D2K-DASK

Data Acquisition Software Development Kit

For DAQ-2000 Devices, Windows NT/98/2000/XP

User's Guide

Manual Rev. 1.61 : Mar. 17, 2004

Trademarks

CONTENTS

INTRODUCTION TO D2K-DASK 1

1.1 ABOUT THE D2K-DASK SOFTWARE 1
1.2 D2K-DASK HARDWARE SUPPORT 2
1.3 D2K-DASK LANGUAGE SUPPORT 2

THE FUNDAMENTALS OF BUILDING WINDOWSNT/98/2000 APPLICATIONS WITH D2K-DASK 3

2.1 CREATING A WINDOWS NT/98/2000 D2K-DASK APPLICATIONS USING MICROSOFT VISUAL C/C++ 3
2.2 CREATING A WINDOWS NT/98/2000 D2K-DASK APPLICATIONS USING MICROSOFT VISUAL BASIC 3

D2K-DASK UTILITIES 6

3.1 DAQ-2000 REGISTRY/CONFIGURATION UTILITY (D2KUTIL) 6
3.2 D2K-DASK DATA FILE CONVERTER UTILITY (DAQCVT) 10
3.3 D2K-DASK SAMPLE PROGRAMS BROWSER (EXAMPLES.EXE) 11

D2K-DASK OVERVIEW 12

4.1 GENERAL CONFIGURATION FUNCTION GROUP 13
4.2 ANALOG INPUT FUNCTION GROUP 13
4.2.1 Analog Input Configuration Functions 13
4.2.2 One-Shot Analog Input Functions 13
4.2.3 Continuous Analog Input Functions 13
4.2.4 Asynchronous Analog Input Monitoring Functions 14
4.3 ANALOG OUTPUT FUNCTION GROUP 15
4.3.1 Analog output Configuration Functions 15
4.3.2 One-Shot Analog Output Functions 15
4.3.3 Continuous Analog Output Functions 15
4.3.4 Asynchronous Analog Output Monitoring Functions 16
4.4 DIGITAL INPUT FUNCTION GROUP 16
4.4.1 One-Shot Digital Input Functions 16
4.5 DIGITAL OUTPUT FUNCTION GROUP 16
4.5.1 One-Shot Digital Output Functions 16
4.6 GENERAL TIMER/COUNTER FUNCTION GROUP 17
4.6.1 The General-Purpose Timer/Counter Functions 17
4.7 DIO FUNCTION GROUP 17
4.7.1 Digital Input/Output Configuration Functions 17
4.8 SSI FUNCTION GROUP 17
4.9 CALIBRATION FUNCTION GROUP 17

D2K-DASK APPLICATION HINTS 19

5.1 ANALOG INPUT PROGRAMMING HINTS 20
5.1.1 One-Shot Analog input programming Scheme *21*
5.1.2 Continuous Analog input (with initial default settings) programming Scheme *23*
5.1.3 Post Trigger Mode/ Delay Trigger Mode Synchronous Continuous Analog input programming Scheme .. *26*
5.1.4 Post Trigger Mode/ Delay Trigger Mode Non-double-buffered Asynchronous Continuous Analog input programming Scheme *27*
5.1.5 Post Trigger Mode/ Delay Trigger Mode Double-buffered Asynchronous Continuous Analog input programming Scheme *29*
5.1.6 Pre-Trigger Mode/ Middle-Trigger Mode Non-double-buffered Asynchronous Continuous Analog input programming Scheme *32*
5.1.7 Pre-Trigger Mode/ Middle-Trigger Mode Double-buffered Asynchronous Continuous Analog input programming Scheme *34*
5.2 ANALOG OUTPUT PROGRAMMING HINTS 37
5.2.1 One-Shot Analog output programming Scheme *38*
5.2.2 Continuous Analog output (with initial default settings) programming Scheme *40*
5.2.3 Non-double-buffered Asynchronous Continuous Analog output programming Scheme *43*
5.2.4 Double-buffered Asynchronous Continuous Analog output programming Scheme *45*
5.3 DIGITAL INPUT PROGRAMMING HINTS 49
5.3.1 One-Shot Digital input programming Scheme *49*
5.4 DIGITAL OUTPUT PROGRAMMING HINTS 50
5.4.1 One-Shot Digital output programming Scheme *50*
5.5 DAQ EVENT MESSAGE PROGRAMMING HINTS 51

CONTINUOUS DATA TRANSFER IN D2K-DASK 52

6.1 CONTINUOUS DATA TRANSFER MECHANISM 52
6.2 DOUBLE-BUFFERED AI/AO OPERATION 52
6.2.1 Double Buffer Mode Principle *52*
6.2.2 Single-Buffered Versus Double-Buffered Data Transfer *53*
6.3 PRE-TRIGGER MODE/ MIDDLE-TRIGGER MODE DATA ACQUISITION FOR ANALOG INPUT 54

DISTRIBUTION OF APPLICATIONS 55

7.1 FILES 55
7.2 AUTOMATIC INSTALLERS 55
7.3 MANUAL INSTALLATION 56

How to Use This Manual

This manual is to help you use the D2K-DASK software driver for DAQ-2000 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 D2K-DASK" describes the hardware and language support of D2K-DASK.
- Chapter 2, "The Fundamentals of Building Windows NT/98 Applications with D2K-DASK" describes the fundamentals of creating D2K-DASK applications in Windows NT and Windows 98.
- Chapter 3, "D2K-DASK Utilities" describes the utilities D2K-DASK provides.
- Chapter 4, "D2K-DASK Overview" describes the classes of functions in D2K-DASK and briefly describes each function.
- Chapter 5, "D2K-DASK Application Hints" provides the programming schemes showing the function flow of that D2K-DASK performs analog I/O and digital I/O.
- Chapter 6, "Continuous Data Transfer in D2K-DASK" describes the mechanism and techniques that D2K-DASK uses for continuous data transfer.
- Chapter 7, " Distribution of Applications" describes the required files to distribute your applications.

1

Introduction to D2K-DASK

1.1 About the D2K-DASK Software

D2K-DASK is a software development kit for DAQ-2000 data acquisition cards. It contains a high performance data acquisition driver for developing custom applications under Windows NT, Windows 98 and Windows 2000 environments.

The memory and data buffer management capabilities free developers from dealing with there complex issues. That is, D2K-DASK is constructed to provide a simple programming interface in communication with the DAQ-2000 data acquisition cards. The easy-to-use functions provided by D2K-DASK allow a programmer to use the features of the card in a high level way.

Using D2K-DASK also makes you take advantage of the power and features of Microsoft Win32 System for your data acquisition applications, including running multiple applications and using extended memory. Also, using D2K-DASK under Visual Basic environment makes it easy to create custom user interfaces and graphics.

In addition to the software drivers, some sample programs are provided for your reference to save a lot of programming time and get some other benefits as well.

1.2 D2K-DASK Hardware Support

ADLink will periodically upgrade D2K-DASK for new DAQ-2000 data acquisition cards. Please refer to Release Notes for the cards that the current D2K-DASK actually supports. The following cards are those D2K-DASK supports currently or will support in the near future:

- DAQ-2010 : 2MHz 4 channels simultaneous A/D and 2 channels D/A output device with bus mastering DMA transfer capability
- DAQ-2005 : 500kHz 4 channels simultaneous A/D and 2 channels D/A output device with bus mastering DMA transfer capability
- DAQ-2006 : 250kHz 4 channels simultaneous A/D and 2 channels D/A output device with bus mastering DMA transfer capability
- DAQ-2204 : 3MHz 64 channels multiplexed A/D and 2 channels D/A output device with bus mastering DMA transfer capability
- DAQ-2205 : 500kHz 64 channels multiplexed A/D and 2 channels D/A output device with bus mastering DMA transfer capability
- DAQ-2206 : 250kHz 64 channels multiplexed A/D and 2 channels D/A output device with bus mastering DMA transfer capability
- DAQ-2208 : 3MHz 96 channels multiplexed A/D device with bus mastering DMA transfer capability
- DAQ-2213 : 250kHz 16 channels multiplexed A/D device with bus mastering DMA transfer capability
- DAQ-2214 : 250kHz 16 channels multiplexed A/D and 2 channels D/A output device with bus mastering DMA transfer capability
- DAQ-2501 : High Performance 4 channels analog output Multi-function device with bus mastering DMA transfer capability
- DAQ-2502 : High Performance 8 channels analog output Multi-function device with bus mastering DMA transfer capability

1.3 D2K-DASK Language Support

D2K-DASK is DLL (Dynamic-Link Library) version for using under Windows NT, Window 98 and Windows 2000. It can work with any Windows programming language that allows calls to a DLL, such as Microsoft Visual C/C++ (4.0 or above), Borland C++ (5.0 or above), or Microsoft Visual Basic (4.0 or above), etc.

D2K-DASK also provides a D2K-DASK function prototype file, D2KDask.pas for use with Borland Delphi 2.*x* (32-bit) or above.

2

The Fundamentals of Building WindowsNT/98/2000 Applications with D2K-DASK

2.1 Creating a Windows NT/98/2000 D2K-DASK Applications Using Microsoft Visual C/C++

To create a data acquisition application using D2K-DASK and Microsoft Visual C/C++, follow these steps after entering Visual C/C++:

step 1. Open the project in which you want to use D2K-DASK. This can be a new or existing project

step 2. Include header file D2KDASK.H in the C/C++ source files that call D2K-DASK functions. D2KDASK.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 "D2KDASK.H"

step 3. Build your application.

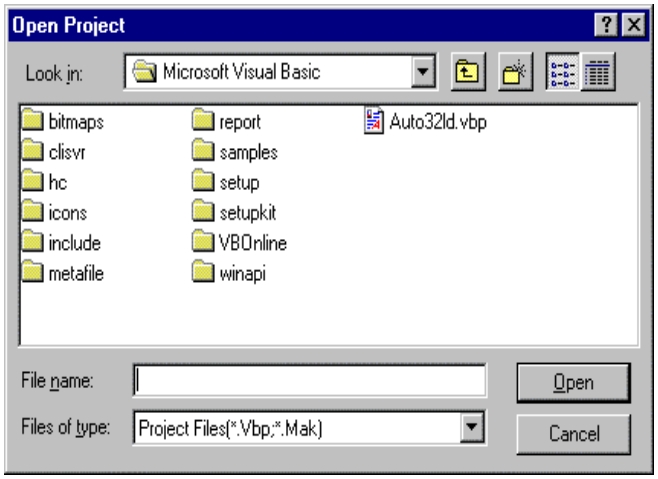
Setting the appropriate compile and link options, then build your application by selecting the Build command from Build menu (Visual C/C++ 5.0). Remember to link D2K-DASK's import library, D2K-DASK.LIB.

2.2 Creating a Windows NT/98/2000 D2K-DASK Applications Using Microsoft Visual Basic

To create a data acquisition application using D2K-DASK and Visual Basic, follow these steps after entering Visual Basic:

step 1. Open the project in which you want to use D2K-DASK. This can be a new or existing project

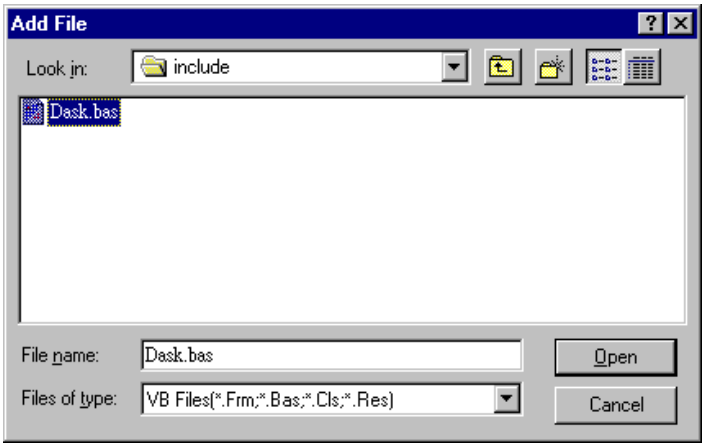
Open a new project by selecting the New Project command from the File menu. If it is an existing project, open it by selecting the Open Project command from the File menu. Then the Open Project dialog box appears.



Changed directory to the place the project file located. Double-click the project file name in the File Name list to load the project.

step 2. Add file D2KDASK.BAS into the project if this file is not included in the project. This file contains all the procedure declarations and constants that you can use to develop your data acquisition application.

From the File menu, select the Add File command. The Add File window appears, displaying a list of files in the current directory.



Select D2KDASK.BAS from the Files list by double clicking on it. If you can't find this file in the list, make sure the list is displaying files from the correct directory. By default, D2KDASK.BAS is installed in C:\ADLink\D2K-DASK\INCLUDE.

step 3. Design the interface for the application.

To design the interface, you place the desired elements, such as command button, list box, text box, etc., on the Visual Basic form. These are standard controls from the Visual Basic Toolbox. To place a control on a form, you just move pointer to Toolbox, select the desired control and draw it on the form. Or you can double-click the control icon in the Toolbox to place it on the form.

step 4. Set properties for the controls.

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

step 5. Write the event code.

The event code defines the action you want to perform when an event occurs. To write the event code, double-click the desired control or form to view the code module and then add code you want. You can call the functions that declared in the file D2KDASK.BAS to perform data acquisition operations.

step 6. Run your application.

To run the application, choose Start from the Run menu, or click the Start icon  on the toolbar (you can also press F5).

step 7. Distribute your application.

Once you have finished a project, you can save the application as an executable (.EXE) file by using the Make EXE File command on the File menu. And once you have saved your application as an executable file, you've ready to distribute it. When you distribute your application, remember also to include the D2K-DASK's DLL and driver files. Please refer to chapter "*Distribution of Applications*" for the details.

3

D2K-DASK Utilities

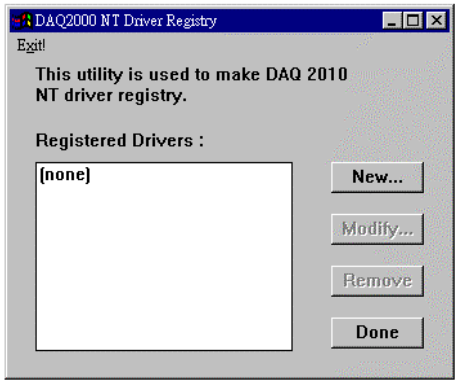
This chapter introduces the tools that accompanied with the D2K-DASK package.

3.1 DAQ-2000 Registry/Configuration utility (D2kUtil)

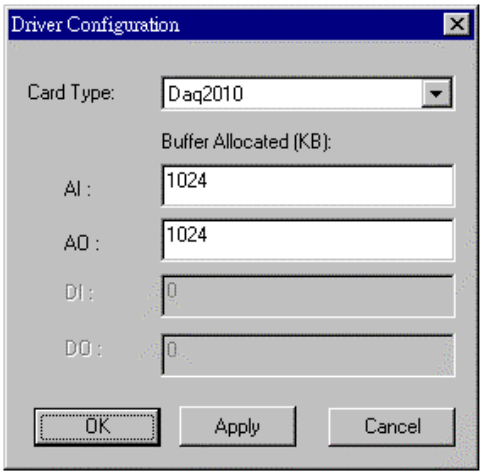
D2kUtil is used for the users to ***register*** D2K-DASK drivers (Windows NT4 only), ***remove*** installed drivers (Windows NT4 only), and ***set/modify*** the allocated buffer sizes of AI, AO, DI and DO. The default location of this utility is <InstallDir>\Util directory.

[D2kUtil in Windows NT]

The *D2kUtil* main window is shown as the following window. If any D2K-DASK/NT driver has been registered, it will be shown on the *Registered Driver* list.



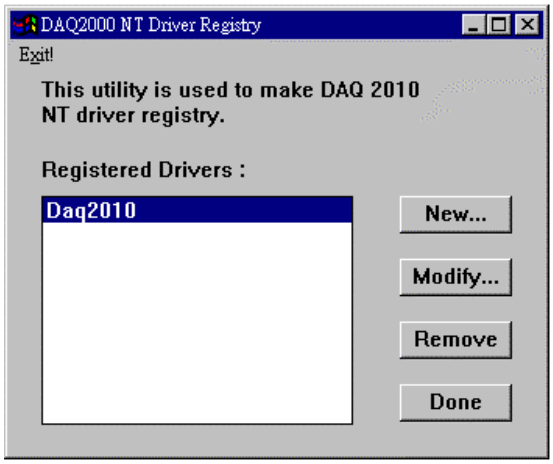
To register one of D2K-DASK drivers, click “New…” button and a *Driver Configuration* window appears.



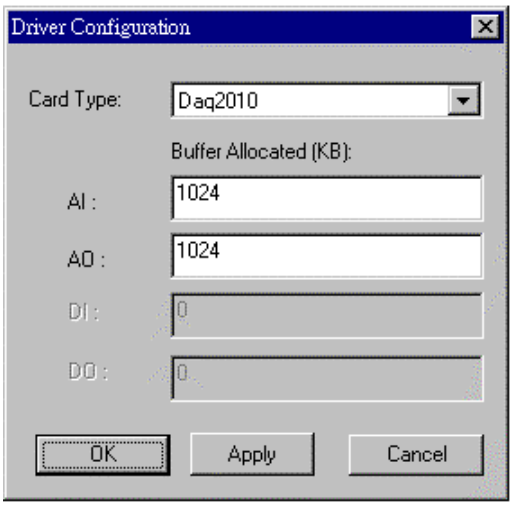
In this window, users can select the driver you want to register and input the parameters in the box corresponding to AI, AO, DI, or DO for the requirement of your applications. The “Buffer Allocated” of AI, AO, DI, DO represent the sizes of contiguous Initially Allocated memory for continuous analog input, analog output, digital input, digital output respectively.

Its unit is KB, i.e. 1024 bytes. Device driver will try to allocate these sizes of memory at system startup time. The size of initially allocated memory is the maximum memory size that DMA or Interrupt transfer can be performed. It will induce an unexpected result in that DMA or Interrupt transfer performed exceeds the initially allocated size.

After the device configurations of the driver you select is finished, click “OK” to register the driver and return to the *D2kUtil* main window. The driver you just registered will be shown on the registered driver list as the following figure:



Using *D2kUtil* to ***change the buffer allocated settings*** of one of the D2K-DASK drivers, select the driver from the *Registered Driver* list and click “Modify…” button and then a “Driver Configuration” window is shown as below.



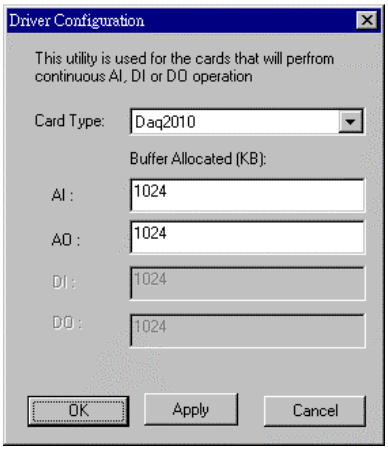
Inside the allocated buffer size fields of AI, AO, DI and DO are the originally set values. Type the value in the box corresponding to AI, AO, DI, or DO according to the requirement of your applications, and then click “OK” button.

To ***remove*** a registered driver, select the driver from the *Registered Driver* list in The *D2kUtil* main window and click “Remove” button. The selected driver will be deleted from the registry table.

[D2kUtil in Windows 98]

This utility is used to ***set/modify*** the allocated buffer sizes of AI, AO, DI and DO. The allocated buffer sizes of AI, AO, DI, DO represent the sizes of contiguous Initially Allocated memory for continuous analog input, analog output, digital input, digital output respectively. Its unit is page *KB*, i.e. 1024 bytes. Device driver will try to allocate these sizes of memory at system startup time. The size of initially allocated memory is the maximum memory size that DMA or Interrupt transfer can be performed. It will induce an unexpected result in that DMA or Interrupt transfer performed exceeds the initially allocated size.

The “Driver Configuration” window is shown as below.



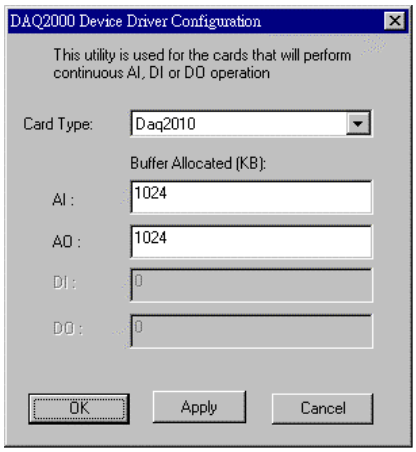
Using *D2kUtil* to ***change the buffer allocated settings*** of one of the D2K-DASK drivers, select the driver from the *Card Type combo box*.

Inside the allocated buffer size fields of AI, AO, DI and DO are the originally set values. Type the value in the box corresponding to AI, AO, DI, or DO according to the requirement of your applications, and then click “Apply” button.

[D2kUtil in Windows 2000]

This utility is used to *set/modify* the allocated buffer sizes of AI, AO, DI and DO. The allocated buffer sizes of AI, AO, DI, DO represent the sizes of contiguous Initially Allocated memory for continuous analog input, analog output, digital input, digital output respectively. Its unit is page *KB*, i.e. 1024 bytes. Device driver will try to allocate these sizes of memory at system startup time. The size of initially allocated memory is the maximum memory size that DMA or Interrupt transfer can be performed. It will induce an unexpected result in that DMA or Interrupt transfer performed exceeds the initially allocated size.

The “Driver Configuration” window is shown as below.

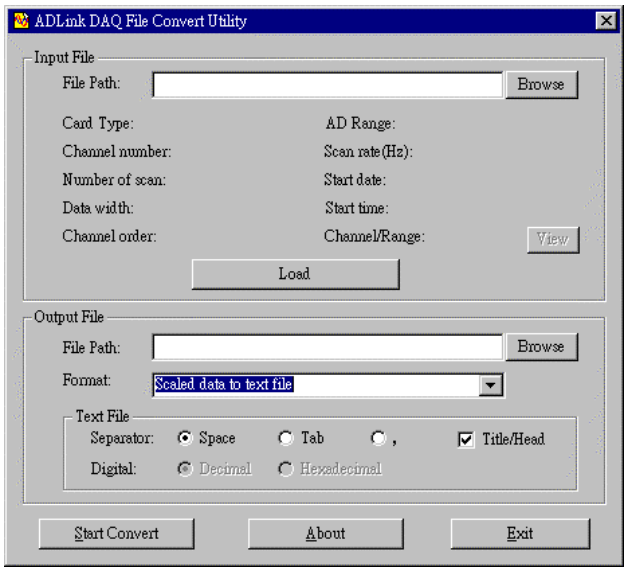


Using *D2kUtil* to *change the buffer allocated settings* of one of the D2K-DASK drivers, select the driver from the *Card Type combo box*.

Inside the allocated buffer size fields of AI, AO, DI and DO are the originally set values. Type the value in the box corresponding to AI, AO, DI, or DO according to the requirement of your applications, and then click “Apply” button.

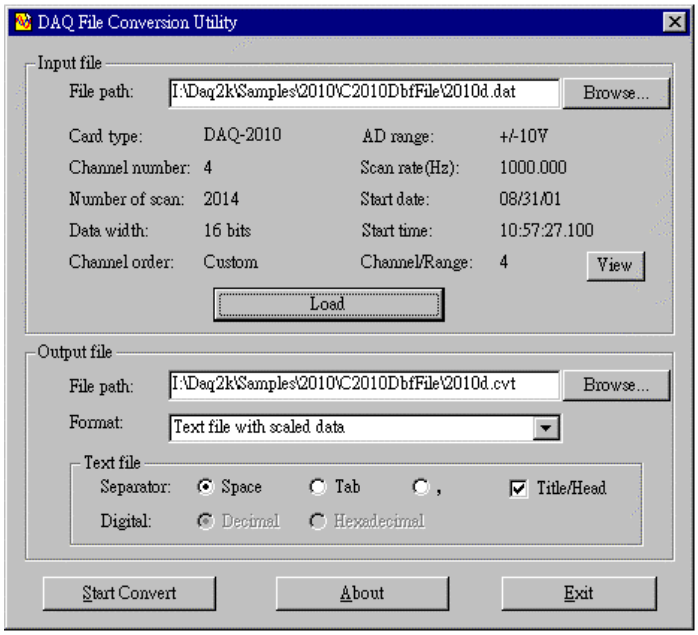
3.2 D2K-DASK Data File Converter utility (DAQCvt)

The data files, generated by the D2K-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, D2K-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. The *DAQCvt* main window is as the following figure:



The *DAQCvt* main window includes two frames. The upper frame, *Input File frame* is used for the source data file and the lower frame is used for the destination file.

To **load the source binary data file**, type the binary data file name in *File Path* field or click *Browser* button to select the source file from *Input File frame*, and then click *Load* button. As the file is loaded, the information related to the data file, e.g. *data type*, *data width*, *AD Range*, …etc., are shown in the corresponding fields in "Input File" frame, and the default converted data file path and format are also listed as the figure below.



The default **destination file** with a *.cvt* extension is located in the same directory as the source one. To change the default setting, type the file path you wish or click the *Browser* button from *Output File* frame to select the destination file location.

DAQCvt provides three types of data format conversion.

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

Binary file with scaled data :

The data in hexadecimal is scaled to engineering unit (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.

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

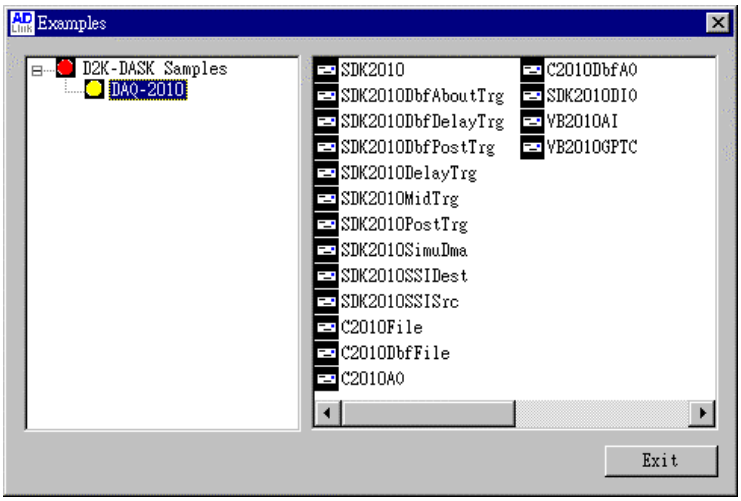
The data separator in converted text file is selectable among *space*, *comma* and *Tab*.

If you want to add title/head which includes the card type information at the beginning of file, check the “Title/Head” box.

After setting the properties (File Path, Format, …etc) related to the converted file, you can push *Start Convert* button from the *Output File* frame to perform the file conversion.

3.3 D2K-DASK Sample Programs Browser (Examples.exe)

D2K-DASK provides a sample program browser, ***Examples.exe***, for you to view and execute the sample programs that D2K-DASK package includes. The default location of this utility is <InstallDir>\Samples directory. After *Examples.exe* utility is running, select the device you wish to operate from the device list in the left frame, and then double click the icon of the sample you wish to execute to run this sample program.



4

D2K-DASK Overview

This chapter describes the classes of functions in D2K-DASK and briefly describes each function.

D2K-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**
  - 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
- **Digital Output Function Group**
  - Digital Output Configuration functions
  - One-Shot Digital Output functions
- **General Timer/Counter Function Group**
- **DIO Function Group**
  - Digital Input/Output Configuration function
- **SSI Function Group**
- **Calibration Function Group**

4.1 General Configuration Function Group

Use these functions to initializes and configures data acquisition card.

D2K_Register_Card Initializes the hardware and software states of an DAQ-2000 data acquisition card. Register_Card must be called before any other D2K-DASK library functions can be called for that card.

D2K_Release_Card Tells D2K-DASK library that this registered card is not used currently and can be released. This would make room for new card to register.

D2K_AIO_Config Informs D2K-DASK library of Timer source, and analog trigger source for the DAQ-2000 device.

4.2 Analog Input Function Group

4.2.1 Analog Input Configuration Functions

D2K_AI_CH_Config Informs D2K-DASK library of the AI range selected for the specified analog input channel of DAQ-2000 device. You must call this function before calling function to perform analog input operation.

D2K_AI_Config Informs D2K-DASK library of trigger source, trigger mode, input mode and trigger properties for the analog input operation of DAQ-2000 device. You must call this function before calling function to perform continuous analog input operation of DAQ-2000 device.

D2K_AI_MuxScanSetup
Informs stores *numChans*, *chans*, and *gain_refGnd* in the Channel-Gain Queue for a scanned data acquisition operation.

D2K_AI_InitialMemoryAllocated
Gets the actual size of analog input memory that is available in the device driver.

4.2.2 One-Shot Analog Input Functions

D2K_AI_ReadChannel
Performs a software triggered A/D conversion (analog input) on an analog input channel and returns the value converted (unscaled).

D2K_AI_ SimuReadChannel
Performs a software triggered A/D conversion (analog input) on analog input channels and returns the values converted (unscaled). This function is only available for Simultaneous AD card (e.q. DAQ-2010).

D2K_AI_ReadMuxScan
Returns readings for all analog input channels selected by *D2K_AI_MuxScanSetup*. This function is only available for Multiplexed AD card (e.q. DAQ-2205).

D2K_AI_ ScanReadChannels
Performs software triggered A/D conversions (analog input) on analog input channels and returns the values converted (unscaled). This function is only available for Multiplexed AD card (e.q. DAQ-2205).

D2K_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.

D2K_AI_VoltScale Converts the result from an D2K_AI_ReadChannel call to the actual input voltage.

4.2.3 Continuous Analog Input Functions

D2K_AI_ContReadChannel Performs continuous A/D conversions on the specified analog input channel at a rate as close to the rate you specified.

D2K_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.

D2K_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.

D2K_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.

D2K_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.

D2K_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.

D2K_AI_ContMuxScan This function initializes the Channel-Gain Queue to point to the start of the scan sequence as specified by *D2K_AI_MuxScanSetup* and starts a multiple-channel scanned data acquisition operation. This function is only available for ***Multiplexed AD card*** (e.g. DAQ-2205)

D2K_AI_ContMuxScanToFile Initializes the Channel-Gain Queue to point to the start of the scan sequence as specified by *D2K_AI_MuxScanSetup*, starts a multiple-channel scanned data acquisition operation and saves the acquired data in a disk file.

D2K_AI_ContVScale Converts the values of an array of acquired data from an continuous A/D conversion call to the actual input voltages.

D2K_AI_ContStatus Checks the current status of the continuous analog input operation.

D2K_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.

D2K_AI_ContBufferSetup Set up the buffer for continuous analog input.

D2K_AI_ContBufferReset Reset all the buffers set by function *D2K_AI_ContBufferSetup*.

4.2.4 Asynchronous Analog Input Monitoring Functions

D2K_AI_AsyncCheck Checks the current status of the asynchronous analog input operation.

D2K_AI_AsyncClear Stops the asynchronous analog input operation.

D2K_AI_AsyncDblBufferMode
Enables or Disables double buffer data acquisition mode.

D2K_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.

D2K_AI_AsyncDblBufferToFile
Copies half of the data of circular buffer into a disk file.

D2K_AI_AsyncDblBufferOverrun

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

D2K_AI_AsyncDblBufferHandled
Notifies D2K-Dask the ready buffer has been handled in user application.

D2K_AI_AsyncReTrigNextReady
Checks whether the data associated to the next trigger signal is ready during an asynchronous re-triggered analog input operation.

4.3 Analog Output Function Group

4.3.1 Analog output Configuration Functions

D2K_AO_CH_Config Informs D2K-DASK library of the reference voltage value selected for an analog output channel of DAQ-2000 Device. You must call this function before calling function to perform voltage output operation.

D2K_AO_Config Informs D2K-DASK library of trigger source, trigger mode, output mode and trigger properties for the analog output operation of DAQ-2000 device. You must call this function before calling function to perform continuous analog output operation of DAQ-2000 device.

D2K_AO_InitialMemoryAllocated
Gets the actual size of analog output DMA memory that is available in the device driver.

D2K_AO_Group_Setup
Assigns one or more analog output channels to a waveform generation group.

D2K_AO_Group_WFM_StopConfig
Informs D2K-DASK library of stop source and stop mode for the asynchronous analog output operation of a specified group.

4.3.2 One-Shot Analog Output Functions

D2K_AO_WriteChannel
Writes a binary value to the specified analog output channel.

D2K_AO_SimuWriteChannel
Writes binary values to the specified analog output channels simultaneously. This function is only available for Simultaneous DA card.

D2K_AO_VWriteChannel
Accepts a voltage value, scales it to the proper binary value and writes a binary value to the specified analog output channel.

D2K_AO_VoltScale Scales a voltage to a binary value.

D2K_AO_Group_Update
Writes binary values to the specified group of analog output channels simultaneously.

D2K_AO_Group_VUpdate
Accepts voltage values, scales them to the proper binary values and writes binary values to the specified group of analog output channels simultaneously.

4.3.3 Continuous Analog Output Functions

D2K_AO_ContWriteChannel Performs continuous analog output on the specified analog output port at a rate as close to the rate you specified.

D2K_AO_ContWriteMultiChannels
Performs continuous D/A conversions on the specified analog output channels at a rate as close to the rate you specified.

D2K_AO_ContStatus Checks the current status of the continuous analog output operation.

D2K_AO_EventCallBack Controls and notifies the user's application when a specified DAQ event occurs. The notification is performed through a user-specified callback function.

D2K_AO_ContBufferSetup This function set up the buffer for continuous analog output

D2K_AO_ContReset This function reset all the buffers set by function *D2K_AO_ContBufferSetup* for continuous analog output

D2K_AO_ContBufferCompose The function organizes the data for each channel and filled them in the buffer for continuous analog output operation

D2K_AO_ContBufferComposeAll
The function fills the data for a specified channel in the buffer for continuous analog output operation

D2K_AO_Group_FIFOLoad Loads a waveform buffer to on-board DA FIFOs

D2K_AO_Group_WFM_Start Performs continuous D/A conversions on the specified group of analog output channels at a rate as close to the rate you specified.

4.3.4 Asynchronous Analog Output Monitoring Functions

D2K_AO_AsyncCheck Checks the current status of the asynchronous analog output operation.

D2K_AO_AsyncClear Stops the asynchronous analog output operation.

D2K_AO_AsyncDblBufferMode
Enables or Disables double buffer data acquisition mode.

D2K_AO_AsyncDblBufferHalfReady
Checks whether the next half buffer of data in circular buffer is ready during an asynchronous double-buffered analog output operation.

D2K_AO_Group_WFM_AsyncCheck
Checks the current status of the asynchronous analog output operation of a specified group.

D2K_AO_ Group_WFM _AsyncClear
Stops the asynchronous analog output operation of a specified group.

4.4 Digital Input Function Group

4.4.1 One-Shot Digital Input Functions

D2K_DI_ReadLine Reads the digital logic state of the specified digital line in the specified port.

D2K_DI_ReadPort Reads digital data from the specified digital input port.

4.5 Digital Output Function Group

4.5.1 One-Shot Digital Output Functions

D2K_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.

D2K_DO_WritePort
Writes digital data to the specified digital output port.

D2K_DO_ReadLine

Reads the specified digital output line in the specified digital output port.

D2K_DO_ReadPort
Reads digital data from the specified digital output port.

4.6 General Timer/Counter Function Group

4.6.1 The General-Purpose Timer/Counter Functions

D2K_GCTR_Setup	Controls the general-purpose counter to operate in the specified mode.
D2K_GCTR_Read	Reads the counter value of the general-purpose counter without disturbing the counting process.
D2K_GCTR_Control	Controls for the selected counter/timer by software
D2K_GCTR_Reset	Halts the specified general-purpose timer/counter operation and reload the initial value of the timer/counter.
D2K_GCTR_Status	Reads the counter value of the general-purpose counter without disturbing the counting process.

4.7 DIO Function Group

4.7.1 Digital Input/Output Configuration Functions

D2K_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 D2K-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 SSI Function Group

D2K_SSI_SourceConn	Connets a device to the specified SSI bus trigger line.
D2K_SSI_SourceDisConn	Disconnects a device signal from the specified SSI bus trigger line.
D2K_SSI_SourceClear	Disconnects a device signal from the specified SSI bus trigger line

4.9 Calibration Function Group

D2K_DB_Auto_Calibration_ALL
Calibrates your DAQ-2000 device

D2K_EEPROM_CAL_Constant_Update
Save new calibration constants to the specified *bank* of EEPROM

D2K_Load_CAL_Data Load calibration constants from the specified *bank* of EEPROM

DAQ2010_Acquire_AD_Error
Acquires the offset and gain errors of the specified AI channel in the specified polarity mode

DAQ2010_Acquire_DA_Error
Acquires the offset and gain errors of the specified DA channel in the specified polarity mode

DAQ2005_Acquire_AD_Error
Acquires the offset and gain errors of the specified AI channel in the specified polarity mode

DAQ2005_Acquire_DA_Error
Acquires the offset and gain errors of the specified DA channel in the specified polarity mode

DAQ2006_Acquire_AD_Error
Acquires the offset and gain errors of the specified AI channel in the specified polarity mode

DAQ2006_Acquire_DA_Error
Acquires the offset and gain errors of the specified DA channel in the specified polarity mode

DAQ2204_Acquire_AD_Error
Acquires the offset and gain errors of ADC

DAQ2204_Acquire_DA_Error
Acquires the offset and gain errors of the specified DA channel in the specified polarity mode

DAQ2205_Acquire_AD_Error
Acquires the offset and gain errors of ADC

DAQ2205_Acquire_DA_Error
Acquires the offset and gain errors of the specified DA channel in the specified polarity mode

DAQ2206_Acquire_AD_Error
Acquires the offset and gain errors of ADC

DAQ2206_Acquire_DA_Error
Acquires the offset and gain errors of the specified DA channel in the specified polarity mode

DAQ2208_Acquire_AD_Error
Acquires the offset and gain errors of ADC

DAQ2213_Acquire_AD_Error
Acquires the offset and gain errors of ADC

DAQ2214_Acquire_AD_Error
Acquires the offset and gain errors of ADC

DAQ2214_Acquire_DA_Error
Acquires the offset and gain errors of the specified DA channel in the specified polarity mode

DAQ250X_Acquire_DA_Error
Acquires the offset and gain errors of the specified DA channel in the specified polarity mode

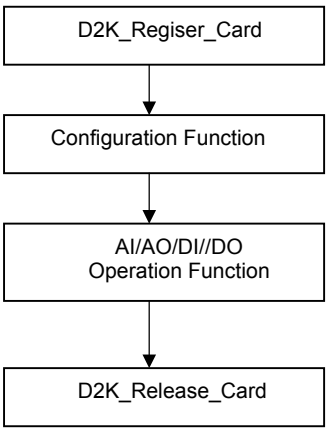
DAQ250X_Acquire_AD_Error
Acquires the offset and gain errors of ADC

5

D2K-DASK Application Hints

This chapter provides the programming schemes showing the function flow of that D2K-DASK performs analog I/O and digital I/O.

The figure below shows the basic building blocks of a D2K-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 D2K-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.

5.1 Analog Input Programming Hints

D2K-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 programming scheme for this kind of AI operation is described in section 5.1.1.

The buffered continuous analog input 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 analog input in the driver. The driver allocates the memory at system boot time (in Window NT) or Windows startup time (in Window 98). We recommend the applications use *D2K_AI_InitialMemoryAllocated* function to get the size of initially allocated memory before performing continuous AI operation.

The buffered continuous analog input includes:

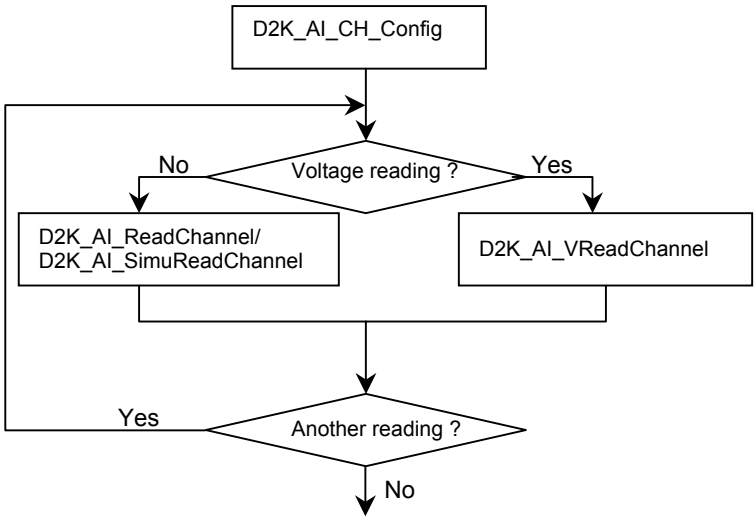
- synchronous continuous AI
- non-double-buffered asynchronous continuous AI
- double-buffered asynchronous continuous AI
- pre/middle triggered non-double-buffered asynchronous continuous AI
- pre/middle triggered double-buffered asynchronous continuous AI

They are described in section 5.1.2 to 5.1.7 section respectively. About the special consideration and performance issues for the buffered continuous analog input, please refer to the *Continuous Data Transfer in D2K-DASK* chapter for the details.

5.1.1 One-Shot Analog input programming Scheme

This section described the function flow typical of non-buffered single-point analog input readings.

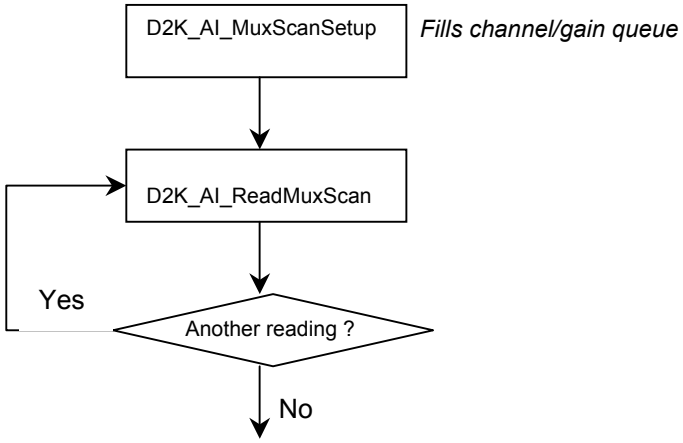
a. all types of DAQ-2000 series



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
            …
D2K_AI_CH_Config (card, channelNo, AD_B_10_V);
D2K_AI_ReadChannel(card, channelNo, &analog_input[i]);
            …
D2K_Release_Card(card);
```

b. *Fills channel gain queue first – only available for DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2208*



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2204, card_number);
CHANNELCOUNT = 1;
chans[0] = 0;
ranges[0] = AD_B_10_V| AI_RSE;
D2K_AI_MuxScanSetup(card, CHANNELCOUNT, chans, ranges);
D2K_AI_ReadMuxScan (card, chan_data);
            …
D2K_Release_Card(card);
```

5.1.2 Continuous Analog input (with initial default settings) programming Scheme

This section described the function flow typical of synchronous analog input operation performed by the device with initial default configuration. For synchronous AI, the *SyncMode* argument in continuous AI functions has to be set as *SYNCH_OP* and for asynchronous AI, the *SyncMode* argument has to be set as *ASYNCH_OP.*

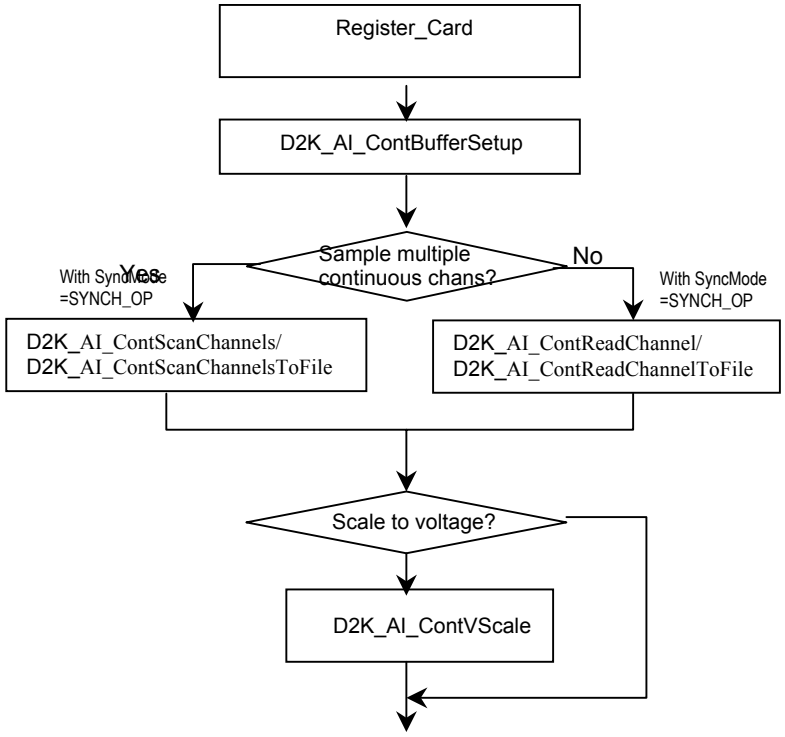
Initial default channel configuration:

AD data range	AD_B_10_V
Reference ground (only available for DAQ-2204, DAQ-2205, DAQ-2206 and DAQ-2208)	AI_RSE

Initial default AI configuration:

A/D Conversion source	DAQ2K_AI_ADCONVSRC_Int (Internal timer pacer)
A/D Trigger mode	DAQ2K_AI_TRGMOD_POST (post trigger)
A/D Trigger source	DAQ2K_AI_TRGSRC_SOFT (software trigger)
Auto buffer reset	TRUE

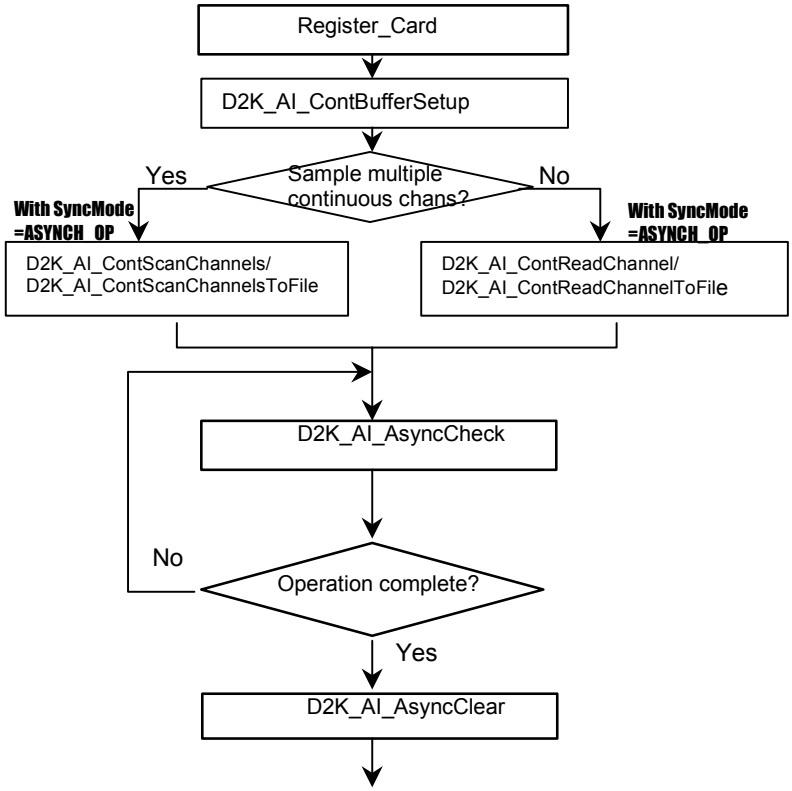
a. Synchronous Operation



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &Id);
D2K_AI_ContScanChannels (card, channel, Id, data_size/(channel+1), scan_intrv, samp_intrv, SYNCH_OP); or
D2K_AI_ContReadChannel(card, channel, Id, data_size, scan_intrv, samp_intrv, SYNCH_OP)
…
D2K_Release_Card(card);
```

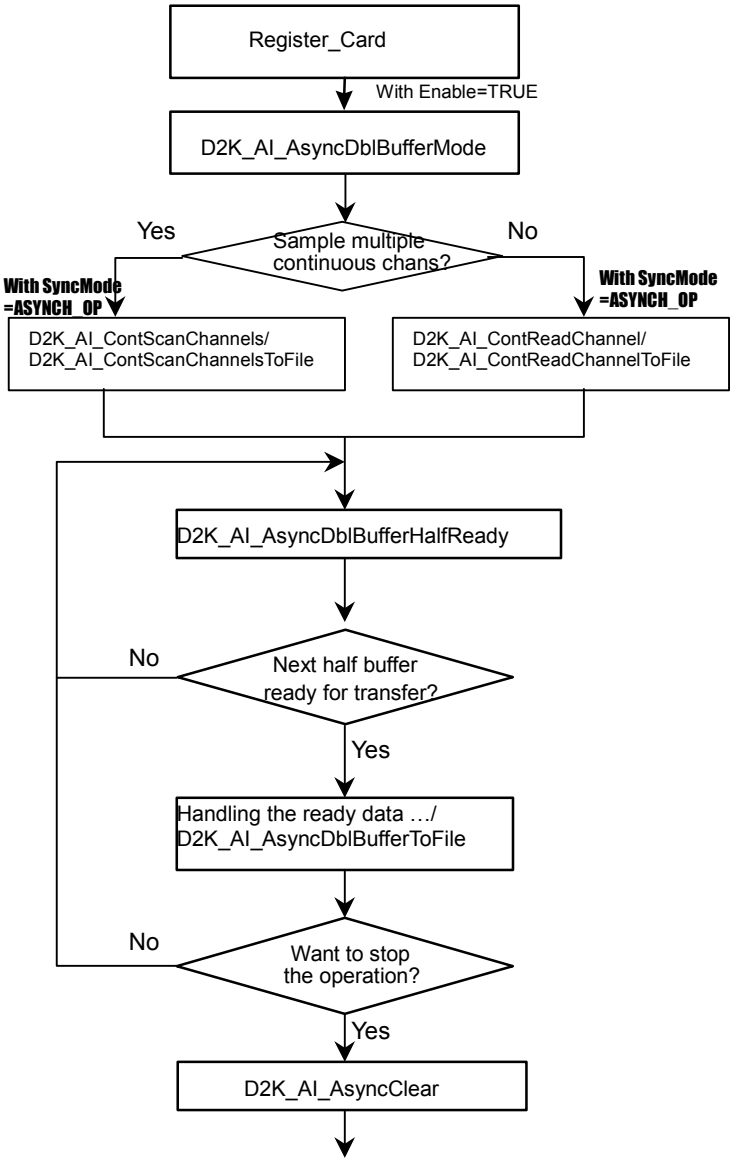
b. Non-Double Buffered Asynchronous Operation



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_AI_ContScanChannels (card, channel, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP);
or
D2K_AI_ContReadChannel(card, channel, BufId, data_size, ScanIntrv, SampIntrv, ASYNCH_OP)
do {
      D2K_AI_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);

D2K_AI_AsyncClear(card, &StartPos, &count);
  …
D2K_Release_Card(card);
```

c. Double Buffered Asynchronous Operation



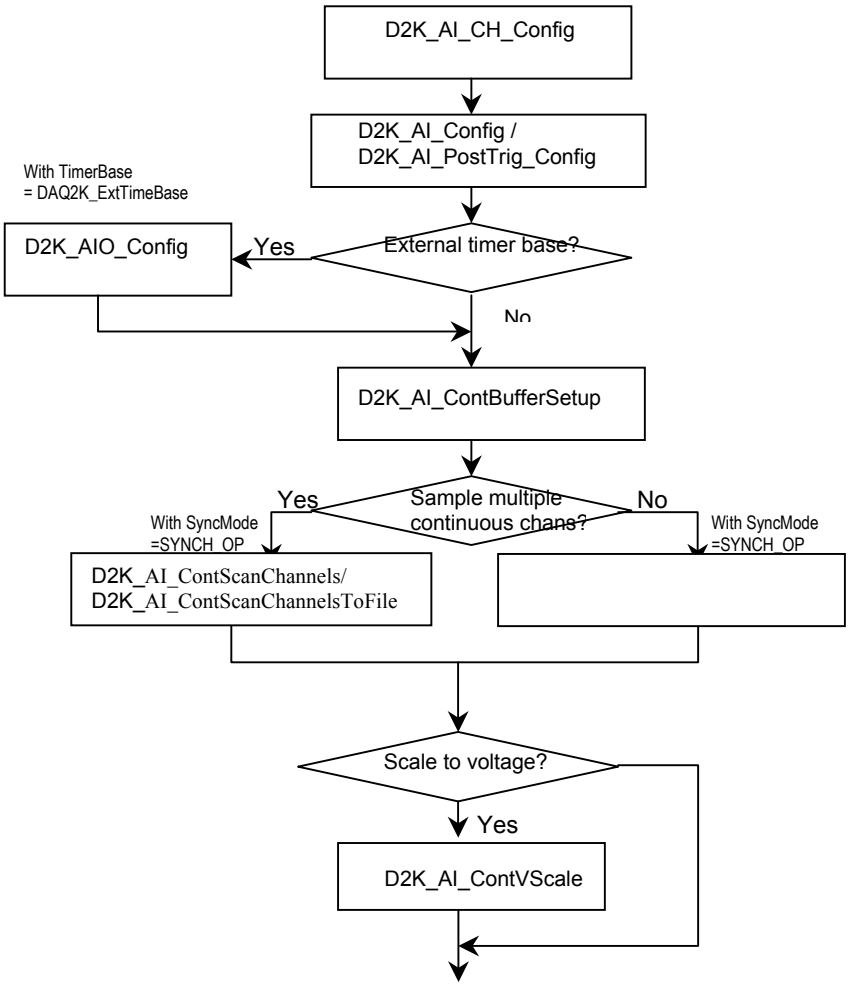
[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
            …
D2K_AI_AsyncDblBufferMode (card, 1); // Double-buffered AI
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_A_ContBufferSetup (card, ai_buf2, data_size, &BufId);
D2K_AI_ContScanChannels (card, channel, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP);
    or
D2K_AI_ContReadChannel(card, channel, BufId, data_size, ScanIntrv, SampIntrv, ASYNCH_OP)
do {
    do {
        D2K_AI_AsyncDblBufferHalfReady(card, &HalfReady, &fstop);
    } while (!HalfReady);

  //Handling the ready data
  …
} while (!clear_op);

D2K_AI_AsyncClear(card, &startPos, &count);
…
D2K_Release_Card(card);
```

5.1.3 Post Trigger Mode/ Delay Trigger Mode Synchronous Continuous Analog input programming Scheme

This section described the function flow typical of post trigger or delay triggered 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.*

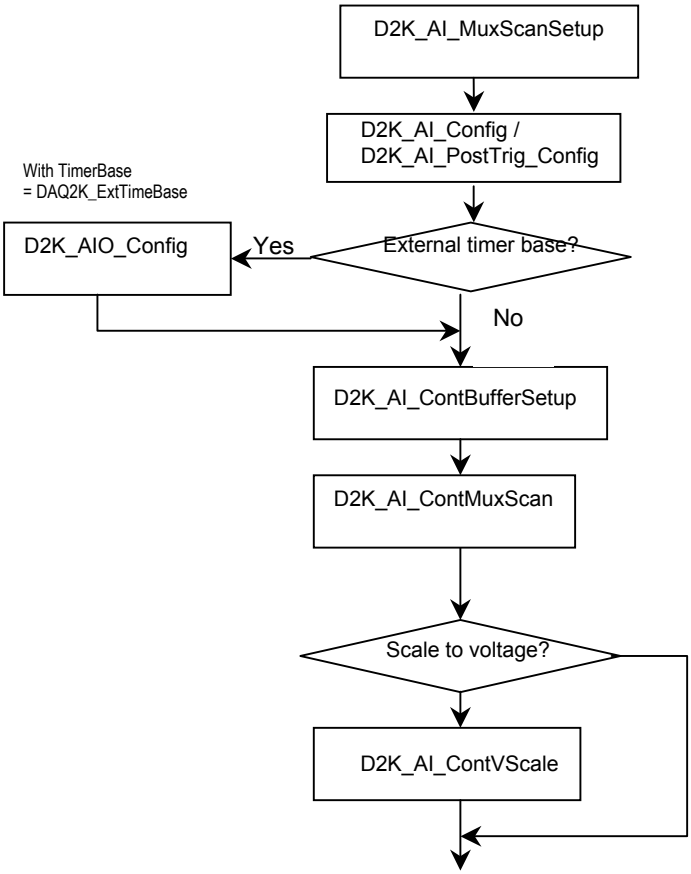
a. all types of DAQ-2000 series



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
D2K_AI_CH_Config (card, channel, range )
D2K_AI_Config (card, 0, DAQ2K_AI_TRGMOD_POST| DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 0, 1);
 // or
 // D2K_AI_PostTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 1);
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &Id);
D2K_AI_ContScanChannels (card, channel, Id, data_size/(channel+1), scan_intrv, samp_intrv, SYNCH_OP); or
D2K_AI_ContReadChannel(card, channel, Id, data_size, scan_intrv, samp_intrv, SYNCH_OP)
…
D2K_Release_Card(card);
```

b. Fills channel gain queue first – only available for DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2208



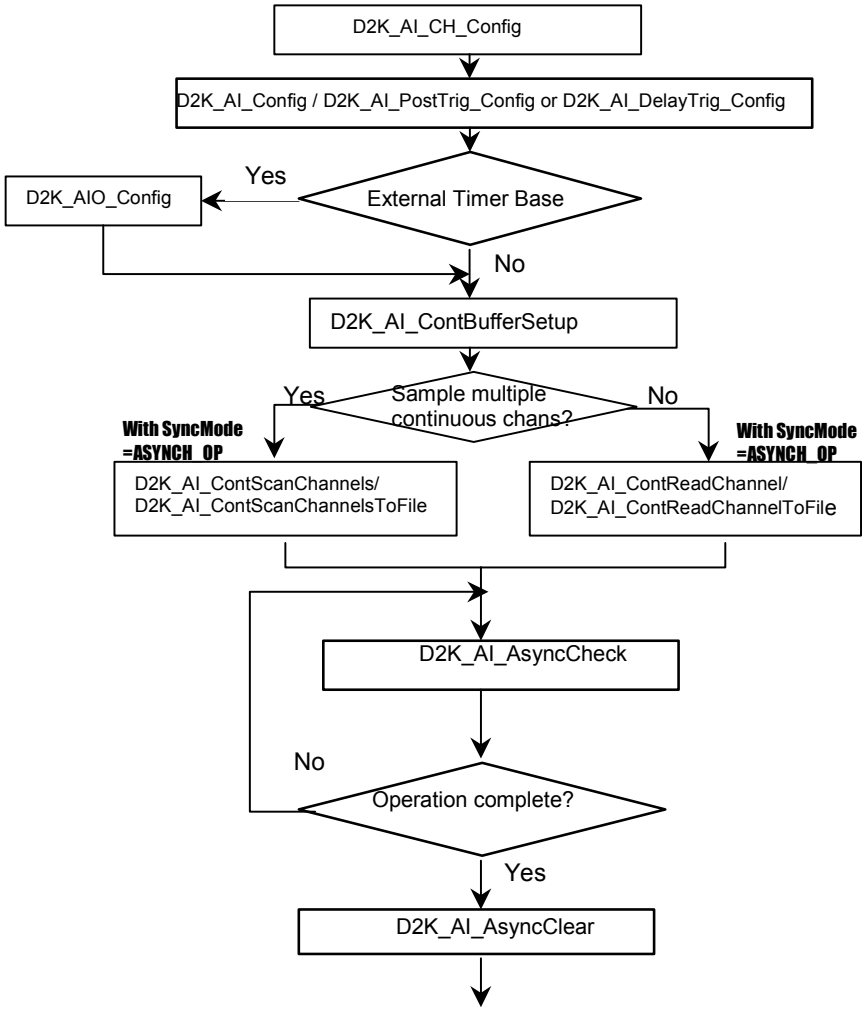
[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2205, card_number);
 CHANNELCOUNT = 1;
 chans[0] = 0;
 ranges[0] = AD_B_10_V| AI_RSE;
 D2K_AI_MuxScanSetup(card, CHANNELCOUNT, chans, ranges);
D2K_AI_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGMOD_POST| DAQ2K_AI_TRGSRC_ExtD|
    DAQ2K_AI_TrgPositive, 0, 0, 0, 1);
 // or
 // D2K_AI_PostTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 1);
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &Id);
D2K_AI_ContMuxScan (card, Id, data_size/CHANNELCOUNT,  SAMPLE_INTERVAL*CHANNELCOUNT,
       SAMPLE_INTERVAL, SYNCH_OP);
       …
D2K_Release_Card(card);
```

5.1.4 Post Trigger Mode/ Delay Trigger Mode Non-double-buffered Asynchronous Continuous Analog input programming Scheme

This section described the function flow typical of post trigger or delay triggered, 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.*

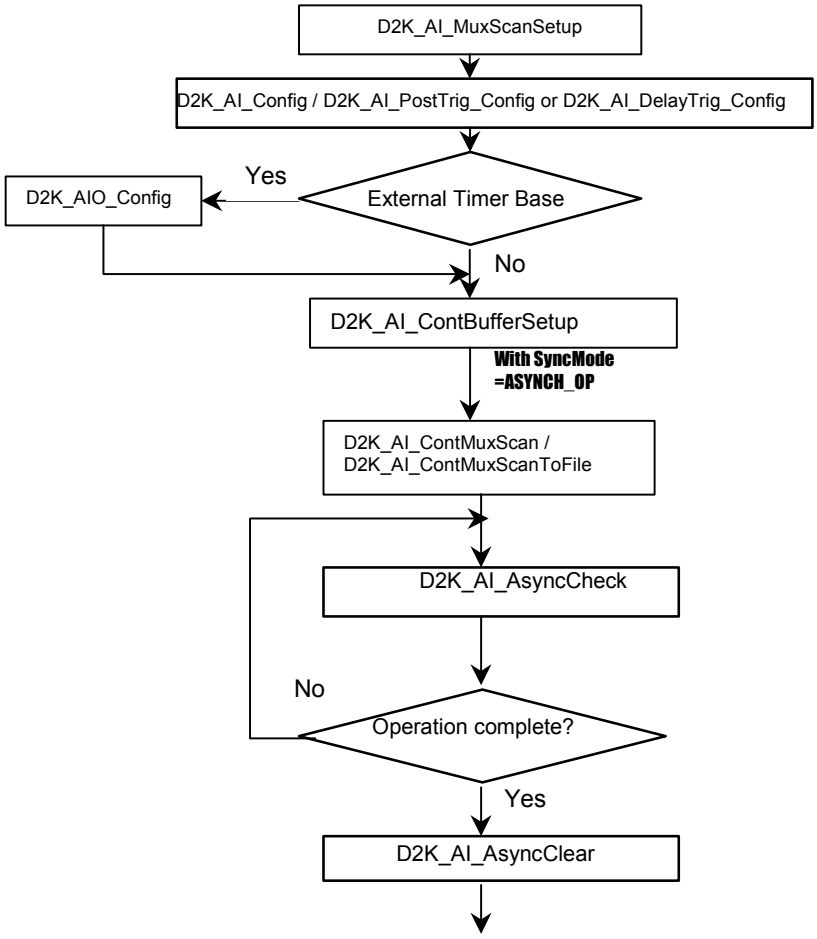
a. all types of DAQ-2000 series



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
…
D2K_AI_CH_Config (card, channel, range )
D2K_AI_Config (card, 0,  DAQ2K_AI_TRGMOD_POST| DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 0, 1);
 // or
 // D2K_AI_PostTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 1);
 D2k_AI_AsyncDblBufferMode (card, 0); //non-double-buffered AI
 D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
 D2K_AI_ContScanChannels (card, channel, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP);
     or
 D2K_AI_ContReadChannel(card, channel, BufId, data_size, ScanIntrv, SampIntrv, ASYNCH_OP)
 do {
       D2K_AI_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);

D2K_AI_AsyncClear(card, &StartPos, &count);
 …
D2K_Release_Card(card);
```

b. Fills channel gain queue first – only available for DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2208



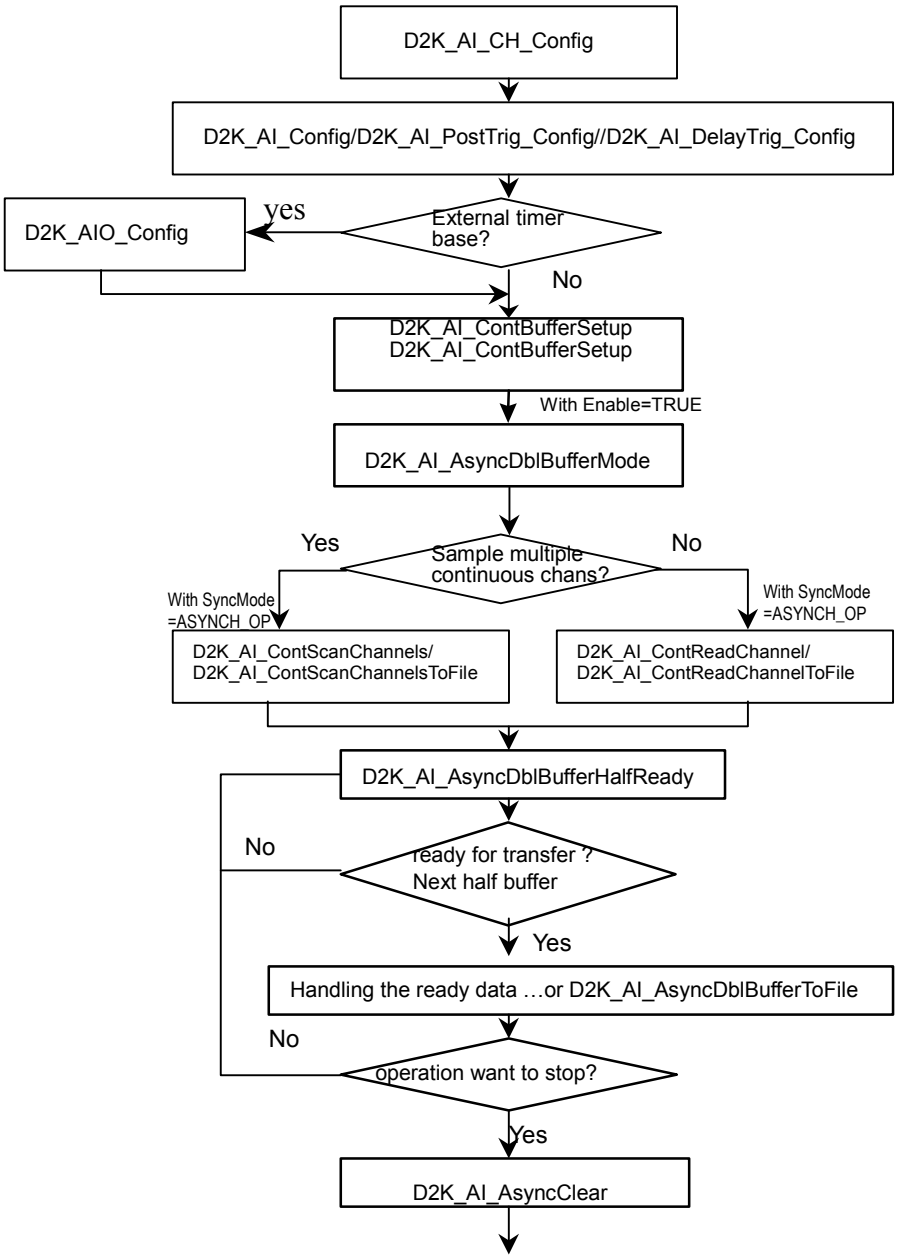
[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2205, card_number);
…
CHANNELCOUNT = 1;
chans[0] = 0;
ranges[0] = AD_B_10_V| AI_RSE;
D2K_AI_MuxScanSetup(card, CHANNELCOUNT, chans, ranges);
D2K_AI_Config (card, 0,  DAQ2K_AI_TRGMOD_POST| DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 0, 1);
// or
// D2K_AI_PostTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 1);
D2k_AI_AsyncDblBufferMode (card, 0); //non-double-buffered AI
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_AI_ContMuxScan (card, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP);
do {
      D2K_AI_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);

D2K_AI_AsyncClear(card, &StartPos, &count);
 …
D2K_Release_Card(card);
```

5.1.5 Post Trigger Mode/ Delay Trigger Mode Double-buffered Asynchronous Continuous Analog input programming Scheme

This section described the function flow typical of post trigger or delay triggered, 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 *D2K_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.

a. all types of DAQ-2000 series

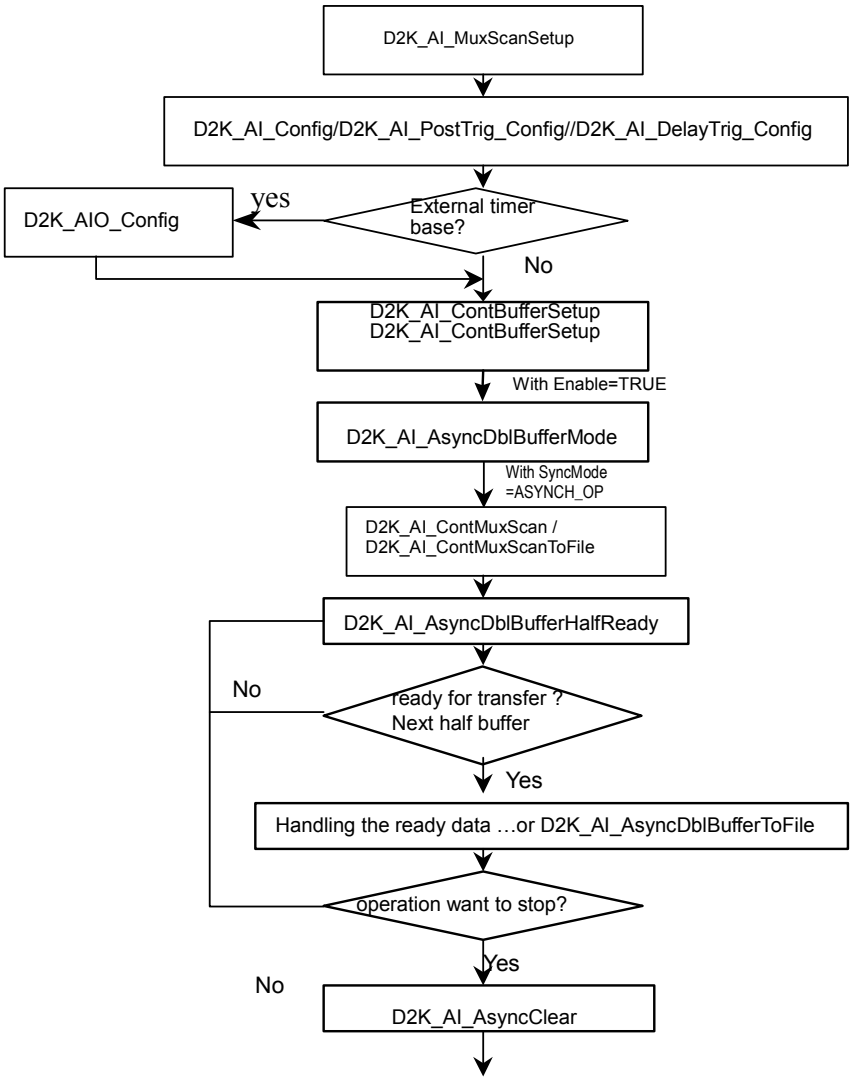


[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
                …
D2K_AI_CH_Config (card, channel, range )
D2K_AI_Config (card, 0,  DAQ2K_AI_TRGMOD_POST| DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 0, 1);
// or
// D2K_AI_PostTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 1);
D2K_AI_AsyncDblBufferMode (card, 1); // Double-buffered AI
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_A_ContBufferSetup (card, ai_buf2, data_size, &BufId);
D2K_AI_ContScanChannels (card, channel, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP);
    or
D2K_AI_ContReadChannel(card, channel, BufId, data_size, ScanIntrv, SampIntrv, ASYNCH_OP)
do {
    do {
        D2K_AI_AsyncDblBufferHalfReady(card, &HalfReady, &fstop);
    } while (!HalfReady);

  //Handling the ready data
  …
} while (!clear_op);
```

D2K_AI_AsyncClear(card, &startPos, &count);
…
D2K_Release_Card(card);

b. Fills channel gain queue first – only available for DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2208



[Example Code Fragment]

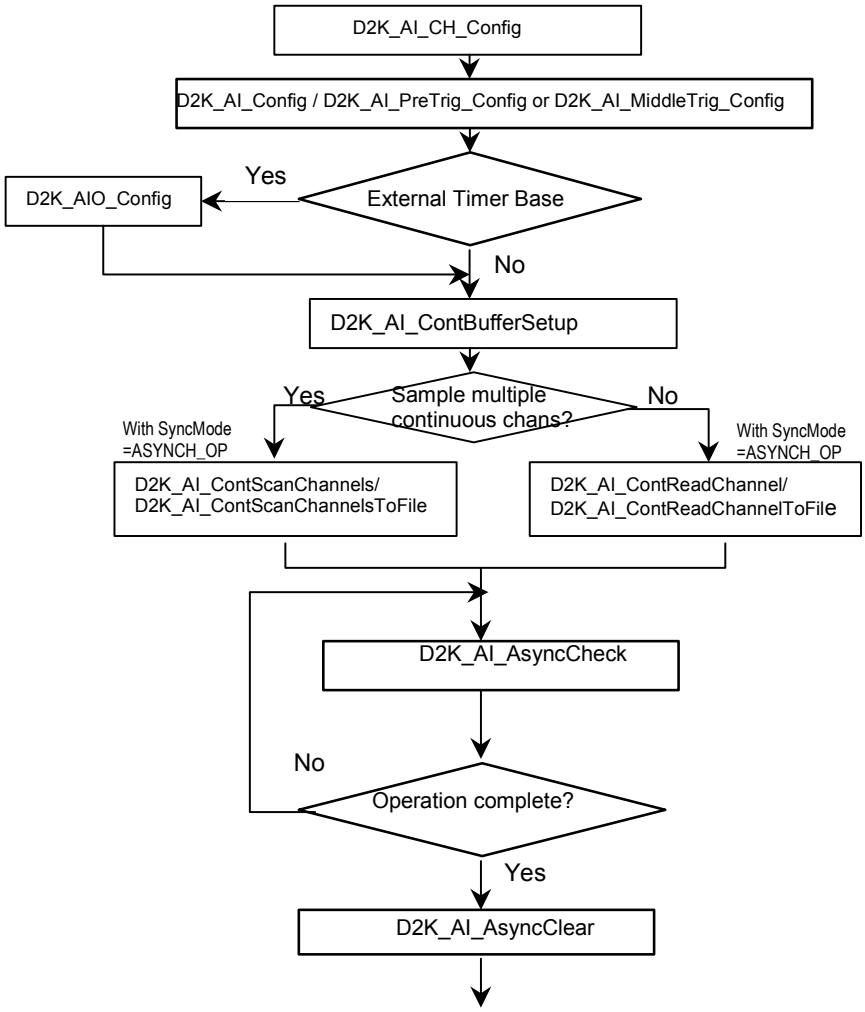
```
card = D2K_Register_Card(DAQ_2205, card_number);
                …
CHANNELCOUNT = 1;
chans[0] = 0;
ranges[0] = AD_B_10_V| AI_RSE;
D2K_AI_MuxScanSetup(card, CHANNELCOUNT, chans, ranges);
D2K_AI_Config (card, 0,  DAQ2K_AI_TRGMOD_POST| DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 0, 1);
// or
 // D2K_AI_PostTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 1);
D2K_AI_AsyncDblBufferMode (card, 1); // Double-buffered AI
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_A_ContBufferSetup (card, ai_buf2, data_size, &BufId);
D2K_AI_ContMuxScan (card, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP);
do {
     do {
         D2K_AI_AsyncDblBufferHalfReady(card, &HalfReady, &fstop);
     } while (!HalfReady);

   //Handling the ready data
   …
```

```
} while (!clear_op);

D2K_AI_AsyncClear(card, &startPos, &count);
…
D2K_Release_Card(card);
```

5.1.6 Pre-Trigger Mode/ Middle-Trigger Mode Non-double-buffered Asynchronous Continuous Analog input programming Scheme

This section described the function flow typical of Pre-trigger and middle 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 D2K-DASK to perform pre-trigger or middle mode data acquisition, the *SyncMode* of continuous AI should be set as *ASYNCH_OP*.

a. all types of DAQ-2000 series

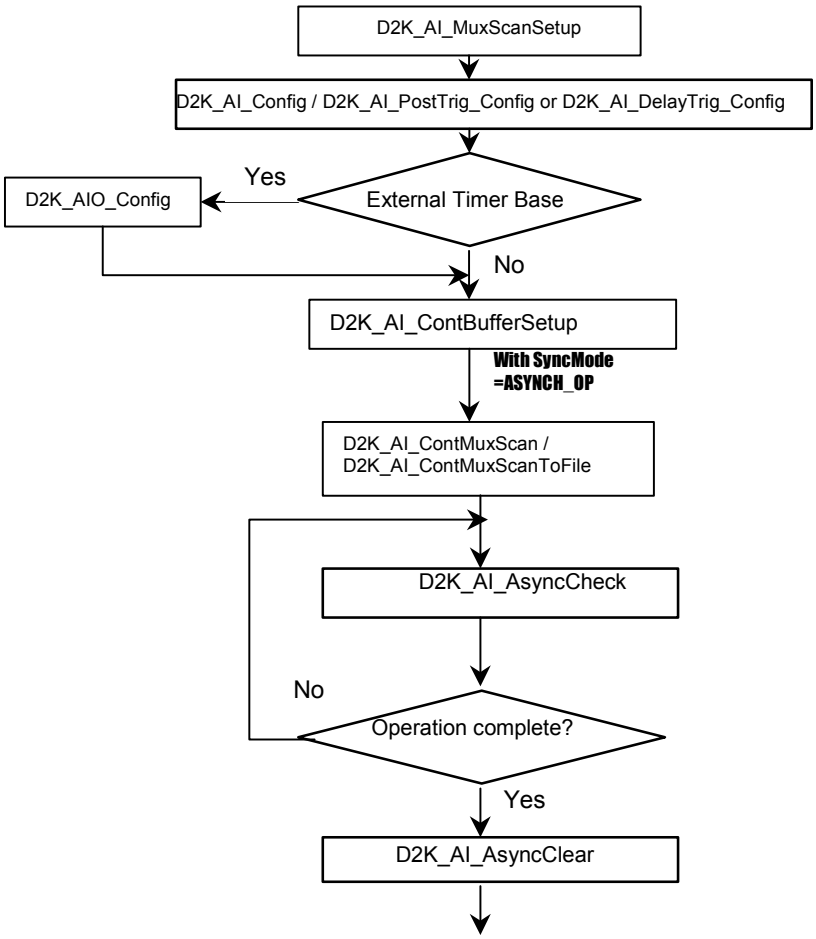


[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
                …
D2K_AI_CH_Config (card, channel, range )
D2K_AI_Config (card, 0,  DAQ2K_AI_TRGMOD_PRE| DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 0, 1);
// or
// D2K_AI_PreTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 1);
D2K_AI_AsyncDblBufferMode (card, 0); //non-double-buffered AI
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_AI_ContScanChannels (card, channel, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP);
    or
D2K_AI_ContReadChannel(card, channel, BufId, data_size, ScanIntrv, SampIntrv, ASYNCH_OP)
```

```
do {
      D2K_AI_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);

D2K_AI_AsyncClear(card, &startPos, &count);
…
D2K_Release_Card(card);
```

b. Fills channel gain queue first – only available for DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2208



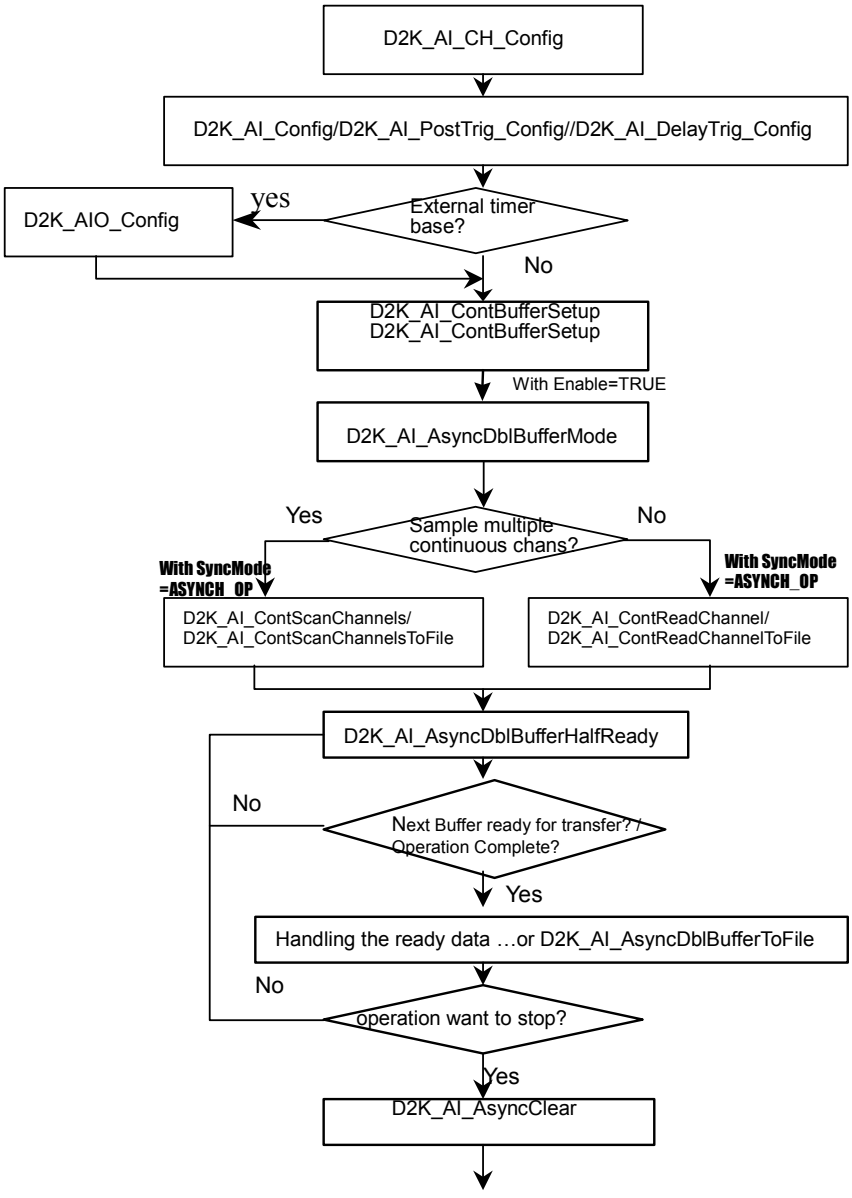
[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2205, card_number);
                …
CHANNELCOUNT = 1;
chans[0] = 0;
ranges[0] = AD_B_10_V| AI_RSE;
D2K_AI_MuxScanSetup(card, CHANNELCOUNT, chans, ranges);
D2K_AI_Config (card, 0,  DAQ2K_AI_TRGMOD_MIDL| DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, POSTCOUNT, 0,
     0, 1);
// or
// D2K_AI_MiddleTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive,
   POSTCOUNT, 0, 0, 1);
D2K_AI_AsyncDblBufferMode (card, 0); //non-double-buffered AI
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_AI_ContMuxScan (card, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP);
do {
      D2K_AI_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);

D2K_AI_AsyncClear(card, &startPos, &count);
…
D2K_Release_Card(card);
```

5.1.7 Pre-Trigger Mode/ Middle-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 D2K-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 *D2K_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.

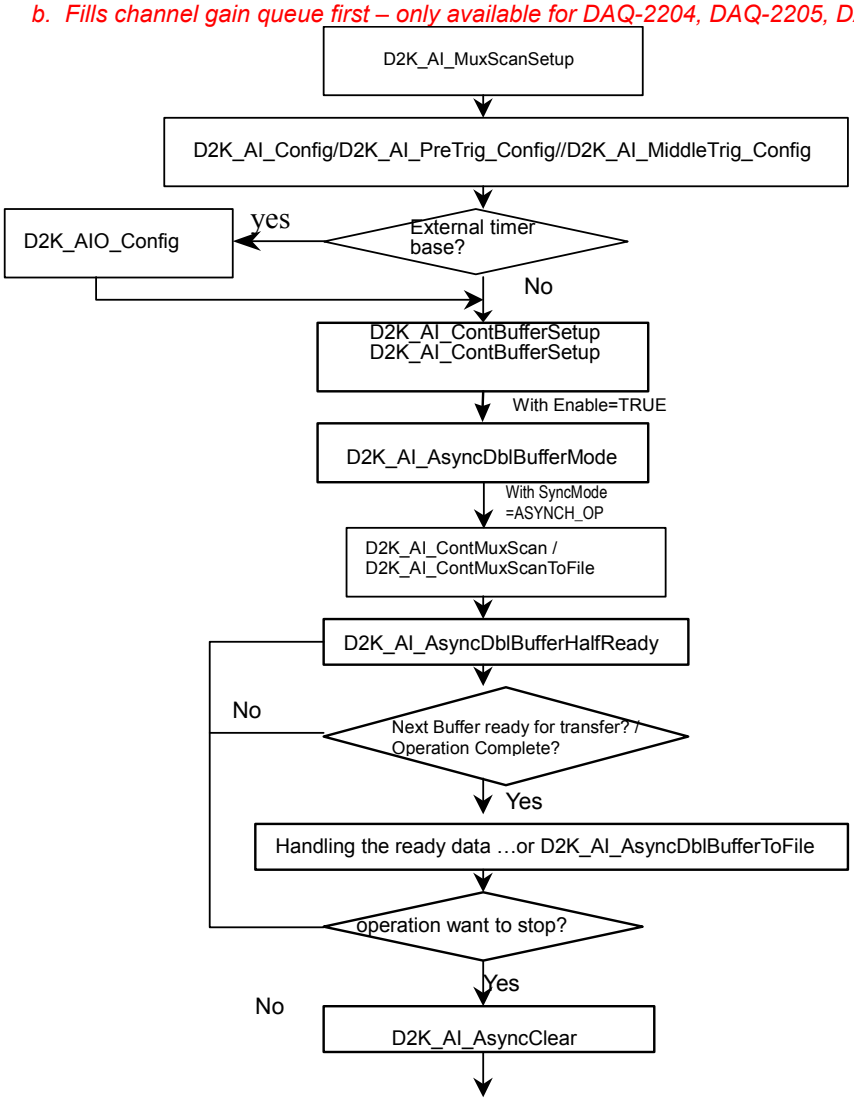
a. all types of DAQ-2000 series



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
                …
D2K_AI_CH_Config (card, channel, range )
D2K_AI_Config (card, 0,  DAQ2K_AI_TRGMOD_PRE|DAQ2K_AI_TRGSRC_ExtD, 0, 0, 0, 1);
// or
 // D2K_AI_MiddleTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0,
    1);
AI_AsyncDblBufferMode (card, 1);  Double-buffered AI
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_A_ContBufferSetup (card, ai_buf2, data_size, &BufId);
D2K_AI_ContScanChannels (card, channel, BufId, data_size/(channel+1), ScanIntrv, U32 SampIntrv,
     ASYNCH_OP); or
D2K_AI_ContReadChannel(card, channel, BufId, data_size, ScanIntrv, U32 SampIntrv, ASYNCH_OP)
do {
     do {
          D2K_AI_AsyncDblBufferHalfReady(card, &HalfReady, &fstop);
     } while (!HalfReady && !fstop);

   //handling the ready data …
     …

} while (!clear_op && !fstop);

D2K_AI_AsyncClear(card, &startPos, &count);
…
D2K_Release_Card(card);
```

b. Fills channel gain queue first – only available for DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2208



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2205, card_number);
                …
CHANNELCOUNT = 1;
chans[0] = 0;
ranges[0] = AD_B_10_V| AI_RSE;
D2K_AI_MuxScanSetup(card, CHANNELCOUNT, chans, ranges);
D2K_AI_Config (card, 0,  DAQ2K_AI_TRGMOD_RRE| DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 0, 1);
// or
 // D2K_AI_PreTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 1);
D2K_AI_AsyncDblBufferMode (card, 1); // Double-buffered AI
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_A_ContBufferSetup (card, ai_buf2, data_size, &BufId);
D2K_AI_ContMuxScan (card, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP);
do {
     do {
          D2K_AI_AsyncDblBufferHalfReady(card, &HalfReady, &fstop);
     } while (!HalfReady);

   //Handling the ready data
   …
} while (!clear_op);

D2K_AI_AsyncClear(card, &startPos, &count);
…
D2K_Release_Card(card);
```

5.2 Analog Output Programming Hints

D2K-DASK provides two kinds of analog output operation — 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 5.2.1.

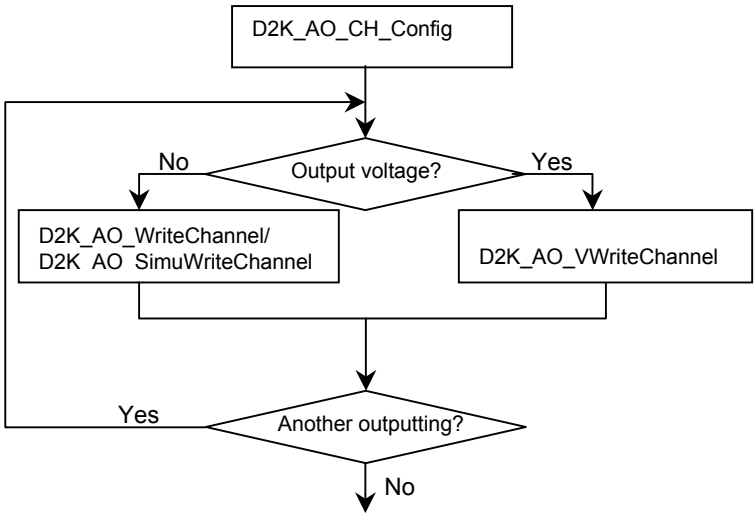
The buffered continuous AO uses DMA transfer method to transfer data from user's buffer to device. The maximum number of count in one transfer depends on the size of initially allocated memory for analog output in the driver. The driver allocates the memory at system boot time (in Window NT) or Windows startup time (in Window 98). We recommend the applications use *D2K_AO_InitialMemoryAllocated* function to get the size of initially allocated memory before start performing continuous AO operation.

About the special consideration and performance issues for the buffered continuous digital output, please refer to the *Continuous Data Transfer in D2K-DASK* chapter for the details.

5.2.1 One-Shot 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.

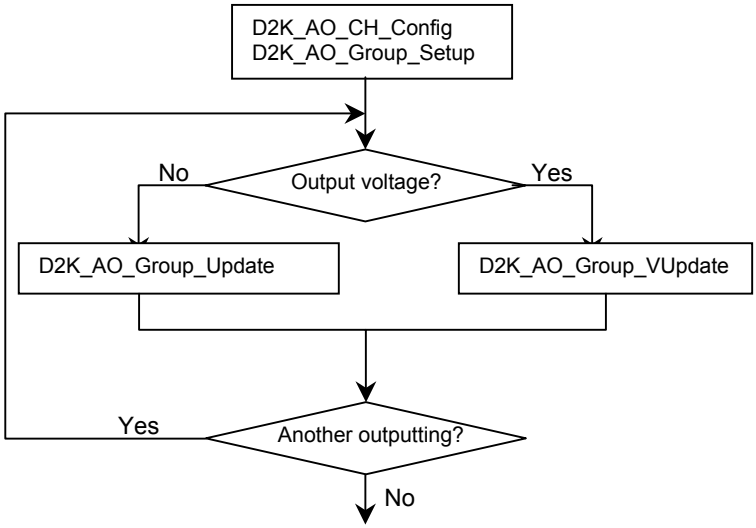
a. DAQ-2005, DAQ-2006, DAQ-2010, DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2214



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
        …
D2K_AO_CH_Config (card, 0, DAQ2K_DA_BiPolar, DAQ2K_DA_Int_REF, 10.0);
D2K_AO_WriteChannel(card, chan, out_value);
        …
D2K_Release_Card(card);
```

b. DAQ-2501, DAQ2502



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2501, card_number);
 da_ch = 0;
D2K_AO_CH_Config (card, da_ch, DAQ2K_DA_BiPolar, DAQ2K_DA_Int_REF, 10.0);
D2K_AO_Group_Setup (card, DA_Group_A, 1, &da_ch);
D2K_AO_Group_VUpdate (card, DA_Group_A, &out_V);
                …
D2K_Release_Card(card);
```

5.2.2 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, he *SyncMode* argument in continuous AO functions has to be set as *ASYNCH_OP.*

Initial default channel configuration:

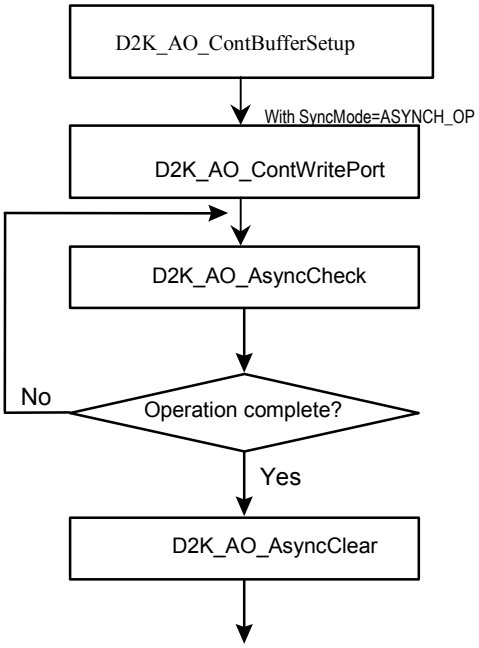
D/A Output Polarity	DAQ2K_DA_BiPolar
D/A Reference voltage source	DAQ2K_DA_Int_REF
D/A Reference voltage value	10.0

Initial default DA configuration:

D/A R/W source	DAQ2K_DA_WRSRC_Int (Internal timer pacer)
D/A Trigger mode	DAQ2K_DA_TRGMOD_POST (post trigger)
D/A Trigger source	DAQ2K_DA_TRGSRC_SOFT (software trigger)
Auto buffer reset	TRUE (except DAQ-2501 and DAQ-2502) FALSE (only for DAQ-2501 and DAQ-2502)

1. Non-double-buffered Asynchronous Continuous Analog output programming Scheme

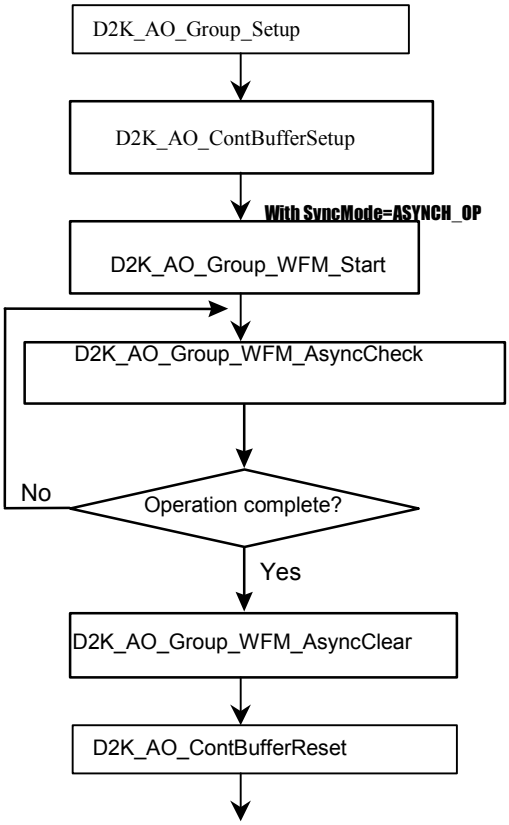
a. DAQ-2005, DAQ-2006, DAQ-2010, DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2214



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
            …
D2K_AO_ContBufferSetup (card, ao_buf, data_size, &DaId);
D2K_AO_ContWriteChannel(card, 0, DaId, data_size, iteration, samp_intrv, samp_intrv, ASYNCH_OP);;
do {
     D2K_AO_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);

D2K_AO_AsyncClear(card, &count, mode);
 …
D2K_Release_Card(card);
```

b. DAQ-2501, DAQ2502

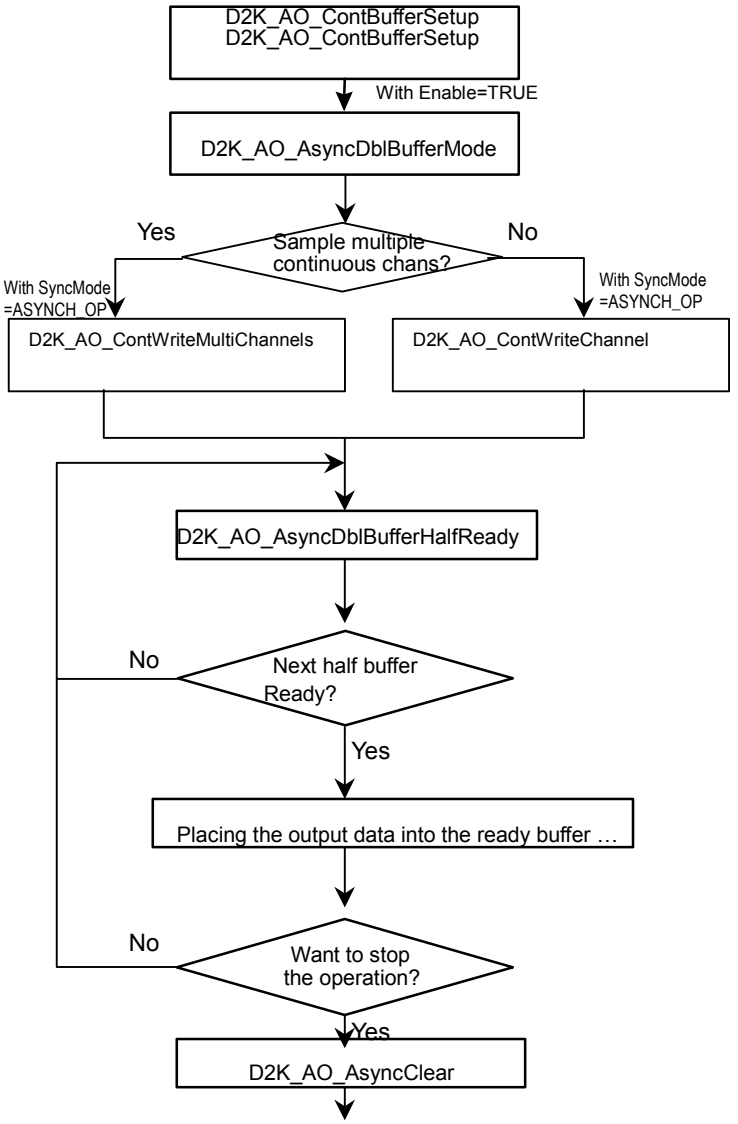


[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2502, card_number);
D2K_AO_Group_Setup (card, DA_Group_A, 1, &da_ch); //DA channel 0 in group A
D2K_AO_ContBufferSetup (card, ao_buf, data_size, &DaId);
D2K_AO_Group_WFM_Start (card, DA_Group_A, Id, DaId, data_size/2, 10, samp_intrv, 1);
do {
      D2K_AO_Group_WFM_AsyncCheck(card, DA_Group_A, &bStopped, &count);
} while (!bStopped);

D2K_AO_Group_WFM_AsyncClear(card, DA_Group_A, &count, 0);
D2K_AO_ContBufferReset (card);
D2K_Release_Card(card);
```

2. Double-buffered Asynchronous Continuous Analog output programming Scheme

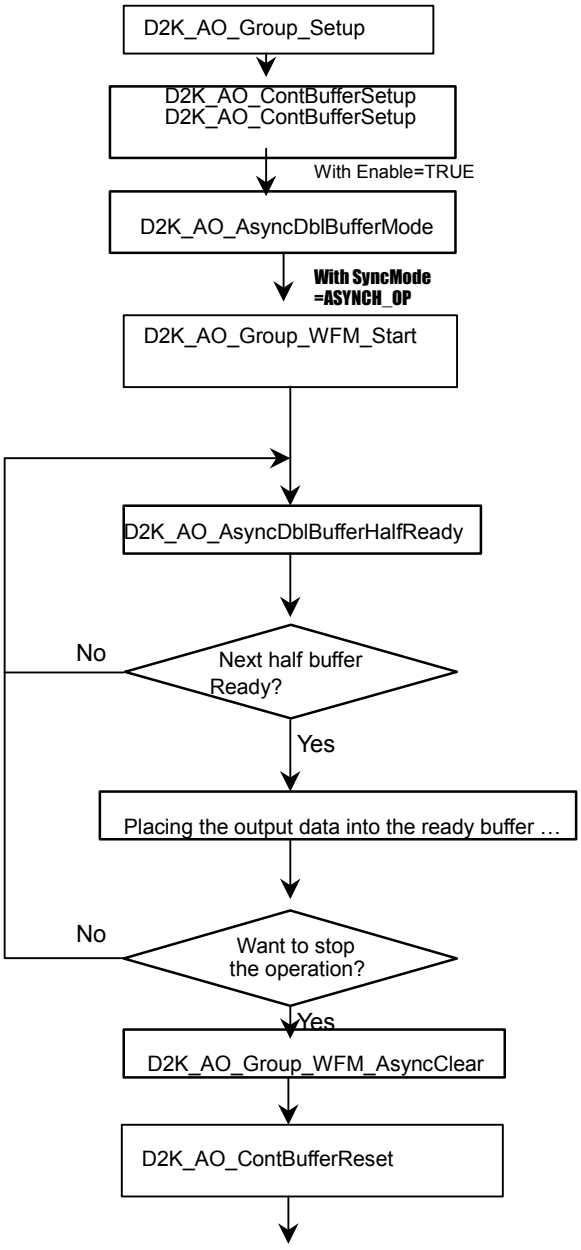
a. DAQ-2005, DAQ-2006, DAQ-2010, DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2214



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
D2K_AO_ContBufferSetup (card, ao_buf, data_size, &DaId);
D2K_AO_ContBufferSetup (card, ao_buf2, data_size, &DaId);
D2K_AO_AsyncDblBufferMode (card, 1);
D2K_AO_ContWriteChannel(card, 0, DaId, data_size, 0, samp_intrv, samp_intrv, ASYNCH_OP);;
do {
     do {
         D2K_AO_AsyncDblBufferHalfReady(card, &HalfReady);
     } while (!HalfReady);

    // Placing the output data into the ready buffer …
     …
} while (!clear_op);

D2K_AO_AsyncClear(card, &count, mode);
  …
D2K_Release_Card(card);
```

b. DAQ-2501, DAQ2502



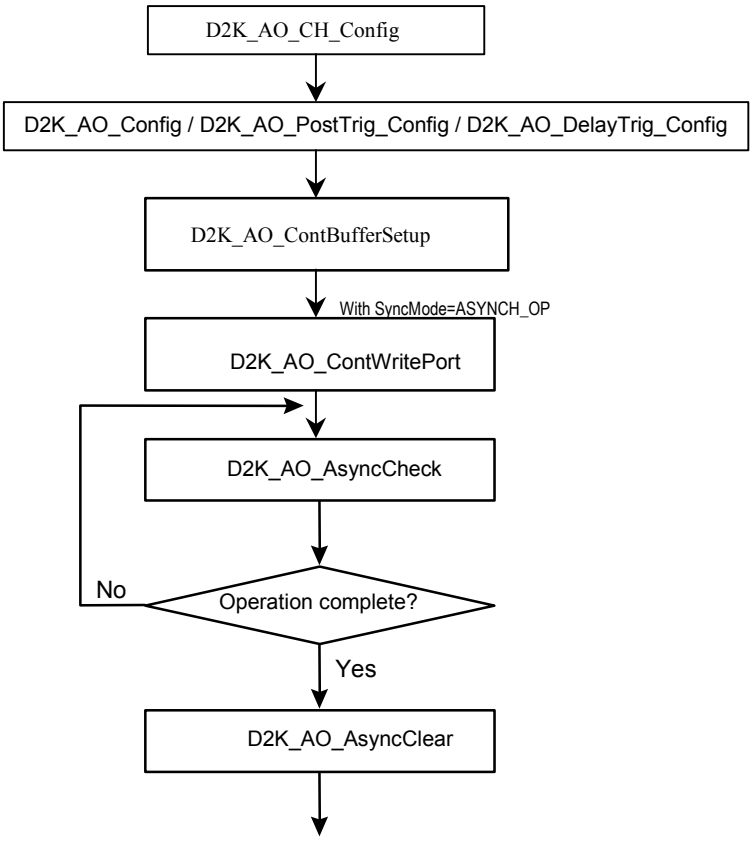
[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2501, card_number);
D2K_AO_Group_Setup (card, DA_Group_A, 1, &da_ch); //DA channel 0 in group A
D2K_AO_ContBufferSetup (card, ao_buf, data_size, &DaId);
D2K_AO_ContBufferSetup (card, ao_buf2, data_size, &DaId);
D2K_AO_AsyncDblBufferMode (card, 1);
D2K_AO_Group_WFM_Start (card, DA_Group_A, Id, DaId, data_size/2, 0, samp_intrv, 1);
do {
     do {
         D2K_AO_AsyncDblBufferHalfReady(card, &HalfReady);
     } while (!HalfReady);

    // Placing the output data into the ready buffer …
    …
} while (!clear_op);

D2K_AO_Group_WFM_AsyncClear(card, DA_Group_A, &count,0);
D2K_AO_ContBufferReset (card);  …
D2K_Release_Card(card);
```

5.2.3 Non-double-buffered Asynchronous Continuous Analog output programming Scheme

This section described the function flow typical of asynchronous analog output operation. While performing continuous AO operation, the AO configuration function has to be called at the beginning of your application. In addition, he *SyncMode* argument in continuous AO functions has to be set as *ASYNCH_OP*.

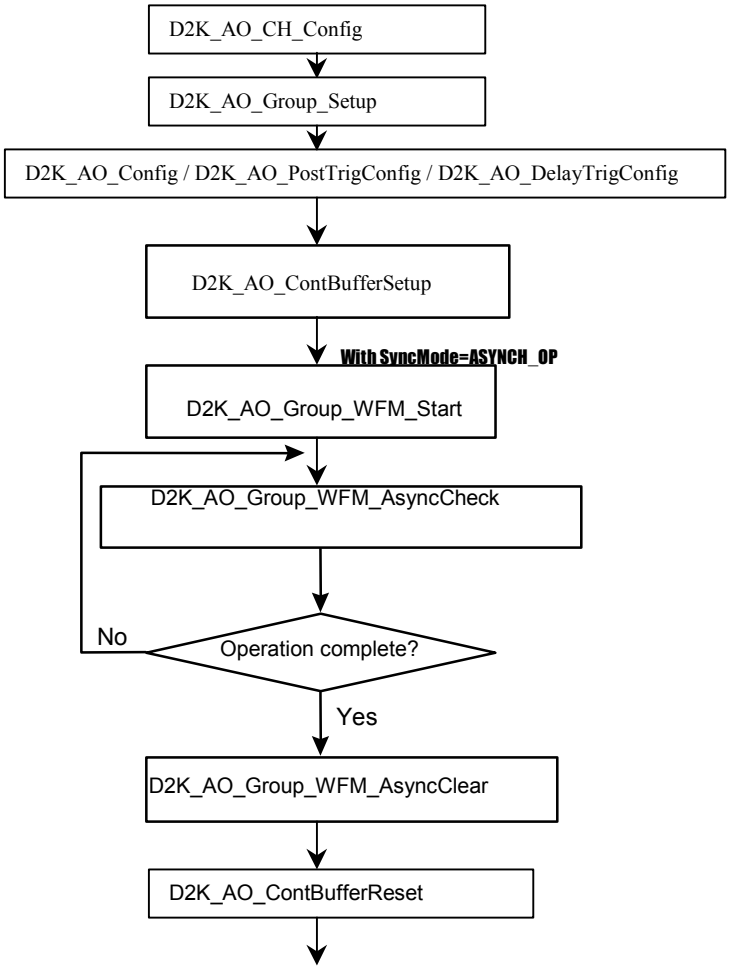
a. DAQ-2005, DAQ-2006, DAQ-2010, DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2214



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
        …
D2K_AO_CH_Config (card, 0, DAQ2K_DA_BiPolar, DAQ2K_DA_Int_REF, 10.0);
D2K_AO_Group_Setup (card, DA_Group_A, 1, &da_ch); //DA channel 0 in group A
D2K_AO_Config (card, 0, DAQ2K_DA_TRGMOD_POST|DAQ2K_DA_TRGSRC_ExtD, 1, 0, 0,1)
D2K_AO_ContBufferSetup (card, ao_buf, data_size, &DaId);
D2K_AO_ContWriteChannel(card, 0, DaId, data_size, iteration, samp_intrv, samp_intrv, ASYNCH_OP);;
do {
    D2K_AO_AsyncCheck(card, &bStopped, &count);
} while (!bStopped);

D2K_AO_AsyncClear(card, &count, mode);
 …
D2K_Release_Card(card);
```

b. DAQ-2501, DAQ2502



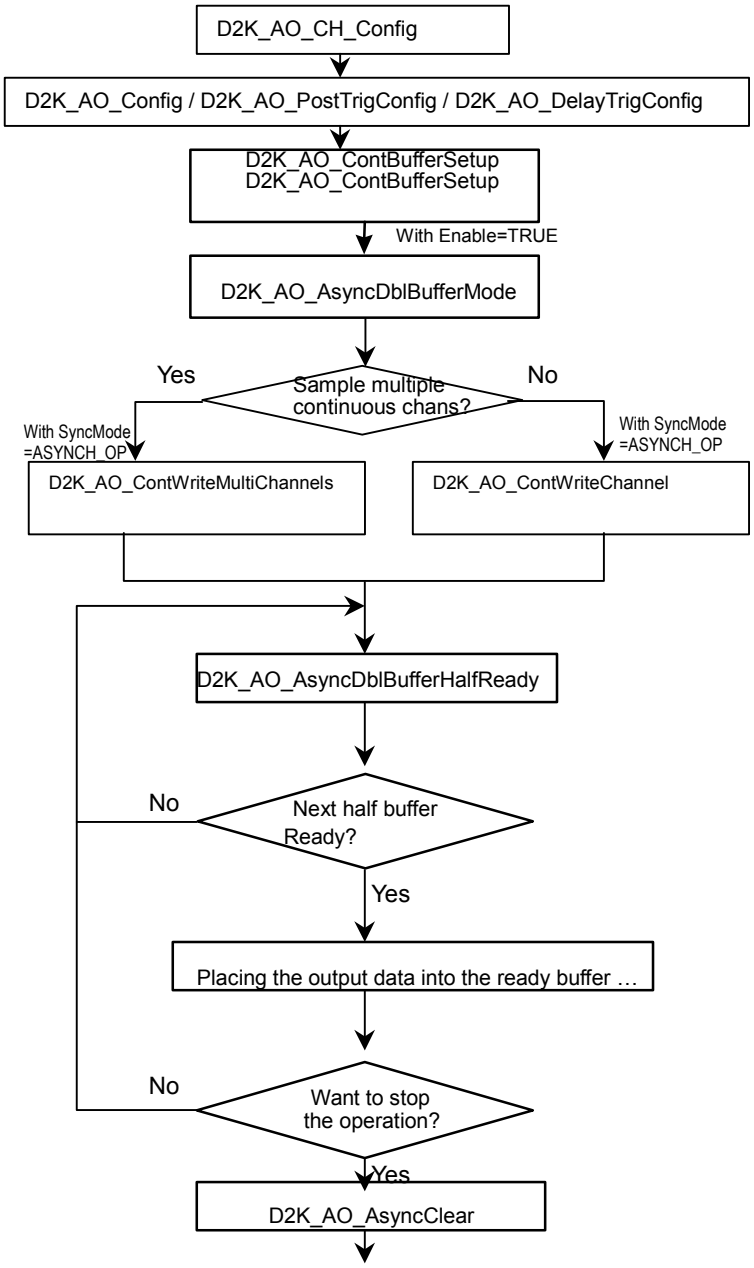
[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2501, card_number);
            …
D2K_AO_CH_Config (card, 0, DAQ2K_DA_BiPolar, DAQ2K_DA_Int_REF, 10.0);
D2K_AO_Group_Setup (card, DA_Group_A, 1, &da_ch); //DA channel 0 in group A
D2K_AO_Config (card, 0, DAQ2K_DA_TRGMOD_POST|DAQ2K_DA_TRGSRC_ExtD, 1, 0, 0,0);
D2K_AO_ContBufferSetup (card, ao_buf, data_size, &DaId);
D2K_AO_Group_WFM_Start (card, DA_Group_A, Id, DaId, data_size/2, 10, samp_intrv, 1);
do {
     D2K_AO_Group_WFM_AsyncCheck(card, DA_Group_A, &bStopped, &count);
 } while (!bStopped);
D2K_AO_Group_WFM_AsyncClear(card, DA_Group_A, &count, 0);
D2K_AO_ContBufferReset (card);
D2K_Release_Card(card);
```

5.2.4 Double-buffered Asynchronous Continuous Analog output programming Scheme

This section described the function flow typical of double-buffered asynchronous analog output operation. While performing continuous AO operation, the AO configuration function has to be called at the beginning of your application. The *SyncMode* argument in continuous AO functions has to be set as *ASYNCH_OP*. In addition, double-buffered AO operation is enabled by setting *Enable* argument of *D2K_AO_AsyncDblBufferMode* function to 1. To learn more about double buffer mode, please refer to section 5.2 *Double-Buffered AI/AO Operation* for the details.

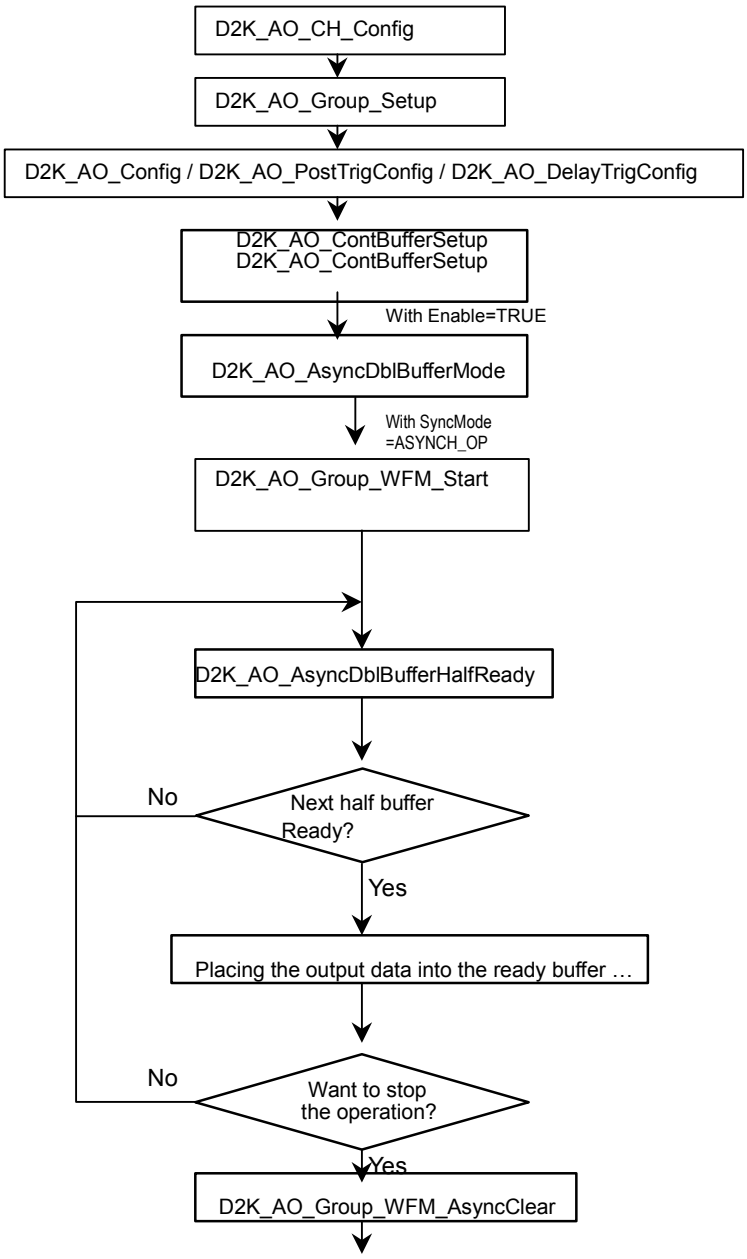
a. DAQ-2005, DAQ-2006, DAQ-2010, DAQ-2204, DAQ-2205, DAQ-2206, DAQ-2214



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
        …
D2K_AO_CH_Config (card, 0, DAQ2K_DA_BiPolar, DAQ2K_DA_Int_REF, 10.0);
D2K_AO_Config (card, 0, DAQ2K_DA_TRGMOD_POST|DAQ2K_DA_TRGSRC_ExtD, 1, 0, 0,1)
D2K_AO_ContBufferSetup (card, ao_buf, data_size, &DaId);
D2K_AO_ContBufferSetup (card, ao_buf2, data_size, &DaId);
D2K_AO_AsyncDblBufferMode (card, 1);
D2K_AO_ContWriteChannel(card, 0, DaId, data_size, 0, samp_intrv, samp_intrv, ASYNCH_OP);;
do {
    do {
        D2K_AO_AsyncDblBufferHalfReady(card, &HalfReady);
    } while (!HalfReady);

   // Placing the output data into the ready buffer …
    …
} while (!clear_op);

D2K_AO_AsyncClear(card, &count, mode);
 …
D2K_Release_Card(card);
```

b. DAQ-2501, DAQ2502



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2502, card_number);
            …
D2K_AO_CH_Config (card, 0, DAQ2K_DA_BiPolar, DAQ2K_DA_Int_REF, 10.0);
D2K_AO_Group_Setup (card, DA_Group_A, 1, &da_ch); //DA channel 0 in group A
D2K_AO_Config (card, 0, DAQ2K_DA_TRGMOD_POST|DAQ2K_DA_TRGSRC_ExtD, 1, 0, 0,0);
D2K_AO_ContBufferSetup (card, ao_buf, data_size, &DaId);
D2K_AO_ContBufferSetup (card, ao_buf2, data_size, &DaId);
D2K_AO_AsyncDblBufferMode (card, 1);
D2K_AO_Group_WFM_Start (card, DA_Group_A, Id, DaId, data_size/2, 0, samp_intrv, 1);
do {
    do {
        D2K_AO_AsyncDblBufferHalfReady(card, &HalfReady);
    } while (!HalfReady);

    // Placing the output data into the ready buffer …
    …
} while (!clear_op);
D2K_AO_Group_WFM_AsyncClear(card, DA_Group_A, &count,0);
```

```
D2K_AO_ContBufferReset (card);  …
D2K_Release_Card(card);
```

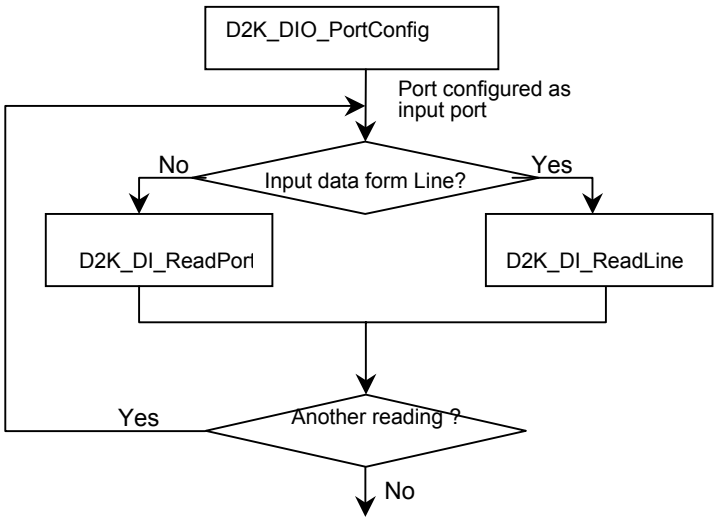
5.3 Digital Input Programming Hints

D2K-DASK provides one kind of digital input operation — non-buffered single-point 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 5.3.1.

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 need to include port configuration function at the beginning of your application.



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
//port configured
D2K_DIO_PortConfig(card ,Channel_P1A, INPUT_PORT);
D2K_DIO_PortConfig(card, Channel_P1B, INPUT_PORT);
D2K_DIO_PortConfig(card, Channel_P1CL, INPUT_PORT);
D2K_DIO_PortConfig(card, Channel_P1CH, INPUT_PORT);
//DI operation
D2K_DI_ReadPort(card, Channel_P1A, &inputA);
…
D2K_Release_Card(card);
```

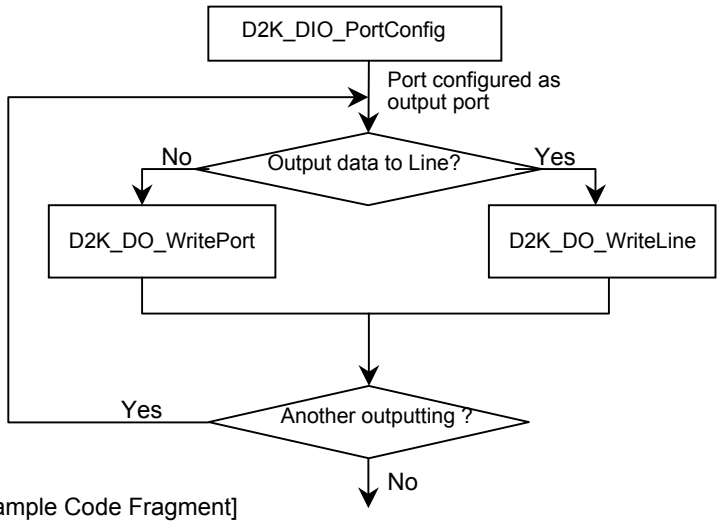
5.4 Digital Output Programming Hints

D2K-DASK provides one kind of digital output operation — non-buffered single-point digital output operation.

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 5.4.1.

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 need to include port configuration function at the beginning of your application.



[Example Code Fragment]

```
card = D2K_Register_Card(DAQ_2010, card_number);
//port configured
D2K_DIO_PortConfig(card ,Channel_P1A, OUTPUT_PORT);
D2K_DIO_PortConfig(card, Channel_P1B, OUTPUT_PORT);
D2K_DIO_PortConfig(card, Channel_P1CL, OUTPUT_PORT);
D2K_DIO_PortConfig(card, Channel_P1CH, OUTPUT_PORT);
//DO operation
D2K_DO_WritePort(card, Channel_P1A, outA_value);
…
D2K_Release_Card(card);
```

5.5 DAQ Event Message Programming Hints

DAQ Event Message functions are an efficient way to monitor your background data acquisition processes, without dedicating your foreground process for status checking. There are two kinds of events, which are *AI/AO operation completeness* notification event and *half buffer ready* notification event.

To receive notification from the D2K-DASK data acquisition process in case of special events, you can call D2K_AI_EventCallBack or D2K_AO_EventCallBack to specify an event in which you are interested.

Event notification is done through user-defined callbacks. When a user-specified DAQ event occurs, D2K-DASK calls the user-defined callback. 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 NT systems; however, the time delay between the 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 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 = D2K_Register_Card(DAQ_2010, card_number);
…
D2K_AI_CH_Config (card, channel, range )
D2K_AI_Config (card, 0,  DAQ2K_AI_TRGMOD_PRE|DAQ2K_AI_TRGSRC_ExtD, 0, 0, 0, 1);
// or
 // D2K_AI_MiddleTrig_Config (card, DAQ2K_AI_ADCONVSRC_Int, DAQ2K_AI_TRGSRC_ExtD| DAQ2K_AI_TrgPositive, 0, 0, 1);
D2K_AI_AsyncDblBufferMode (card, 1); // Double-buffered AI
D2K_AI_ContBufferSetup (card, ai_buf, data_size, &BufId);
D2K_A_ContBufferSetup (card, ai_buf2, data_size, &BufId);
// Enable half buffer ready event notification
D2K_AI_EventCallBack (card, 1, DBEvent, (U32) DB_cbfn );
//Enable AI completeness event notification
D2K_AI_EventCallBack (card, 1, DAQEnd, (U32) AI_cbfn );
D2K_AI_ContScanChannels (card, channel, BufId, data_size/(channel+1), ScanIntrv, SampIntrv, ASYNCH_OP); or
D2K_AI_ContReadChannel(card, channel, BufId, data_size, ScanIntrv, SampIntrv, ASYNCH_OP)
  …
D2K_Release_Card(card);

//Half buffer ready call back function
void DB_cbfn()
{
//half buffer is ready
….
}

//AI completeness call back function
void AI_cbfn()
{
//AI is completed
D2K_AI_AsyncClear(card, &startPos, &count);
….
}
```

.

6

Continuous Data Transfer in D2K-DASK

The continuous data transfer functions in D2K-DASK input or output blocks of data to or from a plug-in DAQ-2000 device. For input operations, D2K-DASK must transfer the incoming data to a buffer in the computer memory. For output operations, D2K-DASK must transfer outgoing data from a buffer in the computer memory to the DAQ-2000 device. This chapter describes the mechanism and techniques that D2K-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

D2K-DASK uses the DMA controller chip to perform a hardware transfer of the data.

6.2 Double-Buffered AI/AO Operation

D2K-DASK uses double-buffering techniques in its driver software for continuous input/output 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 process the data in the first half buffer according to application needs (Figure 6-1b). 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 process the second half of the circular buffer (Figure 7-1c). The process can be repeated endlessly to provide a continuous stream of data to your application (Figure 7-1d).

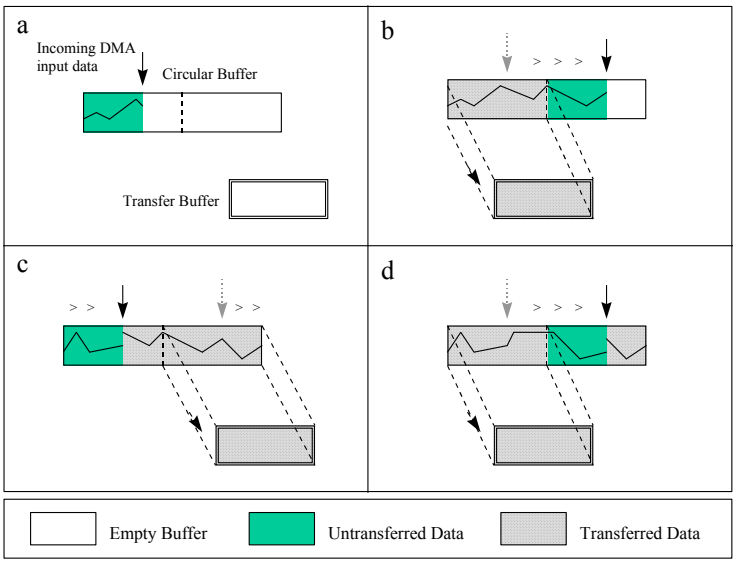


Figure 7-1

The D2K-DASK double buffer mode functions were designed according to the principle described above. If you use **D2K_AI_AsyncDblBufferMode/D2K_AO_AsyncDblBufferMode** to enable double buffer mode, the following continuous AI/AO function will perform double-buffered continuous AI/AO. You can call **D2K_AI_AsyncDblBufferHalfReady/ D2K_AO_AsyncDblBufferHalfReady** to check if data in the circular buffer is half full and ready for copying to the transfer buffer.

6.2.2 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 exits the undesired result of data overwritten for double-buffered data transfer. The device might overwrite data before D2K-DASK has copied it to the transfer buffer. Another data overwritten problem occurs when an input device overwrites data that D2K-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.

6.3 Pre-Trigger Mode/ Middle-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.

D2K-DASK also provides two buffering methods for pre/middle-trigger mode AI – double-buffering and single-buffering. However, the single buffer in pre/middle-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 DAQ-2010, 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 D2K-DASK uses asynchronous AI to perform pre/middle-trigger mode data acquisition, the *SyncMode* of continuous AI should be set as *ASYNCH_OP*.

7

Distribution of Applications

7.1 Files

To install an application using D2K-DASK on another computer, you also must install the necessary driver files and supporting libraries on the target machine. You can create an automatic installer to install your program and all of the files needed to run that program or you can manually install the program and program files. Whichever installation method you choose, you must install the following files:

- Required support DLLs:
  - D2K-DASK.dll
- Driver files

Windows 98

- The corresponding driver files in \DAQ2000\W98NT2K \redist\W98\drivers, e.g. daq2010.sys for DAQ-2010. These files should be copied to Windows\system32\drivers directory.
- The corresponding INF files in \DAQ2000\W98NT2K \redist\W98\Inf, e.g. Daq2010.inf for DAQ-2010. These files should be copied to Windows\inf directory.
- Device configuration utility in \DAQ2000\W98NT2K\redist\W98\Util.
- Device calibration utility in \DAQ2000\W98NT2K \redist\W98\Util.

Windows NT 4.0

- The corresponding driver files in \DAQ2000\W98NT2K \redist\Wnt\drivers, e.g. daq2010.sys for DAQ-2010. These files should be copied to Winnt\system32\drivers directory.
- Device configuration utility in \DAQ2000\W98NT2K/redist\Wnt\Util.
- Device calibration utility in \DAQ2000\W98NT2K \redist\Wnt\Util.

Windows 2000

- The corresponding driver file in \ DAQ2000\W98NT2K \redist\W2000\drivers, e.g. daq2010.sys for DAQ-2010. These files should be copied to Winnt\system32\drivers directory.
- The corresponding INF file in \ DAQ2000\W98NT2K \redist\W2000\Inf, e.g. daq2010.inf for DAQ-2010. These files should be copied to Winnt\inf directory.
- Device configuration utility in \ DAQ2000\W98NT2K \redist\W2000\Util.
- Device calibration utility in \DAQ2000\W98NT2K \redist\W2000\Util.

- Utility file (option)
  - Data Conversion utility DAQCvt.exe in \DAQ2000\ W98NT2K\redist\W98\Util, \ DAQ2000\W98NT2K\redist\Wnt\Util or \ DAQ2000\W98NT2K\redist\W2000\Util to convert the binary data file to the file format read easily.

7.2 Automatic Installers

Many programming environments include some form of setup or distribution kit tool. This tool automatically creates an installation program for your program so that you can easily install it on another computer. To function successfully, this tool must recognize which control files and supporting libraries are required by your program and include these in the installation program it creates.

Some of these tools, such as the Visual Basic 5 Setup Wizard, use dependency files to determine which libraries are required by an VB application.

Some setup tools might not automatically recognize which files are required by a program but 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. You also should verify that the resulting installation program does not copy older versions of a file over a newer version on the target computer.

If your programming environment does not provide a tool or wizard for building an installation program, you can use third-party tools such as InstallShield. Some programming environments provide simplified or trial versions of third-party installer creation tools on their installation CDs.

The installation program not only copies all the required files into the appropriated location, but executes *Device configuration utility* to configure the devices.

7.3 Manual Installation

If your programming environment does not include a setup or distribution kit tool, you can perform the installation task manually. To install your program on another computer, follow these steps:

1. Copy the program executable to the target computer.
2. Copy all required D2K-DASK files described in the section 7.1.1 to the appropriate directory on the target computer.
3. Use *DAQ-2000 Device Configuration utility* to configure the device.

Note: Do not replace any files on the target computer if the file on the target computer has a newer version than the file you are installing.