, trigger mode, input mode, and conversion mode selected for the analog input operation of PCI-9116. You must call this function before calling function to perform continuous analog input operation of PCI-9116.

AI_9118_Config Informs PCIS-DASK library of the trigger source, trigger mode, input mode, and conversion mode selected for the ana-

log input operation of PCI9118. You must call this function before calling function to perform continuous analog input operation of PCI-9118.

AI_9221_Config Informs PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the analog input operation of PCI-9221. You must call this function before calling function to perform continuous analog input operation of PCI-9221.

AI_9222_Config Informs PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the analog input operation of PCI-9222. You must call this function before calling function to perform continuous analog input operation of PCI-9222.

AI_9223_Config Informs PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the analog input operation of PCI-9223. You must call this function before calling function to perform continuous analog input operation of PCI-9223.

AI_9524_Config Informs PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the analog input operation of PCI-9524. You must call this function before calling function to perform continuous analog input operation of PCI-9524.

AI_9812_Config Informs PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the analog input operation of PCI-9812. You must call this function before calling function to per-

form continuous analog input operation of PCI-9812.

AI_9116_CounterInterval Informs PCIS-DASK library of the scan interval value and sample interval value selected for the analog input operation of PCI-9116. You must call this function before calling function to perform continuous analog input operation of PCI-9116.

AI_9221_CounterInterval Informs PCIS-DASK library of the scan interval value and sample interval value selected for the analog input operation of PCI-9221. You must call this function before calling function to perform continuous analog input operation of PCI-9221.

AI_9222_CounterInterval Informs PCIS-DASK library of the scan interval value and sample interval value selected for the analog input operation of PCI-9222. You must call this function before calling function to perform continuous analog input operation of PCI-9222 with internal clock.

AI_9223_CounterInterval Informs PCIS-DASK library of the scan interval value and sample interval value selected for the analog input operation of PCI-9223. You must call this function before calling function to perform continuous analog input operation of PCI-9223 with internal clock.

AI_9524_PollConfig The function should be called if the AI transfer mode is set to polling mode by AI_9524_Config(). The function will start ADC with the set speed and range. You can use AI polling functions to get the acquired value.

AI_9524_SetDSP	The function should be called if the load cell AI operation will be performed. The function sets DSP configurations for AI load cell channels. Please refer PCI-9524 hardware manual for details.
AI_InitialMemoryAllocated	Gets the actual size of analog input memory that is available in the device driver.
AI_GetView	Gets the mapped buffer address of the analog input memory that is available in the device driver.
AI_SetTimeOut	Sets the Timeout period for Sync mode of continuous AI.

One-Shot Analog Input Functions

AI_ReadChannel	Performs a software triggered A/D conversion (analog input) on an analog input channel and returns the value converted (unscaled).
AI_ReadChannel32	Performs a software triggered A/D conversion (analog input) on an analog input channel and returns the 32-bit value converted (unscaled).
AI_VReadChannel	Performs a software triggered A/D conversion (analog input) on an analog input channel and returns the value scaled to a voltage in units of volts.
AI_ReadMultiChannels	Performs software triggered A/D conversions on the specified analog input channels.
AI_ScanReadChannels	Performs software triggered A/D conversions on the specified analog input channels.
AI_ScanReadChannels32	Performs software triggered A/D conversions on the specified analog input channels.

AI_VoltScale Converts the result from an AI_ReadChannel call to the actual input voltage.

AI_VoltScale32 Converts the result from an AI_ReadChannel32 call to the actual input voltage.

Continuous Analog Input Functions

AI_ContReadChannel Performs continuous A/D conversions on the specified analog input channel at a rate as close to the rate you specified.

AI_ContScanChannels Performs continuous A/D conversions on the specified continuous analog input channels at a rate as close to the rate you specified. This function is only available for those cards that support auto-scan functionality.

AI_ContReadMultiChannels
Performs continuous A/D conversions on the specified analog input channels at a rate as close to the rate you specified. This function is only available for those cards that support auto-scan functionality.

AI_ContReadChannelToFile
Performs continuous A/D conversions on the specified analog input channel at a rate as close to the rate you specified and saves the acquired data in a disk file.

AI_ContScanChannelsToFile
Performs continuous A/D conversions on the specified continuous analog input channels at a rate as close to the rate you specified and saves the acquired data in a disk file. This function is only

available for those cards that support auto-scan functionality.

AI_ContReadMultiChannelsToFile

Performs continuous A/D conversions on the specified analog input channels at a rate as close to the rate you specified and saves the acquired data in a disk file. This function is only available for those cards that support auto-scan functionality.

AI_ContVScale — Converts the values of an array of acquired data from an continuous A/D conversion call to the actual input voltages.

AI_ContStatus — Checks the current status of the continuous analog input operation.

AI_EventCallBack — Controls and notifies the user's application when a specified DAQ event occurs. The notification is performed through a user-specified callback function.

AI_ContBufferReset — Resets all the buffers set by function “AI_ContBufferSetup” for continuous analog input.

AI_ContBufferSetup — Sets up a specified buffer for continuous analog input.

Asynchronous Analog Input Monitoring Functions

AI_AsyncCheck — Checks the current status of the asynchronous analog input operation.

AI_AsyncClear — Stops the asynchronous analog input operation.

AI_AsyncDblBufferMode — Enables or disables double buffer data acquisition mode.

AI_AsyncDblBufferHalfReady

Checks whether the next half buffer of

data in circular buffer is ready for transfer during an asynchronous double-buffered analog input operation.

AI_AsyncDblBufferTransfer

Copies half of the data of circular buffer to user buffer. You can execute this function repeatedly to return sequential half buffers of the data.

AI_AsyncDblBufferOverrun

Checks or clears overrun status of the double-buffered analog input operation.

AI_AsyncDblBufferHandled

Notifies the PCIS-DASK that the ready buffer has been handled in user application.

AI_AsyncDblBufferToFile

Logs the data of the circular buffer to a disk file.

AI_AsyncReTrigNextReady

Checks whether the data associated to the next trigger signal is ready during an asynchronous re-triggered analog input operation.

2.4 Analog Output Function Group

Analog Output Configuration Functions

AO_6202_Config Inform the PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the PCI-6202 with card ID CardNumber. You must call this function before calling function to perform continuous analog output operation.

AO_6208A_Config Informs PCIS-DASK library of the current range selected for the analog output operation of PCI-6208A. You must call this function before calling function to perform current output operation.

AO_6308A_Config Informs PCIS-DASK library of the current range selected for the analog output operation of PCI-6308A. You must call this function before calling function to perform current output operation.

AO_6308V_Config Informs PCIS-DASK library of the polarity (unipolar or bipolar) that the output channel is configured for the analog output and the reference voltage value selected for the analog output channel(s) of PCI-6308V. You must call this function before calling function to perform current output operation.

AO_9111_Config Informs PCIS-DASK library of the polarity (unipolar or bipolar) that the output channel is configured for the analog output of PCI-9111. You must call this function before calling function to perform voltage output operation.

AO_9112_Config Informs PCIS-DASK library of the reference voltage value selected for the analog output channel(s) of PCI-9112. You

must call this function before calling function to perform voltage output operation.

AO_9222_Config Inform the PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the PCI-9222 with card ID CardNumber. You must call this function before calling function to perform continuous analog output operation.

AO_9223_Config Inform the PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the PCI-9223 with card ID CardNumber. You must call this function before calling function to perform continuous analog output operation.

AO_InitialMemoryAllocated
Gets the actual size of analog output memory that is available in the device driver.

AO_SetTimeOut Sets the Timeout period for Sync mode of continuous AO.

One-Shot Analog Output Functions

AO_WriteChannel	Writes a binary value to the specified analog output channel.
AO_VWriteChannel	Accepts a voltage value, scales it to the proper binary value and writes a binary value to the specified analog output channel.
AO_VoltScale	Scales a voltage to a binary value.
AO_SimuWriteChannel	Writes binary values to the specified analog output channels simultaneously.
AO_SimuVWriteChannel	Accepts voltage values, scales them to the proper binary values and writes binary values to the specified analog output channels simultaneously.

Continuous Analog Output Functions

AO_ContBufferReset	Reset all the buffers set by function AO_ContBufferSetup for continuous analog output.
AO_ContBufferSetup	Set up the buffer for continuous analog output operation.
AO_ContWriteChannel	Performs continuous D/A conversions on the specified analog output channel at a rate as close to the rate you specified.
AO_ContWriteMultiChannels	Performs continuous D/A conversions on the specified analog output channels at a rate as close to the rate you specified.
AO_EventCallBack	Controls and notifies the user's application when a specified DAQ event occurs.
AO_ContBufferCompose	Fills the data for a specified channel in the buffer for multi-channels of continuous analog output operation.
AO_ContStatus	Returns D/A status while performing continuous D/A conversions.

Asynchronous Analog Output Monitoring Functions

AO_AsyncCheck	Check the current status of the asynchronous analog output operation.
AO_AsyncClear	Stop the asynchronous analog output operation.
AO_AsyncDblBufferHalfReady	Checks whether the next half buffer is ready for new data during an asynchronous double-buffered analog output operation.
AO_AsyncDblBufferMode	Enables or disables double-buffered data acquisition mode.

2.5 Digital Input Function Group

Digital Input Configuration Functions

DI_7200_Config
Informs PCIS-DASK library of the trigger source and trigger properties selected for the digital input operation of PCI-7200. You must call this function before calling function to perform continuous digital input operation of PCI-7200.

DI_7300A_Config

DI_7300B_Config
Informs PCIS-DASK library of the trigger source and trigger properties selected for the digital input operation of PCI-7300A Rev.A or PCI-7300A Rev.B. You must call this function before calling function to perform continuous digital input operation of PCI-7300A Rev.A or PCI-7300A Rev.B.

DI_9222_Config
Inform the PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the PCI-9222 with card ID CardNumber.

DI_9223_Config
Inform the PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the PCI-9223 with card ID CardNumber.

DI_InitialMemoryAllocated
Gets the actual size of digital input DMA memory that is available in the device driver.

DI_GetView
Gets the mapped buffer address of the digital input memory that is available in the device driver.

DI_SetTimeOut
Sets the Timeout period for Sync mode of continuous DI.

One-Shot Digital Input Functions

DI_ReadLine	Reads the digital logic state of the specified digital line in the specified port.
DI_ReadPort	Reads digital data from the specified digital input port.

Continuous Digital Input Functions

DI_ContReadPort	Performs continuous digital input on the specified digital input port at a rate as close to the rate you specified.
DI_ContReadPortToFile	Performs continuous digital input on the specified digital input port at a rate as close to the rate you specified and saves the acquired data in a disk file.
DI_ContStatus	Checks the current status of the continuous digital input operation.
DI_EventCallBack	Controls and notifies the user's application when a specified DAQ event occurs. The notification is performed through a user-specified callback function.
DI_ContMultiBufferSetup	Set up the buffer for multi-buffered continuous digital input.
DI_ContMultiBufferStart	Starts the multi-buffered continuous digital input on the specified digital input port at a rate as close to the rate you specified.
DI_ContBufferReset	Reset all the buffers set by function DI_ContBufferSetup for continuous digital input.
DI_ContBufferSetup	Set up the buffer for continuous digital input operation.

Asynchronous Digital Input Monitoring Functions

DI_AsyncCheck Checks the current status of the asynchronous digital input operation.

DI_AsyncClear Stops the asynchronous digital input operation.

DI_AsyncDblBufferMode Enables or disables double buffer data acquisition mode.

DI_AsyncDblBufferHalfReady
Checks whether the next half buffer of data in circular buffer is ready for transfer during an asynchronous double-buffered digital input operation.

DI_AsyncDblBufferHandled
Notifies PCIS-DASK the ready buffer has been handled in user application.

DI_AsyncDblBufferToFile For double buffer mode of continuous DI, if the continuous DI function is DI_ContReadPortToFile, call this function to log the data of the circular buffer into a disk file.

DI_AsyncDblBufferTransfer
Copies half of the data of circular buffer to user buffer. You can execute this function repeatedly to return sequential half buffers of the data.

DI_AsyncMultiBufferNextReady
Checks whether the next buffer of data in circular buffer is ready for transfer during an asynchronous multi-buffered digital input operation.

DI_AsyncDblBufferOverrun
Checks or clears overrun status of the double-buffered digital input operation.

DI_AsyncMultiBuffersHandled

Notifies PCIS-DASK the ready buffers have been handled in user application.

DI_AsyncReTrigNextReady

Checks whether the data associated to the next trigger signal is ready during an asynchronous re-triggered digital input operation.

2.6 Digital Output Function Group

Digital Output Configuration Functions

DO_7200_Config Informs PCIS-DASK library of the trigger source and trigger properties selected for the digital input operation of PCI-7200. You must call this function before calling function to perform continuous digital output operation of PCI-7200.

DO_7300A_Config

DO_7300B_Config Informs PCIS-DASK library of the trigger source and trigger properties selected for the digital input operation of PCI-7300A Rev.A or PCI-7300A Rev.B. You must call this function before calling function to perform continuous digital output operation of PCI-7300A Rev.A or PCI-7300A Rev.B.

DO_9222_Config Inform the PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the PCI-9222 with card ID CardNumber.

DO_9223_Config Inform the PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the PCI-9223 with card ID CardNumber.

EDO_9111_Config Informs PCIS-DASK library of the mode of EDO channels of PCI-9111.

DO_InitialMemoryAllocated
Gets the actual size of digital output DMA memory that is available in the device driver.

DO_GetView Gets the mapped buffer address of the digital output memory that is available in the device driver.

DO_SetTimeOut	Sets the Timeout period for Sync mode of continuous DO.

One-Shot Digital Output Functions

DO_WriteLine	Sets the specified digital output line in the specified digital output port to the specified state. This function is only available for those cards that support digital output read-back functionality.
DO_WritePort	Writes digital data to the specified digital output port.
DO_SImuWritePort	Write the output digital data to the specified digital output port simultaneously.
DO_ReadLine	Reads the specified digital output line in the specified digital output port.
DO_ReadPort	Reads digital data from the specified digital output port.
DO_WriteExtTrigLine	Sets the digital output trigger line to the specified state. This function is only available for PCI-7200.

Continuous Digital Output Functions

DO_ContWritePort Performs continuous digital output on the specified digital output port at a rate as close to the rate you specified.

DO_ContStatus Checks the current status of the continuous digital output operation.

DO_EventCallBack Controls and notifies the user's application when a specified DAQ event occurs. The notification is performed through a user-specified callback function.

DO_PGStart Performs pattern generation operation.

DO_PGStop Stops pattern generation operation.

DO_ContMultiBufferSetup Set up the buffer for multi-buffered continuous digital output.

DO_ContMultiBufferStart Starts the multi-buffered continuous digital output on the specified digital output port at a rate as close to the rate you specified.

Asynchronous Digital Output Monitoring Functions

DO_ContBufferReset Reset all the buffers set by function DO_ContBufferSetup for continuous digital output.

DO_ContBufferSetup Set up the buffer for continuous digital output operation.

DO_AsyncCheck Checks the current status of the asynchronous digital output operation.

DO_AsyncClear Stops the asynchronous digital output operation.

DO_AsyncMultiBufferNextReady
Checks whether the next buffer is ready for new data during an asynchronous multi-buffered digital output operation.

2.7 Timer/Counter Function Group

Timer/Counter Functions

CTR_Setup	Configures the selected counter to operate in the specified mode.
CTR_Read	Reads the current contents of the selected counter.
CTR_Clear	Sets the output of the selected counter to the specified state.
CTR_Update	Writes a new initial count to the selected counter.
CTR_8554_ClkSrc_Config	Sets the counter clock source.
CTR_8554_CK1_Config	Sets the source of CK1.
CTR_8554_Debounce_Config	Sets the debounce clock.

General-Purpose Timer/Counter Functions

GCTR_Setup	Controls the general-purpose counter to operate in the specified mode.
GCTR_Read	Reads the current counter value of the general-purpose counter.
GCTR_Clear	Clears the general-purpose timer/counter control register and counter register.
GPTC_9524_PG_Config	This function sets the generated pulse number of GPTC pulse generator.
GPTC_Clear	Halts the specified general-purpose counter operation and reloads the initial value of the timer/counter.
GPTC_Control	Controls the selected counter/timer by software.
GPTC_EventSetup	Sets the configurations of the selected event of the counter/timer.

GPTC_EventCallBack	Controls and notifies the user's application when a specified GPTC event occurs.
GPTC_Read	Reads the counter value of the general-purpose counter without disturbing the counting process.
GPTC_Setup	Sets the configurations of the selected counter/timer.
GPTC_Status	Reads the latched GPTC status of the general-purpose counter/timer from GPTC status register.

2.8 Digital Input/Output Function Group

Digital Input/Output Configuration Functions

DIO_LineConfig

This function is only used by the Digital I/O cards whose I/O port can be set as input port or output port. This function informs PCIS-DASK library of the line direction selected for the digital input/output operation. You must call this function before calling functions to perform digital input/output operation.

DIO_LinesConfig

This function is only used by the Digital I/O cards whose I/O port can be set as input port or output port. This function informs PCIS-DASK library of the entire lines direction of the port selected for the digital input/output operation. You must call this function before calling functions to perform digital input/output operation.

DIO_PortConfig

This function is only used by the Digital I/O cards whose I/O port can be set as input port or output port. This function informs PCIS-DASK library of the port direction selected for the digital input/output operation. You must call this function before calling functions to perform digital input/output operation.

Dual-Interrupt System Setting Functions

DIO_SetDualInterrupt	Controls two interrupt sources of Dual Interrupt system.
DIO_SetCOSInterrupt	Sets the ports used for COS interrupt detection.
DIO_SetCOSInterrupt32	Sets the ports with 32-bit data width used for COS interrupt detection.
DIO_GetCOSLatchData	Get the DI data that latched in the COS Latch register while the Change-of-State (COS) interrupt occurred.
DIO_GetCOSLatchData32	Get the DI data with 32-bit data width that latched in the COS Latch register while the Change-of-State (COS) interrupt occurred.
DIO_INT_EventMessage	Controls and notifies the user's application when an interrupt event occurs. The notification is performed through a user-specified callback function or the Windows PostMessage API.
DIO_INT1_EventMessage	Controls the interrupt sources of INT1 of Dual Interrupt system and notifies the user's application when an interrupt event occurs. The notification is performed through a user-specified callback function or the Windows PostMessage API.
DIO_INT2_EventMessage	Controls the interrupt sources of INT2 of Dual Interrupt system and notifies the user's application when an interrupt event occurs. The notification is performed through a user-specified callback function or the Windows PostMessage API.

Local Interrupt Setting Functions

DIO_7300SetInterrupt Controls the interrupt sources (AUXDI and Timer2) of local Interrupt system of PCI-7300A/cPCI-7300A.

DIO_AUXDI_EventMessage Controls AUXDI Interrupt and notifies the user's application when an interrupt event occurs. The notification is performed through a user-specified callback function or the Windows PostMessage API.

DIO_T2_EventMessage Controls Timer2 Interrupt and notifies the user's application when an interrupt event occurs. The notification is performed through a user-specified callback function or the Windows PostMessage API.

2.9 Emergency Shutdown Function Group

EMGShutDownControl Controls emergency shutdown.

EMGShutDownStatus Returns the emergency shutdown condition.

2.10 Watchdog Timer Function Group

WDT_Control Control watchdog timer.

WDT_Reload Reload watchdog timer counter.

WDT_Setup Setup a watchdog timer.

WDT_Status Get the overflow status of a watchdog timer

2.11 Hot-system Reset Hold Function Group

HotResetoldStatus Read hot reset hold status.

HotResetHoldControl Controls hot-system reset DO hold function. Hold the current DO output value while your computer is hot reset if hot-reset-hold is enabled. Otherwise, the initial pattern is outputted.

2.12 Calibration Function Group

PCI_DB_Auto_Calibration_ALL
Calibrates the specified device.

PCI_EEPROM_CAL_Constant_Update
Saves new calibration constants to the specified EEPROM bank.

PCI_Load_CAL_Data Loads calibration constants from the specified EEPROM bank.

PCI9524_Acquire_AD_CalConst
Obtains the AD calibration constants of PCI-9524.

PCI9524_Acquire_DA_CalConst
Obtains the DA calibration constants of PCI-9524.

2.13 SSI Function Group

SSI_SourceClear	Disconnects all of the device signals from the SSI bus trigger lines.
SSI_SourceConn	Connects a device to the specified SSI bus trigger line.
SSI_SourceDisConn	Disconnects a device signal from the specified SSI bus trigger line.

2.14 PWM Function Group

PWM_Output	Start the PWM output.
PWM_Stop	Stop the PWM output.

3 Building Applications

3.1 Contiguous Memory Allocation

The PCIS-DASK features continuous data transfer functions that input or output blocks of data to or from an installed NuDAQ PCI device. To prevent reduced data transfer performance caused by memory fragment, the PCIS-DASK allocates physically contiguous buffers in device driver at system startup time (Windows® 98) or when system boots (Windows® NT/2000/XP/2003 and Linux).

The PCIS-DASK comes with the **PciUtil** applications to set or modify the sizes of contiguous memory allocated in the driver for continuous analog input, analog output, digital input, and digital output. Device drivers allocates these memory sizes. The size of initially allocated memory is the maximum memory size that continuous data transfer can be performed. Refer to the NuDAQ Registry/Configuration utility section for details.

For input operations, the specified data count are transferred to the driver buffer while the PCIS-DASK copies the data from the driver buffer (kernel level) to a user buffer (user level). For output operations, PCIS-DASK copies the data from a user buffer (driver level) to the driver buffer (kernel level) and transfers outgoing data from the driver buffer to the NuDAQ PCI device.

When performing only polling I/O, the initial allocated memory is not needed and you may use the NuDAQ Registry/Configuration utility to set the buffer size to 0.

3.2 Application Building Fundamentals in Windows

The following sections provide fundamental instructions when using PCIS-DASK to build application in Windows® NT/98/2000/ XP/Server 2003 operating environment.

Using Microsoft® Visual C®/C++®

Follow these steps to create a data acquisition application using PCIS-DASK and Microsoft Visual C/C++.

1. Launch the Microsoft Visual C/C++ application.
2. Open a new or existing project that you want to apply the PCIS-DASK.
3. Include header file DASK.H in the C/C++ source files that call PCIS-DASK functions. DASK.H contains all the function declarations and constants that can be used to develop data acquisition applications. Incorporate the following statement in the code to include the header file.

   ```
   #include "DASK.H"
   ```

4. After setting the appropriate compile and link options, build the application by selecting the Build command from Build menu (Visual C/C++ 4.0). Remember to link PCIS-DASK's import library, PCIS-DASK.LIB.

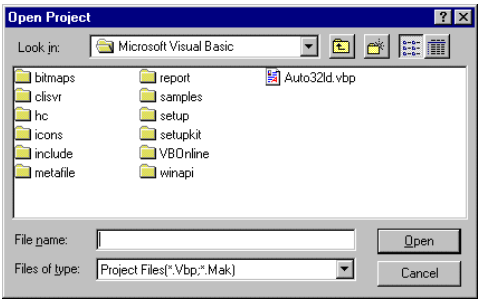
Using Microsoft® Visual Basic®

Follow the steps in the succeeding sections to create a data acquisition application using PCIS-DASK and Visual Basic.

Open a project

Do one of the following to open a new or existing project:

1. Open a new project by selecting the New Project command from the File menu. To open an existing project, select the Open Project command from the File menu to display the Open Project dialog box.



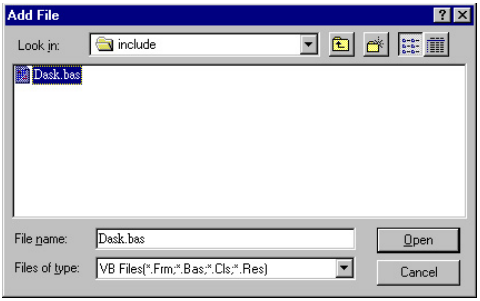
2. Locate the existing project, then double-click on the project file name to load.

Add the file

You must add the file **DASK.BAS** to the project, if the file is not yet included. This file contains all the procedure declarations and con-

stants that can be used to develop the data acquisition application. To add the file:

1. Select Add File from the File menu. The Add File window appears, displaying a list of files in the current directory.



2. Double-click on the DASK.BAS file. If the file is not on the list, make sure the list is displaying files from the correct directory. By default, the DASK.BAS file is installed at C:\ADLink\PCIS-DASK\INCLUDE.

Design the interface

To design the interface for the application, place all the interface elements such as command buttons, list boxes, and text boxes on the Visual Basic form. These standard controls are available from the Visual Basic Toolbox.

To place a control on the form, select the desired control from the Toolbox, then draw it on the form. You may also double-click on the control icon from the Toolbox to place it on the form.

Set the interface controls

To view the property list, click the desired control, then choose the Properties command from the View menu, or press F4. You may also click on the Properties button  from the toolbar.

Write the event code

The event code defines the required action to be performed when an event occurs. To write the event code, double-click on the control or form to view the code module, then add the event code. You can also call the functions declared in the DASK.BAS file to perform data acquisition operations.

Run the application

Do one of the following to run the application:

- Choose **Start** from the **Run** menu
- Click the Start icon  from the toolbar
- Press <F5>

Distribute the application

After completing the project, save the application as an executable (.EXE) file using the **Make EXE File** command from the File menu. The application, after being transformed into an executable file, is now ready for distribution.

You must include the PCIS-DASK's DLL and driver files when the application is distributed. Refer to Chapter 5: Continuous Data Transfer for the details.

3.3 Application Building Fundamentals in Linux

The following sections provide fundamental instructions when using PCIS-DASK to build application in Linux. To create a data acquisition application using PCIS-DASK/X and GNU C/C++, follow these steps:

Edit the source files

Include the header file **dask.h** in the C/C++ source files that call PCIS-DASK/X functions. The dask.h has all the function declarations and constants that you can use to develop your data acquisition application. Add this statement in your code to include the header file.

```
#include "dask.h"
```

Build your application

Using the appropriate C/C++ compiler (gcc or cc) to compile the program. You should add **-lpci_dask** option to link **libpci_dask.so** library. For multi-threaded applications, the **-lpthread** string is required. For example:

```
gcc -o testai testai.c -lpci_dask
```

3.4 Application Building Fundamentals Using .NET

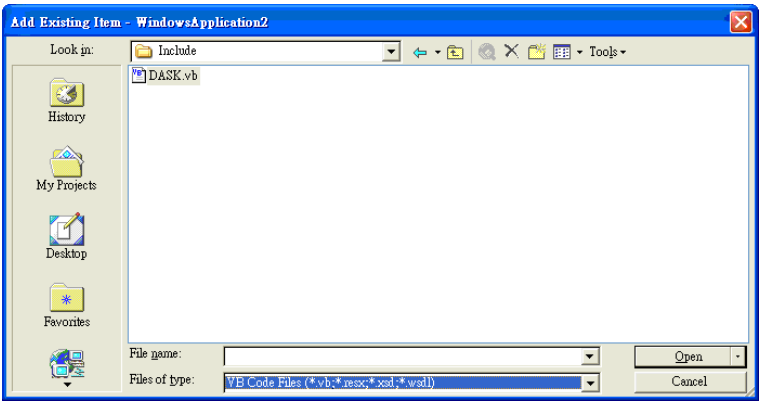
The following sections provide fundamental instructions when using PCIS-DASK to build application in Linux. To create a data acquisition application using PCIS-DASK/X and GNU C/C++, follow these steps:

Using Microsoft VB.net.

To create a data acquisition application using PCIS-DASK and VB.net, follow these steps after entering VB.net:

1. Open a new or existing project.
2. Add the file **DASK.vb** to the project, if the file is not yet included. This file contains all the procedure declarations and constants that can be used to develop the data acquisition application. To add the file:

▶ Select Add File from the File menu. The Add Existing Item window appears, displaying a list of files in the current directory.



▶ Double-click on the DASK.vb file. If the file is not on the list, make sure the list is displaying files from the correct direc-

tory. By default, the DASK.vb file is installed at C:\ADLINK\PCIS-DASK\INCLUDE.

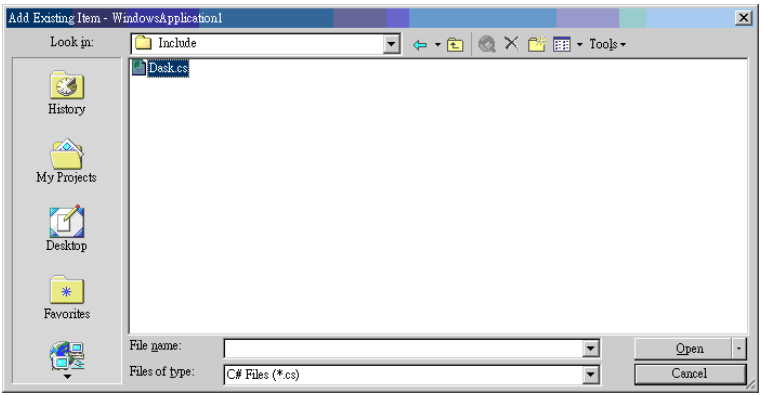
3. Develop the application. You can call the functions that are declared in the file Dask.vb to perform data acquisition operations.
4. Run your application. Do one of the following to run the application:
   - ▷ Choose **Start** from the **Run** menu
   - ▷ Click the Start icon  from the toolbar
   - ▷ Press <F5>
5. Distribute the application. After completing the project, save the application as an executable (.EXE) file using the Make EXE File command from the File menu. The application, after being transformed into an executable file, is now ready for distribution. You must include the PCIS-DASK's DLL and driver files when the application is distributed. Refer to **Chapter 8: Distribution of Applications** for the details.

Using Microsoft C#

To create a data acquisition application using PCIS-DASK and C#, follow these steps after entering C#:

1. Open a new or existing project.
2. Add the file **DASK.cs** to the project, if the file is not yet included. This file contains all the procedure declarations and constants that can be used to develop the data acquisition application. To add the file:

   ▷ Select Add File from the File menu. The Add Existing Item window appears, displaying a list of files in the current directory.From the Project menu, select the Add Existing Item command. The Add Existing Item window appears, displaying a list of files in the current directory.



▶ Double-click on the DASK.cs file. If the file is not on the list, make sure the list is displaying files from the correct direc-

tory. By default, the DASK.cs file is installed at C:\ADLINK\PCIS-DASK\INCLUDE.

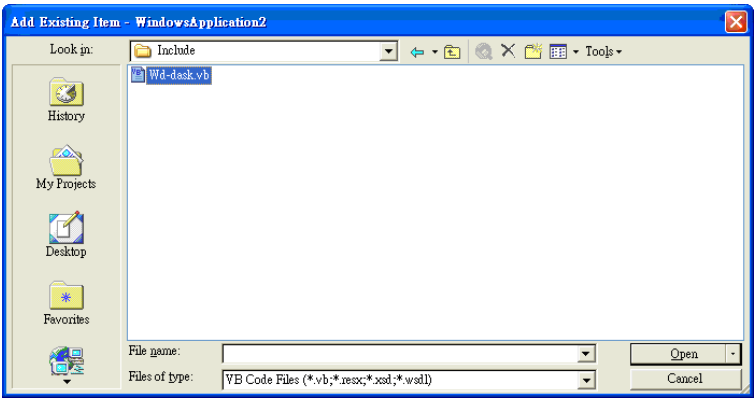
3. Develop the application. You can call the functions that are declared in the file Dask.cs to perform data acquisition operations.
4. Run your application. Do one of the following to run the application:
   - ▷ Choose **Start** from the **Run** menu
   - ▷ Click the Start icon  from the toolbar
   - ▷ Press <F5>
5. Distribute the application. After completing the project, save the application as an executable (.EXE) file using the Make EXE File command from the File menu. The application, after being transformed into an executable file, is now ready for distribution. You must include the PCIS-DASK's DLL and driver files when the application is distributed. Refer to **Chapter 8: Distribution of Applications** for the details.

Creating Windows® PCIS-DASK Application Using Microsoft VB.net.

To create a data acquisition application using DASK and C#, follow these steps after entering VB.net:

1. Open a new or existing project.
2. Add the file **DASK.vb** to the project, if the file is not yet included. This file contains all the procedure declarations and constants that can be used to develop the data acquisition application. To add the file:

   ▷ Select Add File from the File menu. The Add Existing Item window appears, displaying a list of files in the current directory.From the Project menu, select the Add Existing Item command. The Add Existing Item window appears, displaying a list of files in the current directory.



▶ Double-click on the DASK.vb file. If the file is not on the list, make sure the list is displaying files from the correct directory. By default, the DASK.vb file is installed at C:\ADLINK\PCIS-DASK\INCLUDE.

3. Develop the application. You can call the functions that are declared in the file DASK.vb to perform data acquisition operations.

4. Run your application. Do one of the following to run the application:
   - ▷ Choose **Start** from the **Run** menu
   - ▷ Click the Start icon  from the toolbar
   - ▷ Press <F5>
5. Distribute the application. After completing the project, save the application as an executable (.EXE) file using the Make EXE File command from the File menu. The application, after being transformed into an executable file, is now ready for distribution. You must include the PCIS-DASK's DLL and driver files when the application is distributed. Refer to **Chapter 8: Distribution of Applications** for the details.

Using Callback Functions in a VB.net Application with PCIS-DASK

To use callback functions in a VB.net application with PCIS-DASK, follow these steps after creating a Windows® 2000/XP PCIS-DASK application using VB.net:

1. Create a callback function. For example:

```
Sub CallBack()
            //Add the VB.Net function you like.
        End Sub
```

2. Set the callback function. For example:

```
AI_EventCallBack(dev, 1, DBEvent, AddressOf
     CallBack);
```

Using Callback Functions in a C# Application with PCIS-DASK

To use callback functions in a c# Application with PCIS-DASK, follow these steps after creating a Windows® 2000/XP PCIS-DASK application using C#:

1. Create a callback function. For example:

```
private static void CallBack()
{
    //Add the C# function you like.
}
```

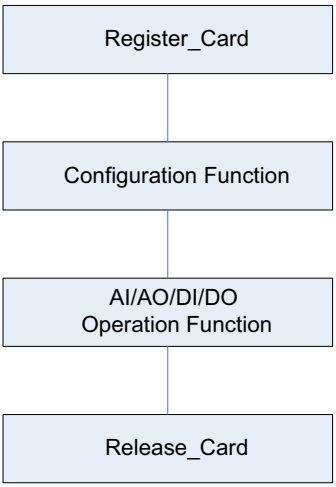
2. Set the callback function. For example:

```
CallbackDelegate del = new Callback Delegate(
     CallBack );
DASK.AI_EventCallBack((ushort)card, 1,
     DASK.DBEvent, del);
```

4 Application Hints

This chapter provides the programming schemes showing the function flow of that PCIS-DASK performs analog I/O and digital I/O.

The figure below shows the basic building blocks of a PCIS-DASK application. However, except using Register_Card at the beginning and Release_Card at the end, depending on the specific devices and applications you have, the PCIS-DASK functions comprising each building block vary.



The programming schemes for analog input/output and digital input/output are described individually in the following sections.

4.1 Analog Input

PCIS-DASK provides two kinds of analog input operation: nonbuffered single-point analog input readings and buffered continuous analog input operation.

The non-buffered single-point AI uses software polling method to read data from the device.

The buffered continuous analog input uses interrupt transfer or DMA transfer method to transfer data from device to user's buffer. The maximum number of count in one transfer depends on the size of initially allocated memory for analog input in the driver. The driver allocates the memory at system boot (in Windows® NT) or Windows startup time (in Windows® 98). It is recommended that the AI_InitialMemoryAllocated function be used to get the size of initially allocated memory before performing continuous AI operation.

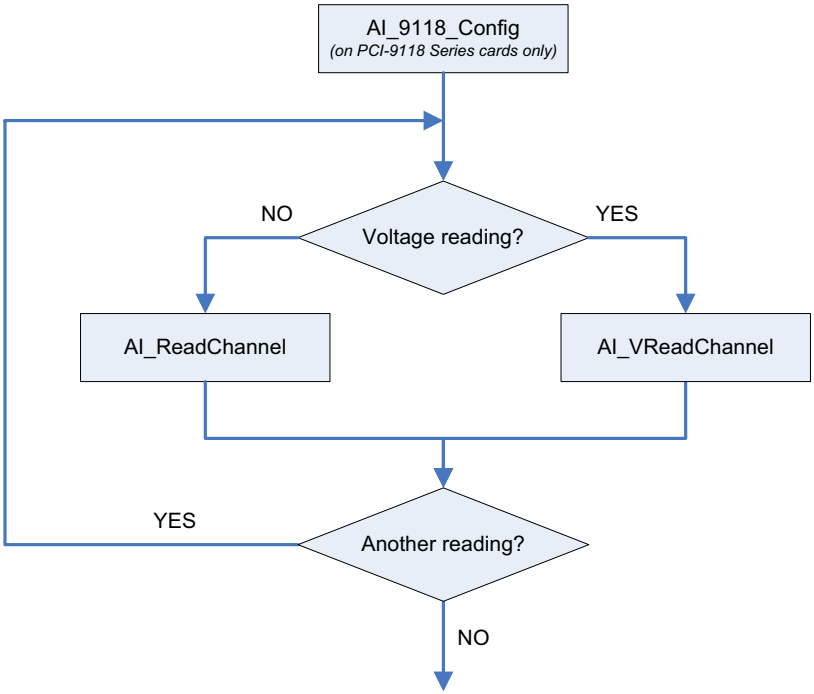
The buffered continuous analog input includes:

- synchronous continuous AI
- non-triggered non-double-buffered asynchronous continuous AI
- non-triggered double-buffered asynchronous continuous AI
- triggered non-double-buffered asynchronous continuous AI
- triggered double-buffered asynchronous continuous AI

These are described in section to section . For special consideration and performance issues for the buffered continuous analog input, refer to **Chapter 5: Continuous Data Transfer**.

One-Shot Analog Input

This section describes the function flow typical of non-buffered single-point analog input readings. While performing one-shot AI operation, most cards (except PCI-9118 Series cards) doesn't need to include the AI configuration step at the beginning of the application.

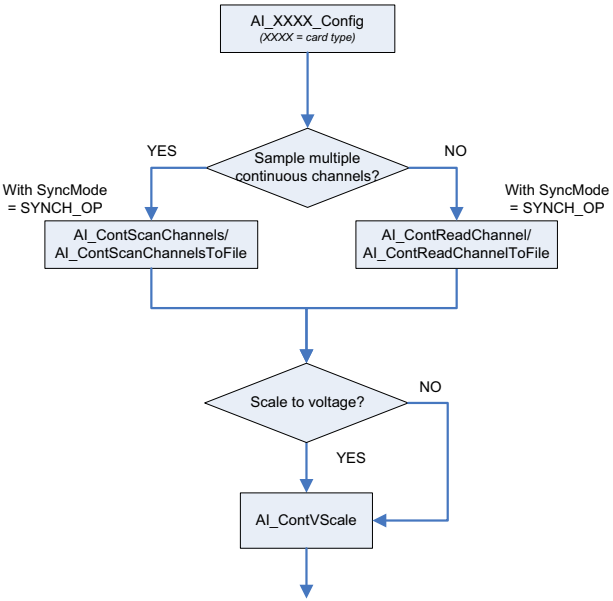


Example code fragment

```
card = Register_Card(PCI_9118, card_number);
…
AI_9118_Config(card,Input_Signal|Input_Mode,0,0,0);
AI_ReadChannel(card, channelNo, range,
   &analog_input[i]);
…
Release_Card(card);
```

Synchronous Continuous Analog Input

This section describes the function flow typical of synchronous analog input operation. While performing continuous AI operation, the AI configuration function has to be called at the beginning of your application. In addition, for synchronous AI, the SyncMode argument in continuous AI functions has to be set to SYNCH_OP.

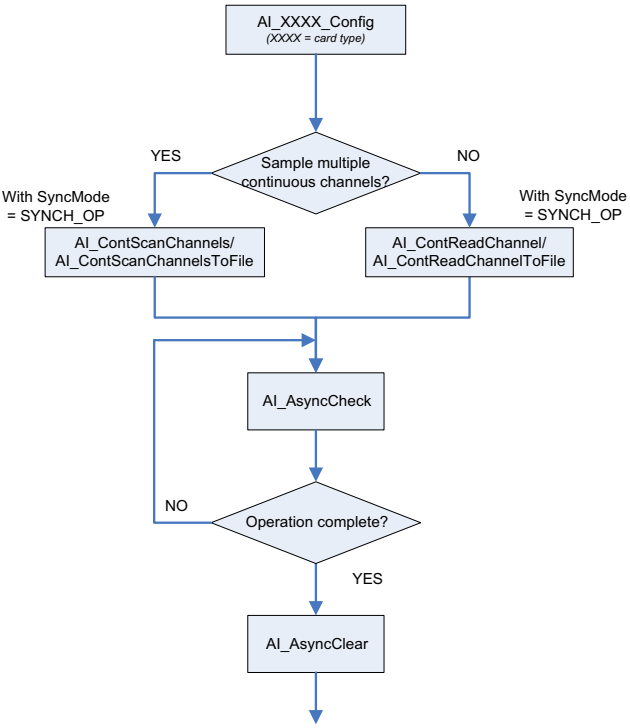


Example code fragment

```
card = Register_Card(PCI_9112, card_number);
…
AI_9112_Config(card,TRIG_INT_PACER);
AI_ContScanChannels (card, channel, range, ai_buf,
   data_size, (F64)sample_rate, SYNCH_OP); or
AI_ContReadChannel(card, channel, range, ai_buf,
   data_size, (F64)sample_rate, SYNCH_OP)
…
Release_Card(card);
```

Non-Trigger Non-double-buffered Asynchronous Continuous Analog Input

This section describes the function flow typical of non-trigger, non-double-buffered asynchronous analog input operation. While performing continuous AI operation, the AI configuration function has to be called at the beginning of your application. In addition, for asynchronous AI, the SyncMode argument in continuous AI functions has to be set to ASYNCH_OP.

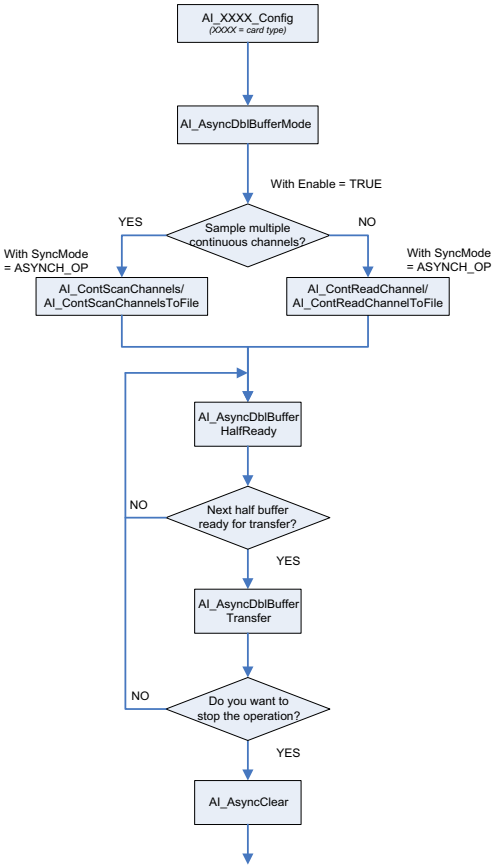


Example code fragment

```
card = Register_Card(PCI_9112, card_number);
…
AI_9112_Config(card,TRIG_INT_PACER);
AI_AsyncDblBufferMode (card, 0); //non-double-buffer
   AI
AI_ContScanChannels (card, channel, range, ai_buf,
   data_size, (F64)sample_rate, ASYNCH_OP); or
AI_ContReadChannel(card, channel, range, ai_buf,
   data_size, (F64)sample_rate, ASYNCH_OP)
          do {
       AI_AsyncCheck(card, &bStopped, &count);
              } while (!bStopped);
AI_AsyncClear(card, &count);
…
Release_Card(card);
```

Non-Trigger Double-buffered Asynchronous Continuous Analog Input

This section describes the function flow typical of non-trigger, double-buffered asynchronous analog input operation. While performing continuous AI operation, the AI configuration function has to be called at the beginning of your application. For asynchronous AI, The SyncMode argument in continuous AI functions has to be set to ASYNCH_OP. In addition, double-buffered AI operation is enabled by setting Enable argument of AI_AsyncDblBufferMode function to 1. For more information on double buffer mode, refer to section 5.2.

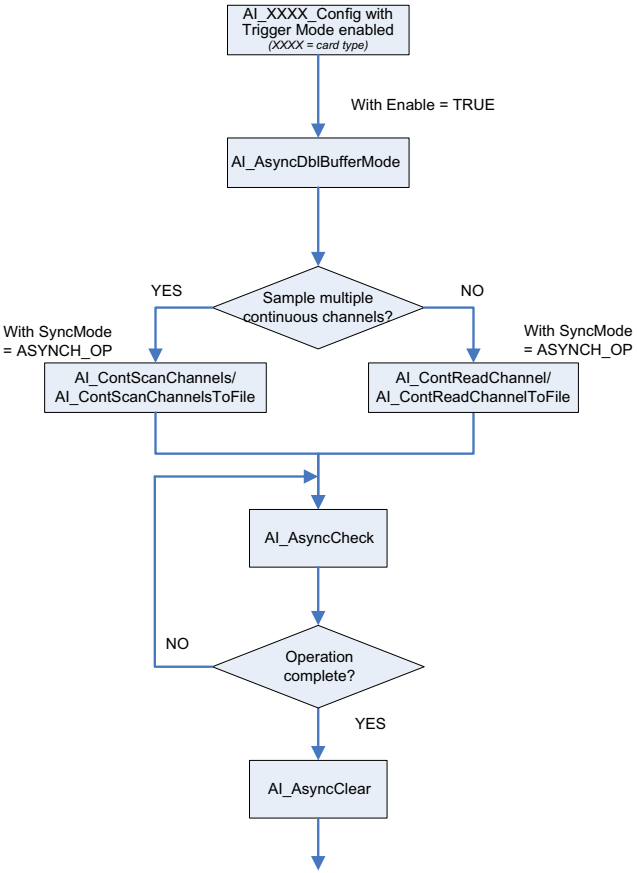


Example code fragment

```
card = Register_Card(PCI_9112, card_number);
…
AI_9112_Config(card,TRIG_INT_PACER);
AI_AsyncDblBufferMode (card, 1); // Double-buffered
AI_ContScanChannels (card, channel, range, ai_buf,
   data_size, (F64)sample_rate, ASYNCH_OP); or
AI_ContReadChannel(card, channel, range, ai_buf,
   data_size, (F64)sample_rate, ASYNCH_OP)
do {
      do {
         AI_AsyncDblBufferHalfReady(card, &HalfRead
   &fstop);
      } while (!HalfReady);

      AI_AsyncDblBufferTransfer(card, ai_buf);
     …
} while (!clear_op);

AI_AsyncClear(card, &count);
…
Release_Card(card);
```

Trigger Mode Non-double-buffered Asynchronous Continuous Analog Input

This section describes the function flow typical of trigger mode double-buffered asynchronous analog input operation. A trigger is an event that occurs based on a specified set of conditions. An interrupt mode or DMA-mode Analog input operation can use a trigger to determinate when acquisition stop. The trigger mode data acquisition programming is almost the same as the non-trigger mode asynchronous analog input programming. When using PCIS-DASK to perform trigger mode data acquisition, the SyncMode of continuous AI should be set to ASYNCH_OP.

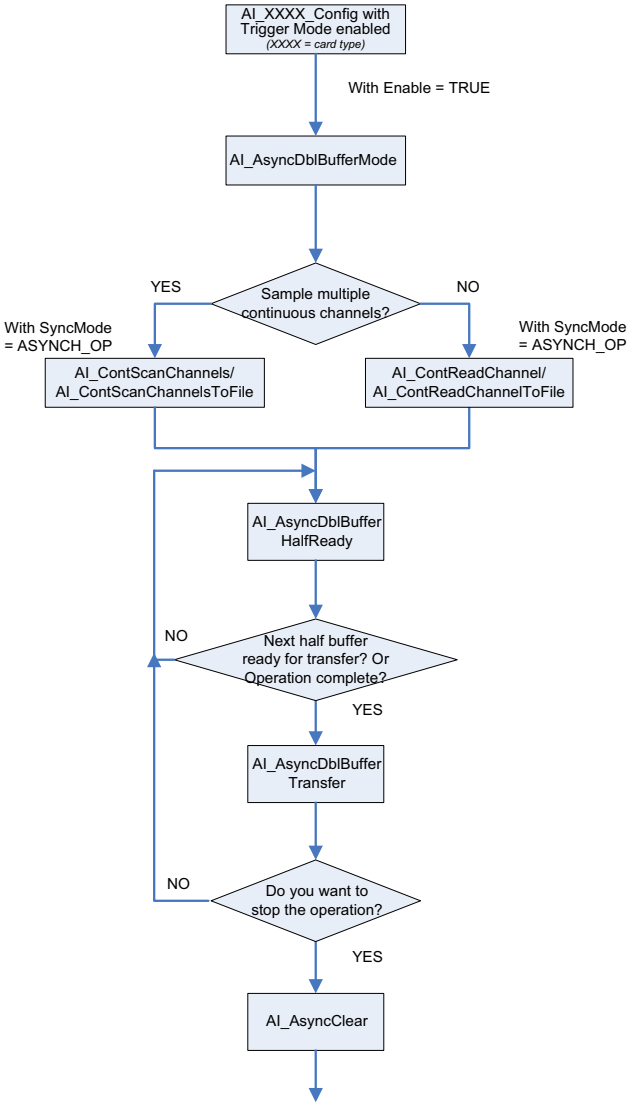


Example code fragment

```
card = Register_Card(PCI_9118, card_number);
…
AI_9118_Config(card,
   P9118_AI_BiPolar|P9118_AI_SingEnded,
P9118_AI_DtrgPositive|P9118_AI_EtrgPositive|
P9118_AI_AboutTrgEn, 0, postCount)
AI_AsyncDblBufferMode (card, 0); //non-double-buffer
   AI
AI_ContScanChannels (card, channel, range, ai_buf,
   data_size, (F64)sample_rate, ASYNCH_OP); or
AI_ContReadChannel(card, channel, range, ai_buf,
   data_size, (F64)sample_rate, ASYNCH_OP)
         do {
      AI_AsyncCheck(card, &bStopped, &count);
            } while (!bStopped);

AI_AsyncClear(card, &count);
                …
Release_Card(card);
```

Trigger Mode Double-buffered Asynchronous Continuous Analog Input

This section describes the function flow typical of trigger mode double-buffered asynchronous analog input operation. A trigger is an event that occurs based on a specified set of conditions. An interrupt mode or DMA-mode Analog input operation can use a trigger to determinate when acquisition stop. The trigger mode data acquisition programming is almost the same as the non-trigger mode asynchronous analog input programming. When using PCIS-DASK to perform trigger mode data acquisition, the SyncMode of continuous AI should be set to ASYNCH_OP. In addition, double-buffered AI operation is enabled by setting Enable argument of AI_AsyncDblBufferMode function to 1. For more information on double buffer mode, refer to section 5.2 for the details.


AI_XXXX_Config with Trigger Mode enabled
(XXXX = card type)
With Enable = TRUE
AI_AsyncDblBufferMode
YES
Sample multiple continuous channels?
NO
With SyncMode = ASYNCH_OP
With SyncMode = ASYNCH_OP
AI_ContScanChannels/ AI_ContScanChannelsToFile
AI_ContReadChannel/ AI_ContReadChannelToFile
AI_AsyncDblBuffer HalfReady
NO
Next half buffer ready for transfer? Or Operation complete?
YES
AI_AsyncDblBuffer Transfer
NO
Do you want to stop the operation?
YES
AI_AsyncClear

Example code fragment

```
card = Register_Card(PCI_9118, card_number);
…
AI_9118_Config(card,P9118_AI_BiPolar|P9118_AI_SingE
   d,
P9118_AI_DtrgPositive|P9118_AI_EtrgPositive|
P9118_AI_AboutTrgEn,0,postCount)
AI_AsyncDblBufferMode (card, 1);  Double-buffered A
AI_ContScanChannels (card, channel, range, ai_buf,
   data_size, (F64)sample_rate, ASYNCH_OP); or
AI_ContReadChannel(card, channel, range, ai_buf,
   data_size, (F64)sample_rate, ASYNCH_OP)
do {
      do {
         AI_AsyncDblBufferHalfReady(card, &HalfRead
   &fstop);
      } while (!HalfReady && !fstop);

      AI_AsyncDblBufferTransfer(card, ai_buf);
     …
} while (!clear_op && !fstop);

AI_AsyncClear(card, &count);
AI_AsyncDblBufferTransfer(card, ai_buf);
…
Release_Card(card);
```

4.2 Analog Output Programming Hints

The PCIS-DASK provides two kinds of analog output operations: non-buffered single-point analog output operation and buffered continuous analog output operation.

The non-buffered single-point AO uses software polling method to write data to the device. The programming scheme for this kind of AO operation is described in section 4.2.1.

The buffered continuous AO uses a DMA transfer method to transfer data from user's buffer to device. The maximum number of count in one transfer depends on the size of initially allocated memory for analog output in the driver. The driver allocates the memory at Windows startup. We recommend that applications use AO_InitialMemoryAllocated function to get the size of initially allocated memory before beginning to perform continuous AO operations.

Single-point Analog Output Programming Scheme

This section described the function flow typical of non-buffered single-point digital output operation. While performing one-shot AO operation, the cards whose I/O port can be set as input or out put port need to include port configuration function at the beginning of your application.

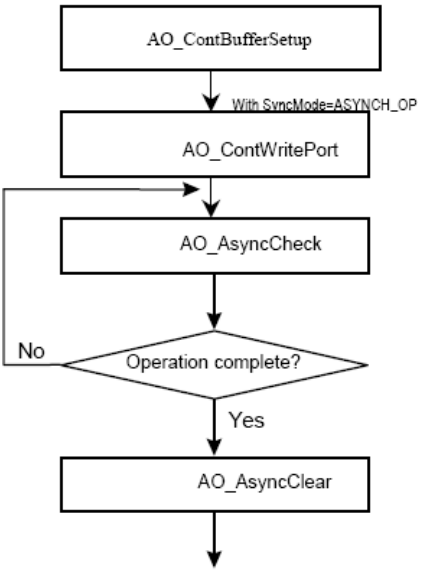
Example code fragment

```
card = _Register_Card(PCI_6202, card_number);
…
AO_WriteChannel(card, chan, out_value);
…
Release_Card(card);
```

Continuous Analog Output (with initial default settings) Programming Scheme

This section described the function flow typical of synchronous analog output operation performed by the device with initial default configuration. While performing continuous AO operation, the AO configuration function has to be called at the beginning of your application. In addition, the SyncMode argument in continuous AO functions has to be set as ASYNCH_OP.

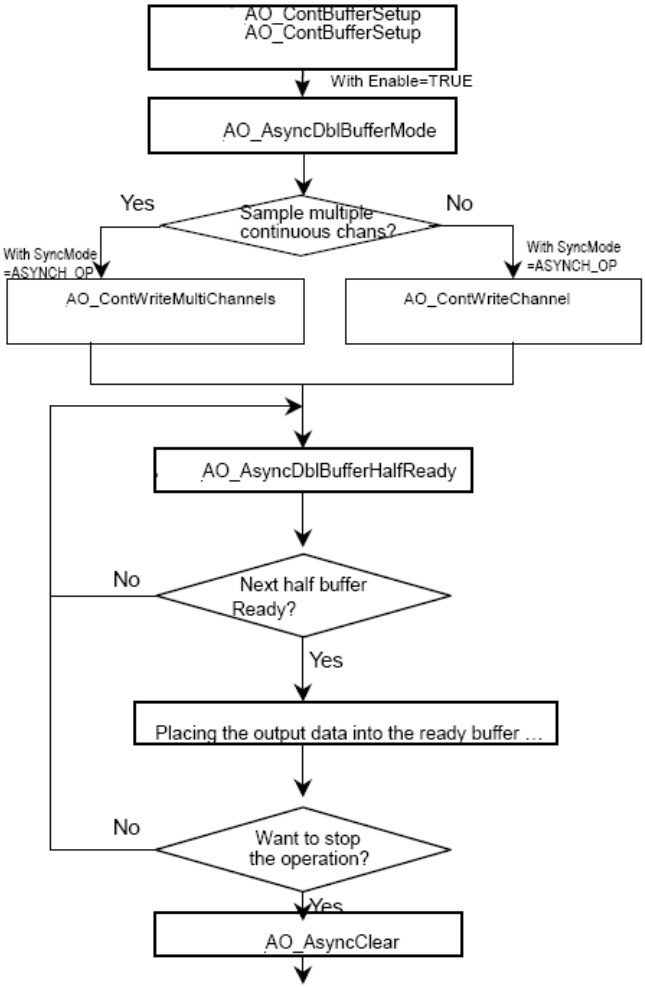
1. Non-double-buffered Asynchronous Continuous Analog Output Programming Scheme



Example code fragment

```
card = Register_Card(PCI_6202, card_number);
…
AO_ContBufferSetup (card, ao_buf, data_size, &DaId)
AO_ContWriteChannel(card, 0, DaId, data_size,
   iteration, samp_intrv, samp_intrv, ASYNCH_OP);;
do {
AO_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);
AO_AsyncClear(card, &count, mode);
…
Release_Card(card);
```

2. Double-buffered Asynchronous Continuous Analog Output Programming Scheme



Example code fragment

```
card = Register_Card(PCI_6202, card_number);
AO_ContBufferSetup (card, ao_buf, data_size, &DaId)
AO_ContBufferSetup (card, ao_buf2, data_size, &DaId
AO_AsyncDblBufferMode (card, 1);
AO_ContWriteChannel(card, 0, DaId, data_size, 0,
   samp_intrv, samp_intrv, ASYNCH_OP);;
do {
do {
AO_AsyncDblBufferHalfReady(card, &HalfReady);
} while (!HalfReady);
// Placing the output data into the ready buffer …
…
} while (!clear_op);
AO_AsyncClear(card, &count, mode);
…
Release_Card(card);
```

4.3 Digital Input Programming Hints

The PCIS-DASK provides two types of digital input operation: non-buffered single-point digital input operation and buffered continuous digital input operation.

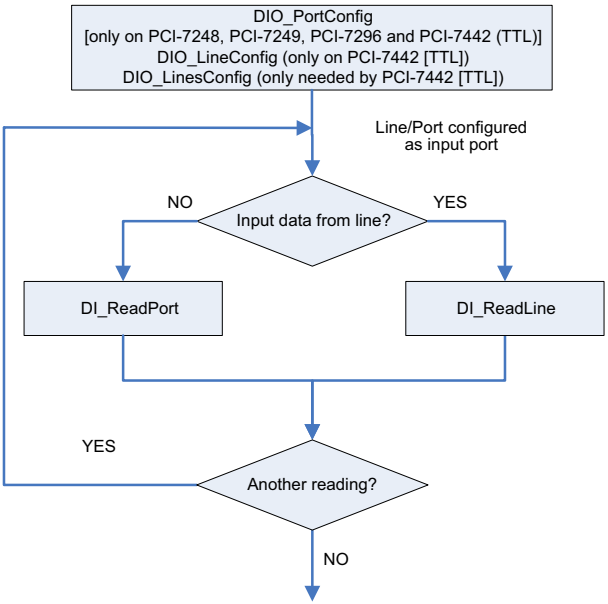
The non-buffered single-point DI uses software polling method to read data from the device. The programming scheme for this kind of DI operation is described in section .

The buffered continuous DI uses DMA transfer method to transfer data from device to user's buffer. The maximum number of count in one transfer depends on the size of initially allocated memory for digital input in the driver. The driver allocates the memory at system boot (in Windows® NT) or during Windows startup (in Windows® 98). It is recommended that the applications use the DI_InitialMemoryAllocated function to get the size of initially allocated memory before performing continuous DI operation.

The buffered continuous analog input includes synchronous continuous DI, non-double-buffered asynchronous continuous DI and double-buffered asynchronous continuous DI. These are described in section to section section. For special consideration and performance issues for the buffered continuous analog input, refer to **Chapter 5: Continuous Data Transfer**.

One-Shot Digital Input

This section describes the function flow typical of non-buffered single-point digital input readings. While performing one-shot DI operation, devices whose I/O port can be set as input or output port (PCI-7248, PCI-7296, and PCI-7442) need to include port configuration function at the beginning of the application.



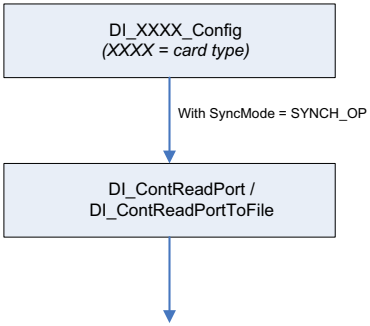
Example code fragments:

```
card = Register_Card(PCI_7248, card_number);
//port configured
DIO_PortConfig(card ,Channel_P1A, INPUT_PORT);
DIO_PortConfig(card, Channel_P1B, INPUT_PORT);
DIO_PortConfig(card, Channel_P1CL, INPUT_PORT);
DIO_PortConfig(card, Channel_P1CH, INPUT_PORT);
//DI operation
DI_ReadPort(card, Channel_P1A, &inputA);
…
Release_Card(card);
```

```
card = Register_Card(PCI_7442, card_number);
//line configured
DIO_LineConfig(card ,P7442_TTL0, 0, INPUT_LINE);
//DI operation
DI_ReadLinet(card, P7442_TTL0, 0, &inDataLine0);
…
Release_Card(card);
```

Synchronous Continuous Digital Input

This section describes the function flow typical of synchronous digital input operation. While performing continuous DI operation, the DI configuration function has to be called at the beginning of the application. For synchronous DI, the SyncMode argument in continuous DI functions has to be set to SYNCH_OP.

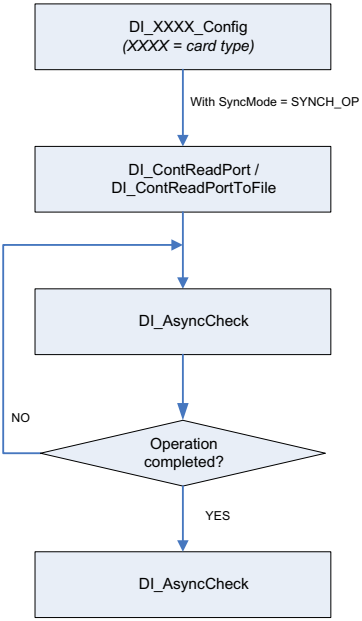


Example code fragment:

```
card = Register_Card(PCI_7200, card_number);
…
DI_7200_Config(card,TRIG_INT_PACER, DI_NOWAITING,
   DI_TRIG_FALLING, IREQ_FALLING);
DI_AsyncDblBufferMode (card, 0); //non-double-buffer
   mode
DI_ContReadPort(card, 0, pMem, data_size,
   (F64)sample_rate, SYNCH_OP)
…
Release_Card(card);
```

Non-double-buffered Asynchronous Continuous Digital Input

This section describes the function flow typical of non-double-buffered asynchronous digital input operation. While performing continuous DI operation, the DI configuration function has to be called at the beginning of the application. For asynchronous DI operation, the SyncMode argument in continuous DI functions has to be set to ASYNCH_OP.

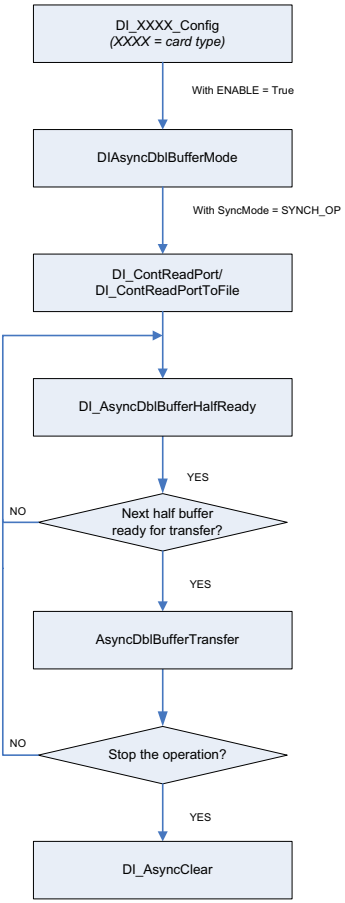


Example code fragment:

```
card = Register_Card(PCI_7200, card_number);
…
DI_7200_Config(card,TRIG_INT_PACER, DI_NOWAITING,
   DI_TRIG_FALLING, IREQ_FALLING);
DI_AsyncDblBufferMode (card, 0); // non-double-
   buffered mode
DI_ContReadPort(card, 0, pMem, data_size,
   (F64)sample_rate, ASYNCH_OP)
          do {
       DI_AsyncCheck(card, &bStopped, &count);
             } while (!bStopped);

DI_AsyncClear(card, &count);
                …
Release_Card(card);
```

Double-buffered Asynchronous Continuous Digital Input

This section describes the function flow typical of double-buffered asynchronous digital input operation. While performing continuous DI operation, the DI configuration function has to be called at the beginning of the application. For asynchronous DI, the SyncMode argument in continuous DI functions has to be set to ASYNCH_OP. In addition, double-buffered DI operation is enabled by setting Enable argument of DI_AsyncDblBufferMode function to 1. For more information on double buffer mode, refer to section 5.2.

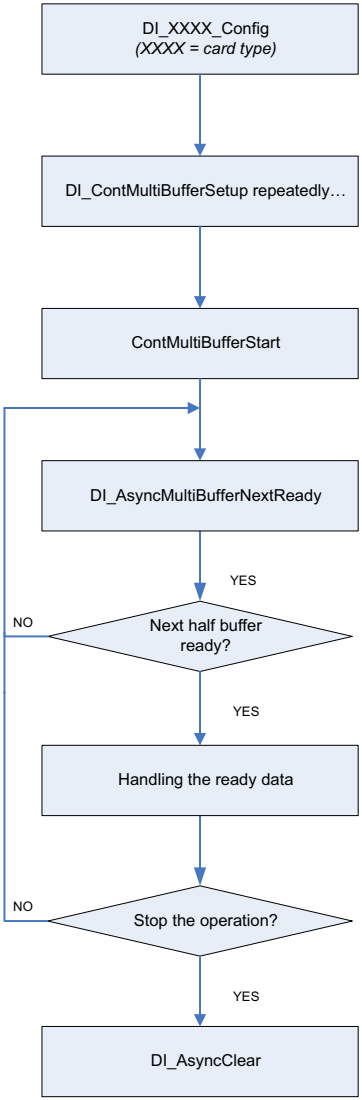


Example code fragment:

```
card = Register_Card(PCI_7200, card_number);
…
DI_7200_Config(card,TRIG_INT_PACER, DI_NOWAITING,
   DI_TRIG_FALLING, IREQ_FALLING);
DI_AsyncDblBufferMode (card, 1); // Double-buffered
   mode
DI_ContReadPort(card, 0, pMem, data_size,
   (F64)sample_rate, ASYNCH_OP)
do {
                do {
           DI_AsyncDblBufferHalfReady(card,
   &HalfReady);
      } while (!HalfReady);

    DI_AsyncDblBufferTransfer(card, pMem);

} while (!clear_op);

DI_AsyncClear(card, &count);
            …
Release_Card(card);
```

Multiple-buffered Asynchronous Continuous Digital Input

This section describes the function flow typical of multi-buffered asynchronous digital input operation. While performing continuous DI operation, the DI configuration function has to be called at the beginning of the application. For asynchronous DI, the SyncMode argument in continuous DI functions has to be set to ASYNCH_OP.


DI_XXXX_Config
(XXXX = card type)
DI_ContMultiBufferSetup repeatedly…
ContMultiBufferStart
DI_AsyncMultiBufferNextReady
YES
NO
Next half buffer ready?
YES
Handling the ready data
NO
Stop the operation?
YES
DI_AsyncClear

Example code fragment:

```
card = Register_Card(PCI_7300A_RevB, card_number);
…
DI_7300B_Config(card, 16, TRIG_CLK_10MHZ,
   P7300_WAIT_NO, P7300_TERM_ON, 0, 1, 1);
//setting the DMA buffers repeatedly
DI_ContMultiBufferSetup (card, in_buf, data_size,
   &BufferId);
DI_ContMultiBufferSetup (card, in_buf, data_size,
   &BufferId);
…
// start multi-buffered DI
DI_ContMultiBufferStart (card, 0, 1);

do {
          do {
          DI_AsyncDblBufferHalfReady(card,
   &HalfReady);
      } while (!HalfReady);

     //Handling the ready data

} while (!clear_op);

DI_AsyncClear(card, &count);
               …
Release_Card(card);
```

4.4 Digital Output Programming Hints

The PCIS-DASK provides three types of digital output operation: non-buffered single-point digital output operation, buffered continuous digital output operation, and pattern generation.

The non-buffered single-point DO uses software polling method to write data to the device. The programming scheme for this kind of DO operation is described in section .

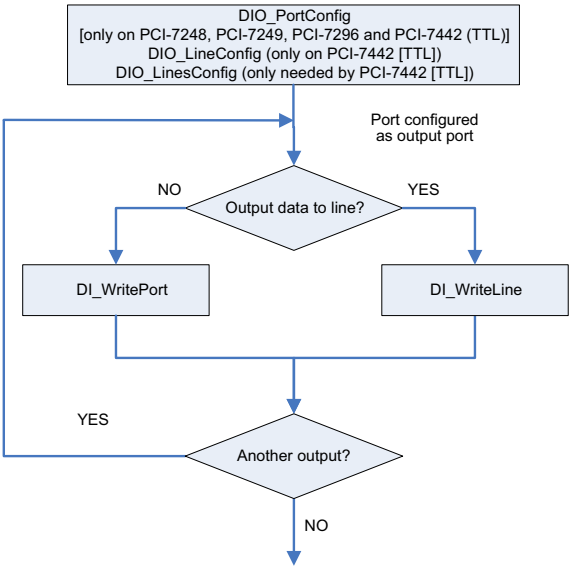
The buffered continuous DO uses DMA transfer method to transfer data from user's buffer to device. The maximum number of count in one transfer depends on the size of initially allocated memory for digital output in the driver. The driver allocates the memory during system boot (in Windows® NT) or Windows startup (in Windows® 98). It is recommended that applications use DO_InitialMemoryAllocated function to get the size of initially allocated memory before performing continuous DO operation.

The buffered continuous digital output includes synchronous continuous DO and asynchronous continuous DO. These are described in section and section . For special consideration and performance issues for the buffered continuous analog input, refer to **Chapter 5: Continuous Data Transfer**.

The Pattern Generation DO outputs digital data pattern repeatedly at a predetermined rate. The programming scheme for this kind of DO operation is described in section .

One-Shot Digital Output

This section describes the function flow typical of non-buffered single-point digital output operation. While performing one-shot DO operation, the cards whose I/O port can be set as input or output port (PCI-7248, PCI7249, PCI-7296, and PCI-7442) need to include port configuration function at the beginning of the application.



Example code fragments:

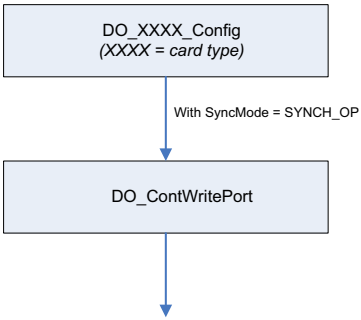
```
card = Register_Card(PCI_7248, card_number);
//port configured
DIO_PortConfig(card ,Channel_P1A, OUTPUT_PORT);
DIO_PortConfig(card, Channel_P1B, OUTPUT_PORT);
DIO_PortConfig(card, Channel_P1CL, OUTPUT_PORT);
DIO_PortConfig(card, Channel_P1CH, OUTPUT_PORT);
//DO operation
DO_WritePort(card, Channel_P1A, outA_value);
…
Release_Card(card);
```

:

```
card = Register_Card(PCI_7442, card_number);
//Lines configured
DIO_LineConfig(card, P7442_TTL0, 0, OUTPUT_LINE);
//DO operation
DO_WriteLine(card, P7442_TTL0, 0, out_value);
…
Release_Card(card);
```

Synchronous Continuous Digital Output

This section describes the function flow typical of synchronous digital output operation. While performing continuous DO operation, the DO configuration function has to be called at the beginning of the application. In addition, the SyncMode argument in continuous DO functions for synchronous mode has to be set to SYNCH_OP.

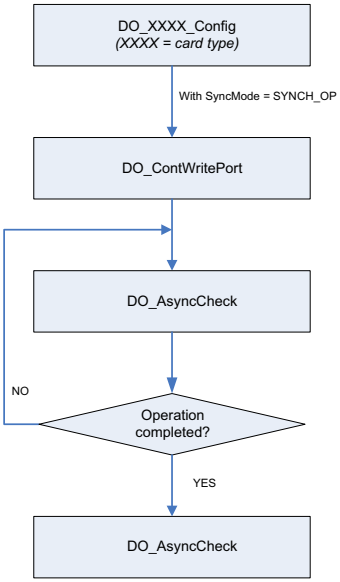


Example code fragment:

```
card = Register_Card(PCI_7200, card_number);
…
DO_7200_Config(card, TRIG_INT_PACER, OREQ_DISABLE,
   OTRIG_LOW);
DO_AsyncDblBufferMode (card, 0); //non-double-buffer
   mode
DO_ContWritePort(card, 0, DoBuf, count, 1,
   (F64)sample_rate, SYNCH_OP);
…
Release_Card(card);
```

Asynchronous Continuous Digital Output

This section describes the function flow typical of asynchronous digital output operation. While performing continuous DO operation, the DO configuration function has to be called at the beginning of the application. In addition, the SyncMode argument in continuous DO functions for asynchronous mode has to be set to ASYNCH_OP.

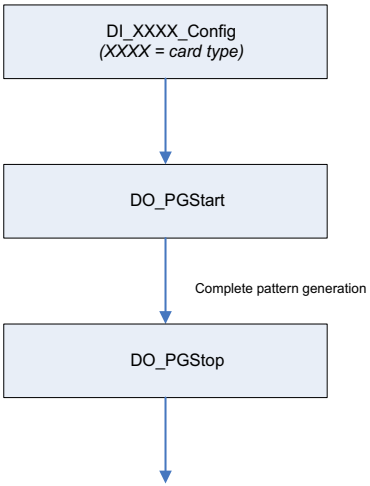


Example code fragment:

```
card = Register_Card(PCI_7200, card_number);
…
DO_7200_Config(card, TRIG_INT_PACER, OREQ_DISABLE,
   OTRIG_LOW);
DO_ContWritePort(card, 0, DoBuf, count, 1,
   (F64)sample_rate, ASYNCH_OP);
          do {
       DO_AsyncCheck(card, &bStopped, &count);
            } while (!bStopped);

DO_AsyncClear(card, &count);
             …
Release_Card(card);
```

Pattern Generation Digital Output

This section describes the function flow typical of pattern generation for digital output. While performing pattern generation of DO, the DO configuration function has to be called at the beginning of the application.

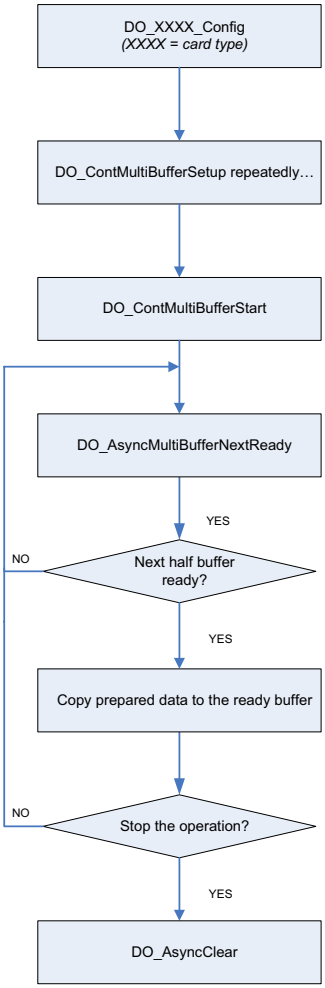


Example code fragment:

```
card = Register_Card(PCI_7300A_RevB, card_number);
…
DO_7300B_Config (card, 16, TRIG_INT_PACER,
   P7300_WAIT_NO, P7300_TERM_ON, 0, 0x40004000);
//start pattern generation
DO_PGStart (card, out_buf, 10000, 5000000);
…
//stop pattern generation
DO_PGStop (card);
Release_Card(card);
```

Multiple-buffered Asynchronous Continuous Digital Output

This section describes the function flow typical of multi-buffered asynchronous digital output operation. While performing continuous DO operation, the DO configuration function has to be called at the beginning of the application. For asynchronous DO, the SyncMode argument in continuous DO functions has to be set as ASYNCH_OP.



Example code fragment:

```
card = Register_Card(PCI_7300A_RevB, card_number);
…
DO_7300B_Config (card, 16, TRIG_CLK_10MHZ,
   P7300_WAIT_NO, P7300_TERM_ON, 0, 0x00040004);
//setting the DMA buffers repeatedly
DO_ContMultiBufferSetup (card, out_buf, data_size,
   &BufferId);
DO_ContMultiBufferSetup (card, out_buf, data_size,
   &BufferId);
…
// start multi-buffered DO
DO_ContMultiBufferStart (card, 0, 1);

do {
                do {
           DO_AsyncDblBufferHalfReady(card,
   &HalfReady);
       } while (!HalfReady);

     // Copy prepared data to the ready buffer

} while (!clear_op);

DO_AsyncClear(card, &count);
                …
Release_Card(card);
```

4.5 DAQ Event Message Programming Hints

DAQ Event Message functions are efficient ways to monitor your background data acquisition processes without dedicating your foreground process for status checking. There are two kinds of events: AI/DI/DO operation completed notification event and half buffer ready notification event.

To receive notification from the PCIS-DASK data acquisition process in case of special events, you can call AI_EventCallBack, DI_EventCallBack, or DO_EventCallBack to specify an event.

Event notification is done through user-defined callbacks. When a user-specified DAQ event occurs, PCIS-DASK calls the user-defined callback. After receiving the message, the user's application carries out the appropriate task.

The event message mechanism is easy and safe in Windows® 98 and Windows® NT systems. However, the time delay between the event and notification is highly variable and depends largely on how your system is loaded. In addition, if a callback function is called, succeeding events will not be handled until your callback has returned. If the time interval between events is smaller than the time taken for callback function processing, the succeeding events will not be handled. Therefore this mechanism is not suitable for the frequent events occurrence condition.

Example code fragment:

```
card = Register_Card(PCI_9118DG, card_number);
AI_9118_Config(card,P9118_AI_BiPolar|P9118_AI_SingE
   d,
   P9118_AI_DtrgPositive|P9118_AI_EtrgPositive|P911
   I_AboutTrgEn,0,postCount);
AI_AsyncDblBufferMode(card, 1);  //double-buffer mo

// Enable half buffer ready event notification
          AI_EventCallBack (card, 1, DBEvent, (U32)
   DB_cbfn );

//Enable AI completeness event notification
AI_EventCallBack (card, 1, AIEnd, (U32) AI_cbfn );

AI_ContScanChannels (card, channel, range, NULL,
   data_size, (F64)sample_rate, ASYNCH_OP); or
AI_ContReadChannel(card, channel, range, NULL,
   data_size, (F64)sample_rate, ASYNCH_OP)
....
Release_Card(card);

//Half buffer ready call back function
void DB_cbfn()
{
//half buffer is ready
AI_AsyncDblBufferTransfer(card, ai_buf);  //transfe
   to user buffer
….
}

//AI completeness call back function
void AI_cbfn()
{
//AI is completed ]
AI_AsyncClear(card, &count);
//Transfer the remainling data into the user buffer
AI_AsyncDblBufferTransfer(card, ai_buf);
….
}
```

4.6 Interrupt Event Message Programming Hints

The PCIS-DASK comes with two methods of performing interrupt occurrence notification for NuDAQ DIO cards that have dual-interrupt system.

The **Event Message** method handles event notification through user-defined callbacks and/or the Windows Message queue (for VB5, through user-defined callbacks only). When a user-specified interrupt event occurs, PCIS-DASK calls the user-defined callback (if defined) and/or puts a message into the Windows Message queue, if you specified a window handle. After receiving the message, the user's application can carry out the appropriate task.

The event message mechanism is easy and safe in Windows® 98 and Windows® NT systems. However, the time delay between the event and notification is highly variable and depends largely on how your system is loaded. In addition, if a callback function is called, succeeding events will not be handled until your callback has returned. If the time interval between events is smaller than the time taken for callback function processing, the succeeding events will not be handled. Therefore this mechanism is not suitable for the frequent events occurrence condition.

The **Event Status** checking and waiting method handles interrupt event status checking through Win32 wait functions, such as WaitForSingleObject or WaitForMultipleObjects. This method is useful for situations when the interrupt event occurs very often and when the applications written in the language doesn't support function pointers (e.g. VB4).

Through user-defined callbacks and Windows Message queue

Example code fragment:

```
card = Register_Card(PCI_7230, card_number);

//INT1 event notification is through window message
DIO_INT1_EventMessage (card, INT1_EXT_SIGNAL, hWnd,
   WM_INT, NULL);

//INT2 event notification is through a callback
   function
DIO_INT2_EventMessage (card, INT2_EXT_SIGNAL, hWnd,
   NULL, (void *) cbfn);
….
//window message handling function
long  PASCAL MainWndProc(hWnd, message, wParam,
   lParam)
{
   switch(message) {
   ….
   case WM_INT: //interrupt event occurring message
   ….
   break;
   ….
   case WM_DESTROY:
//Disable interrupts
DIO_INT1_EventMessage (card, INT1_DISABLE,   hMainWn
   NULL, NULL);
DIO_INT2_EventMessage (card, INT2_DISABLE, hMainWnd
   NULL, NULL);
//Release card
if (card >= 0) Release_Card(card);
PostQuitMessage(0);
break;
….
 }
  }
….
//call back function
LRESULT CALLBACK cbfn()
{
          ….
}
```

Through a Win32 wait function

Example code fragment:

```
card = Register_Card(PCI_7230, card_number);
DIO_SetDualInterrupt(card, INT1_EXT_SIGNAL,
   INT2_EXT_SIGNAL, hEvent);
….
//wait for  INT1 event
if (WaitForSingleObject(hEvent[0], INFINITE) ==
   WAIT_OBJECT_0) {
         ResetEvent(hEvent[0]);
……
   }
  …..
//wait for  INT2 event
if (WaitForSingleObject(hEvent[1], INFINITE) ==
   WAIT_OBJECT_0) {
         ResetEvent(hEvent[1]);
……
}
…..
if (card >= 0) Release_Card(card);
```

5 Continuous Data Transfer

The continuous data transfer function in the PCIS-DASK inputs or outputs blocks of data to or from a plugged-in NuDAQ PCI device. For input operations, the PCIS-DASK transfers the incoming data to a buffer in the system memory. For output operations, the PCIS-DASK transfers outgoing data from a buffer in the computer memory to the NuDAQ PCI device.

This chapter describes the mechanism and techniques that PCIS-DASK use for continuous data transfer and the considerations for selecting the continuous data transfer mode (synchronous or asynchronous, double buffered, triggered or non-triggered mode).

5.1 Mechanisms

The PCIS-DASK uses two mechanisms to perform continuous data transfer: interrupt transfer and DMA.

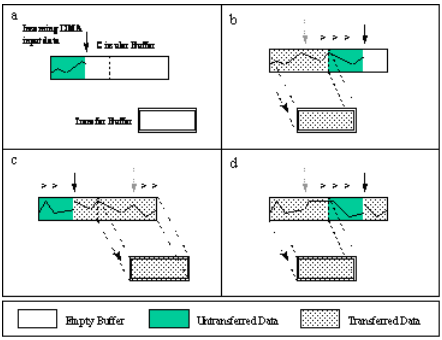
Interrupt transfer transfers data through the interrupt mechanism, while the DMA controller chip performs data transfer via a hardware. The PCIS-DASK uses the interrupt or DMA depending on the device. When the device supports both mechanisms, the PCIS-DASK decides on the data transfer method that takes maximum advantage of available resources. For example, PCI-9112 supports both interrupt and DMA for data transfers. The PCIS-DASK uses DMA data transfer in this instance since it is faster. For PCI-9111 that supports FIFO Half-Full and EOC interrupt transfer modes, the PCIS-DASK uses the FIFO Half-Full interrupt transfer mode since the CPU is interrupted to do data transfer only when the FIFO becomes half-full.

5.2 Double-Buffered AI/DI Operation

The PCIS-DASK uses double-buffering techniques in its driver software for continuous input of large amounts of data.

Double Buffer Mode Principle

The data buffer for double-buffered continuous input operation is a logical circular buffer. It is logically divided into two equal halves. The double-buffered input begins when the device starts writing data into the first half of the circular buffer (a). Refer to figure below. When the device starts writing to the second half of the circular buffer, the data is copied from the first half to the transfer buffer (b) also known as user buffer. You can now process the data in the transfer buffer depending on the application needs. After the board has filled the second half of the circular buffer, the board returns to the first half buffer and overwrites the old data. The data is copied from the second half of the circular buffer to the transfer buffer (c). The data in the transfer buffer is again available for process. The process may be repeated endlessly to provide a continuous stream of data to your application (d).



The PCIS-DASK double buffer mode functions were designed according to the principle described above. If you use:

```
AI_AsyncDblBufferMode or
DI_AsyncDblBufferMode
```

to enable double buffer mode, the following continuous AI/DI function performs double-buffered continuous AI/DI. You may call

```
AI_AsyncDblBufferHalfReady or
DI_AsyncDblBufferHalfReady
```

to check if data in the circular buffer is half full and ready for copying to the transfer buffer. Then you may call:

```
AI_AsyncDblBufferTransfer or
DI_AsyncDblBufferTransfer
```

to copy data from the ready half buffer to the transfer buffer.

Single-Buffered Versus Double-Buffered Data Transfer

Single-buffered data transfer is the most common method for continuous data transfer. In single-buffered input operations, a fixed number of samples are acquired at a specified rate and transferred into user's buffer. After the user's buffer stores the data, the application can analyze, display, or store the data to the hard disk for later processing. Single-buffered operations are relatively simple to implement and can usually take advantage of the full hardware speed of the device. However, the major disadvantage of single-buffered operation is that the maximum amount of data that can be input at any one time is limited to the amount of initially allocated memory allocated in driver and the amount of free memory available in the computer.

In double-buffered operations, as mentioned above, the data buffer is configured as a circular buffer. Therefore, unlike single-buffered operations, double-buffered operations reuse the same buffer and are able to input or output an infinite number of data points without requiring an infinite amount of memory. However, there exists the undesired result of data overwritten for double-buffered data transfer. The device might overwrite data before PCIS-DASK has copied it to the transfer buffer. Another data overwritten problem occurs when an input device overwrites data that PCIS-DASK is simultaneously copying to the transfer buffer. Therefore, the data must be processed by the application at least as fast as the rate at which the device is reading data. For most of the applications, this requirement depends on the speed and efficiency of the computer system and programming language.

Hence, double buffering might not be practical for high-speed input applications.

5.3 Trigger Mode Data Acquisition for Analog Input

A trigger is an event that occurs based on a specified set of conditions. An interrupt mode or DMA-mode analog input operation can use a trigger to determinate when acquisition stops or starts.

The PCIS-DASK also provides two buffering methods for trigger mode AI double-buffering and single-buffering. However, the single buffer in trigger mode AI is different from that in non-trigger mode AI. It is a circular buffer just like that in double buffer mode but the data stored in the buffer can be processed only when the continuous data reading is completed. The buffer is reused until the data acquisition operation is completed. Therefore, to keep the data you want to transfer from being overwritten, the size of the single buffer should be the same as or larger than the amount of data you want to access.

For example, if you want to perform single-buffered middle-trigger AI with PCI-9812, and the amount of data you want to collect before and after the trigger event are 1000 and 3000, respectively, the size of single buffer should be at least 4000. Since the data are handled after the input operation is completed, data loss problems are eliminated.

Since PCIS-DASK uses asynchronous AI to perform trigger mode data acquisition, the SyncMode of continuous AI should be set as ASYNCH_OP.

6 Utilities

This chapter introduces the tools that came with the PCIS-DASK package.

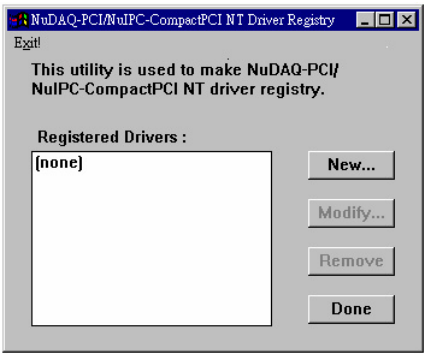
6.1 Win32 Utilities

NuDAQ Registry/Configuration (PciUtil)

The **PciUtil** registers the PCIS-DASK drivers (Windows® NT4 only), removes installed drivers (Windows® NT4 only), and sets/modifies the allocated buffer sizes of AI, AO, DI, and DO. By default, the utility is located at <InstallDir>\Util directory.

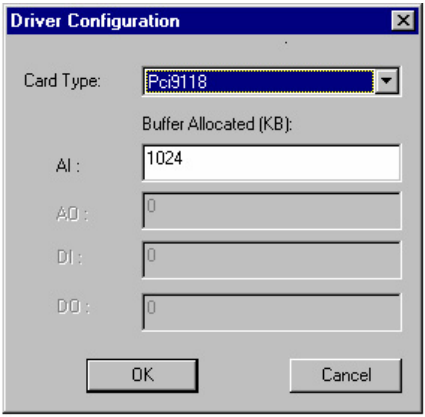
Using PciUtil in Windows® NT

The PciUtil main window shows all registered PCIS-DASK/NT drivers. When detected, PciUtil displays the driver in the **Registered Drivers** section.



To register a PCIS-DASK driver, click **New**.

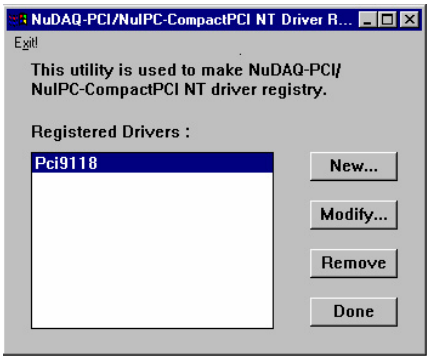
A **Driver Configuration** window appears.



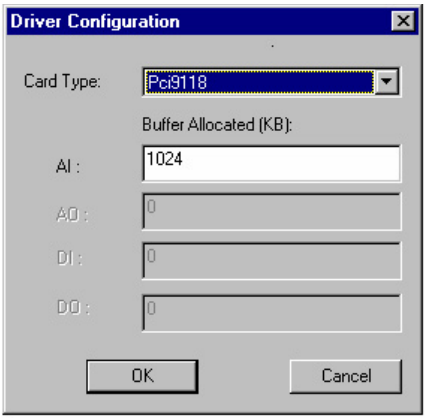
From the **Card Type** drop-down menu, select the driver you want to register, then key-in the allocated buffer (KB) for the AI, AO, DI, or DO functions depending on your application requirements.

The allocated buffer represents the size of contiguous, initially allocated memory for continuous analog input, analog output, digital input, and digital output. The device driver allocates the memory size during system startup. The size of initially allocated memory is the maximum memory size that DMA or interrupt transfer can be performed. An unexpected result occurs when the DMA or interrupt transfer performs an operation exceeding the initially allocated size.

After setting the driver configuration, click **OK** to register the driver and return to the PciUtil main window. The registered driver appears on the registered driver list.



To change the allocated buffer, select the driver from the Registered Driver list, then click **Modify**. The **Driver Configuration** window appears.



Key-in the new allocated buffer size in each available AI, AO, DI and DO fields, then click **OK**.

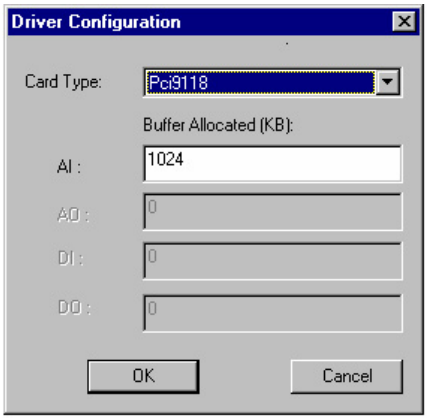
To remove a registered driver, select the driver from the **Registered Driver** list from the PciUtil main window, then click **Remove**. The selected driver is deleted from the registry table.

Using PciUtil in Windows® 98/2000/XP/Server 2003

The **PciUtil** sets or modifies the allocated buffer sizes of AI, AO, DI and DO in Windows® 98/2000/XP/Server 2003 environment.

The allocated buffer represents the size of contiguous, initially allocated memory for continuous analog input, analog output, digital input, and digital output. The device driver allocates the memory size during system startup. The size of initially allocated memory is the maximum memory size that DMA or interrupt transfer can be performed. An unexpected result occurs when the DMA or interrupt transfer performs an operation exceeding the initially allocated size.

To set the buffer size, key-in the allocated buffer (KB) for the AI, AO, DI, or DO functions depending on your application requirements from the **Driver Configuration** window. Click **OK** when finished.

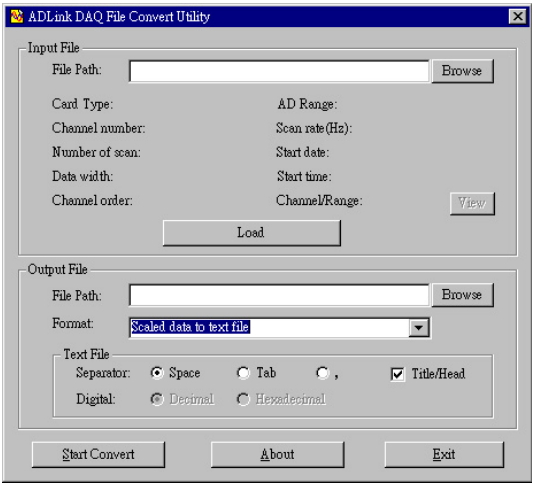


Data File Converter (DAQCvt)

When performing continuous data acquisition followed by storage to a disk operation, the data files generated by the PCIS-DASK functions are written in binary format. Normal text editors may not be able to read binary files and spreadsheet applications may not recognize binary files for analysis.

The PCIS-DASK comes with the **DAQCvt** tool to conveniently convert these binary files into easily-read formats. The utility may be found at <InstallDir>\Util directory.

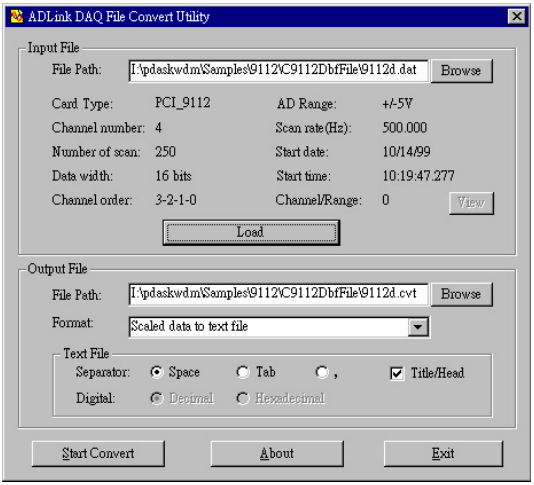
The DAQCvt main window is shown below.



The DAQCvt main window is divided into two sections: **Input File** and **Output File**. The Input File section identifies the source data file while the Output File section identifies the destination for the converted file.

To convert a binary file:

1. Click **Browse** to locate the binary file.
2. After locating the binary file, click **Load**. The binary file information displays on the Input File section for your reference. The default converted data file path and format also appear in the Output File section.



NOTE The default destination for the converted file with a .cvt extension is in the same directory as the source file.

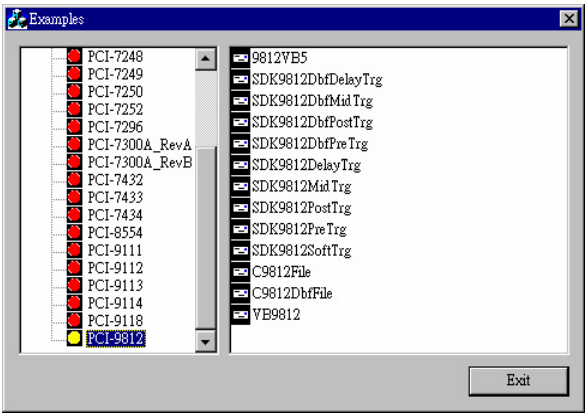
3. To change the default file path, click **Browse**, then select the destination for the converted file.
4. To change the format of the converted file, click the **Format** drop-down menu, then select from three available data formats. Refer to the formats' description below:
   - ▷ **Scaled data to text file**. The data in hexadecimal format is scaled to engineering unit (voltage, ample, etc.) according to the card type, data width, and data range, then written to disk in text file format. This type is available for the data accessed from continuous AI operation only.
   - ▷ **Scaled data to binary file (float)**. The data in hexadecimal is scaled to engineering unit (voltage, ample, etc.) according to the card type, data width and data range, then written to disk in binary file format. This type is available for the data accessed from continuous AI operation only.

- ▷ **Binary codes to text file**. The data in hexadecimal format or converted to a decimal value is written to disk in text file format. If the original data includes channel information, the raw value is handled to get the real data value. This type is available for data accessed from continuous AI and DI operations.

5. Select the text file separator. You may separate data using a space, a comma, or a tab.
6. Check the **Title/Head** option if you want to add a title/head, including the card type information, at the beginning of the file.
7. When finished, click **Start Convert** to convert the file.

Sample Programs Browser

The PCIS-DASK comes with **Examples.exe** — a sample program browser that allows you to view and execute all bundled sample programs. Examples.exe is located at the **<InstallDir>\Samples** directory. After launching Examples.exe, double-click the icon of the sample you want to execute.



6.2 PCIS-DASK/X Utilities

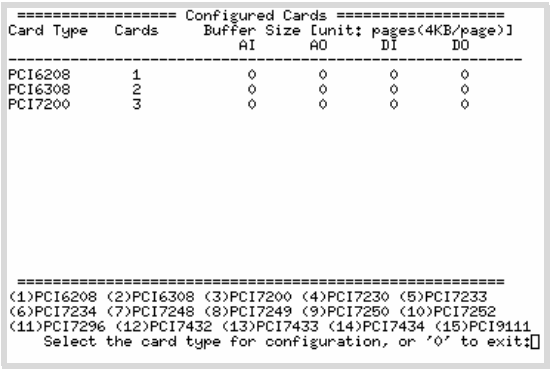
This section introduces the tools that comes with the PCIS-DASK/X package for Linux distributions.

dask_conf

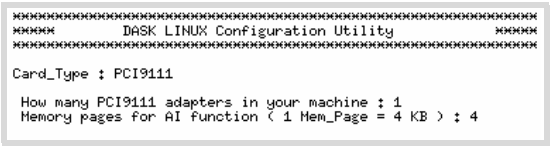
The **dask_conf** configures the PCIS-DASK drivers, removes configured drivers, and sets or modifies the allocated buffer sizes of AI, AO, DI and DO. By default, the dask_conf is located at pci-dask_xxx/util (where xxx is the version number) directory.

Using dask_conf in Linux

The dask_conf main screen shows all configured PCIS-DASK/X drivers in the **Configured Cards** list.



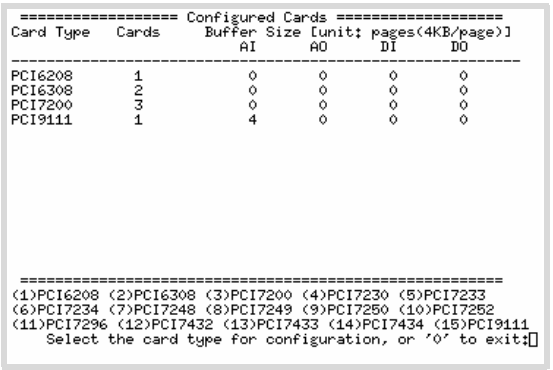
To configure a PCIS-DASK/X driver, type the card type number. A **Driver Configuration** screen appears.



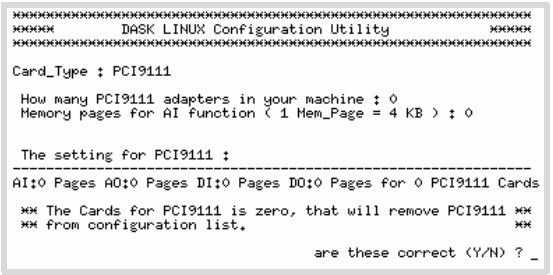
From this screen, key-in the number of cards and buffer size for continuous operations. To be platform-independent, the buffer size is set by the memory-page. The PAGE_SIZE for Intel platform is 4 KB. The **Memory Pages** of AI, AO, DI, and DO represent the

number of pages of contiguous initially allocated memory for continuous analog input, analog output, digital input, and digital output. The device driver allocates these memory sizes from the memory management module.

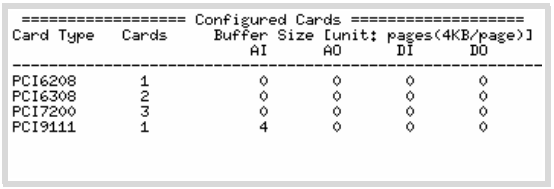
After the selected driver is configured, type **Y** to confirm and return to the dask_conf main screen. The configured driver now appears at the Configured Cards list.



To modify the driver configuration, including the number of cards and the buffer size, select the driver from the list, then assign the new settings. When the number of cards is set to zero, the configuration for the selected driver is removed.



When configuration is finished, the device configuration information is saved in pci-dask_xxx/drivers/pcidask.conf. The content of dask.conf is shown on the following illustration.



```
 ================== Configured Cards ====================
Card Type   Cards     Buffer Size [unit: pages(4KB/page)]
                          AI       AO       DI       DO
-----------------------------------------------------------
PCI6208         1          0        0        0        0
PCI6308         2          0        0        0        0
PCI7200         3          0        0        0        0
PCI9111         1          4        0        0        0
```

6.3 Module Installation Script

The PCI-bus architecture allows automatic detection of PCI devices right after these are installed and device nodes are created.

The following commands are necessary:

```
“insmod p9111”
“grep'p9111' /proc/devices”
“mknod /dev/PCI9111W0 c 254 0”
“mknod /dev/PCI9111W1 c 254 1”…
```

You can do these commands manually or use the provided installation scripts. The installation script is located at pci-dask_xxx/drivers.

Using the **pcidask.conf** configuration file, the installation script inserts all previously configured device modules and the memory management module, if required. The script then makes device nodes according to the number of cards. To install, execute this script:

```
<InstallDir>/pci-dask_xxx/drivers/dask_inst.pl
```

By default, the installation script reads the configuration file in the current directory. You may specify the work directory for the PCIS-DASK/X to install script from the command argument. For example, if the pcis-dask/x had been installed in /usr/local/pdask, you may install the driver using the following command:

```
dask_inst.pl   /usr/local/pdask
```

The installation script reads the related configuration file by its argument and inserts the modules needed by the configured devices. This may be useful if the installation needs to be executed by init after system starts up.

For example, if you install the PCIS-DASK/X in the /usr/pdask directory and the system needs to insert the modules automatically. You may add the following command in the /etc/inittab, then the init process inserts the modules automatically.

```
ad:2345:wait:/usr/pdask/drivers/dask_inst.pl  /
     usr/pdask   "Insert ADLINK modules"
```

Because the current modules are designed based on Uni-Processor kernel, these modules may not work with SMP kernel. The installation script checks the kernel version through the /proc/sys/kernel/version file. For SMP kernel, the version-checking procedure displays the additional error/warning messages and stops the installation.

6.4 Uninstallation Script

The dask_remove.pl removes the PCIS-DASK/X installed in Linux. By default, this script is located at pci-dask_xxx/util directory.

To remove the PCIS-DASK for Linux, execute the uninstallation script:

```
<InstallDir>/pci-dask_xxx/util/dask_remove.pl
```

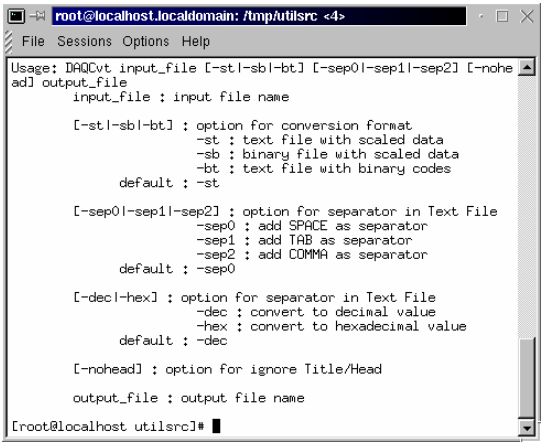
The script removes the device nodes made in /dev and the library copied into /usr/lib.

6.5 Data File Converter (DAQCvt)

When performing continuous data acquisition followed by storage to a disk operation, the data files generated by the PCIS-DASK functions are written in binary format. Normal text editors may not be able to read binary files and spreadsheet applications may not recognize binary files for analysis.

The PCIS-DASK comes with the **DAQCvt** tool to conveniently convert these binary files into easily-read formats. The utility may be found at <InstallDir>\Util directory.

DAQCvt may be launched using the "--help" argument. The DAQCvt main screen is shown below.



Options for data format conversion

DAQCvt provides three data format options.

```
-st : text file with scaled data
```

The data in hexadecimal format is scaled to engineering unit (voltage, ample, etc.) according to the card type, data width, and data range, then written to disk in text file format. This type is available for the data accessed from continuous AI operation only.

```
-sb : binary file with scaled data (float)
```

The data in hexadecimal is scaled to engineering unit (voltage, ample, etc.) according to the card type, data width and data range, then written to disk in binary file format. This type is available for the data accessed from continuous AI operation only.

```
-bt : text file with binary codes
```

The data in hexadecimal format or converted to a decimal value is written to disk in text file format. If the original data includes channel information, the raw value is handled to get the real data value. This type is available for data accessed from continuous AI and DI operations.

The default option for data format conversion is -st.

Options for separator in text file

The data separator in the converted text file may either be a space, tab, or comma.

```
-sep0 : add space as separator
-sep1 : add Tab as separator
-sep2 : add comma as separator
```

The default option for data format conversion is -sep0.

Options for Title/Head in text file

If you do not want to add the title/head at the beginning of the file, add the -nohead option.

After specifying the input filename, output filename, and the options in the command line, DAQCvt converts the file and saves it into the default location.

7 Sample Programs

Several sample programs are provided in the software CD. These sample program are designed to assist you when creating your own applications using PCIS-DASK.

NOTE ADLINK periodically upgrades the PCIS-DASK for new cards/modules. Check the card/modules's Release Notes to know if PCIS-DASK supports it.

7.1 Brief Program Descriptions

Below is a list of programs and their description.

Card Type	Sample Name	Description
PCI-6202	**C6202AO_SSI_Source**	D/A Continuous Conversion with SSI signal out *Visual C++ Program*
	C6202AO_SSI_Target	D/A Continuous Conversion with SSI signal in *Visual C++ Program*
	C6202AODoubleBuffer	D/A Continuous Conversion in double buffer mode *Visual C++ Program*
	C6202AOSingleBuffer	D/A Continuous Conversion *Visual C++ Program*
	C6202DIO	DIO Polling *Visual C++ Program*
	C6202Encoder	Encoder with Encoder Position Trigger *Visual C++ Program*
	C6202GPTC	General Purpose Timer/Counter *Visual C++ Program*
	CAOOnePoint	D/A Polling *Visual C++ Program*
PCI-6208	**SDK6208V**	D/A conversion of PCI-6208V/16V *Visual C/C++ Program*
	SDK6208A	D/A conversion of PCI-6208A *Visual C/C++ Program*
	VB6208	D/A conversion of PCI-6208A *Visual Basic Program*
	VB6216	D/A conversion of PCI-6208V/16V *Visual Basic Program*
PCI-6308	**SDK6308V**	D/A conversion of PCI-6308V *Visual C/C++ Program*
	SDK6308A	D/A conversion of PCI-6308A *Visual C/C++ Program*
	VB6308A	D/A conversion of PCI-6308A *Visual Basic Program*
	VB6308V	D/A conversion of PCI-6308V *Visual Basic Program*

Card Type	Sample Name	Description
PCI-7200	**C7200File**	1. Digital input of PCI-7200/cPCI-7200 through DMA transfer 2. Storing the data to disk *Visual C/C++ console Program*
	C7200DbfFile	1. Double buffer mode digital input of PCI-7200/ cPCI-7200 through DMA transfer 2. Storing the data to disk *Visual C/C++ console Program*
	SDK7200Wave	Digital input of PCI-7200/cPCI-7200 through DMA transfer *Visual C/C++ Program*
	SDK7200DbfWav	Double buffer mode digital input of PCI-7200/cPCI-7200 through DMA transfer *Visual C/C++ Program*
	SDK7200HdSk	HandShanking mode digital input of PCI-7200/cPCI-7200 through DMA transfer *Visual C/C++ program*
	SDKETrigLine	ExtTrig Line setting of PCI-7200/cPCI-7200 *Visual C/C++ Program*
	VB7200Dma	Digital input of PCI-7200/cPCI-7200 through DMA transfer *Visual Basic Program*
PCI-7230	**SDK7230**	D/I, and D/O of PCI-7230/cPCI-7230 *Visual C/C++ Program*
	SDK7230Int	D/I, and D/O of PCI-7230/cPCI-7230 by Interrupt Event Status checking and waiting method Visual C/C++ Program
	SDK7230DbEvt	D/I, and D/O of PCI-7230/cPCI-7230 by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ Program*
	SDK7230IntMsg	D/I, and D/O of PCI-7230/cPCI-7230 by Interrupt Event Message method
	SDK7230DbEvtMsg	D/I, and D/O of PCI-7230/cPCI-7230 by Interrupt Event Message method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7230	D/I, and D/O of PCI-7230/cPCI-7230 *Visual Basic Program*
PCI-7233	**SDK7233**	D/I of PCI-7233 *Visual C/C++ Program*
	SDK7233Int	D/I of PCI-7233 by Interrupt Event Status checking and waiting method *Visual C/C++ Program*
	SDK7233DbEvt	D/I of PCI-7233 by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7233	D/I of PCI-7233 *Visual Basic Program*
PCI-7234	**SDK7234**	D/O of PCI-7234 *Visual C/C++ Program*
	VB7234	D/O of PCI-7234 *Visual Basic Program*

Card Type	Sample Name	Description
PCI-7248	**SDK7248**	D/I, and D/O of PCI-7248/cPCI-7248 *Visual C/C++ Program*
	SDK7248Int	D/I, and D/O of PCI-7248/cPCI-7248 by Interrupt Event Status checking and waiting method *Visual C/C++ Program*
	SDK7248DbEvt	D/I, and D/O of PCI-7248/cPCI-7248 by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ Program*
	SDK7248IntMsg	D/I, and D/O of PCI-7248/cPCI-7248 by Interrupt Event Message method *Visual C/C++ Program*
	SDK7248DbEvtMsg	D/I, and D/O of PCI-7248/cPCI-7248 by Interrupt Event Message method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7248	D/I, and D/O of PCI-7248/cPCI-7248 *Visual Basic Program*
PCI-7249	**SDK7249**	D/I, and D/O of cPCI-7249 *Visual C/C++ Program*
	SDK7249Int	D/I, and D/O of cPCI-7249 by Interrupt Event Status checking and waiting method *Visual C/C++ Program*
	SDK7249DbEvt	D/I, and D/O of cPCI-7249 by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7249	D/I, and D/O of cPCI-7249 *Visual Basic Program*
PCI-7250	**SDK7250**	D/I, and D/O of PCI-7250/51 *Visual C/C++ Program*
	VB7250	D/I, and D/O of PCI-7250/51 *Visual Basic Program*
PCI-7252	**SDK7252**	D/I, and D/O of cPCI-7252 *Visual C/C++ Program*
	VB7252	D/I, and D/O of cPCI-7252 *Visual Basic Program*
PCI-7256	**SDK7256**	D/I, and D/O of PCI-7256 *Visual C/C++ Program*
	SDK7256Int	D/I, and D/O of PCI-7256 by Interrupt Event Status checking and waiting method *Visual C/C++ Program*
	SDK7256DbEvt	D/I, and D/O of PCI-7256 by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7256	D/I, and D/O of PCI-7256 *Visual Basic Program*

Card Type	Sample Name	Description
PCI-7296	**SDK7296**	D/I, and D/O of PCI-7296 *Visual C/C++ sample program*
	SDK7296Int	D/I, and D/O of PCI-7296 by Interrupt Event Status checking and waiting method *Visual C/C++ sample program*
	SDK7296DbEvt	D/I, and D/O of PCI-7296 by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ sample program*
	SDK7248IntMsg	D/I, and D/O of PCI-7296 by Interrupt Event Message method *Visual C/C++ Program*
	SDK7248DbEvtMsg	D/I, and D/O of PCI-7296 by Interrupt Event Message method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7296	D/I, and D/O of PCI-7296 *Visual Basic Program*
PCI-7300 Rev.A	**SDK7300Wave**	Digital input of PCI-7300A_Rev.A/cPCI-7300A_Rev.A through DMA transfer *Visual C/C++ Program*
	S7300PGwav	Pattern generation of PCI-7300A_Rev.A/cPCI-7300A_Rev.A *Visual C/C++ program*
	SDK7300aMBufWav	Multiple buffer mode digital input of PCI-7300A_Rev.A/cPCI-7300A_Rev.A through DMA transfer *Visual C/C++ sample program*
	SDK7300Int	Interrupt operation of PCI-7300A_Rev.A/cPCI-7300A_Rev.A by Event Status checking and waiting method *Visual C/C++ program*
	SDK7300DbEvt	Interrupt operation of PCI-7300A_Rev.A/cPCI-7300A_Rev.A by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ Program*
	C7300File	1. Digital input of PCI-7300A_Rev.A/ cPCI-7300A_Rev.A through DMA transfer 2. Storing the data to disk *Visual C/C++ console program*
PCI-7300 Rev.B	**SDK7300Wave**	Digital input of PCI-7300A_Rev.B/cPCI-7300A_Rev.B through DMA transfer *Visual C/C++ Program*
	S7300PGwav	Pattern generation of PCI-7300A_Rev.B/cPCI-7300A_Rev.B *Visual C/C++ program*
	SDK7300aMBufWav	Multiple buffer mode digital input of PCI-7300A_Rev.B/cPCI-7300A_Rev.B through DMA transfer *Visual C/C++ Program*
	SDK7300Int	Interrupt operation of PCI-7300A_Rev.B/cPCI-7300A_Rev.B by Event Status checking and waiting method *Visual C/C++ program*
	SDK7300DbEvt	Interrupt operation of PCI-7300A_Rev.B/cPCI-7300A_Rev.B by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ sample program*
	C7300bDbfDO	Double buffer mode digital output of PCI-7300A_Rev.B/cPCI-7300A_Rev.B through DMA transfer *Visual C/C++ console Program*
	C7300File	1. Digital input of PCI-7300A_Rev.B/cPCI-7300A_Rev.B through DMA transfer 2. Storing the data to disk *Visual C/C++ sample program*

Card Type	Sample Name	Description
PCI-7348 PCI-7396	**SDK7348**	D/I, and D/O of PCI-7348 *Visual C/C++ sample program*
	SDK7348Int	D/I, and D/O of PCI-7348 by Interrupt Event Status checking and waiting method *Visual C/C++ Program*
	SDK7348DbEvt	D/I, and D/O of PCI-7348 by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ Program*
	SDK7348COSi	COS of Interrup operation of D/I, and D/O of PCI-7348 by Interrupt Event Status checking and waiting method *Visual C/C++ sample program*
	SDK7348IntMsg	D/I, and D/O of PCI-7348 by Interrupt Event Message method *Visual C/C++ Program*
	SDK7348DbEvtMsg	D/I, and D/O of PCI- PCI-7348 by Interrupt Event Message method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7348	D/I, and D/O of PCI-7348 *Visual Basic Program*
	SDK7396	D/I, and D/O of PCI-7396 *Visual C/C++ sample program*
	SDK7396Int	D/I, and D/O of PCI-7396 by Interrupt Event Status checking and waiting method *Visual C/C++ Program*
	SDK7396DbEvt	D/I, and D/O of PCI-7396 by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ sample program*
	SDK7396COSi	COS of Interrup operation of D/I, and D/O of PCI-7396 by Interrupt Event Status checking and waiting method *Visual C/C++ Program*
	SDK7396IntMsg	D/I, and D/O of PCI-7396 by Interrupt Event Message method *Visual C/C++ Program*
	SDK7396DbEvtMsg	D/I, and D/O of PCI- PCI-7396 by Interrupt Event Message method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7396	D/I, and D/O of PCI-7396 *Visual Basic Program*
PCI-7432	**SDK7432**	D/I, and D/O of PCI-7432/cPCI-7432 *Visual C/C++ sample program*
	SDK7432Int	D/I, and D/O of PCI-7432/cPCI-7432 by Interrupt Event Status checking and waiting method *Visual C/C++ Program*
	SDK7432DbEvt	D/I, and D/O of PCI-7432/cPCI-7432 by Interrupt Event Status checking and waiting method (Dual Interrupt Events) *Visual C/C++ Program*
	SDK7432IntMsg	D/I, and D/O of PCI-7432/cPCI-7432 by Interrupt Event Message method *Visual C/C++ Program*
	SDK7432DbEvtMsg	D/I, and D/O of PCI-7432/cPCI-7432 by Interrupt Event Message method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7432	D/I, and D/O of PCI-7432/cPCI-7433 *Visual Basic Program*

Card Type	Sample Name	Description
PCI-7433	**SDK7433**	D/I of PCI-7433/cPCI-7433 *Visual C/C++ sample program*
	SDK7433R	D/I of cPCI-7433R *Visual C/C++ sample program*
	SDK7433Int	D/I of PCI-7433/cPCI-7433 through Interrupt operation *Visual C/C++ Program*
	SDK7433DbEvt	D/I of PCI-7433/cPCI-7433 through Interrupt operation (Dual Interrupt Events) *Visual C/C++ Program*
	SDK7433IntMsg	D/I of PCI-7433/cPCI-7433 by Interrupt Event Message method *Visual C/C++ Program*
	SDK7433DbEvtMsg	D/I of PCI-7433/cPCI-7433 by Interrupt Event Message method (Dual Interrupt Events) *Visual C/C++ Program*
	VB7433	D/I of PCI-7433/cPCI-7433 *Visual Basic Program*
PCI-7434	**SDK7434**	D/O of PCI-7434/cPCI-7434 *Visual C/C++ sample program*
	SDK7434R	D/O of cPCI-7434R *Visual C/C++ sample program*
	VB7434	D/O of PCI-7434/cPCI-7434 *Visual Basic Program*
PCI-7442	**C7442TTL_Line**	Programmable D/I and D/O of PCI-7442 *Visual C/C++ Program*
	C7442TTL_Port	Programmable D/I and D/O of PCI-7442 *Visual C/C++ Program*
	CDIOOnePoint	D/I and D/O of PCI-7442 *Visual C/C++ Program*
	CWdtOvflow	Watchdog timer of PCI-7442 *Visual C/C++ Program*
	SDK7442DbEvt	Change-of-state of PCI-7442 *Visual C/C++ Program*
	SDK7442DBEvtMsg	Change-of-state of PCI-7442 *Visual C/C++ Program*
	SDK7442int	Change-of-state of PCI-7442 *Visual C/C++ Program*
	SDK7442intMsg	Change-of-state of PCI-7442 *Visual C/C++ Program*
	SDK7442TTL	Programmable D/I and D/O of PCI-7442 *Visual C/C++ Program*
	SDK7442DIO	D/I and D/O of PCI-7442 *Visual C/C++ Program*
	SDKWdtOvflow	Watchdog Timer of PCI-7442 *Visual C/C++ Program*
	VB7442TTL	Programmable D/I and D/O of PCI-7442 *Visual Basic Program*
	VB7442DIO	D/I and D/O of PCI-7442 *Visual Basic Program*

Card Type	Sample Name	Description
PCI-7443	**C7443TTL_Line**	Programmable D/I and D/O of PCI-7443 *Visual C/C++ Program*
	C7443TTL_Port	Programmable D/I and D/O of PCI-7443 *Visual C/C++ Program*
	CDIOnePoint	D/I of PCI-7443 *Visual C/C++ Program*
	SDK7443intMsg	Change-of-state of PCI-7443 *Visual C/C++ Program*
	SDK7443MultiEvt	Change-of-state of PCI-7443 *Visual C/C++ Program*
	SDK7443MultiEvtMsg	Change-of-state of PCI-7443 *Visual C/C++ Program*
	SDK7443TTL	Programmable D/I and D/O of PCI-7443 *Visual C/C++ Program*
	VB7443TTL	Programmable D/I and D/O of PCI-7443 *Visual C/C++ Program*
PCI-7444	**C7444TTL_Line**	Programmable D/I and D/O of PCI-7444 *Visual C/C++ Program*
	C7444TTL_Port	Programmable D/I and D/O of PCI-7444 *Visual C/C++ Program*
	CDOOnePoint	D/O of PCI-7444 *Visual C/C++ Program*
	CWdtOvflow	Watchdog timer of PCI-7444 *Visual C/C++ Program*
	SDK7444TTL	Programmable D/I and D/O of PCI-7444 *Visual C/C++ Program*
	SDKWdtOvflow	Watch-dog Timer of PCI-7444 *Visual C/C++ Program*
	VB7444TTL	Programmable D/I and D/O of PCI-7444 *Visual Basic Program*
PCI-8554	**SDK8554**	Timer/counter of PCI-8554 *Visual C/C++ sample program*
	SDKEventCnt	Event counter of PCI-8554 *Visual C/C++ sample program*
	VB8554	Timer/counter of PCI-8554 *Visual Basic Program*

Card Type	Sample Name	Description
PCI-9111	**SDK9111**	A/D conversion, D/A conversion, D/I, and D/O of PCI9111 *Visual C/C++ Program*
	SDK9111Int	Analog input of PCI-9111 through Interrupt operation *Visual C/C++ Program*
	SDK9111DbfPreTrg	Pre-trigger with Double buffer mode analog input of PCI-9111 through Interrupt operation *Visual C/C++ Program*
	SDK9111SpreTrg	Pre-trigger with Double buffer mode analog input of PCI-9111 through Interrupt operation *Visual C/C++ Program*
	C9111File	1. Analog input of PCI-9111 through Interrupt operation 2. Storing the data to disk *Visual C/C++ console Program*
	C9111DbfFile	1. Double buffer mode analog input of PCI-9111 through Interrupt operation 2. Storing the data to disk *Visual C/C++ console Program*
	VB9111	A/D conversion, D/A conversion, D/I, and D/O of PCI9111 *Visual Basic Program*
	VB9111Int	Analog input of PCI-9111 through Interrupt operation *Visual Basic Program*
	VB9111PreTrg	Pre-trigger with Double buffer mode analog input of PCI-9111 through Interrupt operation *Visual Basic Program*
	VB9111Scan	Autoscan Analog input of PCI-9111 *Visual Basic Program*
PCI-9112	**SDK9112**	A/D conversion, D/A conversion, D/I, and D/O of PCI9112/cPCI-9112 *Visual C/C++ program*
	SDK9112DMA	Analog input of PCI-9112/cPCI-9112 through DMA data transfer *Visual C/C++ Program*
	SDK9112DbfDma	Double buffer mode analog input of PCI-9112/cPCI-9112 through DMA data transfer *Visual C/C++ sample program*
	C9112File	1. Analog input of PCI-9112 through DMA data transfer 2. Storing the data to disk *Visual C/C++ console Program*
	C9112DbfFile	1. Double buffer mode analog input of PCI-9112 through DMA data transfer 2. Storing the data to disk *Visual C/C++ console Program*
	VB9112	A/D conversion, D/A conversion, D/I, and D/O of PCI9112/cPCI-9112 *Visual Basic Program*
	VB9112DbfDma	Double buffer mode analog input of PCI-9112/cPCI-9112 through DMA data transfer *Visual Basic Program*

Card Type	Sample Name	Description
PCI-9113	**SDK9113**	A/D conversion, D/A conversion, D/I, and D/O of PCI-9113 *Visual C/C++ Program*
	SDK9113Int	Analog input of PCI-9113 through Interrupt operation *Visual C/C++ Program*
	SDK9113DbfInt	Double buffer mode analog input of PCI-9113 through Interrupt operation *Visual C/C++ sample program*
	C9113File	1. Analog input of PCI-9113 through Interrupt operation 2. Storing the data to disk *Visual C/C++ console Program*
	C9113DbfFile	1. Double buffer mode analog input of PCI-9113 through Interrupt operation 2. Storing the data to disk *Visual C/C++ console program*
	VB9113	A/D conversion, D/A conversion, D/I, and D/O of PCI-9113 *Visual Basic Program*
	VB9113Int	Analog input of PCI-9113 through Interrupt operation *Visual Basic Program*
	VB9113Scan	Autoscan Analog input of PCI-9113 *Visual Basic Program*
PCI-9114	**SDK9114**	A/D conversion, D/A conversion, D/I, and D/O of PCI-9114 *Visual C/C++ Program*
	SDK9114Int	Analog input of PCI-9114 through Interrupt operation *Visual C/C++ Program*
	SDK9114DbfInt	Double buffer mode analog input of PCI-9114 through Interrupt operation *Visual C/C++ sample program*
	C9114File	1. Analog input of PCI-9114 through Interrupt operation 2. Storing the data to disk *Visual C/C++ console Program*
	C9114DbfFile	1. Double buffer mode analog input of PCI-9114 through Interrupt operation 2. Storing the data to disk *Visual C/C++ console Program*
	VB9114	A/D conversion, D/A conversion, D/I, and D/O of PCI-9114 *Visual Basic Program*
	VB9114Int	Analog input of PCI-9114 through Interrupt operation *Visual Basic Program*
	VB9114Scan	Autoscan Analog input of PCI-9114 *Visual Basic Program*

Card Type	Sample Name	Description
cPCI-9116	**SDK9116**	A/D conversion of cPCI-9116 *Visual C/C++ Program*
	SDK9116ScanDma	Software trigger with Single buffer mode analog input of cPCI-9116 through DMA data transfer *Visual C/C++ Program*
	SDK9116PostTrg	Post trigger with Single buffer mode analog input of cPCI-9116 through DMA data transfer *Visual C/C++ Program*
	SDK9116MidTrg	Middle trigger with Single buffer mode analog input of cPCI-9116 through DMA data transfer *Visual C/C++ Program*
	SDK9116DlyTrg	Delay trigger with Single buffer mode analog input of cPCI-9116 through DMA data transfer *Visual C/C++ Program*
	SDK9116DbfDma	Double buffer mode analog input of cPCI-9116 through DMA data transfer *Visual C/C++ Program*
	SDK9116DbfAboutTrg	Middle trigger with Double buffer mode analog input of cPCI-9116 through DMA data transfer *Visual C/C++ Program*
	SDK9116DbfPostTrg	Post trigger with Double buffer mode analog input of cPCI-9116 through DMA data transfer *Visual C/C++ Program*
	SDK9116DbfDlyTrg	Delay trigger with Double buffer mode analog input of CPCI-9116 through DMA data transfer *Visual C/C++ Program*
	C9116File	1. Analog input of cPCI-9116 through DMA data transfer 2. Storing the data to disk *Visual C/C++ console Program*
	C9116DbfFile	1. Double buffer mode analog input of cPCI-9116 through DMA data transfer 2. Storing the data to disk *Visual C/C++ console Program*
	VB9116	Analog input of CPCI-9116 through DMA data transfer *Visual Basic Program*
PCI-9118	**SDK9118**	A/D conversion, D/A conversion, D/I, and D/O of PCI-9118 *Visual C/C++ Program*
	SDK9118DbfAboutTrg	About trigger with Double buffer mode analog input of PCI-9118 through DMA data transfer *Visual C/C++ Program*
	SDK9118BurstDma	Analog input of PCI-9118 through Burst Mode DMA data transfer *Visual C/C++ Program*
	SDK9118DbfDma	Double buffer mode analog input of PCI-9118 through DMA data transfer *Visual C/C++ Program*

Card Type	Sample Name	Description
PCI-9118	**SDK9118HRDbfDma**	Double buffer mode analog input of PCI-9118HR through DMA data transfer *Visual C/C++ Program*
	SDK9118ScanDma	Autoscan Analog input of PCI-9118 through DMA data transfer *Visual C/C++ Program*
	SDK9118HRScanDma	Autoscan Analog input of PCI-9118HR through DMA data transfer *Visual C/C++ Program*
	SDK9118DbfPreTrg	Pre-trigger with Double buffer mode analog input of PCI-9118 through DMA data transfer *Visual C/C++ Program*
	SDK9118DbfPostTrg	Post trigger with Double buffer mode analog input of PCI-9118 through DMA data transfer *Visual C/C++ sample program*
	SDK9118AboutTrg	About trigger with Single buffer mode analog input of PCI-9118 through DMA data transfer *Visual C/C++ Program*
	SDK9118HRAboutTrg	About trigger with Single buffer mode analog input of PCI-9118HR through DMA data transfer *Visual C/C++ Program*
	SDK9118PostTrg	Post trigger with Single buffer mode analog input of PCI-9118 through DMA data transfer *Visual C/C++ Program*
	C9118File	1. Analog input of PCI-9118 through DMA data transfer 2. Storing the data to disk *Visual C/C++ console Program*
	C9118DbfFile	1. Double buffer mode analog input of PCI-9118 through DMA data transfer 2. Storing the data to disk *Visual C/C++ console Program*
	VB9118DgHr	A/D conversion, D/A conversion, D/I, and D/O of PCI9118DG/HR *Visual Basic Program*
	VB9118Hg	A/D conversion, D/A conversion, D/I, and D/O of PCI9118HG *Visual Basic Program*
	VB9118AboutTrg	About trigger with Single buffer mode analog input of PCI-9118 through DMA data transfe *Visual Basic Program*
	VB9118PostTrg	Post trigger with Single buffer mode analog input of PCI-9118 through DMA data transfe *Visual Basic sample program*
	VB9118Dma	Analog input of PCI-9118 through DMA data transfer *Visual Basic Program*

Card Type	Sample Name	Description
PCI-9221	**C9221AIDma**	A/D conversion of PCI-9221 *Visual C/C++ Program*
	C9221AIDma_ExtD	A/D conversion of PCI-9221 with external trigger *Visual C/C++ Program*
	C9221AIDmaToFile	A/D conversion of PCI-9221 Stores acquired data to a disk file *Visual C/C++ Program*
	C9221AIPoll	A/I Polling of PCI-9221 *Visual C/C++ Program*
	C9221AIPoll_MultiChn	A/I Polling of PCI-9221 for multiple channels *Visual C/C++ Program*
	C9221AO	A/O of PCI-9221 *Visual C/C++ Program*
	C9221Cal	Calibration of PCI-9221 *Visual C/C++ Program*
	C9221DIO_Line	D/I and D/O of PCI-9221 *Visual C/C++ Program*
	C9221DIO_Port	D/I and D/O of PCI-9221 *Visual C/C++ Program*
	C9221GPTC	General-Purpose Timer/Counter of PCI-9221 *Visual C/C++ Program*
	SDK9221AIDma	A/D conversion of PCI-9221 *Visual C/C++ Program*
	SDK9221AIDmaDbf	Double buffer mode analog input of PCI-9221 through DMA data transfer *Visual C/C++ Program*
	SDK9221AIDmaDbfCallBack	Double buffer mode analog input of PCI-9221 through DMA data transfer *Visual C/C++ Program*
	SDK9221DIO	D/I and D/O of PCI-9221 *Visual C/C++ Program*
	VB9221AIDma	A/D conversion of PCI-9221 *Visual Basic Program*

Card Type	Sample Name	Description
PCI-9222	**C9222_AI_Polling**	A/D Polling *Visual C/C++ Program*
	C9222_AI_DMA	A/D Continuous Conversion *Visual C/C++ Program*
	C9222_AI_DMA_Event	A/D Continuous Conversion with AIEnd Event Call Back *Visual C/C++ Program*
	C9222_AI_DMA_ExtDtoFile	A/D Continuous Conversion with External Digital Trigger *Visual C/C++ Program*
	C9222_AI_DMA_ReTrigger	A/D Continuous Conversion with External Digital Re-trigger Visual C/C++ Program
	C9222_AI_DMA_InfiniteReTrigger	A/D Continuous Conversion with Infinite External Digital Re-trigger *Visual C/C++ Program*
	C9222_AI_DMA_ReTrigEvent	A/D Continuous Conversion with External Digital Re-trigger and TrigEvent/AIEnd Event Call Back *Visual C/C++ Program*
	C9222_AI_DMA_GatedTrigger	A/D Continuous Conversion with Gated Trigger Mode *Visual C/C++ Program*
	C9222_AI_DMA_DBFtoFile	A/D Infinite Conversion with Double Buffer Mode *Visual C/C++ Program*
	C9222_AI_DMA_DBFOverrun	A/D Infinite Conversion with Double Buffer Mode and Double Buffer Overrun Check *Visual C/C++ Program*
	C9222_AI_DMA_DBFEvent	A/D Infinite Conversion with Double Buffer Mode and DBEvent/AIEnd Event Call Back *Visual C/C++ Program*
	C9222_AO_Polling	D/A Polling *Visual C/C++ Program*
	C9222_AO_DMA_SimuWrites	D/A Simultaneous Continuous Update *Visual C/C++ Program*
	C9222_AO_DMA_WaveRepeat	D/A Continuous Update with Waveform Repeat *Visual C/C++ Program*
	C9222_AO_DMA_ReTrigEvent	D/A Continuous Update with External Digital Re-trigger Mode and TrigEvent/AOEnd Event Call Back *Visual C/C++ Program*
	C9222_AO_DMA_InfiniteDReTrigger	D/A Continuous Update with Infinite External Digital Re-trigger *Visual C/C++ Program*
	C9222_AO_DMA_DBF	D/A Infinite Continuous Update with Double Buffer Mode *Visual C/C++ Program*
	C9222_AO_DMA_DBFEvent	D/A Infinite Continuous Update with Double Buffer Mode and DBEvent/AOEnd Event Call Back *Visual C/C++ Program*
	C9222_DIO_Polling	DIO Polling *Visual C/C++ Program*
	C9222_DI_DMA	DI Continuous Conversion *Visual C/C++ Program*
	C9222_DI_DMA_Retrigger	DI Continuous Conversion with External Digital Re-trigger *Visual C/C++ Program*
	C9222_DI_DMA_InfiniteReTrigger	DI Continuous Conversion with Infinite External Digital Re-trigger *Visual C/C++ Program*

Card Type	Sample Name	Description
PCI-9222 (cont.)	**C9222_DI_DMA_DBFtoFile**	DI Infinite Continuous Conversion with Double Buffer Mode *Visual C/C++ Program*
	C9222_DI_DMA_DBFEvent	DI Infinite Continuous Conversion with Double Buffer Mode and DBEvent/DIEnd Event Call Back *Visual C/C++ Program*
	C9222_DO_DMA	DO Continuous Update *Visual C/C++ Program*
	C9222_DO_DMA_Event	DO Continuous Update with DOEnd Event Call Back *Visual C/C++ Program*
	C9222_DO_DMA_DelayTrigger	DO Continuous Update with Delay Trigger *Visual C/C++ Program*
	C9222_DO_DMA_ReTrigEvent	DO Continuous Update with External Digital Re-trigger and TrigEvent/DOEnd Event Call Back *Visual C/C++ Program*
	C9222_DO_DMA_InfiniteReTrigger	DO Continuous Update with Infinite External Digital Re-trigger *Visual C/C++ Program*
	C9222_Encoder	CW/CCW, x1, x2, x4 AB Phase Encoder *Visual C/C++ Program*
	C9222_Encoder_Event	CW/CCW, x1, x2, x4 AB Phase Encoder with Encoder Position Trig-ger/Event *Visual C/C++ Program*
	C9222_GPTC	General Purpose Timer/Counter *Visual C/C++ Program*
	C9222_Calibration	A/D and D/A Auto Calibration *Visual C/C++ Program*

Card Type	Sample Name	Description
PCI-9223	**C9223_AI_Polling**	A/D Polling *Visual C/C++ Program*
	C9223_AI_DMA	A/D Continuous Conversion *Visual C/C++ Program*
	C9223_AI_DMA_Event	A/D Continuous Conversion with AIEnd Event Call Back *Visual C/C++ Program*
	C9223_AI_DMA_ExtDtoFile	A/D Continuous Conversion with External Digital Trigger *Visual C/C++ Program*
	C9223_AI_DMA_ReTrigger	A/D Continuous Conversion with External Digital Re-trigger Visual C/C++ Program
	C9223_AI_DMA_InfiniteReTrigger	A/D Continuous Conversion with Infinite External Digital Re-trigger *Visual C/C++ Program*
	C9223_AI_DMA_ReTrigEvent	A/D Continuous Conversion with External Digital Re-trigger and TrigEvent/AIEnd Event Call Back *Visual C/C++ Program*
	C9223_AI_DMA_GatedTrigger	A/D Continuous Conversion with Gated Trigger Mode *Visual C/C++ Program*
	C9223_AI_DMA_DBFtoFile	A/D Infinite Conversion with Double Buffer Mode *Visual C/C++ Program*
	C9223_AI_DMA_DBFOverrun	A/D Infinite Conversion with Double Buffer Mode and Double Buffer Overrun Check *Visual C/C++ Program*
	C9223_AI_DMA_DBFEvent	A/D Infinite Conversion with Double Buffer Mode and DBEvent/AIEnd Event Call Back *Visual C/C++ Program*
	C9223_AO_Polling	D/A Polling *Visual C/C++ Program*
	C9223_AO_DMA_SimuWrites	D/A Simultaneous Continuous Update *Visual C/C++ Program*
	C9223_AO_DMA_WaveRepeat	D/A Continuous Update with Waveform Repeat *Visual C/C++ Program*
	C9223_AO_DMA_ReTrigEvent	D/A Continuous Update with External Digital Re-trigger Mode and TrigEvent/AOEnd Event Call Back *Visual C/C++ Program*
	C9223_AO_DMA_InfiniteDReTrigger	D/A Continuous Update with Infinite External Digital Re-trigger *Visual C/C++ Program*
	C9223_AO_DMA_DBF	D/A Infinite Continuous Update with Double Buffer Mode *Visual C/C++ Program*
	C9223_AO_DMA_DBFEvent	D/A Infinite Continuous Update with Double Buffer Mode and DBEvent/AOEnd Event Call Back *Visual C/C++ Program*
	C9223_DIO_Polling	DIO Polling *Visual C/C++ Program*
	C9223_DI_DMA	DI Continuous Conversion *Visual C/C++ Program*
	C9223_DI_DMA_Retrigger	DI Continuous Conversion with External Digital Re-trigger *Visual C/C++ Program*
	C9223_DI_DMA_InfiniteReTrigger	DI Continuous Conversion with Infinite External Digital Re-trigger *Visual C/C++ Program*

Card Type	Sample Name	Description
PCI-9223 (cont.)	**C9223_DI_DMA_DBFtoFile**	DI Infinite Continuous Conversion with Double Buffer Mode *Visual C/C++ Program*
	C9223_DI_DMA_DBFEvent	DI Infinite Continuous Conversion with Double Buffer Mode and DBEvent/DIEnd Event Call Back *Visual C/C++ Program*
	C9223_DO_DMA	DO Continuous Update *Visual C/C++ Program*
	C9223_DO_DMA_Event	DO Continuous Update with DOEnd Event Call Back *Visual C/C++ Program*
	C9223_DO_DMA_DelayTrigger	DO Continuous Update with Delay Trigger *Visual C/C++ Program*
	C9223_DO_DMA_ReTrigEvent	DO Continuous Update with External Digital Re-trigger and TrigEvent/DOEnd Event Call Back *Visual C/C++ Program*
	C9223_DO_DMA_InfiniteReTrigger	DO Continuous Update with Infinite External Digital Re-trigger *Visual C/C++ Program*
	C9223_Encoder	CW/CCW, x1, x2, x4 AB Phase Encoder *Visual C/C++ Program*
	C9223_Encoder_Event	CW/CCW, x1, x2, x4 AB Phase Encoder with Encoder Position Trig-ger/Event *Visual C/C++ Program*
	C9223_GPTC	General Purpose Timer/Counter *Visual C/C++ Program*
	C9223_Calibration	A/D and D/A Auto Calibration *Visual C/C++ Program*

Card Type	Sample Name	Description
PCI-9524	AI_Polling	A/D Fast Polling *Visual C/C++ Program*
	AI_ScanChannels	A/D Fast Channels-Scan Polling *Visual C/C++ Program*
	AI_ContDma	A/D Continuous Conversion *Visual C/C++ Program*
	AI_ContDmaFile	A/D Continuous Conversion *Visual C/C++ Program*
	AI_ContDmaExtD	A/D Continuous Conversion with External Digital Trigger *Visual C/C++ Program*
	AI_ContScans	A/D Continuous Channels-Scan Conversion *Visual C/C++ Program*
	AI_ContScansFile	A/D Continuous Channels-Scan Conversion *Visual C/C++ Program*
	AI_ContScansExtTrig	A/D Continuous Channels-Scan Conversion with QD/PG Internal Trigger *Visual C/C++ Program*
	AI_ContDbfDma	A/D Infinite Continuous Conversion with Double Buffer Mode *Visual C/C++ Program*
	AI_ContDbfDmaFileExtD	A/D Infinite Continuous Conversion with Double Buffer Mode and External Digital Trigger *Visual C/C++ Program*
	AI_ContDbfScansFile	A/D Infinite Continuous Channels-Scan Conversion with Double Buffer Mode *Visual C/C++ Program*
	AO_PollUpdate	D/A Polling *Visual C/C++ Program*
	AO_SimuUpdate	D/A Simultaneous Polling *Visual C/C++ Program*
	DIO_Polling	DIO Polling *Visual C/C++ Program*
	CTR_PulseGenration	CLK/DIR, CW/CCW Pulse Generation *Visual C/C++ Program*
	CTR_QuadratureDecoder	X4 AB Phase Decoder *Visual C/C++ Program*
	Calibration	D/A Infinite Continuous Update with Double Buffer Mode *Visual C/C++ Program*

Card Type	Sample Name	Description
PCI-9812	**SDK9812SoftTrg**	Software trigger with Single buffer mode analog input of PCI-9812/ cPCI-9812 through DMA data transfer *Visual C/C++ Program*
	SDK9812PreTrg	Pre-trigger with Single buffer mode analog input of PCI-9812/cPCI-9812 through DMA data transfer *Visual C/C++ Program*
	SDK9812PostTrg	Post trigger with Single buffer mode analog input of PCI-9812/cPCI-9812 through DMA data transfer *Visual C/C++ Program*
	SDK9812MidTrg	Middle trigger with Single buffer mode analog input of PCI-9812/cPCI-9812 through DMA data transfer *Visual C/C++ Program*
	SDK9812DelayTrg	Delay trigger with Single buffer mode analog input of PCI-9812/cPCI-9812 through DMA data transfer *Visual C/C++ Program*
	SDK9812DbfMidTrg	Middle trigger with Double buffer mode analog input of PCI-9812/ cPCI-9812 through DMA data transfer *Visual C/C++ Program*
	SDK9812DbfPreTrg	Pre-trigger with Double buffer mode analog input of PCI-9812/cPCI-9812 through DMA data transfer *Visual C/C++ Program*
	SDK9812DbfPostTrg	Post trigger with Double buffer mode analog input of PCI-9812/cPCI-9812 through DMA data transfer *Visual C/C++ Program*
	SDK9812DbfDelayTrg	Delay trigger with Double buffer mode analog input of PCI-9812/cPCI-9812 through DMA data transfer *Visual C/C++ Program*
	C9812File	1. Analog input of PCI-9812/10 through DMA data transfer 2. Storing the data to disk *Visual C/C++ console Program*
	C9812DbfFile	1. Double buffer mode analog input of PCI-9812/10 through DMA data transfer 2. Storing the data to disk *Visual C/C++ console Program*
	VB9812	Analog input of PCI-9812/cPCI-9812 through DMA data transfer *Visual Basic 4.0 Program*
	9812 VB5	Analog input of PCI-9812/cPCI-9812 through DMA data transfer *Visual Basic 5.0 Program*

NOTE The PCIS-DASK comes with Examples.exe - a sample program browser that allows you to view and execute all bundled sample programs. Examples.exe is located in **<InstallDir>\Samples** directory. After launching Examples.exe, double-click the icon of the sample you want to execute.

7.2 Development Environments

Visual Basic Sample Programs

Several Visual Basic sample programs are provided for each card. Using VB9112DMA as example, the following files are included in each sample program:

- VB project file - VB9112D.VBP
- VB form files - VB9112D.FRM
- Executable file - VB9112D.EXE

You must install a 32-bit Microsoft® Visual Basic 4.0 Professional Edition or higher to view these sample programs. Refer to Microsoft® Visual Basic 4.0 Professional Edition manual or related reference books to get the information on using Visual Basic 4.0.

If you want to execute the VB sample programs without installing Microsoft® Visual Basic 4.0, use the VB4 Runtime package. The VB4 Runtime package includes the required library and DLL files to run the VB sample programs. You can find this package from the main setup window or root directory of the ADLINK All-In-One CD.

Microsoft C/C++ Sample Programs

The PCIS-DASK also includes Microsoft® C/C++ sample programs featuring similar functions as those provided by VB samples. These may be directly executed and do not require installation of any additional package. It is recommended that you use Microsoft® C/C++ sample programs when testing the PCIS-DASK packages.

Using SDK7200WAV as example, the following files are included in each sample program:

- C source file - 7200WAV.C
- Workspace file - 7200WAV.MDP
- Resource script file - 7200WAV.RC, RESOURCE.H
- Make file - 7200WAV.MAK
- Executable file - 7200WAV.EXE

You can use any Microsoft® Visual C++ 4.0 editor to view or modify these source files. However, you must install Microsoft® Visual C++ 4.0 or higher to build the executable 7200WAV.EXE. Refer to the Microsoft® Visual C++ manual or related reference books for additional information.

7.3 Execute Sample Programs

To run the sample programs:

1. Open the sample program

   You can use Microsoft Visual C++ 4.0 or Visual Basic 4.0 to open and execute the sample programs. Or you can run the executable files directly.

2. Set the testing parameters

   Depending on your requirements, set the testing parameters such as A/D or D/A conversion, testing channels, sampling rate, transfer count, etc.

3. Click the **Start** button to run the program.

7.4 Detailed Descriptions of Programs

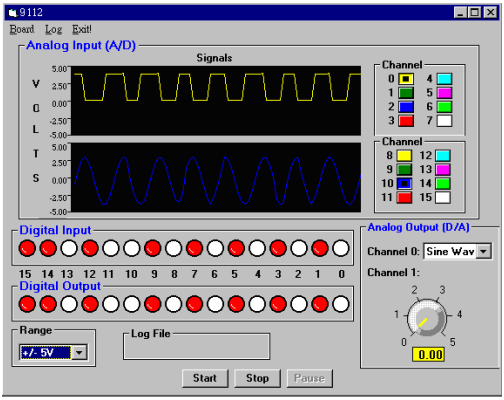
Four types of sample programs are provided together with the PCIS-DASK software driver:

- AD conversion, D/A conversion, and D/O
- Data I/O through DMA Data Transfer or Interrupt operation
- Double buffer mode data I/O through DMA transfer or Interrupt operation
- Trigger Mode Data I/O through DMA Data Transfer or Interrupt operation

NOTE The following sections describing these types of sample programs use the VB 9112, SDK 9112DMA, SDK 9112CDMA and SDK 9118 DbfPreTrg screens as examples.

A/D Conversion, D/A Conversion, D/I, and D/O

This sample illustrates how to use the PCIS-DASK to operate software trigger with program polling data mode and read/write data from digital input/output channels on PCI-9112. The main program main screen is shown below:



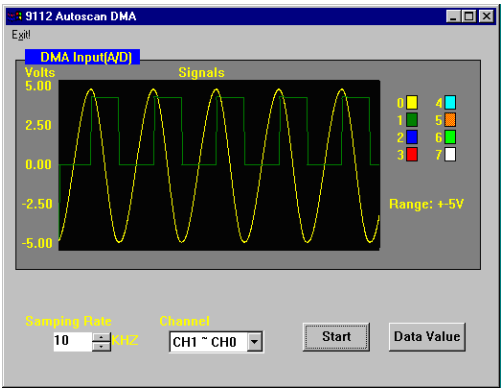
Analog Input (A/D). Shows the results of A/D conversion. You can select the input channels (allows multiple channels) and the input range (gain) you want to test.

Analog output (D/A). Shows the results of D/A conversion. You may turn the tuner to set the output voltage. You can also set the output waveform to sine or square.

D/I and D/O. Shows the results of read/write data from/to digital input/output channels. To set the output value, click the channel buttons. A red color indicates an ON channel, while a white color indicates an OFF channel.

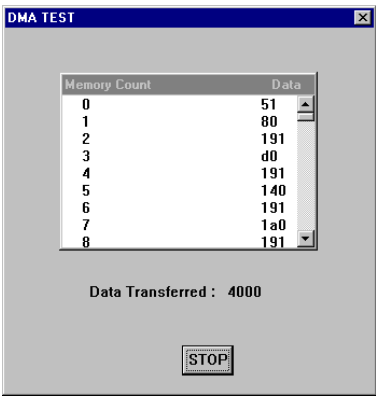
Data I/O Through DMA Data Transfer or Interrupt Operation

This program demonstrates the use of PCIS-DASK to operate data I/O through DMA data transfer or Interrupt operation. The program main screen is shown below.



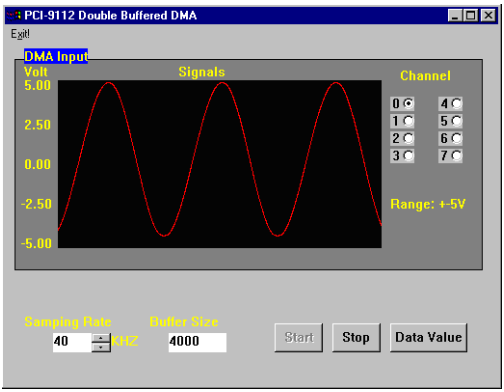
These programs allow you to adjust the input channels, input range (PCI-7200 does not have these two options), sampling rate, and data size (transfer count).

To view the input data, click on the **Data Value** button in the main screen when data transfer is finished. Refer to the following screen.



Double Buffer Mode Data I/O Through DMA Transfer or Interrupt Operation

This program tells you how to use PCIS-DASK to operate double-buffered data I/O through DMA transfer or Interrupt operation. The program main screen is shown below:



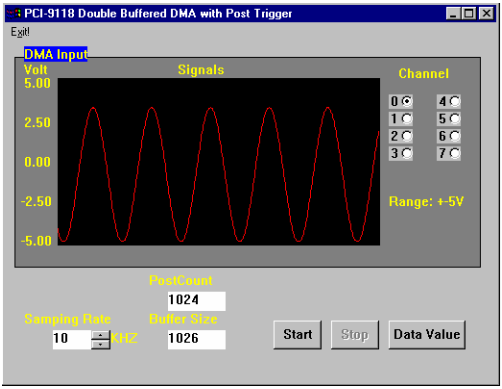
These programs lets you input channel, input range (PCI-7200 does not have this two options), sampling rate, and data size (transfer count).

To view the input data, click on the **Data Value** button in the main screen after you stop the double-buffered operation.

Trigger Mode Data I/O Through DMA Data Transfer or Interrupt Operation

These programs tell you how to use PCIS-DASK to operate trigger mode data I/O through DMA data transfer or interrupt operation. Except for the additional input item **postCount**, the main screen of these programs are similar with Single-Buffer Mode or Double-Buffer Mode programs. Please refer to these two sections for the details.

The **postCount** item represents the number of data accessed after a specific trigger event or the counter value for deferring to access data after a specific trigger event. Refer to the description of AI configuration functions (AI_9111_Config, AI_9118_Config, AI_9812_Config) for details.



NOTE Except for VB9812, all trigger mode data acquisition sample programs use external digital trigger source to provide trigger signal. Refer to the card's documentation if you you want to operate for the detailed description of trigger mode data acquisition.

8 Distribution of Applications

8.1 Required Files

When installing an application that uses PCIS-DASK on another computer, you must install the necessary driver files and supporting libraries on the target system. You can create an automatic installer that installs your program and all files needed to run the program or you can install the program and program files manually For both installation methods, you must install the following files:

Required support DLLs: **Pci-dask.dll**

Driver files

Windows® 98

- Corresponding driver files in **\Software\Pcis-dask\W98NT2K \redist\W98\drivers** (e.g. pci7200.sys for PCI-7200). These files must be copied to the **Windows\system32\drivers** directory.
- Corresponding INF files in **\Software\Pcis-dask\W98NT2K \redist\W98\Inf** (e.g. p7200.inf for PCI-7200). These files must be copied to the **Windows\inf** directory.
- Device configuration utility in **\Software\Pcis-dask\W98NT2K\redist\W98\Util**.

Windows® NT 4.0

- **adldask.sys** in **\Software\Pcis-dask\W98NT2K\redist\Wnt\drivers**. This file must be copied to **Winnt\system32\drivers** directory.
- Corresponding driver files in **\Software\Pcis-dask\W98NT2K \redist\Wnt\drivers** (e.g. pci7200.sys for PCI-7200). These files must be copied to **Winnt\system32\drivers** directory.
- Device configuration utility in **\Software\Pcis-dask\W98NT2K/redist\Wnt\Util**.

Windows® 2000

- Corresponding driver file in **\Software\Pcis-dask\W98NT2K \redist\W2000\drivers** (e.g. pci7200.sys

for PCI-7200). These files must be copied to **Winnt\system32\drivers** directory.

- Corresponding INF file in **\Software\Pcis-dask\W98NT2K \redist\W2000\Inf** (e.g. p7200.inf for PCI-7200). These files must be copied to **Winnt\inf** directory.
- Device configuration utility in **\Software\Pcis-dask\W98NT2K \redist\W2000\Util**.

Utility file (option)

- Data conversion utility **DAQCvt.exe** in **\Software\Pcis-dask\ W98NT2K\redist\W98\Util**, **\Software\Pcis-dask\W98NT2K\redist\Wnt\Util**, or **\Software\Pcis-dask\W98NT2K\redist\W2000\Util** to convert the binary data file to an easily read file format.

8.2 Automatic Installers

Several programming environments provide setups or distribution kit tools that automatically create an installation program so that it can be conveniently installed from one computer to another. For the application to function properly, this tool must locate and include the required control files and supporting libraries in the installation program that it creates.

Some tools, such as the Visual Basic 5 Setup Wizard, uses dependency files to determine which libraries are required by a VB application. Some setup tools may not automatically recognize which files are required by a program, but they provide an option to add additional files to the installation program. In this case, verify that all the necessary files described in the previous section are included. The user should also check if the resulting installation program does not copy older versions of a file over a newer version on the target computer.

If the programming environment does not provide a tool or wizard for building an installation program, third-party tools such as InstallShield may be used instead. Some programming environments provide simplified or trial versions of third-party installer creation tools on their installation CDs.

8.3 Manual Installation

If the programming environment does not include a setup or distribution kit tool, the installation task may be performed manually. To install the program to another computer:

1. Copy the program executable to the target computer.
2. Copy all required PCIS-DASK files described in the section 8.1 to the appropriate directory on the target computer.
3. Use NuDAQ Device Configuration utility to configure the device.

NOTE Do not replace a newer version of a file installed in the target computer.