DAS-1800 Series
Function Call Driver

User's Guide

Contains Operating and Programming Information

KEITHLEY

Warranty

Hardware

Keithley Instruments, Inc. warrants that, for a period of one (1) year from the date of shipment (2 years for Model 199 and 3 years for Models 2000, 2001, 2002, and 2010), the Keithley Hardware product will be free from defects in materials or workmanship. This warranty will be honored provided the defect has not been caused by use of the Keithley Hardware not in accordance with the instructions for the product. This warranty shall be null and void upon: (1) any modification of Keithley Hardware that is made by other than Keithley and not approved in writing by Keithley or (2) operation of the Keithley Hardware outside of the environmental specifications therefore.

Upon receiving notification of a defect in the Keithley Hardware during the warranty period, Keithley will, at its option, either repair or replace such Keithley Hardware. During the first ninety days of the warranty period, Keithley will, at its option, supply the necessary on site labor to return the product to the condition prior to the notification of a defect. Failure to notify Keithley of a defect during the warranty shall relieve Keithley of its obligations and liabilities under this warranty.

Other Hardware

The portion of the product that is not manufactured by Keithley (Other Hardware) shall not be covered by this warranty, and Keithley shall have no duty of obligation to enforce any manufacturers' warranties on behalf of the customer. On those other manufacturers' products that Keithley purchases for resale, Keithley shall have no duty of obligation to enforce any manufacturers' warranties on behalf of the customer.

Software

Keithley warrants that for a period of one (1) year from date of shipment (2 years for Model 199 and 3 years for Models 2000, 2001, 2002, and 2010), the Keithley produced portion of the software or firmware (Keithley Software) will conform in all material respects with the published specifications provided such Keithley Software is used on the product for which it is intended and otherwise in accordance with the instructions therefore. Keithley does not warrant that operation of the Keithley Software will be uninterrupted or error-free and/or that the Keithley Software will be adequate for the customer's intended application and/or use. This warranty shall be null and void upon any modification of the Keithley Software that is made by other than Keithley and not approved in writing by Keithley.

If Keithley receives notification of a Keithley Software nonconformity that is covered by this warranty during the warranty period, Keithley will review the conditions described in such notice. Such notice must state the published specification(s) to which the Keithley Software fails to conform and the manner in which the Keithley Software fails to conform to such published specification(s) with sufficient specificity to permit Keithley to correct such nonconformity. If Keithley determines that the Keithley Software does not conform with the published specifications, Keithley will, at its option, provide either the programming services necessary to correct such nonconformity or develop a program change to bypass such nonconformity in the Keithley Software. Failure to notify Keithley of a nonconformity during the warranty shall relieve Keithley of its obligations and liabilities under this warranty.

Other Software

OEM software that is not produced by Keithley (Other Software) shall not be covered by this warranty, and Keithley shall have no duty or obligation to enforce any OEM's warranties on behalf of the customer.

Other Items

Keithley warrants the following items for 90 days from the date of shipment: probes, cables, rechargeable batteries, diskettes, and documentation.

Items not Covered under Warranty

This warranty does not apply to fuses, non-rechargeable batteries, damage from battery leakage, or problems arising from normal wear or failure to follow instructions.

Limitation of Warranty

This warranty does not apply to defects resulting from product modification made by Purchaser without Keithley's express written consent, or by misuse of any product or part.

Disclaimer of Warranties

EXCEPT FOR THE EXPRESS WARRANTIES ABOVE KEITHLEY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEITHLEY DISCLAIMS ALL WARRANTIES WITH RESPECT TO THE OTHER HARDWARE AND OTHER SOFTWARE.

Limitation of Liability

KEITHLEY INSTRUMENTS SHALL IN NO EVENT, REGARDLESS OF CAUSE, ASSUME RESPONSIBILITY FOR OR BE LIABLE FOR: (1) ECONOMICAL, INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL, PUNITIVE OR EXEMPLARY DAMAGES, WHETHER CLAIMED UNDER CONTRACT, TORT OR ANY OTHER LEGAL THEORY, (2) LOSS OF OR DAMAGE TO THE CUSTOMER'S DATA OR PROGRAMMING, OR (3) PENALTIES OR PENALTY CLAUSES OF ANY DESCRIPTION OR INDEMNIFICATION OF THE CUSTOMER OR OTHERS FOR COSTS, DAMAGES, OR EXPENSES RELATED TO THE GOODS OR SERVICES PROVIDED UNDER THIS WARRANTY.

DAS-1800 Series
Function Call Driver
User's Guide

Worldwide Addresses

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
(440) 248-0400
Fax: (440) 248-6168
http://www.keithley.com

CHINA
Keithley Instruments China
Yuan Chen Xin Building, Room 705
No. 12 Yumin Road, Dewei, Madian
Beijing, China 100029
8610-2022856
Fax: 8610-2022892

FRANCE
Keithley Instruments SARL
BP 60
3 allée des Garays
91122 Palaiseau Cédex
31-6-0115155
Fax: 31-6-0117726

GERMANY
Keithley Instruments GmbH
Landsberger Straße 65
82110 Germering
49-89-849307-0
Fax: 49-89-84930759

GREAT BRITAIN
Keithley Instruments, Ltd.
The Minster
58 Portman Road
Reading, Berkshire RG30 1EA
44-01734-575666
Fax: 44-01734-596469

ITALY
Keithley Instruments SRL
Viale S. Gimignano 38
20146 Milano
39-2-48303008
Fax: 39-2-48302274

NETHERLANDS
Keithley Instruments BV
Avelingen West 49
4202 MS Gorinchem
31-(0)183-635333
Fax: 31-(0)183-630821

SWITZERLAND
Keithley Instruments SA
Kriesbachstrasse 4
8600 Dübendorf
41-1-8219444
Fax: 41-1-8203081

TAIWAN
Keithley Instruments Taiwan
1, Ming-Yu First Street
Hsinchu. Taiwan, R.O.C.
886-35-778462
Fax: 886-35-778455

Manual Print History

The print history shown below lists the printing dates of all Revisions and Addenda created for this manual. The Revision Level letter increases alphabetically as the manual undergoes subsequent updates. Addenda, which are released between Revisions, contain important change information that the user should incorporate immediately into the manual. Addenda are numbered sequentially. When a new Revision is created, all Addenda associated with the previous Revision of the manual are incorporated into the new Revision of the manual. Each new Revision includes a revised copy of this print history page.

Revision B (Document Number 77160 Rev. B) .... April 1994
Revision C (Document Number 77160 Rev. C) .... August 1997

Safety Precautions

The following safety precautions should be observed before using this product and any associated instrumentation. Although some instruments and accessories would normally be used with non-hazardous voltages, there are situations where hazardous conditions may be present.

This product is intended for use by qualified personnel who recognize shock hazards and are familiar with the safety precautions required to avoid possible injury. Read the operating information carefully before using the product.

The types of product users are:

Responsible body is the individual or group responsible for the use and maintenance of equipment, and for ensuring that operators are adequately trained.

Operators use the product for its intended function. They must be trained in electrical safety procedures and proper use of the instrument. They must be protected from electric shock and contact with hazardous live circuits.

Maintenance personnel perform routine procedures on the product to keep it operating, for example, setting the line voltage or replacing consumable materials. Maintenance procedures are described in the manual. The procedures explicitly state if the operator may perform them. Otherwise, they should be performed only by service personnel.

Service personnel are trained to work on live circuits, and perform safe installations and repairs of products. Only properly trained service personnel may perform installation and service procedures.

Exercise extreme caution when a shock hazard is present. Lethal voltage may be present on cable connector jacks or test fixtures. The American National Standards Institute (ANSI) states that a shock hazard exists when voltage levels greater than 30V RMS, 42.4V peak, or 60VDC are present. **A good safety practice is to expect that hazardous voltage is present in any unknown circuit before measuring.**

Users of this product must be protected from electric shock at all times. The responsible body must ensure that users are prevented access and/or insulated from every connection point. In some cases, connections must be exposed to potential human contact. Product users in these circumstances must be trained to protect themselves from the risk of electric shock. If the circuit is capable of operating at or above 1000 volts, **no conductive part of the circuit may be exposed.**

As described in the International Electrotechnical Commission (IEC) Standard IEC 664, digital multimeter measuring circuits (e.g., Keithley Models 175A, 199, 2000, 2001, 2002, and 2010) measuring circuits are Installation Category II. All other instruments' signal terminals are Installation Category I and must not be connected to mains.

Do not connect switching cards directly to unlimited power circuits. They are intended to be used with impedance limited sources. NEVER connect switching cards directly to AC mains. When connecting sources to switching cards, install protective devices to limit fault current and voltage to the card.

Before operating an instrument, make sure the line cord is connected to a properly grounded power receptacle. Inspect the connecting cables, test leads, and jumpers for possible wear, cracks, or breaks before each use.

For maximum safety, do not touch the product, test cables, or any other instruments while power is applied to the circuit under test. ALWAYS remove power from the entire test system and discharge any capacitors before: connecting or disconnecting cables or jumpers, installing or removing switching cards, or making internal changes, such as installing or removing jumpers.

Do not touch any object that could provide a current path to the common side of the circuit under test or power line (earth) ground. Always make measurements with dry hands while standing on a dry, insulated surface capable of withstanding the voltage being measured.

Do not exceed the maximum signal levels of the instruments and accessories, as defined in the specifications and operating information, and as shown on the instrument or test fixture panels, or switching card.

When fuses are used in a product, replace with same type and rating for continued protection against fire hazard.

Chassis connections must only be used as shield connections for measuring circuits, NOT as safety earth ground connections.

If you are using a test fixture, keep the lid closed while power is applied to the device under test. Safe operation requires the use of a lid interlock.

If a ⏚ screw is present, connect it to safety earth ground using the wire recommended in the user documentation.

The ⚠ symbol on an instrument indicates that the user should refer to the operating instructions located in the manual.

The ⚠ symbol on an instrument shows that it can source or measure 1000 volts or more, including the combined effect of normal and common mode voltages. Use standard safety precautions to avoid personal contact with these voltages.

The **WARNING** heading in a manual explains dangers that might result in personal injury or death. Always read the associated information very carefully before performing the indicated procedure.

The **CAUTION** heading in a manual explains hazards that could damage the instrument. Such damage may invalidate the warranty.

Instrumentation and accessories shall not be connected to humans.

Before performing any maintenance, disconnect the line cord and all test cables.

To maintain protection from electric shock and fire, replacement components in mains circuits, including the power transformer, test leads, and input jacks, must be purchased from Keithley Instruments. Standard fuses, with applicable national safety approvals, may be used if the rating and type are the same. Other components that are not safety related may be purchased from other suppliers as long as they are equivalent to the original component. (Note that selected parts should be purchased only

through Keithley Instruments to maintain accuracy and functionality of the product.) If you are unsure about the applicability of a replacement component, call a Keithley Instruments office for information.

To clean the instrument, use a damp cloth or mild, water based cleaner. Clean the exterior of the instrument only. Do not apply cleaner directly to the instrument or allow liquids to enter or spill on the instrument.

Keithley Instruments, Inc.

28775 Aurora Road Cleveland, OH 44139

Telephone: (440) 248-0400 ● FAX: (440) 248-6168

Table of Contents

3   Programming with the Function Call Driver

4 Function Reference

A   Error/Status Codes

B   Data Formats

Index

List of Figures

List of Tables

Preface

The *DAS-1800 Series Function Call Driver User's Guide* describes how to write application programs for DAS-1800 Series boards using the DAS-1800 Series Function Call Driver. The DAS-1800 Series Function Call Driver supports the following DOS-based languages:

- Microsoft® QuickBasic™ (Versions 4.0 and 4.5)

- Microsoft Professional Basic (Version 7.0 and higher)

- Microsoft Visual Basic™ for DOS (Version 1.0)

- Microsoft C/C++ (Version 4.0 and higher)

- Borland® C/C++ (Version 1.0 and higher)

- Borland Turbo Pascal® for DOS (Version 6.0 and higher)

The DAS-1800 Series Function Call Driver also supports the following Windows™-based languages:

- Microsoft Visual Basic for Windows (Version 1.0 and higher)

- Microsoft QuickC® for Windows (Version 1.0)

- Microsoft Visual C++™ (Version 1.0)

- Borland Turbo Pascal for Windows (Version 1.0 and higher)

The manual is intended for application programmers using a DAS-1800 Series board in an IBM® PC AT® or compatible computer. It is assumed that users have read the user's guide for their board to familiarize themselves with the board's features, and that they have completed the appropriate hardware installation and configuration. It is also assumed that users are experienced in programming in their selected language and that they are familiar with data acquisition principles.

The *DAS-1800 Series Function Call Driver User's Guide* is organized as follows:

- Chapter 1 contains the information needed to install the DAS-1800 Series Function Call Driver and to get help.

- Chapter 2 contains the background information needed to use the functions included in the DAS-1800 Series Function Call Driver.

- Chapter 3 contains programming guidelines and language-specific information related to using the DAS-1800 Series Function Call Driver.

- Chapter 4 contains detailed descriptions of the DAS-1800 Series Function Call Driver functions, arranged in alphabetical order.

- Appendix A contains a list of the error codes returned by DAS-1800 Series Function Call Driver functions.

- Appendix B contains instructions for converting raw counts to voltage and for converting voltage to raw counts.

An index completes this manual.

Keep the following conventions in mind as you use this manual:

- References to DAS-1800 Series boards apply to all members of the DAS-1800 family. When a feature applies to a particular board, that board's name is used.

- References to BASIC apply to all DOS-based BASIC languages (Microsoft QuickBasic, Microsoft Professional Basic, and Microsoft Visual Basic for DOS). When a feature applies to a specific language, the complete language name is used. References to Visual Basic for Windows apply to Microsoft Visual Basic for Windows.

- Keyboard keys are enclosed in square brackets ([]).

1

Getting Started

The DAS-1800 Series Function Call Driver is a library of data acquisition and control functions (referred to as the Function Call Driver or FCD functions). It is part of the following two software packages:

- **DAS-1800 standard software package** - This is the software package that is shipped with DAS-1800 Series boards; it includes the following:

    - Libraries of FCD functions for Microsoft QuickBasic, Microsoft Professional Basic, and Microsoft Visual Basic for DOS.

    - Support files, containing such program elements as function prototypes and definitions of variable types, which are required by the FCD functions.

    - Utility programs, running under DOS, that allow you to configure, calibrate, and test the functions of DAS-1800 Series boards.

    - Language-specific example programs.

- **ASO-1800 software package** - This is the Advanced Software Option for DAS-1800 Series boards. You purchase the ASO-1800 software package separately from the board; it includes the following:

    - Libraries of FCD functions for Microsoft C/C++, Borland C/C++, and Borland Turbo Pascal.

    - Dynamic Link Libraries (DLLs) of FCD functions for Microsoft Visual Basic for Windows, Microsoft QuickC for Windows, Microsoft Visual C++, and Borland Turbo Pascal for Windows.

    - Support files, containing program elements, such as function prototypes and definitions of variable types, that are required by the FCD functions.

- Utility programs, running under DOS and Windows, that allow you to configure, calibrate, and test the functions of DAS-1800 Series boards.

- Language-specific example programs.

Before you use the Function Call Driver, make sure that you have installed the software, set up the board, and created a configuration file using the setup and installation procedures described in Chapter 3 of the user's guide for your DAS-1800 Series board.

If you need help installing or using the DAS-1800 Series Function Call Driver, call your local sales office or the Keithley Instruments, Inc. Applications Engineering Department at:

(440) 248-1520
Monday - Friday, 8:00 A.M. - 6:00 P.M., Eastern Time

An applications engineer will help you diagnose and resolve your problem over the telephone. Please make sure that you have the following information available before you call:

DAS-1800ST/HR Series Board Configuration	Model	_____
	Serial #	_____
	Revision code	_____
	Base address setting	_____
	Interrupt level setting	_____
	Number of channels	_____
	Input (S.E. or Diff.)	_____
	Mode (uni. or bip.)	_____
	DMA chan(s)	_____
	Number of SSH-8s	_____
	Number of EXPs.	_____
Computer	Manufacturer	_____
	CPU type	_____
	Clock speed (MHz)	_____
	KB of RAM	_____
	Video system	_____
	BIOS type	_____
Operating System	DOS version	_____
	Windows version	_____
	Windows mode	_____
Software package	Name	_____
	Serial #	_____
	Version	_____
	Invoice/Order #	_____
Compiler (if applicable)	Language	_____
	Manufacturer	_____
	Version	_____
Accessories	Type	_____
	Type	_____
	Type	_____
	Type	_____
	Type	_____
	Type	_____
	Type	_____
	Type	_____

Technical Support

Before returning any equipment for repair, call Keithley Instruments, Inc., for technical support at:

(440) 248-1520
Monday - Friday, 8:00 A.M. - 6:00 P.M., Eastern Time

An applications engineer will help you diagnose and resolve your problem over the telephone.

If a telephone resolution is not possible, the applications engineer will issue you a Return Material Authorization (RMA) number and ask you to return the equipment. Include the RMA number with any documentation regarding the equipment.

When returning equipment for repair, include the following information:

- Your name, address, and telephone number.

- The invoice or order number and date of equipment purchase.

- A description of the problem or its symptoms.

- The RMA number on the **outside** of the package.

Repackage the equipment using the original anti-static wrapping, if possible, and handle it with ground protection. Ship the equipment to:

ATTN: RMA #_____
Repair Department
Keithley Instruments, Inc.
31300 Bainbridge Road
Cleveland, OH 44139

Telephone (440) 248-1520
FAX (440) 248-6168

Note: If you are submitting your equipment for repair under warranty, you must include the invoice number and date of purchase.

To enable Keithley Instruments, Inc., to respond as quickly as possible, you must include the RMA number on the outside of the package.

2

Available Operations

This chapter contains the background information you need to use the FCD functions to perform operations on DAS-1800 Series boards. The supported operations are listed in Table 2-1.

Table 2-1. Supported Operations

Operation	Page Reference
System	page 2-1
Analog input	page 2-4
Analog output	page 2-26
Digital input and output (I/O)	page 2-31

System Operations

This section describes the miscellaneous operations and general maintenance operations that apply to DAS-1800 Series boards and to the DAS-1800 Series Function Call Driver. It includes information on initializing a driver, initializing a board, retrieving revision levels, and handling errors.

Initializing the Driver

Before you can use any of the functions included in the DAS-1800 Series Function Call Driver, you must initialize the driver using one of the following driver initialization functions:

- **Board-specific driver initialization function** - If you want to initialize the DAS-1800 Series Function Call Driver only, use the board-specific driver initialization function **DAS1800_DevOpen**. You specify a configuration file; **DAS1800_DevOpen** initializes the driver according to the configuration file you specify.

- **Generic driver initialization function** - If you want to initialize several different Keithley DAS Function Call Drivers from the same application program, use the generic driver initialization function **K_OpenDriver**. You specify the Keithley DAS board you are using, the configuration file that defines this particular use of the driver, and the driver handle (a name that uniquely identifies the particular use of the driver). You can specify a maximum of 30 driver handles for all the Keithley DAS boards accessed from your application program.

  If a particular use of a driver is no longer required and you want to free some memory or if you have used all 30 driver handles, you can use the **K_CloseDriver** function to free a driver handle and close the associated use of the driver.

  If the driver handle you free is the last driver handle specified for a Function Call Driver, the driver is shut down. (For Windows-based languages only, the DLLs associated with the Function Call Driver are shut down and unloaded from memory.)

Initializing a Board

The DAS-1800 Series Function Call Driver supports up to three boards. You must use a board initialization function to specify the board(s) you want to use and the name you want to use to identify each board; this name is called the board handle. Board handles allow you to communicate with more than one board. You use the board handle you specify in the board initialization function in all subsequent function calls related to the board.

The DAS-1800 Series Function Call Driver provides the following board initialization functions:

- **Board-specific board initialization function** - If you want to initialize a DAS-1800 Series board only, use the board-specific board initialization function **DAS1800_GetDevHandle**.

- **Generic board initialization function** - If you want to initialize several different Keithley DAS boards from the same application program, use the generic board initialization function **K_GetDevHandle**. You can specify a maximum of 30 board handles for all the Keithley DAS boards accessed from your application program.

  If a board is no longer being used and you want to free some memory or if you have used all 30 board handles, you can use the **K_FreeDevHandle** function to free a board handle.

To reinitialize a board during an operation, use the **K_DASDevInit** function, which performs the following tasks:

- Abort all operations currently in progress that are associated with the board identified by the board handle.

- Verify that the board identified by the board handle is the board specified in the configuration file.

Retrieving Revision Levels

If you are using functions from different Keithley DAS Function Call Drivers in the same application program or if you are having problems with your application program, you may want to verify which versions of the Function Call Driver, Keithley DAS Driver Specification, and Keithley DAS Shell are installed on your board. The **K_GetVer** function allows you to get both the revision number of the DAS-1800 Series Function Call Driver and the revision number of the Keithley DAS Driver Specification to which the driver conforms. The **K_GetShellVer** function allows you to get the revision number of the Keithley DAS Shell (the Keithley DAS Shell is a group of functions that are shared by all DAS boards).

Handling Errors

Each FCD function returns a code indicating the status of the function. To ensure that your application program runs successfully, it is recommended that you check the returned code after the execution of each function. If the status code equals 0, the function executed successfully and your program can proceed. If the status code does not equal 0, an error occurred; ensure that your application program takes the appropriate action. Refer to Appendix A for a complete list of error codes.

For C-language application programs only, the DAS-1800 Series Function Call Driver provides the **K_GetErrMsg** function, which gets the address of the string corresponding to an error code.

Analog Input Operations

This section describes the following:

- Analog input operation modes available.

- How to allocate and manage memory for analog input operations.

- How to specify the following for an analog input operation: channels and gains, a conversion mode, a clock source, a buffering mode, a trigger source, and a hardware gate.

Operation Modes

The operation mode determines which attributes you can specify for an analog input operation and how data is transferred from the board to the computer. You can perform analog input operations in one of the following modes:

- **Single mode** - In single mode, the board acquires a single sample from an analog input channel. The driver initiates conversions; you cannot perform any other operation until the single-mode operation is complete.

  Use the **K_ADRead** function to start an analog input operation in single mode. You specify the board you want to use, the analog input channel, the gain at which you want to read the signal, and the variable in which to store the converted data.

- **Interrupt mode** - In interrupt mode, the board acquires a single sample or multiple samples from one or more analog input channels. A hardware clock initiates conversions. Once the analog input operation begins, control returns to your application program. The hardware temporarily stores the acquired data in the onboard FIFO (first-in, first-out data buffer) and then transfers the data to a user-defined buffer in the computer using an interrupt service routine.

  Use the **K_IntStart** function to start an analog input operation in interrupt mode. You specify the board, analog input channel(s), gain(s), clock source, buffering mode, buffer address(es), trigger source, and gate use.

  You can specify either single-cycle or continuous buffering mode for interrupt-mode operations. Refer to page 2-18 for more information on buffering modes. Use the **K_IntStop** function to stop a continuous-mode interrupt operation. Use the **K_IntStatus** function to determine the current status of an interrupt operation.

- **DMA mode** - In DMA mode, the board acquires a single sample or multiple samples from one or more analog input channels. A hardware clock initiates conversions. Once the analog input operation begins, control returns to your application program. The hardware temporarily stores the acquired data in the onboard FIFO (first-in,

first-out data buffer) and then transfers the data to a user-defined DMA buffer in the computer.

Note: You can perform an analog input operation in single-DMA mode or dual-DMA mode, depending on whether you specified one or two DMA channels in your configuration file. Refer to your DAS-1800 Series board user's guide for more information.

Use the **K_DMAStart** function to start an analog input operation in DMA mode. You specify the board, analog input channel(s), gain(s), clock source, buffering mode, buffer address(es), trigger source, and gate use.

You can specify either single-cycle or continuous buffering mode for DMA-mode operations. Refer to page 2-18 for more information on buffering modes. Use the **K_DMAStop** function to stop a continuous-mode DMA operation. Use the **K_DMAStatus** function to determine the current status of a DMA operation.

The converted data are stored as raw counts. For information on converting raw counts to voltage, refer to Appendix B.

Memory Allocation and Management

Interrupt-mode and DMA-mode analog input operations require memory buffers in which to store the acquired data. You can reserve a single memory buffer, or you can reserve multiple buffers (up to a maximum of 150) to increase the number of samples you can acquire. The maximum number of samples each memory buffer can store (32K or 64K) depends on the language you are using. See "Language-Specific Programming Information" on page 3-22 for more information.

You can reserve the required memory buffer(s) in one of the following ways:

- **Within your application program's memory area** - The simplest way to reserve memory buffers is to dimension arrays within your application program. The advantage of this method is that the arrays are directly accessible to your application program. The limitations of this method are as follows:

  - Certain programming languages limit the size of local arrays.

  - Local arrays may not be suitable for DMA-mode operations.

  - Local arrays occupy permanent memory areas; these memory areas cannot be freed to make them available to other programs or processes.

  Since the DAS-1800 Series Function Call Driver stores data in 16-bit integers, you must dimension all local arrays as integers.

- **Outside of your application program's memory area** - This is the recommended way to reserve memory buffers. The advantages of this method are as follows:

  - The number of buffers and the size of each buffer are limited by the amount of free physical memory available in your computer at run-time.

  - The dynamically allocated memory buffers can be freed to make them available to other programs or processes.

  The limitation of this method is that, for BASIC and Visual Basic languages, the data in a dynamically allocated memory buffer is not directly accessible by your program. (The DAS-1800 Series Function Call Driver provides a function, **K_MoveBufToArray**, to make this data accessible; refer to page 4-169 for more information.)

Use the **K_IntAlloc** function to allocate memory dynamically for interrupt-mode operations and the **K_DMAAlloc** function to allocate memory dynamically for DMA-mode operations. You specify the operation requiring the buffer, the number of samples to store in the buffer, the variable to store the starting address of the buffer, and the name you want to use to identify the buffer (this name is called the memory handle). When the buffer is no longer required, you can free the buffer for another use by specifying this memory handle in the **K_IntFree** function (for interrupt-mode operations) or the **K_DMAFree** function (for DMA-mode operations).

Notes: For DOS-based languages, the area used for dynamically allocated memory buffers is referred to as the far heap; for Windows-based languages, this area is referred to as the global heap. These heaps are areas of memory left unoccupied as your application program and other programs run.

For DOS-based languages, the **K_IntAlloc** and **K_DMAAlloc** functions use the DOS Int 21H function 48H to dynamically allocate far heap memory. For Windows-based languages, the **K_IntAlloc** and **K_DMAAlloc** functions call the **GlobalAlloc** API function to allocate the desired buffer size from the global heap.

For Windows-based languages, dynamically allocated memory is guaranteed to be fixed and locked in memory.

To eliminate page wrap conditions and to guarantee that dynamically allocated memory is suitable for use by the computer's 8237 DMA controller, **K_DMAAlloc** may allocate an area twice as large as actually needed. Once the data in this buffer is processed and/or saved elsewhere, use **K_DMAFree** to free the memory for other uses.

After you allocate your buffer(s), you must assign the starting address of the buffer(s) and the number of samples to store in the buffer(s). Each supported programming language requires a particular procedure for allocating memory buffers and assigning starting addresses. Refer to page 3-23 for information when programming in C. Refer to page 3-31 for information when programming in Pascal. Refer to page 3-40 for information when programming in Visual Basic for Windows. Refer to page 3-46 for information when programming in BASIC.

If you are using multiple buffers, use the **K_BufListAdd** function to add each buffer to the list of multiple buffers associated with each operation and to assign the starting address of each buffer. Use the **K_BufListReset** function to clear the list of multiple buffers.

Note: If you are using multiple buffers, it is recommended that you use the Keithley Memory Manager before you begin programming to ensure that you can allocate large enough buffers. Refer to your DAS-1800 Series board user's guide for more information about the Keithley Memory Manager.

Gains

Each channel on a DAS-1800 Series board can measure analog input signals in one of four, software-selectable unipolar or bipolar analog input ranges. The input range type (unipolar or bipolar) is initially set according to your configuration file; use **K_SetADMode** to reset the input range type. Refer to your DAS-1800 Series board user's guide for more information.

Table 2-2 lists the analog input ranges supported by DAS-1800 Series boards and the gain and gain code associated with each range. (The gain code is used by the FCD functions to represent the gain.)

Table 2-2. Analog Input Ranges

| Boards | Analog Input Range | | Gain | Gain Code |
	Bipolar	Unipolar		
DAS-1801HC DAS-1801ST	±5 V	0 to 5 V	1	0
	±1 V	0 to 1 V	5	1
	±100 mV	0 to 100 mV	50	2
	±20 mV	0 to 20 mV	250	3
DAS-1802HC DAS-1802ST DAS-1802HR	±10 V	0 to 10 V	1	0
	±5 V	0 to 5 V	2	1
	±2.5 V	0 to 2.5 V	4	2
	±1.25 V	0 to 1.25 V	8	3
DAS-1801ST with EXP-1800 attached	±0.1 V	0 to 0.1 V	50	4
	±20 mV	0 to 20 mV	250	5
	±2 mV	0 to 2 mV	2500	6
	±0.4 mV	0 to 0.4 mV	12.5k	7
DAS-1802ST with EXP-1800 attached; DAS-1802HR with EXP-1800 attached	±0.2 V	0 to 0.2 V	50	4
	±0.1 V	0 to 0.1 V	100	5
	±5 mV	0 to 5 mV	200	6
	±2.5 mV	0 to 2.5 mV	400	7

Channels

DAS-1800HC Series boards are configured with either 64 single-ended or 32 differential analog input channels, depending on the input configuration specified in your configuration file. DAS-1800ST/HR Series boards are configured with either 16 onboard single-ended or 8 onboard differential analog input channels. On DAS-1800ST/HR Series boards, you can increase the number of channels to 256 single-ended or 128 differential channels using the EXP-1800 expansion board, described in the next section.

The input channel configuration is initially set according to the configuration file; use **K_SetADConfig** to reset the input channel configuration. Use **K_SetADCommonMode** to set the common-mode ground reference for DAS-1800ST/HR Series boards in single-ended input channel configuration.

You can perform an analog input operation on a single channel or on a group of multiple channels. The following subsections describe how to specify the channel(s) you are using.

Specifying Channels When Using EXP-1800 Expansion Boards (DAS-1800ST/HR Series Only)

To increase the number of analog input channels, you can attach up to 16 EXP-1800 expansion boards to the DAS-1800 Series board. Each EXP-1800 board has 16 analog input channels. If you are using *N* EXP-1800 boards, you must attach them to DAS-1800 channels 0 to *N-1*. Refer to the *DAS-1800ST/HR Series User's Guide* for information on connecting EXP-1800 boards to DAS-1800ST/HR Series boards.

The analog input channel connections on a DAS-1800 Series board or EXP-1800 board are labelled with white-on-green numbers from 0 to 15. These numbers are the *physical channel numbers*. If a system includes a DAS-1800 Series board and one or more EXP-1800 boards, then that system contains duplicate physical channel numbers. To uniquely identify a physical channel, the Function Call Driver uses a scheme of *logical channel numbers*. The *channel#* argument for any FCD function must be specified as a logical channel number.

The logical channel number corresponding to a particular physical channel number is given by one of the following equations:

If the physical channel is on a DAS-1800 Series board:

$$LogicalChan\# = PhysicalChan\# + (15 \times NumEXPs)$$

If the physical channel is on an EXP-1800 board:

$$LogicalChan\# = PhysicalChan\# + (16 \times EXP\#)$$

where

NumEXPs is an integer from 0 to 15 that identifies the number of EXP-1800 boards connected to the DAS-1800 Series board, and

EXP# is an integer from 0 to 15 that indicates on which EXP-1800 board the physical channel is located (0 indicates the first EXP-1800 board).

For example, consider the system illustrated in Figure 2-1, in which three EXP-1800 boards are connected to a DAS-1801ST.
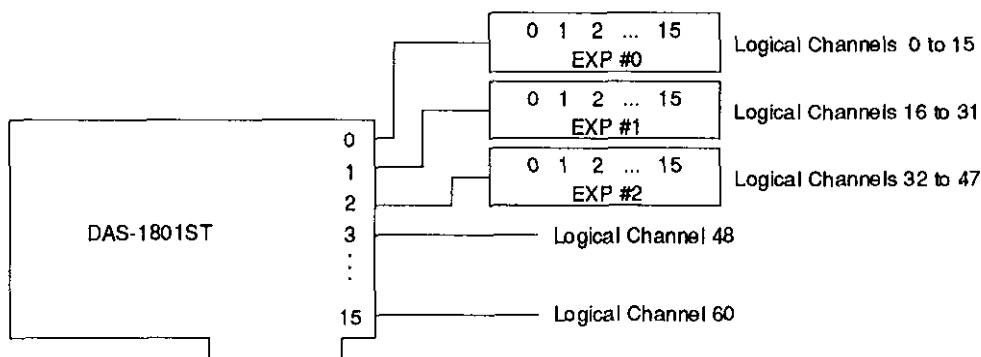
Figure 2-1. Example of Logical Channel Assignments

The logical channel that identifies channel 3 on the DAS-1801 board is given by:

$$LogicalChan\# = 3 + (15 \times 3) = 3 + 45 = 48$$

The logical channel that identifies channel 15 on the third EXP-1800 board is given by:

$$LogicalChan\# = 15 + (16 \times 2) = 15 + 32 = 47$$

Acquiring Samples from a Single Channel

You can acquire a single sample or multiple samples from a single analog input channel.

For single-mode analog input operations, you can acquire a single sample from a single analog input channel. Use the **K_ADRead** function to specify the channel and the gain code.

For interrupt-mode and DMA-mode analog input operations, you can acquire a single sample or multiple samples from a single analog input channel. Use the **K_SetChn** function to specify the channel and the **K_SetG** function to specify the gain code.

Acquiring Samples from a Group of Consecutive Channels

For interrupt-mode and DMA-mode analog input operations, you can acquire samples from a group of consecutive channels. Use the **K_SetStartStopChn** function to specify the first and last channels in the group. The channels are sampled in order from first to last; the channels are then sampled again until the required number of samples are read.

For example, assume that the start channel is 14, the stop channel is 17, and you want to acquire five samples. Your program reads data first from channel 14, then from channels 15, 16, and 17, and finally from channel 14 again.

You can specify a start channel that is higher than the stop channel. For example, assume that you are using a differential input configuration, the start channel is 31, the stop channel is 2, and you want to acquire five samples. Your program reads data first from channel 31, then from channels 0, 1, and 2, and finally from channel 31 again.

Use the **K_SetG** function to specify the gain code for all channels in the group. (All channels must use the same gain code.) Use the **K_SetStartStopG** function to specify the gain code, the start channel, and the stop channel in a single function call.

Refer to Table 2-2 on page 2-10 for a list of the analog input ranges supported by DAS-1800 Series boards and the gain code associated with each range.

Acquiring Samples Using a Channel-Gain Queue

For interrupt-mode and DMA-mode analog input operations, you can acquire samples from channels in a hardware channel-gain queue. In the channel-gain queue, you specify the channels you want to sample, the order in which you want to sample them, and a gain code for each channel.

You can set up the channels in a channel-gain queue either in consecutive order or in nonconsecutive order. You can also specify the same channel more than once (up to a total of 64 entries in the queue for a DAS-1800HC Series board, and up to 256 entries for a DAS-1800ST/HR Series board).

The channels are sampled in order from the first channel in the queue to the last channel in the queue; the channels in the queue are then sampled again until the board reads the specified number of samples.

Refer to Table 2-2 on page 2-10 for a list of the analog input ranges supported by DAS-1800 Series boards and the gain code associated with each range.

The way that you specify the channels and gains in a channel-gain queue depends on the language you are using. Refer to page 3-27 for information when programming in C or C++. Refer to page 3-37 for information when programming in Pascal. Refer to page 3-44 for information when programming in Visual Basic for Windows. Refer to page 3-50 for information when programming in BASIC.

After you create the channel-gain queue in your program, use the **K_SetChnGAry** function to transfer the contents of the channel-gain queue to the driver/board.

Conversion Modes

The conversion mode determines how the board regulates the timing of conversions when you are acquiring multiple samples from a single channel or from a group of multiple channels (known as a scan). For interrupt-mode and DMA-mode analog input operations, you can specify one of the following conversion modes:

- **Paced mode** - Use paced mode if you want to accurately control the period between conversions of individual channels in a scan. Paced mode is the default conversion mode.

- **Burst mode** - Use burst mode if you want to accurately control both the period between conversions of individual channels in a scan and the period between conversions of the entire scan. Use the **K_SetADFreeRun** function to specify burst mode.

  Use burst mode with SSH if you want to simultaneously sample all channels in a scan using the SSH-8 accessory board. Use the **K_SetSSH** function to specify burst mode with SSH.

Note: If you use an SSH-8 accessory board, you must use burst mode with SSH. One extra tick of the burst mode conversion clock is required to allow the SSH-8 board to sample and hold the values. Refer to the SSH-8 board documentation for more information.

Refer to your DAS-1800 Series board user's guide for more information about conversion modes.

Clock Sources

DAS-1800 Series boards provide two clock sources: a pacer clock and a burst mode conversion clock. Each clock has a dedicated use. When performing interrupt-mode and DMA-mode analog input operations in paced mode, you use only the pacer clock; when performing interrupt-mode and DMA-mode analog input operations in burst mode and burst mode with SSH, you use both the pacer clock and the burst mode conversion clock. These clock sources are described in the following subsections.

Pacer Clock

In paced mode, the pacer clock determines the period between the conversion of one channel and the conversion of the next channel. In burst mode and burst mode with SSH, the pacer clock determines the period between the conversions of one scan and the conversions of the next scan. Use the **K_SetClk** function to specify an internal or an external pacer clock. The internal pacer clock is the default pacer clock.

The internal and external pacer clocks are described as follows:

- **Internal pacer clock** - The internal pacer clock uses two cascaded counters of the onboard counter/timer circuitry. The counters are normally in an idle state. When you start the analog input operation (using **K_IntStart** or **K_DMAStart**), a conversion is initiated. Note that a slight time delay occurs between the time the operation is started and the time conversions begin.

  After the first conversion is initiated, the counters are loaded with a count value and begin counting down. When the counters count down to 0, another conversion is initiated and the process repeats.

  Because the counters use a 5 MHz time base, each count represents 0.2 µs. Use the **K_SetClkRate** function to specify the number of counts (clock ticks) between conversions. For example, if you specify a count of 30, the period between conversions is 6 µs (166.67 ksamples/s); if you specify a count of 87654, the period between conversions is 17.53 ms (57 samples/s).

  You can specify a count between 15 and 4,294,967,295. The period between conversions ranges from 3 µs to 14.3 minutes.

  When using an internal pacer clock, use the following formula to determine the number of counts to specify:

  $$\text{counts} = \frac{5 \text{ MHz time base}}{\text{conversion rate}}$$

For example, if you want a conversion rate of 10 ksamples/s, specify a count of 500, as shown in the following equation:

$$\frac{5,000,000}{10,000} = 500$$

- **External pacer clock** - You connect an external pacer clock to the DI0/XPCLK pin (pin B39) on the main I/O connector of the DAS-1800HC Series board or to the XPCLK pin (pin 44) on the main I/O connector of DAS-1800ST/HR Series boards. When you start an analog input operation (using **K_IntStart** or **K_DMAStart**), conversions are armed. At the next active edge of the external pacer clock (and at every subsequent active edge of the external pacer clock), a conversion is initiated. Use the **K_SetExtClkEdge** function to specify the active edge (rising or falling) of the external pacer clock. A falling edge is the default active edge for the external pacer clock.

Note: The rate at which the computer can reliably read data from the board depends on a number of factors, including your computer, the operating system/environment, the gains of the channels, and other software issues. If you are using an external pacer clock, make sure that the clock initiates conversions at a rate that the analog-to-digital converter can handle.

Refer to your DAS-1800 Series board user's guide for more information about the pacer clock.

Burst Mode Conversion Clock

In burst mode and burst mode with SSH, the burst mode conversion clock determines the period between the conversion of one channel in a scan and the conversion of the next channel in the scan.

Because the burst mode conversion clock uses a 1 MHz time base, each clock tick represents 1 µs. Use the **K_SetBurstTicks** function to specify the number of clock ticks between conversions. For example, if you specify 30 clock ticks, the period between conversions is 30 µs (33.33 ksamples/s).

You can specify between 3 and 255 clock ticks. The period between conversions ranges from 3 μs to 0.255 ms.

When using the burst mode conversion clock, use the following formula to determine the number of clock ticks to specify:

$$\text{clock ticks} = \frac{1 \text{ MHz time base}}{\text{burst mode conversion rate}}$$

For example, if you want a burst mode conversion rate of 10 ksamples/s, specify 100 clock ticks, as shown in the following equation:

$$\frac{1,000,000}{10,000} = 100$$

Refer to your DAS-1800 Series board user's guide for more information about the burst mode conversion clock.

Buffering Modes

The buffering mode determines how the driver stores the converted data in the buffer. For interrupt-mode and DMA-mode analog input operations, you can specify one of the following buffering modes:

- **Single-cycle mode** - In single-cycle mode, after the board converts the specified number of samples and stores them in the buffer, the operation stops automatically. Single-cycle mode is the default buffering mode.

- **Continuous mode** - In continuous mode, the board continuously converts samples and stores them in the buffer until it receives a stop function; any values already stored in the buffer are overwritten. Use the **K_SetContRun** function to specify continuous buffering mode.

Triggers

A trigger is an event that starts or stops an interrupt-mode or DMA-mode analog input operation. An operation can use either one or two triggers. Every operation must have a *start trigger* that marks the beginning of the operation. You can use an optional second trigger, the *about trigger*, to define when the operation stops. If you specify an about trigger, the operation stops when a specified number of samples has been acquired after the occurrence of the about-trigger event.

A post-trigger acquisition refers to an operation that only uses a start trigger. The about trigger provides the capability to define operations that acquire data before a trigger event (pre-trigger acquisition) and operations that acquire data about (before and after) a trigger event (about-trigger acquisition).

The following subsections describe the supported trigger sources and post-, pre-, and about-trigger acquisitions.

Trigger Sources

The Function Call Driver supports three sources of triggers: internal, analog, and digital. For interrupt-mode and DMA-mode analog input operations, use **K_SetTrig** to specify the trigger source. The trigger events for each trigger source are described below. Note that the trigger event is not significant until the operation the trigger governs has been enabled (using **K_DMAStart** or **K_IntStart**).

Internal Trigger

An internal trigger is a software trigger. It does not impose any external conditions that must be satisfied before the operation executes. An operation governed by an internal start trigger begins executing as soon as the operation is enabled. Consequently, the call to **K_DMAStart** or **K_IntStart** is considered the trigger event for an internal trigger. The internal trigger is the default trigger source.

Analog Trigger

You can use the signal on any analog input channel as the trigger signal for an analog trigger. The trigger events for analog triggers are illustrated in Figure 2-2 and described as follows:

- If the trigger polarity is positive, a trigger event occurs the first time the trigger signal changes from a voltage that is less than the trigger level to a voltage that is greater than the trigger level.

- If the trigger polarity is negative, a trigger event occurs the first time the trigger signal changes from a voltage that is greater than the trigger level to a voltage that is less than the trigger level.
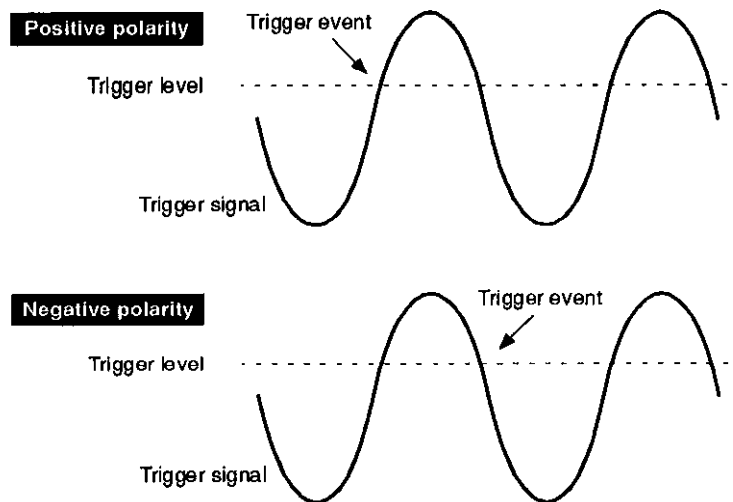
Figure 2-2. Trigger Events for Analog Triggers

Note: Analog triggering is a feature of the Function Call Driver and is not implemented at the hardware level. Consequently, there is a delay between the time the trigger event occurs and the time the driver recognizes that the trigger event occurred.

You can specify a hysteresis value to prevent noise from triggering an operation. Use the **K_SetTrigHyst** function to specify the hysteresis value. For a positive-edge trigger, the analog signal must be below the specified voltage level by at least the amount of the hysteresis value and then rise above the voltage level before the trigger occurs; for a negative-edge trigger, the analog signal must be above the specified voltage level by at least the amount of the hysteresis value and then fall below the voltage level before the trigger occurs.

The hysteresis value is an absolute number, which you specify as a raw count value between 0 and 4095 for DAS-1800HC/ST Series boards and between 0 and 65,535 for DAS-1800HR Series boards. When you add the hysteresis value to the voltage level (for a negative-edge trigger) or subtract the hysteresis value from the voltage level (for a positive-edge trigger), the resulting value must also be between 0 and 4095 for DAS-1800ST/HC Series boards or between 0 and 65,535 for DAS-1800HR Series boards. For example, assume that you are using a negative-edge trigger on a channel of a DAS-1800HC/ST Series board configured for an analog input range of ±5 V. If the voltage level is +4.8 V (4014 counts), you can specify a hysteresis value of 0.1 V (41 counts) because 4014 + 41 is less than 4095, but you cannot specify a hysteresis value of 0.3 V (123 counts) because 4014 + 123 is greater than 4095. Refer to Appendix B for information on how to convert a voltage value to a raw count value.

In Figure 2-3, the specified voltage level is +4 V and the hysteresis value is 0.1 V. The analog signal must be below +3.9 V and then rise above +4 V before a positive-edge trigger occurs; the analog signal must be above +4.1 V and then fall below +4 V before a negative-edge trigger occurs.
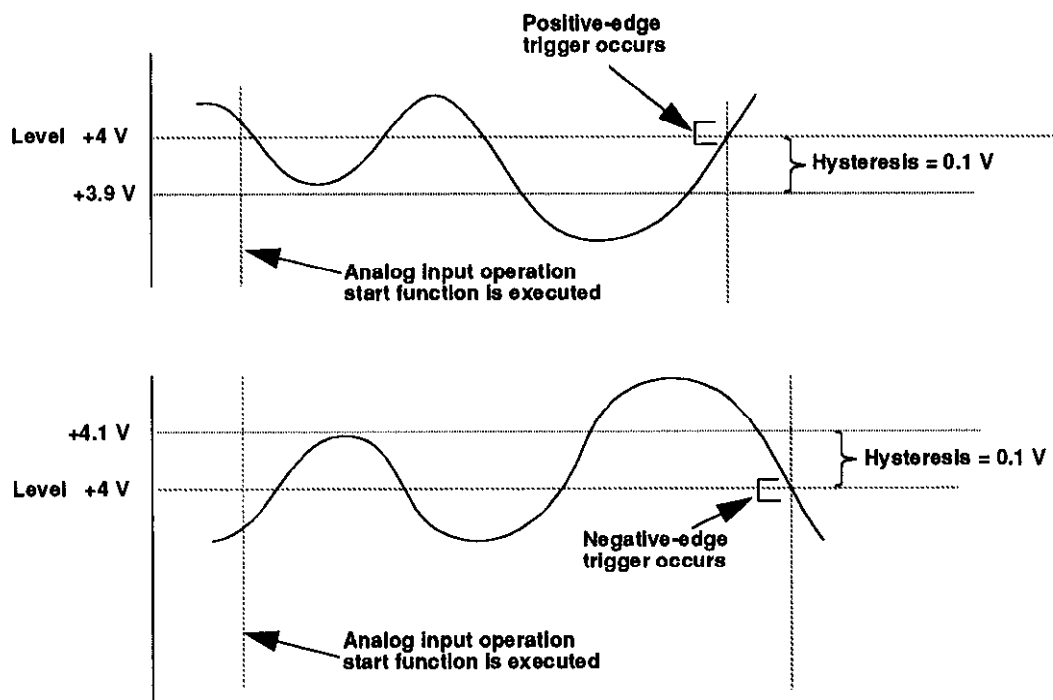
Figure 2-3. Using a Hysteresis Value

Digital Trigger

The digital trigger signal is available on the DI1/TGIN pin (pin B40) on the main I/O connector of DAS-1800HC Series boards and on the TGIN pin (pin 46) on the main I/O connector of DAS-1800ST/HR Series boards. Use **K_SetDITrig** to specify whether you want the trigger event to occur on a rising or falling edge. If the trigger polarity is positive, then a trigger event occurs at each rising edge of the trigger signal. If the trigger polarity is negative, then a trigger event occurs at each falling edge of the trigger signal. These trigger events are illustrated in Figure 2-4.
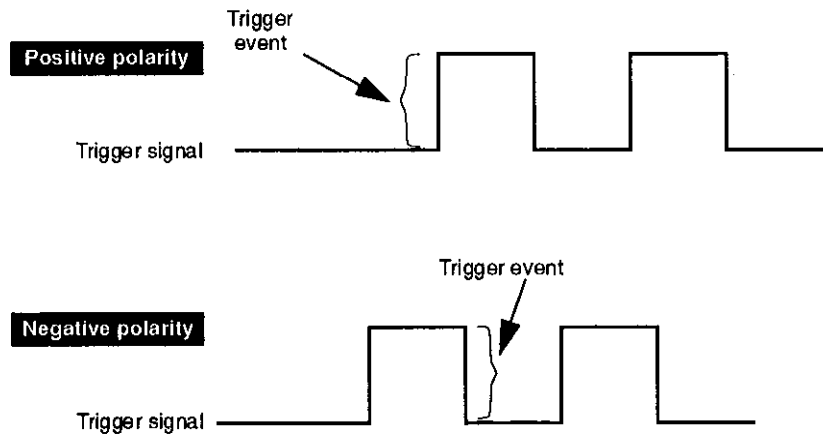
Figure 2-4. Trigger Events For Digital Triggers

Post-Trigger Acquisition

Use post-trigger acquisition in applications where you want to collect data after a specific event. Acquisition starts on an internal, analog, or digital trigger event and continues until a specified number of samples has been acquired or until the operation is stopped by a call to **K_DMAStop** or **K_IntStop**.

To specify post-trigger acquisition, use the following function calls:

1. If you want acquisition to continue until you stop it using **K_DMAStop** or **K_IntStop**, use **K_SetContRun** to set the buffering mode to continuous.

2. If you want acquisition to stop after a specified number of samples has been acquired, use **K_ClrContRun** to set the buffering mode to single-cycle (in this buffering mode, the operation stops as soon as the board has acquired the number of samples specified by **K_SetBuf, K_SetDMABuf, K_SetBufI,** or **K_BufListAdd**).

3. Specify the trigger that will start the operation. Use **K_SetTrig** to specify the trigger source (internal for an internal trigger, external for an analog or digital trigger).

4. If you are using an analog or digital trigger, use **K_SetADTrig** (for an analog trigger) or **K_SetDITrig** (for a digital trigger) to define the trigger conditions.

5. Use **K_ClrAboutTrig** to disable the about trigger.

Pre-Trigger Acquisition

Use pre-trigger acquisition in applications where you want to collect data before a specific digital trigger event (this is the about trigger event). Acquisition starts on an internal, analog, or digital trigger event and continues until the about-trigger event. Pre-trigger acquisition is available with DMA-mode operations only.

To specify pre-trigger acquisition, use the following function calls:

1. Specify the trigger that will start the operation. Use **K_SetTrig** to specify the trigger source (internal for an internal trigger, external for an analog or digital trigger).

2. If using an analog or digital start trigger, use **K_SetADTrig** (for an analog trigger) or **K_SetDITrig** (for a digital trigger) to define the trigger conditions.

3. Use **K_SetAboutTrig** to enable the about trigger and to set the number of post-trigger samples to 1.

4. If the start trigger is not digital, specify the trigger conditions for the about trigger. Use **K_SetTrig** to specify an external trigger, then use **K_SetDITrig** to specify the trigger conditions. (If the start trigger is digital, then its trigger conditions are also used for the about trigger).

About-Trigger Acquisition

Use about-trigger acquisition in applications where you want to collect data both before and after a specific digital trigger event (this is the about trigger event). Acquisition starts on an internal, analog, or digital trigger event and continues until a specified number of samples has been acquired after the about-trigger event. About-trigger acquisition is available with DMA-mode operations only.

To specify about-trigger acquisition, use the following function calls:

1. Specify the trigger that will start the operation. Use **K_SetTrig** to specify the trigger source (internal for an internal trigger, external for an analog or digital trigger).

2. If using an analog or digital start trigger, use **K_SetADTrig** (for an analog trigger) or **K_SetDITrig** (for a digital trigger) to define the trigger conditions.

3. Use **K_SetAboutTrig** to enable the about trigger and to specify the desired number of post-trigger samples.

4. Specify the trigger conditions for the about trigger. Use **K_SetDITrig** to specify the trigger conditions. (If the start trigger is digital, then its trigger conditions are also used for the about trigger).

Hardware Gates

A hardware gate is an externally applied digital signal that determines whether conversions occur. You connect the gate signal to the DI1/TGIN pin (pin B40) on the main I/O connector of DAS-1800HC Series boards or on the TGIN pin (pin 46) on the main I/O connector of DAS-1800ST/HR Series boards. If you have started an interrupt-mode or DMA-mode analog input operation (using **K_IntStart** or **K_DMAStart**) and the hardware gate is enabled, the state of the gate signal determines whether conversions occur.

If the board is configured with a positive gate, conversions occur only if the signal to DI1/TGIN (DAS-1800HC Series boards) or TGIN (DAS-1800ST/HR Series boards) is high; if the signal to DI1/TGIN or TGIN is low, conversions are inhibited. If the board is configured with a negative gate, conversions occur only if the signal to DI1/TGIN is low; if

the signal to DI1/TGIN is high, conversions are inhibited. Use the
K_SetGate function to enable and disable the hardware gate and to
specify the gate polarity (positive or negative). The default state of the
hardware gate is disabled.

You can use the hardware gate with an external analog trigger. The
software waits until the analog trigger conditions are met, and then the
hardware checks the state of the gate signal.

If you are not using an analog trigger, the gate signal itself can act as a
trigger. If the gate signal is in the inactive state when you start the analog
input operation, the hardware waits until the gate signal is in the active
state before conversions begin.

Note: You cannot use the hardware gate with an external digital trigger. If
you use a digital trigger at one point in your application program and later
want to use a hardware gate, you must first disable the digital trigger. You
disable the digital trigger by specifying an internal trigger in **K_SetTrig**
or by setting up an analog trigger (using the **K_SetADTrig** function).

Analog Output Operations (DAS-1800HC Series Only)

This section describes the following:

● Analog output operation modes available.

● How to allocate and manage memory for analog output operations.

● How to specify the following for an analog output operation:
channels, a clock rate, and a buffering mode.

Note: You cannot use an external trigger or external pacer clock with an
analog output operation.

Operation Modes

The operation mode determines which attributes you can specify for an analog output operation. You can perform analog output operations in one of the following modes:

- **Single mode** - In single mode, the driver writes a single value to one or both analog output channels; you cannot perform any other operation until the single-mode operation is complete.

  Use the **K_DAWrite** function to start an analog output operation in single mode. You specify the board you want to use, the analog output channel(s), and the value you want to write.

- **Interrupt mode** - In interrupt mode, the driver writes a single value or multiple values to one or both analog output channels. A hardware clock paces the updating of the analog output channel(s). Once the analog output operation begins, control returns to your application program. You store the values you want to write in a user-defined buffer in the computer.

  Use the **K_IntStart** function to start an analog output operation in interrupt mode. You specify the board, analog output channel(s), clock rate, buffering mode, and buffer address.

  You can specify either single-cycle or continuous buffering mode for interrupt-mode operations. Refer to page 2-30 for more information on buffering modes. Use the **K_IntStop** function to stop a continuous-mode interrupt operation. Use the **K_IntStatus** function to determine the current status of an interrupt operation.

For an analog output operation, the values are written as raw counts. For information on converting voltage to raw counts, refer to Appendix B.

Memory Allocation and Management

Interrupt-mode analog output operations use a single memory buffer to store the data to be written to the analog output channel(s). The maximum number of samples each memory buffer can store (32K or 64K) depends on the language you are using. See "Language-Specific Programming Information" on page 3-22 for more information.

Since analog output operations typically require small arrays of data, you can reserve a memory buffer by dimensioning an array within your application program's memory area. Since the DAS-1800 Series Function Call Driver writes data as 16-bit integers, you must dimension all local arrays as integers.

Note: You can also use the **K_IntAlloc** function to allocate memory dynamically, if desired. You specify the operation requiring the buffer, the number of values you want to store in the buffer, the starting address of the buffer, and the name you want to use to identify the buffer (this name is called the memory handle). When the buffer is no longer required, you can free the buffer for another use by specifying this memory handle in the **K_IntFree** function.

After you dimension your array, you must assign the starting address of the array and the number of samples stored in the array. Each supported programming language requires a particular procedure for dimensioning an array and assigning the starting address. Refer to page 3-23 for information when programming in C or C++. Refer to page 3-31 for information when programming in Pascal. Refer to page 3-40 for information when programming in Visual Basic for Windows. Refer to page 3-46 for information when programming in BASIC.

Channels

DAS-1800HC Series boards contain two digital-to-analog converters, each of which is associated with an analog output channel. You can perform an analog output operation on a single channel or on both channels.

For single-mode analog output operations, you can write a single value to one or both analog output channels. Use the **K_DAWrite** function to specify the channel(s).

For interrupt-mode analog output operations, you can write a single value or multiple values to one or both analog output channels. Use the **K_SetChn** function to specify a single channel. Use the **K_SetStartStopChn** function to specify analog output channel 0 as the start channel and analog output channel 1 as the stop channel. When using

both channels, the first value in the buffer is written to channel 0, the second value is written to channel 1, the third value is written to channel 0 again, and so on. After all the values in the buffer are written once, the values are written again until the required number of values are written.

For example, assume that your buffer contains three values (123, 456, and 789) and you want to write five values. Your program writes 123 to channel 0, 456 to channel 1, 789 to channel 0, 123 to channel 1, and 456 to channel 0.

Clock Source

When performing interrupt-mode analog output operations, you can use the internal pacer clock to determine the period between the updating of a single analog output channel or between each simultaneous updating of both analog output channels.

Note: You can use the internal pacer clock only if it is not being used by another operation.

The internal pacer clock uses two cascaded counters of the onboard counter/timer circuitry. The counters are normally in an idle state. When you start the analog output operation (using **K_IntStart**), the analog output channel(s) are updated. Note that a slight time delay occurs between the time the operation is started and the time the channel(s) are updated.

The counters are loaded with a count value and begin counting down. When the counters count down to 0, the channel(s) are updated again and the process repeats.

Because the counters use a 5 MHz time base, each count represents 0.2 µs. Use the **K_SetClkRate** function to specify the number of counts (clock ticks) between updates. For example, if you specify a count of 5000, the period between updates is 1 ms (1 ksamples/s); if you specify a count of 87654, the period between updates is 17.53 ms (57 samples/s).

You can specify a count between 15 and 4,294,967,295. The period between updates ranges from 3 µs to 14.3 minutes.

Note: The driver accepts a count value as low as 15. However, since the FIFO is not used to buffer values for analog output operations, a low count value may cause an overrun error. The maximum observed update rates for the internal pacer clock are 1 ksamples/s when running under Windows and 5 ksamples/s when running under DOS. These rates would indicate a minimum count of 5,000 when running under Windows and 1,000 when running under DOS.

Use the following formula to determine the number of counts to specify:

$$counts = \frac{5 \text{ MHz time base}}{update \text{ rate}}$$

For example, if you want to update the analog output channels at a rate of 500 samples/s, specify a count of 10,000, as shown in the following equation:

$$\frac{5,000,000}{500} = 10,000$$

Buffering Modes

The buffering mode determines how the driver writes the values in the buffer to the analog output channels. For interrupt-mode analog output operations, you can specify one of the following buffering modes:

- **Single-cycle mode** - In single-cycle mode, after the driver writes the values stored in the buffer, the operation stops automatically. Single-cycle mode is the default buffering mode.

- **Continuous mode** - In continuous mode, the driver continuously writes values from the buffer until the application program issues a stop function; when all the values in the buffer have been written, the driver writes the values again. Use the **K_SetContRun** function to specify continuous buffering mode.

Digital I/O Operations

This section describes the following:

- Digital I/O operation modes available.

- How to allocate and manage memory for digital I/O operations.

- Digital I/O channels.

- How to specify the following for a digital I/O operation: a clock rate and a buffering mode.

Note: You cannot use an external trigger or external pacer clock with a digital I/O operation.

Operation Modes

The operation mode determines which attributes you can specify for a digital I/O operation. You can perform digital I/O operations in one of the following modes:

- **Single mode** - In a single-mode digital input operation, the driver reads the value of digital input channel 0 once; in a single-mode digital output operation, the driver writes a value to digital output channel 0 once. You cannot perform any other operation until the single-mode operation is complete.

    Use the **K_DIRead** function to start a digital input operation in single mode; use the **K_DOWrite** function to start a digital output operation in single mode. You specify the board you want to use, the digital I/O channel, and the variable in which the value is stored.

Notes: Since digital input channel 0 is only four bits wide, you must mask the value stored by **K_DIRead** with 15 (0Fh) to obtain the actual digital input value.

The value written by **K_DOWrite** must be a 32-bit value. For DAS-1800HC Series boards, the eight least significant bits contain the actual digital output value, and all other bits are irrelevant. For DAS-1800ST/HR Series boards, the four least significant bits contain the actual digital output value, and all other bits are irrelevant.

- **Interrupt mode** - In an interrupt-mode digital input operation, the driver reads the value of digital input channel 0 multiple times; in an interrupt-mode digital output operation, the driver writes a single value or multiple values to digital output channel 0 multiple times. A hardware clock paces the digital I/O operation. Once the digital I/O operation begins, control returns to your application program. The driver stores digital input values in a user-defined buffer in the computer; you store digital output values in a user-defined buffer in the computer.

Note: The digital input buffer and the digital output buffer each contain 16-bit integers. Each digital input value is stored in the four least significant bits of each integer in the digital input buffer. For DAS-1800HC Series boards, each digital output value is stored in the eight least significant bits of each integer in the digital output buffer. For DAS-1800ST/HR Series boards, each digital output value is stored in the four least significant bits of each integer in the digital output buffer.

Use the **K_IntStart** function to start a digital I/O operation in interrupt mode. You specify the board, digital I/O channel, clock rate, buffering mode, and buffer address.

You can specify either single-cycle or continuous buffering mode for interrupt-mode operations. Refer to page 2-38 for more information on buffering modes. Use the **K_IntStop** function to stop a continuous-mode interrupt operation. Use the **K_IntStatus** function to determine the current status of an interrupt operation.

Memory Allocation and Management

Interrupt-mode digital I/O operations use a single memory buffer to store the data to be read or written. The maximum number of samples each memory buffer can store (32K or 64K) depends on the language you are using. See "Language-Specific Programming Information" on page 3-22 for more information.

Since digital I/O operations typically require small arrays of data, you can reserve a memory buffer by dimensioning an array within your application program's memory area. Since the DAS-1800 Series Function Call Driver reads and writes data as 16-bit integers, you must dimension all local arrays as integers.

Note: You can also use the **K_IntAlloc** function to allocate memory dynamically, if desired. You specify the operation requiring the buffer, the number of values to store in the buffer, the variable in which to store the starting address of the buffer, and the name you want to use to identify the buffer (this name is called the memory handle). When the buffer is no longer required, you can free the buffer for another use by specifying this memory handle in the **K_IntFree** function.

After you dimension or allocate your array, you must assign the starting address of the array and the number of samples to store in the array. Each supported programming language requires a particular procedure for dimensioning an array and assigning the starting address. Refer to page 3-23 for information when programming in C or C++. Refer to page 3-31 for information when programming in Pascal. Refer to page 3-40 for information when programming in Visual Basic for Windows. Refer to page 3-46 for information when programming in BASIC.

Digital Input Channel

DAS-1800 Series boards contain one 4-bit digital input channel (channel 0). As shown in Figure 2-5, bit 0 contains the value of digital input line 0 (DI0/XPCLK on DAS-1800HC Series boards, DI0 on DAS-1800ST/HR Series boards); bit 1 contains the value of digital input line 1 (DI1/TGIN on DAS-1800HC Series boards, DI1 on DAS-1800ST/HR Series boards); bit 2 contains the value of digital input line 2 (DI2); bit 3 contains the value of digital input line 3 (DI3).

	bit 3	bit 2	bit 1	bit 0
DAS-1800HC	DI3	DI2	DI1/ TGIN	DI0/ XPCLK

	bit 3	bit 2	bit 1	bit 0
DAS-1800ST/HR	DI3	DI2	DI1	DI0

Figure 2-5. Digital Input Bits

A value of 1 in the bit position indicates that the input is high; a value of 0 in the bit position indicates that the input is low. For example, if the value is 5 (0101), the input at DI0/XPCLK and DI2 is high and the input at DI1/TGIN and DI3 is low.

Available Operations

Notes: If no signal is connected to a digital input line, the input appears high (value is 1).

(DAS-1800HC Series boards only) If you are using an external pacer clock, you cannot use digital input line 0 for general-purpose digital input operations. If you are using an external digital trigger, you cannot use digital input line 1 for general-purpose digital input operations. When reading digital input channel 0, ignore the value of these bits.

Digital Output Channel

DAS-1800HC Series boards contain one 8-bit digital output channel (channel 0). DAS-1800ST/HR Series boards contain one 4-bit digital output channel (channel 0). As shown in Figure 2-6, bit 0 contains the value to be written to digital output line 0 (DO0), bit 1 contains the value to be written to digital output line 1 (DO1), and so on.

DAS-1800ST/HR Series

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

DAS-1800HC Series
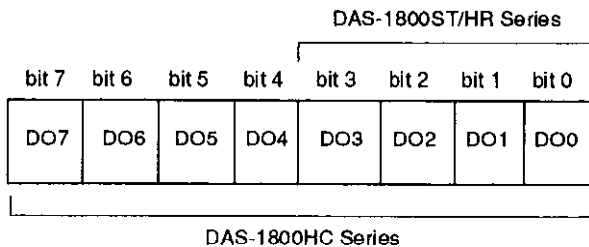
Figure 2-6. Digital Output Bits

A value of 1 in the bit position indicates that the output is high; a value of 0 in the bit position indicates that the output is low. For example, if the value written is 12 (00001100), the output at DO0, DO1, DO4, DO5, DO6, and DO7 is forced low and the output at DO2 and DO3 is forced high.

Note: The DAS-1800 Series Function Call Driver provides the **K_GetDOCurVal** function to read the last digital output value written to digital output channel 0 using **K_DOWrite**.

Clock Source

When performing interrupt-mode digital I/O operations, you can use the internal pacer clock to determine the period between reading the digital input channel or writing to the digital output channel.

Note: You can use the internal pacer clock only if it is not being used by another operation.

The internal pacer clock uses two cascaded counters of the onboard counter/timer circuitry. The counters are normally in an idle state. When you start the digital I/O operation (using **K_IntStart**), a value is read or written. Note that a slight time delay occurs between the time the operation is started and the time the reading or writing begins.

The counters are loaded with a count value and begin counting down. When the counters count down to 0, another value is read or written and the process repeats.

Because the counters use a 5 MHz time base, each count represents 0.2 μs. Use the **K_SetClkRate** function to specify the number of counts (clock ticks) between reads or writes. For example, if you specify a count of 5000, the period between reads or writes is 1 ms (1 ksamples/s); if you specify a count of 87654, the period between reads or writes is 17.53 ms (57 samples/s).

You can specify a count between 15 and 4,294,967,295. The period between reads or writes ranges from 3 μs to 14.3 minutes.

Note: The driver accepts a count value as low as 15. However, since the FIFO is not used to buffer values for digital I/O operations, a low count value may cause overrun errors. The maximum observed update rates for the internal pacer clock are 1 ksamples/s when running under Windows and 5 ksamples/s when running under DOS. These rates would indicate a minimum count of 5,000 when running under Windows and 1,000 when running under DOS.

Use the following formula to determine the number of counts to specify:

$$counts = \frac{5 \text{ MHz time base}}{\text{read/write rate}}$$

For example, if you want to write data to digital output channel 0 at a rate of 500 samples/s, specify a count of 10,000, as shown in the following equation:

$$\frac{5,000,000}{500} = 10,000$$

Buffering Modes

The buffering mode determines how the driver reads or writes the values in the buffer. For interrupt-mode digital I/O operations, you can specify one of the following buffering modes:

- **Single-cycle mode** - In a single-cycle-mode digital input operation, after the driver fills the buffer, the operation stops automatically. In a single-cycle-mode digital output operation, after the driver writes the values stored in the buffer, the operation stops automatically. Single-cycle mode is the default buffering mode.

- **Continuous mode** - In a continuous-mode digital input operation, the driver continuously reads digital input channel 0 and stores the values in the buffer until the application program issues a stop function; any values already stored in the buffer are overwritten. In a continuous mode digital output operation, the driver continuously writes values from the buffer to digital output channel 0 until the application program issues a stop function; when all the values in the buffer have been written, the driver writes the values again. You use the **K_SetContRun** function to specify continuous buffering mode.

3

Programming with the Function Call Driver

This chapter contains an overview of the structure of the DAS-1800 Series Function Call Driver, as well as programming guidelines and language-specific information to assist you when writing application programs with the DAS-1800 Series Function Call Driver.

How the Driver Works

When writing application programs, you can use functions from one or more Keithley DAS Function Call Drivers. You initialize each driver according to a particular configuration file. If you are using more than one driver or more than one configuration file with a single driver, the driver handle uniquely identifies each driver or each use of the driver.

You can program one or more boards in your application program. You initialize each board using a board handle to uniquely identify each board. Each board handle is associated with a particular driver.

The Function Call Driver(s) allow you to perform I/O operations in various operation modes. For single mode, the I/O operation is performed with a single call to a function; the attributes of the I/O operation are specified as arguments to the function. Figure 3-1 illustrates the syntax of the single-mode, analog input operation function **K_ADRead**.

Single-Mode Function   Attributes of Operation

K_ADRead   (*board,*   ←————————→   Board number

            channel,   ←————————→   Analog Input channel

            gain,   ←————————→   Gain applied to channel

            buffer)   ←————————→   Buffer for data

Figure 3-1. Single-Mode Function

For other operation modes, such as interrupt mode and DMA mode, the
driver uses frames to perform the I/O operation. A frame is a data
structure whose elements define the attributes of the I/O operation. Each
frame is associated with a particular board, and therefore, to a particular
driver.

Frames help you create structured application programs. You set up the
attributes of the I/O operation in advance, using a separate function call
for each attribute, and then start the operation at an appropriate point in
your program. Frames are useful for operations that have many defining
attributes, since providing a separate argument for each attribute could
make a function's argument list unmanageably long. In addition, some
attributes, such as the clock source and trigger source, are only available
for I/O operations that use frames.

You indicate that you want to perform an I/O operation by getting an
available frame for the driver and specifying the name you want to use to
identify the frame; this name is called the frame handle. You then specify
the attributes of the I/O operation by using setup functions to define the
elements of the frame associated with the operation. For example, to
specify the channel on which to perform an I/O operation, you might use
the **K_SetChn** setup function.

For each setup function, the Function Call Driver provides a readback
function, which reads the current definition of a particular element. For
example, the **K_GetChn** readback function reads the channel number
specified for the I/O operation.

You use the frame handle you specified when accessing the frame in all setup functions, readback functions, and other functions related to the I/O operation. This ensures that you are defining the same I/O operation.

When you are ready to perform the I/O operation you have set up, you can start the operation in the appropriate operation mode, referencing the appropriate frame handle. Figure 3-2 illustrates the syntax of the interrupt-mode operation function **K_IntStart**.

K_IntStart (*frameHandle*)

Frame	**Attributes of Operation**
Start Channel	← → First analog input channel
Stop Channel	← → Last analog input channel
Clock Source	← → Pacer clock source
Trigger Source	← → Trigger source
.	.
.	.
.	.
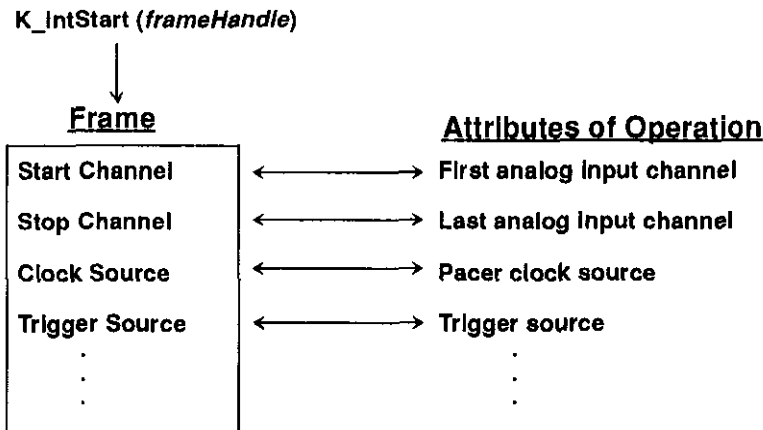
Figure 3-2. Interrupt-Mode Operation

Different I/O operations require different types of frames. For example, to perform a digital input operation, you use a digital input frame; to perform an analog output operation, you use an analog output frame.

For DAS-1800 Series boards, interrupt-mode and DMA-mode operations require frames. The DAS-1800 Series Function Call Driver provides the following types of frames:

- Analog input frames, called A/D (analog-to-digital) frames. You use the **K_GetADFrame** function to access an available A/D frame and specify a frame handle.

- Analog output frames, called D/A (digital-to-analog) frames. You use the **K_GetDAFrame** function to access an available D/A frame and specify a frame handle.

- Digital input frames, called DI frames. You use the **K_GetDIFrame** function to access an available DI frame and specify a frame handle.

- Digital output frames, called DO frames. You use the **K_GetDOFrame** function to access an available DO frame and specify a frame handle.

If you want to perform an interrupt-mode or DMA-mode operation and all frames of a particular type have been accessed, you can use the **K_FreeFrame** function to free a frame that is no longer in use. You can then redefine the elements of the frame for the next operation.

When you access a frame, the elements are set to their default values. You can also use the **K_ClearFrame** function to reset all the elements of a frame to their default values.

Table 3-1 lists the elements of a DAS-1800 Series A/D frame; Table 3-2 lists the elements of a DAS-1800 Series D/A frame; Table 3-3 lists the elements of a DAS-1800 Series DI frame; Table 3-4 lists the elements of a DAS-1800 Series DO frame. These tables also list the default value of each element, the setup function(s) used to define each element, and the readback function(s) used to read the current definition of the element.

Table 3-1. A/D Frame Elements

Element	Default Value	Setup Function	Readback Function
Buffer[1]	0 (NULL)	K_SetBuf K_SetBufI K_SetDMABuf K_BufListAdd	K_GetBuf
Number of Samples	0	K_SetBuf K_SetBufI K_BufListAdd	K_GetBuf
Buffering Mode	Single-cycle	K_SetContRun K_ClrContRun[2]	K_GetContRun
Start Channel	0	K_SetChn K_SetStartStopChn K_SetStartStopG	K_GetChn K_GetStartStopChn K_GetStartStopG
Stop Channel	0	K_SetStartStopChn K_SetStartStopG	K_GetStartStopChn K_GetStartStopG
Gain	0 (gain of 1)	K_SetG K_SetStartStopG	K_GetG K_GetStartStopG
Channel-Gain Queue	0 (NULL)	K_SetChnGAry	K_GetChnGAry
Conversion Mode	Paced	K_SetADFreeRun K_ClrADFreeRun[2]	K_GetADFreeRun
SSH Mode	Disabled	K_SetSSH	K_GetSSH
Clock Source	Internal	K_SetClk	K_GetClk
Pacer Clock Rate[1]	0	K_SetClkRate	K_GetClkRate
External Clock Edge	Negative	K_SetExtClkEdge	K_GetExtClkEdge
Burst Clock Rate	3 (333 ksamples/s)	K_SetBurstTicks	K_GetBurstTicks
Trigger Source	Internal	K_SetTrig	K_GetTrig
Trigger Type	Digital	K_SetADTrig K_SetDITrig	K_GetADTrig K_GetDITrig

Table 3-1. A/D Frame Elements (cont.)

Element	Default Value	Setup Function	Readback Function
Trigger Channel	0 (for analog trigger)	K_SetADTrig	K_GetADTrig
	0 (channel 0, bit 0) (for digital trigger)	Not applicable[3]	Not applicable[3]
Trigger Polarity	Positive (for analog trigger)	K_SetADTrig	K_GetADTrig
	Positive (for digital trigger)	K_SetDITrig	K_GetDITrig
Trigger Sensitivity	Edge (for analog and digital trigger)	Not applicable[3]	Not applicable[3]
Trigger Level	0	K_SetADTrig	K_GetADTrig
Trigger Hysteresis	0	K_SetTrigHyst	K_GetTrigHyst
Trigger Pattern	Not used[4]	Not applicable[3]	Not applicable[3]
About-Trigger Mode	Disabled	K_SetAboutTrig K_ClrAboutTrig[2]	K_GetAboutTrig
Hardware Gate	Disabled	K_SetGate	K_GetGate

Notes

[1] This element must be set.
[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.
[3] The default value of this element cannot be changed.
[4] This element is not currently used; it is included for future compatibility.

Programming with the Function Call Driver

Table 3-2. D/A Frame Elements

Element	Default Value	Setup Function	Readback Function
Buffer[1]	0 (NULL)	K_SetBuf K_SetBufI	K_GetBuf
Buffering Mode	Single-cycle	K_SetContRun K_ClrContRun[2]	K_GetContRun
Number of Samples	0	K_SetBuf K_SetBufI	K_GetBuf
Start Channel	0	K_SetChn K_SetStartStopChn	K_GetChn K_GetStartStopChn
Stop Channel	0	K_SetStartStopChn	K_GetStartStopChn
Clock Source	Internal	Not applicable[3]	Not applicable[3]
Pacer Clock Rate[1]	0	K_SetClkRate	K_GetClkRate

Notes

[1] This element must be set.

[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.

[3] The default value of this element cannot be changed.

Table 3-3. DI Frame Elements

Element	Default Value	Setup Function	Readback Function
Buffer[1]	0 (NULL)	K_SetBuf K_SetBufI	K_GetBuf
Buffering Mode	Single-cycle	K_SetContRun K_ClrContRun[2]	K_GetContRun
Number of Samples	0	K_SetBuf K_SetBufI	K_GetBuf
Start Channel	0	Not applicable[3]	Not applicable[3]
Stop Channel	0	Not applicable[3]	Not applicable[3]
Clock Source	Internal	Not applicable[3]	Not applicable[3]
Pacer Clock Rate[1]	0	K_SetClkRate	K_GetClkRate

Notes

[1] This element must be set.

[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.

[3] The default value of this element cannot be changed.

Table 3-4. DO Frame Elements

Element	Default Value	Setup Function	Readback Function
Buffer[1]	0 (NULL)	K_SetBuf K_SetBufI	K_GetBuf
Buffering Mode	Single-cycle	K_SetContRun, K_ClrContRun[2]	K_GetContRun
Number of Samples	0	K_SetBuf K_SetBufI	K_GetBuf
Start Channel	0	Not applicable[3]	Not applicable[3]
Stop Channel	0	Not applicable[3]	Not applicable[3]
Clock Source	Internal	Not applicable[3]	Not applicable[3]
Pacer Clock Rate[1]	0	K_SetClkRate	K_GetClkRate

Notes

[1] This element must be set.

[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.

[3] The default value of this element cannot be changed.

Note: The DAS-1800 Series Function Call Driver provides many other functions that are not related to controlling frames, defining the elements of frames, or reading the values of frame elements. These functions include single-mode operation functions, initialization functions, memory management functions, and miscellaneous functions

---

For information about using the FCD functions in your application program, refer to the following sections of this chapter. For detailed information about the syntax of FCD functions, refer to Chapter 4.

Programming Overview

To write an application program using the DAS-1800 Series Function Call Driver, perform the following steps:

1. Define the application's requirements. Refer to Chapter 2 for a description of the board operations supported by the Function Call Driver and the functions that you can use to define each operation.

2. Write your application program. Refer to the following for additional information:

   - Preliminary Tasks, the next section, describes the programming tasks that are common to all application programs.

   - Operation-Specific Programming Tasks, on page 3-11, describes operation-specific programming tasks and the sequence in which these tasks must be performed.

   - Chapter 4 contains detailed descriptions of the FCD functions.

   - The DAS-1800 Series standard software package and the ASO-1800 software package contain several example programs. The FILES.TXT file in the installation directory lists and describes the example programs.

3. Compile and link the program. Refer to Language-Specific Programming Information, starting on page 3-22, for compile and link statements and other language-specific considerations for each supported language.

Preliminary Tasks

For every Function Call Driver application program, you must perform the following preliminary tasks:

1. Include the function and variable type definition file for your language. Depending on the specific language you are using, this file is included in the DAS-1800 Series standard software package or the ASO-1800 software package.

2. Declare and initialize program variables.

3. Use a driver initialization function (**DAS1800_DevOpen** or **K_OpenDriver**) to initialize the driver.

4. Use a board initialization function (**DAS1800_GetDevHandle** or **K_GetDevHandle**) to specify the board you want to use and to initialize the board. If you are using more than one board, use the board initialization function once for each board you are using.

Operation-Specific Programming Tasks

After completing the preliminary tasks, perform the appropriate operation-specific programming tasks. The operation-specific tasks for analog and digital I/O operations are described in the following sections.

Note: Any FCD functions that are not mentioned in the operation-specific programming tasks can be used at any point in your application program.

Analog Input Operations

The following subsections describe the operation-specific programming tasks required to perform single-mode, interrupt-mode, and DMA-mode analog input operations.

Single Mode

For a single-mode analog input operation, perform the following tasks:

1. Declare the buffer or variable in which to store the single analog input value.

2. Use the **K_ADRead** function to read the single analog input value; specify the attributes of the operation as arguments to the function.

Interrupt Mode

For an interrupt-mode analog input operation, perform the following tasks:

1. Use the **K_GetADFrame** function to access an A/D frame.

2. Allocate the buffer(s) or dimension the array(s) in which to store the acquired data. Use the **K_IntAlloc** function if you want to allocate the buffer(s) dynamically outside your program's memory area.

3. *If you want to use a channel-gain queue to specify the channels acquiring data*, define and assign the appropriate values to the queue and note the starting address. Refer to page 2-11 for more information about channel-gain queues.

4. Use the appropriate setup functions to specify the attributes of the operation. The setup functions are listed in Table 3-5.

Note: When you access a new A/D frame, the frame elements contain default values. If the default value of a particular element is suitable for your operation, you do not have to use the setup function associated with that element. Refer to Table 3-1 on page 3-5 for a list of the default values of A/D frame elements.

Table 3-5. Setup Functions for Interrupt-Mode Analog Input Operations

Attribute	Setup Function(s)
Buffer[1]	K_SetBuf K_SetBufI K_BufListAdd
Number of Samples	K_SetBuf K_SetBufI K_BufListAdd
Buffering Mode	K_SetContRun K_ClrContRun[2]
Start Channel	K_SetChn K_SetStartStopChn K_StartStopG
Stop Channel	K_SetStartStopChn K_SetStartStopG
Channel Configuration	K_SetADConfig
Input Range Type	K_SetADMode
Common-mode ground reference	K_SetADCommonMode
Gain	K_SetG K_SetStartStopG
Channel-Gain Queue	K_SetChnGAry
Conversion Mode	K_SetADFreeRun K_ClrADFreeRun[2]
SSH Mode	K_SetSSH
Clock Source	K_SetClk
Pacer Clock Rate[1]	K_SetClkRate
External Clock Edge	K_SetExtClkEdge
Burst Clock Rate	K_SetBurstTicks
Trigger Source	K_SetTrig
Trigger Type	K_SetADTrig K_SetDITrig

Table 3-5. Setup Functions for Interrupt-Mode Analog Input Operations (cont.)

Attribute	Setup Function(s)
Trigger Channel	K_SetADTrig
Trigger Polarity	K_SetADTrig
Trigger Level	K_SetADTrig
Trigger Hysteresis	K_SetTrigHyst
Hardware Gate	K_SetGate

Notes

[1] This element must be set.

[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame.

Refer to Chapter 2 for background information about the setup functions; refer to Chapter 4 for detailed descriptions of the setup functions.

5. Use the **K_IntStart** function to start the interrupt-mode operation.

6. Use the **K_IntStatus** function to monitor the status of the interrupt-mode operation.

7. *If you specified continuous buffering mode*, use the **K_IntStop** function to stop the interrupt-mode operation when the appropriate number of samples has been acquired.

8. *If you are programming in Visual Basic for Windows or BASIC and you used K_IntAlloc to allocate your buffer(s)*, use the **K_MoveBufToArray** function to transfer the acquired data from the allocated buffer to a local array that your program can use.

9. *If you used K_IntAlloc to allocate your buffer(s)*, use the **K_IntFree** function to deallocate the buffer(s).

10. *If you used K_BufListAdd to specify a list of multiple buffers*, use the **K_BufListReset** function to clear the list.

Programming with the Function Call Driver

11. Use the **K_FreeFrame** function to return the frame you accessed in step 1 to the pool of available frames.

DMA Mode

For a DMA-mode analog input operation, perform the following tasks:

1. Use the **K_GetADFrame** function to access an A/D frame.

2. Allocate the buffer(s) or dimension the array(s) in which to store the acquired data. Use the **K_DMAAlloc** function if you want to allocate the buffer(s) dynamically outside your program's memory area.

3. *If you want to use a channel-gain queue to specify the channels acquiring data,* define and assign the appropriate values to the queue and note the starting address. Refer to page 2-11 for more information about channel-gain queues.

4. Use the appropriate setup functions to specify the attributes of the operation. The setup functions are listed in Table 3-6.

Note: When you access a new A/D frame, the frame elements contain default values. If the default value of a particular element is suitable for your operation, you do not have to use the setup function associated with that element. Refer to Table 3-1 on page 3-5 for a list of the default values of A/D frame elements.

Table 3-6. Setup Functions for DMA-Mode Analog Input Operations

Attribute	Setup Function(s)
Buffer[1]	K_SetDMABuf K_BufListAdd
Number of Samples	K_SetBuf K_SetBufI K_BufListAdd
Buffering Mode	K_SetContRun K_ClrContRun[2]
Start Channel	K_SetChn K_SetStartStopChn K_StartStopG
Stop Channel	K_SetStartStopChn K_SetStartStopG
Channel Configuration	K_SetADConfig
Input Range Type	K_SetADMode
Common-mode ground reference	K_SetADCommonMode
Gain	K_SetG K_SetStartStopG
Channel-Gain Queue	K_SetChnGAry
Conversion Mode	K_SetADFreeRun K_ClrADFreeRun[2]
SSH Mode	K_SetSSH
Clock Source	K_SetClk
Pacer Clock Rate[1]	K_SetClkRate
External Clock Edge	K_SetExtClkEdge
Burst Clock Rate	K_SetBurstTicks
Trigger Source	K_SetTrig
Trigger Type	K_SetADTrig K_SetDITrig

Programming with the Function Call Driver

Table 3-6. Setup Functions for DMA-Mode Analog Input Operations (cont.)

Attribute	Setup Function(s)
Trigger Channel	K_SetADTrig
Trigger Polarity	K_SetADTrig
Trigger Level	K_SetADTrig
Trigger Hysteresis	K_SetTrigHyst
About-Trigger Mode	K_SetAboutTrig K_ClrAboutTrig[2]
Hardware Gate	K_SetGate

Notes

[1] This element must be set.

[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame.

Refer to Chapter 2 for background information about the setup functions; refer to Chapter 4 for detailed descriptions of the setup functions.

5. Use the **K_DMAStart** function to start the DMA-mode operation.

6. Use the **K_DMAStatus** function to monitor the status of the DMA-mode operation.

7. *If you specified continuous buffering mode,* use the **K_DMAStop** function to stop the DMA-mode operation when the appropriate number of samples has been acquired.

8. *If you are programming in Visual Basic for Windows or BASIC and you used K_DMAAlloc to allocate your buffer(s),* use the **K_MoveBufToArray** function to transfer the acquired data from the allocated buffer to a local array that your program can use.

9. *If you used K_DMAAlloc to allocate your buffer(s),* use the **K_DMAFree** function to deallocate the buffer(s).

10. *If you used K_BufListAdd to specify a list of multiple buffers*, use the
    K_BufListReset function to clear the list.

11. Use the **K_FreeFrame** function to return the frame you accessed in
    step 1 to the pool of available frames.

Analog Output Operations (DAS-1800HC Series Only)

The following subsections describe the operation-specific programming
tasks required to perform single-mode and interrupt-mode analog output
operations.

Single Mode

For a single-mode analog output operation, perform the following tasks:

1. Declare the buffer or variable in which to store the single analog
   output value.

2. Use the **K_DAWrite** function to write the single analog output value;
   specify the attributes of the operation as arguments to the function.

Interrupt Mode

For an interrupt-mode analog output operation, perform the following
tasks:

1. Use the **K_GetDAFrame** function to access a D/A frame.

2. Allocate the buffer or dimension the array in which to store the data to
   be written. Use the **K_IntAlloc** function if you want to allocate the
   buffer dynamically outside your program's memory area.

3. Use the appropriate setup functions to specify the attributes of the
   operation. The setup functions are listed in Table 3-7.

Note: When you access a new D/A frame, the frame elements contain default values. f the default value of a particular element is suitable for your operation, you do not have to use the setup function associated with that element. Refer to Table 3-2 on page 3-7 for a list of the default values of D/A frame elements.

Table 3-7. Setup Functions for Interrupt-Mode Analog Output Operations

Attribute	Setup Function(s)
Buffer[1]	K_SetBuf K_SetBufI
Number of Samples	K_SetBuf K_SetBufI
Buffering Mode	K_SetContRun K_ClrContRun[2]
Start Channel	K_SetChn K_SetStartStopChn
Stop Channel	K_SetStartStopChn
Pacer Clock Rate[1]	K_SetClkRate

Notes
[1] This element must be set.
[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame.

Refer to Chapter 2 for background information about the setup functions; refer to Chapter 4 for detailed descriptions of the setup functions.

4. *If you are programming in Visual Basic for Windows or BASIC and you used **K_IntAlloc** to allocate your buffer,* use the **K_MoveArrayToBuf** function to transfer the data from the local array to the dynamically allocated buffer that the driver can use.

5. Use the **K_IntStart** function to start the interrupt-mode operation.

6. Use the **K_IntStatus** function to monitor the status of the interrupt-mode operation.

7. *If you specified continuous buffering mode*, use the **K_IntStop** function to stop the interrupt-mode operation when the appropriate number of samples has been written.

8. *If you used K_IntAlloc to allocate your buffer*, use the **K_IntFree** function to deallocate the buffer.

9. Use the **K_FreeFrame** function to return the frame you accessed in step 1 to the pool of available frames.

Digital I/O Operations

The following subsections describe the operation-specific programming tasks required to perform single-mode and interrupt-mode digital I/O operations.

Single Mode

For a single-mode digital I/O operation, perform the following tasks:

1. Declare the buffer or variable in which to store the single digital I/O value.

2. Use one of the following digital I/O single-mode operation functions, specifying the attributes of the operation as arguments to the function:

Function	Purpose
K_DIRead	Reads a single digital input value.
K_DOWrite	Writes a single digital output value.

Interrupt Mode

For an interrupt-mode digital I/O operation, perform the following tasks:

1. Use the **K_GetDIFrame** function to access a DI frame; use the **K_GetDOFrame** function to access a DO frame.

2. Allocate the buffer or dimension the array in which to store the data to be read or written. Use the **K_IntAlloc** function if you want to allocate the buffer dynamically outside your program's memory area.

3. Use the appropriate setup functions to specify the attributes of the operation. The setup functions are listed in Table 3-8.

Note: When you access a new DI or DO frame, the frame elements contain default values. If the default value of a particular element is suitable for your operation, you do not have to use the setup function associated with that element. Refer to Table 3-3 on page 3-8 for a list of the default values of DI frame elements; refer to Table 3-4 on page 3-9 for a list of the default values of DO frame elements.

Table 3-8. Setup Functions for Interrupt-Mode Digital Input and Digital Output Operations

Attribute	Setup Function(s)
Buffer[1]	K_SetBuf K_SetBufI
Number of Samples	K_SetBuf K_SetBufI
Buffering Mode	K_SetContRun K_ClrContRun[2]
Pacer Clock Rate[1]	K_SetClkRate

Notes
[1] This element must be set.
[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame.

Refer to Chapter 2 for background information about the setup functions; refer to Chapter 4 for detailed descriptions of the setup functions.

4. *If you are performing a digital output operation, you are programming in Visual Basic for Windows or BASIC, and you used K_IntAlloc to allocate your buffer,* use the **K_MoveArrayToBuf** function to transfer the data from the local array to the dynamically allocated buffer that the driver can use.

5. Use the **K_IntStart** function to start the interrupt-mode operation.

6. Use the **K_IntStatus** function to monitor the status of the interrupt-mode operation.

7. *If you specified continuous buffering mode,* use the **K_IntStop** function to stop the interrupt-mode operation when the appropriate number of samples has been written.

8. *If you are performing a digital input operation, you are programming in Visual Basic for Windows or BASIC, and you used K_IntAlloc to allocate your buffer,* use the **K_MoveBufToArray** function to transfer the data from the allocated buffer to a local array that your program can use.

9. *If you used K_IntAlloc to allocate your buffer,* use the **K_IntFree** function to deallocate the buffer.

10. Use the **K_FreeFrame** function to return the frame you accessed in step 1 to the pool of available frames.

Language-Specific Programming Information

This section provides programming information for each of the supported languages. Note that the compilation procedures for all languages assume that the paths and/or environment variables are set correctly.

C/C++ Languages

The following sections contain information you need to allocate and assign memory buffers and to create channel-gain queues when programming in C or C++, as well as language-specific information for Microsoft C/C++, Borland C/C++, Microsoft QuickC for Windows, and Microsoft Visual C++.

Note: When programming in C/C++, proper typecasting may be required to avoid C/C++ type-mismatch warnings.

Allocating and Assigning Dynamically Allocated Memory Buffers

This section provides code fragments that describe how to allocate and assign dynamically allocated memory buffers when programming in C or C++. Refer to the example programs on disk for more information.

Notes: The code fragments for dynamically allocated memory assume that you are using DMA mode; the code for interrupt mode is identical, except that you use the appropriate interrupt-mode functions instead of the DMA-mode functions.

If you are programming in Windows' Enhanced mode, you may be limited in the amount of memory you can allocate. It is recommended that you install the Keithley Memory Manager before you begin programming to ensure that you can allocate a large enough buffer or buffers. Refer to your DAS-1800 Series board user's guide for more information about the Keithley Memory Manager.

Single Memory Buffer

You can use a single, dynamically allocated memory buffer for interrupt-mode analog input, analog output, and digital I/O operations and for DMA-mode analog input operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate a buffer of size Samples for the frame defined by hFrame and

how to use **K_SetDMABuf** to assign the starting address of the buffer; the buffer can store a maximum of 65,536 samples.

```
. . .
void far *AcqBuf;          //Declare pointer to buffer
WORD hMem;                 //Declare word for memory handle
. . .
wDasErr = K_DMAAlloc (hFrame, Samples, &AcqBuf, &hMem);
wDasErr = K_SetDMABuf (hFrame, AcqBuf, Samples);
. . .
```

The following code illustrates how to use **K_DMAFree** to later free the allocated buffer, using the memory handle stored by **K_DMAAlloc**.

```
    . . .
    wDasErr = K_DMAFree (hMem);
    . . .
```

Note: Make sure that you always check the returned value (wDasErr in the previous examples) for possible errors.

Multiple Memory Buffers

You can use multiple, dynamically allocated memory buffers for interrupt-mode analog input operations and for DMA-mode analog input operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate five buffers of size Samples each for the frame defined by hADFrame and how to use **K_BufListAdd** to assign the starting addresses of the five buffers; each buffer can store a maximum of 65,536 samples.

```
. . .
void far *AcqBuf[5];              //Declare 5 pointers to 5 buffers
WORD hMem[5];                     //Declare 5 words for 5 memory handles
. . .
for (i = 0; i < 5; i++)          {
wDasErr = K_DMAAlloc (hADFrame, Samples, &AcqBuf[i],&hMem[i]);
wDasErr = K_BufListAdd (hADFrame, AcqBuf[i], Samples);
}
. . .
```

The following code illustrates how to use **K_DMAFree** to later free the allocated buffers, using the memory handles stored by **K_DMAAlloc**; if you free the allocated buffers, you must also use **K_BufListReset** to reset the buffer list associated with the frame.

```
. . .
for (i = 0; i < 5; i++) {
    wDasErr = K_DMAFree (hMem[i]);
}
wDasErr = K_BufListReset (hADFrame);
. . .
```

Notes: Make sure that you always check the returned value (wDasErr in the previous examples) for possible errors.

Accessing the Data

You access the data stored in dynamically allocated buffers through C/C++ pointer indirection. For example, assume that you want to display the first 10 samples of the second buffer in the multiple-buffer operation described in the previous section (AcqBuf[1]). The following code fragment illustrates how to access and display the data.

```
. . .
int far *pData;               //Declare a pointer called pData
. . .
pData = (int far *) AcqBuf[1]; //Assign pData to 2nd buffer
for (i = 0; i < 10; i++)
   printf ("Sample #%d %X", i, *(pData+i));
. . .
```

Dimensioning and Assigning Local Arrays

This section provides code fragments that describe how to dimension and assign local arrays when programming in C or C++. Refer to the example programs on disk for more information.

Single Array

You can use a single, local array for interrupt-mode analog input, analog output, and digital I/O operations.

The following code fragment illustrates how to dimension an array of 10,000 samples for the frame defined by hFrame and how to use **K_SetBuf** to assign the starting address of the array. The maximum array size is 65,536.

```
. . .
int Data[10000];   //Dimension array of 10,000 samples
. . .
wDasErr = K_SetBuf (hFrame, Data, 10000);
. . .
```

Note: Make sure that you always check the returned value (wDasErr in the previous example) for possible errors.

Multiple Arrays

You can use multiple, local arrays for interrupt-mode analog input operations.

The following code fragment illustrates how to allocate two arrays of 32,000 samples each for the frame defined by hADFrame and how to use **K_BufListAdd** to assign the starting addresses of the arrays. The maximum array size is 65,536.

```
. . .
int Data1[32000];    //Allocate Array #1 of 32,000 samples
int Data2[32000];    //Allocate Array #2 of 32,000 samples
. . .
wDasErr = K_BufListAdd (hADFrame, Data1, 32000);
wDasErr = K_BufListAdd (hADFrame, Data2, 32000);
. . .
```

Note: Make sure that you always check the returned value (wDasErr in the previous example) for possible errors.

Programming with the Function Call Driver

Creating a Channel-Gain Queue

The DASDECL.H and DASDECL.HPP files define a special data type (GainChanTable) that you can use to declare your channel-gain queue. GainChanTable is defined as follows:

```
typedef struct GainChanTable
{
   WORD num_of_codes;
   struct{
      char Chan;
      char Gain;
   } GainChanAry[256];
} GainChanTable;
```

The following example illustrates how to create a channel-gain queue called MyChanGainQueue for a DAS-1802HC board by declaring and initializing a variable of type GainChanTable.

```
GainChanTable MyChanGainQueue =
   {8,          //Number of entries
   0, 0,        //Channel 0, gain of 1
   1, 1,        //Channel 1, gain of 2
   2, 2,        //Channel 2, gain of 4
   3, 3,        //Channel 3, gain of 8
   3, 0,        //Channel 3, gain of 1
   2, 1,        //Channel 2, gain of 2
   1, 2,        //Channel 1, gain of 4
   0, 3};       //Channel 0, gain of 8
```

After you create MyChanGainQueue, you must assign the starting address of MyChanGainQueue to the frame defined by hFrame, as follows:

```
wDasErr = K_SetChnGAry (hFrame, &MyChanGainQueue);
```

Note: Make sure that you always check the returned value (wDasErr in the previous example) for possible errors.

When you start the next analog input operation (using **K_IntStart** or **K_DMAStart**), channel 0 is sampled at a gain of 1, channel 1 is sampled at a gain of 2, channel 2 is sampled at a gain of 4, and so on.

Programming in Microsoft C/C++

To program in Microsoft C/C++, you need the following files; these files are provided in the ASO-1800 software package.

File	Description
DAS1800.LIB	Linkable driver.
DASRFACE.LIB	Linkable driver.
DASDECL.H	Include file when compiling in C (.c programs).
DAS1800.H	Include file when compiling in C (.c programs).
DASDECL.HPP	Include file when compiling in C++ (.cpp programs).
DAS1800.HPP	Include file when compiling in C++ (.cpp programs).
USE1800.OBJ	Linkable object.

To create an executable file in Microsoft C/C++, use the following compile and link statements. Note that *filename* indicates the name of your application program.

Type of Compile	Compile and Link Statements
C	CL /c *filename*.c LINK *filename*+use1800.obj,,,das1800+dasrface;
C++	CL /c *filename*.cpp LINK *filename*+use1800.obj,,,das1800+dasrface;

Refer to page 3-23 for information about allocating and assigning dynamically allocated memory buffers when programming in Microsoft C/C++. Refer to page 3-25 for information about dimensioning and assigning local arrays when programming in Microsoft C/C++. Refer to page 3-27 for information about creating a channel-gain queue when programming in Microsoft C/C++.

Programming with the Function Call Driver

Programming in Borland C/C++

To program in Borland C/C++, you need the following files; these files are provided in the ASO-1800 software package.

File	Description
DAS1800.LIB	Linkable driver.
DASRFACE.LIB	Linkable driver.
DASDECL.H	Include file when compiling in C (.c programs).
DAS1800.H	Include file when compiling in C (.c programs).
DASDECL.HPP	Include file when compiling in C++ (.cpp programs).
DAS1800.HPP	Include file when compiling in C++ (.cpp programs).
USE1800.OBJ	Linkable object.

To create an executable file in Borland C/C++, use the following compile and link statements. Note that *filename* indicates the name of your application program.

Type of Compile	Compile and Link Statements[1]
C	BCC -ml *filename*.c use1800.obj das1800.lib dasrface.lib
C++	BCC -ml *filename*.cpp use1800.obj das1800.lib dasrface.lib

Notes

[1] These statements assume a large memory model; however, any memory model is acceptable.

Programming in Microsoft QuickC for Windows

To program in Microsoft QuickC for Windows, you need the following files; these files are provided in the ASO-1800 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library.
DASSUPRT.DLL	Dynamic Link Library.
DAS1800.DLL	Dynamic Link Library.
DASDECL.H	Include file.
DAS1800.H	Include file.
DASIMP.LIB	DAS Shell Imports
D1800IMP.LIB	DAS-1800 Imports

To create an executable file in Microsoft QuickC for Windows, perform the following steps:

1. Load *filename*.c into the QuickC for Windows environment, where *filename* indicates the name of your application program.

2. Create a project file. The project file should contain all necessary files, including *filename*.c, *filename*.rc, *filename*.def, *filename*.h, DASIMP.LIB, and D1800IMP.LIB, where *filename* indicates the name of your application program.

3. From the Project menu, choose Build to create a stand-alone executable file (.EXE) that you can execute from within Windows.

Programming with the Function Call Driver

Programming in Microsoft Visual C++

To program in Microsoft Visual C++, you need the following files; these files are provided in the ASO-1800 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library.
DASSUPRT.DLL	Dynamic Link Library.
DAS1800.DLL	Dynamic Link Library.
DASDECL.H	Include file.
DAS1800.H	Include file.
DASIMP.LIB	DAS Shell Imports
D1800IMP.LIB	DAS-1800 Imports

To create an executable file in Visual C++, perform the following steps:

1. Create a project file by choosing New from the Project menu. The project file should contain all necessary files, including *filename*.c, *filename*.rc, *filename*.def, DASIMP.LIB, and D1800IMP.LIB, where *filename* indicates the name of your application program.

2. From the Project menu, choose Rebuild All FILENAME.EXE to create a stand-alone executable file (.EXE) that you can execute from within Windows.

Pascal Languages

The following sections contain information you need to allocate and assign memory buffers and to create channel-gain queues when programming in Pascal, as well as language-specific information for Borland Turbo Pascal (for DOS) and Borland Turbo Pascal for Windows.

Allocating and Assigning Dynamically Allocated Memory Buffers

This section provides code fragments that describe how to allocate and assign dynamically allocated memory buffers when programming in Pascal. Refer to the example programs on disk for more information.

Notes: The code fragments for dynamically allocated memory assume that you are using DMA mode; the code for interrupt mode is identical, except that you use the appropriate interrupt-mode functions instead of the DMA-mode functions.

If you are using Borland Turbo Pascal for Windows in Enhanced mode, you may be limited in the amount of memory you can allocate. It is recommended that you use the Keithley Memory Manager before you begin programming to ensure that you can allocate a large enough buffer or buffers. Refer to your DAS-1800 Series board user's guide for more information about the Keithley Memory Manager.

Reducing the Memory Heap

Note: Reducing the memory heap is recommended for Borland Turbo Pascal (for DOS) only; if you are programming in Borland Turbo Pascal for Windows, proceed to the next section.

By default, when Borland Turbo Pascal (for DOS) programs begin to run, Pascal reserves all available DOS memory for use by the internal memory manager; this allows you to perform **GetMem** and **FreeMem** operations. Pascal uses the compiler directive $M to distribute the available memory. The default configuration is {$m 16384, 0, 655360}, where 16384 bytes is the stack size, 0 bytes is the minimum heap size, and 655360 is the maximum heap size.

It is recommended that you use the compiler directive $M to reduce the maximum heap reserved by Pascal to zero bytes by entering the following:

```
{$m (16384, 0, 0)}
```

Reducing the maximum heap size to zero bytes makes all far heap memory available to DOS (and therefore available to the driver) and allows your application program to take maximum advantage of the **K_IntAlloc** and **K_DMAAlloc** functions. You can reserve some space for the internal memory manager or for DOS, if desired. Refer to your Borland Turbo Pascal (for DOS) documentation for more information.

Single Memory Buffer

You can use a single, dynamically allocated memory buffer for interrupt-mode analog input, analog output, and digital I/O operations and for DMA-mode analog input operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate a buffer of size Samples for the frame defined by hFrame and how to use **K_SetDMABuf** to assign the starting address of the buffer. The maximum array size is 65,536.

It is recommended that you declare a dummy type array of ^Integer. The dimension of this array is irrelevant; it is used only to satisfy Pascal's type-checking requirements.

```
($m (16384, 0, 0))          ( Turbo Pascal for DOS only )
. . .
Type
    IntArray = Array[0..1] of Integer;
. . .
Var
    AcqBuf : ^IntArray;    ( Declare buffer of dummy type )
    hMem : Word;     ( Declare word for memory handle, hMem )
. . .
wDasErr := K_DMAAlloc (hFrame, Samples, @AcqBuf, hMem);
wDasErr := K_SetDMABuf (hFrame, AcqBuf, Samples);
. . .
```

The following code illustrates how to use **K_DMAFree** to later free the allocated buffer, using the memory handle stored by **K_DMAAlloc**.

```
    . . .
    wDasErr := K_DMAFree (hMem);
    . . .
```

Note: Make sure that you always check the returned value (wDasErr in the previous examples) for possible errors.

Multiple Memory Buffers

You can use multiple, dynamically allocated memory buffers for interrupt-mode analog input operations and for DMA-mode analog input operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate five buffers of size Samples each for the frame defined by hADFrame and how to use **K_BufListAdd** to assign the starting addresses of the five buffers. The maximum array size is 65,536.

It is recommended that you declare a dummy type array of ^Integer. The dimension of this array is irrelevant; it is used only to satisfy Pascal's type-checking requirements.

```
{$m (16384, 0, 0)}              { Turbo Pascal for DOS only }
. . .
Type
IntArray = Array[0..1] of Integer;
. . .
Var
AcqBuf : Array[0..4] of ^IntArray;{5 buffers, dummy type}
hMem : Array[0..4] of Word; {5 words for 5 memory handles}
. . .
For i := 0 to 4 do begin
   wDasErr := K_DMAAlloc(hADFrame, Samples, @AcqBuf[i], hMem[i]);
   wDasErr := K_BufListAdd (hADFrame, AcqBuf[i], Samples);
End;
. . .
```

The following code illustrates how to use **K_DMAFree** to later free the allocated buffers, using the memory handles stored by **K_DMAAlloc**; if you free the allocated buffers, you must also use **K_BufListReset** to reset the buffer list associated with the frame.

```
. . .
For i := 0 to 4 do begin
   wDasErr := K_DMAFree (hMem[i]);
End;
```

Programming with the Function Call Driver

```
wDasErr := K_BufListReset (hADFrame);
. . .
```

Note: Make sure that you always check the returned value (wDasErr in the previous examples) for possible errors.

Accessing the Data

You access the data stored in dynamically allocated buffers through Pascal pointer indirection. For example, assume that you want to display the first 10 samples of the second buffer in the multiple-buffer operation described in the previous section (AcqBuf[1]). The following code fragment illustrates how to access and display the data.

```
. . .
for i := 0 to 10 do begin
    writeln ('Sample #', i,' =', AcqBuf[1]^[i]);
End;
. . .
```

Dimensioning and Assigning Local Arrays

This section provides code fragments that describe how to dimension and assign local arrays when programming in Pascal. Refer to the example programs on disk for more information.

Single Array

You can use a single, local array for interrupt-mode analog input, analog output, and digital I/O operations.

The following code fragment illustrates how to dimension an array of 10,000 samples for the frame defined by hFrame and how to use **K_SetBuf** to assign the starting address of the array; the array can store a maximum of 65,536 samples.

```
. . .
Data : Array[0..9999] of Integer;
. . .
wDasErr := K_SetBuf (hFrame, Data(0), 10000);
. . .
```

Note: Make sure that you always check the returned value (wDasErr in the previous example) for possible errors.

Multiple Arrays

You can use multiple, local arrays for interrupt-mode analog input operations.

The following code fragment illustrates how to allocate two arrays of 32,000 samples each for the frame defined by hADFrame and how to use **K_BufListAdd** to assign the starting addresses of the arrays; each array can store a maximum of 65,536 samples.

```
. . .
Data1 : Array[0..31999] of Integer; { Allocate Array #1 }
Data2 : Array[0..31999] of Integer; { Allocate Array #2 }
. . .
wDasErr := K_BufListAdd (hADFrame, Data1(0), 32000);
wDasErr := K_BufListAdd (hADFrame, Data2(0), 32000);
. . .
```

Note: Make sure that you always check the returned value (wDasErr in the previous example) for possible errors.

Creating a Channel-Gain Queue

The following example illustrates how to create a channel-gain queue
called MyChanGainQueue for a DAS-1802HC board by defining a
Record as a new type. You must use **K_SetChnGAry** to assign the
starting address of MyChanGainQueue to the frame defined by hFrame.

```
Type
    GainChanTable = Record;
    num_of_codes : Integer;
    queue : Array[0..255] of Byte;
    end;
. . .
Const
    MyChanGainQueue : GainChanTable = (
    num_of_codes : (8);        { Number of entries }
    queue :(0, 0,   { Channel 0, gain of 1 }
            1, 1,        { Channel 1, gain of 2 }
            2, 2,        { Channel 2, gain of 4 }
            3, 3,        { Channel 3, gain of 8 }
            3, 0,        { Channel 3, gain of 1 }
            2, 1,        { Channel 2, gain of 2 }
            1, 2,        { Channel 1, gain of 4 }
            0, 3)        { Channel 0, gain of 8 }
        );
        . . .
wDasErr := K_SetChnGAry (hFrame, MyChanGainQueue.num_of_codes);
```

> **Note:** Make sure that you always check the returned value (wDasErr in
> the previous example) for possible errors.

When you start the next analog input operation (using **K_IntStart** or
K_DMAStart), channel 0 is sampled at a gain of 1, channel 1 is sampled
at a gain of 2, channel 2 is sampled at a gain of 4, and so on.

Programming in Borland Turbo Pascal (for DOS)

To program in Borland Turbo Pascal, you need the following files; these files are provided in the ASO-1800 software package.

File[1]	Description
D1800TP6.TPU	Turbo Pascal unit for Version 6.0.
D1800TP7.TPU	Turbo Pascal unit for Version 7.0.

Notes

[1] If you must create a new Turbo Pascal unit when compiling in Borland Turbo Pascal for versions higher than 7.0, refer to FILES.TXT for a list of the files to use.

To create an executable file in Borland Turbo Pascal, use the following compile and link statement:

```
TPC filename.pas
```

where *filename* indicates the name of your application program.

Refer to page 3-32 for information about allocating and assigning dynamically allocated memory buffers when programming in Borland Turbo Pascal. Refer to page 3-35 for information about dimensioning and assigning local arrays when programming in Borland Turbo Pascal. Refer to page 3-37 for information about creating a channel-gain queue when programming in Borland Turbo Pascal.

Programming in Borland Turbo Pascal for Windows

To program in Borland Turbo Pascal for Windows, you need the following files; these files are provided in the ASO-1800 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library.
DASSUPRT.DLL	Dynamic Link Library.
DAS1800.DLL	Dynamic Link Library.
DASDECL.INC	Include file.
DAS1800.INC	Include file.

To create an executable file in Borland Turbo Pascal for Windows, perform the following steps:

1. Load *filename*.pas into the Borland Turbo Pascal for Windows environment, where *filename* indicates the name of your application program.

2. From the Compile menu, choose Make.

Refer to page 3-32 for information about allocating and assigning dynamically allocated memory buffers when programming in Borland Turbo Pascal for Windows. Refer to page 3-35 for information about dimensioning and assigning local arrays when programming in Borland Turbo Pascal for Windows. Refer to page 3-37 for information about creating a channel-gain queue when programming in Borland Turbo Pascal for Windows.

Microsoft Visual Basic for Windows

The following sections contain information you need to allocate and assign memory buffers and to create channel-gain queues when programming in Microsoft Visual Basic for Windows, as well as language-specific information for Microsoft Visual Basic for Windows.

Allocating and Assigning Dynamically Allocated Memory Buffers

This section provides code fragments that describe how to allocate and assign dynamically allocated memory buffers when programming in Microsoft Visual Basic for Windows. Refer to the example programs on disk for more information.

Note: The code fragments for dynamically allocated memory assume that you are using DMA mode; the code for interrupt mode is identical, except that you use the appropriate interrupt-mode functions instead of the DMA-mode functions.

If you are using Windows Enhanced mode, you may be limited in the amount of memory you can allocate. It is recommended that you use the Keithley Memory Manager before you begin programming to ensure that you can allocate a large enough buffer or buffers. Refer to your DAS-1800 Series board user's guide for more information about the Keithley Memory Manager.

Single Memory Buffer

You can use a single, dynamically allocated memory buffer for interrupt-mode analog input, analog output, and digital I/O operations and for DMA-mode analog input operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate a buffer of size Samples for the frame defined by hFrame and how to use **K_SetDMABuf** to assign the starting address of the buffer; the buffer can store a maximum of 32,767 samples.

```
. . .
Global AcqBuf As Long     ' Declare pointer to buffer
Global hMem As Integer    ' Declare integer for memory handle
. . .
wDasErr = K_DMAAlloc (hFrame, Samples, AcqBuf, hMem)
wDasErr = K_SetDMABuf (hFrame, AcqBuf, Samples)
. . .
```

The following code illustrates how to use **K_DMAFree** to later free the allocated buffer, using the memory handle stored by **K_DMAAlloc**.

```
. . .
wDasErr = K_DMAFree (hMem)
. . .
```

Note: Make sure that you always check the returned value (wDasErr in the previous examples) for possible errors.

Multiple Memory Buffers

You can use multiple, dynamically allocated memory buffers for interrupt-mode analog input operations and for DMA-mode analog input operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate five buffers of size Samples each for the frame defined by hADFrame and how to use **K_BufListAdd** to assign the starting addresses of the five buffers; each buffer can store a maximum of 32,767 samples.

```
. . .
Global AcqBuf(5) As Long          ' Declare 5 pointers to 5 buffers
Global hMem(5) As Integer         ' Declare 5 memory handles
. . .
for i% = 0 to 4
    wDasErr = K_DMAAlloc (hFrame, Samples, AcqBuf(i%), hMem(i%))
    wDasErr = K_BufListAdd (hFrame, AcqBuf(i%), Samples)
next i%
. . .
```

The following code illustrates how to use **K_DMAFree** to later free the allocated buffers, using the memory handles stored by **K_DMAAlloc**; if you free the allocated buffers, you must also use **K_BufListReset** to reset the buffer list associated with the frame.

```
.  .  .
for i% = 0 to 4
    wDasErr = K_DMAFree (hMem(i%))
next i%
wDasErr = K_BufListReset (hADFrame)
.  .  .
```

Note: Make sure that you always check the returned value (wDasErr in the previous examples) for possible errors.

Accessing the Data

In Microsoft Visual Basic for Windows, you cannot directly access analog input samples stored in dynamically allocated memory buffers. You must use **K_MoveBufToArray** to move a subset of the data into a local buffer as required. The following code fragment illustrates how to move the first 100 samples of the second buffer in the multiple-buffer operation described in the previous section (AcqBuf(1)) to a local memory buffer.

```
.  .  .
Dim Buffer(1000) As Integer    ' Declare local memory buffer
.  .  .
wDasErr = K_MoveBufToArray (Buffer(0), AcqBuf(1), 100)
.  .  .
```

Dimensioning and Assigning Local Arrays

This section provides code fragments that describe how to dimension and assign local arrays when programming in Microsoft Visual Basic for Windows. Refer to the example programs on disk for more information.

Single Array

You can use a single, local array for interrupt-mode analog input, analog output, and digital I/O operations.

Programming with the Function Call Driver

The following code fragment illustrates how to dimension an array of
10,000 samples for the frame defined by hFrame and how to use
K_SetBufI to assign the starting address of the array; the local array can
store a maximum of 32,767 samples.

```
.  .  .
Global Data(10000) As Integer    ' Allocate array
.  .  .
wDasErr = K_SetBufI (hFrame, Data(0), 10000)
.  .  .
```

Notes: Make sure that you always check the returned value (wDasErr in
the previous example) for possible errors.

Multiple Arrays

You can use multiple, local arrays for interrupt-mode analog input
operations.

The following code fragment illustrates how to dimension two arrays of
32,000 samples each for the frame defined by hADFrame and how to use
K_BufListAdd to assign the starting addresses of the arrays; each local
array can store a maximum of 32,767 samples.

```
.  .  .
Global Data1(32000) As Integer    ' Allocate Array #1
Global Data2(32000) As Integer    ' Allocate Array #2
.  .  .
wDasErr = K_BufListAdd (hADFrame, Data1(0), 32000)
wDasErr = K_BufListAdd (hADFrame, Data2(0), 32000)
.  .  .
```

Notes: Make sure that you always check the returned value (wDasErr in
the previous example) for possible errors.

Creating a Channel-Gain Queue

Before you create your channel-gain queue, you must declare an array of integers to accommodate the required number of entries. To accommodate the maximum possible channel-gain queue (256 entries), declare an array of 513 integers ((256 x 2) + 1). Next, you must fill the array with the channel-gain information. After you create the channel-gain queue, use **K_FormatChnGAry** to reformat the channel-gain queue so that it can be used by the DAS-1800 Series Function Call Driver.

The following code fragment illustrates how to create a four-entry channel-gain queue called MyChanGainQueue for a DAS-1802HC board and how to use **K_SetChnGAry** to assign the starting address of MyChanGainQueue to the frame defined by hFrame.

```
. . .
Global MyChanGainQueue(513) As Integer 'Maximum # of entries
. . .
MyChanGainQueue(0) = 4      ' Number of channel-gain pairs
MyChanGainQueue(1) = 0      ' Channel 0
MyChanGainQueue(2) = 0      ' Gain of 1
MyChanGainQueue(3) = 1      ' Channel 1
MyChanGainQueue(4) = 1      ' Gain of 2
MyChanGainQueue(5) = 2      ' Channel 2
MyChanGainQueue(6) = 2      ' Gain of 4
MyChanGainQueue(7) = 2      ' Channel 2
MyChanGainQueue(8) = 3      ' Gain of 8
. . .
wDasErr = K_FormatChnGAry (MyChanGainQueue(0))
wDasErr = K_SetChnGAry (hFrame, MyChanGainQueue(0))
. . .
```

Once the channel-gain queue is formatted, your Visual Basic for Windows program can no longer read it. To read or modify the array after it has been formatted, you must use **K_RestoreChnGAry** as follows:

```
. . .
wDasErr = K_RestoreChnGAry (MyChanGainQueue(0))
. . .
```

Notes: Make sure that you always check the returned value (wDasErr in the previous example) for possible errors.

When you start the next analog input operation (using **K_IntStart** or **K_DMAStart**), channel 0 is sampled at a gain of 1, channel 1 is sampled at a gain of 2, channel 2 is sampled at a gain of 4, and so on.

Programming in Microsoft Visual Basic for Windows

To program in Microsoft Visual Basic for Windows, you need the following files; these files are provided in the ASO-1800 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library.
DASSUPRT.DLL	Dynamic Link Library.
DAS1800.DLL	Dynamic Link Library.
DASDECL.BAS	Include file; must be added to the Project List.
DAS1800.BAS	Include file; must be added to the Project List.

To create an executable file from the Microsoft Visual Basic for Windows environment, choose Make EXE File from the Run menu.

Refer to page 3-40 for information about allocating and assigning dynamically allocated memory buffers when programming in Microsoft Visual Basic for Windows. Refer to page 3-42 for information about dimensioning and assigning local arrays when programming in Microsoft Visual Basic for Windows. Refer to page 3-44 for information about creating a channel-gain queue when programming in Microsoft Visual Basic for Windows.

BASIC Languages

The following sections contain information you need to allocate and assign memory buffers and to create channel-gain queues when programming in BASIC, as well as language-specific information for Microsoft QuickBasic (Versions 4.0 and 4.5), Microsoft Professional Basic (Version 7.0), and Microsoft Visual Basic for DOS.

Allocating and Assigning Dynamically Allocated Memory Buffers

This section provides code fragments that describe how to allocate and assign dynamically allocated memory buffers when programming in BASIC. Refer to the example programs on disk for more information.

Note: The code fragments for dynamically allocated memory assume that you are using DMA mode; the code for interrupt mode is identical, except that you use the appropriate interrupt-mode functions instead of the DMA-mode functions.

Reducing the Memory Heap

By default, when BASIC programs run, all available memory is left for use by the internal memory manager. BASIC provides the SetMem function to distribute the available memory (the Far Heap). It is necessary to re-distribute the Far Heap if you want to use dynamically allocated buffers. It is recommended that you include the following code at the beginning of BASIC programs to free the Far Heap for the driver's use:

```
FarHeapSize& = SetMem(0)
NewFarHeapSize& = SetMem(-FarHeapSize&/2)
```

Single Memory Buffer

You can use a single, dynamically allocated memory buffer for interrupt-mode analog input, analog output, and digital I/O operations and for DMA-mode analog input operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate a buffer of size Samples for the frame defined by hFrame and

how to use **K_SetDMABuf** to assign the starting address of the buffer;
the buffer can store a maximum of 65,536 samples.

```
. . .
Dim AcqBuf As Long        ' Declare pointer to buffer
Dim hMem As Integer       ' Declare integer for memory handle
. . .
wDasErr = KDMAAlloc (hFrame, Samples, AcqBuf, hMem)
wDasErr = KSetDMABuf (hFrame, AcqBuf, Samples)
. . .
```

The following code illustrates how to use **K_DMAFree** to later free the
allocated buffer, using the memory handle stored by **K_DMAAlloc**.

```
. . .
wDasErr = KDMAFree (hMem)
. . .
```

Note: Make sure that you always check the returned value (wDasErr in
the previous examples) for possible errors.

Multiple Memory Buffers

You can use multiple, dynamically allocated memory buffers for
interrupt-mode analog input operations and for DMA-mode analog input
operations.

The following code fragment illustrates how to use **K_DMAAlloc** to
allocate five buffers of size Samples each for the frame defined by
hADFrame and how to use **K_BufListAdd** to assign the starting
addresses of the five buffers; each buffer can store a maximum of 32,767
samples.

```
. . .
Dim AcqBuf(5) As Long              ' Declare 5 pointers to 5 buffers
Dim hMem(5) As Integer             ' Declare 5 memory handles
. . .
for i% = 0 to 4
   wDasErr = KDMAAlloc (hFrame, Samples, AcqBuf(i%), hMem(i%))
   wDasErr = KBufListAdd (hFrame, AcqBuf(i%), Samples)
next i%
. . .
```

The following code illustrates how to use **K_DMAFree** to later free the allocated buffers, using the memory handles stored by **K_DMAAlloc**; if you free the allocated buffers, you must also use **K_BufListReset** to reset the buffer list associated with the frame.

```
.  .  .
for i% = 0 to 4
   wDasErr = K_DMAFree (hMem(i%))
next i%
wDasErr = K_BufListReset (hADFrame)
.  .  .
```

Note: Make sure that you always check the returned value (wDasErr in the previous examples) for possible errors.

Accessing the Data

In BASIC, you cannot directly access analog input samples stored in dynamically allocated memory buffers. You must use **K_MoveBufToArray** to move a subset of the data into a local buffer as required. The following code fragment illustrates how to move the first 100 samples of the second buffer in the multiple-buffer operation described in the previous section (AcqBuf(1)) to a local memory buffer.

```
.  .  .
Dim Buffer(1000) As Integer    ' Declare local memory buffer
.  .  .
wDasErr = K_MoveBufToArray (Buffer(0), AcqBuf(1), 100)
.  .  .
```

Dimensioning and Assigning Local Arrays

This section provides code fragments that describe how to dimension and assign local arrays when programming in BASIC. Refer to the example programs on disk for more information.

Single Array

You can use a single, local array for interrupt-mode analog input, analog output, and digital I/O operations.

The following code fragment illustrates how to dimension an array of 10,000 samples for the frame defined by hFrame and how to use **K_SetBufI** to assign the starting address of the array; the local array can store a maximum of 32,767 samples.

```
 .  .  .
Dim Data(10000) As Integer        ' Allocate array
 .  .  .
wDasErr = K_SetBufI (hFrame, Data(0), 10000)
 .  .  .
```

Notes: Make sure that you always check the returned value (wDasErr in the previous example) for possible errors.

Multiple Arrays

You can use multiple, local arrays for interrupt-mode analog input operations.

The following code fragment illustrates how to dimension two arrays of 32,000 samples each for the frame defined by hADFrame and how to use **K_BufListAdd** to assign the starting addresses of the arrays; each local array can store a maximum of 32,767 samples.

```
 .  .  .
Dim Data1(32000) As Integer        ' Allocate Array #1
Dim Data2(32000) As Integer        ' Allocate Array #2
 .  .  .
wDasErr = KBufListAdd (hADFrame, Data1(0), 32000)
wDasErr = KBufListAdd (hADFrame, Data2(0), 32000)
 .  .  .
```

Notes: Make sure that you always check the returned value (wDasErr in the previous example) for possible errors.

Creating a Channel-Gain Queue

Before you create your channel-gain queue, you must declare an array of integers to accommodate the required number of entries. To accommodate the maximum possible channel-gain queue (256 entries), declare an array of 513 integers ((256 x 2) + 1). Next, you must fill the array with the channel-gain information. After you create the channel-gain queue, use **K_FormatChnGAry** to reformat the channel-gain queue so that it can be used by the DAS-1800 Series Function Call Driver.

The following code fragment illustrates how to create a four-entry channel-gain queue called MyChanGainQueue for a DAS-1802HC board and how to use **K_SetChnGAry** to assign the starting address of MyChanGainQueue to the frame defined by hFrame.

```
. . .
Dim MyChanGainQueue(513) As Integer 'Maximum # of entries
. . .
MyChanGainQueue(0) = 4     ' Number of channel-gain pairs
MyChanGainQueue(1) = 0     ' Channel 0
MyChanGainQueue(2) = 0     ' Gain of 1
MyChanGainQueue(3) = 1     ' Channel 1
MyChanGainQueue(4) = 1     ' Gain of 2
MyChanGainQueue(5) = 2     ' Channel 2
MyChanGainQueue(6) = 2     ' Gain of 4
MyChanGainQueue(7) = 2     ' Channel 2
MyChanGainQueue(8) = 3     ' Gain of 8
. . .
wDasErr = KFormatChnGAry (MyChanGainQueue(0))
wDasErr = KSetChnGAry (hFrame, MyChanGainQueue(0))
. . .
```

Once the channel-gain queue is formatted, your BASIC program can no longer read it. To read or modify the array after it has been formatted, you must use **K_RestoreChnGAry** as follows:

```
. . .
wDasErr = KRestoreChnGAry (MyChanGainQueue(0))
. . .
```

Notes: Make sure that you always check the returned value (wDasErr in the previous examples) for possible errors.

When you start the next analog input operation (using **K_IntStart** or **K_DMAStart**), channel 0 is sampled at a gain of 1, channel 1 is sampled at a gain of 2, channel 2 is sampled at a gain of 4, and so on.

Programming in Microsoft QuickBasic (Version 4.0)

To program in Microsoft QuickBasic (Version 4.0), you need the following files; these files are provided in the DAS-1800 Series standard software package.

File	Description
D1800Q40.LIB	Linkable driver for QuickBasic, Version 4.0, stand-alone, executable (.EXE) programs.
D1800Q40.QLB	Command-line loadable driver for the QuickBasic, Version 4.0, integrated environment.
QB4DECL.BI	Include file.
DASDECL.BI	Include file.
DAS1800.BI	Include file.

For Microsoft QuickBasic (Version 4.0), you can create an executable file from within the programming environment, or you can use a compile and link statement.

To create an executable file from within the programming environment, perform the following steps:

1. Enter the following to invoke the environment:

    ```
    QB /L D1800Q40 filename.bas
    ```

    where *filename* indicates the name of your application program.

2. From the Run menu, choose Make EXE File.

To use a compile and link statement, enter the following:

```
BC filename.bas /O
Link filename.obj,,,D1800Q40.lib+BCOM40.lib;
```

where *filename* indicates the name of your application program.

Refer to page 3-48 for information about dimensioning and assigning local arrays when programming in Microsoft QuickBasic (Version 4.0). Refer to page 3-48 for information about creating a channel-gain queue when programming in Microsoft QuickBasic (Version 4.0).

Programming in Microsoft QuickBasic (Version 4.5)

To program in Microsoft QuickBasic (Version 4.5), you need the following files; these files are provided in the DAS-1800 Series standard software package.

File	Description
D1800Q45.LIB	Linkable driver for QuickBasic, Version 4.5, stand-alone, executable (.EXE) programs.
D1800Q45.QLB	Command-line loadable driver for the QuickBasic, Version 4.5, integrated environment.
QB4DECL.BI	Include file.
DASDECL.BI	Include file.
DAS1800.BI	Include file.

For Microsoft QuickBasic (Version 4.5), you can create an executable file from within the programming environment, or you can use a compile and link statement.

Programming with the Function Call Driver

To create an executable file from within the programming environment, perform the following steps:

1. Enter the following to invoke the environment:

```
QB /L D1800Q45 filename.bas
```

where *filename* indicates the name of your application program.

2. From the Run menu, choose Make EXE File.

To use a compile and link statement, enter the following:

```
BC filename.bas /O
Link filename.obj,,,D1800Q45.lib+BCOM45.lib;
```

where *filename* indicates the name of your application program.

Refer to page 3-48 for information about dimensioning and assigning local arrays when programming in Microsoft QuickBasic (Version 4.5). Refer to page 3-50 for information about creating a channel-gain queue when programming in Microsoft QuickBasic (Version 4.5).

Programming in Microsoft Professional Basic (Version 7.0)

To program in Microsoft Professional Basic (Version 7.0), you need the following files; these files are provided in the DAS-1800 Series standard software package.

File	Description
D1800QBX.LIB	Linkable driver for Professional Basic, Version 7.0, stand-alone, executable (.EXE) programs.
D1800QBX.QLB	Command-line loadable driver for the Professional Basic, Version 7.0, integrated environment.
DASDECL.BI	Include file.
DAS1800.BI	Include file.

For Microsoft Professional Basic (Version 7.0), you can create an executable file from within the programming environment, or you can use a compile and link statement.

To create an executable file from within the programming environment, perform the following steps:

1. Enter the following to invoke the environment:

```
QBX /L D1800QBX filename.bas
```

   where *filename* indicates the name of your application program.

2. From the Run menu, choose Make EXE File.

To use a compile and link statement, enter the following:

```
BC filename.bas /o;
Link filename.obj,,,D1800QBX.lib;
```

where *filename* indicates the name of your application program.

Refer to page 3-50 for information about dimensioning and assigning local arrays when programming in Microsoft Professional Basic (Version 7.0). Refer to page 3-50 for information about creating a channel-gain queue when programming in Microsoft Professional Basic (Version 7.0).

Programming in Microsoft Visual Basic for DOS

To program in Microsoft Visual Basic for DOS, you need the following files; these files are provided in the DAS-1800 Series standard software package.

File	Description
D1800VBD.LIB	Linkable driver for Visual Basic for DOS stand-alone, executable (.EXE) programs.
D1800VBD.QLB	Command-line loadable driver for the Visual Basic for DOS integrated environment.
DASDECL.BI	Include file.
DAS1800.BI	Include file.

To create an executable file in Microsoft Visual Basic for DOS, perform the following steps:

1. Invoke the Visual Basic for DOS environment by entering the following:

   ```
   VBDOS /L D1800VBD.QLB filename.BAS
   ```

   where *filename* indicates the name of your application program.

2. From the Run menu, choose Make EXE File.

Refer to page 3-50 for information about dimensioning and assigning local arrays when programming in Microsoft Visual Basic for DOS. Refer to page 3-50 for information about creating a channel-gain queue when programming in Microsoft Visual Basic for DOS.

4

Function Reference

The FCD functions are organized into the following groups:

- Initialization functions

- Operation functions

- Frame management functions

- Memory management functions

- Buffer address functions

- Buffering mode functions

- Conversion mode functions

- Channel and gain functions

- Clock functions

- Trigger functions

- Gate functions

- Miscellaneous functions

The particular functions associated with each function group are presented in Table 4-1. The remainder of the chapter presents detailed descriptions of all the FCD functions, arranged in alphabetical order.

Table 4-1. Functions

Function Type	Function Name	Page Number
Initialization	DAS1800_DevOpen	page 4-8
	K_OpenDriver	page 4-171
	K_CloseDriver	page 4-25
	DAS1800_GetDevHandle	page 4-11
	K_GetDevHandle	page 4-105
	K_FreeDevHandle	page 4-61
	K_DASDevInit	page 4-33
Operation	K_ADRead	page 4-14
	K_DAWrite	page 4-35
	K_DIRead	page 4-38
	K_DOWrite	page 4-56
	K_DMAStart	page 4-47
	K_DMAStatus	page 4-49
	K_DMAStop	page 4-53
	K_IntStart	page 4-156
	K_IntStatus	page 4-158
	K_IntStop	page 4-162
Frame Management	K_GetADFrame	page 4-71
	K_GetDAFrame	page 4-102
	K_GetDIFrame	page 4-107
	K_GetDOFrame	page 4-116
	K_FreeFrame	page 4-63
	K_ClearFrame	page 4-23

Table 4-1. Functions (cont.)

Function Type	Function Name	Page Number
Memory Management	K_DMAAlloc	page 4-41
	K_DMAFree	page 4-45
	K_IntAlloc	page 4-151
	K_IntFree	page 4-154
	KMakeDMABuf	page 4-165
	K_MoveArrayToBuf	page 4-167
	K_MoveBufToArray	page 4-169
Buffer Address	K_SetBuf	page 4-191
	K_SetBufI	page 4-194
	K_GetBuf	page 4-82
	K_SetDMABuf	page 4-215
	K_BufListAdd	page 4-17
	K_BufListReset	page 4-21
Buffering Mode	K_ClrContRun	page 4-31
	K_SetContRun	page 4-210
	K_GetContRun	page 4-99
Conversion Mode	K_SetADFreeRun	page 4-183
	K_ClrADFreeRun	page 4-29
	K_GetADFreeRun	page 4-73
	K_GetSSH	page 4-132
	K_SetSSH	page 4-224

Table 4-1. Functions (cont.)

Function Type	Function Name	Page Number
Channel and Gain	K_SetChn	page 4-198
	K_SetStartStopChn	page 4-226
	K_SetG	page 4-220
	K_SetStartStopG	page 4-230
	K_SetChnGAry	page 4-201
	K_FormatChnGAry	page 4-59
	K_RestoreChnGAry	page 4-174
	K_GetChn	page 4-88
	K_GetStartStopChn	page 4-135
	K_GetG	page 4-124
	K_GetStartStopG	page 4-138
	K_GetChnGAry	page 4-91
	K_SetADCommonMode	page 4-179
	K_SetADConfig	page 4-181
	K_SetADMode	page 4-185
	K_GetADCommonMode	page 4-67
	K_GetADConfig	page 4-69
	K_GetADMode	page 4-76
Clock	K_SetClk	page 4-204
	K_SetClkRate	page 4-207
	K_GetClk	page 4-93
	K_GetClkRate	page 4-96
	K_SetBurstTicks	page 4-196
	K_GetBurstTicks	page 4-85
	K_SetExtClkEdge	page 4-218
	K_GetExtClkEdge	page 4-121

Table 4-1. Functions (cont.)

Function Type	Function Name	Page Number
Trigger	K_SetTrig	page 4-233
	K_SetADTrig	page 4-187
	K_SetTrigHyst	page 4-236
	K_SetDITrig	page 4-212
	K_SetAboutTrig	page 4-176
	K_ClrAboutTrig	page 4-27
	K_GetTrig	page 4-142
	K_GetADTrig	page 4-78
	K_GetTrigHyst	page 4-145
	K_GetDITrig	page 4-110
	K_GetAboutTrig	page 4-65
Gate	K_SetGate	page 4-222
	K_GetGate	page 4-126
Miscellaneous	K_GetErrMsg	page 4-119
	K_GetVer	page 4-148
	K_GetShellVer	page 4-129
	K_GetDOCurVal	page 4-113

Keep the following conventions in mind throughout this chapter:

- Under "Boards Supported," *All* refers to the following boards: DAS-1801HC, DAS-1802HC, DAS-1801ST, DAS-1802ST, DAS-1802HR.

- Although the function names are shown with underscores, do not use the underscores in the BASIC languages.

- The data types DDH, FRAMEH, DWORD, WORD, and BYTE are defined in the language-specific include files.

- Variable names are shown in italics.

- The return value for all DAS-1800 Series FCD functions is the error/status code. Refer to Appendix A for more information.

- The description shows the prototype for the function.

- In the examples, the variables are not defined. It is assumed that they are defined as shown in the syntax.

The name of each function argument in the Description and Usage sections includes a prefix that indicates the associated data type. These prefixes are described in Table 4-2.

Table 4-2. Data Type Prefixes

Prefix	Data Type	Comments
sz	Pointer to string terminated by zero	This type is typically used for variables that specify the driver's configuration file name, for example, "DAS1800.CFG" in the call to **DAS1800_DevOpen**.
h	Handle to device, frame, and memory block	Handle-type variables are declared in the user program as long or DWORD, depending on what the language allows. The actual user variable is passed to the driver by value.
ph	Pointer to a handle-type variable	This unique type is used when calling the driver functions to obtain a driver handle, a frame handle or a memory handle. The actual user variable is passed to the driver by reference.
p	Pointer to a variable	These are pointers to all types of variables, except handles (h). This type is typically used when passing a parameter of any type to the driver by reference.
n	A number value	This type is used when passing a number, typically a byte, to the driver by value.
w	A 16-bit word	This type is typically used when passing an unsigned integer to the driver by value.
a	Array	This type is usually used in conjunction with other prefixes listed here; for example, *anVar* denotes an array of numbers.
f	Float	Denotes a single-precision floating-point number.
d	Double	Denotes a double-precision floating-point number.
dw	A 32-bit double word	This type is typically used when passing an unsigned long to the driver by value.

DAS1800_DevOpen

Boards Supported	All
Purpose	Initializes the DAS-1800 Series Function Call Driver.

Prototype

C/C++
DASErr far pascal DAS1800_DevOpen (char far *szCfgFile, char far *pBoards);

Turbo Pascal
Function DAS1800_DevOpen (Var szCfgFile : char;
Var pBoards : Integer) : Word; far; external 'DAS1800';

Turbo Pascal for Windows
Function DAS1800_DevOpen (Var szCfgFile : char;
Var pBoards : Integer) : Word; far; external 'DAS1800';

Visual Basic for Windows
Declare Function DAS1800_DevOpen Lib "DAS1800.DLL"
(ByVal szCfgFile As String, pBoards As Integer) As Integer

BASIC
DECLARE FUNCTION DAS1800DEVOPEN% ALIAS
"DAS1800_DevOpen" (BYVAL szCfgFile AS LONG,
SEG pBoards AS INTEGER)

Parameters

	szCfgFile	Driver configuration file. Valid values: The name of a configuration file.
	pBoards	Number of boards defined in *szCfgFile*. Valid values: 1 to 3

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function initializes the driver according to the information in the
                     configuration file specified by *szCfgFile* and stores the number of boards
                     defined in *pBoards*.

                     You create a configuration file using the D1800CFG.EXE utility. Refer to
                     your DAS-1800 Series board user's guide for more information.

See Also   K_OpenDriver

Usage   **C/C++**
```
#include "DAS1800.H"    // Use "DAS1800.HPP for C++
...
int nBoards;
...
wDasErr = DAS1800_DevOpen ("DAS1802.CFG", &nBoards);
...
```

                     Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
szCfgName : String;
nBoards : Integer;
...
szCfgName := 'DAS1802.CFG' + #0;
wDasErr := DAS1800_DevOpen( szCfgName[1], nBoards );
```

                     Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
szCfgName : String;
nBoards : Integer;
...
szCfgName := 'DAS1802.CFG' + #0;
wDasErr := DAS1800_DevOpen( szCfgName[1], nBoards );
```

Visual Basic for Windows

(Include DAS1800.BAS in your program make file)

```
. . .
DIM nBoards AS INTEGER
DIM szCfgName AS STRING
. . .
wDasErr = DAS1800_DevOpen(szCfgName, nBoards)
```

BASIC

```
' $INCLUDE: 'DAS1800.BI'
. . .
DIM nBoards AS INTEGER
DIM szCfgName AS STRING
. . .
szCfgName = "DAS1802.CFG" + CHR$(0)
wDasErr = DAS1800DEVOPEN%(SSEGADD(szCfgName), nBoards)
. . .
```

Boards Supported	All

Purpose	Initializes a DAS-1800 Series board.

Prototype

C/C++
DASErr far pascal DAS1800_GetDevHandle (WORD *nBrdNum*,
DWORD far **phDev*);

Turbo Pascal
Function DAS1800_GetDevHandle (*nBrdNum* : Word;
Var *phDev* : Longint) : Word; far; external 'DAS1800';

Turbo Pascal for Windows
Function DAS1800_GetDevHandle (*nBrdNum* : Word;
Var *phDev* : Longint) : Word; far; external 'DAS1800';

Visual Basic for Windows
Declare Function DAS1800_GetDevHandle Lib "DAS1800.DLL"
(ByVal *nBrdNum* As Integer, *phDev* As Long) As Integer

BASIC
DECLARE FUNCTION DAS1800GETDEVHANDLE% ALIAS
"DAS1800_GetDevHandle" (BYVAL *nBrdNum* AS INTEGER,
SEG *phDev* AS LONG)

Parameters

nBrdNum	Board number. Valid values: **0** to **2**
phDev	Handle associated with the board.

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks
This function initializes the board specified by *nBrdNum*, and stores the board handle of the specified board in *phDev*.

The value stored in *phDev* is intended to be used exclusively as an argument to functions that require a board handle. Your program should not modify the value stored in *phDev*.

See Also
K_GetDevHandle

Usage
C/C++
```
#include "DAS1800.H"    // Use "DAS1800.HPP for C++
. . .
DWORD hDev;
. . .
wDasErr = DAS1800_GetDevHandle(0, &hDev);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
. . .
hDev : Longint;    { Device Handle }
. . .
wDasErr := DAS1800_GetDevHandle( 0, hDev );
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
. . .
hDev : Longint;    { Device Handle }
. . .
wDasErr := DAS1800_GetDevHandle( 0, hDev );
```

Visual Basic for Windows
(Include DAS1800.BAS in your program make file)

```
. . .
Global hDev As Long    ' Device Handle
. . .
wDasErr = DAS1800_GetDevHandle (0, hDev)
```

BASIC

```
' $INCLUDE: 'DAS1800.BI'
...
DIM hDev AS LONG    ' Device Handle
...
wDasErr = DAS1800GetDevHandle%(0, hDev)
```

K_ADRead

Boards Supported	All

Purpose   Reads a single analog input value.

Prototype

C/C++
DASErr far pascal K_ADRead (DWORD *hDev*, BYTE *nChan*,
BYTE *nGain*, void far *pData*);

Turbo Pascal
Function K_ADRead (*hDev* : Longint; *nChan* : Byte; *nGain* : Byte;
pData : Pointer) : Word;

Turbo Pascal for Windows
Function K_ADRead (*hDev* : Longint; *nChan* : Byte; *nGain* : Byte;
pData : Pointer) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_ADRead Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, ByVal *nChan* As Integer,
ByVal *nGain* As Integer, *pData* As Integer) As Integer

BASIC
DECLARE FUNCTION KADRead% ALIAS "K_ADRead"
(BYVAL *hDev* AS LONG, BYVAL *nChan* AS INTEGER,
BYVAL *nGain* AS INTEGER, SEG *pData* AS INTEGER)

Parameters   *hDev*   Handle associated with the board.

nChan	Analog input channel. Valid values:

	Valid channel numbers	
Board	**Differential**	**Single-ended**
DAS-1800HC	0 to 31	0 to 63
DAS-1800ST/HR without EXP-1800 expansion boards attached	0 to 7	0 to 15
DAS-1800ST/HR with *N* EXP-1800 expansion boards attached	Not applicable	0 to 15(*N* + 1)

nGain	Gain code.
	Valid values: **0** to **3** for DAS board channels
	0 to **7** for EXP-1800 channels
	Refer to Table 2-2 on page 2-10 for the gain and input ranges associated with each gain code.

pData	Acquired analog input value.

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

This function reads the analog input channel *nChan* on the board specified by *hDev* at the gain represented by *nGain*, and stores the raw count in *pData*.

Refer to Appendix B for information on converting the raw count stored in *pData* to voltage.

See Also

K_DMAStart, K_IntStart

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
int wADValue;
...
wDasErr = K_ADRead (hDev, 0, 0, &wADValue)
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wADValue : Integer;
...
wDasErr := K_ADRead (hDev, 0, 0, @wADValue);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wADValue : Integer;
...
wDasErr := K_ADRead (hDev, 0, 0, @wADValue);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wADValue As Integer
...
wDasErr = K_ADRead (hDev, 0, 0, wADValue)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM wADValue AS INTEGER
...
wDasErr = KADRead% (hDev, 0, 0, wADValue)
```

Boards
Supported

All

Purpose

Adds a buffer to the list of multiple buffers.

Prototype

C/C++

DASErr far pascal K_BufListAdd (DWORD *hFrame*, void far **pBuf*, DWORD *dwSamples*);

Turbo Pascal

Function K_BufListAdd (*hFrame* : Longint; *pBuf* : Pointer; *dwSamples* : Longint) : Word;

Turbo Pascal for Windows

Function K_BufListAdd (*hFrame* : Longint; *pBuf* : Pointer; *dwSamples* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_BufListAdd Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *pBuf* As Long, ByVal *dwSamples* As Long) As Integer

BASIC

DECLARE FUNCTION KBufListAdd% ALIAS "K_BufListAdd" (BYVAL *hFrame* AS LONG, SEG *pBuf* AS INTEGER, BYVAL *dwSamples* AS LONG)

Parameters

hFrame	Handle to the frame that defines the operation.
pBuf	Starting address of buffer.
dwSamples	Number of samples in the buffer.

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the operation defined by *hFrame*, this function adds the buffer at the
address pointed to by *pBuf* to the list of multiple buffers; the number of
samples in the buffer is specified in *dwSamples*. The driver supports
multiple buffers for analog input operations only.

Before you add the buffer to the multiple-buffer list, you must either
allocate the buffer dynamically (using **K_IntAlloc** or **K_DMAAlloc**), or
dimension the buffer locally.

Make sure that you add buffers to the multiple-buffer list in the order in
which you want to use them. The first buffer you add is Buffer 1, the
second buffer you add is Buffer 2, and so on. You can add up to 149
buffers. You can use **K_IntStatus** or **K_DMAStatus** to determine which
buffer is currently in use.

See Also   K_BufListReset, K_DMAAlloc, K_IntAlloc

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
void far *pBuf[5];   // Buffer pointers
WORD hMem[5];   // Buffer handles
...
for (i = 0; i < 5; i++) {
   wDasErr = K_DMAAlloc (hAD, dwSamples, &pBuf[i], &hMem[i]);
   wDasErr = K_BufListAdd (hAD, pBuf[i], dwSamples);
}
```

Turbo Pascal

```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : Array [0..4] of ^BufType;    { Buffer pointers }
hMem : Array [0..4] of Word;    { Buffer handles }
...
FOR I := 0 to 4 DO
   BEGIN
   wDasErr := K_DMAAlloc(hAD, dwSamples, Addr(pBuf[I]), hMem[I]);
   wDasErr := K_BufListAdd (hAD, pBuf[I], dwSamples);
   END;
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : Array [0..4] of ^BufType;    { Buffer pointers }
hMem : Array [0..4] of Word;    { Buffer handles }
...
FOR I := 0 to 4 DO
   BEGIN
   wDasErr := K_DMAAlloc(hAD, dwSamples, Addr(pBuf[I]), hMem[I]);
   wDasErr := K_BufListAdd (hAD, pBuf[I], dwSamples);
   END;
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global pBuf(5) As Long    ' Buffer pointers
Global hMem(5) As Integer    ' Buffer handles
...
For I% = 0 To 4
   wDasErr = K_DMAAlloc (hAD, dwSamples, pBuf(I%), hMem(I%))
   wDasErr = K_BufListAdd (hAD, pBuf(I%), dwSamples)
Next I%
```

K_BufListAdd (cont.)

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM pBuf(5) AS LONG    ' Buffer pointers
DIM hMem(5) AS INTEGER   ' Buffer handles
...
For I% = 0 To 4
   wDasErr = KDMAAlloc% (hAD, dwSamples, pBuf(I%), hMem(I%))
   wDasErr = KBufListAdd% (hAD, pBuf(I%), dwSamples)
Next I%
```

Boards Supported	All
Purpose	Clears the list of multiple buffers.
Prototype	**C/C++** DASErr far pascal K_BufListReset (DWORD *hFrame*);
	Turbo Pascal Function K_BufListReset (*hFrame* : Longint) : Word;
	Turbo Pascal for Windows Function K_BufListReset (*hFrame* : Longint) : Word; far; external 'DASSHELL';
	Visual Basic for Windows Declare Function K_BufListReset Lib "DASSHELL.DLL" (ByVal *hFrame* As Long) As Integer
	BASIC DECLARE FUNCTION KBufListReset% ALIAS "K_BufListReset" (BYVAL *hFrame* AS LONG)
Parameters	*hFrame* Handle to the frame that defines the operation.
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.
Remarks	For the operation defined by *hFrame*, this function clears all buffers from the list of multiple buffers. This function does not deallocate the buffers in the list. If dynamically allocated buffers are no longer needed, you can use **K_IntFree** or **K_DMAFree** to free the buffers before resetting the buffer list.

K_BufListReset (cont.)

See Also   K_DMAFree, K_IntFree, K_SetBuf, K_SetDMABuf

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_BufListReset (hAD);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_BufListReset (hAD);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_BufListReset (hAD);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_BufListReset (hAD)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KBufListReset% (hAD)
```

Boards Supported	All
Purpose	Sets the elements of a frame to their default values.

Prototype

C/C++
DASErr far pascal K_ClearFrame (DWORD *hFrame*);

Turbo Pascal
Function K_ClearFrame (*hFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_ClearFrame (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_ClearFrame Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KClearFrame% ALIAS "K_ClearFrame"
(BYVAL *hFrame* AS LONG)

Parameters   *hFrame*   Handle to the frame that defines the operation.

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function sets the elements of the frame specified by *hFrame* to their default values.

Refer to Table 3-1 on page 3-5 for the default values of the elements of an A/D frame, Table 3-2 on page 3-7 for the default values of the elements of an D/A frame, Table 3-3 on page 3-8 for the default values of the elements of an DI frame, and Table 3-4 on page 3-9 for the default values of the elements of an DO frame.

K_ClearFrame (cont.)

See Also   K_GetADFrame, K_GetDAFrame, K_GetDIFrame, and K_GetDOFrame

Usage   **C/C++**

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_ClearFrame (hAD);
```

Turbo Pascal

```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_ClearFrame (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_ClearFrame (hAD);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
wDasErr = K_ClearFrame (hAD)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KClearFrame% (hAD)
```

Boards
Supported
All

Purpose
Closes a previously initialized Keithley DAS Function Call Driver.

Prototype
C/C++
DASErr far pascal K_CloseDriver (DWORD *hDrv*);

Turbo Pascal
Not supported

Turbo Pascal for Windows
Function K_CloseDriver (*hDrv* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_CloseDriver Lib "DASSHELL.DLL"
(ByVal *hDrv* As Long) As Integer

BASIC
Not supported

Parameters
hDrv   Driver handle you want to free.

Return Value
This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks
This function frees the driver handle specified by *hDrv* and closes the associated use of the Function Call Driver. This function also frees all board handles and frame handles associated with *hDrv*.

If *hDrv* is the last driver handle specified for the Function Call Driver, the driver is shut down (for all languages) and unloaded (for Windows-based languages only).

K_CloseDriver (cont.)

See Also   K_FreeDevHandle

Usage   **C/C++**

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_CloseDriver (hDrv);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_CloseDriver (hDrv);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
wDasErr = K_CloseDriver (hDrv)
```

Boards Supported	All
Purpose	Disables the about trigger for an analog input operation.

Prototype

C/C++
DASErr far pascal K_ClrAboutTrig (DWORD *hFrame*);

Turbo Pascal
Function K_ClrAboutTrig (*hFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_ClrAboutTrig (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_ClrAboutTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KClrAboutTrig% ALIAS "K_ClrAboutTrig"
(BYVAL *hFrame* AS LONG)

Parameters	*hFrame*	Handle to the frame that defines the operation.

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks This function disables the about trigger for the operation defined by *hFrame*.

K_GetADFrame and **K_ClearFrame** also disables the about trigger.

See Also K_ClearFrame, K_GetADFrame, K_SetAboutTrig

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
. . .
wDasErr = K_ClrAboutTrig (hAD);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
. . .
wDasErr := K_ClrAboutTrig (hAD);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
. . .
wDasErr := K_ClrAboutTrig (hAD);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
. . .
wDasErr = K_ClrAboutTrig (hAD)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
. . .
wDasErr = KClrAboutTrig% (hAD)
```

**Boards
Supported**   All

Purpose   Sets paced conversion mode for an analog input operation.

Prototype   **C/C++**
DASErr far pascal K_ClrADFreeRun (DWORD *hFrame*);

Turbo Pascal
Function K_ClrADFreeRun (*hFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_ClrADFreeRun (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_ClrADFreeRun Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KClrADFreeRun% ALIAS "K_ClrADFreeRun"
(BYVAL *hFrame* AS LONG)

Parameters   *hFrame*   Handle to the frame that defines the operation.

Return Value   This function returns an integer error/status code. Error/status code 0
indicates that the function executed successfully. A non-zero error/status
code indicates that an error occurred. Refer to Appendix A for additional
information.

Remarks   This function sets the conversion mode for the operation defined by
hFrame to paced mode and sets the Conversion Mode element in the
frame accordingly.

K_GetADFrame and **K_ClearFrame** also enable paced conversion
mode.

K_ClrADFreeRun (cont.)

See Also   K_ClearFrame, K_GetADFrame, K_SetADFreeRun

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_ClrADFreeRun (hAD);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_ClrADFreeRun (hAD);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_ClrADFreeRun (hAD);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_ClrADFreeRun (hAD)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KClrADFreeRun% (hAD)
```

Boards Supported   All

Purpose   Sets single-cycle buffering mode.

Prototype

C/C++
DASErr far pascal K_ClrContRun (DWORD *hFrame*);

Turbo Pascal
Function K_ClrContRun (*hFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_ClrContRun (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_ClrContRun Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KClrContRun% ALIAS "K_ClrContRun"
(BYVAL *hFrame* AS LONG)

Parameters   *hFrame*   Handle to the frame that defines the operation.

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function sets the buffering mode for the operation defined by *hFrame* to single-cycle mode and sets the Buffering Mode element in the frame accordingly.

K_GetADFrame, K_GetDAFrame, K_GetDIFrame, K_GetDOFrame, and **K_ClearFrame** also enable single-cycle buffering mode.

K_ClrContRun (cont.)

Refer to page 2-18 for more information on buffering modes for analog input operations, page 2-30 for more information on buffering modes for analog output operations, and page 2-38 for more information on buffering modes for digital I/O operations.

See Also   K_SetContRun

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
. . .
wDasErr = K_ClrContRun (hAD);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
. . .
wDasErr := K_ClrContRun (hAD);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
. . .
wDasErr := K_ClrContRun (hAD);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
. . .
wDasErr = K_ClrContRun (hAD)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
. . .
wDasErr = KClrContRun% (hAD)
```

Boards Supported	All
Purpose	Reinitializes a board.
Prototype	**C/C++** DASErr far pascal K_DASDevInit (DWORD *hDev*); **Turbo Pascal** Function K_DASDevInit (*hDev* : Longint) : Longint; **Turbo Pascal for Windows** Function K_DASDevInit (*hDev* : Longint) : Longint; far; external 'DASSHELL'; **Visual Basic for Windows** Declare Function K_DASDevInit Lib "DASSHELL.DLL" (ByVal *hDev* As Long) As Integer **BASIC** DECLARE FUNCTION KDASDevInit% ALIAS "K_DASDevInit" (BYVAL *hDev* AS LONG)
Parameters	*hDev* Handle associated with the board.
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.
Remarks	This function stops all current operations and resets the board specified by *hDev* and the driver to their power-up states.
Usage	**C/C++**

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
. . .
wDasErr = K_DASDevInit (hDev);
```

Turbo Pascal

```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_DASDevInit (hDev);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_DASDevInit (hDev);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
wDasErr = K_DASDevInit (hDev)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KDASDevInit% (hDev)
```

Boards
Supported

DAS-1801HC, DAS-1802HC

Purpose

Writes a single analog output value.

Prototype

C/C++
DASErr far pascal K_DAWrite (DWORD *hDev*, BYTE *nChan*, DWORD *dwData*);

Turbo Pascal
Function K_DAWrite (*hDev* : Longint; *nChan* : Byte; *dwData* : Longint) : Word;

Turbo Pascal for Windows
Function K_DAWrite (*hDev* : Longint; *nChan* : Byte; *dwData* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_DAWrite Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, ByVal *nChan* As Integer,
ByVal *dwData* As Long) As Integer

BASIC
DECLARE FUNCTION KDAWrite% ALIAS "K_DAWrite"
(BYVAL *hDev* AS LONG, BYVAL *nChan* AS INTEGER,
BYVAL *dwData* AS LONG)

Parameters

hDev	Handle associated with the board.
nChan	Analog output channel. Valid values: **0** = Channel 0 **1** = Channel 1 **2** = Both channels
dwData	Analog output value. Valid values: **0** to **4,095**

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

This function writes the value *dwData* to the analog output channel specified by *nChan* on the board specified by *hDev*. Refer to page 2-26 for more information on analog output operations.

dwData is a 32-bit variable, but the output value must contain only 12 bits. Refer to Appendix B for a description of the data format.

See Also

K_IntStart

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
DWORD dwDAValue;
...
dwDAValue = (DWORD) (5.0 * 4096 / 20) + 2048;
wDasErr = K_DAWrite (hDev, 0, &dwDAValue);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
dwDAValue : Longint;
...
dwDAValue := Round((5.0 * 4096.0 / 20.0) + 2048);
wDasErr := K_DAWrite (hDev, 0, dwDAValue);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
dwDAValue : Longint;
...
dwDAValue := Round((5.0 * 4096.0 / 20.0) + 2048);
wDasErr := K_DAWrite (hDev, 0, dwDAValue);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
. . .
Global dwDAValue As Long
. . .
dwDAValue = INT(5.0 * 4096! / 20!) + 2048
wDasErr = K_DAWrite (hDev, 0, dwDAValue)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
. . .
DIM dwDAValue AS LONG
. . .
dwDAValue = INT(5.0 * 4096! / 20!) + 2048
wDasErr = KDAWrite% (hDev, 0, dwDAValue)
```

K_DIRead

Boards Supported	All
Purpose	Reads a single digital input value.

Prototype

C/C++
DASErr far pascal K_DIRead (DWORD *hDev*, BYTE *nChan*,
void far **pData*);

Turbo Pascal
Function K_DIRead (*hDev* : Longint; *nChan* : Byte;
pData : Pointer) : Word;

Turbo Pascal for Windows
Function K_DIRead (*hDev* : Longint; *nChan* : Byte;
pData : Pointer) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_DIRead Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, ByVal *nChan* As Integer, *pData* As Integer) As
Integer

BASIC
DECLARE FUNCTION KDIRead% ALIAS "K_DIRead"
(BYVAL *hDev* AS LONG, BYVAL *nChan* AS INTEGER,
SEG *pData* AS INTEGER)

Parameters

hDev	Handle associated with the board.
nChan	Digital input channel. Valid value: **0**
pData	Digital input value.

Return Value   This function returns an integer error/status code. Error/status code 0
indicates that the function executed successfully. A non-zero error/status
code indicates that an error occurred. Refer to Appendix A for additional
information.

Remarks

This function reads the values of all digital input lines on the board specified by *hDev*, and stores the value in *pData*.

pData is a 16-bit variable. The acquired digital value is stored in bits 0, 1, 2, and 3; the values in the remaining bits of *pData* are not defined. Refer to page 2-34 for more information.

See Also

K_IntStart

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
WORD wDIValue;
...
wDasErr = K_DIRead (hDev, 0, &wDIValue);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDIValue : Word;
...
wDasErr := K_DIRead (hDev, 0, @wDIValue);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDIValue : Word;
...
wDasErr := K_DIRead (hDev, 0, @wDIValue);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wDIValue As Integer
...
wDasErr = K_DIRead (hDev, 0, wDIValue);
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM wDIValue AS INTEGER
...
wDasErr = KDIRead% (hDev, 0, wDIValue)
```

Boards
Supported   All

Purpose   Allocates a buffer for a DMA-mode analog input operation.

Prototype   **C/C++**
DASErr far pascal K_DMAAlloc (DWORD *hFrame*,
DWORD *dwSamples*, void far * far *pBuf*, WORD far * *phMem*);

Turbo Pascal
Function K_DMAAlloc (*hFrame* : Longint; *dwSamples* : Longint;
pBuf : Pointer; Var *phMem* : Word) : Word;

Turbo Pascal for Windows
Function K_DMAAlloc (*hFrame* : Longint; *dwSamples* : Longint;
pBuf : Pointer; Var *phMem* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_DMAAlloc Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *dwSamples* As Long, *pBuf* As Long,
phMem As Integer) As Integer

BASIC
DECLARE FUNCTION KDMAAlloc% ALIAS "K_DMAAlloc"
(BYVAL *hFrame* AS LONG, BYVAL *dwSamples* AS LONG,
SEG *pBuf* AS LONG, SEG *phMem* AS INTEGER)

Parameters   *hFrame*   Handle to the frame that defines the operation.

                 dwSamples   Number of samples.
                                   Valid values: **1** to **32,767** for Visual Basic for
                                                  Windows and BASIC
                                                  1 to **65,536** for all other languages

                 pBuf   Starting address of the allocated buffer.

                 phMem   Handle associated with the allocated buffer.

Return Value
This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks
For the operation defined by *hFrame*, this function allocates a memory block (a buffer of the size *dwSamples*) from the available memory heap. On return, *pBuf* contains the far memory address of a buffer that is suitable for a DMA-mode analog input operation. Use K_SetDMABuf or K_BufListAdd to assign *pBuf* to an A/D frame. *phMem*, as returned by this function, is later used to free the allocated memory block by calling K_DMAFree.

Turbo Pascal and BASIC require that you re-distribute available memory before you dynamically allocate a buffer. Refer to "Reducing the Memory Heap" on page 3-32 (Turbo Pascal) or page 3-46 (BASIC) for additional information.

See Also
K_DMAFree, K_SetDMABuf, K_BufListAdd

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
void far *pBuf[5];    // Pointers to allocated DMA buffer
WORD hMem[5];    // Memory Handles to buffers
...
for (i = 0; i < 5; i++) {
   wDasErr = K_DMAAlloc (hAD, dwSamples, &pBuf[i], &hMem[i]);
   wDasErr = K_BufListAdd (hAD, pBuf[i], dwSamples);
}
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : Array [0..4] of ^BufType;    { DMA buffer pointers }
hMem : Array [0..4] of Word;    { Handles to DMA buffers }
...
FOR I := 0 to 4 DO
   BEGIN
   wDasErr := K_DMAAlloc(hAD, dwSamples, Addr(pBuf[I]), hMem[I]);
   wDasErr := K_BufListAdd (hAD, pBuf[I], dwSamples);
   END;
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : Array [0..4] of ^BufType;    { DMA buffer pointers }
hMem : Array [0..4] of Word;    { Handles to DMA buffers }
...
FOR I := 0 to 4 DO
   BEGIN
   wDasErr := K_DMAAlloc(hAD, dwSamples, Addr(pBuf[I]), hMem[I]);
   wDasErr := K_BufListAdd (hAD, pBuf[I], dwSamples);
   END;
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global pBuf(5) As Long
Global hMem(5) As Integer
...
For I% = 0 To 4
   wDasErr = K_DMAAlloc (hAD, dwSamples, pBuf(I%), hMem(I%))
   wDasErr = K_BufListAdd (hAD, pBuf(I%), dwSamples)
Next I%
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM pBuf(5) AS LONG
DIM hMem(5) AS INTEGER
...
For I% = 0 To 4
   wDasErr = KDMAAlloc% (hAD, dwSamples, pBuf(I%), hMem(I%))
   wDasErr = KBufListAdd% (hAD, pBuf(I%), dwSamples)
Next I%
```

Boards Supported   All

Purpose   Frees a buffer allocated for a DMA-mode analog input operation.

Prototype

C/C++
DASErr far pascal K_DMAFree (WORD *hMem*);

Turbo Pascal
Function K_DMAFree (*hMem* : Word) : Integer;

Turbo Pascal for Windows
Function K_DMAFree (*hMem* : Word) : Integer; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_DMAFree Lib "DASSHELL.DLL"
(ByVal *hMem* As Integer) As Integer

BASIC
DECLARE FUNCTION KDMAFree% ALIAS "K_DMAFree"
(BYVAL *hMem* AS INTEGER)

Parameters   *hMem*   Handle to DMA buffer.

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function frees the buffer specified by *hMem*; the buffer was previously allocated dynamically using **K_DMAAlloc**.

See Also   K_DMAlloc, K_SetDMABuf, K_BufListAdd

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_DMAFree (hMem);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_DMAFree (hMem);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_DMAFree (hMem);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_DMAFree (hMem)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KDMAFree% (hMem)
```

Boards Supported	All
Purpose	Starts a DMA-mode analog input operation.
Prototype	**C/C++** DASErr far pascal K_DMAStart (DWORD *hFrame*);
	Turbo Pascal Function K_DMAStart (*hFrame* : Longint) : Word;
	Turbo Pascal for Windows Function K_DMAStart (*hFrame* : Longint) : Word; far; external 'DASSHELL';
	Visual Basic for Windows Declare Function K_DMAStart Lib "DASSHELL.DLL" (ByVal *hFrame* As Long) As Integer
	BASIC DECLARE FUNCTION KDMAStart% ALIAS "K_DMAStart" (BYVAL *hFrame* AS LONG)
Parameters	*hFrame* Handle to the frame that defines the operation.
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.
Remarks	This function starts the DMA operation defined by *hFrame*. Refer to Chapter 3 for a discussion of the programming tasks associated with DMA operations.
See Also	K_DMAStatus, K_DMAStop

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_DMAStart (hAD);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_DMAStart (hAD);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_DMAStart (hAD);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
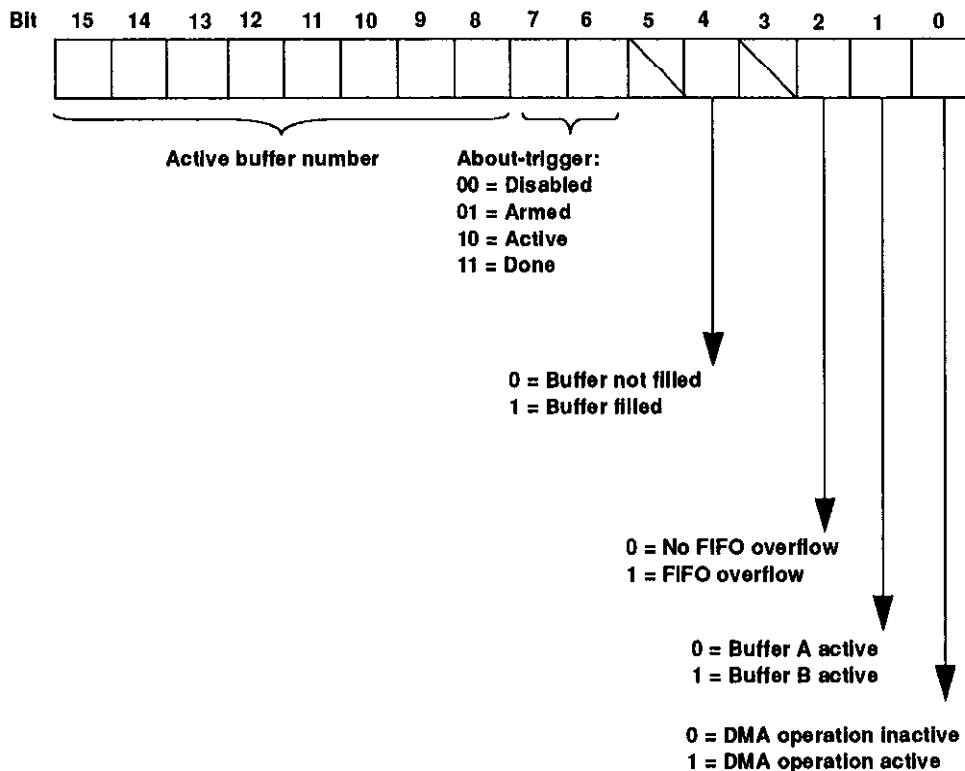```
...
wDasErr = K_DMAStart (hAD)
```

**BASIC**
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KDMAStart% (hAD)
```

| | |
|---|---|
| **Boards Supported** | All |
| **Purpose** | Gets status of a DMA-mode analog input operation. |
| **Prototype** | **C/C++**<br>DASErr far pascal K_DMAStatus (DWORD *hFrame*, short far *\*pStatus*, DWORD far *\*pCount*);<br><br>**Turbo Pascal**<br>Function K_DMAStatus (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word;<br><br>**Turbo Pascal for Windows**<br>Function K_DMAStatus (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word; far; external 'DASSHELL';<br><br>**Visual Basic for Windows**<br>Declare Function K_DMAStatus Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long) As Integer<br><br>**BASIC**<br>DECLARE FUNCTION KDMAStatus% ALIAS "K_DMAStatus" (BYVAL *hFrame* AS LONG, SEG *pStatus* AS INTEGER, SEG *pCount* AS LONG) |
| **Parameters** | *hFrame*        Handle to the frame that defines the operation. |
| | *pStatus*        Status of DMA-mode analog input operation; see **Remarks** below for value stored. |
| | *pCount*        Number of samples that were acquired into the current buffer.<br>Value stored: **0** to **65,536** |

# K_DMAStatus (cont.)

**Return Value**    This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

**Remarks**    For the DMA operation defined by *hFrame*, this function stores the status in *pStatus* and the number of samples acquired in *pCount*.

The value stored in *pStatus* depends on the settings in the Status word, as shown below:



Active buffer number

About-trigger:
00 = Disabled
01 = Armed
10 = Active
11 = Done

0 = Buffer not filled
1 = Buffer filled

0 = No FIFO overflow
1 = FIFO overflow

0 = Buffer A active
1 = Buffer B active

0 = DMA operation inactive
1 = DMA operation active

The bits are described as follows:

- Bit 0: Indicates whether a DMA-mode analog input operation is in progress.

- Bit 1: The Buffer A/B active bit. If you are using multiple buffers, this bit toggles each time acquisition sample storage is switched to a new buffer. If you are using a single buffer and the operation is in continuous mode, this bit toggles each time an acquisition sample is stored at the beginning of the buffer.

- Bit 2: When set, this bit indicates that the onboard FIFO has overflowed. This event automatically stops all conversions.

- Bit 3: Not used for DMA mode.

- Bit 4: This bit is used during continuous buffering mode; it is set when all data acquisition buffers that are currently assigned to the active operation have been filled with data at least once.

- Bit 5: Unassigned

- Bits 6-7: These bits indicate the state of the about trigger.

- Bits 8-15: In multiple-buffer acquisitions, these bits indicate the current active buffer number. The active buffer number is related to the Status word as follows:

$$\text{active buffer} = \frac{\text{Status word}}{256}$$

**See Also**        K_DMAStart, K_DMAStop

**Usage**

**C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
WORD wStatus;
DWORD dwCount;
...
wDasErr = K_DMAStatus (hAD, &wStatus, &dwCount);
```

**Turbo Pascal**
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_DMAStatus (hAD, wStatus, dwCount);
```

**Turbo Pascal for Windows**
```
{$I DASDECL.INC}
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_DMAStatus (hAD, wStatus, dwCount);
```

**Visual Basic for Windows**
*(Include DASDECL.BAS in your program make file)*
```
...
Global wStatus As Integer
Global dwCount As Long
...
wDasErr = K_DMAStatus (hAD, wStatus, dwCount)
```

**BASIC**
```
' $INCLUDE: 'DASDECL.BI'
...
DIM wStatus AS INTEGER
DIM dwCount AS LONG
...
wDasErr = KDMAStatus% (hAD, wStatus, dwCount)
```

**Boards
Supported**                    All

**Purpose**                    Stops a DMA-mode analog input operation.

**Prototype**                  **C/C++**
DASErr far pascal K_DMAStop (DWORD *hFrame*, short far *\*pStatus*,
DWORD far *\*pCount*);

**Turbo Pascal**
Function K_DMAStop (*hFrame* : Longint; Var *pStatus* : Word;
Var *pCount* : Longint) : Word;

**Turbo Pascal for Windows**
Function K_DMAStop (*hFrame* : Longint; Var *pStatus* : Word;
Var *pCount* : Longint) : Word; far; external 'DASSHELL';

**Visual Basic for Windows**
Declare Function K_DMAStop Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long) As
Integer

**BASIC**
DECLARE FUNCTION KDMAStop% ALIAS "K_DMAStop"
(BYVAL *hFrame* AS LONG, SEG *pStatus* AS INTEGER,
SEG *pCount* AS LONG)

**Parameters**     *hFrame*              Handle to the frame that defines the operation.

                   *pStatus*             Status of DMA-mode analog input operation.

                   *pCount*              Number of samples that were acquired into the
                                         current buffer.
                                         Value stored:  **0 to 65,536**

# K_DMAStop (cont.)

**Return Value**    This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

**Remarks**    This function stops the DMA operation defined by *hFrame* and stores the status of the DMA operation in *pStatus* and the number of samples acquired in *pCount*.

Refer to page 4-50 for the meaning of the value stored in *pStatus*.

If a DMA operation is not in progress, **K_DMAStop** is ignored.

**See Also**    K_DMAStart, K_DMAStatus

**Usage**    **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
WORD wStatus;
DWORD dwCount;
...
wDasErr = K_DMAStop (hAD, &wStatus, &dwCount);
```

**Turbo Pascal**
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_DMAStop (hAD, wStatus, dwCount);
```

**Turbo Pascal for Windows**
```
{$I DASDECL.INC}
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_DMAStop (hAD, wStatus, dwCount);
```

### Visual Basic for Windows

*(Include DASDECL.BAS in your program make file)*

```
...
Global wStatus As Integer
Global dwCount As Long
...
wDasErr = K_DMAStop (hAD, wStatus, dwCount)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM wStatus AS INTEGER
DIM dwCount AS LONG
...
wDasErr = KDMAStop% (hAD, wStatus, dwCount)
```

K_DOWrite

Boards Supported	All

Purpose　　Writes a single digital output value to the digital output channel.

Prototype

C/C++
DASErr far pascal K_DOWrite (DWORD *hDev*, BYTE *nChan*, DWORD *dwData*);

Turbo Pascal
Function K_DOWrite (*hDev* : Longint; *nChan* : Byte; *dwData* : Longint) : Word;

Turbo Pascal for Windows
Function K_DOWrite (*hDev* : Longint; *nChan* : Byte; *dwData* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_DOWrite Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, ByVal *nChan* As Integer,
ByVal *dwData* As Long) As Integer

BASIC
DECLARE FUNCTION KDOWrite% ALIAS "K_DOWrite"
(BYVAL *hDev* AS LONG, BYVAL *nChan* AS INTEGER,
BYVAL *dwData* AS LONG)

Parameters

hDev	Handle associated with the board.
nChan	Digital output channel. Valid value:　**0**
dwData	Digital output value. Valid values:　**0** to **255** for DAS-1800HC Series boards **0** to **15** for DAS-1800ST/HR Series boards

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function writes the value *dwData* to the digital output lines on the board specified by *hDev*.

dwData is a 32-bit variable. The value to be written is stored in bits 0 through 7 for DAS-1800HC Series boards or bits 0 through 3 for the DAS-1800ST/HR Series boards; the values in the remaining bits of *dwData* are not defined. Refer to page 2-35 for more information.

See Also   K_IntStart, K_GetDOCurVal

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
. . .
DWORD dwDOValue;
. . .
dwDOValue = 0x5;
wDasErr = K_DOWrite (hDO, 0, dwDOValue);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
. . .
dwDOValue : Longint
. . .
dwDOValue := $5;
wDasErr := K_DOWrite (hDO, 0, dwDOValue);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
. . .
dwDOValue : Longint
. . .
dwDOValue := $5;
wDasErr := K_DOWrite (hDO, 0, dwDOValue);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
. . .
Global dwDOValue As Long
. . .
dwDOValue = &H5
wDasErr = K_DOWrite (hDO, 0, dwDOValue)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
. . .
DIM dwDOValue AS LONG
. . .
dwDOValue = &H5
wDasErr = KDOWrite% (hDO, 0, dwDOValue)
```

Boards All
Supported

Purpose Converts the format of a channel-gain queue.

Prototype **C/C++**
Not supported

Turbo Pascal
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_FormatChnGAry Lib "DASSHELL.DLL"
(*pArray* As Integer) As Integer

BASIC
DECLARE FUNCTION KFormatChanGAry% ALIAS
"K_FormatChnGAry" (SEG *pArray* AS INTEGER)

Parameters *pArray* Channel-gain queue starting address.

Return Value This function returns an integer error/status code. Error/status code 0
indicates that the function executed successfully. A non-zero error/status
code indicates that an error occurred. Refer to Appendix A for additional
information.

Remarks This function converts a channel-gain queue created in BASIC or Visual
Basic for Windows using double-byte (16-bit) values to a channel-gain
queue of single-byte (8-bit) values that the **K_SetChnGAry** function can
use.

After you use this function, your program can no longer read the
converted list. You must use the **K_RestoreChnGAry** function to return
the list to its original format. Refer to page 4-174 for more information.

See Also   K_SetChnGAry, K_RestoreChnGAry

Usage

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
...
Global ChanGainArray(16) As Integer    ' Chan/Gain array
...
' Create the array of channel/gain pairs
ChanGainArray(0) = 2    ' # of chan/gain pairs
ChanGainArray(1) = 0: ChanGainArray(2) = 0
ChanGainArray(3) = 1: ChanGainArray(4) = 1
wDasErr = K_FormatChnGAry (ChanGainArray(0))
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM ChanGainArray(16) AS INTEGER   ' Chan/Gain array
...
' Create the array of channel/gain pairs
ChanGainArray(0) = 2    ' # of chan/gain pairs
ChanGainArray(1) = 0: ChanGainArray(2) = 0
ChanGainArray(3) = 1: ChanGainArray(4) = 1
wDasErr = KFormatChnGAry% (ChanGainArray(0))
```

Boards Supported	All
Purpose	Frees a previously specified board handle.
Prototype	**C/C++** DASErr far pascal K_FreeDevHandle (DWORD *hDev*);
	Turbo Pascal Not supported
	Turbo Pascal for Windows Function K_FreeDevHandle (*hDev* : Longint) : Word; far; external 'DASSHELL';
	Visual Basic for Windows Declare Function K_FreeDevHandle Lib "DASSHELL.DLL" (ByVal *hDev* As Long) As Integer
	BASIC Not supported
Parameters	*hDev* Board handle you want to free.
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.
Remarks	This function frees the board handle specified by *hDev* as well as all frame handles associated with *hDev*.
See Also	K_GetDevHandle

K_FreeDevHandle (cont.)

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_FreeDevHandle (hDev);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_FreeDevHandle (hDev);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_FreeDevHandle (hDev)
```

Boards Supported	All
Purpose	Frees a frame.

Prototype

C/C++
DASErr far pascal K_FreeFrame (DWORD *hFrame*);

Turbo Pascal
Function K_FreeFrame (*hFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_FreeFrame (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_FreeFrame Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KFreeFrame% ALIAS "K_FreeFrame"
(BYVAL *hFrame* AS LONG)

Parameters	*hFrame*	Handle to frame you want to free.

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks This function frees the frame specified by *hFrame*, making the frame available for another operation.

See Also K_GetADFrame, K_GetDAFrame, K_GetDIFrame, K_GetDOFrame

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_FreeFrame (hAD);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_FreeFrame (hAD);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_FreeFrame (hAD);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_FreeFrame (hAD)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KFreeFrame% (hAD)
```

Boards Supported	All

Purpose Gets the number of post-trigger samples as specified by **K_SetAboutTrig**.

Prototype
C/C++
DASErr far pascal K_GetAboutTrig (DWORD *hFrame*, DWORD far **pSamples*);

Turbo Pascal
Function K_GetAboutTrig (*hFrame* : Longint; Var *pSamples* : Longint) : Word;

Turbo Pascal for Windows
Function K_GetAboutTrig (*hFrame* : Longint; Var *pSamples* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetAboutTrig Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pSamples* As Long) As Integer

BASIC
DECLARE FUNCTION KGetAboutTrig% ALIAS "K_GetAboutTrig" (BYVAL *hFrame* AS LONG, SEG *pSamples* AS LONG)

Parameters

hFrame	Handle to the frame that defines the operation.
pSamples	Number of post-trigger samples.

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks For the operation specified by *hFrame*, this function stores the number of post-trigger samples to acquire in *pSamples*.

K_GetAboutTrig (cont.)

See Also   K_SetAboutTrig, K_ClrAboutTrig

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_GetAboutTrig (hAD, &dwSamples);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_GetAboutTrig (hAD, dwSamples);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_GetAboutTrig (hAD, dwSamples);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
...
wDasErr = K_GetAboutTrig (hAD, dwSamples)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KGetAboutTrig% (hAD, dwSamples)
```

Boards
Supported

DAS-1801ST, DAS-1802ST, DAS-1802HR

Purpose

Get a DAS board's A/D common-mode ground reference.

Prototype

C/C++
DASErr far pascal K_GetADCommonMode (DWORD *hDev*,
WORD far **pMode*);

Turbo Pascal
Function K_GetADCommonMode(*hDev* : Longint;
Var *pMode* : Word) : Word;

Turbo Pascal for Windows
Function K_GetADCommonMode (*hDev* : Longint;
Var *pMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetADCommonMode Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *pMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetADCommonMode% ALIAS
"K_GetADCommonMode" (BYVAL *hDev* AS LONG,
SEG *pMode* AS INTEGER)

Parameters

hDev	Handle to the frame that defines the operation.
pMode	A/D common-mode ground reference. Value stored: 0 for LL-GND
	1 for user-defined

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the board specified by *hDev*, this function stores the code that
indicates the A/D common-mode ground reference in *pMode*.

See Also   K_SetADCommonMode

Usage   **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_GetADCommonMode (hDev, &nADCommMode);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_GetADCommonMode (hDev, wADCommMode);
```

Turbo Pascal for Windows
```
($I DASDECL.INC}
...
...
wDasErr := K_GetADCommonMode (hDev, wADCommMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_GetADCommonMode (hDev, wADCommMode)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KGetADCommonMode% (hDev, wADCommMode)
```

Boards Supported
All

Purpose
Get a DAS board's A/D input channel configuration.

Prototype
C/C++
DASErr far pascal K_GetADConfig (DWORD *hDev*,
WORD far **pMode*);

Turbo Pascal
Function K_GetADConfig (*hDev* : Longint; Var *pMode* : Word) : Word;

Turbo Pascal for Windows
Function K_GetADConfig (*hDev* : Longint; Var *pMode* : Word) : Word;
far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetADConfig Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *pMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetADConfig% ALIAS "K_GetADConfig"
(BYVAL *hDev* AS LONG, SEG *pMode* AS INTEGER)

Parameters
hDev   Handle associated with the board.

pMode   A/D input channel configuration.
                              Value stored: **0** for Differential
                                             1 for Single-ended

Return Value
This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks
This function stores the code that indicates the A/D input channel configuration in *pMode* for the board specified by *hDev*.

K_GetADConfig (cont.)

See Also   K_SctADConfig

Usage   **C/C++**

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
DWORD hAD;
...
wDasErr = K_GetADConfig (hDev, &wADConfig);
```

Turbo Pascal

```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
hAD : Longint;
...
wDasErr := K_GetADConfig (hDev, wADConfig);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
hAD : Longint;
...
wDasErr := K_GetADConfig (hDev, wADConfig);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global hAD As Long
...
wDasErr = K_GetADConfig (hDev, wADCommMode)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM hAD AS LONG
...
wDasErr = KGetADConfig% (hDev, wADConfig)
```

Boards Supported   All

Purpose   Accesses an A/D frame for an analog input operation.

Prototype   .   **C/C++**
DASErr far pascal K_GetADFrame (DWORD *hDev*,
DWORD far * *pFrame*);

Turbo Pascal
Function K_GetADFrame (*hDev* : Longint;
Var *pFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_GetADFrame (*hDev* : Longint;
Var *pFrame* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetADFrame Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *pFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KGetADFrame% ALIAS "K_GetADFrame"
(BYVAL *hDev* AS LONG, SEG *pFrame* AS LONG)

Parameters   *hDev*   Handle associated with the board.

pFrame   Handle to the frame that defines the operation.

Remarks   This function specifies that you want to perform a DMA-mode or
interrupt-mode analog input operation on the board specified by *hDev*,
and accesses an available A/D frame with the handle *hFrame*. The frame
is initialized to its default settings; the default settings are given in Table
3-1 on page 3-5.

See Also   K_ClearFrame, K_FreeFrame

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
DWORD hAD;
...
wDasErr = K_GetADFrame (hDev, &hAD);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
hAD : Longint;
...
wDasErr := K_GetADFrame (hDev, hAD);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
hAD : Longint;
...
wDasErr := K_GetADFrame (hDev, hAD);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global hAD As Long
...
wDasErr = K_GetADFrame (hDev, hAD)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM hAD AS LONG
...
wDasErr = KGetADFrame% (hDev, hAD)
```

Boards Supported	All
Purpose	Gets the conversion mode.
Prototype	**C/C++** DASErr far pascal K_GetADFreeRun (DWORD *hFrame*, short far **pStatus*);

Turbo Pascal
Function K_GetADFreeRun (*hFrame* : Longint;
Var *pStatus* : Word) : Word;

Turbo Pascal for Windows
Function K_GetADFreeRun (*hFrame* : Longint;
Var *pStatus* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetADFreeRun Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pStatus* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetADFreeRun% ALIAS
"K_GetADFreeRun" (BYVAL *hFrame* AS LONG,
SEG *pStatus* AS INTEGER)

Parameters	*hFrame*	Handle to the frame that defines the operation.
	pStatus	Code that indicates the conversion mode. Value stored: **0** for Paced **0** for Burst

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the operation defined by *hFrame*, this function stores the code that indicates the conversion mode in *pStatus*.

The *pStatus* variable contains the value of the Conversion Mode element.

Refer to page 2-15 for information on conversion modes.

See Also   K_SetADFreeRun

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
WORD wMode;
...
wDasErr = K_GetADFreeRun (hAD, &wMode);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wMode : Word;
...
wDasErr := K_GetADFreeRun (hAD, wMode);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wMode : Word;
...
wDasErr := K_GetADFreeRun (hAD, wMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wMode As Integer
...
wDasErr = K_GetADFreeRun (hAD, wMode)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM wMode AS INTEGER
...
wDasErr = KGetADFreeRun% (hAD, wMode)
```

K_GetADMode

Boards Supported	All
Purpose	Get a DAS board's A/D input range type.
Prototype	**C/C++** DASErr far pascal K_GetADMode (DWORD *hDev*, WORD far **pMode*);
	Turbo Pascal Function K_GetADMode (*hDev* : Longint; Var *pMode* : Word) : Word;
	Turbo Pascal for Windows Function K_GetADMode (*hDev* : Longint; Var *pMode* : Word) : Word; far; external 'DASSHELL';
	Visual Basic for Windows Declare Function K_GetADMode Lib "DASSHELL.DLL" (ByVal *hDev* As Long, *pMode* As Integer) As Integer
	BASIC DECLARE FUNCTION KGetADMode% ALIAS "K_GetADMode" (BYVAL *hDev* AS LONG, SEG *pMode* AS INTEGER)
Parameters	*hDev* Handle associated with the board.

	pMode A/D input range type.
	                        Value stored: 0 for Bipolar
	                                        1 for Unipolar

Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.
Remarks	This function stores the code that indicates the A/D input range type for the board specified by *hDev* in *pMode*.

See Also   K_SetADMode

Usage   **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
DWORD hAD;
...
wDasErr = K_GetADMode (hDev, &nADMode);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
hAD : Longint;
...
wDasErr := K_GetADMode (hDev, wADMode);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
hAD : Longint;
...
wDasErr := K_GetADMode (hDev, wADMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global hAD As Long
...
wDasErr = K_GetADMode (hDev, wADMode)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM hAD AS LONG
...
wDasErr = KGetADMode% (hDev, wADMode)
```

K_GetADTrig

Boards Supported   All

Purpose   Gets the current analog trigger conditions.

Prototype   **C/C++**
DASErr far pascal K_GetADTrig (DWORD *hFrame*, short far **pOpt*,
short far **pChan*, DWORD far **pLevel*);

Turbo Pascal
Function K_GetADTrig (*hFrame* : Longint;Var *pOpt* : Word;
Var *pChan* : Word; Var *pLev* : Longint) : Word;

Turbo Pascal for Windows
Function K_GetADTrig (*hFrame* : Longint;Var *pOpt* : Word;
Var *pChan* : Word; Var *pLev* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetADTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pOpt* As Integer, *pChan* As Integer,
pLevel As Long) As Integer

BASIC
DECLARE FUNCTION KGetADTrig% ALIAS "K_GetADTrig"
(BYVAL *hFrame* AS LONG, SEG *pOpt* AS INTEGER,
SEG *pChan* AS INTEGER, SEG *pLevel* AS LONG)

Parameters   *hFrame*   Handle to the frame that defines the operation.

               pOpt   Analog trigger polarity.
                                     Value stored: **0** for Positive edge
                                                     2 for Negative edge

pChan   Analog input channel used as trigger channel. Valid values:

	Valid channel numbers	
Board	**Differential**	**Single-ended**
DAS-1800HC	0 to 31	0 to 63
DAS-1800ST/HR without EXP-1800 expansion boards attached	0 to 7	0 to 15
DAS-1800ST/HR with *N* EXP-1800 expansion boards attached	Not applicable	0 to 15(*N* + 1)

pLevel   Level at which the trigger event occurs.

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

For the operation defined by *hFrame*, this function stores the channel used for an analog trigger in *pChan*, the level used for the analog trigger in *pLevel*, and the trigger polarity in *pOpt*.

The *pOpt* variable contains the value of the Trigger Polarity element.

The *pChan* variable contains the value of the Trigger Channel element.

The *pLevel* variable contains the value of the Trigger Level element. The value of *pLevel* is represented in raw counts. Refer to Appendix B for information on converting the raw count stored in *pLevel* to voltage.

See Also

K_SetADTrig

K_GetADTrig (cont.)

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
int nOpt, nChan;
DWORD dwLevel;
...
wDasErr = K_GetADTrig (hAD, &nOpt, &nChan, &dwLevel);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
nOpt : Integer;
nChan : Integer;
dwLevel : Longint;
...
wDasErr := K_GetADTrig (hAD, nOpt, nChan, dwLevel);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
nOpt : Integer;
nChan : Integer;
dwLevel : Longint;
...
wDasErr := K_GetADTrig (hAD, nOpt, nChan, dwLevel);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global nOpt As Integer
Global nChan As Integer
Global dwLevel As Long
...
wDasErr = K_GetADTrig (hAD, nOpt, nChan, dwLevel)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM nOpt AS INTEGER
DIM nChan AS INTEGER
DIM dwLevel AS LONG
...
wDasErr = KGetADTrig% (hAD, nOpt, nChan, dwLevel)
```

K_GetBuf

Boards Supported	All

Purpose

Returns the address and size of a buffer assigned to a frame.

Prototype

C/C++
DASErr far pascal K_GetBuf (DWORD *hFrame*, void far * far **pBuf,
DWORD far **pSamples);

Turbo Pascal
Function K_GetBuf (*hFrame* : Longint; Var *pBuf* : Pointer;
Var *pSamples* : Longint) : Word;

Turbo Pascal for Windows
Function K_GetBuf (*hFrame* : Longint; Var *pBuf* : Pointer;
Var *pSamples* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetBuf Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pBuf* As Long, *pSamples* As Long) As Integer

BASIC
DECLARE FUNCTION KGetBuf% ALIAS "K_GetBuf"
(BYVAL *hFrame* AS LONG, SEG *pBuf* AS LONG,
SEG *pSamples* AS LONG)

Parameters

hFrame	Handle to the frame that defines the operation.
pBuf	Starting address of buffer.
pSamples	Number of samples. Value stored: **0 to 65,535**

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the operation specified by *hFrame*, this function stores either the address of the currently allocated buffer (if you are using a single buffer) or the address of the first buffer (if you are using multiple buffers) in *pBuf* and the number of samples stored in that buffer in *pSamples*.

Use this function to retrieve the address of the buffer whose address was specified by **K_SetBuf, K_SetBufI, or K_BufListAdd.**

The *pBuf* variable contains the value of the Buffer element.

The *pSamples* variable contains the value of the Number of Samples element.

See Also   K_BufListAdd, K_SetBuf, K_SetBufI

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
. . .
void far *pADBuffer;
DWORD dwSamples;
. . .
wDasErr = K_GetBuf (hAD, &pADBuffer, &dwSamples);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
. . .
pADBuffer : Longint;
dwSamples : Longint;
. . .
wDasErr = K_GetBuf (hAD, @pADBuffer, dwSamples);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
. . .
pADBuffer : Longint;
dwSamples : Longint;
. . .
wDasErr = K_GetBuf (hAD, @pADBuffer, dwSamples);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
. . .
Dim pADBuffer As Long
. . .
wDasErr = K_GetBuf (hAD, pADBuffer, dwSamples);
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
. . .
Dim pADBuffer As Long
. . .
wDasErr = K_GetBuf% (hAD, pADBuffer, dwSamples);
```

Boards Supported	All

Purpose   Gets the number of clock ticks between conversions to determine the burst mode conversion rate.

Prototype   **C/C++**
DASErr far pascal K_GetBurstTicks (DWORD *hFrame*,
short far **pTicks*);

Turbo Pascal
Function K_GetBurstTicks (*hFrame* : Longint;
Var *pTicks* : Word) : Word;

Turbo Pascal for Windows
Function K_GetBurstTicks (*hFrame* : Longint;
Var *pTicks* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetBurstTicks Lib "DASSHELL.DLL"
(ByVal *hFrame* as Long, *pTicks* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetBurstTicks% ALIAS "K_GetBurstTicks"
(BYVAL *hFrame* AS LONG, SEG *pTicks* AS INTEGER)

Parameters   *hFrame*   Handle to the frame that defines the operation.

                 pTicks   Number of clock ticks between conversions.
                                        Value stored: **3** to **255**

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

K_GetBurstTicks (cont.)

Remarks   For the operation defined by *hFrame*, this function stores the number of clock ticks between conversions of each channel in a scan in *pTicks*.

The *pTicks* variable contains the value of the Burst Clock Rate element.

See Also   K_SetBurstTicks

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
int nCount;
...
wDasErr = K_GetBurstTicks (hAD, &nCount);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
nCount : Integer;
...
wDasErr := K_GetBurstTicks (hAD, nCount);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
nCount : Integer;
...
wDasErr := K_GetBurstTicks (hAD, nCount);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global nCount As Integer
...
wDasErr = K_GetBurstTicks (hAD, nCount)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM nCount AS INTEGER
...
wDasErr = KGetBurstTicks% (hAD, nCount)
```

K_GetChn

**Boards
Supported**   All

Purpose   Gets a single channel number.

Prototype   **C/C++**
DASErr far pascal K_GetChn (DWORD *hFrame*, short far **pChan*);

Turbo Pascal
Function K_GetChn (*hFrame* : Longint; Var *pChan* : Word) : Word;

Turbo Pascal for Windows
Function K_GetChn (*hFrame* : Longint; Var *pChan* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetChn Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pChan* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetChn% ALIAS "K_GetChn"
(BYVAL *hFrame* AS LONG, SEG *pChan* AS INTEGER)

Parameters

hFrame   Handle to the frame that defines the operation.

pChan   Channel on which to perform the operation.
                            Valid values:

	Valid channel numbers	
Board	**Differential**	**Single-ended**
DAS-1800HC	0 to 31	0 to 63
DAS-1800ST/HR without EXP-1800 expansion boards attached	0 to 7	0 to 15
DAS-1800ST/HR with *N* EXP-1800 expansion boards attached	Not applicable	0 to 15(*N* + 1)

Return Value   This function returns an integer error/status code. Error/status code 0
                        indicates that the function executed successfully. A non-zero error/status
                        code indicates that an error occurred. Refer to Appendix A for additional
                        information.

Remarks   For the operation defined by *hFrame*, this function stores the channel
                        number in *pChan*.

                        The *pChan* variable contains the value of the Start Channel and Stop
                        Channel elements.

See Also   K_SetChn, K_SetStartStopChn, K_SetStartStopG

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
short nChan;
...
wDasErr = K_GetChn (hAD, &nChan);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
nChan : Integer;
...
wDasErr := K_GetChn (hAD, nChan);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
nChan : Integer;
...
wDasErr := K_GetChn (hAD, nChan);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global nChan AS Integer
...
wDasErr = K_GetChn (hAD, nChan)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM nChan AS INTEGER
...
wDasErr = KGetChn% (hAD, nChan)
```

Boards Supported	All
Purpose	Gets the starting address of a channel-gain queue.

Prototype

C/C++
DASErr far pascal K_GetChnGAry (DWORD *hFrame*,
void far * far *pArray*);

Turbo Pascal
Function K_GetChnGAry (*hFrame* : Longint;
Var *pArray* : Integer) : Word;

Turbo Pascal for Windows
Function K_GetChnGAry (*hFrame* : Longint;
Var *pArray* : Integer) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetChnGAry Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pArray* As Long) As Integer

BASIC
DECLARE FUNCTION KGetChnGAry% ALIAS "K_GetChnGAry"
(BYVAL *hFrame* AS LONG, SEG *pArray* AS LONG)

Parameters

hFrame   Handle to the frame that defines the operation.

pArray   Channel-gain queue starting address.

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

K_GetChnGAry (cont.)

Remarks

For the operation defined by *hFrame*, this function stores the starting address of the channel-gain queue in *pArray*.

The *pArray* variable contains the value of the Channel-Gain Queue element.

Refer to page 2-14 for information on setting up a channel-gain queue.

See Also

K_SetChnGAry

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
void far *pArray;
...
wDasErr = K_GetChnGAry (hAD, &pArray);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
pArray : Integer;
...
wDasErr = K_GetChnGAry (hAD, pArray);
```

Turbo Pascal for Windows
```
($I DASDECL.INC)
...
pArray : Integer;
...
wDasErr = K_GetChnGAry (hAD, pArray);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_GetChnGAry (hAD, pArray)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = K_GetChnGAry (hAD, pArray)
```

Boards Supported	All

Purpose Gets the pacer clock source.

Prototype

C/C++
DASErr far pascal K_GetClk (DWORD *hFrame*, short far **pMode*);

Turbo Pascal
Function K_GetClk (*hFrame* : Longint; Var *pMode* : Word) : Word;

Turbo Pascal for Windows
Function K_GetClk (*hFrame* : Longint; Var *pMode* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetClk Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetClk% ALIAS "K_GetClk"
(BYVAL *hFrame* AS LONG, SEG *pMode* AS INTEGER)

Parameters

hFrame   Handle to the frame that defines the operation.

pMode   Pacer clock source.
                Value stored: **0** for Internal
                               1 for External

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks For the operation defined by *hFrame*, this function stores the pacer clock source in *pMode*.

An internal clock source is the output of the onboard counter/timer circuitry; an external clock source is an external signal connected to the DI0/XPCLK pin (DAS-1800HC Series) or XPCLK pin (DAS-1800ST/HR Series).

Refer to page 2-15 (for analog input operations), page 2-29 (for analog output operations), and page 2-36 (for digital I/O operations) for more information about pacer clock sources.

The *pMode* variable contains the value of the Clock Source element.

See Also   K_SetClk, K_SetClkRate

Usage   **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
Word wMode;
...
wDasErr = K_GetClk (hAD, &wMode);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wMode : Word;
...
wDasErr := K_GetClk (hAD, wMode);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wMode : Word;
...
wDasErr := K_GetClk (hAD, wMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wMode As Integer
...
wDasErr = K_GetClk (hAD, wMode)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM wMode AS INTEGER
...
wDasErr = KGetClk% (hAD, wMode)
```

K_GetClkRate

Boards Supported	All
Purpose	Gets the internal clock divisor (clock ticks) for the 5 MHz clock source.

Prototype

C/C++
DASErr far pascal K_GetClkRate (DWORD *hFrame*,
DWORD far **pRate*);

Turbo Pascal
Function K_GetClkRate (*hFrame* : Longint; Var *pRate* : Longint) : Word;

Turbo Pascal for Windows
Function K_GetClkRate (*hFrame* : Longint; Var *pRate* : Longint) : Word;
far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetClkRate Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pRate* As Long) As Integer

BASIC
DECLARE FUNCTION KGetClkRate% ALIAS "K_GetClkRate"
(BYVAL *hFrame* AS LONG, SEG *pRate* AS LONG)

Parameters

hFrame   Handle to the frame that defines the operation.

pRate   Number of clock ticks between conversions.
Value stored: **15** to **4,294,967,295**

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

For the operation defined by *hFrame*, this function stores the number of clock ticks between conversions in *pRate*.

The *pRate* variable contains the value of the Pacer Clock Rate element.

This function applies to an internal clock source only.

After an interrupt-mode or DMA-mode analog input operation, the value stored in *pRate* represents the actual count used, not necessarily the count set by **K_SetClkRate**.

See Also K_SetClkRate

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
DWORD dwRate;
...
wDasErr = K_GetClkRate (hAD, &dwRate);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
dwRate : Longint;
...
wDasErr := K_GetClkRate (hAD, dwRate);
```

Turbo Pascal for Windows
```
($I DASDECL.INC)
...
dwRate : Longint;
...
wDasErr := K_GetClkRate (hAD, dwRate);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global dwRate As Long
...
wDasErr = K_GetClkRate (hAD, dwRate)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM dwRate AS LONG
...
wDasErr = KGetClkRate% (hAD, dwRate)
```

Boards Supported	All
Purpose	Gets the buffering mode.
Prototype	**C/C++** DASErr far pascal K_GetContRun (DWORD *hFrame*, short far **pMode*); **Turbo Pascal** Function K_GetContRun (*hFrame* : Longint;Var *pMode* : Word) : Word; **Turbo Pascal for Windows** Function K_GetContRun (*hFrame* : Longint;Var *pMode* : Word) : Word; far; external 'DASSHELL'; **Visual Basic for Windows** Declare Function K_GetContRun Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pMode* As Integer) As Integer **BASIC** DECLARE FUNCTION KGetContRun% ALIAS "K_GetContRun" (BYVAL *hFrame* AS LONG, SEG *pMode* AS INTEGER)

Parameters	*hFrame*	Handle to the frame that defines the operation.
	pMode	Buffering mode. Value stored: **0** for Single-cycle **0** for Continuous

Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.
Remarks	For the operation defined by *hFrame*, this function stores the buffering mode in *pMode*.

The *pMode* variable contains the value of the Buffering Mode element.

Refer to page 2-18 (for analog input operations), page 2-30 (for analog output operations) section, and page 2-38 (for digital I/O operations) for a description of buffering modes.

See Also   K_SetContRun, K_ClrContRun

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
. . .
WORD wMode;
. . .
wDasErr = K_GetContRun (hAD, &wMode);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
. . .
wMode : Word;
. . .
wDasErr := K_GetContRun (hAD, wMode);
```

Turbo Pascal for Windows
```
($I DASDECL.INC)
. . .
wMode : Word;
. . .
wDasErr := K_GetContRun (hAD, wMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
. . .
Global wMode As Integer
. . .
wDasErr = K_GetContRun (hAD, wMode)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM wMode AS INTEGER
...
wDasErr = KGetContRun% (hAD, wMode)
```

K_GetDAFrame

Boards Supported   DAS-1801HC, DAS-1802HC

Purpose   Accesses a D/A frame for an analog output operation.

Prototype

C/C++
DASErr far pascal K_GetDAFrame (DWORD *hDev*,
DWORD far * *pFrame*);

Turbo Pascal
Function K_GetDAFrame (*hDev* : Longint;
Var *pFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_GetDAFrame (*hDev* : Longint;
Var *pFrame* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetDAFrame Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *pFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KGetDAFrame% ALIAS "K_GetDAFrame"
(BYVAL *hDev* AS LONG, SEG *hFrame* AS LONG)

Parameters   *hDev*   Handle associated with the board.

    pFrame   Handle to the frame that defines the D/A
                          operation.

Return Value   This function returns an integer error/status code. Error/status code 0
indicates that the function executed successfully. A non-zero error/status
code indicates that an error occurred. Refer to Appendix A for additional
information.

Remarks   This function specifies that you want to perform an interrupt-mode analog
                output operation on the board specified by *hDev*, and accesses an
                available D/A frame with the handle *pFrame*. The frame is initialized to
                its default settings; the default settings are given in Table 3-2 on page 3-7.

See Also   K_FreeFrame, K_ClearFrame

Usage   **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
DWORD hDA;
...
wDasErr = K_GetDAFrame (hDev, &hDA);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
hDA : Longint;
...
wDasErr := K_GetDAFrame (hDev, hDA);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
hDA : Longint;
...
wDasErr := K_GetDAFrame (hDev, hDA);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global hDA As Long
...
wDasErr = K_GetDAFrame (hDev, hDA)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM hDA AS LONG
...
wDasErr = KGetDAFrame% (hDev, hDA)
```

Boards Supported	All
Purpose	Initializes any Keithley DAS board.
Prototype	

C/C++
DASErr far pascal K_GetDevHandle (DWORD *hDrv*,
WORD *nBoardNum*, DWORD far * *pDev*);

Turbo Pascal
Not supported

Turbo Pascal for Windows
Function K_GetDevHandle (*hDrv* : Longint; *nBoardNum* : Integer;
Var *pDev* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetDevHandle Lib "DASSHELL.DLL"
(ByVal *hDrv* As Long, ByVal *nBoardNum* As Integer, *pDev* As Long)
As Integer

BASIC
Not supported

Parameters	*hDrv*	Driver handle of the associated Function Call Driver.
	nBoardNum	Board number. Valid values: **0** to **2**
	pDev	Handle associated with the board.

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function initializes the board associated with *hDrv* and specified by *nBoardNum*, and stores the board handle of the specified board in *pDev*.

The value stored in *pDev* is intended to be used exclusively as an argument to functions that require a board handle. Your program should not modify the value stored in *pDev*.

This function is available for C, Borland Turbo Pascal for Windows, and Visual Basic for Windows application programs only.

See Also   K_FreeDevHandle

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
DWORD hDev;
...
wDasErr = K_GetDevHandle (hDrv, 0, &hDev);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
hDev : Longint;
...
wDasErr := K_GetDevHandle (hDrv, 0, hDev);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global hDev As Long
...
wDasErr = K_GetDevHandle (hDrv, 0, hDev)
```

Boards Supported	All
Purpose	Accesses a DI frame for a digital input operation.
Prototype	**C/C++**
	DASErr far pascal K_GetDIFrame (DWORD *hDev*,
	DWORD far * *pFrame*);

Turbo Pascal
Function K_GetDIFrame (*hDev* : Longint; Var *pFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_GetDIFrame (*hDev* : Longint; Var *pFrame* : Longint) : Word;
far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetDIFrame Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *pFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KGetDIFrame% ALIAS "K_GetDIFrame"
(BYVAL *hDev* AS LONG, SEG *pFrame* AS LONG)

Parameters	*hDev*	Handle associated with the board.
	pFrame	Handle to the frame that defines the digital input operation.
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.	

K_GetDIFrame (cont.)

Remarks   This function specifies that you want to perform an interrupt-mode digital input operation on the board specified by *hDev*, and accesses an available digital input frame with the handle *pFrame*. The frame is initialized to its default settings; the default settings are given in Table 3-3 on page 3-8.

See Also   K_FreeFrame, K_ClearFrame

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
DWORD hDI;
...
wDasErr = K_GetDIFrame (hDev, &hDI);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
hDI : Longint;
...
wDasErr := K_GetDIFrame (hDev, hDI);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
hDI : Longint;
...
wDasErr := K_GetDIFrame (hDev, hDI);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global hDI As Long
...
wDasErr = K_GetDIFrame (hDev, hDI)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM hDI AS LONG
...
wDasErr = KGetDIFrame% (hDev, hDI)
```

K_GetDITrig

Boards Supported	All
Purpose	Reads the current digital trigger conditions.
Prototype	**C/C++**

C/C++
DASErr far pascal K_GetDITrig (DWORD *hFrame*, short far * *pOpt*, short far **pChan*, DWORD far **pPattern*);

Turbo Pascal
Function K_GetDITrig (*hFrame* : Longint; Var *pOpt* : Word; Var *pChan* : Word; Var *pPattern* : Longint) : Word;

Turbo Pascal for Windows
Function K_GetDITrig (*hFrame* : Longint; Var *pOpt* : Word; Var *pChan* : Word; Var *pPattern* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetDITrig Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pOpt* As Integer, *pChan* As Integer, *pPattern* As Long) As Integer

BASIC
DECLARE FUNCTION KGetDITrig% ALIAS "K_GetDITrig" (BYVAL *hFrame* AS LONG, SEG *pOpt* AS INTEGER, SEG *pChan* AS INTEGER, SEG *pPattern* AS LONG)

Parameters	*hFrame*	Handle to the frame that defines the operation.
	pOpt	Trigger polarity and sensitivity. Value stored: **0** for Positive edge **2** for Negative edge
	pChan	Trigger channel. Value stored: **0**
	pPattern	Trigger pattern.

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the operation defined by *hFrame*, this function stores the trigger polarity and trigger sensitivity in *pOpt*, the channel used for the digital trigger in *pChan*, and the trigger pattern in *pPattern*.

Since the DAS-1800 Series Function Call Driver does not currently support digital pattern triggering, the value of *pPattern* is meaningless; the *pPattern* parameter is provided for future compatibility.

The *pOpt* variable contains the value of the Trigger Polarity and Trigger Sensitivity elements.

The *pChan* variable contains the value of the Trigger Channel element.

See Also   K_SetDITrig

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
short nOpt, nChan,
WORD wPat;
...
wDasErr = K_GetDITrig (hAD, &nOpt, &nChan, &wPat);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
nOpt : Integer;
nChan : Integer;
wPat : Word;
...
wDasErr := K_GetDITrig (hAD, nOpt, nChan, wPat);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
nOpt : Integer;
nChan : Integer;
wPat : Word;
...
wDasErr := K_GetDITrig (hAD, nOpt, nChan, wPat);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global nOpt As Integer
Global nChan As Integer
Global wPat As Integer
...
wDasErr = K_GetDITrig (hAD, nOpt, nChan, wPat)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM nOpt AS INTEGER
DIM nChan AS INTEGER
DIM wPat AS INTEGER
...
wDasErr = KGetDITrig% (hAD, nOpt, nChan, wPat)
```

Boards Supported	All
Purpose	Gets the digital output value.
Prototype	**C/C++** DASErr far pascal K_GetDOCurVal (DWORD *hFrame*, void far **pValue*);

Turbo Pascal
Function K_GetDOCurVal (*hFrame* : Longint;
Var *pValue* : Longint) : Word;

Turbo Pascal for Windows
Function K_GetDOCurVal (*hFrame* : Longint;
Var *pValue* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetDOCurVal Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pValue* As Long) As Integer

BASIC
DECLARE FUNCTION KGetDOCurVal% ALIAS "K_GetDOCurVal"
(BYVAL *hFrame* AS LONG, SEG *pValue* AS LONG)

Parameters	*hFrame*	Handle to the frame that defines the digital output operation.
	pValue	Digital output value. Value stored: **0** to **255** for DAS-1800HC Series boards **0** to **15** for DAS-1800ST/HR Series boards

Return Value   This function returns an integer error/status code. Error/status code 0
indicates that the function executed successfully. A non-zero error/status
code indicates that an error occurred. Refer to Appendix A for additional
information.

Remarks   On return, *pValue* contains the digital output value that was specified as
the *pValue* parameter in the most recent call to **K_DOWrite**. This value is
not necessarily the current value at the digital output channel.

Only the least-significant eight bits of *pValue* are valid for DAS-1800HC
Series boards; only the least-significant four bits of *pValue* are valid for
DAS-1800ST/HR Series boards.

See Also   K_DOWrite

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
. . .
DWORD dwDOValue;
. . .
wDasErr = K_GetDOCurVal (hDO, &dwDOValue);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
. . .
dwDOValue : Longint;
. . .
wDasErr := K_GetDOCurVal (hDO, dwDOValue);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
. . .
dwDOValue : Longint;
. . .
wDasErr := K_GetDOCurVal (hDO, dwDOValue);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global dwDOValue As Long
...
wDasErr = K_GetDOCurVal (hDO, dwDOValue)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM dwDOValue AS LONG
...
wDasErr = KGetDOCurVal% (hDO, dwDOValue)
```

K_GetDOFrame

Boards Supported	All
Purpose	Accesses a DO frame for a digital output operation.
Prototype	**C/C++** DASErr far pascal K_GetDOFrame (DWORD *hDev*, DWORD far * *pFrame*); **Turbo Pascal** Function K_GetDOFrame (*hDev* : Longint; Var *pFrame* : Longint) : Word; **Turbo Pascal for Windows** Function K_GetDOFrame (*hDev* : Longint; Var *pFrame* : Longint) : Word; far; external 'DASSHELL'; **Visual Basic for Windows** Declare Function K_GetDOFrame Lib "DASSHELL.DLL" (ByVal *hDev* As Long, *pFrame* As Long) As Integer **BASIC** DECLARE FUNCTION KGetDOFrame% ALIAS "K_GetDOFrame" (BYVAL *hDev* AS LONG, SEG *pFrame* AS LONG)

Parameters

hDev	Handle associated with the board.
hFrame	Handle to the frame that defines the digital output operation.

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function specifies that you want to perform an interrupt-mode digital output operation on the board specified by *hDev* and accesses an available digital output frame with the handle *hFrame*. The frame is initialized to its default settings; the default settings are given in Table 3-4 on page 3-9.

See Also   K_FreeFrame, K_ClearFrame

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
DWORD hDO;
...
wDasErr = K_GetDOFrame (hDev, &hDO);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
hDO : Longint;
...
wDasErr := K_GetDOFrame (hDev, hDO);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
hDO : Longint;
...
wDasErr := K_GetDOFrame (hDev, hDO);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global hDO As Long
...
wDasErr = K_GetDOFrame (hDev, hDO)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM hDO AS LONG
...
wDasErr = KGetDOFrame% (hDev, hDO)
```

Boards Supported	All

Purpose   Gets the address of an error message string.

Prototype   **C/C++**
DASErr far pascal K_GetErrMsg (DWORD *hDev*, short *nDASErr*, char far * far * *pErrMsg*);

Turbo Pascal
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Not supported

BASIC
Not supported

Parameters   *hDev*   Handle associated with the board.

nDASErr   Error message number.

pErrMsg   Address of error message string.

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the board specified by *hDev*, this function stores the address of the string corresponding to error message number *nDASErr* in *pErrMsg*.

Refer to page 2-4 for more information about error handling. Refer to Appendix A for a list of error codes and their meanings.

Usage

C/C++

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
char far *pErrMsg;
...
wDasErr = K_GetErrMsg (hDev, wDasErr, &pErrMsg);
```

Boards Supported	All

Purpose   Reads the active edge of the external clock.

Prototype

C/C++
DASErr far pascal K_GetExtClkEdge (DWORD *hFrame*, short far **pEdge*);

Turbo Pascal
Function K_GetExtClkEdge (*hFrame* : Longint; Var *pEdge* : Word) : Word;

Turbo Pascal for Windows
Function K_GetExtClkEdge (*hFrame* : Longint; Var *pEdge* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetExtClkEdge Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pEdge* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetExtClkEdge% ALIAS "K_GetExtClkEdge" (BYVAL *hFrame* AS LONG, SEG *pEdge* AS INTEGER)

Parameters

hFrame	Handle to the frame that defines the operation.
pEdge	Active edge of external clock. Value stored: **0** for Negative edge **1** for Positive edge

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

K_GetExtClkEdge (cont.)

Remarks   For the operation defined by *hFrame*, this function stores the active edge
of the external clock in *pEdge*.

The *pEdge* variable contains the value of the External Clock Edge
element.

See Also   K_SetExtClkEdge

Usage   **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
WORD wMode;
...
wDasErr = K_GetExtClkEdge (hAD, &wMode);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wMode : Word;
...
wDasErr := K_GetExtClkEdge (hAD, wMode);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wMode : Word;
...
wDasErr := K_GetExtClkEdge (hAD, wMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wMode As Integer
...
wDasErr = K_GetExtClkEdge (hAD, wMode)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM wMode AS INTEGER
...
wDasErr = KGetExtClkEdge% (hAD, wMode)
```

K_GetG

Boards Supported	All
Purpose	Gets the gain.
Prototype	**C/C++** DASErr far pascal K_GetG (DWORD *hFrame*, short far **pGain*);

Turbo Pascal
Function K_GetG (*hFrame* : Longint; Var *pGain* : Word) : Word;

Turbo Pascal for Windows
Function K_GetG (*hFrame* : Longint; Var *pGain* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetG Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pGain* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetG% ALIAS "K_GetG"
(BYVAL *hFrame* AS LONG, SEG *pGain* AS INTEGER)

Parameters	*hFrame* Handle to the frame that defines the operation.
	pGain Gain code. Valid values: **0** to **3** for DAS board channels **0** to **7** for EXP-1800 channels Refer to Table 2-2 on page 2-10 for the gain and input ranges associated with each gain code.
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks For the operation defined by *hFrame*, this function stores the gain code for a single channel or for a group of consecutive channels in *pGain*.

See Also K_SetG, K_SetStartStopG

Usage **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
WORD wGain;
...
wDasErr = K_GetG (hAD, &wGain);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wGain : Word;
...
wDasErr := K_GetG (hAD, wGain);
```

Turbo Pascal for Windows
```
($I DASDECL.INC)
...
wGain : Word;
...
wDasErr := K_GetG (hAD, wGain);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wGain As Integer
...
wDasErr = K_GetG (hAD, wGain)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM wGain AS INTEGER
...
wDasErr = KGetG% (hAD, wGain)
```

K_GetGate

Boards Supported	All

Purpose Gets the status of the hardware gate.

Prototype **C/C++**
DASErr far pascal K_GetGate (DWORD *hFrame*, short far **pMode*);

Turbo Pascal
Function K_GetGate (*hFrame* : Longint; Var *pMode* : Word) : Word;

Turbo Pascal for Windows
Function K_GetGate (*hFrame* : Longint; Var *pMode* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetGate Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetGate% ALIAS "K_GetGate"
(BYVAL *hFrame* AS LONG, SEG *pMode* AS INTEGER)

Parameters *hFrame* Handle to the frame that defines the operation.

pMode Status of the hardware gate.
Value stored: **0** for Gate disabled
1 for Positive gate enabled
2 for Negative gate enabled

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

For the operation defined by *hFrame*, this function stores the status of the hardware gate in *pMode*.

The *pMode* variable contains the value of the Hardware Gate element.

See Also

K_SetGate

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
WORD wMode;
...
wDasErr = K_GetGate (hAD, &wMode);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wMode : Word;
...
wDasErr := K_GetGate (hAD, wMode);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wMode : Word;
...
wDasErr := K_GetGate (hAD, wMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wMode As Integer
...
wDasErr = K_GetGate (hAD, wMode)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM wMode AS INTEGER
...
wDasErr = KGetGate% (hAD, wMode)
```

**Boards
Supported**

All

Purpose

Gets the current DAS shell version.

Prototype

C/C++
DASErr far pascal K_GetShellVer (WORD far *pVersion);

Turbo Pascal
Function K_GetShellVer (Var pVersion : Word) : Word;

Turbo Pascal for Windows
Function K_GetShellVer (Var pVersion : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetShellVer Lib "DASSHELL.DLL"
(pVersion As Integer) As Integer

BASIC
DECLARE FUNCTION KGetShellVer% ALIAS "K_GetShellVer"
(SEG pVersion AS INTEGER)

Parameters

pVersion   A word value containing the major and minor
                             version numbers of the DAS shell.

Return Value

This function returns an integer error/status code. Error/status code 0
indicates that the function executed successfully. A non-zero error/status
code indicates that an error occurred. Refer to Appendix A for additional
information.

Remarks

To obtain the major version number of the DAS shell, divide pVersion by
256. To obtain the minor version number of the DAS shell, perform a
Boolean AND operation with pVersion and 256 (0FF hex).

K_GetShellVer (cont.)

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
WORD wShellVer;
...
wDasErr = K_GetShellVer (&wShellVer);
printf ("Shell Ver %d.%d", wShellVer >> 8, wShellVer & 0xff);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wShellVer : Word;
...
wDasErr := K_GetShellVer (wShellVer);
FormatStr(VerStr, ' %4x ', nShellVer / 256, '.', nShellVer And
$ff);
writeln(' Shell Ver ', VerStr);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wShellVer : Word;
...
wDasErr := K_GetShellVer (wShellVer);
FormatStr(VerStr, ' %4x ', nShellVer / 256, '.', nShellVer And
$ff);
writeln(' Shell Ver ', VerStr);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wShellVer As Integer;
...
wDasErr = K_GetShellVer (wShellVer)
ShellVer$ = LTRIM$(STR$(wShellVer / 256)) + "." + :
   LTRIM$(STR$(wShellVer AND &HFF))
PRINT "Driver Ver: " + ShellVer$
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM wShellVer AS INTEGER
...
wDasErr = KGetShellVer% (wShellVer)
ShellVer$ = LTRIM$(STR$(nShellVer / 256)) + "." + :
   LTRIM$(STR$(nShellVer AND &HFF))
PRINT "Shell Ver: " + ShellVer$
```

K_GetSSH

Boards Supported   All

Purpose   Gets the status of the SSH mode.

Prototype

C/C++
DASErr far pascal K_GetSSH (DWORD *hFrame*, WORD far **pMode*);

Turbo Pascal
Function K_GetSSH (*hFrame* : Longint; Var *pMode* : Word) : Word;

Turbo Pascal for Windows
Function K_GetSSH (*hFrame* : Longint; Var *pMode* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetSSH Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetSSH% ALIAS "K_GetSSH"
(BYVAL *hFrame* AS LONG, SEG *pMode* AS INTEGER)

Parameters

hFrame   Handle to the frame that defines the operation.

pMode   Code that indicates the SSH mode.
                         Value stored: **0** for Disabled
                                        1 for Enabled

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks
For the operation defined by *hFrame*, this function stores the code that indicates the SSH mode in *pMode*.

The *pMode* variable contains the value of the SSH Mode element.

Refer to page 2-15 for information on conversion modes.

See Also
K_SetSSH

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
WORD wMode;
...
wDasErr = K_GetSSH (hAD, &wMode);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wMode : Word;
...
wDasErr := K_GetSSH (hAD, wMode);
```

Turbo Pascal for Windows
```
($I DASDECL.INC)
...
wMode : Word;
...
wDasErr := K_GetSSH (hAD, wMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wMode As Integer
...
wDasErr = K_GetSSH (hAD, wMode)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM wMode AS INTEGER
...
wDasErr = KGetSSH% (hAD, wMode)
```

Boards   All
Supported

Purpose   Gets the first and last channels in a group of consecutive channels.

Prototype   **C/C++**
DASErr far pascal K_GetStartStopChn (DWORD *hFrame*,
short far **pStart*, short far **pStop*);

Turbo Pascal
Function K_GetStartStopChn (*hFrame* : Longint; Var *pStart* : Word;
Var *pStop* : Word) : Word;

Turbo Pascal for Windows
Function K_GetStartStopChn (*hFrame* : Longint; Var *pStart* : Word;
Var *pStop* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetStartStopChn Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pStart* As Integer, *pStop* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetStartStopChn% ALIAS
"K_GetStartStopChn" (BYVAL *hFrame* AS LONG,
SEG *pStart* AS INTEGER, SEG *pStop* AS INTEGER)

Parameters   *hFrame*   Handle to the frame that defines the operation.

	Valid channel numbers	
Board	Differential	Single-ended
DAS-1800HC	0 to 31	0 to 63
DAS-1800ST/HR without EXP-1800 expansion boards attached	0 to 7	0 to 15
DAS-1800ST/HR with N EXP-1800 expansion boards attached	Not applicable	0 to 15(N + 1)

pStart   First channel in a group of consecutive channels. Valid values:

pStop   Last channel in a group of consecutive channels. Valid values: Same as for *pStart* above

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the operation defined by *hFrame*, this function stores the first channel in a group of consecutive channels in *pStart* and the last channel in the group of consecutive channels in *stop*.

The *pStart* variable contains the value of the Start Channel element.

The *pStop* variable contains the value of the Stop Channel element.

See Also   K_SetStartStopChn, K_GetStartStopG

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
short nStart, nStop;
...
wDasErr = K_GetStartStopChn (hAD, &nStart, &nStop);
```

Turbo Pascal

```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
nStart : Integer;
nStop : Integer;
...
wDasErr := K_GetStartStopChn (hAD, nStart, nStop)
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
nStart : Integer;
nStop : Integer;
...
wDasErr := K_GetStartStopChn (hAD, nStart, nStop)
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global nStart As Integer
Global nStop As Integer
...
wDasErr = K_GetStartStopChn (hAD, nStart, nStop)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM nStart AS INTEGER
DIM nStop AS INTEGER
...
wDasErr = KGetStartStopChn% (hAD, nStart, nStop)
```

K_GetStartStopG

Boards Supported   All

Purpose   Gets the first and last channels in a group of consecutive channels and the gain for all channels in the group.

Prototype   **C/C++**
DASErr far pascal K_GetStartStopG (DWORD *hFrame*,
short far **pStart*, short far **pStop*, short far **pGain*);

Turbo Pascal
Function K_GetStartStopG (*hFrame* : Longint; Var *pStart* : Word;
Var *pStop* : Word; Var *pGain* : Word) : Word;

Turbo Pascal for Windows
Function K_GetStartStopG (*hFrame* : Longint; Var *pStart* : Word;
Var *pStop* : Word; Var *pGain* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetStartStopG Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pStart* As Integer, *pStop* As Integer,
pGain As Integer) As Integer

BASIC
DECLARE FUNCTION KGetStartStopG% ALIAS "K_GetStartStopG"
(BYVAL *hFrame* AS LONG, SEG *pStart* AS INTEGER,
SEG *pStop* AS INTEGER, SEG *pGain* AS INTEGER)

Parameters *hFrame* Handle to the frame that defines the operation.

pStart First channel in a group of consecutive channels. Valid values:

| | Valid channel numbers | |
Board	Differential	Single-ended
DAS-1800HC	0 to 31	0 to 63
DAS-1800ST/HR without EXP-1800 expansion boards attached	0 to 7	0 to 15
DAS-1800ST/HR with *N* EXP-1800 expansion boards attached	Not applicable	0 to 15(*N* + 1)

pStop Last channel in a group of consecutive channels. Valid values: Same as for *pStart* above

pGain Gain code.
Valid values: **0** to **3** for DAS board channels
0 to **7** for EXP-1800 channels
Refer to Table 2-2 on page 2-10 for the gain and input ranges associated with each gain code.

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the operation defined by *hFrame*, this function stores the first channel
in a group of consecutive channels in *pStart*, the last channel in the group
of consecutive channels in *pStop*, and the gain code for all channels in the
group in *pGain*.

The *pStart* variable contains the value of the Start Channel element.

The *pStop* variable contains the value of the Stop Channel element.

The *pGain* variable contains the value of the Gain element.

See Also   K_SetStartStopG

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
short nStart, nStop, nGain;
...
wDasErr = K_GetStartStopG (hAD, &nStart, &nStop,
&nGain);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
nStart : Integer;
nStop : Integer;
nGain : Integer;
...
wDasErr := K_GetStartStopG (hAD, nStart, nStop, nGain)
```

Turbo Pascal for Windows
```
($I DASDECL.INC)
...
nStart : Integer;
nStop : Integer;
nGain : Integer;
...
wDasErr := K_GetStartStopG (hAD, nStart, nStop, nGain)
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
...
Global nStart As Integer
Global nStop As Integer
Global nGain As Integer
...
wDasErr = K_GetStartStopG (hAD, nStart, nStop, nGain)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM nStart AS INTEGER
DIM nStop AS INTEGER
DIM nGain AS INTEGER
...
wDasErr = KGetStartStopG% (hAD, nStart, nStop, nGain)
```

K_GetTrig

Boards Supported	All
Purpose	Gets the start trigger source.

Prototype

C/C++
DASErr far pascal K_GetTrig (DWORD *hFrame*, short far **pMode*);

Turbo Pascal
Function K_GetTrig (*hFrame* : Longint; Var *pMode* : Word) : Word;

Turbo Pascal for Windows
Function K_GetTrig (*hFrame* : Longint; Var *pMode* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KGetTrig% ALIAS "K_GetTrig"
(BYVAL *hFrame* AS LONG, SEG *pMode* AS INTEGER)

Parameters

hFrame	Handle to the frame that defines the operation.
pMode	Start trigger source.
	Value stored: 0 for Internal trigger
	1 for External trigger

Return Value
This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks
For the operation defined by *hFrame*, this function stores the trigger source in *pMode*.

The *pMode* variable contains the value of the Start Trigger Source element.

An internal trigger is a software trigger; conversions begin when the operation is started. An external trigger is either an analog trigger or a digital trigger; conversions begin when the trigger event occurs. Refer to page 2-25 for more information about internal and external trigger sources.

See Also   K_SetTrig

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
WORD wMode;
...
wDasErr = K_GetTrig (hAD, &wMode);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wMode : Word;
...
wDasErr := K_GetTrig (hAD, wMode);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wMode : Word;
...
wDasErr := K_GetTrig (hAD, wMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global wMode As Integer
...
wDasErr = K_GetTrig (hAD, wMode)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM wMode AS INTEGER
...
wDasErr = KGetTrig% (hAD, wMode)
```

Boards Supported	All
Purpose	Gets the trigger hysteresis value.
Prototype	**C/C++**
	DASErr far pascal K_GetTrigHyst (DWORD *hFrame*, short far **pHyst*);
	Turbo Pascal
	Function K_GetTrigHyst (*hFrame* : Longint; Var *pHyst* : Word) : Word;
	Turbo Pascal for Windows
	Function K_GetTrigHyst (*hFrame* : Longint; Var *pHyst* : Word) : Word; far; external 'DASSHELL';
	Visual Basic for Windows
	Declare Function K_GetTrigHyst Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pHyst* As Integer) As Integer
	BASIC
	DECLARE FUNCTION KGetTrigHyst% ALIAS "K_GetTrigHyst" (BYVAL *hFrame* AS LONG, SEG *pHyst* AS INTEGER)

Parameters	*hFrame*	Handle to the frame that defines the operation.
	pHyst	Hysteresis value.
		Value stored: **0** to **4,095** for DAS-1800HC/ST Series boards
		0 to **65,535** for DAS-1800HR Series boards

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

K_GetTrigHyst (cont.)

Remarks   For the operation defined by *hFrame*, this function stores the hysteresis value used for an analog trigger in *pHyst*. The value is represented in raw counts; refer to Appendix B for information on converting the raw count to voltage.

The *pHyst* variable contains the value of the Trigger Hysteresis element.

Refer to page 2-20 for more information about analog triggers.

See Also   K_SetTrigHyst

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
short nHyst;
...
wDasErr = K_GetTrigHyst (hAD, &nHyst);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
nHyst : Integer;
...
wDasErr := K_GetTrigHyst (hAD, nHyst);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
nHyst : Integer;
...
wDasErr := K_GetTrigHyst (hAD, nHyst);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global nHyst As Integer;
...
wDasErr = K_GetTrigHyst (hAD, nHyst)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM nHyst AS INTEGER
...
wDasErr = KGetTrigHyst% (hAD, nHyst)
```

K_GetVer

Boards Supported	All
Purpose	Gets revision numbers.
Prototype	**C/C++** DASErr far pascal K_GetVer (DWORD *hDev*, short far * *pSpecVer*, short far * *pDrvVer*);

Turbo Pascal
Function K_GetVer (*hDev* : Longint; Var *pSpecVer* : Word;
Var *pDrvVer* : Word) : Word;

Turbo Pascal for Windows
Function K_GetVer (*hDev* : Longint; Var *pSpecVer* : Word;
Var *pDrvVer* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetVer Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *pSpecVer* As Integer, *pDrvVer* As Integer) As
Integer

BASIC
DECLARE FUNCTION KGetVer% ALIAS "K_GetVer"
(BYVAL *hDev* AS LONG, SEG *pSpecVer* AS INTEGER,
SEG *pDrvVer* AS INTEGER)

Parameters	*hDev*	Handle associated with the board.
	pSpecVer	Revision number of the Keithley DAS Driver Specification to which the driver conforms.
	pDrvVer	Driver version number.
Return Value		This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

For the board specified by *hDev*, this function stores the revision number of the DAS-1800 Series Function Call Driver in *pDrvVer* and the revision number of the driver specification in *pSpecVer*.

The values stored in *pSpecVer* and *pDrvVer* are two-byte (16-bit) integers; the high byte of each contains the major revision level and the low byte of each contains the minor revision level. For example, if the driver version number is 2.1, the major revision level is 2 and the minor revision level is 1; therefore, the high byte of *pDrvVer* contains the value of **2** (512) and the low byte of *pDrvVer* contains the value of **1**; the value of both bytes is 513.

To extract the major and minor revision levels from the value stored in *pDrvVer* or *pSpecVer*, use the following equations:

$$\text{major revision level} = \text{Integer portion of}\left(\frac{\text{returned value}}{256}\right)$$

$$\text{minor revision level} = \text{returned value MOD } 256$$

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
. . .
short nSpecVer, nDrvVer;
. . .
wDasErr = K_GetVer (hDev, &nSpecVer, &nDrvVer);
printf ("Driver Ver %d.%d", nDrvVer >> 8, nDrvVer & 0xff);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
. . .
nSpecVer : Integer
nDrvVer : Integer
. . .
wDasErr := K_GetVer (hDev, nSpecVer, nDrvVer);
FormatStr(VerStr, ' %4x ', nDrvVer / 256, '.', nDrvVer And $ff);
writeln(' Driver Ver ', VerStr);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
nSpecVer : Integer
nDrvVer : Integer
...
wDasErr := K_GetVer (hDev, nSpecVer, nDrvVer);
FormatStr(VerStr, ' %4x ', nDrvVer / 256, '.', nDrvVer And $ff);
writeln(' Driver Ver ', VerStr);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global nSpecVer As Integer
Global nDrvVer As Integer
...
wDasErr = K_GetVer (hDev, nSpecVer, nDrvVer)
DrvVer$ = LTRIM$(STR$(nDrvVer / 256)) + "." + :
   LTRIM$(STR$(nDrvVer AND &HFF))
PRINT "Driver Ver: " + DrvVer$
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM nSpecVer AS INTEGER
DIM nDrvVer AS INTEGER
...
wDasErr = KGetVer% (hDev, nSpecVer, nDrvVer)
DrvVer$ = LTRIM$(STR$(nDrvVer / 256)) + "." + :
   LTRIM$(STR$(nDrvVer AND &HFF))
PRINT "Driver Ver: " + DrvVer$
```

Boards Supported All

Purpose Allocates a buffer for an interrupt-mode operation.

Prototype **C/C++**
DASErr far pascal K_IntAlloc (DWORD *hFrame*, DWORD *dwSamples*, void far * far *pBuf*, WORD far *pMem*);

Turbo Pascal
Function K_IntAlloc (*hFrame* : Longint; *dwSamples* : Longint; *pBuf* : Pointer; Var *pMem* : Word) : Word;

Turbo Pascal for Windows
Function K_IntAlloc (*hFrame* : Longint; *dwSamples* : Longint; *pBuf* : Pointer; Var *pMem* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_IntAlloc Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *dwSamples* As Long, *pBuf* As Long, *pMem* As Integer) As Integer

BASIC
DECLARE FUNCTION KIntAlloc% ALIAS "K_IntAlloc"
(BYVAL *hFrame* AS LONG, BYVAL *dwSamples* AS LONG,
SEG *pBuf* AS LONG, SEG *pMem* AS INTEGER)

Parameters

hFrame	Handle to the frame that defines the operation.
dwSamples	Number of samples. Valid values: 1 to **32,767** for Visual Basic for Windows and BASIC 1 to **65,536** for all other languages
pBuf	Starting address of the allocated buffer.
pMem	Handle associated with the allocated buffer.

K_IntAlloc (cont.)

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the operation defined by *hFrame*, this function allocates a buffer of the size specified by *dwSamples*, and stores the starting address of the buffer in *pBuf* and the handle of the buffer in *pMem*.

Turbo Pascal and BASIC require that you re-distribute available memory before you dynamically allocate a buffer. Refer to "Reducing the Memory Heap" on page 3-32 (Turbo Pascal) or page 3-46 (BASIC) for additional information.

See Also   K_IntFree, K_SetBuf, K_BufListAdd

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
void far *pBuf;    // Pointer to allocated buffer
WORD hMem;    // Memory Handle to buffer
...
wDasErr = K_IntAlloc (hAD, dwSamples, &pBuf, &hMem);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : ^BufType;    { buffer pointer }
hMem : Word;    { Handle to buffer }
...
wDasErr := K_IntAlloc(hAD, dwSamples, Addr(pBuf), hMem);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : ^BufType;    ( buffer pointer )
hMem : Word;    ( Handle to buffer )
...
wDasErr := K_IntAlloc(hAD, dwSamples, Addr(pBuf), hMem);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global pBuf As Long
Global hMem As Integer
...
wDasErr = K_IntAlloc (hAD, dwSamples, pBuf, hMem)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM pBuf AS LONG
DIM hMem AS INTEGER
...
wDasErr = KINTAlloc% (hAD, dwSamples, pBuf, hMem)
```

K_IntFree

Boards Supported	All
Purpose	Frees a buffer allocated for an interrupt-mode operation.
Prototype	**C/C++** DASErr far pascal K_IntFree (WORD *hMem*); **Turbo Pascal** Function K_IntFree (*hMem* : Word) : Integer; **Turbo Pascal for Windows** Function K_IntFree (*hMem* : Word) : Integer; far; external 'DASSHELL'; **Visual Basic for Windows** Declare Function K_IntFree Lib "DASSHELL.DLL" (ByVal *hMem* As Integer) As Integer **BASIC** DECLARE FUNCTION KIntFree% ALIAS "K_IntFree" (BYVAL *hMem* AS INTEGER)
Parameters	*hMem* Handle to interrupt buffer.
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.
Remarks	This function frees the buffer specified by *hMem*; the buffer was previously allocated dynamically using **K_IntAlloc**.
See Also	K_IntAlloc

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_IntFree (hMem);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_IntFree (hMem);
```

Turbo Pascal for Windows
```
($I DASDECL.INC}
...
wDasErr := K_IntFree (hMem);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
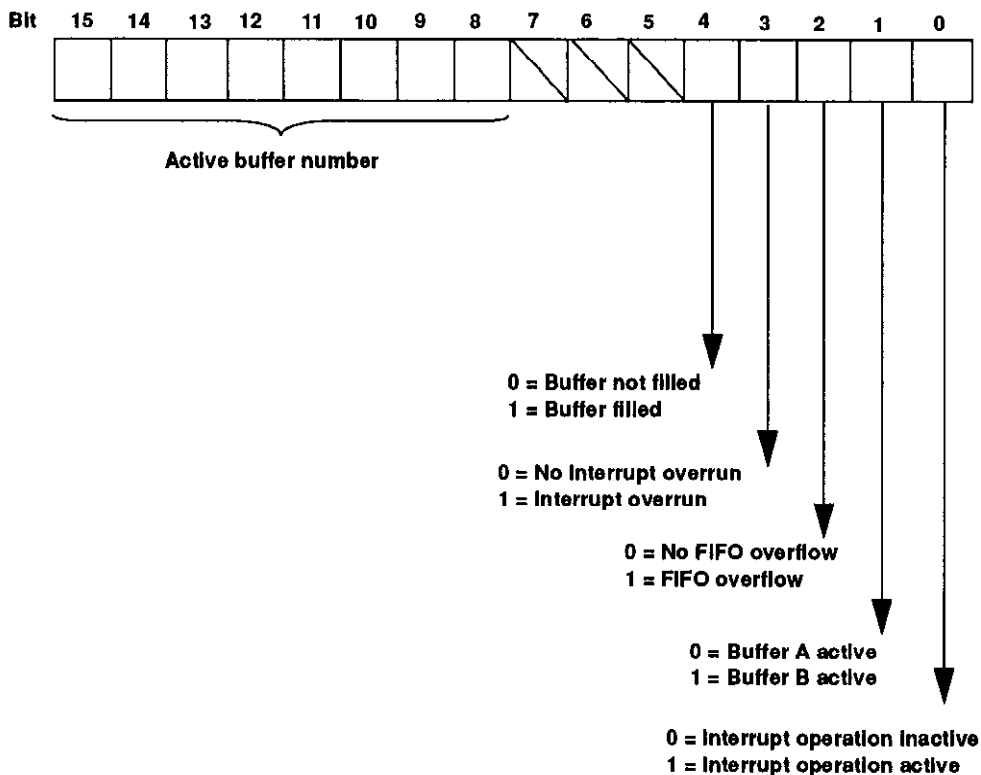```
...
wDasErr = K_IntFree (hMem)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KIntFree% (hMem)
```

K_IntStart

Boards Supported	All
Purpose	Starts an interrupt operation.
Prototype	**C/C++** DASErr far pascal K_IntStart (DWORD *hFrame*);
	Turbo Pascal Function K_IntStart (*hFrame* : Longint) : Word;
	Turbo Pascal for Windows Function K_IntStart (*hFrame* : Longint) : Word; far; external 'DASSHELL';
	Visual Basic for Windows Declare Function K_IntStart Lib "DASSHELL.DLL" (ByVal *hFrame* As Long) As Integer
	BASIC DECLARE FUNCTION KIntStart% ALIAS "K_IntStart" (BYVAL *hFrame* AS LONG)
Parameters	*hFrame* Handle to the frame that defines the operation.
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.
Remarks	This function starts the interrupt operation defined by *hFrame*. Refer to Chapter 3 for a discussion of the programming tasks associated with interrupt operations.
See Also	K_IntStatus, K_IntStop

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_IntStart (hAD);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_IntStart (hAD);
```

Turbo Pascal for Windows
```
($I DASDECL.INC)
...
wDasErr := K_IntStart (hAD);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_IntStart (hAD)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KIntStart% (hAD)
```

K_IntStatus

Boards Supported   All

Purpose   Gets status of interrupt operation.

Prototype   **C/C++**
DASErr far pascal K_IntStatus (DWORD *hFrame*, short far **pStatus*, DWORD far **pCount*);

Turbo Pascal
Function K_IntStatus (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word;

Turbo Pascal for Windows
Function K_IntStatus (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_IntStatus Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long) As Integer

BASIC
DECLARE FUNCTION KIntStatus% ALIAS "K_IntStatus"
(BYVAL *hFrame* AS LONG, SEG *pStatus* AS INTEGER, SEG *pCount* AS LONG)

Parameters   *hFrame*   Handle to the frame that defines the operation.

pStatus   Status of interrupt operation; see **Remarks** below for value stored.

pCount   Number of samples that were acquired. Value stored: **0** to **65,536**

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the interrupt operation defined by *hFrame*, this function stores the status in *pStatus* and the number of samples acquired in *pCount*.

The value stored in *pStatus* depends on the settings in the Status word, as shown below:

Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Active buffer number

0 = Buffer not filled
1 = Buffer filled

0 = No interrupt overrun
1 = Interrupt overrun

0 = No FIFO overflow
1 = FIFO overflow

0 = Buffer A active
1 = Buffer B active

0 = Interrupt operation inactive
1 = Interrupt operation active

The bits are described as follows:

- Bit 0: Indicates whether an interrupt-mode operation is in progress.

- Bit 1: The Buffer A/B active bit. If you are using multiple buffers, this bit toggles each time acquisition sample storage is switched to a new buffer. If you are using a single buffer and the operation is in continuous mode, this bit toggles each time an acquisition sample is stored at the beginning of the buffer.

- Bit 2: When set, this bit indicates that the onboard FIFO has overflowed. This event automatically stops all conversions.

- Bit 3: When set, this bit indicates that the board issued an interrupt while the CPU was processing a previous interrupt from the same board.

- Bit 4: This bit is used during continuous buffering mode; it is set when all data acquisition buffers that are currently assigned to the active operation have been filled with data at least once.

- Bits 5-7: Unassigned.

- Bits 8-15: In multiple-buffer acquisitions, these bits indicate the current active buffer number. The active buffer number is related to the Status word as follows:

$$\text{active buffer} = \frac{\text{Status word}}{256}$$

See Also   K_IntStart, K_IntStop

Usage   **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
WORD wStatus;
DWORD dwCount;
...
wDasErr = K_IntStatus (hAD, &wStatus, &dwCount);
```

Turbo Pascal

```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_IntStatus (hAD, wStatus, dwCount);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_IntStatus (hAD, wStatus, dwCount);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global wStatus As Integer
Global dwCount As Long
...
wDasErr = K_IntStatus (hAD, wStatus, dwCount)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM wStatus AS INTEGER
DIM dwCount AS LONG
...
wDasErr = KIntStatus% (hAD, wStatus, dwCount)
```

K_IntStop

Boards Supported	All

Purpose Stops an interrupt operation.

Prototype

C/C++
DASErr far pascal K_IntStop (DWORD *hFrame*, short far **pStatus*, DWORD far **pCount*);

Turbo Pascal
Function K_IntStop (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word;

Turbo Pascal for Windows
Function K_IntStop (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_IntStop Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long) As Integer

BASIC
DECLARE FUNCTION KIntStop% ALIAS "K_IntStop"
(BYVAL *hFrame* AS LONG, SEG *pStatus* AS INTEGER, SEG *pCount* AS LONG)

Parameters

hFrame	Handle to the frame that defines the operation.
pStatus	Status of interrupt operation.
pCount	Number of samples that were acquired. Value stored: **0 to 65,536**

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

This function stops the interrupt operation defined by *hFrame* and stores the status of the interrupt operation in *pStatus* and the number of samples acquired in *pCount*.

Refer to page 4-159 for the meaning of the value stored in *pStatus*.

If an interrupt operation is not in progress, **K_IntStop** is ignored.

See Also

K_IntStart, K_IntStatus

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
WORD wStatus;
DWORD dwCount;
...
wDasErr = K_IntStop (hAD, &wStatus, &dwCount);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_IntStop (hAD, wStatus, dwCount);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_IntStop (hAD, wStatus, dwCount);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
...
Global wStatus As Integer
Global dwCount As Long
...
wDasErr = K_IntStop (hAD, wStatus, dwCount)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM wStatus AS INTEGER
DIM dwCount AS LONG
...
wDasErr = KIntStop% (hAD, wStatus, dwCount)
```

Boards Supported   All

Purpose   Converts a local array to a buffer suitable for a DMA-mode analog input operations.

Prototype

C/C++
Not supported

Turbo Pascal
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Not supported

BASIC
DECLARE FUNCTION KMakeDMABuf% ALIAS "K_MakeDMABuf" (*dwSamples* AS LONG, *pBuf* AS INTEGER, *pBufAddr* AS LONG, *pStartIx* AS INTEGER)

Parameters

dwSamples	Number of samples.
pBuf	$DYNAMIC integer array.
pBufAddr	Starting address of the DMA buffer.
pStartIx	Index into *pBuf* that identifies the location in which the first sample is stored.

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

KMakeDMABuf (cont.)

Remarks   This function ensures that the array address provided to **K_SetDMABuf** is suitable for a DMA-mode analog input operation.

The size of the array given by *pBuf* must be declared so as to accommodate twice the number of samples as given by *dwSamples;* refer to page 3-46 for additional information.

See Also   K_SetDMABuf, K_BufListAdd

Usage

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
$DYNAMIC
DIM ADBuf(10000)As Integer
$STATIC
DIM pDMABuf AS LONG
...
wDasErr = KMakeDMABuf% (dwSamp, ADBuf, pDMABuf, nStartIx)
```

Boards Supported   All

Purpose   Transfers data from a locally dimensioned buffer to a buffer allocated through **K_IntAlloc** or **K_DMAAlloc**.

Prototype

C/C++
Not supported

Turbo Pascal
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_MoveArrayToBuf Lib "DASSHELL.DLL" Alias "K_MoveDataBuf" (ByVal *pDest* As Long, *pSource* As Integer, ByVal *nCount* As Integer) As Integer

BASIC
DECLARE FUNCTION KMoveArrayToBuf% ALIAS "K_MoveArrayToBuf" (ByVal *pDest* As Long, SEG *pSource* As Integer, ByVal *nCount* As Integer)

Parameters   *pDest*   Address of destination buffer.

pSource   Address of source buffer.

nCount   Number of samples to transfer.
Value values: **1** to **32,767**

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function transfers the number of samples specified by *nCount* from the buffer at address *pSource* to the buffer at address *pDest*.

If the buffer used to store output data for your program was allocated through **K_IntAlloc** or **K_DMAAlloc**, the buffer is not accessible to the driver and you must use this function to move the data to a buffer that the driver can use. If the buffer used to store output data for your program was dimensioned locally within the program's memory area, the buffer is accessible to the driver and you do not have to use this function.

See Also   K_DMAAlloc, K_IntAlloc

Usage

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
. . .
wDasErr = K_IntAlloc ( hDA, dwSamples, pBuf, hMem )
. . .
wDasErr = K_MoveArrayToBuf ( pBuf, DACArray(0), dwSamples )
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
. . .
wDasErr = KIntAlloc% ( hDA, dwSamples, pBuf, hMem )
. . .
wDasErr = KMoveArrayToBuf% ( pBuf, DACArray(0), dwSamples )
```

Boards Supported	All

Purpose
Transfers data from a buffer allocated through **K_IntAlloc** or **K_DMAAlloc** to a locally dimensioned buffer.

Prototype

C/C++
Not supported

Turbo Pascal
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_MoveBufToArray Lib "DASSHELL.DLL" Alias "K_MoveDataBuf" (*pDest* As Integer, ByVal *pSource* As Long, ByVal *nCount* As Integer) As Integer

BASIC
DECLARE FUNCTION KMoveBufToArray% ALIAS "K_MoveBufToArray" (SEG *pDest* As Integer, ByVal *pSource* As Long, ByVal *nCount* As Integer)

Parameters

pDest	Address of destination buffer.
pSource	Address of source buffer.
nCount	Number of samples to transfer. Value values: **1** to **32,767**

Return Value
This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

K_MoveBufToArray (cont.)

Remarks   This function transfers the number of samples specified by *nCount* from the buffer at address *pSource* to the array at address *pDest*.

If the buffer used to store acquired data for your program was allocated through **K_IntAlloc** or **K_DMAAlloc**, the buffer is not accessible to your program and you must use this function to move the data to an accessible buffer. If the buffer used to store acquired data for your program was dimensioned locally within the program's memory area, the buffer is accessible to your program and you do not have to use this function.

See Also   K_DMAAlloc, K_IntAlloc

Usage

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
. . .
wDasErr = K_IntAlloc ( hAD, dwSamples, pBuf, hMem )
. . .
wDasErr = K_MoveBufToArray ( ADArray(0), pBuf, dwSamples)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
. . .
wDasErr = KIntAlloc% ( hAD, dwSamples, pBuf, hMem )
. . .
wDasErr = K_MoveBufToArray% ( ADArray(0), pBuf, dwSamples)
```

Boards Supported	All
Purpose	Initializes any Keithley DAS Function Call Driver.

Prototype

C/C++
DASErr far pascal K_OpenDriver (char far * *szDevName*,
char far * *szCfgName*, DWORD far * *pDrv*);

Turbo Pascal
Not supported

Turbo Pascal for Windows
Function K_OpenDriver (Var *szDevName* : char; Var *szCfgName* : char;
Var *pDrv* : LongInt) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_OpenDriver Lib "DASSHELL.DLL"
(ByVal *szDASName* As String, ByVal *szCfgName* As String,
pDrv As Long) As Integer

BASIC
Not supported

Parameters

szDASName	Board name.	
	Valid value:	**"DAS1800"** (for DAS-1800 Series boards)
szCfgName	Driver configuration file.	
	Valid values:	The name of a configuration file
		0 if driver has already been opened
pDrv	Handle associated with the driver.	

Return Value   This function returns an integer error/status code. Error/status code 0
indicates that the function executed successfully. A non-zero error/status

code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

This function initializes the Function Call Driver for the board associated with *szDASName* according to the information in the configuration file specified by *szCfgName*, and stores the driver handle in *pDrv*.

You can use this function to initialize the Function Call Driver associated with any Keithley DAS board. For DAS-1800 Series boards, the string stored in *szDASName* must be DAS1800. Refer to other Function Call Driver user's guides for the appropriate string to store in *szDASName* for other Keithley DAS boards.

The value stored in *pDrv* is intended to be used exclusively as an argument to functions that require a driver handle. Your program should not modify the value stored in *pDrv*.

You create a configuration file using the D1800CFG.EXE utility. Refer to your DAS-1800 Series board user's guide for more information.

If *szCfgName* = 0, **K_OpenDriver** checks whether the driver has already been opened and linked to a configuration file and if it has, uses the current configuration; this is useful in the Windows environment.

See Also

DAS1800_DevOpen

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
. . .
DWORD hDrv;
. . .
wDasErr = K_OpenDriver ("DAS1800", "DAS1802.CFG", &hDrv);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
szDrvName : String;
szCfgName : String;
hDrv : Longint;
...
szDrvName := 'DAS1800' + #0;
szCfgName := 'DAS1802.CFG' + #0;
wDasErr := K_OpenDriver (szDrvName[1], szCfgName[1], hDrv)
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
DIM hDrv As Long
...
wDasErr = K_OpenDriver("DAS1800", "DAS1802.CFG", hDrv)
```

K_RestoreChnGAry

Boards
Supported
All

Purpose
Restores a converted channel-gain queue.

Prototype
C/C++
Not supported

Turbo Pascal
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_RestoreChnGAry Lib "DASSHELL.DLL"
(*pArray* As Integer) As Integer

BASIC
DECLARE FUNCTION KRestoreChanGAry% ALIAS
"K_RestoreChnGAry" (SEG *pArray* AS INTEGER)

Parameters
pArray   Channel-gain queue starting address.

Return Value
This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks
This function restores a channel-gain queue that was converted using **K_FormatChnGAry** to its original format so that it can be used by your BASIC or Visual Basic for Windows program.

Refer to page 4-59 for more information about the **K_FormatChnGAry** function.

See Also
K_FormatChnGAry, K_SetChnGAry

Usage

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
...
Global ChanGainArray(16) As Integer    ' Chan/Gain
array
...
wDasErr = K_RestoreChnGAry (ChanGainArray(0))
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM ChanGainArray(16) AS INTEGER    ' Chan/Gain array
...
wDasErr = KRestoreChnGAry% (ChanGainArray(0))
```

K_SetAboutTrig

Boards Supported	All
Purpose	Enables the about trigger and specifies the number of post-trigger samples.

Prototype

C/C++
DASErr far pascal K_SetAboutTrig (DWORD *hFrame*,
DWORD *dwSamples*);

Turbo Pascal
Function K_SetAboutTrig (*hFrame* : Longint;
dwSamples : Longint) : Word;

Turbo Pascal for Windows
Function K_SetAboutTrig (*hFrame* : Longint;
dwSamples : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetAboutTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *dwSamples* As Long) As Integer

BASIC
DECLARE FUNCTION KSetAboutTrig% ALIAS "K_SetAboutTrig"
(BYVAL *hFrame* AS LONG, BYVAL *dwSamples* AS LONG)

Parameters

hFrame	Handle to the frame that defines the operation.
dwSamples	Number of post-trigger samples. Valid values: 1 to **65,535**

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the DMA-mode analog input operation defined by *hFrame*, this
function enables the about trigger and specifies the number of post-trigger
samples.

See Also   K_ClrAboutTrig, K_GetAboutTrig

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
DWORD dwSamples;
...
wDasErr = K_SetAboutTrig (hAD, dwSamples);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
dwSamples : Longint;
...
wDasErr := K_SetAboutTrig (hAD, dwSamples);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
dwSamples : Longint;
...
wDasErr := K_SetAboutTrig (hAD, dwSamples);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
Global dwSamples As Long
...
wDasErr = K_SetAboutTrig (hAD, dwSamples)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM dwSamples AS LONG
...
wDasErr = KSetAboutTrig% (hAD, dwSamples)
```

Boards Supported   All

Purpose   Set a DAS board's A/D common-mode ground reference.

Prototype

C/C++
DASErr far pascal K_SetADCommonMode (DWORD *hDev*,
WORD *nMode*);

Turbo Pascal
Function K_SetADCommonMode (*hDev* : Longint;
nMode : Word) : Word;

Turbo Pascal for Windows
Function K_SetADCommonMode (*hDev* : Longint;
nMode : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetADCommonMode Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, ByVal *nMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetADCommonMode% ALIAS
"K_SetADCommonMode" (BYVAL *hDev* AS LONG,
BYVAL *nMode* AS INTEGER)

Parameters   *hDev*   Handle to the frame that defines the operation.

 nMode   A/D common-mode ground reference.
                  Value stored: **0** for LL-GND
                                 1 for user-defined

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

K_SetADCommonMode (cont.)

Remarks   For the board specified by *hDev*, this function specifies the A/D
                   common-mode ground reference in *nMode*.

See Also   K_GetADCommonMode

Usage   **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
WORD nMode;
...
wDasErr = K_SetADCommonMode (hDev, nMode);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
nMode : Word;
...
wDasErr = K_SetADCommonMode (hDev, nMode);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
nMode : Word;
...
wDasErr = K_SetADCommonMode (hDev, nMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
DIM nMode As Integer
...
wDasErr = K_SetADCommonMode (hDev, nMode)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM nMode AS INTEGER
...
wDasErr = KSetADCommonMode% (hDev, nMode)
```

Boards Supported   All

Purpose   Set a DAS board's A/D input channel configuration.

Prototype

C/C++
DASErr far pascal K_SetADConfig (DWORD *hDev*, WORD *nMode*);

Turbo Pascal
Function K_SetADConfig (*hDev* : Longint; *nMode* : Word) : Word;

Turbo Pascal for Windows
Function K_SetADConfig (*hDev* : Longint; *nMode* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetADConfig Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, ByVal *nMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetADConfig% ALIAS "K_SetADConfig"
(BYVAL *hDev* AS LONG, BYVAL *nMode* AS INTEGER)

Parameters   *hFrame*   Handle associated with the board.

nMode   A/D input channel configuration.
Value stored: **0** for Differential
              1 for Single-ended

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function specifies, in *nMode*, the A/D input channel configuration for the board specified by *hDev*.

K_SetADConfig (cont.)

See Also   K_GetADConfig

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
WORD nMode;
...
wDasErr = K_SetADConfig (hDev, nMode);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
nMode : Word;
...
wDasErr = K_SetADConfig (hDev, nMode);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
nMode : Word;
...
wDasErr = K_SetADConfig (hDev, nMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
DIM nMode As Integer
...
wDasErr = K_SetADConfig (hDev, nMode)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM nMode AS INTEGER
...
wDasErr = KSetADConfig% (hDev, nMode)
```

Boards Supported   All

Purpose   Specifies burst conversion mode.

Prototype

C/C++
DASErr far pascal K_SetADFreeRun (DWORD *hFrame*);

Turbo Pascal
Function K_SetADFreeRun (*hFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_SetADFreeRun (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetADFreeRun Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KSetADFreeRun% ALIAS "K_SetADFreeRun"
(BYVAL *hFrame* AS LONG)

Parameters   *hFrame*   Handle to the frame that defines the operation.

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   This function sets the conversion mode for the operation defined by *hFrame* to burst mode. Refer to page 2-15 for information on conversion modes.

See Also   K_ClrADFreeRun, K_GetADFreeRun

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_SetADFreeRun (hAD, 1);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetADFreeRun (hAD, 1);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_SetADFreeRun (hAD, 1);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_SetADFreeRun (hAD, 1)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetADFreeRun% (hAD, 1)
```

Boards Supported	All
Purpose	Set a DAS board's A/D input range type.
Prototype	**C/C++** DASErr far pascal K_SetADMode (DWORD *hDev*, WORD *nMode*); **Turbo Pascal** Function K_SetADMode (*hDev* : Longint; *nMode* : Word) : Word; **Turbo Pascal for Windows** Function K_SetADMode (*hDev* : Longint; *nMode* : Word) : Word; far; external 'DASSHELL'; **Visual Basic for Windows** Declare Function K_SetADMode Lib "DASSHELL.DLL" (ByVal *hDev* As Long, ByVal *nMode* As Integer) As Integer **BASIC** DECLARE FUNCTION KSetADMode% ALIAS "K_SetADMode" (BYVAL *hDev* AS LONG, BYVAL *nMode* AS INTEGER)
Parameters	*hDev* Handle associated with the board. *nMode* A/D input range type. Valid values: 0 for Bipolar 1 for Unipolar
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.
Remarks	For the board specified by *hDev*, this function specifies the A/D input range type in *nMode*.

K_SetADMode (cont.)

See Also   K_GetADMode

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
WORD nMode;
...
wDasErr = K_SetADMode (hDev, nMode);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
nMode : Word;
...
wDasErr = K_SetADMode (hDev, nMode);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
nMode : Word;
...
wDasErr = K_SetADMode (hDev, nMode);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
DIM nMode As Integer
...
wDasErr = K_SetADMode (hDev, nMode)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
DIM nMode AS INTEGER
...
wDasErr = KSetADMode% (hDev, nMode)
```

Boards **Supported**	All
Purpose	Sets up an analog start trigger.
Prototype	**C/C++** DASErr far pascal K_SetADTrig (DWORD *hFrame*, short *nOpt*, short *nChan*, DWORD *dwLevel*); **Turbo Pascal** Function K_SetADTrig (*hFrame* : Longint; *nOpt* : Word; *nChan* : Word; *dwLevel* : Longint) : Word; **Turbo Pascal for Windows** Function K_SetADTrig (*hFrame* : Longint; *nOpt* : Word; *nChan* : Word; *dwLevel* : Longint) : Word; far; external 'DASSHELL'; **Visual Basic for Windows** Declare Function K_SetADTrig Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *nOpt* As Integer, ByVal *nChan* As Integer, ByVal *dwLevel* As Long) As Integer **BASIC** DECLARE FUNCTION KSetADTrig% ALIAS "K_SetADTrig" (BYVAL *hFrame* AS LONG, BYVAL *nOpt* AS INTEGER, BYVAL *nChan* AS INTEGER, BYVAL *dwLevel* AS LONG)

Parameters

hFrame	Handle to the frame that defines the operation.
nOpt	Analog trigger polarity and sensitivity. Valid values: **0** for Positive edge **2** for Negative edge

nChan   Analog input channel used as trigger channel.
                          Valid values:

| Board | Valid channel numbers | |
	Differential	Single-ended
DAS-1800HC	0 to 31	0 to 63
DAS-1800ST/HR without EXP-1800 expansion boards attached	0 to 7	0 to 15
DAS-1800ST/HR with N EXP-1800 expansion boards attached	Not applicable	0 to 15(N + 1)

dwLevel   Level at which the trigger event occurs, specified
                          in raw counts. Valid values:
                          DAS-1800HC/ST Series boards:
                                  0 to **4,095** (Unipolar)
                                  –2048 to **2047** (Bipolar)
                          DAS-1800HR Series boards:
                                  0 to **65,535** (Unipolar)
                                  –32,768 to **32,767** (Bipolar)

Return Value   This function returns an integer error/status code. Error/status code 0
                          indicates that the function executed successfully. A non-zero error/status
                          code indicates that an error occurred. Refer to Appendix A for additional
                          information.

Remarks   For the operation defined by *hFrame*, this function specifies the channel
                          used for an analog trigger in *nChan*, the level used for the analog trigger
                          in *dwLevel*, and the trigger polarity and trigger sensitivity in *nOpt*.

                          You specify the value for *dwLevel* in raw counts. Refer to Appendix B for
                          information on converting the actual voltage to a raw count.

The values you specify set the following elements in the frame identified by *hFrame*:

- *nOpt* sets the value of the Trigger Polarity and Trigger Sensitivity elements.

- *nChan* sets the value of the Trigger Channel element.

- *dwLevel* sets the value of the Trigger Level element.

K_SetADTrig does not affect the operation defined by *hFrame* unless the Trigger Source element is set to External (by a call to **K_SetTrig**) before *hFrame* is used as a calling argument to **K_IntStart** or **K_DMAStart**.

See Also   K_GetADTrig

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_SetADTrig (hAD, 0, 0, 2047);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetADTrig (hAD, 0, 0, 2047);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_SetADTrig (hAD, 0, 0, 2047);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_SetADTrig (hAD, 0, 0, 2047)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetADTrig% (hAD, 0, 0, 2047)
```

Boards Supported   All

Purpose   Specifies the starting address of a previously allocated or dimensioned buffer.

Prototype

C/C++
DASErr far pascal K_SetBuf (DWORD *hFrame*, void far **pBuf*, DWORD *dwSamples*);

Turbo Pascal
Function K_SetBuf (*hFrame* : Longint; *pBuf* : Pointer; *dwSamples* : Longint) : Word;

Turbo Pascal for Windows
Function K_SetBuf (*hFrame* : Longint; *pBuf* : Pointer; *dwSamples* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetBuf Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *pBuf* As Long,
ByVal *dwSamples* As Long) As Integer

BASIC
Not supported

Parameters   *hFrame*   Handle to the frame that defines the operation.

pBuf   Starting address of buffer.

dwSamples   Number of samples.
Valid values: 0 to 65,535

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks For the operation defined by *hFrame*, this function specifies the starting address of a previously allocated buffer in *pBuf* and the number of samples (the size of the buffer) in *dwSamples*.

Do not use this function for BASIC; for the BASIC languages, use **K_SetBufI**. Refer to page 4-194 for more information.

For C and Pascal application programs, use this function whether you dimensioned your buffer locally or allocated your buffer dynamically using **K_IntAlloc**. For buffers allocated dynamically using **K_DMAAlloc**, use **K_SetDMABuf**. For C, make sure that you use proper typecasting to prevent C/C++ type-mismatch warnings. For Pascal, a special procedure is needed to satisfy the type-checking requirements; refer to page 3-33 for more information.

For Visual Basic for Windows, use this function only for buffers allocated dynamically using **K_IntAlloc**. For buffers allocated dynamically using **K_DMAAlloc**, use **K_SetDMABuf**. For locally dimensioned buffers, use **K_SetBufI**.

Do not use this function if you are using multiple buffers. Use **K_BufListAdd** to specify the starting addresses of multiple buffers.

The values you specify set the following elements in the frame identified by *hFrame*:

- *pBuf* sets the value of the Buffer element.

- *dwSamples* sets the value of the Number of Samples element.

See Also K_DMAAlloc, K_IntAlloc, K_BufListAdd, K_SetBufI, K_SetDMABuf

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
void far *pBuf;    // Pointer to allocated buffer
...
wDasErr = K_IntAlloc (hAD, dwSamples, &pBuf, &hMem);
wDasErr = K_SetBuf (hAD, pBuf, dwSamples);
```

Turbo Pascal

```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : ^BufType;    ( buffer pointer )
...
wDasErr := K_IntAlloc(hAD, dwSamples, Addr(pBuf), hMem);
wDasErr := K_SetBuf (hAD, pBuf, dwSamples);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : ^BufType;    ( buffer pointer )
...
wDasErr := K_IntAlloc(hAD, dwSamples, Addr(pBuf), hMem);
wDasErr := K_SetBuf (hAD, pBuf, dwSamples);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global pBuf As Long
...
wDasErr = K_IntAlloc (hAD, dwSamples, pBuf, hMem)
wDasErr = K_SetBuf (hAD, pBuf, dwSamples)
```

K_SetBufI

Boards Supported	All
Purpose	Specifies the starting address of a locally dimensioned integer buffer.
Prototype	**C/C++** Not supported

Turbo Pascal
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_SetBufI Lib "DASSHELL.DLL" Alias "K_SetBuf"
(ByVal *hFrame* As Long, *pBuf* As Integer, ByVal *dwSize* As Long) As
Integer

BASIC
DECLARE FUNCTION K_SETBUFI Alias "K_SetBuf"
(BYVAL *hFrame* AS Long, *pBuf* AS Integer, BYVAL *dwSize* AS Long)
AS INTEGER

Parameters	*hFrame*	Handle to the frame that defines the operation.
	pBuf	Starting address of the user-dimensioned integer buffer.
	dwSize	Number of samples. Valid values: **0** to **65,535**

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the operation defined by *hFrame*, this function specifies the starting
address of a locally dimensioned integer buffer in *pBuf* and the number of
samples stored in the buffer in *dwSize*.

Do not use this function for C and Pascal; for these languages, use
K_SetBuf.

For Visual Basic for Windows, use this function only for locally
dimensioned buffers. For buffers allocated dynamically using
K_IntAlloc, use **K_SetBuf**. For buffers allocated dynamically using
K_DMAAlloc, use **K_SetDMABuf**.

Do not use this function if you are using multiple buffers. Instead, use
K_BufListAdd to specify the starting addresses of multiple buffers.

The values you specify set the following elements in the frame identified
by *hFrame*:

- *pBuf* sets the value of the Buffer element.

- *dwSize* sets the value of the Number of Samples element.

See Also   K_DMAAlloc, K_IntAlloc, K_BufListAdd, K_SetBuf, K_SetDMABuf

Usage   **Visual Basic for Windows**
                  (Include DASDECL.BAS in your program make file)

```
. . .
Dim ADData(2000) As Integer
. . .
wDasErr = K_SetBufI (hAD, ADData(0), 2000 )
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
. . .
Dim ADData(2000) As Integer
. . .
wDasErr = KSetBufI% (hAD, ADData(0), 2000 )
```

K_SetBurstTicks

Boards Supported	All
Purpose	Sets the burst mode conversion rate.

Prototype

C/C++
DASErr far pascal K_SetBurstTicks (DWORD *hFrame*, short *nTicks*);

Turbo Pascal
Function K_SetBurstTicks (*hFrame* : Longint; *nTicks* : Word) : Word;

Turbo Pascal for Windows
Function K_SetBurstTicks (*hFrame* : Longint; *nTicks* : Word) : Word;
far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetBurstTicks Lib "DASSHELL.DLL"
(ByVal hFrame as Long, ByVal *nTicks* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetBurstTicks% ALIAS "K_SetBurstTicks"
(BYVAL *hFrame* AS LONG, BYVAL *nTicks* AS INTEGER)

Parameters

hFrame   Handle to the frame that defines the A/D operation.

nTicks   The number of clock ticks between conversions of each channel in a scan.
Valid values: **3** to **255**

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

For the operation defined by *hFrame*, this function stores the number of clock ticks between conversions in *nTicks*.

Refer to page 2-17 for more information on burst mode conversion rate.

See Also

K_GetBurstTicks

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_SetBurstTicks (hAD, 10);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetBurstTicks (hAD, 10);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_SetBurstTicks (hAD, 10);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_SetBurstTicks (hAD, 10)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetBurstTicks% (hAD, 10)
```

K_SetChn

Boards Supported	All
Purpose	Specifies a single channel.
Prototype	**C/C++**

C/C++
DASErr far pascal K_SetChn (DWORD *hFrame*, short *nChan*);

Turbo Pascal
Function K_SetChn (*hFrame* : Longint; *nChan* : Word) : Word;

Turbo Pascal for Windows
Function K_SetChn (*hFrame* : Longint; *nChan* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetChn Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nChan* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetChn% ALIAS "K_SetChn"
(BYVAL *hFrame* AS LONG, BYVAL *nChan* AS INTEGER)

Parameters *hFrame* Handle to the frame that defines the operation.

nChan Channel on which to perform operation. Valid values:

Board	Valid channel numbers	
	Differential	Single-ended
DAS-1800HC	0 to 31	0 to 63
DAS-1800ST/HR without EXP-1800 expansion boards attached	0 to 7	0 to 15
DAS-1800ST/HR with *N* EXP-1800 expansion boards attached	Not applicable	0 to 15(*N* + 1)

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks For the operation defined by *hFrame*, this function specifies the single channel used in *nChan*.

The value you specify in *nChan* sets the Start Channel element and the Stop Channel element in the frame identified by *hFrame*.

See Also K_GetChn, K_GetStartStopChn, K_GetStartStopChnAry

Usage **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_SetChn (hAD, 2);
```

Turbo Pascal

```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetChn (hAD, 2);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetChn (hAD, 2);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
wDasErr = K_SetChn (hAD, 2)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetChn% (hAD, 2)
```

Boards Supported	All
Purpose	Specifies the starting address of a channel-gain queue.
Prototype	**C/C++** DASErr far pascal K_SetChnGAry (DWORD *hFrame*, void far **pArray*);
	Turbo Pascal Function K_SetChnGAry (*hFrame* : Longint; Var *pArray* : Integer) : Word;
	Turbo Pascal for Windows Function K_SetChnGAry (*hFrame* : Longint; Var *pArray* : Integer) : Word; far; external 'DASSHELL';
	Visual Basic for Windows Declare Function K_SetChnGAry Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pArray* As Integer) As Integer
	BASIC DECLARE FUNCTION KSetChnGAry% ALIAS "K_SetChnGAry" (BYVAL *hFrame* AS LONG, SEG *pArray* AS INTEGER)
Parameters	*hFrame* Handle to the frame that defines the operation.
	pArray Channel-gain queue starting address.
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

K_SetChnGAry (cont.)

Remarks   For the operation defined by *hFrame*, this function specifies the starting address of the channel-gain queue in *pArray*.

The value you specify in *pArray* sets the Channel-Gain Queue element in the frame identified by *hFrame*.

Refer to page 2-14 for information on setting up a channel-gain queue.

If you created your channel-gain queue in BASIC or Visual Basic for Windows, you must use **K_FormatChnGAry** to convert the channel-gain queue before you specify the address with **K_SetChnGAry**.

See Also   K_FormatChnGAry, K_RestoreChnGAry

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
// DECLARE AND INITIALIZE CHAN/GAIN PAIRS
// (GainChanTable-TYPE IS DEFINED IN dasdecl.h)
GainChanTable ChanGainArray= {2,    // # of entries
    0, 0,      // chan 0, gain 1
    1, 1};     // chan 1, gain 2 (DAS-1802)
...
wDasErr = K_SetChnGAry (hAD, &ChanGainArray);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
{ Define Gain/Channel array type }
TYPE GainChanTable = Record
        num_of_codes : Integer;
        queue : Array[0..15] of Byte;
        END;
CONST ChanGainArray : GainChanTable = (
        num_of_codes : (8);  { # of chan/gain pairs }
        queue : (0,0, 1,1)
);
...
wDasErr := K_SetChnGAry (hAD, ChanGainArray.num_of_codes);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
{ Define Gain/Channel array type }
TYPE GainChanTable = Record
      num_of_codes : Integer;
      queue : Array[0..15] of Byte;
      END;
CONST ChanGainArray : GainChanTable = (
      num_of_codes : (8);    { # of chan/gain pairs }
      queue : (0,0, 1,1)
);
...
wDasErr := K_SetChnGAry (hAD, ChanGainArray.num_of_codes);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
...
Global ChanGainArray(16) As Integer
...
' Create the array of channel/gain pairs
ChanGainArray(0) = 2    ' # of chan/gain pairs
ChanGainArray(1) = 0: ChanGainArray(2) = 0
ChanGainArray(3) = 1: ChanGainArray(4) = 1
wDasErr = K_FormatChnGAry (ChanGainArray(0))
wDasErr = K_SetChnGAry (hAD, ChanGainArray(0))
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM ChanGainArray(16) AS INTEGER
...
' Create the array of channel/gain pairs
ChanGainArray(0) = 2    ' # of chan/gain pairs
ChanGainArray(1) = 0: ChanGainArray(2) = 0
ChanGainArray(3) = 1: ChanGainArray(4) = 1
wDasErr = KFormatChnGAry% (ChanGainArray(0))
wDasErr = KSetChnGAry% (hAD, ChanGainArray(0))
```

K_SetClk

Boards Supported	All
Purpose	Specifies the pacer clock source.

Prototype

C/C++
DASErr far pascal K_SetClk (DWORD *hFrame*, short *nMode*);

Turbo Pascal
Function K_SetClk (*hFrame* : Longint; *nMode* : Word) : Word;

Turbo Pascal for Windows
Function K_SetClk (*hFrame* : Longint; *nMode* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetClk Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetClk% ALIAS "K_SetClk"
(BYVAL *hFrame* AS LONG, BYVAL *nMode* AS INTEGER)

Parameters

hFrame   Handle to the frame that defines the operation.

nMode   Pacer clock source.
Valid values: **0** for Internal
       1 for External

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

For the operation defined by *hFrame*, this function specifies the pacer clock source in *nMode*.

The value you specify in *nMode* sets the Clock Source element in the frame identified by *hFrame*.

The internal clock source is the output of the onboard counter/timer circuitry; an external clock source is an external signal connected to the DIO/XPCLK pin (DAS-1800HC Series) or XPCLK pin (DAS-1800ST/HR Series). Refer to page 2-15 (for analog input operations), page 2-29 (for analog output operations), and page 2-36 (for digital I/O operations) for more information about pacer clock sources.

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame**, and **K_ClearFrame** specify internal as the default clock source. The default active edge is negative for an external clock source; use **K_SetExtClkEdge** to specify a positive active edge.

See Also

K_GetClk

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
. . .
wDasErr = K_SetClk (hAD, 1);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
. . .
wDasErr := K_SetClk (hAD, 1);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
. . .
wDasErr := K_SetClk (hAD, 1);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)

```
. . .
wDasErr = K_SetClk (hAD, 1)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetClk% (hAD, 1)
```

Boards Supported   All

Purpose   Specifies the clock divisor for the internal 5 MHz clock source.

Prototype

C/C++
DASErr far pascal K_SetClkRate (DWORD *hFrame*,
DWORD *dwDivisor*);

Turbo Pascal
Function K_SetClkRate (*hFrame* : Longint; *dwDivisor* : LongInt) : Word;

Turbo Pascal for Windows
Function K_SetClkRate (*hFrame* : Longint; *dwDivisor* : LongInt) : Word;
far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetClkRate Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *dwDivisor* As Long) As Integer

BASIC
DECLARE FUNCTION KSetClkRate% ALIAS "K_SetClkRate"
(BYVAL *hFrame* AS LONG, BYVAL *dwDivisor* AS LONG)

Parameters

hFrame	Handle to the frame that defines the operation.
dwDivisor	Number of clock ticks between conversions. Valid values: **15** to **4,294,967,295**

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks   For the operation defined by *hFrame*, this function specifies the number
of clock ticks between conversions in *dwDivisor*.

The value you specify in *dwDivisor* sets the Pacer Clock Rate element in
the frame identified by *hFrame*.

This function applies to an internal clock source only. The tick resolution
is 0.2 μs.

Refer to page 2-15 for more information on the pacer clock.

See Also   K_GetClkRate

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
DWORD dwClkDiv;
...
dwClkDiv = 5000000 / 10000
wDasErr = K_SetClkRate (hAD, dwClkDiv);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
dwClkDiv : Longint;
...
dwClkDiv := 5000000 / 10000
wDasErr := K_SetClkRate (hAD, dwClkDiv);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
dwClkDiv : Longint;
...
dwClkDiv := 5000000 / 10000
wDasErr := K_SetClkRate (hAD, dwClkDiv);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global dwClkDiv As Long
...
dwClkDiv = 5000000 / 10000
wDasErr = K_SetClkRate (hAD, dwClkDiv);
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM dwClkDiv AS LONG
...
dwClkDiv = 5000000 / 10000
wDasErr = KSetClkRate% (hAD, dwClkDiv)
```

K_SetContRun

Boards Supported	All
Purpose	Specifies continuous buffering mode.

Prototype

C/C++
DASErr far pascal K_SetContRun (DWORD *hFrame*);

Turbo Pascal
Function K_SetContRun (*hFrame* : Longint) : Word;

Turbo Pascal for Windows
Function K_SetContRun (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetContRun Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

BASIC
DECLARE FUNCTION KSetContRun% ALIAS "K_SetContRun"
(BYVAL *hFrame* AS LONG)

Parameters *hFrame* Handle to the frame that defines the operation.

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks For the operation defined by *hFrame*, this function sets the buffering mode to continuous mode and sets the Buffering Mode element in the frame accordingly.

K_GetADFrame, K_GetDAFrame, K_GetDIFrame, K_GetDOFrame, and **K_ClearFrame** specify single-cycle as the default buffering mode.

Refer to page 2-38 (for analog input operations), page 2-38 (for analog output operations) section, and page 2-38 (for digital I/O operations) for a description of buffering modes.

See Also   K_GetContRun

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_SetContRun (hAD)
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetContRun (hAD)
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_SetContRun (hAD)
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_SetContRun (hAD)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetContRun% (hAD)
```

K_SetDITrig

Boards Supported	All
Purpose	Sets up a digital trigger.
Prototype	**C/C++** DASErr far pascal K_SetDITrig (DWORD *hFrame*, short *nOpt*, short *nChan*, DWORD *nPattern*);

Turbo Pascal
Function K_SetDITrig (*hFrame* : Longint; *nOpt* : Word; *nChan* : Word; *nPattern* : Longint) : Word;

Turbo Pascal for Windows
Function K_SetDITrig (*hFrame* : Longint; *nOpt* : Word; *nChan* : Word; *nPattern* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetDITrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nOpt* As Integer,
ByVal *nChan* As Integer, ByVal *nPattern* As Long) As Integer

BASIC
DECLARE FUNCTION KSetDITrig% ALIAS "K_SetDITrig"
(BYVAL *hFrame* AS LONG, BYVAL *nOpt* AS INTEGER,
BYVAL *nChan* AS INTEGER, BYVAL *nPattern* AS LONG)

Parameters	*hFrame*	Handle to the frame that defines the operation.
	nOpt	Trigger polarity and sensitivity. Valid values: **0** for Positive edge **2** for Negative edge
	nChan	Digital input channel. Valid value: **0**
	nPattern	Trigger pattern.

Return Value
This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks
This function specifies the use of a digital trigger for the operation defined by *hFrame*.

Since the DAS-1800 Series Function Call Driver does not currently support digital pattern triggering, the value of *nPattern* is meaningless; the *nPattern* parameter is provided for future compatibility.

The values you specify set the following elements in the frame identified by *hFrame*:

- *nOpt* sets the value of the Trigger Polarity and Trigger Sensitivity elements.

- *nChan* sets the value of the Trigger Channel element.

- *nPattern* sets the value of the Trigger Pattern element.

K_SetDITrig does not affect the operation defined by *hFrame* unless the Trigger Source element is set to External (by a call to **K_SetTrig**) before *hFrame* is used as a calling argument to **K_IntStart** or **K_DMAStart**.

See Also
K_GetDITrig

Usage
C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
. . .
wDasErr = K_SetDITrig (0, 0, 0);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
. . .
wDasErr := K_SetDITrig (0, 0, 0);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
. . .
wDasErr := K_SetDITrig (0, 0, 0);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
. . .
wDasErr = K_SetDITrig (0, 0, 0)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
. . .
wDasErr = KSetDITrig% (0, 0, 0)
```

Boards Supported   All

Purpose   Sets the values of a DMA buffer address and number of samples elements.

Prototype

C/C++
DASErr far pascal K_SetDMABuf (DWORD *hFrame*, void far **pBuf*, DWORD *dwSamples*);

Turbo Pascal
Function K_SetDMABuf (*hFrame* : Longint; *pBuf* : Pointer; *dwSamples* : Longint) : Word;

Turbo Pascal for Windows
Function K_SetDMABuf (*hFrame* : Longint; *pBuf* : Pointer; *dwSamples* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetDMABuf Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *pBuf* As Long,
ByVal *dwSamples* As Long) As Integer

BASIC
DECLARE FUNCTION KSetDMABuf% ALIAS "K_SetDMABuf"
(BYVAL *hFrame* AS LONG, BYVAL *pBuf* AS LONG,
BYVAL *dwSamples* AS LONG)

Parameters

hFrame	Handle to the frame that defines the DMA-mode analog input operation.
pBuf	Starting address of buffer.
dwSamples	Number of samples. Valid values: **0** to **65,535**

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

For the operation specified by *hFrame*, this function stores the address of the currently allocated buffer in *pBuf* and the number of samples stored in the buffer in *dwSamples*.

The *pBuf* variable contains the value of the Buffer element.

The *dwSamples* variable contains the value of the Number of Samples element.

See Also

K_DMAAlloc, KMakeDMABuf, K_BufListAdd

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
void far *pBuf;   // Pointer to allocated buffer
...
wDasErr = K_DMAAlloc (hAD, dwSamples, &pBuf, &hMem);
wDasErr = K_SetDMABuf (hAD, pBuf, dwSamples);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : ^BufType;   ( buffer pointer )
...
wDasErr := K_DMAAlloc(hAD, dwSamples, Addr(pBuf), hMem);
wDasErr := K_SetDMABuf (hAD, pBuf, dwSamples);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : ^BufType;    { buffer pointer }
...
wDasErr := K_DMAAlloc(hAD, dwSamples, Addr(pBuf), hMem);
wDasErr := K_SetDMABuf (hAD, pBuf, dwSamples);
```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
Global pBuf As Long
...
wDasErr = K_DMAAlloc (hAD, dwSamples, pBuf, hMem)
wDasErr = K_SetDMABuf (hAD, pBuf, dwSamples)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
DIM pBuf AS LONG
...
wDasErr = KDMAAlloc% (hAD, dwSamples, pBuf, hMem)
wDasErr = KSetDMABuf% (hAD, pBuf, dwSamples)
```

K_SetExtClkEdge

Boards Supported	All

Purpose Specifies the active edge of the external pacer clock.

Prototype **C/C++**
DASErr far pascal K_SetExtClkEdge (DWORD *hFrame*, short *nEdge*);

Turbo Pascal
Function K_SetExtClkEdge (*hFrame* : Longint; *nEdge* : Word) : Word;

Turbo Pascal for Windows
Function K_SetExtClkEdge (*hFrame* : Longint; *nEdge* : Word) : Word;
far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetExtClkEdge Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nEdge* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetExtClkEdge% ALIAS "K_SetExtClkEdge"
(BYVAL *hFrame* AS LONG, BYVAL *nEdge* AS INTEGER)

Parameters *hFrame* Handle to the frame that defines the operation.

nEdge Active edge of external pacer clock.
Valid values: 0 for Negative edge
                1 for Positive edge

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks For the operation defined by *hFrame*, this function sets the active edge of the external pacer clock and sets the External Clock Edge element in the frame accordingly.

K_SetExtClkEdge does not affect the operation defined by *hFrame* unless the Trigger Source element is set to External (by a call to **K_SetTrig**) before *hFrame* is used as a calling argument to **K_IntStart** or **K_DMAStart**.

See Also   K_GetExtClkEdge

Usage   **C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_SetExtClkEdge (hAD, 1)
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetExtClkEdge (hAD, 1)
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_SetExtClkEdge (hAD, 1)
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_SetExtClkEdge (hAD, 1)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetExtClkEdge% (hAD, 1)
```

K_SetG

Boards Supported	All

Purpose

Sets the gain.

Prototype

C/C++
DASErr far pascal K_SetG (DWORD *hFrame*, short *nGain*);

Turbo Pascal
Function K_SetG (*hFrame* : Longint; *nGain* : Word) : Word;

Turbo Pascal for Windows
Function K_SetG (*hFrame* : Longint; *nGain* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetG Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nGain* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetG% ALIAS "K_SetG"
(BYVAL *hFrame* AS LONG, BYVAL *nGain* AS INTEGER)

Parameters

hFrame   Handle to the frame that defines the operation.

nGain   Gain code.
Valid values: 0 to 3 for DAS board channels
              0 to 7 for EXP-1800 channels
Refer to Table 2-2 on page 2-10 for the gain and input ranges associated with each gain code.

Return Value

This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

For the operation defined by *hFrame*, this function specifies the gain code for a single channel or for a group of consecutive channels in *nGain*.

The value you specify in *nGain* sets the Gain element in the frame identified by *hFrame*.

K_GetADFrame, K_GetDAFrame, K_GetDIFrame, K_GetDOFrame, and **K_ClearFrame** specify 1 (gain code 0) as the default gain.

This function is valid for A/D frames only.

See Also

K_GetG, K_SetStartStopG

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_SetG (hAD, 1)
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetG (hAD, 1)
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_SetG (hAD, 1)
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_SetG (hAD, 1)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetG% (hAD, 1)
```

K_SetGate

Boards Supported	All
Purpose	Specifies the status of the hardware gate.
Prototype	**C/C++** DASErr far pascal K_SetGate (DWORD *hFrame*, short *nMode*);

Turbo Pascal
Function K_SetGate (*hFrame* : Longint; *nMode* : Word) : Word;

Turbo Pascal for Windows
Function K_SetGate (*hFrame* : Longint; *nMode* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetGate Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetGate% ALIAS "K_SetGate"
(BYVAL *hFrame* AS LONG, BYVAL *nMode* AS INTEGER)

Parameters	*hFrame*	Handle to the frame that defines the operation.
	nMode	Status of the hardware gate. Valid values: 0 for Gate disabled 1 for Positive gate enabled 2 for Negative gate enabled
Return Value	This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.	

Remarks

For the operation defined by *hFrame*, this function specifies the status of the hardware gate in *nMode*.

External gating is supported for analog input operations only. Also, you cannot enable the hardware gate if you are using an external digital trigger.

K_GetADFrame, K_GetDAFrame, K_GetDIFrame, K_GetDOFrame, and **K_ClearFrame** specify disabled as the default gate setting.

See Also

K_GetGate

Usage

C/C++
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_SetGate (hAD, 1)
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetGate (hAD, 1)
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_SetGate (hAD, 1)
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_SetGate (hAD, 1)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetGate% (hAD, 1)
```

K_SetSSH

Boards Supported   All

Purpose   Enables and disables SSH mode.

Prototype

C/C++
DASErr far pascal K_SetSSH (DWORD *hFrame*, WORD *nMode*);

Turbo Pascal
Function K_SetSSH (*hFrame* : Longint; *nMode* : Word) : Word;

Turbo Pascal for Windows
Function K_SetSSH (*hFrame* : Longint; *nMode* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetSSH Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetSSH% ALIAS "K_SetSSH"
(BYVAL *hFrame* AS LONG, BYVAL *nMode* AS INTEGER)

Parameters   *hFrame*   Handle to the frame that defines the operation.

nMode   Code that indicates the status of SSH mode.
Valid values: **0** for Disabled
   1 for Enabled

Return Value   This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks

For the operation defined by *hFrame*, this function stores the code that indicates the SSH mode in *nMode*.

K_GetADFrame and **K_ClearFrame** also disable SSH mode.

Refer to page 2-15 for information on SSH mode.

See Also

K_GetSSH

Usage

C/C++
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_SetSSH (hAD, 1)
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetSSH (hAD, 1)
```

Turbo Pascal for Windows
```
($I DASDECL.INC}
...
wDasErr := K_SetSSH (hAD, 1)
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_SetSSH (hAD, 1)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetSSH% (hAD, 1)
```

K_SetStartStopChn

Boards Supported	All
Purpose	Specifies the first and last channels in a group of consecutive channels.
Prototype	

C/C++
DASErr far pascal K_SetStartStopChn (DWORD *hFrame*, short *nStart*, short *nStop*);

Turbo Pascal
Function K_SetStartStopChn (*hFrame* : Longint; *nStart* : Word; *nStop* : Word) : Word;

Turbo Pascal for Windows
Function K_SetStartStopChn (*hFrame* : Longint; *nStart* : Word; *nStop* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetStartStopChn Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nStart* As Integer,
ByVal *nStop* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetStartStopChn% ALIAS
"K_SetStartStopChn" (BYVAL *hFrame* AS LONG,
BYVAL *nStart* AS INTEGER, BYVAL *nStop* AS INTEGER)

Parameters   *hFrame*   Handle to the frame that defines the operation.

               nStart   First channel in a group of consecutive channels.
                                       Valid values:

	Valid channel numbers	
Board	**Differential**	**Single-ended**
DAS-1800HC	0 to 31	0 to 63
DAS-1800ST/HR without EXP-1800 expansion boards attached	0 to 7	0 to 15
DAS-1800ST/HR with *N* EXP-1800 expansion boards attached	Not applicable	0 to 15(*N* + 1)

               nStop   Last channel in a group of consecutive channels.
                                       Valid values: Same as for *nStart* above

Return Value   This function returns an integer error/status code. Error/status code 0
               indicates that the function executed successfully. A non-zero error/status
               code indicates that an error occurred. Refer to Appendix A for additional
               information.

Remarks   For the operation defined by *hFrame*, this function specifies the first
                     channel in a group of consecutive channels in *nStart* and the last channel
                     in the group of consecutive channels in *nStop*.

                     The values you specify set the following elements in the frame identified
                     by *hFrame*:

                     • *nStart* sets the value of the Start Channel element.

                     • *nStop* sets the value of the Stop Channel element.

                     **K_GetADFrame, K_GetDAFrame, K_GetDIFrame,
                     K_GetDOFrame** and **K_ClearFrame** set the Start Channel and Stop
                     Channel elements to 0.

See Also   K_GetStartStopChn, K_SetStartStopG

Usage   **C/C++**
                     ```
                     #include "DASDECL.H"   // Use "DASDECL.HPP for C++
                     . . .
                     wDasErr = K_SetStartStopChn (hAD, 0, 7);
                     ```

                     Turbo Pascal
                     ```
                     uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
                     . . .
                     wDasErr := K_SetStartStopChn (hAD, 0, 7);
                     ```

                     Turbo Pascal for Windows
                     ```
                     {$I DASDECL.INC}
                     . . .
                     wDasErr := K_SetStartStopChn (hAD, 0, 7);
                     ```

                     Visual Basic for Windows
                     (Include DASDECL.BAS in your program make file)
                     ```
                     . . .
                     wDasErr = K_SetStartStopChn (hAD, 0, 7)
                     ```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetStartStopChn% (hAD, 0, 7)
```

K_SetStartStopG

Boards Supported All

Purpose Specifies the first and last channels in a group of consecutive channels and sets the gain for all channels in the group.

Prototype

C/C++
DASErr far pascal K_SetStartStopG (DWORD *hFrame*, short *nStart*, short *nStop*, short *nGain*);

Turbo Pascal
Function K_SetStartStopG (*hFrame* : Longint; *nStart* : Word; *nStop* : Word; *nGain* : Word) : Word;

Turbo Pascal for Windows
Function K_SetStartStopG (*hFrame* : Longint; *nStart* : Word; *nStop* : Word; *nGain* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetStartStopG Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *nStart* As Integer, ByVal *nStop* As Integer, ByVal *nGain* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetStartStopG% ALIAS "K_SetStartStopG" (BYVAL *hFrame* AS LONG, BYVAL *nStart* AS INTEGER, BYVAL *nStop* AS INTEGER, BYVAL *nGain* AS INTEGER)

Parameters   *hFrame*   Handle to the frame that defines the operation.

                 nStart   First channel in a group of consecutive channels.
                                             Valid values:

	Valid channel numbers	
Board	**Differential**	**Single-ended**
DAS-1800HC	0 to 31	0 to 63
DAS-1800ST/HR without EXP-1800 expansion boards attached	0 to 7	0 to 15
DAS-1800ST/HR with *N* EXP-1800 expansion boards attached	Not applicable	0 to 15(*N* + 1)

                 nStop   Last channel in a group of consecutive channels.
                                             Valid values: Same as for *nStart* above

                 nGain   Gain code.
                                             Valid values: 0 to 3 for DAS board channels
                                                              0 to 7 for EXP-1800 channels
                                             Refer to Table 2-2 on page 2-10 for the gain and
                                             input ranges associated with each gain code.

Return Value   This function returns an integer error/status code. Error/status code 0
                     indicates that the function executed successfully. A non-zero error/status
                     code indicates that an error occurred. Refer to Appendix A for additional
                     information.

Remarks   For the operation defined by *hFrame*, this function specifies the first
                     channel in a group of consecutive channels in *nStart*, the last channel in a
                     group of consecutive channels in *nStop*, and the gain code for all channels
                     in the group in *nGain*.

The values you specify set the following elements in the frame identified by *hFrame*:

- *nStart* sets the value of the Start Channel element.

- *nStop* sets the value of the Stop Channel element.

- *nGain* sets the value of the Gain element.

K_GetADFrame and **K_ClearFrame** set the Start Channel, Stop Channel, and Gain elements to 0.

See Also   K_GetStartStopG

Usage   **C/C++**
```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_SetStartStopG (hAD, 0, 7, 0);
```

Turbo Pascal
```
uses D1800TP7;   (* Use D1800TP6 for TP ver 6.0 *)
...
wDasErr := K_SetStartStopG (hAD, 0, 7, 0);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
...
wDasErr := K_SetStartStopG (hAD, 0, 7, 0);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
...
wDasErr = K_SetStartStopG (hAD, 0, 7, 0)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetStartStopG% (hAD, 0, 7, 0)
```

Boards Supported All

Purpose Specifies the trigger source.

Prototype **C/C++**
DASErr far pascal K_SetTrig (DWORD *hFrame*, short *nMode*);

Turbo Pascal
Function K_SetTrig (*hFrame* : Longint; *nMode* : Word) : Word;

Turbo Pascal for Windows
Function K_SetTrig (*hFrame* : Longint; *nMode* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nMode* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetTrig% ALIAS "K_SetTrig"
(BYVAL *hFrame* AS LONG, BYVAL *nMode* AS INTEGER)

Parameters *hFrame* Handle to the frame that defines the operation.

nMode Trigger source.
Valid values: **0** for Internal trigger
1 for External trigger

Return Value This function returns an integer error/status code. Error/status code 0 indicates that the function executed successfully. A non-zero error/status code indicates that an error occurred. Refer to Appendix A for additional information.

Remarks For the operation defined by *hFrame*, this function specifies the trigger source in *nMode*.

An internal trigger is a software trigger; conversions begin when the operation is started. An external trigger is either an analog trigger or a digital trigger; conversions begin when the trigger event occurs. Refer to page 2-25 for more information about internal and external trigger sources.

When performing a pre-trigger or about-trigger acquisition operation, mode, *nMode* refers to the start trigger.

If *nMode* = 1, an external digital trigger (positive edge on DI1/TGIN for DAS-1800HC Series boards, positive edge on TGIN for DAS-1800ST/HR Series boards) is assumed. Use **K_SetDITrig** to change the conditions of the digital trigger. Use **K_SetADTrig** to specify the conditions for an external analog trigger.

K_GetADFrame and **K_ClearFrame** set the trigger source to internal.

The external trigger source is relevant for analog input operations only.

See Also　　K_GetTrig

Usage　　**C/C++**
```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
. . .
wDasErr = K_SetTrig (hAD, 1);
```

Turbo Pascal
```
uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
. . .
wDasErr := K_SetTrig (hAD, 1);
```

Turbo Pascal for Windows
```
{$I DASDECL.INC}
. . .
wDasErr := K_SetTrig (hAD, 1);
```

Visual Basic for Windows
(Include DASDECL.BAS in your program make file)
```
. . .
wDasErr = K_SetTrig (hAD, 1)
```

BASIC
```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetTrig% (hAD, 1)
```

K_SetTrigHyst

Boards Supported	All
Purpose	Specifies the hysteresis value.

Prototype

C/C++
DASErr far pascal K_SetTrigHyst (DWORD *hFrame*, short *nHyst*);

Turbo Pascal
Function K_SetTrigHyst (*hFrame* : Longint; *nHyst* : Word) : Word;

Turbo Pascal for Windows
Function K_SetTrigHyst (*hFrame* : Longint; *nHyst* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetTrigHyst Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nHyst* As Integer) As Integer

BASIC
DECLARE FUNCTION KSetTrigHyst% ALIAS "K_SetTrigHyst"
(BYVAL *hFrame* AS LONG, BYVAL *nHyst* AS INTEGER)

Parameters

hFrame   Handle to the frame that defines the operation.

nHyst   Hysteresis value, specified in raw counts.
Valid values: **0** to **4,095** for DAS-1800HC/ST
Series boards
0 to **65,535** for DAS-1800HR
Series boards

Return Value   This function returns an integer error/status code. Error/status code 0
indicates that the function executed successfully. A non-zero error/status
code indicates that an error occurred. Refer to Appendix A for additional
information.

Remarks   For the operation defined by *hFrame*, this function specifies the
                hysteresis value used for an analog trigger in *nHyst*. You must specify the
                hysteresis value in raw counts. Refer to Appendix B for information on
                converting the hysteresis voltage to a raw count.

                The value you specify in *hyst* sets to the Trigger Hysteresis element in the
                frame identified by *hFrame*.

                K_SetTrigHyst does not affect the operation defined by *hFrame* unless
                the Trigger Source element is set to External (by a call to **K_SetTrig**)
                before *hFrame* is used as a calling argument to **K_IntStart** or
                K_DMAStart.

                Refer to page 2-19 for more information about analog triggers.

See Also   K_GetTrigHyst

Usage   **C/C++**
                ```
                #include "DASDECL.H"    // Use "DASDECL.HPP for C++
                ...
                wDasErr = K_SetTrigHyst (hAD, 50);
                ```

                Turbo Pascal
                ```
                uses D1800TP7;    (* Use D1800TP6 for TP ver 6.0 *)
                ...
                wDasErr := K_SetTrigHyst (hAD, 50);
                ```

                Turbo Pascal for Windows
                ```
                {$I DASDECL.INC}
                ...
                wDasErr := K_SetTrigHyst (hAD, 50);
                ```

Visual Basic for Windows

(Include DASDECL.BAS in your program make file)

```
...
wDasErr = K_SetTrigHyst (hAD, 50)
```

BASIC

```
' $INCLUDE: 'DASDECL.BI'
...
wDasErr = KSetTrigHyst% (hAD, 50)
```

A

Error/Status Codes

Table A-1 lists the error/status codes that are returned by the DAS-1800 Series Function Call Driver, possible causes for error conditions, and possible solutions for resolving error conditions.

If you cannot resolve an error condition, contact the Keithley Applications Engineering Department.

Table A-1. Error/Status Codes

Error Code		Cause	Solution
Hex	Decimal		
0	0	No error has been detected.	Status only; no action is necessary.
6000	24576	**Error in configuration file:** The configuration file you specified in the driver initialization function is corrupt, does not exist, or contains one or more undefined keywords.	Check that the file exists at the specified path. Check for illegal keywords in file; you can avoid illegal keywords by using the D1800CFG.EXE utility to create and modify configuration files.
6001	24577	**Illegal base address in configuration file**: The base address specified in the configuration file is invalid.	Use the D1800CFG.EXE utility to change the base address in the configuration file. The address must be on a 16-byte boundary between 200h and 3F0h.
6005	24581	**Illegal Channel Number:** The specified channel is out of range.	Specify a legal channel number. Refer to the table on page 4-15 for valid channel numbers.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6006	24582	**Illegal gain code**: The gain code specified for an analog input operation is out of range.	Specify a legal gain code. Refer to Table 2-2 on page 2-10 for valid gain codes.
6007	24583	**Illegal DMA address**: A FCD function specified an invalid address for the buffer required in a DMA analog input operation.	Specify a valid DMA buffer address.
6008	24584	**Illegal number in configuration file**: The configuration file contains a numeric value that is not in the correct format.	Check all numeric entries in the configuration file; make sure that &H precedes hexadecimal numbers. Use the D1800CFG.EXE utility to modify the configuration file.
600A	24586	**Configuration file not found**: The driver cannot find the configuration file specified as an argument to the driver initialization function.	Check that the file exists at the specified path; check that the file name is spelled correctly in the driver initialization function parameter list.
600B	24587	**Error returning DMA buffer**: DOS returned an error in INT 21H function 49H during the execution of **K_DMAFree**.	Check that the memory handle passed as an argument to **K_DMAFree** was previously obtained via **K_DMAAlloc**.
600C	24588	**Error returning interrupt buffer**: The memory handle specified in K_IntFree is invalid.	Check the memory handle stored by **K_IntAlloc** and make sure that it was not modified.
600D	24589	**Illegal frame handle**: The specified frame handle is not valid for this operation.	Check that the frame handle exists. Check that you are using the appropriate frame handle.
600E	24590	**No more frame handles**: No frames are left in the pool of available frames.	Use **K_FreeFrame** to free a frame that the application is no longer using.
600F	24591	**Requested buffer size too large**: The number of samples specified in K_IntAlloc is too large.	Specify a smaller number of samples; the number of samples must be in the range 1 to 65,536.

Table A-1. Error/Status Codes (cont.)

| Error Code | | Cause | Solution |
Hex	Decimal		
6010	24592	**Cannot allocate interrupt buffer:** (Windows-based languages only) K_IntAlloc failed because there was not enough available DOS memory.	Remove some Terminate and Stay Resident programs (TSRs) that are no longer needed.
6012	24594	**Interrupt buffer deallocation error:** (Windows-based languages only) An error occurred when K_IntFree attempted to free a memory handle.	Remove some Terminate and Stay Resident programs (TSRs) that are no longer needed.
6015	24597	**DMA Buffer too large:** The number of samples specified in K_DMAAlloc is too large.	Specify a smaller number of samples; the number of samples must be in the range 1 to 65,536.
602B	24619	**Not enough memory to accommodate request:** The number of samples you requested in the Keithley Memory Manager is greater than the largest contiguous block available in the reserved heap.	Specify a smaller number of samples; free a previously allocated buffer; use the KMMSETUP utility to expand the reserved heap.
602C	24620	**Requested buffer size > 65536:** The number of samples you requested from the Keithley Memory Manager is greater than 65,536.	Specify a value between 1 and 65,536 when calling K_DMAAlloc in Windows enhanced mode.
602D	24621	**Illegal device handle:** A bad device handle was passed to a function such as **K_GetADFrame**. The handle used was not initialized through a call to **K_GetDevHandle** or **DAS1800_GetDevHandle**, or it was corrupted by your program.	Check device handle value.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	**Decimal**		
602E	24622	**Dynamic memory block destroyed**: An illegal option was specified to a function that accepts a user option, such as **K_SetDITrig**.	Check the option value passed to the function where the error occurred.
6030	24624	**DMA word-page wrap**: During **K_DMAAlloc**, a DMA word-page wrap condition occurred and the allocation attempt failed since there is not enough free memory to accommodate the allocation request.	Reduce the number of samples and retry. If in Windows enhanced mode, install and configure VDMAD.386.
6031	24625	**Requested buffer size exceeds maximum**: A bad memory handle was passed to **K_IntFree** or **K_DMAFree**. The handle used was not initialized through a call to **K_IntAlloc** or **K_DMAAlloc** or was corrupted by you program.	Restart your program and monitor the memory handle value(s).
6032	24626	**Out of memory handles**: An attempt to allocate a memory block using **K_IntAlloc** or **K_DMAAlloc** failed because the maximum number of handles (50) has already been assigned.	Use **K_IntFree** or **K_DMAFree** to free previously allocated memory blocks before allocating again.
6033	24627	**Illegal interrupt setup**: You have requested multiple buffers whose aggregate size is less than 512 samples for an interrupt-mode acquisition operation.	Use a single buffer for operations in which less than 512 samples are acquired.

Table A-1. Error/Status Codes (cont.)

| Error Code | | Cause | Solution |
Hex	Decimal		
6034	24628	**Memory corrupted:** Int 21H function 48H, used to allocate a memory block from the DOS far heap, returned the DOS error 7; memory corrupted. It is likely that you stored (through a DMA-mode or interrupt-mode operation) data into an illegal area of the DOS memory.	Recheck the parameters set by **K_DMAAlloc** and **K_SetDMABuf**. If fatal system error; restart your computer.
6035	24629	**Driver in use:** The driver attempted to configure a device that had already been configured by a call to **K_OpenDriver** (this can occur since, under Windows, it is possible to open the same driver from multiple programs that are running simultaneously).	The driver for a particular device should be configured only once during a single Windows session. If the driver has already been configured, pass a null string as the second argument to **K_OpenDriver**.
6036	24630	**Illegal driver handle:** The specified driver handle is not valid.	Someone may have closed the driver; if so, use **K_OpenDriver** to reopen the driver with the desired driver handle. Try again using another driver handle.
6037	24631	**Driver not found:** The specified driver cannot be found.	Check your link statement to make sure the specified driver is included. Make sure that the device name string is entered correctly in **K_OpenDriver**.
7000	28672	**No board name:** The driver initialization function did not find a board name in the specified configuration file.	Specify a legal board name in the configuration file.
7001	28673	**Bad board name:** The board name in the specified configuration file is illegal.	Specify a legal board name in the configuration file.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
7002	28674	**Bad board number**: The driver initialization function found an illegal board number in the specified configuration file.	Specify a legal board number: 0, 1, or 2
7003	28675	**Bad base address**: The driver initialization function found an illegal base address in the specified configuration file.	Specify a base address in the inclusive range &H200 (512) to &H3F0 (1008) in increments of 10H (16). Make sure that &H precedes hexadecimal numbers.
7004	28676	**Bad DMA channel**: The driver initialization function found an illegal DMA channel in the specified configuration file.	Specify a legal DMA channel: 5, 6, 7, 5+6, 6+7, or 7+5
7005	28677	**Bad interrupt level**: The driver initialization function found an illegal interrupt level in the specified configuration file.	Specify a legal interrupt level: 3, 5, 7, 10, 11, or 15
7007	28679	**Bad A/D channel mode**: The driver initialization function found an illegal input range type in the specified configuration file.	Specify a legal input range type: bipolar, unipolar
7008	28680	**Bad A/D channel configuration**: The driver initialization function found an illegal input configuration in the specified configuration file.	Specify a legal input configuration: single-ended, differential
700A	28682	**Bad number of SSH8**: The number of SSH-8s in the configuration file is not valid.	Run D1800CFG.EXE and specify the number of SSH-8s as a number in the range 0 to 8.
700B	28683	**Bad SSH-8 channel**: The SSH-8 channel in the configuration file is not valid.	Run D1800CFG.EXE and specify the SSH-8 channel as a number in the range 0 to 7.
700C	28684	**Bad SSH-8 gain**: The SSH-8 channel gain in the configuration file is not valid.	Run D1800CFG.EXE and specify the SSH-8 channel gain as 0.5, 5, 50, or 250.

Error/Status Codes

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
700E	28686	**Error - Resource busy:** The application program attempted to start an operation while a similar operation was in progress.	Use **K_IntStop** or **K_DMAStop** to stop the in-progress operation before initiating the second operation.
700F	28687	**Unknown error number:** The error number passed to **K_GetErrMsg** was invalid.	Check the error number passed to **K_GetErrMsg**.
7012	28690	**Bad burst divider:** The burst rate divider passed to **K_SetBurstTicks** is out of range.	Specify a burst rate divider in the range 3 to 255.
7013	28691	**Error - DMA channel busy:** The application program attempted to start a DMA-mode analog input operation while another DMA-mode analog input operation was active.	Use **K_DMAStop** to stop the active operation before initiating the second operation.
7014	28692	**Error - Counter 0 busy:** The application program attempted to start a DMA-mode analog input operation with about-trigger mode enabled while another DMA-mode with about-trigger operation was active.	Use **K_DMAStop** to stop the active operation before initiating the second operation.
7015	28693	**Error - About count illegal:** The number of samples passed to **K_SetAboutTrig** is out of range.	Specify a number of samples in the range 1 to 65,536.
7016	28694	**Error - About trigger illegal:** About-trigger mode was enabled for an interrupt-mode operation.	Disable about-trigger mode (about-trigger mode is available for DMA-mode analog input operations only).
7017	28695	**Illegal number of EXP-1800:** The number of EXP-1800 expansion boards specified in the configuration file is not valid.	Run D1800CFG.EXE and specify the number of EXP-1800 expansion boards as a number in the range 0 to 16.

Table A-1. Error/Status Codes (cont.)

| Error Code | | Cause | Solution |
Hex	Decimal		
8001	32769	**Function not supported**: You have attempted to use a function not supported by the DAS-1800 Series Function Call Driver.	Contact the Keithley Applications Engineering Department.
8003	32771	**Illegal board number**: An illegal board number was specified in the board initialization function.	Specify a legal board number: 0, 1, or 2.
8004	32772	**Illegal error number**: The error message number specified in K_GetErrMsg is invalid.	Check the error message number and try again.
8005	32773	**Board not found at configured address**: The board initialization function does not detect the presence of a board.	Make sure that the base address setting of the switches on the board matches the base address setting in the configuration file.
800B	32779	**Conversion overrun**: Data was overwritten before it was transferred to the computer's memory.	Adjust the clock source to slow down the rate at which the board acquires data. Remove other application programs that are running and using computer resources.
8016	32790	**Interrupt overrun**: During an interrupt-mode analog output or digital I/O operation, an interrupt was detected from a DAS-1800 Series board while the software was servicing a previous interrupt from the same board.	Use **K_SetClkRate** to reduce the pacer clock rate. Analog output and digital I/O operations are limited to the following throughputs: 5 kHz in DOS and Windows Standard mode; 1 kHz in Windows enhanced mode (the throughputs listed are approximate; they are limited by the PC's resources and Windows setup).
801A	32794	**Interrupts already active**: You have attempted to start an operation whose interrupt level is being used by another system resource.	Use **K_IntStop** to stop the first operation before starting the second operation.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
801B	32795	**DMA already active:** You attempted to start an DMA-mode analog input operation with **K_DMAStart** while another was already in progress.	Use **K_DMAStop** to stop the first operation before starting the second operation.
8020	32800	**FIFO Overflow event detected:** During a DMA-mode or interrupt-mode input operation, the onboard data FIFO overflowed; the hardware automatically stopped the acquisition.	The conversion rate you are using is too fast for the operating environment you are in. Use **K_SetClkRate** to reduce your conversion rate and/or reconfigure your board to use dual-DMA if using DMA mode (run D1800CFG.EXE and restart your program).
FFFF	65535	**User aborted operation**	You pressed **[Ctrl]+[Break]** while waiting for an analog trigger event to occur.

B

Data Formats

The DAS-1800 Series Function Call Driver can read and write raw counts only. When reading a value (as in **K_ADRead**), you may want to convert the raw count to a more meaningful voltage value; when writing a value (as in **K_SetTrigHyst**), you must convert the voltage value to a raw count.

The remainder of this appendix contains instructions for converting raw counts to voltage and for converting voltage to raw counts.

Converting Raw Counts to Voltage

You may want to convert raw counts to voltage when reading an analog input value or when reading the analog trigger level or hysteresis value.

To convert an analog input value to a voltage, use one of the following equations, where *count* is the count value, and *span* is the appropriate value from Table B-1 on page B-2:

$$Voltage = \frac{count \times span}{4096} \quad \text{(DAS-1800HC/ST Series boards)}$$

$$Voltage = \frac{count \times span}{65,536} \quad \text{(DAS-1800HR Series boards)}$$

Table B-1. Span Values For Data Conversion Equations

Board	Input Range Type	Gain	Input Range	Span (V)
DAS-1801HC DAS-1801ST	Unipolar	1	0 to 5 V	5
		5	0 to 1 V	1
		50	0 to 100 mV	0.1
		250	0 to 20 mV	0.02
	Bipolar	1	−5 to 5 V	10
		5	−1 to 1 V	2
		50	−100 to 100 mV	0.2
		250	−20 to 20 mV	0.04
DAS-1802HC DAS-1802ST DAS-1802HR	Unipolar	1	0 to 10 V	10
		2	0 to 5 V	5
		4	0 to 2.5 V	2.5
		8	0 to 1.25 V	1.25
	Bipolar	1	−10 to 10 V	20
		2	−5 to 5 V	10
		4	−2.5 to 2.5 V	5
		8	−1.25 to 1.25 V	2.5

For example, assume that you want to read analog input data from a channel on a DAS-1801HC board configured for unipolar input range type; the channel collects the data at a gain of 1. The count value is 3072. The voltage is determined as follows:

$$\frac{3072 \times 5 \text{ V}}{4096} = 3.75 \text{ V}$$

Data Formats

As another example, assume that you want to read analog input data from a channel on a DAS-1802HC board configured for a bipolar input range type; the channel collects the data at a gain of 2. The count value is 1024. The voltage is determined as follows:

$$\frac{1024 \times 10 \text{ V}}{4096} = 2.5 \text{ V}$$

Converting Voltage to Raw Counts

You must convert voltage to raw counts when specifying an analog output value, analog trigger level or hysteresis value.

Specifying an Analog Output Value (DAS-1800HC Series only)

To convert a voltage value to a raw count when specifying an analog output value, use the following equation, where *voltage* is the desired voltage:

$$\text{Count} = \frac{\text{voltage} \times 4096}{20 \text{ V}} + 2048$$

For example, assume that you want to specify an analog output value of 5 V for a channel on a DAS-1802HC. The raw count is determined as follows:

$$\frac{5 \text{ V} \times 4096}{20 \text{ V}} + 2048 = 3072$$

Specifying an Analog Trigger Level

To convert a voltage value to a raw count when specifying an analog trigger level, use one of the following equations, where V_{trig} is the desired voltage, and *span* is the appropriate value from Table B-1 on page B-2:

$$\text{Count} = \frac{V_{trig} \times 4096}{\text{span}} \quad \text{(DAS-1800HC/ST Series boards)}$$

$$\text{Count} = \frac{V_{trig} \times 65536}{\text{span}} \quad \text{(DAS-1800HR Series boards)}$$

Note: When converting voltage to raw counts to specify an analog trigger level, always use a gain of 1 to determine which span value to use from Table B-1, no matter what the gain of the channel is.

For example, assume that you want to specify an analog trigger level of 2.5 V for a channel on a DAS-1801HC board configured for a bipolar input range type. The raw count is determined as follows:

$$\frac{2.5 \text{ V} \times 4096}{10 \text{ V}} = 1024$$

Specifying a Hysteresis Value

To convert a voltage value to a raw count when specifying a hysteresis value, use one of the following equations, where V_{hyst} is the desired voltage, and *span* is the appropriate value from Table B-1 on page B-2:

$$\text{Count} = \frac{V_{hyst} \times 4096}{\text{span}} \qquad \text{(DAS-1800HC/ST Series boards)}$$

$$\text{Count} = \frac{V_{hyst} \times 65536}{\text{span}} \qquad \text{(DAS-1800HR Series boards)}$$

Note: When converting voltage to raw counts to specify a hysteresis value, always use a gain of 1 to determine which span value to use from Table B-1, no matter what the gain of the channel is.

For example, assume that you want to specify an analog trigger hysteresis value of 0.5 V for a channel on a DAS-1801HC board configured for a bipolar input range type. The raw count is determined as follows:

$$\frac{1.25 \text{ V} \times 4096}{10 \text{ V}} = 512$$

.

Index

F

files required
    Borland C/C++ 3-29
    Microsoft C/C++ 3-28
    Professional Basic 3-53, 3-55
    QuickBasic (Version 4.0) 3-51
    QuickBasic (Version 4.5) 3-52
    QuickC for Windows 3-30
    Turbo Pascal 3-38
    Turbo Pascal for Windows 3-39
    Visual Basic for Windows 3-45
    Visual C++ 3-31
frame management functions 4-2
frames 3-2
    frame elements 3-2
    frame handle 3-2
    frame types 3-3
Function Call Driver
    initialization 2-2
    structure 3-1
functions
    buffer address 4-1
    buffering mode 4-1
    channel and gain 4-1
    clock 4-1
    conversion mode 4-1
    frame management 4-1
    gate 4-1
    initialization 4-1
    memory management 4-1
    miscellaneous 4-1
    operation 4-1
    trigger 4-1

G

gain codes 2-10
gains 2-10
    see also analog input ranges 2-9
gains: *see* Analog input ranges
gate functions 4-5
gates 2-25
group of consecutive channels 2-13

H

hardware gates: *see* gates
hysteresis 2-21

I

initialization functions 4-2
initializing a board 2-2
initializing the driver 2-2
input range type 2-9
internal pacer clock 2-16, 2-29, 2-36
interrupt mode
    analog input operations 2-5
    analog output operations 2-27
    digital I/O operations 2-32

K

K_ADRead 2-5, 2-13, 2-27, 2-28, 4-14
K_BufListAdd 2-9, 4-17
K_BufListReset 2-9, 4-21
K_ClearFrame 3-4, 4-23
K_CloseDriver 2-2, 4-25
K_ClrAboutTrig 4-27
K_ClrADFreeRun 4-29
K_ClrContRun 4-31
K_DASDevInit 2-3, 4-33

V