DAS-1800AO Series LabVIEW® VI Driver

USER'S GUIDE

DAS-1800AO Series LabVIEW® VI Driver User's Guide

Revision A – October 1994
Part Number: 93180

New Contact Information

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, OH 44139

Technical Support: 1-888-KEITHLEY
Monday – Friday 8:00 a.m. to 5:00 p.m (EST)
Fax: (440) 248-6168

Visit our website at http://www.keithley.com

Keithley MetraByte Division

Keithley Instruments, Inc.

440 Myles Standish Blvd. Taunton, MA 02780

Telephone: (508) 880-3000 • FAX: (508) 880-0179

Preface

The *DAS-1800AO Series LabVIEW® VI Driver User's Guide* explains how to write LabVIEW application programs for DAS-1800AO Series boards using the Keithley MetraByte DAS-1800 Series VI Driver.

This manual is intended for LabVIEW application programmers using a DAS-1800AO Series board in an IBM® PC AT® or compatible computer. It is assumed that users have read the user's guide for the board and are familiar with the board's features, and that they have completed the appropriate hardware installation and configuration. It is further assumed that users are experienced in programming in LabVIEW and are familiar with Windows™ and with data acquisition principles.

Manual Organization

The manual is organized as follows:

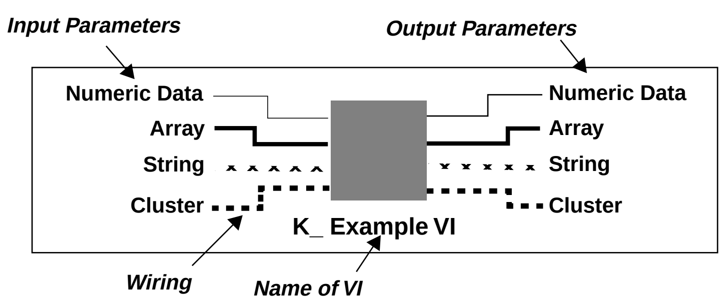
- Chapter 1 explains how to install the DAS-1800 Series VI Driver and how to get help, if necessary.
- Chapter 2 contains the background information needed to use the VIs included in the DAS-1800 Series VI Driver.
- Chapter 3 provides guidelines for using the DAS-1800 Series VIs.
- Chapter 4 contains detailed descriptions of the DAS-1800 Series VIs, arranged in alphabetical order.
- Appendix A describes the error codes returned by DAS-1800 Series VIs.
- Appendix B provides instructions for converting raw counts to voltage and for converting voltage to raw counts.

An index completes the manual.

Conventions Used in this Manual

The following conventions are used throughout this manual:

- References to DAS-1800AO Series boards apply to the DAS-1801AO board and the DAS-1802AO board.
- All VIs supported by the DAS-1800 Series VI Driver are illustrated graphically, as shown in the example below. The name of the VI is shown beneath the DAS-1800 icon; the wires connecting the inputs to and the outputs from the DAS-1800 icon represent the data type of the parameters.



- The data types of the inputs and outputs are represented as follows:

Inputs	Outputs	Data Type
I16	I16	Signed 16-bit integer
I32	I32	Signed 32-bit integer
U8	U8	Unsigned 8-bit integer
U16	U16	Unsigned 16-bit integer
U32	U32	Unsigned 32-bit integer
[I16]	[I16]	Array of signed 16-bit integers
[U16]	[U16]	Array of unsigned 16-bit integers
[U32]	[U32]	Array of unsigned 32-bit integers
		Cluster
a b c	a b c	String
T F	T F	Boolean

Related Documents

For more information, refer to the following documents:

- *DAS-1800AO Series User's Guide*
- LabVIEW manuals

Table of Contents

3 Programming with the VI Driver

4 VI Reference

List of Figures

List of Tables

1

Getting Started

The DAS-1800 Series VI Driver is a library of data acquisition and control VIs (Virtual Instruments) used to write application programs for DAS-1800AO Series data acquisition boards.

This chapter describes how to install the DAS-1800 Series VI Driver and how to get help, if required.

Installing the VI Driver

To install the DAS-1800 Series VI Driver, perform the following procedure:

1. Insert the VI Driver disk into the appropriate disk drive of your computer.
2. Enter Windows.
3. From the Program Manager File menu, select Run.
4. Assuming you are using drive A, type the following command line in the Run dialog box:

   ```
   A:SETUP
   ```

5. Select OK.
6. Respond to the installation prompts as appropriate.

The program creates a Program Manager setup group called KEITHLEY DAS-1800 VI Driver. This group contains files for the VI driver, utilities, and example programs using the DAS-1800 Series VIs.

Once you have installed the DAS-1800 Series VI Driver, install your DAS-1800AO Series board and its software, run the Keithley Memory Manager utility, and run the configuration program. Refer to the user's guide for your board for the information required to perform these steps.

The above steps must be completed in order to open the VI Driver example programs. You can open LabVIEW from the Program Manager group by opening a VI Driver example program.

After installation, you may want to review the following files:

- Readme.Txt - An ASCII file containing information available after the publication of this manual.
- Files.Txt - An ASCII file that describes all of the files available.

Getting Help

If you need help installing or using the DAS-1800 Series VI Driver, call your local sales office or the Keithley Metrabyte Applications Engineering Department at:

(508) 880-3000

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:

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	Windows version	______________________
	Windows mode	______________________
LabVIEW Package	Version	______________________
Accessories	Type	______________________
	Type	______________________
	Type	______________________
	Type	______________________
	Type	______________________
	Type	______________________
	Type	______________________
	Type	______________________

2

Available Operations

This chapter contains the background information you need to use the VIs to perform operations on DAS-1800AO 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 2-23
Digital input and output (I/O)	page 2-34

System Operations

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

Initializing the Driver

You must initialize the DAS-1800 Series VI Driver and any other Keithley DAS VI Drivers you are using in your application program. To initialize the drivers, use **K_OpenDriver**. You specify the configuration file that defines this particular use of the driver. The driver returns a unique identifier for the particular use of the driver; this identifier is called the driver handle. A maximum of 30 driver handles can be specified for all the Keithley MetraByte 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 all 30 driver handles have been used, you can use **K_CloseDriver** 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 VI Driver, the driver is shut down.

Initializing a Board

The DAS-1800 Series VI Driver supports up to three boards. You must use **K_GetDevHandle** to specify the boards you want to use. The driver returns a unique identifier for each board; this identifier is called the board handle. Board handles allow you to communicate with more than one board. In subsequent VIs related to the board, you use the board handle returned by **K_GetDevHandle**. A maximum of 30 board handles can be specified 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 all 30 board handles have been used, you can use **K_FreeDevHandle** to free a board handle.

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

- Aborts all operations currently in progress that are associated with the board identified by the board handle.
- Verifies that the board identified by the board handle is the board specified in the configuration file.

Retrieving Revision Levels

If you are having problems with your application program, you may want to verify which versions of the VI Driver, Keithley DAS Driver Specification, and Keithley DAS Shell are installed on your board. **K_GetVer** allows you to get both the revision number of the DAS-1800 Series VI Driver and the revision number of the Keithley DAS Driver Specification to which the driver conforms. **K_GetShellVer** allows you to get the revision number of the Keithley DAS Shell (the Keithley DAS Shell is a group of VIs that are shared by all DAS boards).

Handling Errors

Error information is passed from one VI to the next in your application program. You must first create an error cluster, which consists of three variables:

- A Boolean error status (True/False: True = error)
- A numeric error code for the number of the error, if an error occurred (0 = no error, nonzero = error occurred)
- A string for the name of the VI (error source) that returned the error, if an error occurred

You then wire the cluster to each VI in your program, normally starting with **K_OpenDriver.** When the program begins, the first VI checks the error status; if the status is False (no error), the VI runs. When it has finished, the VI sets the error status. If an error occurred during the execution of the VI, the error status is set to True, the error code is set to a nonzero value identifying the error, and the error source is set to the name of the VI that caused the error. The next VI in the program reads the error status; if it finds that the error status is True, the VI does not execute. All VIs remaining in the program do likewise.

You can read the error information by placing an Unbundle by Name function after a VI (normally the last VI in your program, **K_CloseDriver**). You create a variable for each element in the error cluster; once the variables are wired to the Unbundle by Name cluster, the error information is displayed there.

Appendix A contains a complete list of error codes and their descriptions.

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, conversion mode, clock source, buffering mode, trigger source, and hardware gate.

Operation Mode

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 **K_ADRead** to start an analog input operation in single mode. You specify the board you want to use, the analog input channel, and the gain code for the gain at which you want to read the signal.

- **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 A/D 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 **K_IntStart** to start an analog input operation in interrupt mode.

  You can specify either single-cycle or continuous buffering mode for interrupt-mode operations. Refer to page 2-16 for more information on buffering modes. Use **K_IntStop** to stop an interrupt-mode operation. Use **K_IntStatus** 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 A/D FIFO 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-1800AO Series User's Guide* for more information.

  Use **K_DMAStart** to start an analog input operation in DMA mode.

  You can specify either single-cycle or continuous buffering mode for DMA-mode operations. Refer to page 2-16 for more information on buffering modes. Use **K_DMAStop** to stop a continuous-mode DMA operation. Use **K_DMAStatus** to determine the current status of a DMA operation.

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

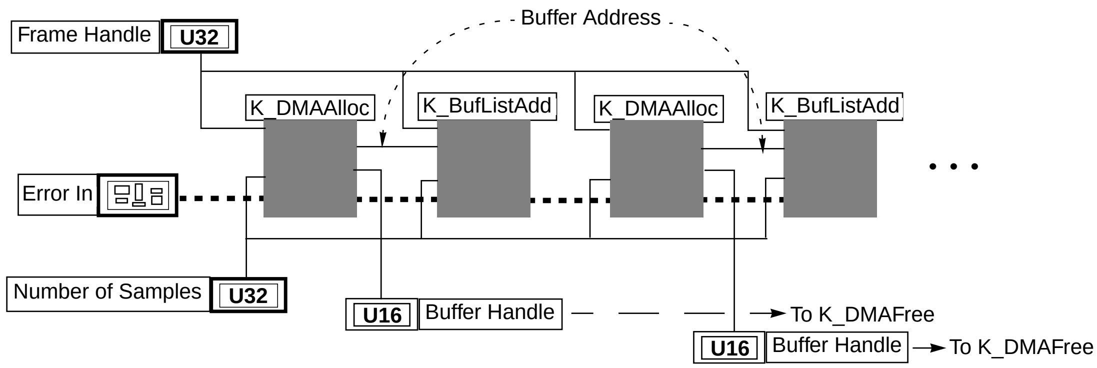
Memory Allocation and Management

Interrupt-mode and DMA-mode analog input operations require memory buffers in which to store the acquired data. You can reserve a single buffer, or you can reserve multiple buffers (up to a maximum of 150) to increase the number of samples you can acquire. Buffers must be dynamically allocated outside of your application program's memory area.

Use **K_IntAlloc** to allocate memory dynamically for interrupt-mode operations; use **K_DMAAlloc** to allocate memory dynamically for DMA-mode operations. You specify the operation requiring the buffer and the number of samples to store in the buffer (up to 65,536). The driver returns the starting address of the buffer and a unique identifier for the buffer; this identifier is called the buffer handle.

To assign the starting address of a buffer and the number of samples in the buffer, use **K_SetBuf** for interrupt operations or **K_SetDMABuf** for DMA operations. If you are using multiple buffers, use **K_BufListAdd** to add each buffer to the list of multiple buffers associated with each operation. To move the contents of an allocated buffer to a LabVIEW array, use **K_MoveBufToArray**.

The following example shows how to allocate multiple buffers using **K_DMAAlloc** and **K_BufListAdd**. For each **K_DMAAlloc** VI used, you use the **K_BufListAdd** VI to add the allocated buffer to the list of buffers. The example is illustrated in DMA mode; interrupt mode is identical except that you use the appropriate interrupt-mode VIs. Refer to the examples on disk for more information.



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 enough buffers and large enough buffers. Refer to your DAS-1800 Series board user's guide for more information about the Keithley Memory Manager.

When a buffer is no longer required, you can free its memory for another use by specifying the buffer handle in **K_IntFree** for interrupt-mode operations or in **K_DMAFree** for DMA-mode operations.

Gains and Ranges

Each analog input channel on a DAS-1800AO Series board can measure 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-1800AO Series User's Guide* for more information about analog input ranges.

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

Table 2-2. Analog Input Ranges and Gains

Boards	Analog Input Range		Gain	Gain Code
	Bipolar	Unipolar		
DAS-1801AO	±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-1802AO	±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-1801AO with EXP-1800 attached	±100 mV	0 to 100 mV	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

Table 2-2. Analog Input Ranges and Gains (cont.)

Boards	Analog Input Range		Gain	Gain Code
	Bipolar	Unipolar		
DAS-1802AO with EXP-1800 attached	±200 mV	0 to 200 mV	50	4
	±100 mV	0 to 100 mV	100	5
	±50 mV	0 to 50 mV	200	6
	±25 mV	0 to 25 mV	400	7

Channels

DAS-1800AO Series boards are configured with either 16 onboard single-ended or eight onboard differential analog input channels. You can increase the number of channels to 256 single-ended channels using EXP-1800 expansion boards, described in the next section.

The input channel configuration (differential or single-ended) 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 boards configured for single-ended input.

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 channels you are using.

Specifying Channels When Using EXP-1800 Expansion Boards

To increase the number of analog input channels, you can attach up to 16 EXP-1800 expansion boards to the DAS-1800AO 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-1800AO channels 0 to *N-1*. Refer to the user's guide for information on connecting EXP-1800 boards to DAS-1800AO Series boards.

The analog input channel connections on a DAS-1800AO Series board or EXP-1800 board are designated with numbers from 0 to 15. These numbers are the *physical channel numbers*. If a system includes a DAS-1800AO Series board and one or more EXP-1800s, then that system contains duplicate physical channel numbers. To uniquely identify a physical channel, the VI Driver uses a scheme of *logical channel numbers*. The *channel#* argument for any VI 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-1800AO Series board:

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

If the physical channel is on an EXP-1800:

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

where

NumEXPs is an integer from 0 to 16 that identifies the number of EXP-1800s connected to the DAS-1800AO Series board, and

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

For example, consider the system illustrated in Figure 2-1, in which three EXP-1800s are connected to a DAS-1801AO.

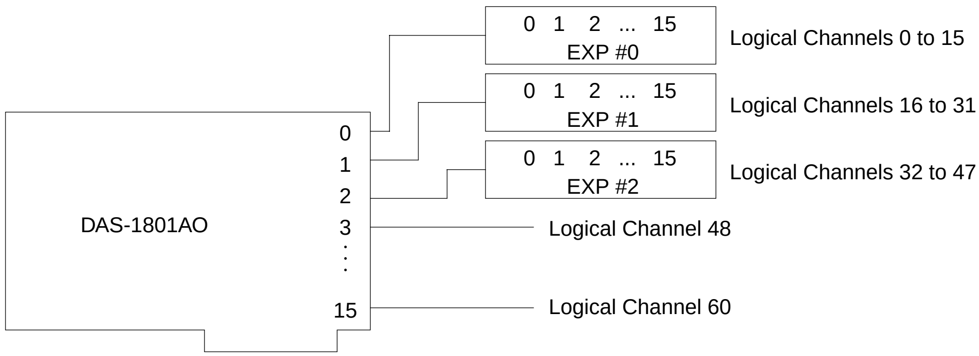


Figure 2-1. Example of Logical Channel Assignments

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

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

The logical channel that identifies channel 15 on the third EXP-1800 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 **K_ADRead** 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 **K_SetChn** to specify the channel and **K_SetG** 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 **K_SetStartStopChn** 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 single-ended input configuration with no expansion boards, the start channel is 15, the stop channel is 2, and you want to acquire five samples. Your program reads data first from channel 15, then from channels 0, 1, and 2, and finally from channel 15 again.

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

Refer to Table 2-2 on page 2-7 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 Array

For interrupt-mode and DMA-mode analog input operations, you can acquire samples from channels in a hardware channel-gain queue. You create an array and 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 the channels in the channel-gain array in consecutive order or in nonconsecutive order. You can also specify the same channel more than once. The channel gain array can contain up to 256 entries.

The channels are sampled in order from the first channel specified in the array to the last channel specified in the array; the channels in the array are then sampled again until the specified number of samples is read.

For example, assume you want to sample channels 0, 5, and 3. Channel 0 uses a gain code of 1, channel 5 uses a gain code of 2 and channel 3 uses a gain code of 3. Your array would look like this:

# of Entries	Chan	Gain Code	Chan	Gain Code	Chan	Gain Code
3	0	1	5	2	3	3

where the first element is the number of entries and the remaining pairs of elements represent the channel to read and its associated gain code.

After you create the channel-gain array, you allocate space for the channel-gain array in your program using **K_AllocChnGAry**; you initialize the channel-gain array using **K_FormatChnGAry**; you set the frame element for the channel-gain array using **K_SetChnGAry**. When the operation is finished with the channel-gain array, you can free its space using **K_FreeChnGAry**.

Refer to Table 2-2 on page 2-7 for a list of the analog input ranges supported by DAS-1800AO Series boards and the gain code associated with each range.

Conversion Mode

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 **K_SetADFreeRun** to specify burst mode.

  Use burst mode with SSH (sample-and-hold) if you want to simultaneously sample all channels in a scan using the SSH-8 accessory board. Use **K_SetSSH** 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-1800AO Series User's Guide* for more information about conversion modes.

Clock Source

DAS-1800AO Series boards provide two clock sources for analog input operations: an A/D 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 A/D pacer clock; when performing interrupt-mode and DMA-mode analog input operations in burst mode and burst mode with SSH, you use both the A/D pacer clock and the burst mode conversion clock. These clock sources are described in the following subsections.

A/D Pacer Clock

In paced mode, the A/D 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 A/D pacer clock determines the period between the conversions of one scan and the conversions of the next scan. Use **K_SetClk** to specify an internal or an external A/D pacer clock source. The internal A/D pacer clock is the default pacer clock.

The internal and external A/D pacer clocks are described as follows:

- **Internal A/D pacer clock -** The internal A/D 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 **K_SetClkRate** 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).

  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 the internal A/D pacer clock, use the following formula to determine the number of counts to specify:

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

  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 A/D pacer clock -** You connect an external pacer clock to the XPCLK pin (pin 44) on the board's main I/O connector. 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 **K_SetExtClkEdge** 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 issues. If you are using an external pacer clock for analog input operations, make sure that the clock initiates conversions at a rate that the ADC can handle.

Refer to your *DAS-1800AO Series 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 **K_SetBurstTicks** 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 63 clock ticks. The period between conversions ranges from 3 µs to 63 µs.

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

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

For example, if you want a burst mode conversion rate of 20 ksamples/s, specify 50 clock ticks, as shown in the following equation:

$$\frac{1,000,000}{20,000} = 50$$

Refer to your *DAS-1800AO Series User's Guide* for more information about the burst mode conversion clock.

Buffering Mode

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 the process is stopped; any values already stored in the buffer are overwritten. Use **K_SetContRun** to specify continuous buffering mode.

Trigger

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 uses only 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 supported trigger sources and post-trigger, pre-trigger, and about-trigger acquisitions are described in the following subsections.

Trigger Source

The VI Driver supports two trigger sources: internal and external. For interrupt-mode and DMA-mode analog input operations, use **K_SetTrig** to specify the trigger source. External triggers can be analog triggers or digital triggers.

The trigger event is not significant until the operation the trigger governs has been started (using **K_DMAStart** or **K_IntStart**). The point at which conversions begin depends on the pacer clock; refer to page 2-13 for more information.

The internal trigger, external analog trigger, and external digital trigger are described as follows:

- **Internal trigger -** An internal trigger is a software trigger. The trigger event occurs immediately after you start the operation. Consequently, **K_DMAStart** or **K_IntStart** is considered the trigger event for an internal trigger. The internal trigger is the default trigger source.
- **External analog trigger -** You can use the signal on any analog input channel as the trigger signal for an analog trigger. Trigger events for analog triggers (illustrated in Figure 2-2) are described as follows:
  - **Positive trigger -** The trigger signal changes from a voltage that is less than the trigger level to a voltage that is greater than the trigger level.
  - **Negative trigger -** The trigger signal changes from a voltage that is greater than the trigger level to a voltage that is less than the trigger level.

Note: Analog triggering is a feature of the VI 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.

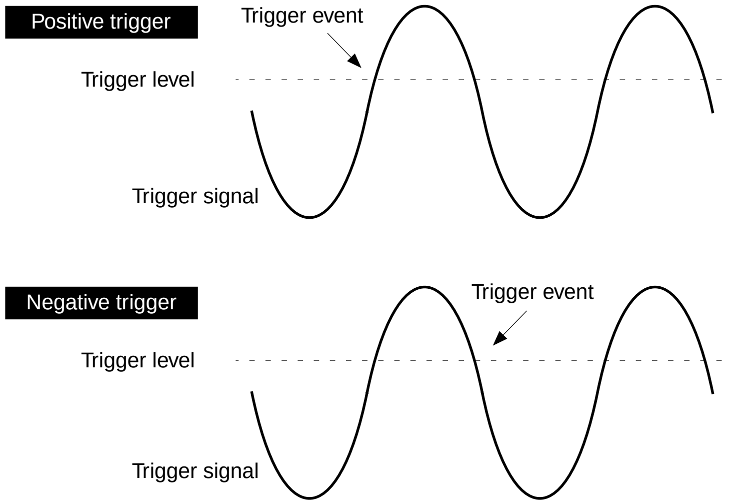


Figure 2-2. Trigger Events for Analog Triggers

Use **K_SetADTrig** to specify the analog input channel to use as the trigger channel, the trigger level, and the trigger polarity (positive or negative).

You specify the trigger level as a raw count value. Refer to Appendix B for information on how to convert a voltage value to a raw count value.

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

The hysteresis value is an absolute number, which you specify as a raw count value between 0 and 4095. When you add the hysteresis value to the trigger level (for a negative trigger) or subtract the hysteresis value from the trigger level (for a positive trigger), the resulting value must also be between 0 and 4095.

For example, assume that you are using a negative trigger on a channel of a board configured for an analog input range of ±5 V. If the trigger 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 trigger 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 trigger occurs; the analog signal must be above +4.1 V and then fall below +4 V before a negative trigger occurs.

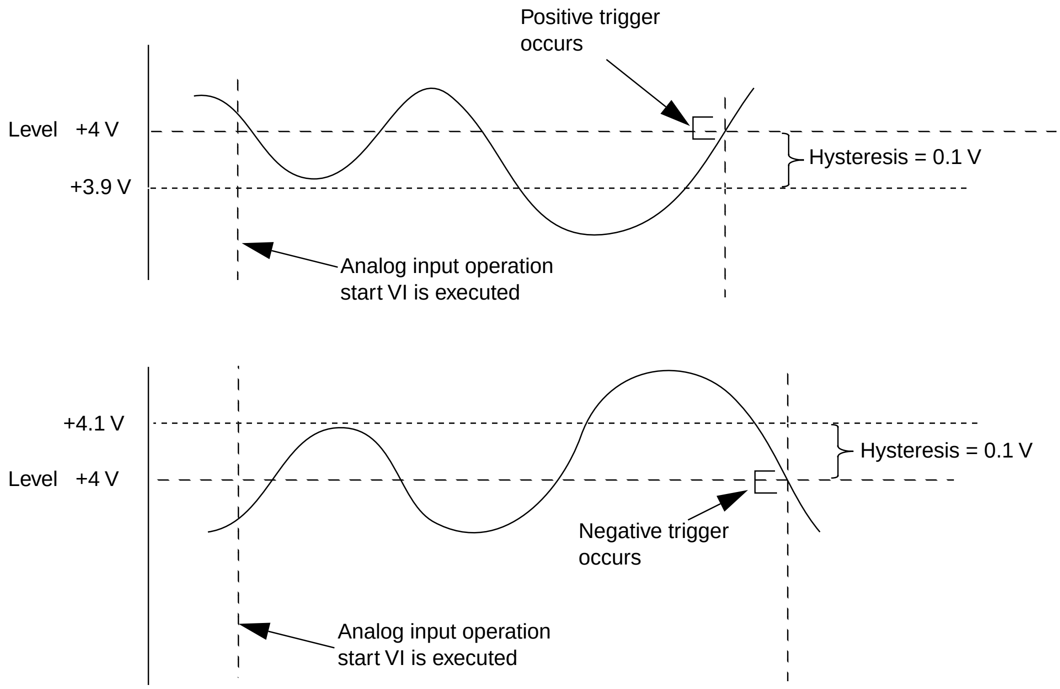


Figure 2-3. Using a Hysteresis Value

- **External digital trigger -** The digital trigger signal is available on the TGIN pin (pin 46) on the board's main I/O connector. Use **K_SetDITrig** to specify whether you want the trigger event to occur on a rising edge (positive polarity) or a falling edge (negative polarity). These trigger events are illustrated in Figure 2-4.

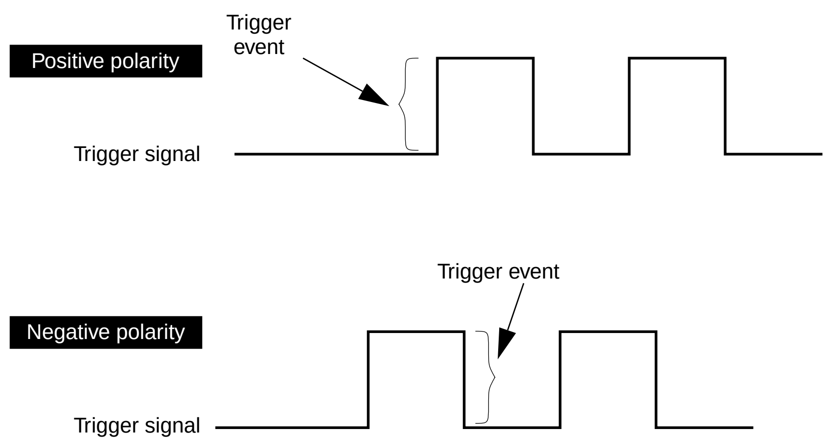


Figure 2-4. Trigger Events For Digital Triggers

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 **K_DMAStop** or **K_IntStop**.

To specify post-trigger acquisition, use the following VIs:

1. If you want acquisition to continue until you stop it with **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**, or **K_BufListAdd**).
3. Use **K_SetTrig** to specify the trigger source that will start the operation (internal for an internal trigger, external for an analog or digital trigger).
4. If you are using an analog trigger, use **K_SetADTrig** to define the trigger conditions; if you are using a digital trigger, use **K_SetDITrig** 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 VIs:

1. Use **K_SetTrig** to specify the trigger source that will start the operation (internal for an internal trigger, external for an analog or digital trigger).
2. If you are using an analog start trigger, use **K_SetADTrig** to define the trigger conditions; if you are using a digital start trigger, use **K_SetDITrig** 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, use **K_SetDITrig** to specify the active edge for the about trigger. (If the start trigger is digital, then its active edge is 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 VIs:

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 you are using an analog start trigger, use **K_SetADTrig** to define the trigger conditions; if you are using a digital start trigger, use **K_SetDITrig** 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. If the start trigger is not digital, use **K_SetDITrig** to specify the active edge for the about trigger. (If the start trigger is digital, then its active edge is also used for the about trigger).

Hardware Gate

A hardware gate is an externally applied digital signal that determines whether conversions occur. You connect the gate signal to the TGIN pin (pin 46) on the board's main I/O connector. 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 gate signal to TGIN is high; if the gate signal to TGIN is low, conversions are inhibited. If the board is configured with a negative gate, conversions occur only if the gate signal to TGIN is low; if the gate signal to TGIN is high, conversions are inhibited. Use **K_SetGate** 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 **K_SetADTrig**).

Analog Output Operations

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 and gains, clock source, buffering mode, trigger source, and hardware gate.

Operation Mode

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 analog output channel; you cannot perform any other operation until the single-mode operation is complete.

  Use **K_DAWriteGain** to start an analog output operation in single mode. You specify the board you want to use, the analog output channel, the gain code, 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 channels. 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. The hardware temporarily stores the output data in the onboard D/A FIFO and then writes the data using an interrupt service routine. Use **K_IntStart** to start an analog output operation in interrupt mode.

  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 **K_IntStop** to stop an interrupt operation. Use **K_IntStatus** to determine the current status of an interrupt operation.

- **DMA mode -** In DMA mode, the driver writes a single sample or multiple samples to one or both analog output channels. A hardware clock paces the updating of the analog output channels. Once the analog output operation begins, control returns to your application program. You store the values you want to write in a user-defined DMA buffer in the computer. The hardware temporarily stores the output data in the onboard D/A FIFO and then writes the data. Use **K_DMAStart** to start an analog output operation in DMA mode.

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

- **Recycle mode -** In recycle mode, the driver writes a single sample or up to a total of 2048 samples to one or both analog output channels. A hardware clock paces the updating of the analog output channels. 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. The hardware temporarily stores the output data in the onboard D/A FIFO and then writes the data. The data in the D/A FIFO is continuously recycled until the operation is stopped. Use **K_DMAStart** or **K_IntStart** to start an analog output operation in recycle mode.

  If you are performing a recycle mode analog output operation, the board automatically uses the onboard D/A FIFO; the PC's interrupt or DMA resources are not used. In this case, the board attains its highest transfer rate (up to 500 ksamples/s).

You must specify continuous buffering mode for recycle-mode operations. Refer to page 2-30 for more information on buffering modes. Use **K_DMAStop** or **K_IntStop** to stop a recycle-mode operation. Use **K_DMAStatus** or **K_IntStatus** to determine the current status of a recycle-mode 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 and DMA-mode analog output operations require memory buffers in which to store the data to be written to the analog output channels. You can reserve a single buffer, or you can reserve multiple buffers (up to a maximum of 150) to increase the number of samples. Recycle-mode analog output operations require a single memory buffer of no more than 2048 samples. Buffers must be dynamically allocated outside of your application program's memory area.

Use **K_IntAlloc** to allocate memory dynamically for interrupt-mode or recycle-mode operations; use **K_DMAAlloc** to allocate memory dynamically for DMA-mode or recycle-mode operations. You specify the operation requiring the buffer and the number of samples to store in the buffer (up to 65,536). The driver returns the starting address of the buffer and a unique identifier for the buffer; this identifier is called the buffer handle.

To assign the starting address of a buffer and the number of samples in the buffer, use **K_SetBuf** for buffers allocated with **K_IntAlloc** or **K_SetDMABuf** for buffers allocated with **K_DMAAlloc**. If you are using multiple buffers, use **K_BufListAdd** to add each buffer to the list of multiple buffers associated with each operation. Refer to page 2-5 for an example of using multiple buffers. To move the contents of a LabVIEW buffer to an allocated buffer, use **K_MoveArrayToBuf**.

When a buffer is no longer required, you can free it for another use by specifying the buffer handle in **K_IntFree** for buffers allocated with **K_IntAlloc** or in **K_DMAFree** for buffers allocated with **K_DMAAlloc**.

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 enough buffers and large enough buffers. Refer to the *DAS-1800AO Series User's Guide* for more information about the Keithley Memory Manager.

Gains and Ranges

Each analog output channel on a DAS-1800AO Series board can write an analog output signal in one of two software-selectable ranges. Table 2-3 lists the analog output ranges supported by DAS-1800AO Series boards and the gain code associated with each range.

Table 2-3. Analog Output Ranges

Analog Output Range	Gain Code
±5 V	0
±10 V	1

Channels

DAS-1800AO Series boards contain two digital-to-analog converters (DACs), 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. The following subsections explain how to specify the channels.

Writing Values to a Single Channel

For single-mode operations, you can write a single value to a single analog output channel. Use **K_DAWriteGain** to specify the channel and the gain code.

For interrupt-mode, DMA-mode, and recycle-mode operations, you can write a single value or multiple values to a single analog channel. Use **K_SetChn** to specify the channel and **K_SetG** to specify the gain code.

Writing Values to Both Channels Using the Same Gain Code

For interrupt-mode, DMA-mode, and recycle-mode analog output operations, you can write a single value or multiple values to both analog output channels simultaneously when both channels use the same gain code. Use **K_SetStartStopChn** to specify channel 0 as the start channel and channel 1 as the stop channel; use **K_SetG** to specify the gain code for both channels. You can also use **K_SetStartStopG** to specify the start channel, the stop channel, and the gain code in a single VI.

At each pacer clock pulse, two values in the buffer are written simultaneously. The first value is written to channel 0 and the second value is written to channel 1. After all the values in the buffer are written once, the values are written again until the required number of values are written.

Writing Values to Both Channels Using Different Gain Codes

For interrupt-mode, DMA-mode, and recycle-mode analog output operations, you can write a single value or multiple values to both analog output channels simultaneously when each channel uses a different gain code. Both channels are updated simultaneously until the specified number of values is written.

To specify one gain code for channel 0 and another gain code for channel 1, create a two-entry channel-gain array with channel 0 and its gain code as the first channel-gain pair and channel 1 and its gain code as the second channel-gain pair. For example, assume you want channel 0 configured for a ±5 V range (gain code of 0) and channel 1 configured for a ±10 V range (gain code of 1). Your channel-gain array would look like the following example:

# of Entries	Chan	Gain Code	Chan	Gain Code
2	0	0	1	1

where the first element is the number of entries in the channel-gain array.

After you create the channel-gain array, you allocate space for the channel-gain array in your program using **K_AllocChnGAry**; you initialize the channel-gain array using **K_FormatChnGAry**; you set the channel-gain array element using **K_SetChnGAry**. When the operation is finished with the channel-gain array, you can free its space using **K_FreeChnGAry**.

Refer to Table 2-3 for the analog output ranges supported by DAS-1800AO Series boards and the gain code associated with each range.

Clock Source

When performing interrupt-mode, DMA-mode, or recycle-mode analog output operations, you can use one of three pacer clocks to determine the period between the updating of a single analog output channel or between each simultaneous updating of both analog output channels: the D/A pacer clock, an external pacer clock, or the A/D pacer clock. These clock sources are described in the following subsections.

D/A Pacer Clock

To specify the internal D/A pacer clock source, use **K_SetClk** to set the clock source to internal.

Since the D/A pacer clock uses a 5 MHz time base, each count represents 0.2 µs. The driver automatically enables the divide-by-10 prescaler. Use **K_SetClkRate** to specify the number of counts (clock ticks) between updates. For example, if you specify a count of 30, the period between updates is 6 µs (166.67 ksamples/s). If two channels are selected, they are updated simultaneously at the rate of the pacer clock.

You can specify a count between 10 and 655,350. The period between updates ranges from 2 µs to 131 ms.

When using the D/A pacer clock, use the following formula to determine the number of counts to specify:

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

For example, if you want an update 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

To specify an external pacer clock, use **K_SetClk** to set the clock source to external.

You connect an external pacer clock to the XPCLK pin (pin 44) on the board's main I/O connector. When you start an analog output operation (using **K_IntStart** or **K_DMAStart**), the driver starts monitoring the state of the external pacer clock. At the next active edge of the external pacer clock (and at every subsequent active edge of the external pacer clock), the analog output channels are updated. Use **K_SetExtClkEdge** 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 write data to the board depends on a number of factors, including your computer, the operating system/environment, the range of the channels, and other issues. If you are using an external pacer clock for analog output operations, make sure that the clock initiates conversions at a rate that the DACs can handle.

Refer to your *DAS-1800AO Series User's Guide* for more information about the external pacer clock.

A/D Pacer Clock

A DAS-1800AO Series board can synchronize digital-to-analog (D/A) conversions with analog-to-digital (A/D) conversions. Use **K_SetClk** to set the clock source to internal, and then use **K_SetSync** to specify that the analog output operation will be synchronized with the analog input operation.

Note that the ADC must be running using the internal A/D pacer clock before a synchronized analog output operation can occur. Simultaneous A/D and D/A conversions occur on each pacer clock pulse.

The update rate of a synchronized analog output operation is determined by the internal A/D pacer clock; use **K_SetClkRate**, specifying an A/D frame, to set the update rate.

Buffering Mode

The buffering mode determines how the driver writes the values in the buffer to the analog output channels. For interrupt-mode, DMA-mode, and recycle-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 process is stopped; when all the values in the buffer have been written, the driver writes the values again. Use **K_SetContRun** to specify continuous buffering mode.

Trigger

You can use a trigger to start an interrupt-mode, DMA-mode, or recycle-mode analog output operation. You can also retrigger an analog output operation. The following subsections describe the supported trigger sources and the retrigger operation.

Trigger Source

The VI Driver supports two trigger sources: internal and external. For interrupt-mode and DMA-mode analog output operations, use **K_SetTrig** to specify the trigger source. External triggers can be either analog triggers or digital triggers.

The trigger event is not significant until the operation the trigger governs has been started (using **K_DMAStart** or **K_IntStart**). The point at which conversions begin depends on the pacer clock; refer to page 2-28 for more information.

The internal trigger, external analog trigger, and external digital trigger are described as follows:

- **Internal trigger -** An internal trigger is a software trigger. The trigger event occurs immediately after you start the operation. Consequently, **K_DMAStart** or **K_IntStart** is considered the trigger event for an internal trigger. The internal trigger is the default trigger source.
- **External analog trigger -** If no analog input operations are running, 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 on page 2-18.

  Note: Analog triggering is a feature of the VI 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.

  Use **K_SetADTrig** to specify the analog input channel to use as the trigger channel, the trigger level, and the trigger polarity (positive or negative).

  You specify the trigger level as a raw count value between 0 and 4095. Refer to Appendix B for information on how to convert a voltage value to a raw count value.

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

The hysteresis value is an absolute number, which you specify as a raw count value between 0 and 4095. When you add the hysteresis value to the trigger level (for a negative trigger) or subtract the hysteresis value from the trigger level (for a positive trigger), the resulting value must also be between 0 and 4095.

For example, assume that you are using a negative trigger on a channel of a board configured for an analog input range of ±5 V. If the trigger 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.

Refer to Figure 2-3 on page 2-19 for an illustration of hysteresis.

- **External digital trigger -** The digital trigger signal is available on the TGIN pin (pin 46) on the board’s main I/O connector. Use **K_SetDITrig** to specify whether you want the trigger event to occur on a rising edge (positive polarity) or a falling edge (negative polarity). These trigger events are shown in Figure 2-4 on page 2-20.

Retriggering

DAS-1800AO Series boards support analog output retriggering for data sets of up to and including 2048 values. During a retriggered analog output operation, after each external digital trigger, the board starts writing the output values from the beginning of the D/A FIFO.

Use the following procedure to define a retriggered analog output operation:

1. Use **K_SetContRun** to set the buffering mode to continuous.
2. Use **K_SetTrig** to set the trigger source to external.
3. Use **K_SetDITrig** to set up the digital trigger, setting the trigger type to retrigger.
4. Use **K_IntStart** or **K_DMAStart** to start the operation.

Note: To retrigger an analog output operation, the values must fit in the D/A FIFO, which can hold up to 2048 samples. If the user-defined buffer contains more than 2048 samples and you specify retrigger mode, the driver returns an error.

Hardware Gate

A hardware gate is an externally applied digital signal that determines whether conversions occur. You connect the gate signal to the TGIN pin (pin 46) on the board's main I/O connector. If you have started an interrupt-mode, DMA-mode, or recycle-mode analog output 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 gate signal to TGIN is high; if the gate signal to TGIN is low, conversions are inhibited. If the board is configured with a negative gate, conversions occur only if the gate signal to TGIN is low; if the gate signal to TGIN is high, conversions are inhibited. Use **K_SetGate** 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 output 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 **K_SetADTrig**).

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 a clock rate and buffering mode for a digital I/O operation.

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

Operation Mode

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 **K_DIRead** to start a digital input operation in single mode; you specify the board you want to use and the digital input channel. Use **K_DOWrite** to start a digital output operation in single mode; you specify the board you want to use, the digital output channel, and the digital output value.

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. The four least significant bits contain the actual digital output value; 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. Use **K_IntStart** to start a digital I/O operation in interrupt mode.

Note: The digital input buffer and the digital output buffer each contain 16-bit integers. Each digital I/O value is stored in the four least significant bits of each integer in the digital I/O buffer.

You can specify either single-cycle or continuous buffering mode for interrupt-mode operations. Refer to page 2-39 for more information on buffering modes. Use **K_IntStop** to stop a continuous-mode interrupt operation. Use **K_IntStatus** 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 memory buffer must be dynamically allocated outside of your application program's memory area.

Use **K_IntAlloc** to allocate memory dynamically for interrupt-mode operations. You specify the operation requiring the buffer and the number of samples to store in the buffer (up to 65,536). The driver returns the starting address of the buffer and a unique identifier for the buffer; this identifier is called the buffer handle.

After you allocate your buffer, you must assign the starting address of the buffer using **K_SetBuf**. To move the contents of an allocated buffer to a LabVIEW buffer, use **K_MoveBufToArray**. To move the contents of a LabVIEW buffer to an allocated buffer, use **K_MoveArrayToBuf**.

When the buffer is no longer required, you can free it for another use by specifying the buffer handle in **K_IntFree**.

Digital Input Channel

DAS-1800AO 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); bit 1 contains the value of digital input line 1 (DI1); 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
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 and DI2 is high and the input at DI1 and DI3 is low.

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

Digital Output Channel

DAS-1800AO 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.

bit 3	bit 2	bit 1	bit 0
DO3	DO2	DO1	DO0

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 (1100), the output at DO0 and DO1 is forced low and the output at DO2 and DO3 is forced high.

Clock Source

When performing interrupt-mode digital I/O operations, you can use the internal A/D 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 A/D pacer clock only if it is not being used by another operation.

The internal A/D 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 **K_SetClkRate** 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 a FIFO is not used to buffer values for digital I/O operations, a low count value may cause overrun errors. The maximum typical read/write rate for the internal A/D pacer clock is 1 ksamples/s. This rate would indicate a minimum count of 5,000.

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

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

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 Mode

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 process is stopped; 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 process is stopped; when all the values in the buffer have been written, the driver writes the values again. Use **K_SetContRun** to specify continuous buffering mode.

3

Programming with the VI Driver

This chapter contains an overview of the structure of the DAS-1800 Series VI Driver, as well as programming guidelines to assist you when writing LabVIEW application programs with DAS-1800 Series VIs.

How the Driver Works

When writing LabVIEW application programs, you can use VIs from one or more Keithley MetraByte DAS VI 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; when you initialize a board, the driver returns a handle that uniquely identifies the board. Each board handle is associated with a particular driver.

The VI Driver supports a variety of operation modes. For single mode, the I/O operation is performed using a single VI; the attributes of the I/O operation are specified as input parameters to the VI. Figure 3-1 illustrates a single-mode analog input operation using the VI, **K_ADRead**.

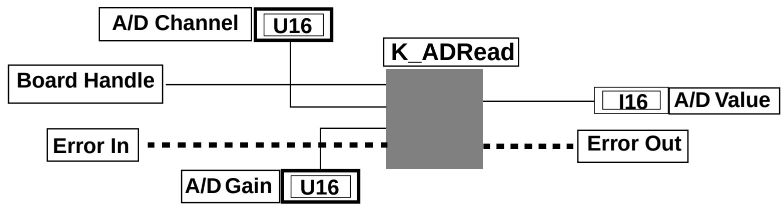


Figure 3-1. Single-Mode Operation

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 with 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 VI 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; 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. The driver returns a unique identifier for the frame; this identifier is called the frame handle. You then specify the attributes of the I/O operation by using the applicable VIs 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 VI, **K_SetChn**.

You use the frame handle you specified when you accessed the frame in all VIs 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 by referencing the appropriate frame handle. Figure 3-2 shows the frame elements referenced by the *Frame Handle* parameter specified by the VI, **K_IntStart**.

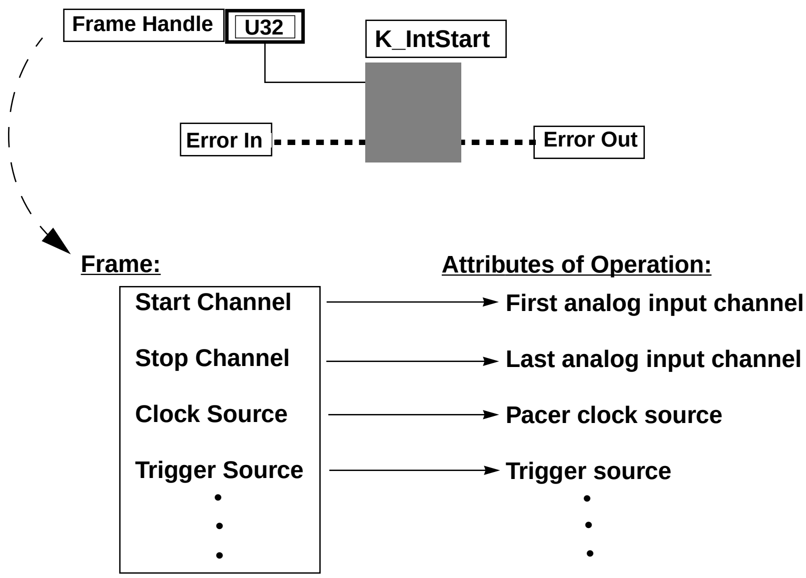


Figure 3-2. Using a Frame for an Interrupt-Mode Operation

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-1800AO Series boards, interrupt-mode, DMA-mode, and recycle-mode operations require frames. The DAS-1800 Series VI Driver provides the following types of frames:

- Analog input frames, called A/D (analog-to-digital) frames, that can be used with interrupt-mode and DMA-mode operations. You use **K_GetADFrame** to access an available A/D frame and a frame handle.
- Analog output frames, called D/A (digital-to-analog) frames, that can be used with interrupt-mode, DMA-mode, and recycle-mode operations. You use **K_GetDAFrame** to access an available D/A frame and a frame handle.
- Digital input frames, called DI frames, that can be used with interrupt-mode operations. You use **K_GetDIFrame** to access an available DI frame and a frame handle.

- Digital output frames, called DO frames, that can be used with interrupt-mode operations. You use **K_GetDOFrame** to access an available DO frame and a frame handle.

If you want to perform an interrupt-mode, DMA-mod, or recycle-mode operation and all frames of a particular type have been accessed, you can use **K_FreeFrame** 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 **K_ClearFrame** to reset all the elements of a frame to their default values.

The tables on the following pages list the elements of frames for DAS-1800AO Series boards: Table 3-1 lists the elements of an A/D frame; Table 3-2 lists the elements of a D/A frame; Table 3-3 lists the elements of a DI frame; Table 3-4 lists the elements of a DO frame. These tables also list the default value of each element and the VIs used to define each element.

Table 3-1. A/D Frame Elements

Element	Default Value	VIs
Buffer[1]	0 (NULL)	K_SetBuf K_SetDMABuf K_BufListAdd
Number of Samples	0	K_SetBuf K_BufListAdd
Buffering Mode	Single-cycle	K_SetContRun K_ClrContRun[2]
Gain	0 (gain of 1)	K_SetG K_SetStartStopG
Channel-Gain Array	0 (NULL)	K_SetChnGAry
SSH Mode	Disabled	K_SetSSH
Clock Source	Internal	K_SetClk
Pacer Clock Rate[1]	0	K_SetClkRate

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

Element	Default Value	VIs
External Clock Edge	Negative	K_SetExtClkEdge
Burst Clock Rate	3 (333 ksamples/s)	K_SetBurstTicks
Trigger Source	Internal	K_SetTrig
Trigger Type	Digital	K_SetADTrig K_SetDITrig
Trigger Channel	0 (for analog trigger)	K_SetADTrig
	0 (channel 0, bit 0) (for digital trigger)	Not applicable[3]
Trigger Polarity	Positive (for analog trigger)	K_SetADTrig
	Positive (for digital trigger)	K_SetDITrig
Trigger Sensitivity	Edge (for analog and digital trigger)	Not applicable[3]
Trigger Level	0	K_SetADTrig
Trigger Hysteresis	0	K_SetTrigHyst
Trigger Pattern	Not used[4]	Not applicable[3]
Hardware Gate	Disabled	K_SetGate

Notes

[1] This element must be set.
[2] Use this VI 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.

Table 3-2. D/A Frame Elements

Element	Default Value	VIs
Buffer[1]	0 (NULL)	K_SetBuf K_SetDMABuf K_BufListAdd
Number of Samples	0	K_SetBuf K_SetDMABuf K_BufListAdd
Buffering Mode	Single-cycle	K_SetContRun K_ClrContRun[2]
Start Channel	0	K_SetChn K_SetStartStopChn K_SetStartStopG
Stop Channel	0	K_SetStartStopChn K_SetStartStopG
Gain	0 (gain of 1)	K_SetG K_SetStartStopG
Channel-Gain Array	0 (NULL)	K_SetChnGAry
Conversion Mode	Paced	K_SetADFreeRun K_ClrADFreeRun[2]
Clock Source	Internal D/A	K_SetClk K_SetSync
Pacer Clock Rate[1]	0	K_SetClkRate
External Clock Edge	Negative	K_SetExtClkEdge
Trigger Source	Internal	K_SetTrig
Trigger Type	Digital	K_SetADTrig K_SetDITrig
Trigger Channel	0 (for analog trigger)	K_SetADTrig
	0 (channel 0, bit 0) (for digital trigger)	Not applicable[3]

Table 3-2. D/A Frame Elements (cont.)

Element	Default Value	VIs
Trigger Polarity	Positive (for analog trigger)	K_SetADTrig
	Positive (for digital trigger)	K_SetDITrig
Trigger Sensitivity	Edge (for analog and digital trigger)	Not applicable[3]
Trigger Level	0	K_SetADTrig
Trigger Hysteresis	0	K_SetTrigHyst
Trigger Pattern	Not used[4]	Not applicable[3]
Hardware Gate	Disabled	K_SetGate

Notes

[1] This element must be set.
[2] Use this VI 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.

Table 3-3. DI Frame Elements

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

Notes

[1] This element must be set.
[2] Use this VI 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	VIs
Buffer[1]	0 (NULL)	K_SetBuf
Buffering Mode	Single-cycle	K_SetContRun K_ClrContRun[2]
Number of Samples	0	K_SetBuf
Start Channel	0	Not applicable[3]
Stop Channel	0	Not applicable[3]
Clock Source	Internal	Not applicable[3]
Pacer Clock Rate[1]	0	K_SetClkRate

Notes

[1] This element must be set.
[2] Use this VI 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.

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

For information about using VIs in your application program, refer to the following sections of this chapter. For detailed information about each VI, refer to Chapter 4.

General Programming Tasks

For every LabVIEW program that uses DAS-1800 Series VIs, you must perform the following tasks:

1. Create an error cluster by selecting a cluster control, defining the elements, and initializing the values of the elements, as shown in Table 3-5.

Table 3-5. Error Cluster Elements

Element	Data Type	Default Value	Description
VI Status	T F	False	Boolean: Used to store the status of the error
Error Code	I16	0	Numeric: Used to store the error code
Error Source	a b c	Null	String: Used to store the name of the VI that caused the error

2. Define and initialize the parameters for each DAS-1800 Series VI in your program and wire the appropriate parameters to the VIs. (See the next section for defining the VIs specific to analog and digital operations.) Note that the error cluster defined in step 1 should be wired to the first DAS-1800 Series VI in your program, normally **K_OpenDriver**.
3. Select **K_OpenDriver** to initialize the driver.
4. Initialize the DAS board by selecting **K_GetDevHandle**. If you are using more than one DAS board, select the VI once for each board you are using.

Note: At the end of your program, it is recommended that you read the error information (using an Unbundle by Name function, as described on page 2-3) and close the driver using **K_CloseDriver**.

Operation-Specific Programming Tasks

The programming tasks specific to analog and digital I/O operations are described in the following sections. Refer to Chapter 2 for detailed information about these VIs.

Note that any VIs that are not mentioned in the operation-specific programming tasks can be used at any point in your application program. Refer to Chapter 4 for detailed descriptions of each VI.

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, use **K_ADRead** to read the single analog input value; specify the attributes of the operation as inputs to the VI.

Interrupt Mode

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

1. Use **K_GetADFrame** to access an A/D frame.
2. Use **K_IntAlloc** to allocate the buffers in which to store the acquired data.
3. *If you want to use a channel-gain array to specify the channels*, use **K_AllocChnGAry**, **K_FormatChnGAry**, and **K_SetChnGAry** to define and set the array. Refer to page 2-12 for more information about channel-gain arrays.
4. Use the appropriate VIs to specify the attributes of the operation. These VIs 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 VI associated with that element. Refer to Table 3-1 on page 3-4 for a list of the default values of A/D frame elements.

Table 3-6. VIs Used for Interrupt-Mode Analog Input Operations

Attribute	VIs
Buffer[1]	K_SetBuf K_BufListAdd
Number of Samples	K_SetBuf K_BufListAdd
Buffering Mode	K_SetContRun K_ClrContRun[2]
Start Channel	K_SetChn K_SetStartStopChn K_SetStartStopG
Stop Channel	K_SetStartStopChn K_SetStartStopG
Gain	K_SetG K_SetStartStopG
Channel-Gain Array	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

Table 3-6. VIs Used for Interrupt-Mode Analog Input Operations (cont.)

Attribute	VIs
Trigger Type	K_SetADTrig K_SetDITrig
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 VI to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame.

5. Use **K_IntStart** to start the interrupt-mode operation.
6. Use **K_IntStatus** to monitor the status of the interrupt-mode operation.
7. *If you specified continuous buffering mode*, use **K_IntStop** to stop the interrupt-mode operation when the appropriate number of samples has been acquired.
8. Use **K_MoveBufToArray** to transfer the acquired data from the allocated buffer to a LabVIEW array.
9. Use **K_IntFree** to deallocate the buffers.
10. *If you used **K_BufListAdd** to specify a list of multiple buffers*, use **K_BufListReset** to clear the list.
11. Use **K_FreeFrame** 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 **K_GetADFrame** to access an A/D frame.
2. Use **K_DMAAlloc** to allocate the buffers in which to store the acquired data.
3. *If you want to use a channel-gain array to specify the channels*, use **K_AllocChnGAry**, **K_FormatChnGAry**, and **K_SetChnGAry** to define and set the array. Refer to page 2-12 for more information about channel-gain arrays.
4. Use the appropriate VIs to specify the attributes of the operation; these VIs are listed in Table 3-7.

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 VI associated with that element. Refer to Table 3-1 on page 3-4 for a list of the default values of A/D frame elements.

Table 3-7. VIs Used for DMA-Mode Analog Input Operations

Attribute	**VIs**
Buffer[1]	K_SetDMABuf K_BufListAdd
Number of Samples	K_SetBuf K_BufListAdd
Buffering Mode	K_SetContRun K_ClrContRun[2]
Start Channel	K_SetChn K_SetStartStopChn K_SetStartStopG
Stop Channel	K_SetStartStopChn K_SetStartStopG

Table 3-7. VIs Used for DMA-Mode Analog Input Operations (cont.)

Attribute	VIs
Gain	K_SetG K_SetStartStopG
Channel-Gain Array	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
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 VI to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame.

5. Use **K_DMAStart** to start the DMA-mode operation.
6. Use **K_DMAStatus** to monitor the status of the DMA-mode operation.

7. *If you specified continuous buffering mode*, use **K_DMAStop** to stop the DMA-mode operation when the appropriate number of samples has been acquired.
8. Use **K_MoveBufToArray** to transfer the acquired data from the allocated buffer to a LabVIEW array.
9. Use **K_DMAFree** to deallocate the buffers.
10. *If you used **K_BufListAdd** to specify a list of multiple buffers*, use **K_BufListReset** to clear the list.
11. Use **K_FreeFrame** to return the frame you accessed in step 1 to the pool of available frames.

Analog Output Operations

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

Single Mode

For a single-mode analog output operation, use **K_DAWriteGain** to write the single analog output value; specify the attributes of the operation as inputs to the VI.

Interrupt Mode

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

1. Use **K_GetDAFrame** to access a D/A frame.
2. Use **K_IntAlloc** to allocate the buffer in which to store the data to be written.

3. *If you want to use a channel-gain array to specify the channels*, use **K_AllocChnGAry**, **K_FormatChnGAry**, and **K_SetChnGAry** to define and set the array. Refer to page 2-27 for more information about channel-gain arrays.

4. Use the appropriate VIs to specify the attributes of the operation; these VIs are listed in Table 3-10.

Note: When you access a new D/A 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 VI associated with that element. Refer to Table 3-2 on page 3-6 for a list of the default values of D/A frame elements.

Table 3-8. VIs Used for Interrupt-Mode Analog Output Operations

Attribute	VIs
Buffer[1]	K_SetBuf
Number of Samples	K_SetBuf
Buffering Mode	K_SetContRun K_ClrContRun[2]
Gain	K_SetG K_SetStartStopG
Channel-Gain Array	K_SetChnGAry
Clock Source/Sync	K_SetClk K_SetSync
Pacer Clock Rate[1]	K_SetClkRate
External Clock Edge	K_SetExtClkEdge
Trigger Source	K_SetTrig
Trigger Type	K_SetADTrig K_SetDITrig
Trigger Channel	K_SetADTrig
Trigger Polarity	K_SetADTrig

Table 3-8. VIs Used for Interrupt-Mode Analog Output Operations (cont.)

Attribute	VIs
Trigger Level	K_SetADTrig
Trigger Hysteresis	K_SetTrigHyst
Hardware Gate	K_SetGate

Notes

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

5. Use **K_MoveArrayToBuf** to transfer the data from a LabVIEW array to the allocated buffer.
6. Use **K_IntStart** to start the interrupt-mode operation.
7. Use **K_IntStatus** to monitor the status of the interrupt-mode operation.
8. *If you specified continuous buffering mode*, use **K_IntStop** to stop the interrupt-mode operation when the appropriate number of samples has been written.
9. Use **K_IntFree** to deallocate the buffer.
10. Use **K_FreeFrame** to return the frame you accessed in step 1 to the pool of available frames.

DMA Mode

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

1. Use **K_GetDAFrame** to access a D/A frame.
2. Use **K_DMAAlloc** to allocate the buffer dynamically outside your program's memory area.
3. *If you want to use a channel-gain array to specify the channels*, use **K_AllocChnGAry**, **K_FormatChnGAry**, and **K_SetChnGAry** to define and set the array. Refer to page 2-27 for more information about channel-gain arrays.

4. Use the appropriate VIs to specify the attributes of the operation. These VIs are listed in Table 3-10.

Note: When you access a new D/A 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 VI associated with that element. Refer to Table 3-2 on page 3-6 for a list of the default values of D/A frame elements.

Table 3-9. VIs Used for DMA-Mode Analog Output Operations

Attribute	VIs
Buffer[1]	K_SetDMABuf K_SetBufListAdd
Number of Samples	K_SetBuf K_SetBufListAdd
Buffering Mode	K_SetContRun K_ClrContRun[2]
Start Channel	K_SetChn K_SetStartStopChn
Stop Channel	K_SetStartStopChn
Gain	K_SetG K_SetStartStopG
Channel-Gain Array	K_SetChnGAry
Pacer Clock Rate[1]	K_SetClkRate
External Clock Edge	K_SetExtClkEdge
Trigger Source	K_SetTrig
Trigger Type	K_SetADTrig K_SetDITrig
Trigger Channel	K_SetADTrig
Trigger Polarity	K_SetADTrig

Table 3-9. VIs Used for DMA-Mode Analog Output Operations (cont.)

Attribute	VIs
Trigger Level	K_SetADTrig
Trigger Hysteresis	K_SetTrigHyst
Hardware Gate	K_SetGate

Notes

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

5. Use **K_MoveArrayToBuf** to transfer the data from a LabVIEW array to the allocated buffer.
6. Use **K_DMAStart** to start the DMA-mode operation.
7. Use **K_DMAStatus** to monitor the status of the DMA-mode operation.
8. *If you specified continuous buffering mode*, use **K_DMAStop** to stop the DMA-mode operation when the appropriate number of samples has been written.
9. Use **K_DMAFree** to deallocate the buffer.
10. Use **K_FreeFrame** to return the frame you accessed in step 1 to the pool of available frames.

Recycle Mode

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

1. Use **K_GetDAFrame** to access a D/A frame.
2. Use **K_IntAlloc** or **K_DMAAlloc** to allocate the buffer dynamically outside your program's memory area. The buffer must contain 2048 samples or fewer.
3. Use **K_SetContRun** to specify continuous buffering mode.
4. Use the appropriate VIs to specify the attributes of the operation. These VIs are listed in Table 3-10.

Note: When you access a new D/A 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 VI associated with that element. Refer to Table 3-2 on page 3-6 for a list of the default values of D/A frame elements.

Table 3-10. VIs Used for Recycle-Mode Analog Output Operations

Attribute	VIs
Buffer[1]	K_SetDMABuf K_SetBufListAdd
Number of Samples	K_SetBuf K_SetBufListAdd
Gain	K_SetG K_SetStartStopG
Channel-Gain Array	K_SetChnGAry
Pacer Clock Rate[1]	K_SetClkRate
External Clock Edge	K_SetExtClkEdge
Trigger Source	K_SetTrig
Trigger Type	K_SetADTrig K_SetDITrig
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.

5. Use **K_MoveArrayToBuf** to transfer the data from a LabVIEW array to the allocated buffer.
6. Use **K_IntStart** or **K_DMAStart** to start the recycle-mode operation.
7. Use **K_IntStatus** or **K_DMAStatus** to monitor the status of the recycle-mode operation.
8. Use **K_IntStop** or **K_DMAStop** to stop the recycle-mode operation when the appropriate number of samples has been written.
9. If you used **K_IntAlloc** to allocate the buffer, use **K_IntFree** to deallocate the buffer; if you used **K_DMAAlloc** to allocate the buffer, use **K_DMAFree** to deallocate the buffer.
10. Use **K_FreeFrame** 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, use **K_DIRead** to read a single digital input value or use **K_DOWrite** to write a single digital output value. Specify the attributes of the operation as inputs to the VI.

Interrupt Mode

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

1. Use K_GetDIFrame to access a DI frame; use K_GetDOFrame to access a DO frame.
2. Use **K_IntAlloc** to allocate the buffer in which to store the data to be read or written.
3. Use the appropriate VIs to specify the attributes of the operation; these VIs are listed in Table 3-11.

Note: When you access a new DI or DO frame, the frame elements contain default values. If the default value of a particular frame element is suitable for your operation, you do not have to use the VI 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 3-11. VIs Used for Interrupt-Mode Digital Input and Digital Output Operations

Attribute	VIs
Buffer[1]	K_SetBuf
Number of Samples	K_SetBuf
Buffering Mode	K_SetContRun K_ClrContRun[2]
Pacer Clock Rate[1]	K_SetClkRate

Notes

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

4. *If you are performing a digital output operation*, use **K_MoveArrayToBuf** to transfer the data from a LabVIEW array to the allocated buffer.

5. Use **K_IntStart** to start the interrupt-mode operation.
6. Use **K_IntStatus** to monitor the status of the interrupt-mode operation.
7. *If you specified continuous buffering mode*, use **K_IntStop** to stop the interrupt-mode operation when the appropriate number of samples has been written.
8. *If you are performing a digital input operation*, use **K_MoveBufToArray** to transfer the data from the allocated buffer to a LabVIEW array.
9. Use **K_IntFree** to deallocate the buffer.
10. Use **K_FreeFrame** to return the frame you accessed in step 1 to the pool of available frames.

4

VI Reference

The DAS-1800 Series VIs are organized into the following functional groups:

- Initialization
- Operation mode
- Frame management
- Memory management
- Buffer address
- Buffering mode
- Conversion mode
- Channel and gain
- Clock
- Trigger
- Gate
- Miscellaneous

The particular VIs associated with each group are listed in Table 4-1. The remainder of the chapter presents detailed descriptions of each VI, arranged in alphabetical order.

Table 4-1. VIs by Functional Group

VI Functional Groups	VI Name	Page Number
Initialization	K_OpenDriver	page 4-54
	K_CloseDriver	page 4-11
	K_GetDevHandle	page 4-38
	K_FreeDevHandle	page 4-29
	K_DASDevInit	page 4-15
Operation Mode	K_ADRead	page 4-5
	K_DAWriteGain	page 4-16
	K_DIRead	page 4-18
	K_DOWrite	page 4-26
	K_DMAStart	page 4-21
	K_DMAStatus	page 4-22
	K_DMAStop	page 4-25
	K_IntStart	page 4-47
	K_IntStatus	page 4-48
	K_IntStop	page 4-51
Frame Management	K_GetADFrame	page 4-33
	K_GetDAFrame	page 4-37
	K_GetDIFrame	page 4-39
	K_GetDOFrame	page 4-40
	K_FreeFrame	page 4-30
	K_ClearFrame	page 4-10

Table 4-1. VIs by Functional Group (cont.)

VI Functional Groups	VI Name	Page Number
Memory Management	K_DMAAlloc	page 4-19
	K_DMAFree	page 4-20
	K_IntAlloc	page 4-45
	K_IntFree	page 4-46
	K_MoveArrayToBuf	page 4-52
	K_MoveBufToArray	page 4-53
Buffer Address	K_SetBuf	page 4-62
	K_SetDMABuf	page 4-73
	K_BufListAdd	page 4-8
	K_BufListReset	page 4-9
Buffering Mode	K_ClrContRun	page 4-14
	K_SetContRun	page 4-70
Conversion Mode	K_SetADFreeRun	page 4-58
	K_ClrADFreeRun	page 4-13
	K_SetSSH	page 4-77
Channel and Gain	K_SetChn	page 4-64
	K_SetStartStopChn	page 4-78
	K_SetG	page 4-75
	K_SetStartStopG	page 4-80
	K_AllocChnGAry	page 4-7
	K_FormatChnGAry	page 4-27
	K_FreeChnGAry	page 4-28
	K_SetChnGAry	page 4-65
	K_SetADCommonMode	page 4-56
	K_SetADConfig	page 4-57
	K_SetADMode	page 4-59

Table 4-1. VIs by Functional Group (cont.)

VI Functional Groups	VI Name	Page Number
Channel and Gain (cont.)	K_GetADCommonMode	page 4-31
	K_GetADConfig	page 4-32
	K_GetADMode	page 4-34
Clock	K_SetClk	page 4-66
	K_SetClkRate	page 4-68
	K_SetExtClkEdge	page 4-74
	K_GetClkRate	page 4-35
	K_SetBurstTicks	page 4-63
	K_SetSync	page 4-82
Trigger	K_SetTrig	page 4-83
	K_SetADTrig	page 4-60
	K_SetTrigHyst	page 4-84
	K_SetDITrig	page 4-71
	K_SetAboutTrig	page 4-55
	K_ClrAboutTrig	page 4-12
Gate	K_SetGate	page 4-76
Miscellaneous	K_GetErrMsg	page 4-41
	K_GetVer	page 4-43
	K_GetShellVer	page 4-42

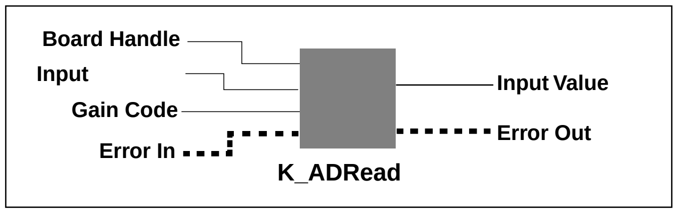
For a description of the error information in the Error In and Error Out parameters in this chapter, see page 2-3.

Purpose

Reads a single analog input value.

Description

This VI reads the analog input channel represented by *Input Channel* on the board specified by *Board Handle* at the gain represented by *Gain Code*, and stores the raw count in *Input Value*.



Parameters

U32 *Board Handle* Handle associated with the board.

U16 *Input Channel* Analog input channel.
Valid values are shown below:

Board Configuration	Valid Channel Numbers	
	Differential	**Single-ended**
DAS-1800AO Series board	**0** to **7**	**0** to **15**
DAS-1800AO Series board with ***N*** EXP-1800s attached	Not applicable	**0** to **15(*N* + 1)**

U16 *Gain Code* Valid values:
0 to **3** = DAS board channels
0 to **7** = EXP-1800 channels

I16 *Input Value* Acquired analog input value.

Error In Error information.

K_ADRead (cont.)

 *Error Out* Error information.

Remarks

Refer to Table 2-2 on page 2-7 for the gain and input ranges associated with each gain code.

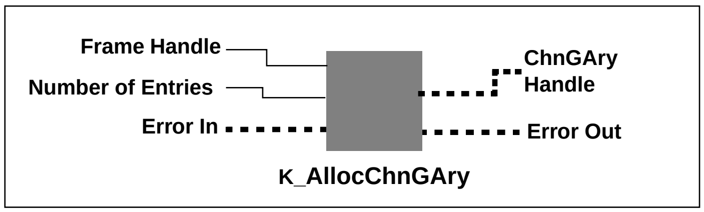
Refer to Appendix B for converting the raw count stored in *Input Value* to voltage.

See Also

K_DMAStart, K_IntStart

Purpose Allocates space for a channel-gain array.

Description For the operation defined by *Frame Handle*, this VI uses the number of entries in *Number of Entries* to allocate space for a channel-gain array and creates a handle for the array in *ChnGAry Handle*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Number of Entries*	Number of channel-gain pairs in the channel-gain array.
	ChnGAry Handle	Handle associated with the allocated channel-gain array.
	Error In	Error information.
	Error Out	Error information.

Remarks Refer to page 2-12 for information on setting up a channel-gain array for analog input operations: refer to page 2-27 for information on setting up a channel-gain array for analog output operations.

See Also K_FormatChnGAry, K_FreeChnGAry, K_SetChnGAry

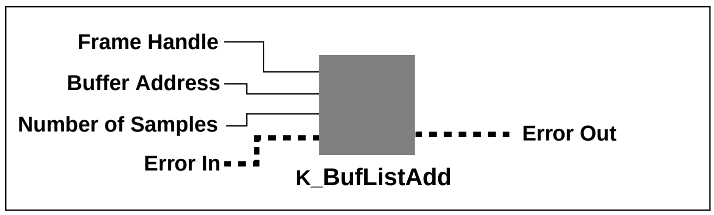
K_BufListAdd

Purpose

Adds a buffer to the list of multiple buffers.

Description

For the operation defined by *Frame Handle*, this VI adds the buffer at the address pointed to by *Buffer Address* to the list of multiple buffers; the number of samples in the buffer is specified in *Number of Samples*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U32	*Buffer Address*	Starting address of buffer.
U32	*Number of Samples*	Number of samples in the buffer.
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks

The driver supports multiple buffers for analog input and analog output operations. Before you add the buffer to the multiple-buffer list, you must allocate the buffer dynamically using **K_IntAlloc** or **K_DMAAlloc**.

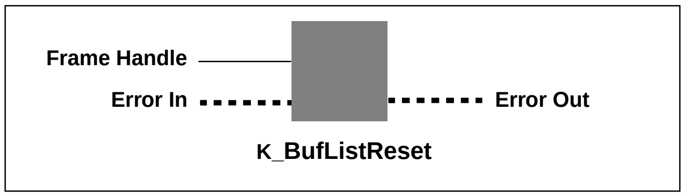
Make sure that you add buffers to the multiple-buffer list in the order in which you want to use them. The first buffer you add is Buffer 1, the second buffer you add is Buffer 2, and so on. You can add up to 149 buffers. You can use **K_IntStatus** or **K_DMAStatus** to determine which buffer is currently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

See Also

K_BufListReset, K_DMAAlloc, K_IntAlloc

Purpose Clears the list of multiple buffers.

Description For the operation defined by *Frame Handle*, this VI clears all buffers from the list of multiple buffers.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks This VI does not deallocate the buffers in the list of multiple buffers. 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.

See Also K_DMAFree, K_IntFree, K_SetBuf, K_SetDMABuf

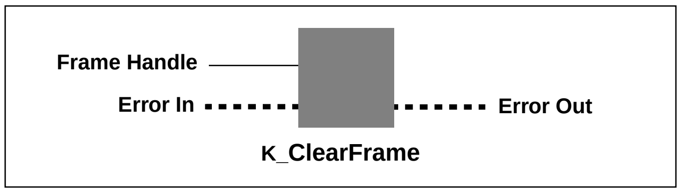
K_ClearFrame

Purpose

Sets the elements of a frame to their default values.

Description

This VI sets the elements of the frame specified by *Frame Handle* to their default values.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
	Error In	Error information.
	Error Out	Error information.

Remarks

For the default values of the elements of frames, refer to the following tables:

Frame Type	See
A/D frames	Table 3-1 on page 3-4
D/A frames	Table 3-2 on page 3-6
DI frames	Table 3-3 on page 3-8
DO frames	Table 3-4 on page 3-9

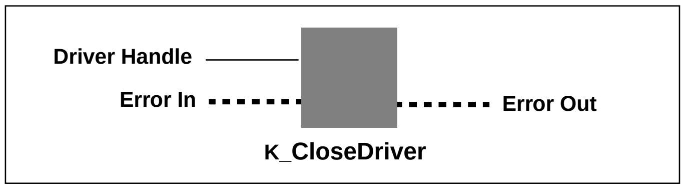
See Also

K_GetADFrame, K_GetDAFrame, K_GetDIFrame, K_GetDOFrame

K_CloseDriver

Purpose Closes a previously initialized Keithley DAS VI Driver.

Description This VI frees the driver handle specified by *Driver Handle* and closes the associated use of the VI Driver. This VI also frees all board handles and frame handles associated with *Driver Handle*.



Parameters

U32	*Driver Handle*	Driver handle you want to free.
	Error In	Error information.
	Error Out	Error information.

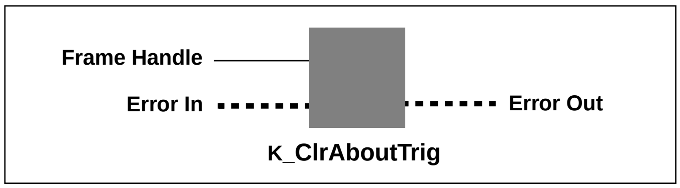
Remarks If *Driver Handle* is the last driver handle specified for the VI Driver, the driver is shut down and unloaded.

See Also K_FreeDevHandle

K_ClrAboutTrig

Purpose Disables the about trigger for an analog input operation.

Description This VI disables the about trigger for the operation defined by *Frame Handle*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
	Error In	Error information.
	Error Out	Error information.

Remarks **K_GetADFrame** and **K_ClearFrame** also disable the about trigger.

See Also K_ClearFrame, K_GetADFrame, K_SetAboutTrig

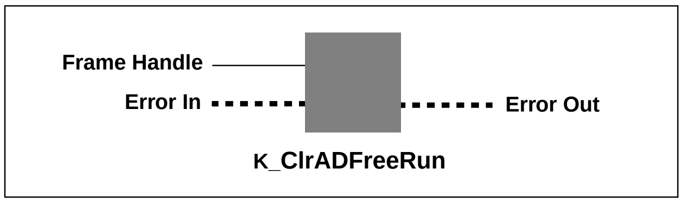
K_ClrADFreeRun

Purpose

Sets paced conversion mode for an analog input operation.

Description

This VI sets the conversion mode for the operation defined by *Frame Handle* to paced mode and sets the Conversion Mode element in the frame accordingly.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
	Error In	Error information.
	Error Out	Error information.

Remarks

K_GetADFrame and **K_ClearFrame** also enable paced conversion mode.

See Also

K_ClearFrame, K_GetADFrame, K_SetADFreeRun

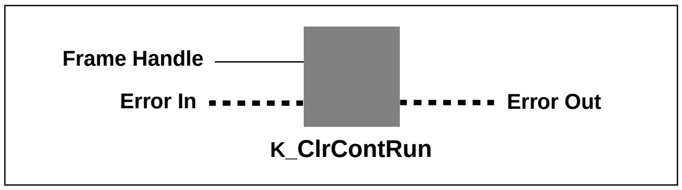
K_ClrContRun

Purpose

Sets single-cycle buffering mode.

Description

This VI sets the buffering mode for the operation defined by *Frame Handle* to single-cycle mode and sets the Buffering Mode element in the frame accordingly.



Parameters

Frame Handle Handle to the frame that defines the operation.

Error In Error information.

Error Out Error information.

Remarks

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame,** and **K_ClearFrame** also enable single-cycle buffering mode. For more information on buffering modes, refer to the following pages:

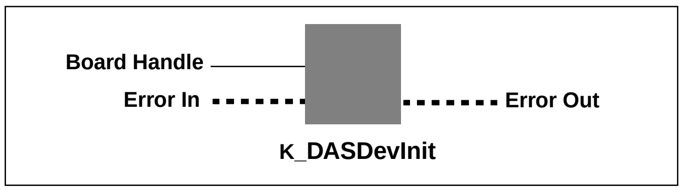
Operation	See
Analog input	page 2-16
Analog output	page 2-30
Digital I/O	page 2-39

See Also

K_SetContRun

Purpose Reinitializes a board.

Description This VI stops all current operations and resets the board specified by *Board Handle* and the driver to their power-up states.



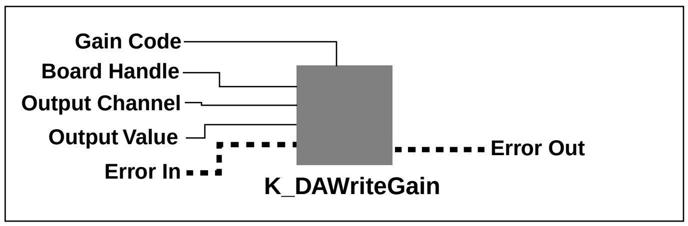
Parameters

	*Board Handle*	Handle associated with the board.
	*Error In*	Error information.
	*Error Out*	Error information.

K_DAWriteGain

Purpose Writes a single analog output value.

Description For the operation defined by *Board Handle*, this VI writes the single analog output value *Output Value* to the channel represented by *Output Channel*. The output range is specified by *Gain Code*.



Parameters

U32	*Board Handle*	Handle to the board that defines the operation.
U16	*Output Channel*	Analog output channel. Valid values: **0** for DAC 0 **1** for DAC 1
U32	*Output Value*	Analog output value. Valid values: –**2,048** to **2,047**
U16	*Gain Code*	Valid values: **0** for ±5 V D/A range **1** for ±10 V D/A range
	Error In	Error information.
	Error Out	Error information.

Remarks The value of *Output Value* comprises only the least significant 12 bits.

Refer to page 2-26 for more information on output ranges and their corresponding gain codes.

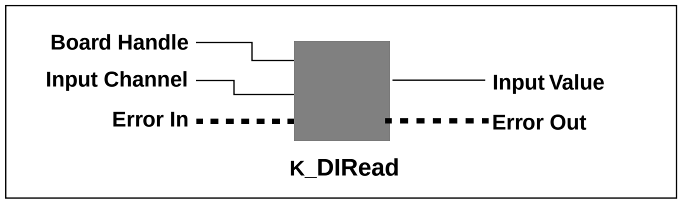
Refer to Appendix B for converting a voltage value to a raw count.

See Also K_IntStart

K_DIRead

Purpose Reads a single digital input value.

Description This VI reads the values of all digital input lines on the board specified by *Board Handle*, and stores the value in *Input Value*.



Parameters

U32	*Board Handle*	Handle associated with the board.
U16	*Input Channel*	Digital input channel. Valid value: **0**
U16	*Input Value*	Digital input value.
	*Error In*	Error information.
	*Error Out*	Error information.

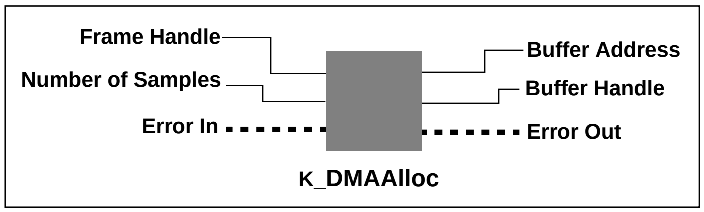
Remarks The acquired digital value in *Input Value* is stored in bits 0, 1, 2, and 3; the values in the remaining bits of *Input Value* are not defined. Refer to Figure on page 2-36 for more information.

See Also K_IntStart

K_DMAAlloc

Purpose Allocates a buffer for a DMA-mode operation.

Description For the operation defined by *Frame Handle*, this VI allocates a buffer of the size *Number of Samples*. On return, *Buffer Address* contains the address of a buffer that is suitable for a DMA-mode operation and *Buffer Handle* is the handle associated with the buffer.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U32	*Number of Samples*	Number of samples. Valid values: **1** to **65,536**
U32	*Buffer Address*	Starting address of the allocated buffer.
U16	*Buffer Handle*	Handle associated with the allocated buffer.
	Error In	Error information.
	Error Out	Error information.

Remarks Use **K_SetDMABuf** or **K_BufListAdd** to assign *Buffer Address* to the frame that defines the operation. *Buffer Handle*, as returned by this VI, is later used to free the allocated memory block when used with **K_DMAFree**.

See Also K_DMAFree, K_SetDMABuf, K_BufListAdd

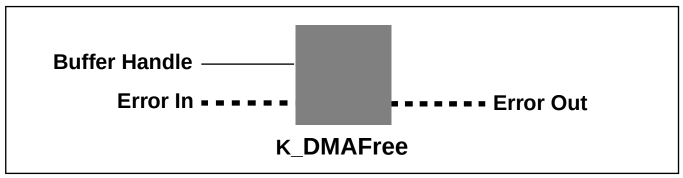
K_DMAFree

Purpose

Frees a buffer allocated for a DMA-mode operation.

Description

This VI frees the buffer specified by *Buffer Handle*; the buffer was previously allocated dynamically using **K_DMAAlloc**.



Parameters

	*Buffer Handle*	Handle to DMA buffer.
	*Error In*	Error information.
	*Error Out*	Error information.

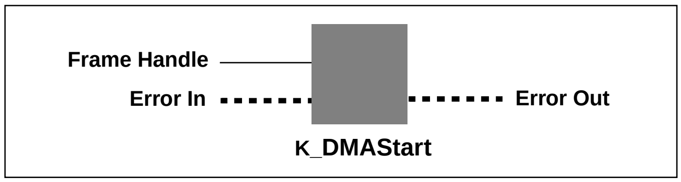
See Also

K_DMAlloc, K_SetDMABuf, K_BufListAdd

K_DMAStart

Purpose Starts a DMA-mode operation or a recycle-mode operation.

Description This VI starts the DMA-mode operation or recycle-mode operation defined by *Frame Handle*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
	Error In	Error information.
	Error Out	Error information.

Remarks For analog output operations, if the user-defined buffer contains less than 2047 samples, the DAS-1800AO Series board does not use the DMA resources of the board; this allows the board to provide the maximum transfer rate (up to 500 kHz). However, your program must still use **K_DMAStart** to start the operation regardless of whether the DMA resources are used.

Refer to Chapter 2 for more information on DMA-mode operations and on recycling data from the D/A FIFO.

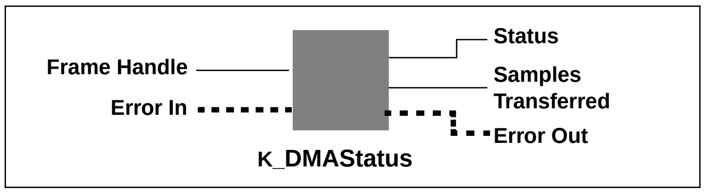
Refer to Chapter 3 for a discussion of the programming tasks associated with DMA-mode and recycle-mode operations.

See Also K_DMAStatus, K_DMAStop

K_DMAStatus

Purpose Gets the status of a DMA-mode operation or a recycle-mode operation.

Description For the DMA-mode operation or recycle-mode operation defined by *Frame Handle*, this VI stores the status in *Status*.

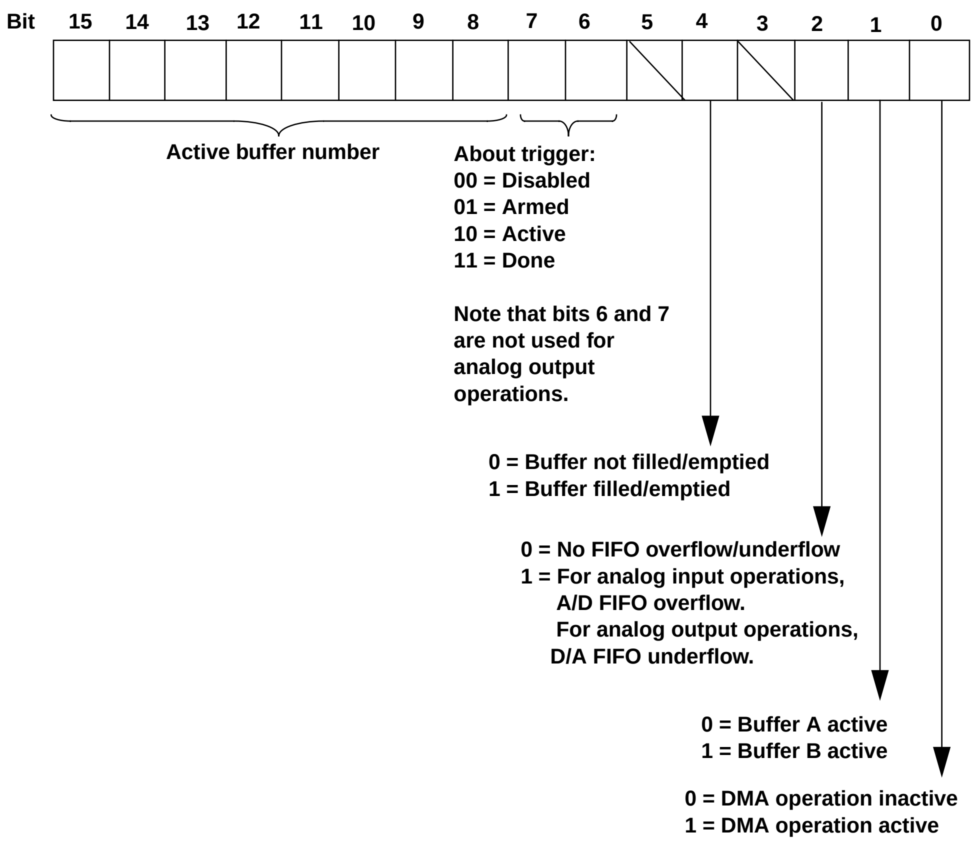


Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Status*	Status of DMA-mode operation or recycle-mode operation. Valid values: See **Remarks** below for value stored.
U32	*Samples Transferred*	Number of samples.
	Error In	Error information.
	Error Out	Error information.

Remarks For analog input operations, *Samples Transferred* stores the number of samples acquired into the current buffer. For analog output operations, *Samples Transferred* stores the number of samples transferred to the D/A FIFO.

The value stored in *Status* depends on the settings in the Status word, as shown in the following diagram:



K_DMAStatus (cont.)

The bits are described as follows:

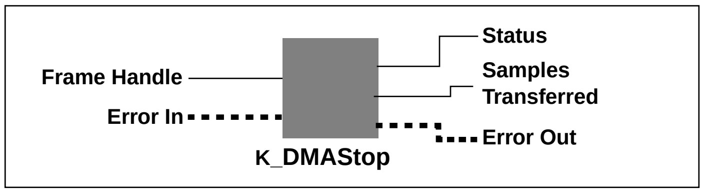
- Bit 0: Indicates whether a DMA-mode operation is in progress.
- Bit 1: The Buffer A/B active bit. If you are using multiple buffers, this bit toggles each time a buffer is switched. If you are using a single buffer, this bit is always 0.
- Bit 2: For analog input operations, this bit indicates whether the onboard A/D FIFO overflowed. For analog output operations, this bit indicates whether the onboard D/A FIFO underflowed. The overflow or underflow event automatically stops all conversions.
- Bit 3: Not used for DMA mode.
- Bit 4: This bit is used during continuous buffering mode. For analog input operations, this bit is set when all buffers that are currently assigned to the active operation have been filled with data at least once. For analog output operations, this bit is set when all buffers that are currently assigned to the active operation have been emptied at least once.
- Bit 5: Unassigned
- Bits 6-7: For analog input operations, these bits indicate the state of the about trigger. For analog output operations, these bits are not used.
- Bits 8-15: In multiple-buffer operations, 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

Purpose Stops a DMA-mode operation or a recycle-mode operation.

Description This VI stops the DMA-mode operation or a recycle-mode operation defined by *Frame Handle* and stores the status of the operation in *Status.*



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Status*	Status of operation. Valid values: Refer to page 4-23 for the meaning of the value stored.
U32	*Samples Transferred*	Number of samples that were transferred into the current buffer.
	Error In	Error information.
	Error Out	Error information.

Remarks For analog input operations, *Samples Transferred* stores the number of samples acquired into the current buffer. For analog output operations, *Samples Transferred* stores the number of samples transferred to the D/A FIFO.

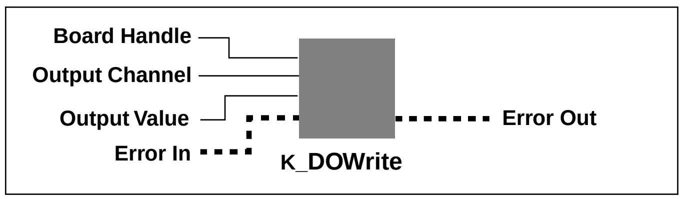
If a DMA or recycle operation is not in progress, **K_DMAStop** is ignored.

See Also K_DMAStart, K_DMAStatus

K_DOWrite

Purpose Writes a single digital output value to the digital output channel.

Description This VI writes the value *Output Value* to the digital output lines on the board specified by *Board Handle*.



Parameters

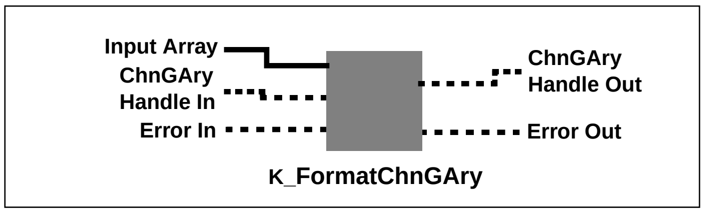
U32	*Board Handle*	Handle associated with the board.
U16	*Output Channel*	Digital output channel. Valid value: **0**
U32	*Output Value*	Digital output value. Valid values: **0** to **15**
	Error In	Error information.
	Error Out	Error information.

Remarks The value to be written is stored in bits 0 through 3; the values in the remaining bits of *Output Value* are not defined. Refer to page 2-37 for more information.

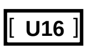
See Also K_IntStart

Purpose Initializes a channel-gain array.

Description For the channel-gain data represented by *Input Array*, this VI initializes the handle *ChnGAry Handle In* and outputs the handle *ChnGAry Handle Out*.



Parameters

	*Input Array*	LabVIEW array with channel-gain data.
	*ChnGAry Handle In*	Handle associated with the allocated channel-gain array.
	*ChnGAry Handle Out*	Initialized handle associated with the allocated channel-gain array.
	*Error In*	Error information.
	*Error Out*	Error information.

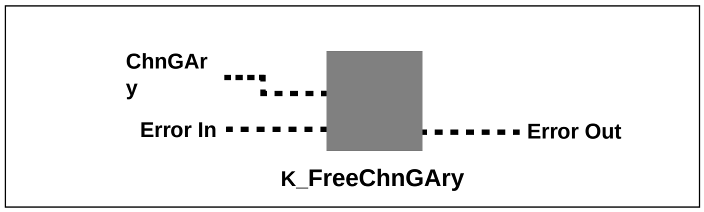
Remarks Refer to page 2-12 for information on setting up a channel-gain array for analog input operations: refer to page 2-27 for information on setting up a channel-gain array for analog output operations.

See Also K_AllocChnGAry, K_FreeChnGAry, K_SetChnGAry

K_FreeChnGAry

Purpose Frees space previously allocated for a channel-gain array.

Description This VI frees the space previously allocated for the channel-gain array defined by *ChnGAry Handle*.



Parameters

ChnGAry Handle	Handle to the channel-gain array.
Error In	Error information.
Error Out	Error information.

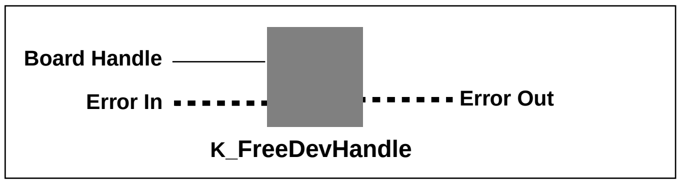
Remarks Refer to page 2-12 for information on setting up a channel-gain array for analog input operations: refer to page 2-27 for information on setting up a channel-gain array for analog output operations.

See Also K_AllocChnGAry, K_FormatChnGAry, K_SetChnGAry

K_FreeDevHandle

Purpose Frees a previously specified board handle.

Description This VI frees the board handle specified by *Board Handle* as well as all frame handles associated with *Board Handle*.



Parameters

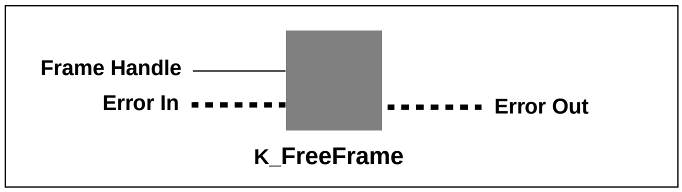
U32	*Board Handle*	Board handle you want to free.
	*Error In*	Error information.
	*Error Out*	Error information.

See Also K_GetDevHandle

K_FreeFrame

Purpose Frees a frame.

Description This VI frees the frame specified by *Frame Handle*, making the frame available for another operation.



Parameters

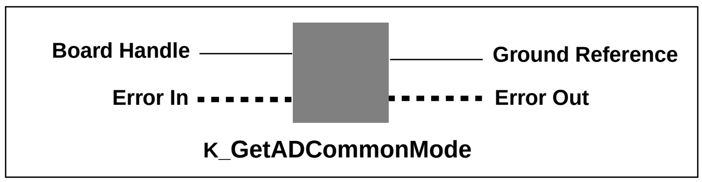
	*Frame Handle*	Handle to frame you want to free.
	*Error In*	Error information.
	*Error Out*	Error information.

See Also K_GetADFrame, K_GetDAFrame, K_GetDIFrame, K_GetDOFrame

K_GetADCommonMode

Purpose Gets the A/D common-mode ground reference.

Description For the board specified by *Board Handle*, this VI stores the code that represents the A/D common-mode ground reference in *Ground Reference*.



Parameters

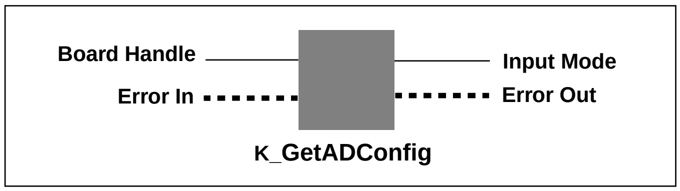
U32	*Board Handle*	Handle to the board that defines the operation.
U16	*Ground Reference*	A/D common-mode ground reference. Value stored: **0** for LL-GND **1** for user-defined
	Error In	Error information.
	Error Out	Error information.

See Also K_SetADCommonMode

K_GetADConfig

Purpose Gets the A/D input channel configuration.

Description This VI stores the code that represents the A/D input channel configuration in *Input Mode* for the board specified by *Board Handle*.



Parameters

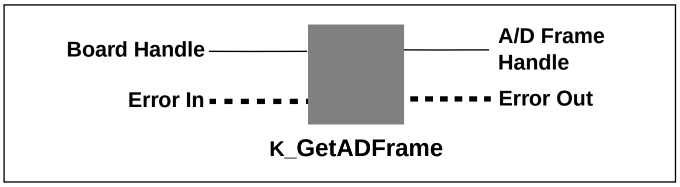
U32	*Board Handle*	Handle associated with the board.
U16	*Input Mode*	A/D input channel configuration. Value stored: **0** for Differential **1** for Single-ended
	*Error In*	Error information.
	*Error Out*	Error information.

See Also K_SetADConfig

K_GetADFrame

Purpose Accesses an A/D frame for an analog input operation.

Description This VI specifies that you want to perform a DMA-mode or interrupt-mode analog input operation on the board specified by *Board Handle*, and accesses an available A/D frame with the handle *A/D Frame Handle*.



Parameters

	*Board Handle*	Handle associated with the board.
	*A/D Frame Handle*	Handle to the frame that defines the operation.
	*Error In*	Error information.
	*Error Out*	Error information.

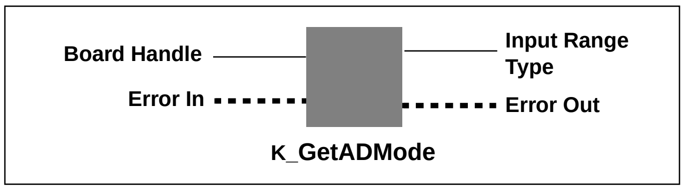
Remarks The frame is initialized to its default settings; the default settings are given in Table 3-1 on page 3-4.

See Also K_ClearFrame, K_FreeFrame

K_GetADMode

Purpose Gets the A/D input range type.

Description This VI stores the code that represents the A/D input range type for the board specified by *Board Handle* in *Input Range Type.*



Parameters

U32	*Board Handle*	Handle associated with the board.
U16	*Input Range Type*	A/D input range type. Value stored: **0** for Bipolar **1** for Unipolar
	Error In	Error information.
	Error Out	Error information.

See Also K_SetADMode

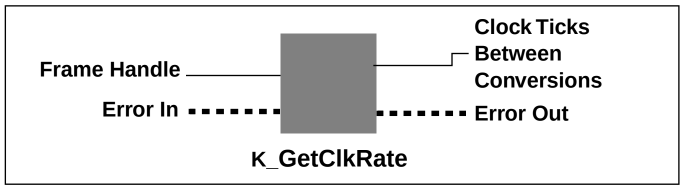
K_GetClkRate

Purpose

For analog input operations, gets the number of clock ticks used by the internal A/D pacer clock source. For analog output operations, gets the number of clock ticks used by the internal D/A pacer clock source.

Description

For the operation defined by *Frame Handle*, this VI stores the number of clock ticks between conversions in *Clock Ticks Between Conversions*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U32	*Clock Ticks Between Conversions*	Number of clock ticks between conversions. Value stored: **15** to **4,294,967,295** for A/D pacer clock **1** to **655,350** for D/A pacer clock
	Error In	Error information.
	Error Out	Error information.

Remarks

The *Clock Ticks Between Conversions* variable contains the value of the Pacer Clock Rate element. After an interrupt-mode, DMA-mode, or recycle-mode operation, the value stored in *Clock Ticks Between Conversions* represents the actual count used, not necessarily the count set by **K_SetClkRate**.

For A/D frames, this VI applies to the 5 MHz internal A/D pacer clock source only. The tick resolution is 2 µs.

For D/A frames, this VI applies to the 5 MHz internal D/A pacer clock source only. The tick resolution is 2 µs.

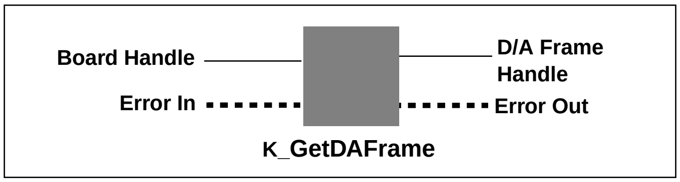
K_GetClkRate (cont.)

See Also K_SetClkRate

K_GetDAFrame

Purpose Accesses a D/A frame for an analog output operation.

Description This VI specifies that you want to perform a DMA-mode, an interrupt-mode, or a recycle-mode analog output operation on the board specified by *Board Handle*, and accesses an available D/A frame with the handle *D/A Frame Handle*.



Parameters

U32	*Board Handle*	Handle associated with the board.
U32	*D/A Frame Handle*	Handle to the frame that defines the analog output operation.
	Error In	Error information.
	Error Out	Error information.

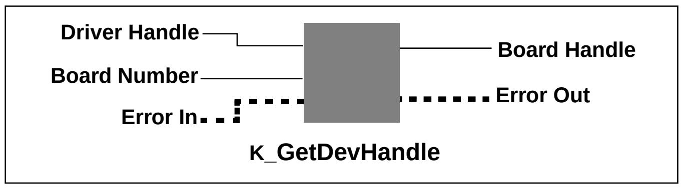
Remarks The frame is initialized to its default settings; the default settings are given in Table 3-2 on page 3-6.

See Also K_FreeFrame, K_ClearFrame

K_GetDevHandle

Purpose Initializes any Keithley DAS board.

Description This VI initializes the board associated with *Driver Handle* and specified by *Board Number*, and stores the board handle of the specified board in *Board Handle*.



Parameters

U32	*Driver Handle*	Handle of the associated VI Driver.
U16	*Board Number*	Board number. Valid values: **0** to **2**
U32	*Board Handle*	Handle associated with the board.
	Error In	Error information.
	Error Out	Error information.

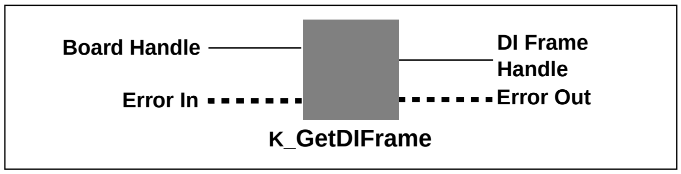
Remarks The value stored in *Board Handle* is intended to be used exclusively as an input to VIs that require a board handle. Your program should not modify the value stored in *Board Handle*.

See Also K_FreeDevHandle

K_GetDIFrame

Purpose Accesses a DI frame for a digital input operation.

Description This VI specifies that you want to perform an interrupt-mode digital input operation on the board specified by *Board Handle*, and accesses an available digital input frame with the handle *DI Frame Handle*.



Parameters

U32	*Board Handle*	Handle associated with the board.
U32	*DI Frame Handle*	Handle to the frame that defines the digital input operation.
	Error In	Error information.
	Error Out	Error information.

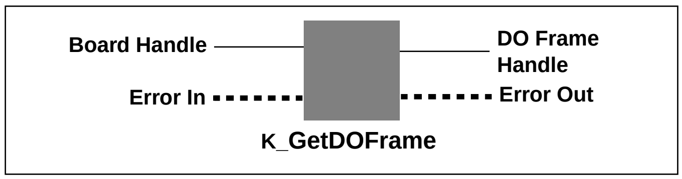
Remarks The frame is initialized to its default settings; the default settings are given in Table 3-3 on page 3-8.

See Also K_FreeFrame, K_ClearFrame

K_GetDOFrame

Purpose Accesses a DO frame for a digital output operation.

Description This VI specifies that you want to perform an interrupt-mode digital output operation on the board specified by *Board Handle* and accesses an available digital output frame with the handle *DO Frame Handle*.



Parameters

U32	*Board Handle*	Handle associated with the board.
U32	*DO Frame Handle*	Handle to the frame that defines the digital output operation.
	Error In	Error information.
	Error Out	Error information.

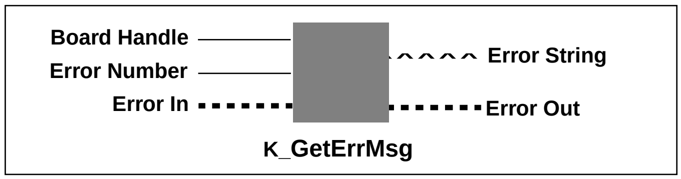
Remarks The frame is initialized to its default settings; the default settings are given in Table 3-4 on page 3-9.

See Also K_FreeFrame, K_ClearFrame

K_GetErrMsg

Purpose Gets an error message string.

Description For the board specified by *Board Handle*, this VI outputs the error message string *Error String* corresponding to the error message number represented by *Error Number*.



Parameters

U32	*Board Handle*	Handle associated with the board.
U16	*Error Number*	Error message number.
abc	*Error String*	Error message string.
	Error In	Error information.
	Error Out	Error information.

Remarks Refer to page 2-3 for information about error handling. Refer to Appendix A for a list of error codes and their meanings.

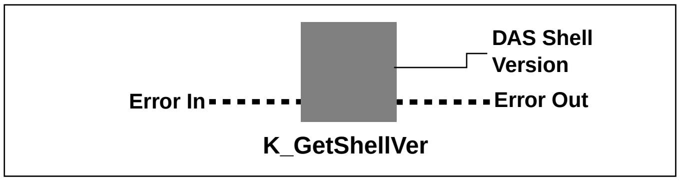
K_GetShellVer

Purpose

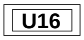
Gets the current DAS shell version.

Description

This VI stores the major version number and the minor version number of the current DAS shell in *DAS Shell Version*.



Parameters



DAS Shell Version A word value containing the major and minor version numbers of the DAS shell.

Error In Error information.

Error Out Error information.

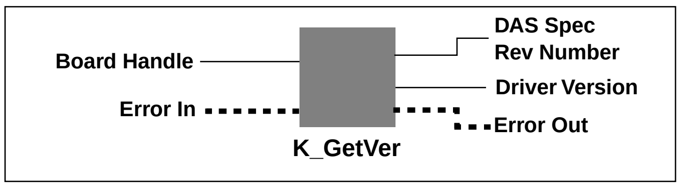
Remarks

To obtain the major version number of the DAS shell, divide *DAS Shell Version* by 256. To obtain the minor version number of the DAS shell, perform a Boolean AND operation with *DAS Shell Version* and 255 (0FF hex).

K_GetVer

Purpose Gets revision numbers.

Description For the board specified by *Board Handle*, this VI stores the revision number of the DAS-1800 Series VI Driver in *Driver Version* and the revision number of the driver specification in *DAS Spec Rev Number*.



Parameters

U32	*Board Handle*	Handle associated with the board.
U16	*DAS Spec Rev Number*	Revision number of the Keithley DAS Driver Specification to which the driver conforms.
U16	*Driver Version*	Driver version number.
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks The high byte of *DAS Spec Rev Number* and *Driver Version* 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 *Driver Version* contains the value of **2** (512) and the low byte of *Driver Version* 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 *Driver Version* or *DAS Spec Rev Number*, use the following equations:

$$\text{major revision level} = \text{integer portion of}\left(\frac{\text{returned value}}{256}\right)$$

$$\text{minor revision level} = \text{returned value MOD } 256$$

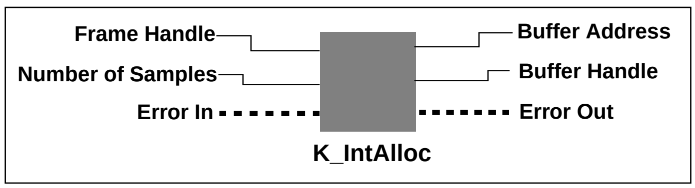
K_IntAlloc

Purpose

Allocates a buffer for an interrupt-mode operation.

Description

For the operation defined by *Frame Handle*, this VI allocates a buffer of the size specified by *Number of Samples*, and stores the starting address of the buffer in *Buffer Address* and the handle to the buffer in *Buffer Handle*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U32	*Number of Samples*	Number of samples. Valid values: **1** to **65,536**
U32	*Buffer Address*	Starting address of the allocated buffer.
U16	*Buffer Handle*	Handle associated with the allocated buffer.
	Error In	Error information.
	Error Out	Error information.

Remarks

Use **K_SetBuf** or **K_BufListAdd** to assign *Buffer Address* to the frame that defines the operation. *Buffer Handle*, as returned by this VI, is later used to free the allocated memory block when used with **K_IntFree**.

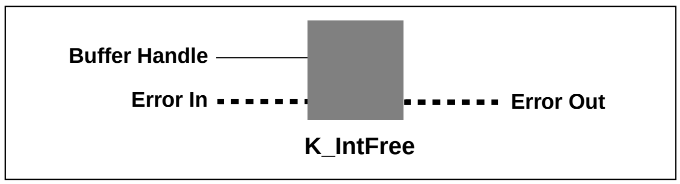
See Also

K_IntFree, K_SetBuf, K_BufListAdd

K_IntFree

Purpose Frees a buffer allocated for an interrupt-mode operation.

Description This VI frees the buffer specified by *Buffer Handle*; the buffer was previously allocated dynamically using **K_IntAlloc**.



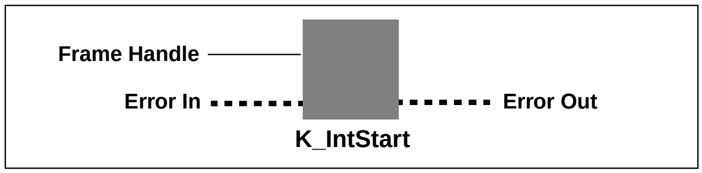
Parameters

	*Buffer Handle*	Handle to interrupt buffer.
	*Error In*	Error information.
	*Error Out*	Error information.

See Also K_IntAlloc

Purpose Starts an interrupt-mode operation or a recycle-mode operation.

Description This VI starts the interrupt-mode operation or recycle-mode operation defined by *Frame Handle*.



Parameters

	*Frame Handle*	Handle to the frame that defines the operation.
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks Refer to page 2-24 for more information on interrupt operations and on recycling data from the D/A FIFO.

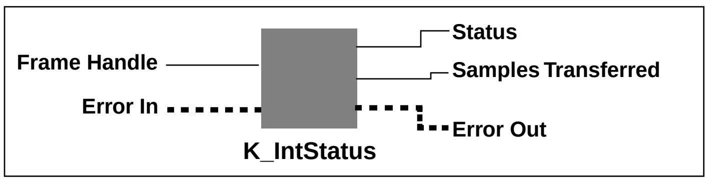
For analog output operations, if the user-defined buffer contains less than 2048 samples, the DAS-1800AO Series board does not use the interrupt resources of the board; this allows the board to provide the maximum transfer rate (up to 500 kHz). However, your program must still use **K_IntStart** to start the operation regardless of whether the interrupt resources are used.

See Also K_IntStatus, K_IntStop

K_IntStatus

Purpose Gets the status of an interrupt-mode operation or recycle-mode operation.

Description For the interrupt-mode operation or recycle-mode operation defined by *Frame Handle*, this VI stores the status of the operation in *Status* and the number of samples transferred in *Samples Transferred*.

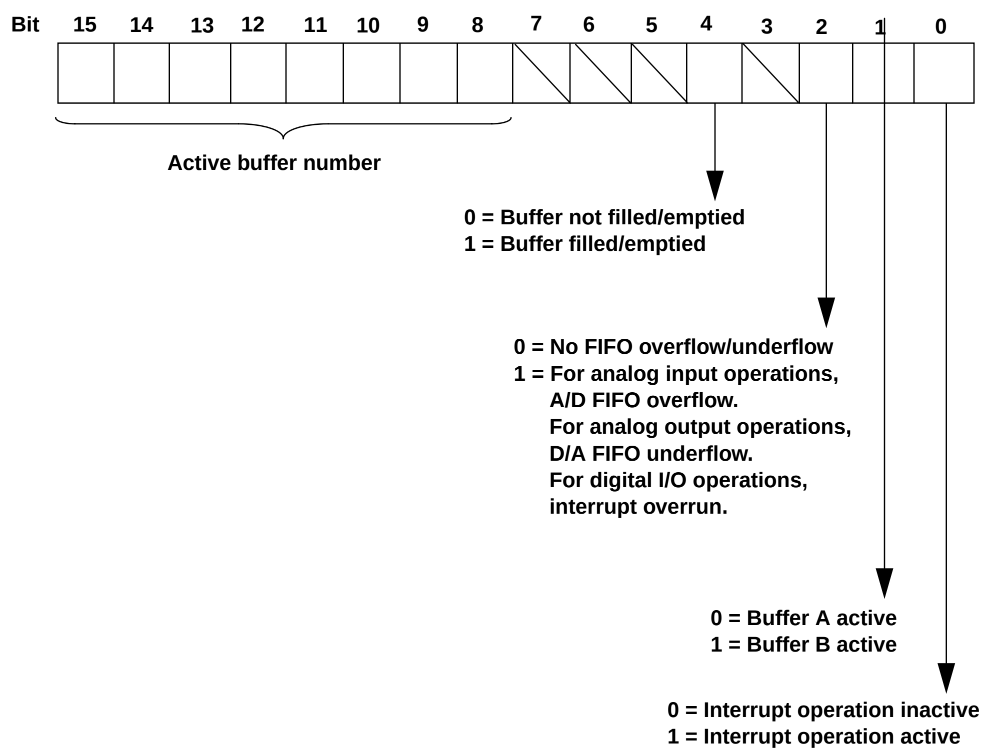


Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Status*	Status of interrupt-mode operation or recycle-mode operation. Valid stored: see **Remarks** below for value stored.
U32	*Samples Transferred*	Number of samples transferred.
	Error In	Error information.
	Error Out	Error information.

Remarks For input operations, *Samples Transferred* stores the number of samples acquired into the current buffer. For output operations, *Samples Transferred* stores the number of samples transferred from the current buffer.

The value stored in *Status* depends on the settings in the Status word, as shown in the following illustration:



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 a buffer is switched. If you are using a single buffer, this bit is always 0.
- Bit 2: For analog input operations, this bit indicates whether the onboard A/D FIFO overflowed. For analog output operations, this bit indicates whether the onboard D/A FIFO underflowed. The overflow or underflow event automatically stops all conversions. For digital I/O operations, this bit indicates that the board issued an interrupt while the CPU was processing a previous interrupt from the same board.
- Bit 3: Reserved.
- Bit 4: This bit is used during continuous buffering mode. For input operations, this bit is set when all buffers that are currently assigned to the active operation have been filled with data at least once. For output operations, this bit is set when all buffers that are currently assigned to the active operation have been emptied at least once.
- Bits 5-7: Unassigned.
- Bits 8-15: In multiple-buffer operations, 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

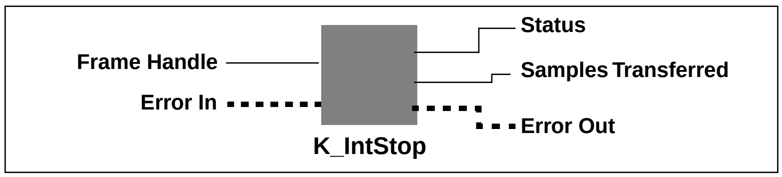
K_IntStop

Purpose

Stops an interrupt-mode operation or a recycle-mode operation.

Description

This VI stops the interrupt-mode operation or recycle-mode operation defined by *Frame Handle* and stores the status of the operation in *Status* and the number of samples transferred in *Samples Transferred*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Status*	Status of operation. Value stored: refer to page 4-49 for the meaning of the value stored.
U32	*Samples Transferred*	Number of samples transferred.
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks

For input operations, *Samples Transferred* stores the number of samples acquired into the current buffer. For output operations, *Samples Transferred* stores the number of samples written from the current buffer.

If an interrupt-mode operation or recycle-mode operation is not in progress, **K_IntStop** is ignored.

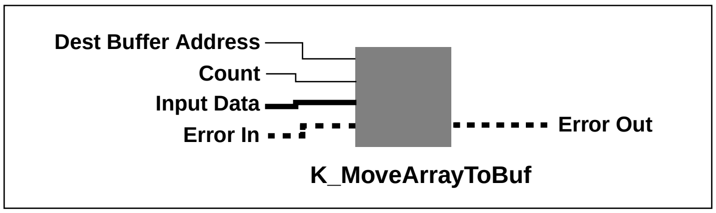
See Also

K_IntStart, K_IntStatus

K_MoveArrayToBuf

Purpose Moves the contents of a LabVIEW array to an allocated buffer.

Description This VI transfers the number of samples represented by *Count* from the array represented by *Input Data* to the buffer at address *Dest Buffer Address*.



Parameters

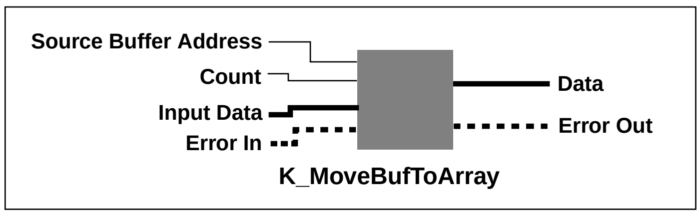
U32	*Dest Buffer Address*	Address of destination buffer.
U32	*Count*	Number of samples to transfer. Valid values: **1** to **32,767**
[I16]	*Input Data*	Source array containing the data to transfer.
	Error In	Error information.
	Error Out	Error information.

See Also K_DMAAlloc, K_IntAlloc, K_MoveBufToArray

K_MoveBufToArray

Purpose Moves the contents of an allocated buffer to a LabVIEW array.

Description This VI transfers the number of samples represented by *Count* from the buffer at address *Source Buffer Address* to the LabVIEW array *Input Data* and returns the filled array in *Data*.



Parameters

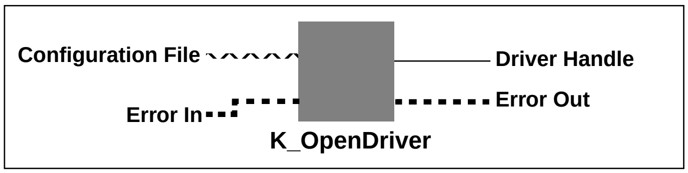
U32	*Source Buffer Address*	Address of source buffer.
U32	*Count*	Number of samples to transfer. Valid values: **1** to **32,767**
[I16]	*Input Data*	Array used to store data from the source buffer.
[I16]	*Data*	LabVIEW array containing data from an allocated buffer.
	Error In	Error information.
	Error Out	Error information.

See Also K_DMAAlloc, K_IntAlloc, K_MoveArrayToBuf

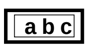
K_OpenDriver

Purpose Initializes a Keithley DAS VI Driver.

Description This VI initializes the DAS VI Driver according to the information in the configuration file specified by *Configuration File*, and stores the driver handle in *Driver Handle*.



Parameters

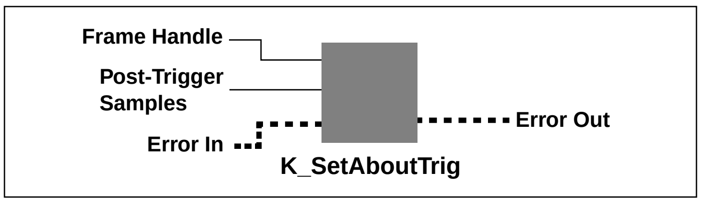
	*Configuration File*	Driver configuration file. Valid values: The name of a configuration file. **Null string** if driver has already been opened (see **Remarks** below).
	*Driver Handle*	Handle associated with the driver.
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks If *Configuration File* = Null, **K_OpenDriver** checks whether the driver has already been opened and linked to a configuration file and if it has, uses the current configuration. You create a configuration file using the configuration utility. Refer to your DAS-1800 Series board user's guide for more information.

The value stored in *Driver Handle* is intended to be used exclusively as an input to VIs that require a driver handle. Your program should not modify the value stored in *Driver Handle*.

Purpose Enables the about trigger and specifies the number of samples after the trigger occurs.

Description For the DMA-mode analog input operation defined by *Frame Handle*, this VI enables the about trigger and specifies the number of samples after the trigger occurs.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U32	*Post-Trigger Samples*	Number of post-trigger samples. Valid values: **1** to **65,535**
	Error In	Error information.
	Error Out	Error information.

See Also K_ClrAboutTrig

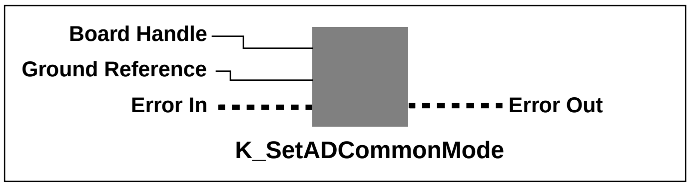
K_SetADCommonMode

Purpose

Sets the A/D common-mode ground reference.

Description

For the board specified by *Board Handle*, this VI specifies the A/D common-mode ground reference in *Ground Reference*.



Parameters

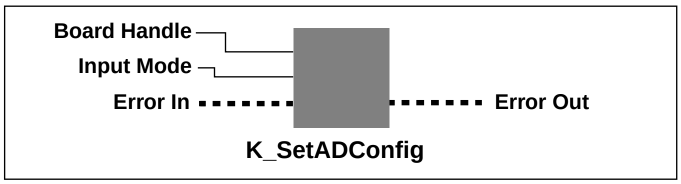
U32	*Board Handle*	Handle to the board that defines the operation.
U16	*Ground Reference*	A/D common-mode ground reference. Value stored: **0** for LL-GND **1** for user-defined
	Error In	Error information.
	Error Out	Error information.

See Also

K_GetADCommonMode

Purpose Sets the A/D input channel configuration.

Description This VI specifies the A/D input channel configuration in *Input Mode* for the board specified by *Board Handle*.



Parameters

U32	*Board Handle*	Handle associated with the board.
U16	*Input Mode*	A/D input channel configuration. Value stored: **0** for Differential **1** for Single-ended
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks If an SSH-8 or EXP-1800 accessory is enabled in the configuration file, any use of **K_SetADConfig** that attempts to set the A/D channel configuration to differential returns error 8001.

See Also K_GetADConfig

K_SetADFreeRun

Purpose Specifies burst conversion mode.

Description This VI sets the conversion mode for the operation defined by *Frame Handle* to burst mode.

Parameters

	*Frame Handle*	Handle to the frame that defines the operation.
	*Error In*	Error information.
	*Error Out*	Error information.

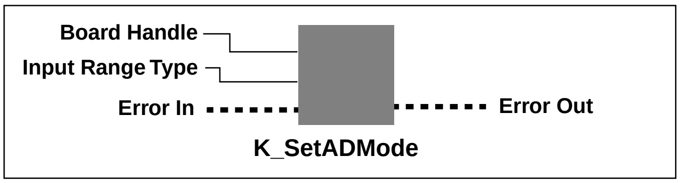
Remarks Refer to page 2-13 for information on conversion modes.

See Also K_ClrADFreeRun

K_SetADMode

Purpose Sets the A/D input range type.

Description For the board specified by *Board Handle*, this VI specifies the A/D input range type in *Input Range Type.*



Parameters

U32	*Board Handle*	Handle associated with the board.
U16	*Input Range Type*	A/D input range type. Valid values: **0** for Bipolar **1** for Unipolar
	*Error In*	Error information.
	*Error Out*	Error information.

See Also K_GetADMode

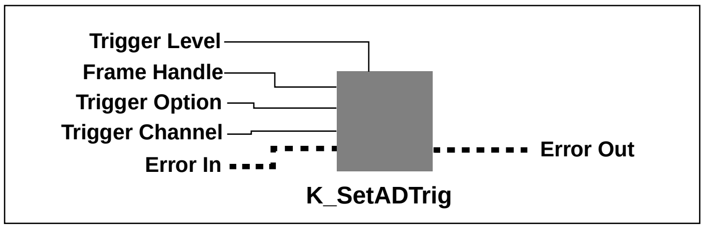
K_SetADTrig

Purpose

Sets up an analog start trigger.

Description

For the operation defined by *Frame Handle*, this VI specifies the channel used for an analog trigger in *Trigger Channel*, the level used for the analog trigger in *Trigger Level*, and the trigger polarity and trigger sensitivity in *Trigger Option*.



Parameters

U32 *Frame Handle* Handle to the frame that defines the operation.

U16 *Trigger Option* Analog trigger polarity.
Valid values: **0** for Positive edge
2 for Negative edge

U16 *Trigger Channel* Analog input channel used as trigger channel. Valid channel numbers are shown below:

Board Configuration	Valid Channel Numbers	
	Differential	**Single-ended**
DAS-1800AO Series board	**0** to **7**	**0** to **15**
DAS-1800AO Series board with ***N*** EXP-1800 expansion boards attached	Not applicable	**0** to **15(*N* + 1)**

	*Trigger Level*	Level at which the trigger event occurs, specified in raw counts. Valid values : **0** to **4,095** for Unipolar –**2048** to **2047** for Bipolar
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks

Trigger Option sets the value of the Trigger Polarity and Trigger Sensitivity elements: *Trigger Channel* sets the value of the Trigger Channel element: *Trigger Level* sets the value of the Trigger Level element.

You specify the value for *Trigger Level* in raw counts. Refer to Appendix B for information on converting voltage to a raw count.

K_SetADTrig does not affect the operation defined by *Frame Handle* unless the Trigger Source element is set to external (using **K_SetTrig**) before *Frame Handle* is used as an input to **K_IntStart** or **K_DMAStart**.

See Also

K_SetTrig

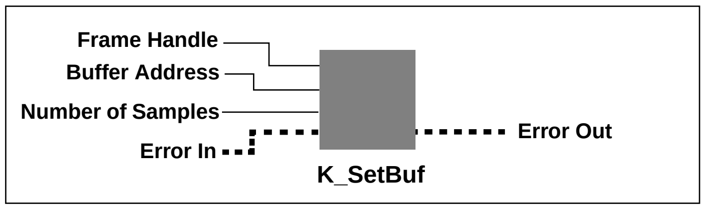
K_SetBuf

Purpose

Specifies the starting address of a previously allocated buffer.

Description

For the operation defined by *Frame Handle*, this VI specifies the starting address of a previously allocated buffer in *Buffer Address* and the number of samples (the size of the buffer) in *Number of Samples*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U32	*Buffer Address*	Starting address of buffer.
U32	*Number of Samples*	Number of samples. Valid values: **0** to **65,535**
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks

Use this VI for buffers allocated using **K_IntAlloc**. For buffers allocated using **K_DMAAlloc**, use **K_SetDMABuf**.

Do not use this VI if you are using multiple buffers: use **K_BufListAdd** to specify the starting addresses of multiple buffers.

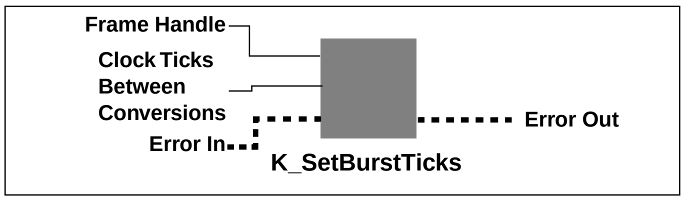
Buffer Address sets the value of the Buffer element: *Number of Samples* sets the value of the Number of Samples element.

See Also

K_DMAAlloc, K_IntAlloc, K_BufListAdd, K_SetDMABuf

Purpose Sets the burst mode conversion rate.

Description For the operation defined by *Frame Handle*, this VI stores the number of clock ticks between conversions of each channel in a scan in *Clock Ticks Between Conversions*.



Parameters

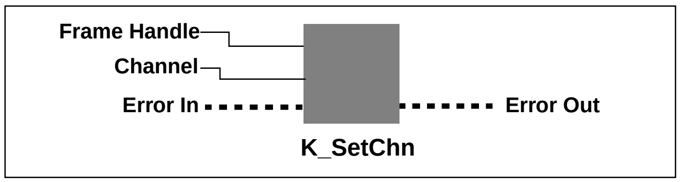
U32	*Frame Handle*	Handle to the frame that defines the A/D operation.
U16	*Clock Ticks Between Conversions*	The number of clock ticks between conversions of each channel in a scan. Valid values: **3** to **63**
	Error In	Error information.
	Error Out	Error information.

Remarks Refer to page 2-15 for more information on burst mode conversion rate.

K_SetChn

Purpose Specifies a single channel.

Description For the operation defined by *Frame Handle*, this VI specifies the single channel used in *Channel*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Channel*	Channel on which to perform operation. Valid channel numbers are shown below:

Operation	**Valid Channel Numbers**
Analog input; no EXP-1800 expansion boards attached	Differential: **0** to **7** Single-ended: **0 to 15**
Analog input; ***N*** EXP-1800 expansion boards attached	Differential: Not applicable Single-ended: **0** to **15(*N* + 1)**
Analog output	**0** = DAC 0 **1** = DAC 1

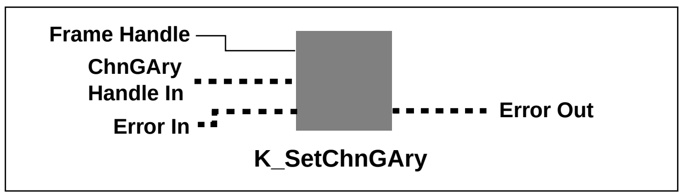
 *Error In* Error information.

 *Error Out* Error information.

Remarks The value you specify in *Channel* sets the Start Channel element and the Stop Channel element in the frame identified by *Frame Handle*.

Purpose Sets the channel-gain array element.

Description For the operation defined by *Frame Handle*, this VI sets the channel-gain array element defined by *ChnGAry Handle In*.



Parameters

	*Frame Handle*	Handle to the frame that defines the operation.
	*ChnGAry Handle In*	Handle to the channel-gain array.
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks For analog input operations, the maximum number of channel-gain entries is 256. Refer to page 2-12 for information on setting up a channel-gain array for analog input operations.

For analog output operations, the maximum number of channel-gain entries is two. In addition, you cannot read channel 0 after channel 1. Refer to page 2-27 for information on setting up a channel-gain array for analog output operations.

See Also K_AllocChnGAry, K_FormatChnGAry, K_FreeChnGAry

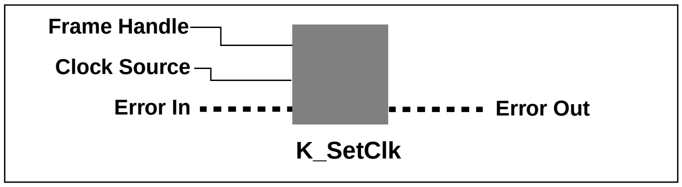
K_SetClk

Purpose

Specifies the pacer clock source.

Description

For the operation defined by *Frame Handle*, this VI specifies the pacer clock source in *Clock Source*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Clock Source*	Pacer clock source. Valid values: **0** for Internal **1** for External
	Error In	Error information.
	Error Out	Error information.

Remarks

For A/D, DI, and DO frames, the internal clock source is the internal A/D pacer clock. For D/A frames, the internal clock source is the internal D/A pacer clock.

If you want to pace an analog output operation using the internal A/D pacer clock source, set the clock source to internal, then use **K_SetSync**.

The external pacer clock source is an external signal connected to the XPCLK pin; this clock source can pace either an analog input or an analog output operation.

Refer to page 2-13 (for analog input operations), page 2-28 (for analog output operations), and page 2-38 (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.

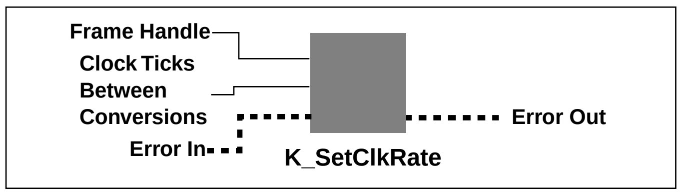
K_SetClkRate

Purpose

For analog input operations, specifies the number of clock ticks used by the internal A/D pacer clock. For analog output operations, specifies the number of clock ticks used by the internal D/A pacer clock.

Description

For the operation defined by *Frame Handle*, this VI specifies the number of clock ticks between conversions in *Clock Ticks Between Conversions*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U32	*Clock Ticks Between Conversions*	Number of clock ticks. Valid values: **15** to **4,294,967,295** for A/D pacer clock **1** to **655,350** for D/A pacer clock
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks

The value you specify in *Clock Ticks Between Conversions* sets the Pacer Clock Rate element in the frame identified by *Frame Handle*.

For analog input frames, this VI applies to the 5 MHz internal A/D pacer clock source only. The tick resolution is 0.2 µs.

For analog output frames, this VI applies to the 5 MHz internal D/A pacer clock source only. The tick resolution is 0.2 µs. The driver enables the optional divide-by-ten prescaler based on the value of *Clock Ticks Between Conversions.*

When synchronizing D/A conversions with A/D conversions, the sampling rate of the analog output operation is determined by the A/D pacer clock source. Use **K_SetClkRate**, specifying an A/D frame, to set the sampling rate of a synchronized analog output operation. Use **K_SetClk** to specify the clock source as internal.

Refer to page 2-13 for more information on the internal A/D pacer clock. Refer to page 2-28 for more information on the internal D/A pacer clock.

See Also K_GetClkRate

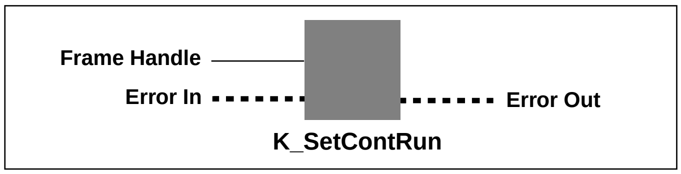
K_SetContRun

Purpose

Specifies continuous buffering mode.

Description

For the operation defined by *Frame Handle*, this VI sets the buffering mode to continuous mode and sets the Buffering Mode element in the frame accordingly.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
	Error In	Error information.
	Error Out	Error information.

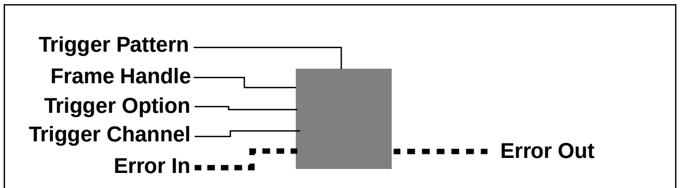
Remarks

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame**, and **K_ClearFrame** specify single-cycle as the default buffering mode.

Operation	See
Analog input	page 2-16
Analog output	page 2-30
Digital I/O	page 2-39

Purpose Sets up an external digital trigger.

Description This VI specifies the digital trigger polarity in *Trigger Option* for the operation defined by *Frame Handle*.



K_SetDITrig

Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Trigger Option*	Trigger polarity and sensitivity. Valid values: **0** for Positive-edge, pre-, post-, or about-trigger operation **2** for Negative-edge, pre-, post-, or about-trigger operation **4** for Positive-edge, retrigger operation **6** for Negative-edge, retrigger operation
U16	*Trigger Channel*	Digital input channel. Valid value: **0**
U32	*Trigger Pattern*	Trigger pattern.
	Error In	Error information.
	Error Out	Error information.

K_SetDITrig (cont.)

Remarks

Trigger Option sets the value of the Trigger Polarity and Trigger Sensitivity elements used to either start a pre-, post-, or about-trigger analog input operation or to retrigger an analog output operation.

Trigger Channel sets the value of the Trigger Channel element.

Trigger Pattern sets the value of the Trigger Pattern element. Since the DAS-1800 Series VI driver does not currently support digital pattern triggering, *Trigger Pattern* is not used. It is provided for future compatibility.

K_SetDITrig does not affect the operation defined by *Frame Handle* unless the Trigger Source element is set to External (using **K_SetTrig**) before *Frame Handle* is used as an input to **K_IntStart** or **K_DMAStart**. Additionally, if you want to retrigger a waveform from the D/A FIFO, the Buffering Mode element must be set to Continuous (using **K_SetContRun**) before *Frame Handle* is used as a input to **K_IntStart** or **K_DMAStart**.

See Also

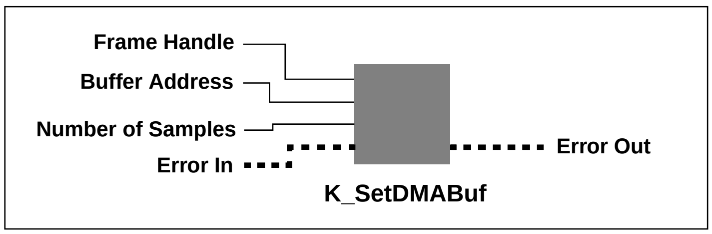
K_SetTrig

Purpose

Sets the values of a DMA buffer address and the Number of Samples element.

Description

For the operation specified by *Frame Handle*, this VI stores the address of the currently allocated buffer in *Buffer Address* and the number of samples stored in the buffer in *Number of Samples*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the DMA-mode operation.
U32	*Buffer Address*	Starting address of buffer.
U32	*Number of Samples*	Number of samples. Valid values: **0** to **65,535**
	Error In	Error information.
	Error Out	Error information.

Remarks

Use this VI for buffers allocated using **K_DMAAlloc**. For buffers allocated using **K_IntAlloc**, use **K_SetBuf**.

Buffer Address contains the value of the Buffer element.

Number of Samples contains the value of the Number of Samples element.

See Also

K_DMAAlloc, K_BufListAdd

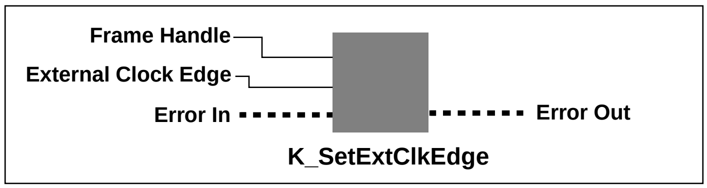
K_SetExtClkEdge

Purpose

Specifies the active edge of the external pacer clock.

Description

For the operation defined by *Frame Handle*, this VI sets the active edge of the external pacer clock to the value represented by *External Clock Edge* and sets the External Clock Edge element in the frame accordingly.



Parameters

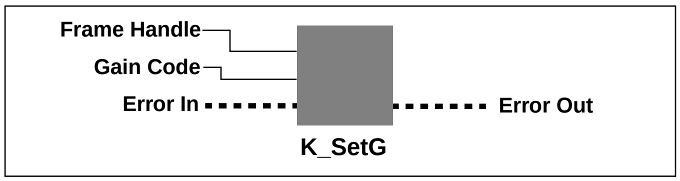
U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*External Clock Edge*	Active edge of external pacer clock. Valid values: **0** for Negative edge **1** for Positive edge
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks

K_SetExtClkEdge does not affect the operation defined by *Frame Handle* unless the Trigger Source element is set to External (using **K_SetTrig**) before *Frame Handle* is used as an input to **K_IntStart** or **K_DMAStart**.

Purpose Sets the gain code for an analog input or analog output operation.

Description For the operation defined by *Frame Handle*, this VI specifies the gain code for a single channel or for a group of consecutive channels in *Gain Code*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Gain Code*	Valid values: **0** to **3** for DAS board channels **0** to **7** for EXP-1800 channels
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks This VI is valid for A/D and D/A frames.

The value you specify in *Gain Code* sets the Gain element in the frame identified by *Frame Handle*.

For analog input operations, refer to Table 2-2 on page 2-7 for the gain and input range associated with each gain code. For analog output operations, refer to Table 2-3 on page 2-26 for the range associated with each gain code

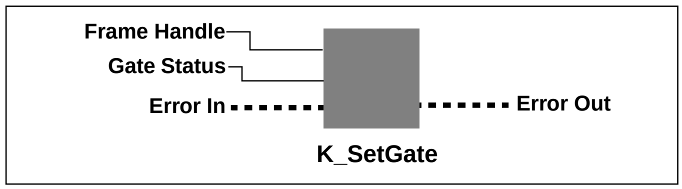
K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame,** and **K_ClearFrame** specify 0 as the default gain code.

See Also K_SetStartStopG

K_SetGate

Purpose Specifies the status of the hardware gate.

Description For the operation defined by *Frame Handle*, this VI specifies the status of the hardware gate in *Gate Status*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Gate Status*	Status of the hardware gate. Valid values: **0** for Gate disabled **1** for Positive gate enabled **2** for Negative gate enabled
	Error In	Error information.
	Error Out	Error information.

Remarks For the operation defined by *Frame Handle*, this VI specifies the status of the hardware gate in *Gate Status*.

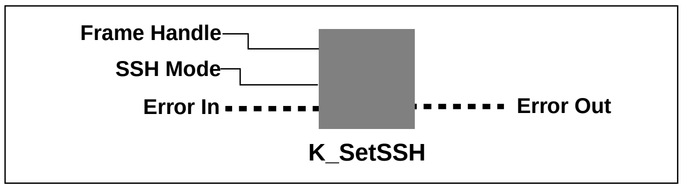
External gating is supported for analog input and analog output operations. You cannot enable the hardware gate if you are using an external digital trigger.

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame,** and **K_ClearFrame** specify disabled as the default gate setting.

K_SetSSH

Purpose Enables or disables SSH mode.

Description For the operation defined by *Frame Handle*, this VI stores the code that indicates the SSH mode in *SSH Mode*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*SSH Mode*	Code that indicates the status of SSH mode. Valid values: **0** for Disabled **1** for Enabled
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks **K_GetADFrame** and **K_ClearFrame** also disable SSH mode.

Refer to page 2-13 for information on SSH mode.

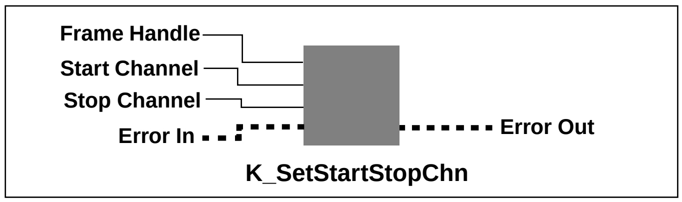
K_SetStartStopChn

Purpose

Specifies the first and last channels in a group of consecutive channels.

Description

For the operation defined by *Frame Handle*, this VI specifies the first channel in a group of consecutive channels in *Start Channel* and the last channel in the group of consecutive channels in *Stop Channel*.



Parameters

U32 *Frame Handle* — Handle to the frame that defines the operation.

U16 *Start Channel* — First channel in a group of consecutive channels. Valid values are shown below:

Operation	**Valid Channel Numbers**
Analog input; no EXP-1800 expansion boards attached	Differential: **0** to **7** Single-ended: **0 to 15**
Analog input; ***N*** EXP-1800 expansion boards attached	Differential: Not applicable Single-ended: **0** to **15(*N* + 1)**
Analog output	**0** for DAC 0 **1** for DAC 1

U16 *Stop Channel* — Last channel in a group of consecutive channels. Valid values: Same as for *Start Channel* above.

Error In — Error information.

Error Out — Error information.

Remarks

For analog input channels, the start channel can be higher than the stop channel. For example, the start channel can be 3 and the stop channel can be 0. Refer to page 2-8 for more information.

For analog output channels, the start channel must be less than or equal to the stop channel. For example, if the start channel is DAC 1, the stop channel must be DAC 1. Refer to page 2-26 for more information.

The values you specify set the following elements in the frame identified by *Frame Handle*:

- *Start Channel* sets the value of the Start Channel element.
- *Stop Channel* sets the value of the Stop Channel element.

K_GetADFrame and **K_ClearFrame** set the Start Channel and Stop Channel elements to 0.

See Also

K_SetStartStopG

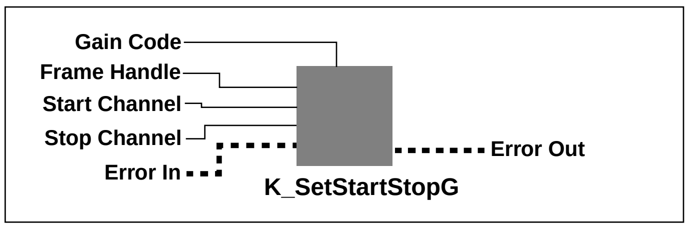
K_SetStartStopG

Purpose

Specifies the first and last channels in a group of consecutive channels and sets the gain code for all channels in the group.

Description

For the operation defined by *Frame Handle*, this VI specifies the first channel in a group of consecutive channels in *Start Channel*, the last channel in a group of consecutive channels in *Stop Channel*, and the gain code for all channels in the group in *Gain Code*.



Parameters

U32 *Frame Handle* Handle to the frame that defines the operation.

U16 *Start Channel* First channel in a group of consecutive channels. Valid values are shown below:

Operation	Valid Channel Numbers
Analog input; no EXP-1800 expansion boards attached	Differential: **0** to **7** Single-ended: **0 to 15**
Analog input; ***N*** EXP-1800 expansion boards attached	Differential: Not applicable Single-ended: **0** to **15(*N*** + **1)**
Analog output	**0** for DAC 0 **1** for DAC 1

U16 *Stop Channel* Last channel in a group of consecutive channels. Valid values: Same as for *Start Channel* above.

U16 *Gain Code* Valid values: **0** to **3** for DAS board channels
0 to **7** for EXP-1800 channels.

 *Error In* Error information.

 *Error Out* Error information.

Remarks

For analog input channels, the start channel can be higher than the stop channel. For example, the start channel can be 3 and the stop channel can be 0. Refer to page 2-8 for more information.

For analog output channels, the start channel must be less than or equal to the stop channel. For example, if the start channel is DAC 1, the stop channel must be DAC 1. Refer to page 2-26 for more information.

The values you specify set the following elements in the frame identified by *Frame Handle*:

- *Start Channel* sets the value of the Start Channel element.
- *Stop Channel* sets the value of the Stop Channel element.
- *Gain Code* sets the value of the Gain element.

For analog input operations, refer to Table 2-2 on page 2-7 for the gain and input range associated with each gain code.

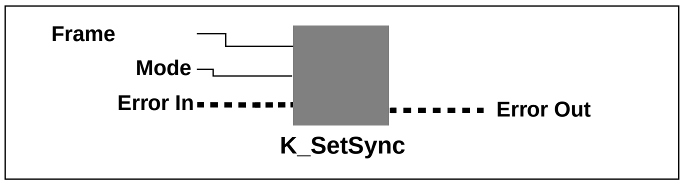
For analog output operations, refer to Table 2-3 on page 2-26 for the range associated with each gain code.

K_GetADFrame and **K_ClearFrame** set the Start Channel, Stop Channel, and Gain elements to 0.

K_SetSync

Purpose Specifies the synchronizing clock source for analog output operations.

Description This VI sets up conversion clock synchronization between the *Frame Handle* and another frame denoted by the mode that is set.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Mode*	Synchronizing clock source. Valid values: **0** for None **1** for the internal A/D pacer clock if the board is performing A/D conversions in paced mode; or the burst mode conversion clock if the board is performing A/D conversions in burst mode.
	*Error In*	Error information.
	*Error Out*	Error information.

Remarks DAS-1800AO Series boards allow you to synchronize D/A conversions with A/D conversions provided that the ADC is running using the internal A/D pacer clock. D/A conversions are synchronized with A/D channel conversions.

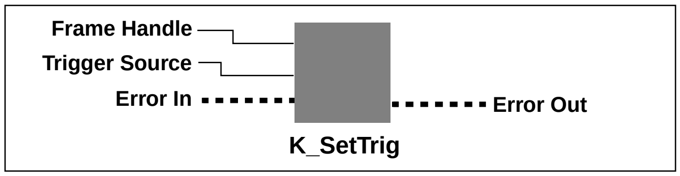
The sampling rate of a synchronized analog output operation is determined by the internal A/D pacer clock source; use **K_SetClkRate**, specifying an A/D frame, to set the sampling rate. Use **K_SetClk** to specify the clock source as external for a D/A frame**.**

Purpose

Specifies the trigger source.

Description

For the operation defined by *Frame Handle*, this VI specifies the trigger source in *Trigger Source*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Trigger Source*	Valid values: **0** for Internal trigger **1** for External trigger
	Error In	Error information.
	Error Out	Error information.

Remarks

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.

When performing a pre-trigger or about-trigger acquisition operation, mode, *Trigger Source* refers to the start trigger.

If *Trigger Source* = **1**, an external digital trigger (positive edge on TGIN) 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.

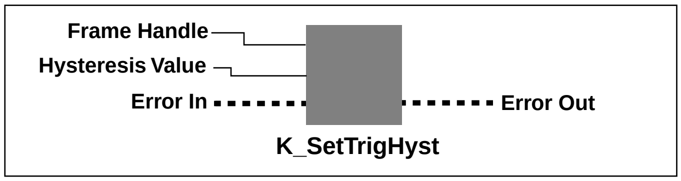
K_SetTrigHyst

Purpose

Specifies the hysteresis value.

Description

For the operation defined by *Frame Handle*, this VI specifies the hysteresis value used for an analog trigger in *Hysteresis Value*.



Parameters

U32	*Frame Handle*	Handle to the frame that defines the operation.
U16	*Hysteresis Value*	Hysteresis value, specified in raw counts. Valid values: **0** to **4,095**
	Error In	Error information.
	Error Out	Error information.

Remarks

The value you specify in *Hysteresis Value* sets the Trigger Hysteresis element in the frame identified by *Frame Handle*.

You must specify the hysteresis value in raw counts. Refer to Appendix B for information on converting the hysteresis voltage to a raw count.

K_SetTrigHyst does not affect the operation defined by *Frame Handle* unless the Trigger Source element is set to External (using **K_SetTrig**) before *Frame Handle* is used as an input to **K_IntStart** or **K_DMAStart**.

Refer to page 2-17 for more information about analog triggers.

A

Error Codes

Table A-1 lists the error codes that are returned by the DAS-1800 Series VI Driver, possible causes for error conditions, and possible solutions for resolving error conditions.

If you cannot resolve an error condition, contact the Keithley MetraByte Applications Engineering Department.

Table A-1. Error Codes

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
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 **K_OpenDriver** 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 configuration utility to create and modify configuration files.
6001	24577	**Illegal base address in configuration file:** The board's base I/O address in the configuration file is illegal and/or does not match the base address switches on the board.	Use the configuration utility to change the base I/O address to one that matches the base address switches on the board.
6002	24578	**Illegal IRQ level in configuration file:** The interrupt level in the configuration file is illegal.	Use the configuration utility to change the interrupt level to a legal one for your board. Refer to the user’s guide for legal interrupt levels.

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
6003	24579	**Illegal DMA channel in configuration file:** The DMA channel in the configuration file is illegal.	Use the configuration utility to change the DMA channel to a legal one for your board. Refer to the user's guide for legal DMA channels.
6005	24581	**Illegal channel number:** The specified channel number is illegal for the board and/or for the range type (unipolar or bipolar).	Specify a legal channel number. Refer to the user's guide or to **K_SetStartStopChn** in Chapter 4 for legal channel numbers.
6006	24582	**Illegal gain code:** The specified analog I/O channel gain code is illegal for this board.	Specify a legal gain code. Refer to the user's guide or to the description of **K_SetG** in Chapter 4 for a list of legal gain codes.
6007	24583	**Illegal DMA address:** A VI specified a buffer address that is not suitable for a DMA operation for the number of samples required.	Use **K_DMAAlloc** to allocate dynamic buffers for DMA operations. In Windows, make sure that the Keithley Memory Manager is installed; refer to Appendix D of the user's guide for information.
6008	24584	**Illegal number in configuration file:** The configuration file contains one or more numeric values that are illegal.	Use the configuration utility to check and then change the configuration file.
600A	24586	**Configuration file not found:** The driver cannot find the configuration file specified as an argument to **K_OpenDriver**.	Check that the file exists at the specified path. Check that the file name is spelled correctly in the **K_OpenDriver** 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 buffer handle passed to **K_DMAFree** was previously obtained using **K_DMAAlloc**.
600C	24588	**Error returning interrupt buffer:** The buffer handle specified in **K_IntFree** is invalid.	Check the buffer handle stored by **K_IntAlloc** and make sure that it was not modified.

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
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 requested buffer cannot be dynamically allocated because of its size.	Specify a smaller buffer size; refer to the description of **K_IntAlloc** in Chapter 4 for the legal range. If in Windows Enhanced mode with the Keithley Memory Manager (VDMAD.386) installed, use KMMSETUP.EXE to increase the reserved buffer heap size.
6010	24592	**Cannot allocate interrupt buffer: 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:** An error occurred when **K_IntFree** attempted to free a buffer handle.	Make sure that the buffer handle passed as an argument to **K_IntFree** was previously obtained using **K_IntAlloc**.
6015	24597	**DMA Buffer too large:** The number of samples specified in **K_DMAAlloc** is too large.	Refer to the description of **K_DMAAlloc** in Chapter 4 for the buffer size range.
6016	24598	**VDS - Region not contiguous:** An error occurred while using Windows Virtual DMA Services. You tried to use **K_DMAAlloc** in Windows Enhanced mode and the Keithley Memory Manager (VDMAD.386) was not installed	Refer to Appendix D in the user's guide for information on how to install and set up the Keithley Memory Manager (VDMAD.386).
6017	24599	**VDS - DMA wraparound:** See error 6016.	See error 6016.

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
6018	24600	**VDS - Unable to lock region:** See error 6016.	See error 6016.
6019	24601	**VDS - No buffer available:** See error 6016.	See error 6016.
601A	24602	**VDS - Region too large:** See error 6016.	See error 6016.
601B	24603	**VDS - Buffer in use:** See error 6016.	See error 6016.
601C	24604	**VDS - Illegal region:** See error 6016.	See error 6016.
601D	24605	**VDS - Region not locked:** See error 6016.	See error 6016.
601E	24606	**VDS - Illegal page:** See error 6016.	See error 6016.
601F	24607	**VDS - Illegal buffer:** See error 6016.	See error 6016.
6020	24608	**VDS - Copy out of range:** See error 6016.	See error 6016.
6021	24609	**VDS - Illegal DMA channel:** See error 6016.	See error 6016.
6022	24610	**VDS - Count overflow:** See error 6016.	See error 6016.
6023	24611	**VDS - Count underflow:** See error 6016.	See error 6016.
6024	24612	**VDS - Function not supported:** See error 6016.	See error 6016.
6025	24613	**Illegal OBM mode:** The mode number specified in **K_SetOBMMode** is illegal.	If applicable to your board, refer to the description of **K_SetOBMMode** in Chapter 4 for legal mode values.

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
6026	24614	**Illegal DMA structure:** An error occurred during the execution of **K_DMAFree**.	Try using **K_DMAFree** again. If the error continues, contact the Keithley MetraByte Applications Engineering Department.
6027	24615	**DMA allocation error:** See error 6026.	See error 6026.
6028	24616	**NULL DMA handle:** See error 6026.	See error 6026.
6029	24617	**DMA unlock error:** See error 6026.	See error 6026.
602A	24618	**DMA free error:** See error 6026.	See error 6026.
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 exceeds maximum:** The number of samples you requested from the Keithley Memory Manager is greater than the allowed maximum.	Specify a value within the legal range when calling **K_DMAAlloc** in Windows Enhanced mode. Refer to the description of **K_DMAAlloc** in Chapter 4 for legal values.
602D	24621	**Illegal device handle:** A bad board handle was passed to a VI such as **K_GetADFrame**. The handle used was not initialized through a call to **K_GetDevHandle**, or it was corrupted by your program.	Check the board handle value.
602E	24622	**Illegal Setup option:** An illegal option was specified to a VI that accepts a user option, such as **K_SetDITrig**.	Check the option value passed to the VI where the error occurred.

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
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, use the KMMSETUP utility to expand the reserved heap.
6031	24625	**Illegal memory handle:** A bad buffer 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 it was corrupted by you program.	Restart your program and monitor the buffer handle values.
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 has already been assigned.	Use **K_IntFree** or **K_DMAFree** to free previously allocated memory blocks before allocating again.
6034	24628	**Memory corrupted:** Int 21H function 48H, used to allocate a memory block from the DOS far heap, returned the DOS error 7; this means that memory is corrupted. It is likely that you stored data (through a DMA-mode or interrupt-mode operation) into an illegal area of DOS memory.	Recheck the parameters set by **K_DMAAlloc** and **K_SetDMABuf**. If a 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.)	Make sure that you configure the driver for a particular device only once during a single Windows session. If a driver has already been configured, pass a null string as the second argument to **K_OpenDriver**.

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
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 board name string is entered correctly in **K_OpenDriver**.
6038	24632	**Invalid source pointer:** The pointer to the source buffer that you passed as an argument to **K_MoveBufToArray** is invalid for the specified count. (The source pointer, when added to the number of samples, exceeds the programmed addressing range of that pointer.)	Check the pointer to the source buffer and the number of samples to transfer that you specified in **K_MoveBufToArray**.
6039	24633	**Invalid destination pointer:** The pointer to the destination buffer (local array) that you passed to **K_MoveBufToArray** is invalid for the specified count. (The destination pointer, when added to the number of samples, exceeds the dimension of the local array.)	Check the dimension of the local array and the number of samples to transfer that you specified in **K_MoveBufToArray**.
603A	24634	**Illegal setup value:** An illegal value was passed to the VI in which the error occurred.	Check the legal ranges of all parameters passed to this VI.
603B	24635	**Error freeing buffer selector:** **K_DMAFree** or **K_IntFree** failed because one or more of the selectors that reference the memory buffer could not be freed.	Check that the memory buffer being freed was previously obtained through **K_DMAAlloc** or **K_IntAlloc.**

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
603C	24636	**Error allocating buffer selector: K_DMAAlloc** or **K_IntAlloc** failed because a selector could not be allocated from Window's Local Descriptor Table.	Close all applications and restart Windows. If the error continues, contact the Keithley MetraByte Applications Engineering Department.
603D	24637	**Error allocating memory buffer: K_DMAAlloc** or **K_IntAlloc** failed because a necessary internal buffer could not be allocated to complete the operation.	Close all applications and restart Windows. If the error continues, contact the Keithley MetraByte Applications Engineering Department.
7000	28672	**No board name**: **K_OpenDriver** did not find a board name in the specified configuration file.	Specify a legal board name in the configuration file.
7001	28673	**Illegal board name**: The board name in the specified configuration file is illegal.	Specify a legal board name in the configuration file.
7002	28674	**Illegal board numbe**r: **K_OpenDriver** found an illegal board number in the specified configuration file.	Specify a legal board number: 0, 1, or 2
7003	28675	**Illegal base address**: **K_OpenDriver** 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	**Illegal DMA channel**: **K_OpenDriver** 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	**Illegal interrupt level**: **K_OpenDriver** found an illegal interrupt level in the specified configuration file.	Specify a legal interrupt level: 3, 5, 7, 10, 11, or 15

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
7007	28679	**Illegal A/D channel mode**: **K_OpenDriver** found an illegal input range type in the specified configuration file.	Specify a legal input range type: bipolar, unipolar
7008	28680	**Illegal A/D channel configuration**: **K_OpenDriver** found an illegal input configuration in the specified configuration file.	Specify a legal input configuration: single-ended, differential
700A	28682	**Illegal number of SSH boards**: The number of SSH-8s in the configuration file is not valid.	Use the configuration utility to specify the number of SSH-8s as a number in the range 0 to 8.
700B	28683	**Illegal SSH8 channel**: The SSH-8 channel in the configuration file is not valid.	Use the configuration utility to specify the SSH-8 channel as a number in the range 0 to 7.
700C	28684	**Illegal SSH8 gain**: The SSH-8 channel gain in the configuration file is not valid.	Use the configuration utility to specify the SSH-8 channel gain as 0.5, 5, 50, or 250.
700D	28685	**DAS Spec rev number is bad**: A board-specific component is incompatible with the DAS shell version.	Re-install the DAS driver software from the original disks for this board.
700E	28686	**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	**Illegal analog trigger, A/D busy:** An analog input operation was in progress when the application attempted to start an analog output operation for which an analog trigger was defined.	Wait until the analog input operation is done or use **K_IntStop** or **K_DMAStop** to stop the analog input operation before initiating the analog output operation.
7010	28688	**Illegal retrigger mode:** The number of output values for a re-triggered analog output operation exceeds 2048.	Specify the number of output values as 2048 or less.

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
7011	28689	**D/A FIFO Underflow:** The pacer clock rate specified for an analog output operation is too fast.	Specify a slower pacer clock rate.
7012	28690	**Illegal burst mode conversion clock 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	**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	**Counter 0 resource 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	**Illegal number of about-trigger samples**: 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	**Illegal about-trigger mode**: 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 boards:** The number of EXP-1800 expansion accessories specified in the configuration file is not valid.	Run CFG1800.EXE and specify the number of EXP-1800s as a number in the range 0 to 16.
8001	32769	**Function not supported:** You have attempted to use a VI not supported by the VI Driver.	Contact the Keithley MetraByte Applications Engineering Department.

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
8003	32771	**Illegal board number:** An illegal board number was specified in **K_OpenDriver**.	Refer to the description of **K_GetDevHandle** in Chapter 4 for legal board numbers.
8004	32772	**Illegal error number:** The error message number specified in **K_GetErrMsg** is invalid.	The error number must be one the error numbers listed in this appendix.
8005	32773	**Board not found at configured address: K_OpenDriver** 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.
8006	32774	**A/D not initialized:** You attempted to start a frame-based analog input operation without the A/D frame being properly initialized.	Always call **K_ClearFrame** before setting up a new frame-based operation.
8007	32775	**D/A not initialized:** You attempted to start a frame-based analog output operation without the D/A frame being properly initialized.	Always call **K_ClearFrame** before setting up a new frame-based operation.
8008	32776	**Digital input not initialized:** You attempted to start a frame-based digital input operation without the DI frame being properly initialized.	Always call **K_ClearFrame** before setting up a new frame-based operation.
8009	32777	**Digital output not initialized:** You attempted to start a frame-based digital output operation without the DO frame being properly initialized.	Always call **K_ClearFrame** before setting up a new frame-based operation.
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.

Table A-1. Error Codes (cont.)

Error Code			
Hex	**Decimal**	**Cause**	**Solution**
8016	32790	**Interrupt overrun**: The board communicated a hardware event to the software by generating a hardware interrupt, but the software was still servicing a previous interrupt. This is usually caused by a pacer clock rate that is too fast.	Check the maximum throughput rate for your computer's programming environment and use **K_SetClkRate** to specify an appropriate rate.
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.
801B	32795	**DMA already active**: You attempted to start a DMA-mode operation using a DMA channel that is currently used by another active operation.	Use **K_DMAStop** to stop the first operation before starting the second operation.
8020	32800	**FIFO Overflow event detected:** During data acquisition, the temporary on-board data storage (FIFO) overflowed.	The conversion rate is too fast for your computer's programming environment; use **K_SetClkRate** to reduce the conversion rate. If you are using DMA-mode and your board supports dual-DMA, use the configuration utility to reconfigure your board to use dual-DMA.
8021	32801	**Illegal clock sync mode**: The two operations you are trying to synchronize cannot be synchronized on your board.	Check the synchronizing clock source that you specified in **K_SetSync**.
FFFF	65535	**User aborted operation**	You pressed **`[Ctrl]+[Break]`** while waiting for an analog trigger event to occur.

B

Converting Data Formats

The DAS-1800 Series VI 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:

$$\text{Voltage} = \frac{\text{count} \times \text{span}}{4096}$$

Table B-1. Span Values For Analog Input Data Conversion Equations

Board	Input Range Type	Gain	Input Range	Span (V)
DAS-1801AO	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-1802AO	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-1801AO 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:

$$\frac{3072 \times 5\ \text{V}}{4096} = 3.75\ \text{V}$$

As another example, assume that you want to read analog input data from a channel on a DAS-1802AO 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:

$$\frac{1024 \times 10\ \mathrm{V}}{4096} = 2.5\ \mathrm{V}$$

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

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, and *span* is 10 V for the ±5 V range and 20 V for the ±10 V range:

$$\mathrm{Count} = \frac{\mathrm{voltage} \times 4096}{\mathrm{span}}$$

For example, assume that you want to specify an analog output value of 5 V for a channel whose output range is ±10 V. The raw count is determined as follows:

$$\frac{5\ \mathrm{V} \times 4096}{20\ \mathrm{V}} = 1024$$

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 10 V for the ±5 V range and 20 V for the ±10 V range:

$$\text{Count} = \frac{V_{trig} \times 4096}{\text{span}}$$

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, 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-1801AO 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 10 V for the ±5 V range and 20 V for the ±10 V range:

$$\text{Count} = \frac{V_{\text{hyst}} \times 4096}{\text{span}}$$

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-1801AO 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

A

B

C

D

E

F