PCIS-DASK/X ver4.01

Data Acquisition Software Development Kit

for PC Compatibles

User's Guide

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

@Copyright 1997-2003 ADLink Technology Inc.
All Rights Reserved.

Manual Rev 4.01 : February 28, 2003

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 the manufacturer.

Trademarks

IBM PC is a registered trademark of International Business Machines Corporation. Intel is a registered trademark of Intel Corporation. Other product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

CONTENTS

HOW TO USE THIS MANUAL IV

INTRODUCTION TO PCIS-DASK 1

1.1 ABOUT THE PCIS-DASK SOFTWARE 1
1.2 PCIS-DASK HARDWARE SUPPORT 2
1.3 PCIS-DASK LANGUAGE SUPPORT 3

THE FUNDAMENTALS OF BUILDING APPLICATIONS WITH PCIS-DASK 4

2.1 CREATING A PCIS-DASK APPLICATION USING C/C++ 4

PCIS-DASK UTILITIES 5

3.1 NUDAQ CONFIGURATION UTILITY (DASK_CONF) 5
3.2 PCIS-DASK MODULE INSTALLATION SCRIPT 8
3.3 PCIS-DASK UN-INSTALLATION SCRIPT 9
3.4 PCIS-DASK DATA FILE CONVERTER UTILITY (DAQCVT) 10

PCIS-DASK OVERVIEW 12

4.1 GENERAL CONFIGURATION FUNCTION GROUP 13
4.2 ACTUAL SAMPLING RATE FUNCTION GROUP 13
4.3 ANALOG INPUT FUNCTION GROUP 13
4.3.1 Analog Input Configuration Functions 13
4.3.2 One-Shot Analog Input Functions 15
4.3.3 Continuous Analog Input Functions 15
4.3.4 Asynchronous Analog Input Monitoring Functions 17
4.4 ANALOG OUTPUT FUNCTION GROUP 17
4.4.1 Analog output Configuration Functions 17
4.4.2 One-Shot Analog Output Functions 18
4.5 DIGITAL INPUT FUNCTION GROUP 19

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.5.1 Digital Input Configuration Functions 19
4.5.2 One-Shot Digital Input Functions 19
4.5.3 Continuous Digital Input Functions 20
4.5.4 Asynchronous Digital Input Monitoring Functions 20
4.6 DIGITAL OUTPUT FUNCTION GROUP 21
4.6.1 Digital Output Configuration Functions 21
4.6.2 One-Shot Digital Output Functions 22
4.6.3 Continuous Digital Output Functions 22
4.6.4 Asynchronous Digital Output Monitoring Functions 23
4.7 TIMER/COUNTER FUNCTION GROUP 23
4.7.1 Timer/Counter Functions 23
4.7.2 The General-Purpose Timer/Counter Functions 23
GCTR_CLEAR CLEARS THE GENERAL-PURPOSE TIMER/COUNTER CONTROL REGISTER AND COUNTER REGISTER. 24
4.8 DIO FUNCTION GROUP 24
4.8.1 Digital Input/Output Configuration Functions 24
4.8.2 Dual-Interrupt System Setting Functions 24
PCIS-DASK APPLICATION HINTS 25
5.1 ANALOG INPUT PROGRAMMING HINTS 26
5.1.1 One-Shot Analog input programming Scheme 27
5.1.2 Synchronous Continuous Analog input programming Scheme 28
5.1.3 Non-Trigger Non-double-buffered Asynchronous Continuous Analog input programming Scheme 30
5.1.4 Non-Trigger Double-buffered Asynchronous Continuous Analog input programming Scheme 32
5.1.5 Trigger Mode Non-double-buffered Asynchronous Continuous Analog input programming Scheme 35
5.1.6 Trigger Mode Double-buffered Asynchronous Continuous Analog input programming Scheme 38
5.2 ANALOG OUTPUT PROGRAMMING HINTS 41

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3 DIGITAL INPUT PROGRAMMING HINTS 42
5.3.1 One-Shot Digital input programming Scheme 44
5.3.2 Synchronous Continuous Digital input programming Scheme 45
5.3.3 Non-double-buffered Asynchronous Continuous Digital input programming Scheme 46
5.3.4 Double-buffered Asynchronous Continuous Digital input programming Scheme 48
5.4 DIGITAL OUTPUT PROGRAMMING HINTS 51
5.4.1 One-Shot Digital output programming Scheme 53
5.4.2 Synchronous Continuous Digital output programming Scheme 54
5.4.3 Asynchronous Continuous Digital output programming Scheme 55
5.4.4 Pattern Generation Digital output programming Scheme 57
5.5 INTERRUPT ASYNCHRONOUS NOTIFICATION PROGRAMMING HINTS 58
CONTINUOUS DATA TRANSFER IN PCIS-DASK 60
6.1 CONTINUOUS DATA TRANSFER MECHANISM 60
6.2 DOUBLE-BUFFERED AI/DI OPERATION 61
6.2.1 Double Buffer Mode Principle 61
Single-Buffered Versus Double-Buffered Data Transfer 62
6.3 TRIGGER MODE DATA ACQUISITION FOR ANALOG INPUT 64

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

How to Use This Manual

This manual is to help you use the PCIS-DASK software driver for NuDAQ PCI-bus data acquisition cards. The manual describes how to install and use the software library to meet your requirements and help you program your own software applications. It is organized as follows:

- Chapter 1, "Introduction to PCIS-DASK" describes the hardware and language support of PCIS-DASK.
- Chapter 2,"The Fundamentals of Building Linux Applications with PCIS-DASK" describes the fundamentals of creating applications under Linux environment.
- Chapter 3, "PCIS-DASK Utilities" describes the utilities PCIS-DASK provides.
- Chapter 4, "PCIS-DASK Overview" describes the classes of functions in PCIS-DASK and briefly describes each function.
- Chapter 5, "PCIS-DASK Application Hints" provides the programming schemes showing the function flow of that PCIS-DASK performs analog I/O and digital I/O.
- Chapter 6, "Continuous Data Transfer in PCIS-DASK" describes the mechanism and techniques that PCIS-DASK uses for continuous data transfer.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1

Introduction to PCIS-DASK

1.1 About the PCIS-DASK Software

PCIS-DASK is a software driver for NuDAQ PCI-bus data acquisition cards. It is a high performance data acquisition driver for developing custom applications under Linux environment.

Using PCIS-DASK also lets you take advantage of the power and features of Linux for your data acquisition applications. These include running multiple applications and using extended memory.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1.2 PCIS-DASK Hardware Support

ADLink will periodically upgrade PCIS-DASK for new NuDAQ PCI-bus data acquisition cards and NuIPC CompactPCI cards. Please refer to *Release Notes* for the cards that the current PCIS-DASK actually supports. The following cards are those that PCIS-DASK supports currently:

- PCI-6208A : 8-channel 16-bit current output card
- PCI-6208V/16V : 8/16-channel 16-bit voltage output card
- PCI-6308A : Isolated 8-channel voltage and current output card
- PCI-6308V : Isolated 8-channel voltage output card
- PCI-7200/cPCI-7200 : high-speed 32-bit digital I/O card with bus mastering DMA transfer capability
- PCI-7230/cPCI-7230 : 32-channel isolated digital I/O card
- PCI-7233/PCI-7233H : Isolated 32 channels DI card with COS detection
- PCI-7234 : 32-channel isolated digital output card
- PCI-7248/cPCI-7248 : 48-bit digital I/O card
- cPCI-7249R : 3U CompactPCI 48 parallel digital I/O card
- PCI-7250 : 8 relay output and 8 isolated input card
- cPCI-7252 : 8 relay output and 16 isolated input card
- PCI-7256 : 16 latching relay and 16 isolated input card
- PCI-7258 : 32 PhotoMOS relay output and 2 isolated input
- PCI-7296 : 96-bit digital I/O card
- PCI-7300A/cPCI-7300A : 40 Mbytes/sec Ultra-high speed 32 channels digital I/O card with bus mastering DMA transfer supporting scatter gather technology
- PCI-7396 : High driving capability 96 channels DIO card

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

- PCI-7432/cPCI-7432 : 32 isolated channels DI & 32 isolated channels DO card
- PCI-7433/cPCI-7433 : 64 isolated channels DI card
- PCI-7434/cPCI-7434 : 64 isolated channels DO card
- cPCI-7432R : Isolation 32 Digital Inputs & 32 Digital Outputs with Rear I/O
- cPCI-7433R : Isolation 64 Digital Inputs Module with Rear I/O
- cPCI-7434R : Isolation 64 Digital Outputs Module with Rear I/O
- PCI-8554 : 16-CH Timer/Counter & DIO card
- PCI-9111 : advanced multi-function card
- PCI-9112/cPCI-9112 : advanced multi-function card with bus mastering DMA transfer capability
- PCI-9113 : 32 isolated channels A/D card
- PCI-9114 : 32-channel high gain multi-function card
- cPCI-9116: 64-channel advanced multi-function card with bus mastering DMA transfer capability
- PCI-9118 : 333KHz high speed multi-function card with bus mastering DMA transfer capability
- PCI-9812/10 : 20MHz Ultra-high speed A/D card with bus mastering DMA transfer capability
- cPCI-9812/10 : 20MHz Ultra-high speed A/D card with bus mastering DMA transfer capability

1.3 PCIS-DASK Language Support

PCIS-DASK is provided as a shared library for Linux. It can work with any 32-bit compiler, such as gcc, etc.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2

The Fundamentals of Building Applications with PCIS-DASK

2.1 Creating a PCIS-DASK Application Using C/C++

To create a data acquisition application using PCIS-DASK and C/C++, follow these steps:

step 1. Edit the source files.

Include the 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 you can use to develop your data acquisition application. Incorporate the following statement in your code to include the header file.

#include "dask.h"

step 3. Build your application.

Using the appropriated C/C++ compiler (gcc or cc) to compile the program. You should also use the -lpci_dask option to link libpci_dask.so library.

ex. gcc -o testai testai.c –lpci_dask.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3

PCIS-DASK Utilities

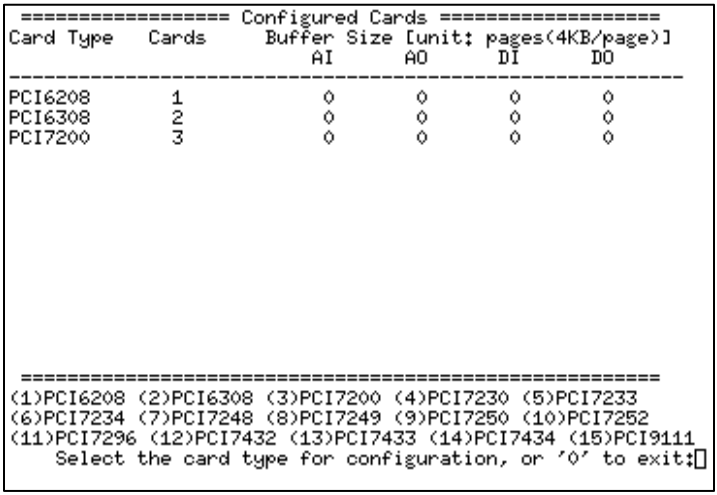
This chapter introduces the tools that accompanied with the PCIS-DASK package.

3.1 NuDAQ Configuration utility (dask_conf)

dask_conf is used for the users to ***configure*** PCIS-DASK drivers, ***remove*** configured drivers, and ***set/modify*** the allocated buffer sizes of AI, AO, DI and DO. The default location of this utility is pci-dask_xxx/util (where 'xxx' in *pci-dask_xxx*: is the version number) directory.

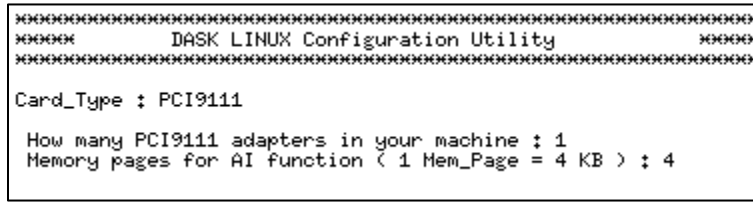
[dask_util in Linux]

The *dask_util* main screen is shown as following. If any PCIS-DASK driver has been configured, it will be shown on the *Configured Driver* list.



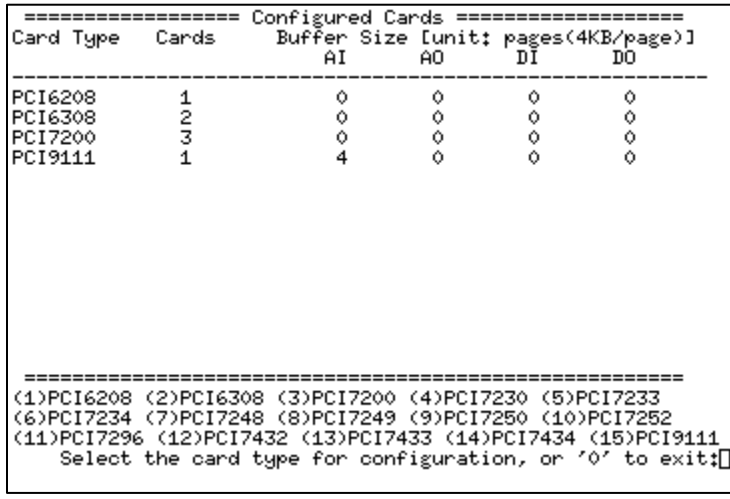
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

To configure one of the PCIS-DASK drivers, type the number corresponding to the Card Type and a *Driver Configuration* screen appears.



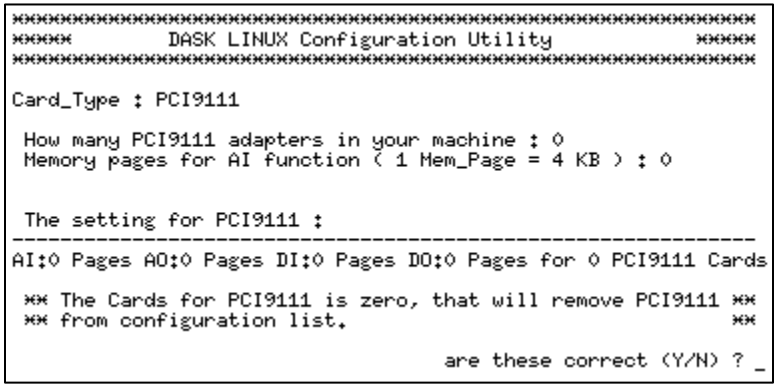
In this screen, users can input the number of cards and buffer size for continuous operation. To be platform-independent, the buffer size is set by the memory-page. The PAGE_SIZE in Intel platform is four kilo-byte. The “Memory Pages” of AI, AO, DI, DO represent the number of pages of contiguous *Initially Allocated memory* for continuous analog input, analog output, digital input and digital output respectively. Device driver will allocate these sizes of memory from the memory management module.

After the device configuration of the driver you select is finished, type “Y” to confirm the input data and return to the *dask_conf* main screen. The driver you just configured will be shown on the configured driver list as the following figure:



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

To ***modify*** the configuration, including the number of cards and the buffer size, you just select the driver and assign the settings again. Similarly, if the number of cards is set to zero, the configuration for the selected driver will be ***removed***.



When the configuration is finished, the configuration information of the devices will be saved into pci-dask_xxx/drivers/ pcidask.conf*.*
The content of "pcidask.conf" is similar to the following:

```
 ================== 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
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.2 PCIS-DASK Module Installation Script

Because of the PCI-bus architecture, the PCI devices can be detected automatically. All the user has to do is inserting the device modules and making the nodes for the devices.

A list of commands are needed, such as “insmod p9111”, “grep ‘p9111’ /proc/devices”, “mknod /dev/PCI9111W0 c 254 0” and “mknod /dev/PCI9111W1 c 254 1”.

You can do these manually, or use the installation script we provide. The installation script is located in pci-dask_xxx/drivers.

By the configuration file, *pcidask.conf*, the installation script inserts the device modules configured before and the memory management module if required. Then the script makes device nodes according to the number of cards. To make installation, execute the script as follows:

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

By default, the installation script will read the configuration file in the current directory. However, you could assign work directory for PCIS-DASK/X to installation script from the command argument. For example, if the pci-dask/x had been installed in /usr/local/pdask, you could install the driver with the following command:

dask_inst.pl /usr/local/pdask

The installation script will read the related configuration file by its argument and insert 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 modules are needed to be inserted by system automatically. You could add the following command in /etc/inittab, then the ***init*** process will insert the modules automatically.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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 cannot work fine in SMP kernel. The installation script will check the kernel version through the /proc/sys/kernel/version file. For SMP kernel, the version-checking procedure will display the additional error/warning messages and stop the installation.

3.3 PCIS-DASK Un-installation Script

The *dask_remove.pl* is written to remove *PCIS-DASK* installed in linux. The default location of this script is pci-dask_xxx/util directory.

To remove *PCIS-DASK* for linux, execute the un-installation script as follows:

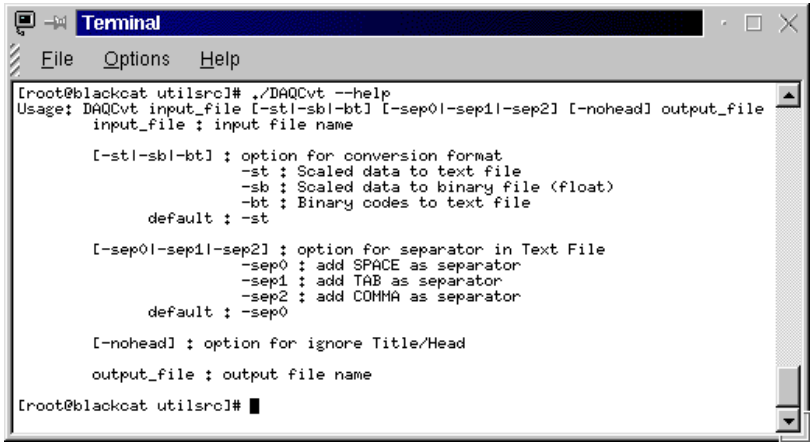
<InstallDir>/pci-dask_xxx/util/dask_remove.pl

Then the script will remove the device nodes made in */dev* and the library copied into */usr/lib*.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.4 PCIS-DASK Data File Converter utility (DAQCvt)

The data files, generated by the PCIS-DASK functions performing continuous data acquisition followed by storing the data to disk, is written in binary format. Since a binary file can't be read by the normal text editor and can't be used to analyze the accessed data by Excel, PCIS-DASK provides a convenient tool *DAQCvt* to convert the binary file to the file format read easily. The default location of this utility is <InstallDir>\util directory. Excuting *DAQCvt with the "--help"* argument, the *DAQCvt* shows the help information as the following figure:



[Option for data format conversion]

DAQCvt provides three options for data format conversion.

-st : 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 and then written to disk in text file format. This type is available for the data accessed from continuous AI operation only.

-sb : Scaled data to binary file (float)

The data in hexadecimal is scaled to engineering unit

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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

-bt : 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 will be handled to get the real data value. This type is available for the data accessed form continuous AI and DI operations.

The default option for data format conversion is *–st.*

[Option for separator in text file]

The data separator in converted text file is selectable among *space*, *Tab* and *comma* according to the option for separator.

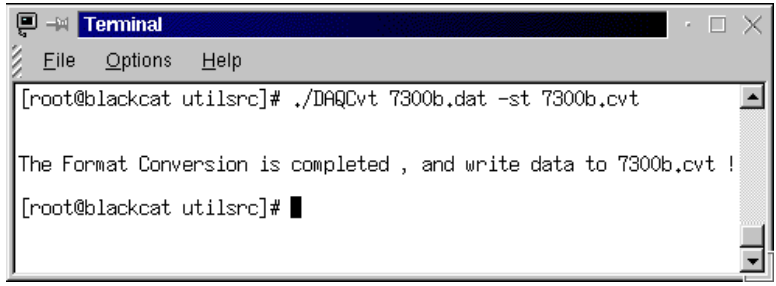
-sep0 : add *space* as separator
-sep1 : add *Tab* as separator
-sep2 : add *comma* as separator

The default option for data format conversion is *-sep0.*

[Option for Title/Head in text file]

If you want to ignore title/head which includes the card type information at the beginning of file, add the *–nohead* option.

After adding input filename, output filename and the options in command line, *DAQCvt* will perform the file conversion and save the data into the output file.The example of data conversion with *Scaled data to text file* option is shown as the following figure.



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4

PCIS-DASK Overview

This chapter describes the classes of functions in PCIS-DASK and briefly describes each function.

PCIS-DASK functions are grouped to the following classes:

- **General Configuration 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**
- **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 function
  - Dual-Interrupt System Setting function

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.1 General Configuration Function Group

Use these functions to initializes and configures data acquisition card.

Register_Card Initializes the hardware and software states of a NuDAQ PCI-bus data acquisition card. Register_Card must be called before any other DASK library functions can be called for that card.

Release_Card Tells DASK library that this registered card is not used currently and can be released. This would make room for new card to register.

4.2 Actual Sampling Rate Function Group

GetActualRate Returns the actual sampling rate the device will perform for the defined sampling rate value.

4.3 Analog Input Function Group

4.3.1 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 PCI9111. You must call this function before calling function to perform continuous analog input operation of PCI9111.

AI_9112_Config Informs PCIS-DASK library of the trigger source selected for the analog input operation of PCI9112. You must call this function before calling function

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

to perform continuous analog input operation of PCI9112.

AI_9113_Config Informs PCIS-DASK library of the trigger source selected for the analog input operation of PCI9113. You must call this function before calling function to perform continuous analog input operation of PCI9113.

AI_9114_Config Informs PCIS-DASK library of the trigger source selected for the analog input operation of PCI9114. You must call this function before calling function to perform continuous analog input operation of PCI9114.

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 PCI9116. You must call this function before calling function to perform continuous analog input operation of PCI9116.

AI_9118_Config Informs PCIS-DASK library of the trigger source, trigger mode, input mode, and conversion mode selected for the analog input operation of PCI9118. You must call this function before calling function to perform continuous analog input operation of PCI9118.

AI_9812_Config Informs PCIS-DASK library of the trigger source, trigger mode, and trigger properties selected for the analog input operation of PCI9812. You must call this function before

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

calling function to perform continuous analog input operation of PCI9812.

AI_InitialMemoryAllocated

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

4.3.2 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_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_VoltScale Converts the result from an AI_ReadChannel call to the actual input voltage.

4.3.3 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.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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

4.3.4 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.

4.4 Analog Output Function Group

4.4.1 Analog output Configuration Functions

AO_6208A_Config	Informs PCIS-DASK library of the current range selected for the analog output operation of PCI6208A. You must call this function before calling

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

function to perform current output operation.

AO_6308A_Config Informs PCIS-DASK library of the current range selected for the analog output operation of PCI6308A. 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 PCI6308V. 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 PCI9111. 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 PCI9112. You must call this function before calling function to perform voltage output operation.

4.4.2 One-Shot Analog Output Functions

AO_WriteChannel Writes a binary value to the specified analog output channel.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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.

4.5 Digital Input Function Group

4.5.1 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 PCI7200. You must call this function before calling function to perform continuous digital input operation of PCI7200.

DI_7300A_Config/ DI_7300B_Config
Informs PCIS-DASK library of the trigger source and trigger properties selected for the digital input operation of PCI7300A Rev.A or PCI7300A Rev.B. You must call this function before calling function to perform continuous digital input operation of PCI7300A Rev.A or PCI7300A Rev.B.

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

4.5.2 One-Shot Digital Input Functions

DI_ReadLine Reads the digital logic state of the specified digital line in the specified port.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

DI_ReadPort	Reads digital data from the specified digital input port.

4.5.3 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_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.

4.5.4 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.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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_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.

4.6 Digital Output Function Group

4.6.1 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 PCI7200. You must call this function before calling function to perform continuous digital output operation of PCI7200.

DO_7300A_Config/ DO_7300B_Config
Informs PCIS-DASK library of the trigger source and trigger properties selected for the digital input operation of PCI7300A Rev.A or PCI7300A Rev.B. You must call this function before calling function to perform

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

continuous digital output operation of PCI7300A Rev.A or PCI7300A Rev.B.

EDO_9111_Config Informs PCIS-DASK library of the mode of EDO channels of PCI9111.

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

4.6.2 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_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_Write ExtTrigLine
Sets the digital output trigger line to the specified state. This function is only available for PCI-7200.

4.6.3 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.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

DO_PGStart Performs pattern generation operation.

DO_PGStop Stops pattern generation operation.

4.6.4 Asynchronous Digital Output Monitoring Functions

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

DO_AsyncClear Stops the asynchronous digital output operation.

4.7 Timer/Counter Function Group

4.7.1 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_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.

4.7.2 The General-Purpose Timer/Counter Functions

GCTR_Setup Controls the general-purpose counter to operate in the specified mode.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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.

4.8 DIO Function Group

4.8.1 Digital Input/Output Configuration Functions

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.

4.8.2 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.

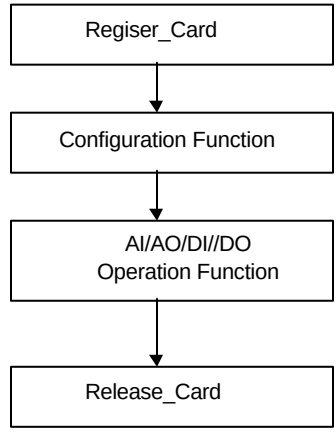
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5

PCIS-DASK 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.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.1 Analog Input Programming Hints

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

The nonbuffered single-point AI uses software polling method to read data from the device. The programming scheme for this kind of AI operation is described in section 5.1.1.

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 the moment the device module is inserted. We recommend the applications use *AI_InitialMemoryAllocated* function to get the size of initially allocated memory before performing continuous AI operation.

The buffered continuous analog input includes:

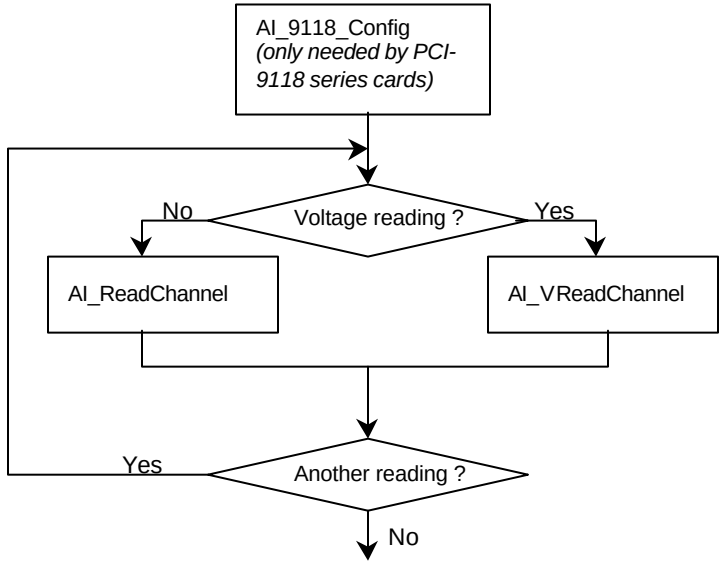
- synchronous continuous AI
- non-triggered non-double-buffered asynchronous continuous AI
- non-triggered double-buffered asynchronous continuous AI
- triggered non-double-buffered asynchronous continuous AI
- triggered double-buffered asynchronous continuous AI

They are described in section 5.1.2 to 5.1.6 section respectively. About the special consideration and performance issues for the buffered continuous analog input, please refer to the *Continuous Data Transfer in PCIS-DASK* chapter for the details.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.1.1 One-Shot Analog input programming Scheme

This section described the function flow typical of nonbuffered single-point analog input readings. While performing one-shot AI operation, most of the cards (except PCI-9118 series cards) don't need to include AI configuration step at the beginning of your application.



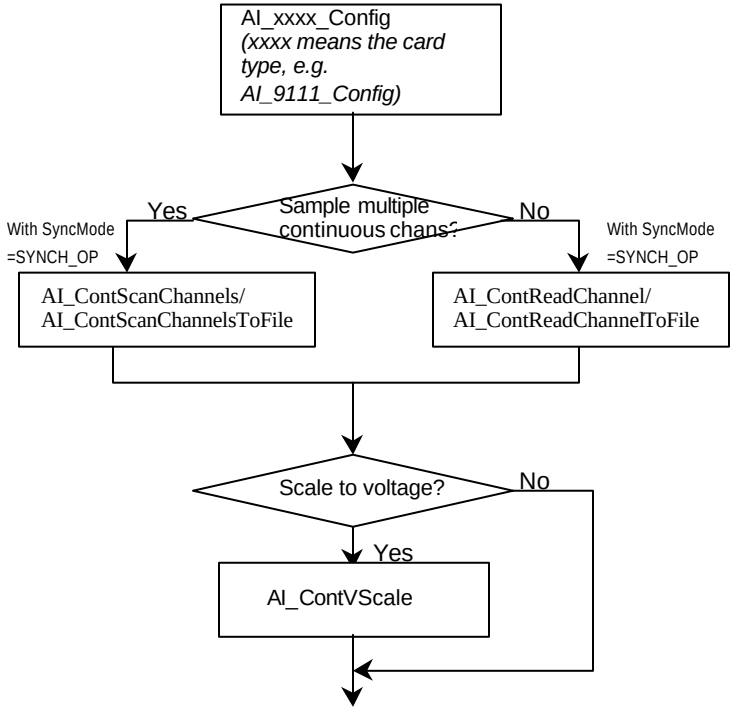
[Example Code Fragment]

```
card = Register_Card(PCI_9118, card_number);
        …
// only PCI-9118 is need
AI_9118_Config(card,Input_Signal|Input_Mode,0,0,0);
AI_ReadChannel(card, channelNo, range, &analog_input[i]);
        …
Release_Card(card);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.1.2 Synchronous Continuous Analog input programming Scheme

This section described 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 as *SYNCH_OP.*



[Example Code Fragment]

card = Register_Card(PCI_9111DG, card_number);
…

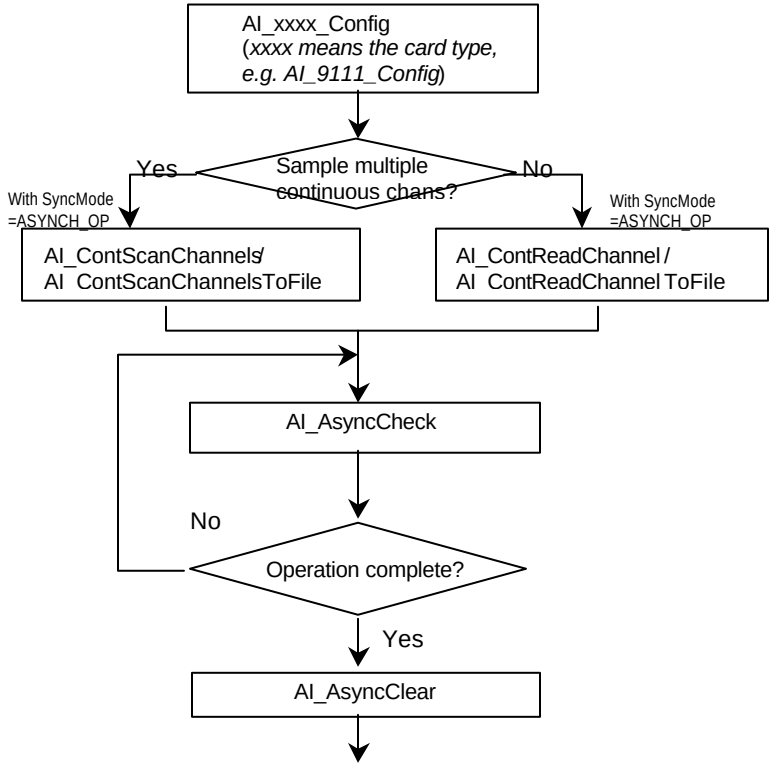
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
AI_9111_Config(card,TRIG_INT_PACER, 0, 1024);
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);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.1.3 Non-Trigger Non-double-buffered Asynchronous Continuous Analog input programming Scheme

This section described 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 as *ASYNCH_OP.*



[Example Code Fragment]

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
card = Register_Card(PCI_9111DG, card_number);
                …
AI_9111_Config(card,TRIG_INT_PACER, 0, 1024);
AI_AsyncDblBufferMode (card, 0); //non-double-buffered 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);

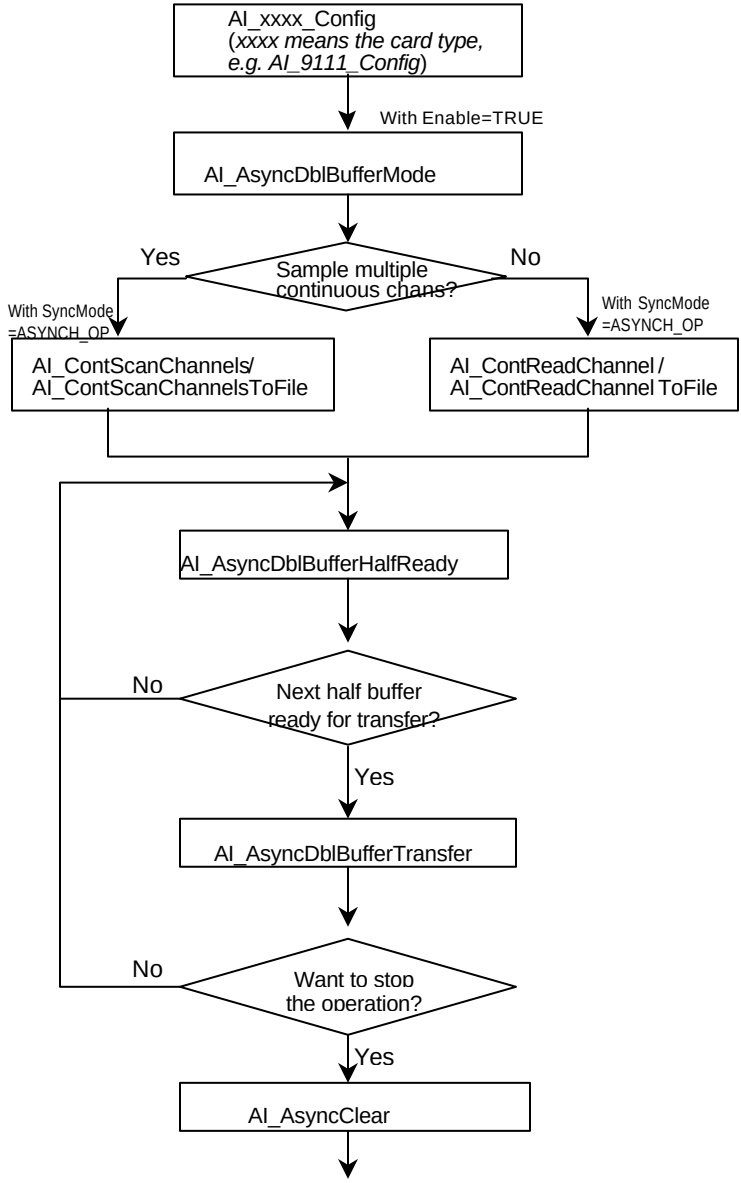
AI_AsyncClear(card, &count);
…
Release_Card(card);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.1.4 Non-Trigger Double-buffered Asynchronous Continuous Analog input programming Scheme

This section described 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 as *ASYNCH_OP.* In addition, double-buffered AI operation is enabled by setting *Enable* argument of *AI_AsyncDblBufferMode* function to 1. To learn more about double buffer mode, please refer to section 5.2 *Double-Buffered AI/DI Operation* for the details.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

[Example Code Fragment]

```
card = Register_Card(PCI_9111DG, card_number);
            …
AI_9111_Config(card,TRIG_INT_PACER, 0, 1024);
AI_AsyncDblBufferMode (card, 1); // Double-buffered 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 {
    do {
        AI_AsyncDblBufferHalfReady(card, &HalfReady, &fstop);
    } while (!HalfReady);

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

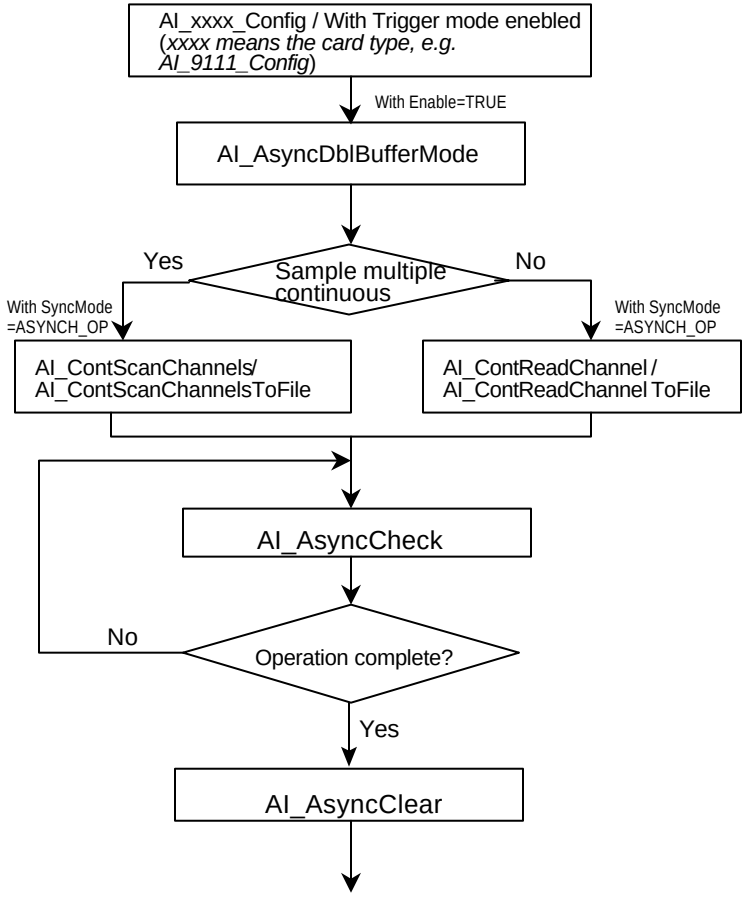
AI_AsyncClear(card, &count);
…
Release_Card(card);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.1.5 Trigger Mode Non-double-buffered Asynchronous Continuous Analog input programming Scheme

This section described 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. Using PCIS-DASK to perform trigger mode data acquisition, the *SyncMode* of continuous AI should be set as *ASYNCH_OP*.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



[Example Code Fragment]

card = Register_Card(PCI_9111DG, card_number);

…

AI_9111_Config(card,TRIG_INT_PACER, 0, postCount)
AI_AsyncDblBufferMode (card, 0); //non-double-buffered AI
AI_ContScanChannels (card, channel, range, ai_buf, data_size, (F64)sample_rate, ***ASYNCH_OP****); or*

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
AI_ContReadChannel(card, channel, range, ai_buf, data_size,
(F64)sample_rate, ASYNCH_OP)
do {
    AI_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);

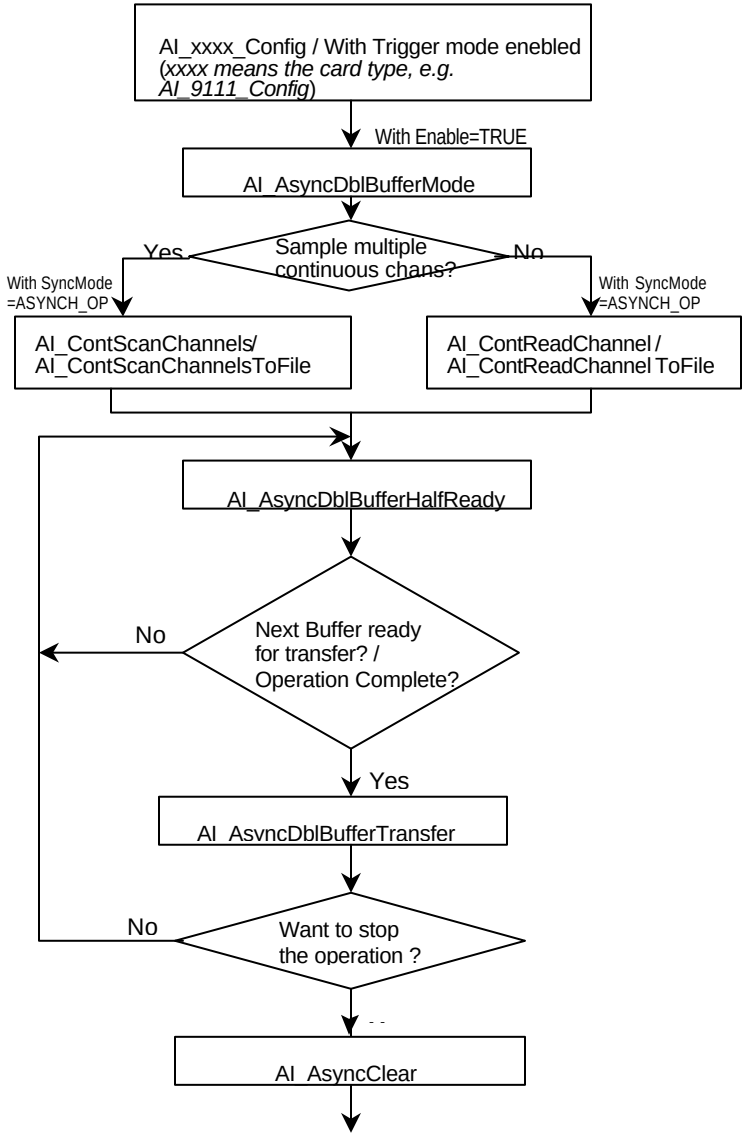
AI_AsyncClear(card, &count);
…
Release_Card(card);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.1.6 Trigger Mode Double-buffered Asynchronous Continuous Analog input programming Scheme

This section described 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. Using PCIS-DASK to perform trigger mode data acquisition, the *SyncMode* of continuous AI should be set as *ASYNCH_OP*. In addition, double-buffered AI operation is enabled by setting *Enable* argument of *AI_AsyncDblBufferMode* function to 1. To learn more about double buffer mode, please refer to section 5.2 *Double-Buffered AI/DI Operation* for the details.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



[Example Code Fragment]

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
card = Register_Card(PCI_9111,DG card_number);
                …
AI_9111_Config(card, TRIG_INT_PACER, 0, postCount)
AI_AsyncDblBufferMode (card, 1);  Double-buffered 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 {
     do {
          AI_AsyncDblBufferHalfReady(card, &HalfReady, &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);
```

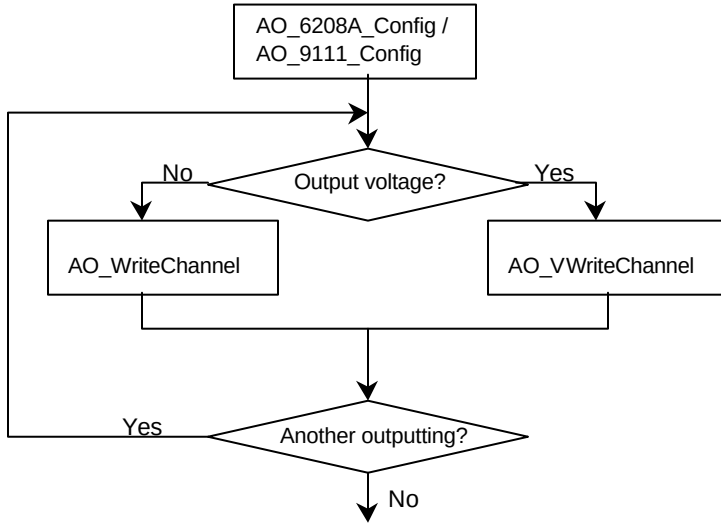

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.2 Analog Output Programming Hints

This section described the function flow typical of single-point analog output conversion. While performing the following operation, the AO configuration function has to be called at the beginning of your application:

a. Use PCI-6208A, PCI-6308A to perform current output

b. Use the analog output function that can convert a voltage value to a binary value and then write it to device, the AO configuration function has to be called at the beginning of your application.



[Example Code Fragment]

```
card = Register_Card(PCI_6208A, card_number);
        …
AO_6208A_Config(card, P6208_CURRENT_4_20MA);
AO_WriteChannel(card, chan, out_value);
        …
Release_Card(card);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3 Digital Input Programming Hints

PCIS-DASK provides two kinds of digital input operation — nonbuffered single-point digital input operation and buffered continuous digital input operation.

The nonbuffered 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 5.3.1.

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 you configure for the device. At the loading time, the driver allocates the memory from our private memory module witch manager the reserved memory space.

The buffered continuous analog input includes synchronous continuous DI, non-double-buffered asynchronous continuous DI and double-buffered asynchronous continuous DI. They are described in section 5.3.2 to 5.3.4 section respectively. About the special consideration and performance issues for the buffered continuous digital input, please refer to the *Continuous Data Transfer in PCIS-DASK* chapter for the details.

For some data acquisition card that supports scatter-gather technology(PCI-7300A_RevA, PCI-7300A_Rev.B), the DMA operation can use the memory space allocated in user's process. This mechanism improves the performance by eliminating the time of memory duplication.

By the design of DMA, DMA master controller transfers data only by the physical memory. If some pages are swapped out by MMU, the DMA controller will transfer the wrong data because the data indicated by the virtual address is not resident in system memory space. To ensure the virtual memory is resident in system memory space, the process must lock the memory space before starting DMA operation.

The relative functions of memory locking/unlocking are included in the PCIS-DASK library. The memory space allocated in your process will be locked before passing the virtual address to the device drivers, and be unlocked after the data acquisition is completed. However, Linux does not give the permission of mlock/munlock to everyone. Only the processes with ***root***

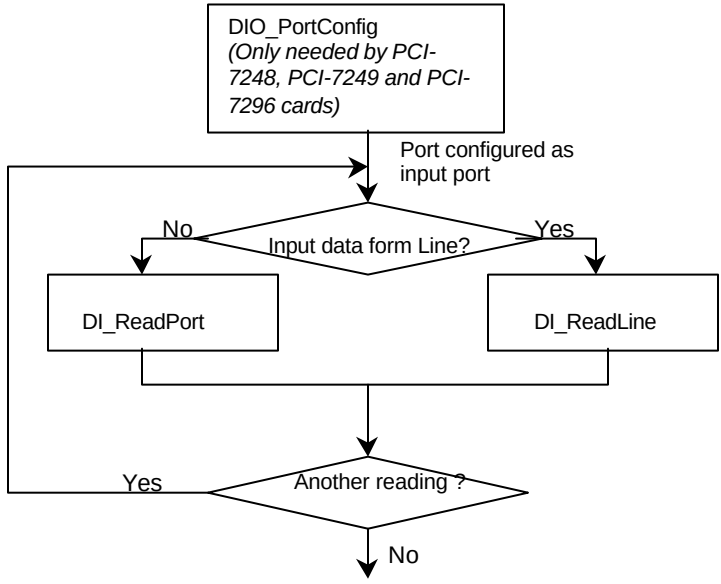
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

privilege can lock/unlock the memory. If you have no right permission, a –EPERM error will be returned

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3.1 One-Shot Digital input programming Scheme

This section described the function flow typical of non-buffered single-point digital input readings. While performing one-shot DI operation, the devices whose I/O port can be set as input or out put port (PCI-7248 and PCI7296) need to include port configuration function at the beginning of your application.



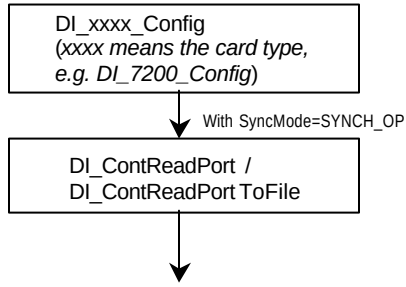
[Example Code Fragment]

```
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);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3.2 Synchronous Continuous Digital input programming Scheme

This section described 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 your application. In addition, for synchronous DI, the *SyncMode* argument in continuous DI functions has to be set as *SYNCH_OP.*



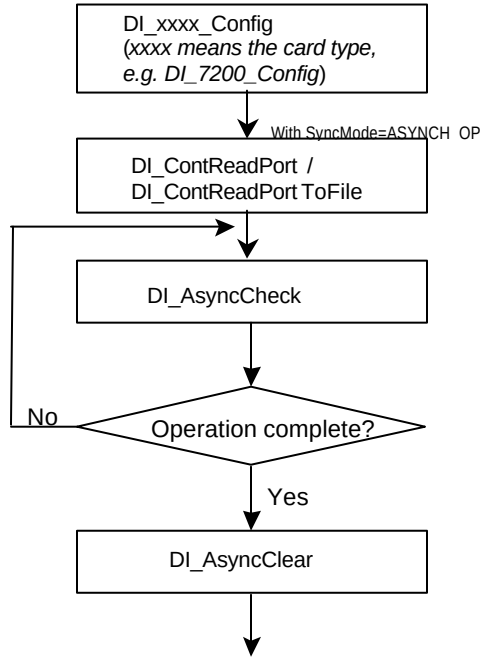
[Example Code Fragment]

```
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-buffered mode
DI_ContReadPort(card, 0, pMem, data_size, (F64)sample_rate,
SYNCH_OP)
        …
Release_Card(card);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3.3 Non-double-buffered Asynchronous Continuous Digital input programming Scheme

This section described 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 your application. In addition, for asynchronous DI operation, the *SyncMode* argument in continuous DI functions has to be set as *ASYNCH_OP.*



[Example Code Fragment]

card = Register_Card(PCI_7200, card_number);
…

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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);

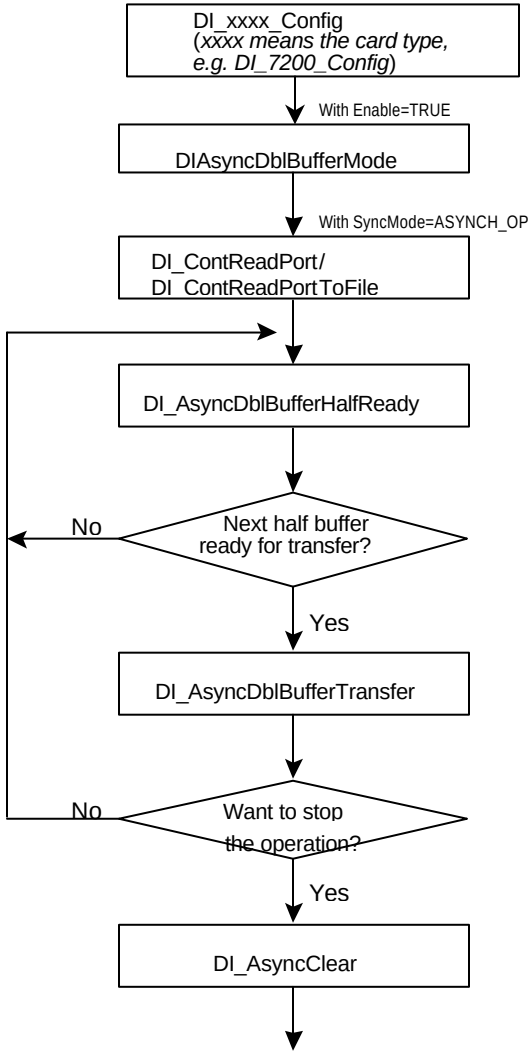
DI_AsyncClear(card, &count);
…
Release_Card(card);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3.4 Double-buffered Asynchronous Continuous Digital input programming Scheme

This section described 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 your application. For asynchronous DI, the *SyncMode* argument in continuous DI functions has to be set as *ASYNCH_OP.* In addition, double-buffered AI operation is enabled by setting *Enable* argument of *DI_AsyncDblBufferMode* function to 1. To learn more about double buffer mode, please refer to the *Double-Buffered AI/DI operation* section for the details.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



[Example Code Fragment]

card = Register_Card(PCI_7200, card_number);

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
            …
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);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.4 Digital Output Programming Hints

PCIS-DASK provides three kinds of digital output operation — nonbuffered single-point digital output operation, buffered continuous digital output operation and pattern generation.

The nonbuffered 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 5.4.1.

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 you configure for the device. At the loading time, the driver allocates the memory from our private memory module witch manager the reserved memory space.

The buffered continuous digital output includes synchronous continuous DO and asynchronous continuous DO. They are described in section 5.4.2 and 5.4.3 section individually. About the special consideration and performance issues for the buffered continuous digital output, please refer to the *Continuous Data Transfer in PCIS-DASK* chapter for the details.

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 5.4.4.

For some data acquisition card that supports scatter-gather technology(PCI-7300A_RevA, PCI-7300A_Rev.B), the DMA operation can use the memory space allocated in user's process. This mechanism improves the performance by eliminating the time of memory duplication.

By the design of DMA, DMA master controller transfers data only by the physical memory. If some pages are swapped out by MMU, the DMA controller will transfer the wrong data because the data indicated by the virtual address is not resident in system memory space. To ensure the virtual memory is resident in system memory space, the process must lock the memory space before starting DMA operation.

The relative functions of memory locking/unlocking are included in the PCIS-DASK library. The memory space allocated in your process will be locked before passing the virtual address to the device drivers, and be unlocked after the data acquisition is

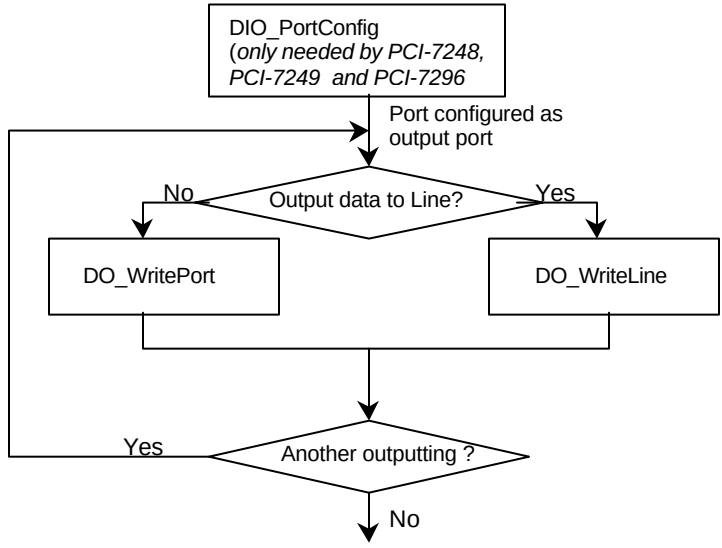
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

completed. However, Linux does not give the permission of mlock/munlock to everyone. Only the processes with ***root*** privilege can lock/unlock the memory. If you have no right permission, a –EPERM error will be returned

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.4.1 One-Shot Digital output programming Scheme

This section described 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 out put port (PCI-7248, PCI7249 and PCI-7296) need to include port configuration function at the beginning of your application.



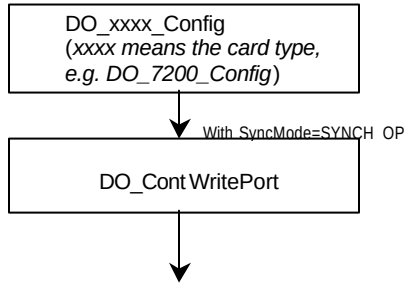
[Example Code Fragment]

```
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);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.4.2 Synchronous Continuous Digital output programming Scheme

This section described 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 your application. In addition, for synchronous DO operation, the *SyncMode* argument in continuous DO functions for synchronous mode has to be set as *SYNCH_OP.*



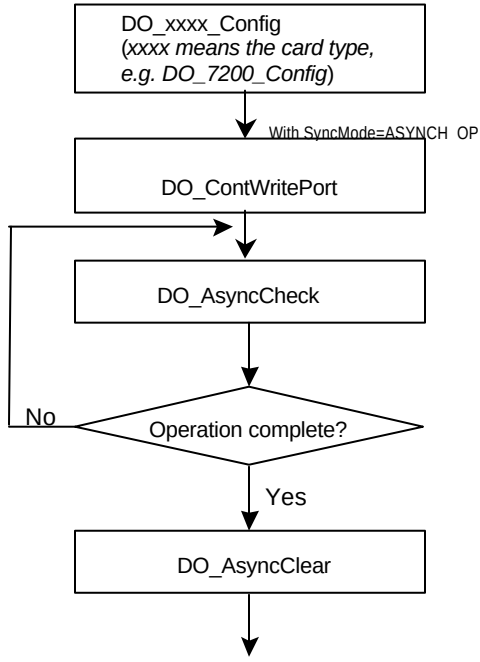
[Example Code Fragment]

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


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.4.3 Asynchronous Continuous Digital output programming Scheme

This section described 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 your application. In addition, for asynchronous DO operation, the *SyncMode* argument in continuous DO functions for asynchronous mode has to be set as *ASYNCH_OP.*



[Example Code Fragment]

card = Register_Card(PCI_7200, card_number);
…
DO_7200_Config(card, TRIG_INT_PACER, OREQ_DISABLE, OTRIG_LOW);

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

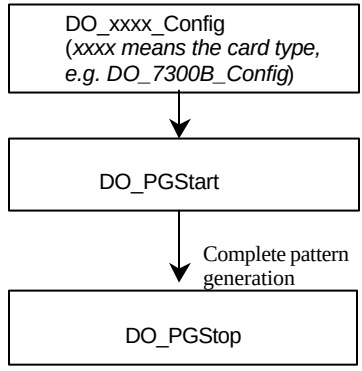
```
DO_ContWritePort(card, 0, DoBuf, count, 1, (F64)sample_rate, ASYNCH_OP);
do {
    DO_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);

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


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.4.4 Pattern Generation Digital output programming Scheme

This section described 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 your 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);
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.5 Interrupt Asynchronous Notification Programming Hints

PCIS-DASK/X provides the asynchronous signaling method to perform interrupt occurrence notification for NuDAQ DIO cards that have dual interrupt system.
The interrupt notification is through user-defined *signal handler*. When a user-specified interrupt event occurs, the user-defined *signal handler* is called to carry out the appropriate task.
The time delay between the interrupt event and notification is highly variable and depends largely on how loaded your system is. In addition, if a callback function is called, succeeding events will not be handled until your callback has returned. If the time interval between interrupt events is smaller than the time taken for callback function processing, the succeeding signals are put in a definite size of signal queue. It might induce unexpected result that the amount of the queued signals exceeds the signal number limit (1024). The user application design has to avoid such situation.

[Example Code Fragment]

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

//INT  notification is through sig1_handler
DIO_SetDualInterrupt(card, INT1_EXT_SIGNAL,
INT2_EXT_SIGNAL, sig1_handler, sig2_handler);
            …
//disable interrupt
DIO_SetDualInterrupt(card, INT1_EXT_SIGNAL,
INT2_EXT_SIGNAL, INT1_DISABLE, INT1_DISABLE);


//signal handler for INT1
void sig1_handler( int signo )
{
   …
}
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
//signal handler for INT2
void sig2_handler( int signo )
{
    …
}
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6

Continuous Data Transfer in PCIS-DASK

The continuous data transfer functions in PCIS-DASK input or output blocks of data to or from a plug-in NuDAQ PCI device. For input operations, PCIS-DASK must transfer the incoming data to a buffer in the computer memory. For output operations, PCIS-DASK must transfer outgoing data from a buffer in the computer memory to the NuDAQ PCI device. This chapter describes the mechanism and techniques that PCIS-DASK uses for continuous data transfer and the considerations for selecting the continuous data transfer mode (sync. or async., double buffered or not, triggered or non-triggered mode).

6.1 Continuous Data Transfer Mechanism

PCIS-DASK uses two mechanisms to perform the continuous data transfer. The first one, interrupt transfer, transfers data through the interrupt mechanism. The second one is to use the DMA controller chip to perform a hardware transfer of the data. Whether PCIS-DASK uses interrupt or DMA depends on the device. If the device support both of these two mechanisms, PCIS-DASK decides on the data transfer method that typically takes maximum advantage of available resources. For example, PCI-9112 supports interrupt and DMA for data transfers. The DMA data transfer is typically faster, so PCIS-DASK takes advantage of it. PCI-9111 supports FIFO Half-Full and EOC interrupt transfer modes. PCIS-DASK takes FIFO Half-Full interrupt transfer mode, because the CPU is interrupted do data

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

transfer only when the FIFO becomes half-full.

6.2 Double-Buffered AI/DI Operation

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

6.2.1 Double Buffer Mode Principle

The data buffer for double-buffered continuous input operation is a circular buffer logically. It is logically divided into two equal halves. The double-buffered input begins when device starts writing data into the first half of the circular buffer (Figure 6-1a). After device begins writing to the second half of the circular buffer, you can copy the data from the first half into the transfer buffer (user buffer) (Figure 6-1b). You now can process the data in the transfer buffer according to 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. You now can copy the second half of the circular buffer to the transfer buffer (Figure 6-1c). The data in the transfer buffer is again available for process. The process can be repeated endlessly to provide a continuous stream of data to your application (Figure 6-1d).

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

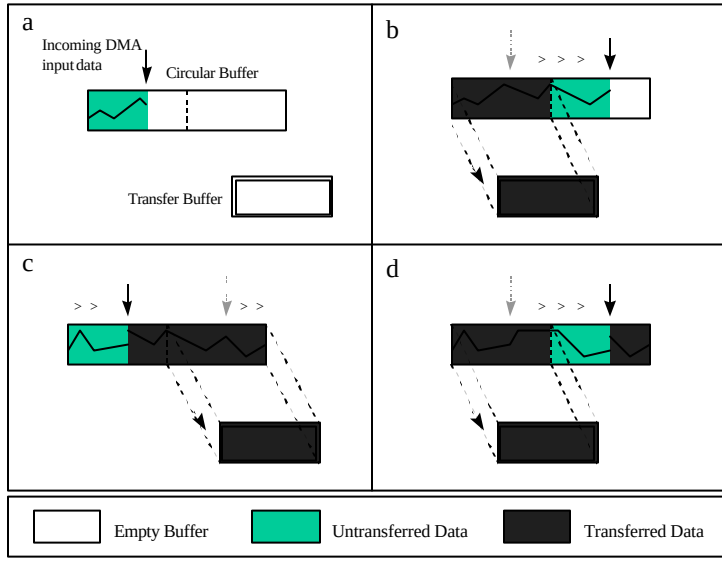


Figure 7-1

The PCIS-DASK double buffer mode functions were designed according to the principle described above. If you use **AI_AsyncDblBufferMode/DI_AsyncDblBufferMode** to enable double buffer mode, the following continuous AI/DI function will perform double-buffered continuous AI/DI. You can call **AI_AsyncDblBufferHalfReady/DI_AsyncDblBufferHalfReady** to check if data in the circular buffer is half full and ready for copying to the transfer buffer. Then you can call **AI_AsyncDblBufferTransfer/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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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 exits 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.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3 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.

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 will be reused until the data acquisition operation is completed. Therefore, to protect the data you want to get from being overwritten, the size of the single buffer should be the same as or larger than the amount of data you wish to access. For example, if you want to perform single-buffered middle-trigger AI with PCI-9111, 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 is at least 4000 in order to get all the data you want to collect. Since the data are handled after the input operation is completed, the desired data loss problem hardly occurs.

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

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000