ASO-TC
User's Guide

Revision A
Printed June, 1993
Part No. 24469

WARNING

Keithley Data Acquisition assumes no liability for damages consequent to the use of this Product. This Product is not designed with components of a level of reliability that is suitable for use in life support or critical applications.

The information contained in this manual is believed to be accurate and reliable. However, Keithley Data Acquisition assumes no responsibility for its use; nor for any infringements or patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent rights of Keithley Data Acquisition.

Keithley Data Acquisition does not warrant that the Product will meet the Customer's requirements or will operate in the combinations which may be selected for use by the Customer or that the operation of the Program will be uninterrupted or error free or that all Program defects will be corrected.

Keithley Data Acquisition does not and cannot warrant the performance or results that may be obtained by using the Program. Accordingly, the Program and its documentation are sold "as is" without warranty as to their performance merchantability, or fitness for any particular purpose. The entire risk as to the results and performance of the program is assumed by you.

Keithley Data Acquisition • 440 Myles Standish Blvd. • Taunton, MA 02780
Telephone: (508) 880-3000 • Fax: (508) 880-0179

Contents

Chapter 1 Introduction 1

1.1 About the ASO-TC 1

1.2 Prerequisites 2

1.3 Getting help 2

1.4 Installing the ASO for DOS 2

1.5 Installing the ASO for Windows 4

Chapter 2 The Function Call Driver 7

2.1 Available operations 7

2.2 Overview of programming with the Function Call Driver 12

2.3 Board/Driver initialization tasks 13

2.4 Operation-specific programming tasks 14

2.5 Language-specific programming notes 21

Chapter 3 Functions 43

3.1 Functional grouping 43

3.2 Function reference 47

Appendix A Function Call Driver error messages 85

A.1 Error Codes 85

A.2 Error Conditions 99

Introduction 1

1.1 About the ASO-TC

The ASO-TC is the Advanced Software Option (ASO) for the DAS-TC analog input board. The ASO includes a set of software components that you can use, in conjunction with a programming language, to create application programs that execute the operations available on the DAS-TC.

The primary component of the ASO is the Function Call Driver. This driver provides your application program with high-level access to the acquisition and control operations available on the DAS-TC. The ASO also includes support files, example programs, a configuration utility, and a data logging utility. For information on the configuration and data logging utilities, refer to the *DAS-TC User's Guide*.

The Function Call Driver enables your program to define and execute board operations by using calls to driver-provided functions. For example, your program can call the driver-provided **K_ADRead** function to execute a single-point, A/D input operation.

The ASO includes several different versions of the Function Call Driver. The .LIB and .TPU versions are provided for DOS application development in 'C' and Pascal languages. The Dynamic Link Library (DLL) is provided for Windows application development.

The ASO and this manual provide the necessary tools, example programs and information to develop Function Call Driver programs in the following languages:

- Borland C/C++ (version 2.0 and higher)

- Borland Turbo Pascal (version 6.0)
- Borland Turbo Pascal for Windows (version 1.0)
- Microsoft C (version 5.1 and above)
- Microsoft C++ (version 7.0)
- Microsoft Quick C for Windows (version 1.0)
- Microsoft Visual Basic for Windows (version 1.0 and higher)

Note If you are using a version of Turbo Pascal higher than version 6.0, see section 2.5 for the procedure required to make a Turbo Pascal unit compatible with your version.

1.2 Prerequisites

The ASO is designed exclusively for use with the DAS-TC. This manual assumes that you understand the information presented in the *DAS-TC User's Guide*. Additionally, you must complete the board installation and configuration procedures outlined in the *DAS-TC User's Guide* before you attempt any of the procedures described in this manual.

The fundamental goal of this manual is to provide you with the information you need to write DAS-TC application programs that use the ASO driver. It is recommended that you proceed through this manual according to the sequence suggested by the table of contents; this will minimize the amount of time and effort required to develop your ASO application programs for the DAS-TC.

1.3 Getting help

The following resources provide information about using the ASO:

- this manual
- the *DAS-TC User's Guide*
- the ASO example programs (these are copied to your system's hard disk during the installation procedure)
- the documentation for the programming language you are using

Call our Applications Engineering Department if you need additional assistance. An applications engineer will help you diagnose and solve your problem over the telephone.

Keithley Data Acquisition
Applications Engineering
508–880–3000
Monday – Friday, 8 A.M. – 7 P.M.

For the most efficient and helpful assistance, please compile the following information before calling our Applications Engineering Department:

ASO package	Version	_____
	Invoice/Order #	_____
DAS-TC	Serial #	_____
	Base address setting	_____
Computer	Manufacturer	_____
	CPU type	8088 286 386 486 Other
	Clock speed (MHz)	8 12 20 25 33 Other
	Math co-processor?	Yes No
	Amount of RAM	_____
	Video system	CGA Hercules EGA VGA
Compiler	Language	_____
	Manufacturer	_____
	Version	_____

1.4 Installing the ASO for DOS

To code ASO applications programs in a DOS-based language, load the software using the ASO-DOS distribution diskettes.

The files on the ASO-DOS distribution diskettes are in compressed format. You must use the installation program included on the diskettes to install the ASO software. Since the aggregate size of the expanded ASO files is approximately 1.5 MB, check that there is at least this much space available on your PC's hard disk before you attempt to install the ASO.

Perform the following procedure to install the ASO software (note that it is assumed that the floppy drive is designated as drive **A**):

1. Make a back-up copy of the distribution diskette(s).
2. Insert ASO-DOS diskette #1 into the floppy drive
3. Type the following commands at the DOS prompt:

   A: [**Enter**]
   install [**Enter**]

The installation program prompts you for your installation preferences, including the name of the subdirectory into which the ASO-DOS files are copied. The installation program expands the files on the ASO diskette(s) and copies them into the ASO-TC subdirectory you specified; refer to the file FILES.DOC in your ASO-TC subdirectory for the names and descriptions of these files.

1.5 Installing the ASO for Windows

To code ASO applications programs in a Windows-based language, load the software using the ASO-Windows distribution diskettes.

The files on the ASO-Windows diskette are in compressed format. You must use the setup program included on the diskette to install the software. Since the aggregate size of the expanded files is approximately 2 MB, check that there is at least this much space available on your PC's hard disk before you attempt to install the files.

Perform the following procedure to install the Windows-based software (assume that the floppy drive is designated as drive A):

1. Make a back-up copy of the ASO-Windows diskette.
2. Start Windows.
3. Insert the ASO-Windows diskette into the floppy drive.
4. From the Program Manager menu, choose **File** then **Run...**.
5. At the Command Line type `A:\SETUP.EXE`

The setup program prompts you for your installation preferences, including the name of the subdirectory into which the ASO-Windows files are copied. If you press **Continue** after you type in the pathname, the setup program expands the files and copies them into the ASO-TC subdirectory you specified; refer to the file FILES.DOC in your ASO-TC subdirectory for the names and descriptions of these files.

The installation process also creates a DAS-TC icon. This icon includes a C example program, the WDASTCCF.EXE configuration utility, the datalogger utility, and FILES.DOC. The configuration utility and the datalogger are described in the *DAS-TC User Guide*.

2 The Function Call Driver

2.1 Available operations

The ASO-TC provides you with two types of analog-to-digital (A/D) input operations:

- Single-call
- Frame-based

The following subsections describe these operations in more detail.

Both types of operations are implemented with functions, to which you pass parameters. As with any function, you declare the corresponding arguments before making the call.

Single-call A/D Input operations

In a *single-call* A/D input operation, you read an analog input value using a single call to a function. Analog-to-Digital conversion is performed automatically.

You specify the attributes of the operation, such as the board that executes the operation, the channel from which to read data, and the buffer in which to store the data, as arguments to the function. The data is returned as a single voltage or temperature value in engineering units.

Note The Function Call Driver reads the configuration file to determine the gain; therefore, the gain parameter is ignored.

Use the **K_ADRead** function to read a single analog input value from a specified analog input channel.

The **DASTC_GETCJC** function is a special-purpose single-call function for reading the value of the CJC (Cold Junction Compensation) channel. You can use the resulting value to correct a temperature reading in cases where you want to perform your own linearization.

If you wish, you can use **K_ADRead** or **DASTC_GETCJC** with software looping to acquire more than one value from one or more channels. Typically, when you are acquiring more than one value you may want to exercise more control over the data transfer than is possible with single-call operations. In such cases, use a frame-based operation, described next.

Frame-based A/D input operations

A *frame-based* input operation is normally used to sample more than one value from one or more channels. In the case of the DAS-TC, the data returned consists of as many voltage or temperature values as there are analog input samples. The values are returned in engineering units.

A frame-based operation uses a single data structure called a *frame* to represent the controllable attributes of the operation for a particular board. You request a frame by calling the function, **K_ADFrame**.

A frame-based operation is realized as a sequence of function calls. At a minimum, a frame-based sequence includes functions that manage and set frame elements, followed by a function that performs the actual transfer of values.

The controllable attributes of the operation, such as the start channel, stop channel, and number of samples, are known as frame *elements*. The following table lists the frame elements available for the ASO-TC and the corresponding function used to set each element. Refer to the appropriate function description in Section 3.2 for the valid settings of a frame's elements.

Element	Function	Page
Start/Stop Channel	K_SetStartStopChn	82
Channel-Gain Array Address	K_FormatChnGAry	57
	K_RestoreChnGAry	75
	K_SetChnGAry	79
Number of Samples	K_SetBuf	76
	K_SetBufL	77
	K_SetBufR	78
	K_IntAlloc	69
Data Buffer Address	K_SetBuf	76
	K_SetBufL	77
	K_SetBufR	78
Buffering Mode	K_SetContRun	81
	K_ClrContRun	55

One frame corresponds to one set of element values. Once you set the frame's elements, you can pass all of the settings to the function that starts the A/D operation, using only the *frame handle*, which identifies the frame (and the board from which you called **K_GetADFrame**).

If several operations acquiring data from a particular board use the same element settings, they can pass the same frame handle. Afterwards, you should release the frame by calling **K_FreeFrame**. The Function Call Driver allows you to request up to eight frames, regardless of which board you are using when you call **K_GetADFrame**. For example, you could use five frames for board 1 and three frames for board 2. Similarly, you could use eight frames for board 1; however, no frames would be available for board 2 in this example.

Note Each of the programming languages is supported by a file that contains a definition of the FRAMEH variable type. Therefore, you must declare all frame handles to be of this type.

Operation Modes

For the DAS-TC, frame-based A/D operations are available in two modes:

- Synchronous
- Interrupt

In *Synchronous* mode, the frame-based sequence passes control to the Function Call Driver, which acquires and converts data in the foreground. After the specified number of samples is acquired, the driver returns control to the application program. This operation mode is easier to program than interrupt mode operations. It should not be used if some procedure requires a block of data before executing and/or needs to monitor or control the transfer. Use the **K_SyncStart** function to start a frame-based operation in synchronous mode.

Interrupt mode allows the board to acquire and convert data in the background while the application program retains control. The DAS-TC interrupts the application when an acquired block of samples is ready to be transferred to a user-defined buffer. The Function Call Driver's interrupt handler gets control just long enough to complete a block transfer; this period is sufficiently brief as to be imperceptible. Interrupt mode is useful when monitoring and control over the transfer is desired, concurrent processing (without loss of data integrity) is desired, or when blocks of acquired data must be partially processed before the requested transfer is completed. Use the **K_IntStart** function to start a frame-based operation in interrupt mode.

Note On the DAS-TC, data is transferred in blocks, where block size = the number of channels specified. Suppose, for example, you have requested 43 samples using ten channels. The Function Call Driver actually acquires 50 values in five blocks of ten samples each. The first 40 values are transferred from the first four blocks that have been acquired, and the remaining three samples are transferred from the fifth acquired block of ten samples.

Input Buffers

The Function Call Driver stores acquired samples in a buffer that you define with one of two methods:

- Locally defined (user-defined)
- Dynamically allocated

Once you have defined a buffer by one of the two methods, use a **K_SetBuf** call to pass the buffer address to the Function Call Driver.

You must define a local buffer as an array before you call **K_SetBuf**. You can also use a local buffer for more permanent storage by using **K_MoveDataBuf** to move acquired data into your local buffer.

Use **K_IntAlloc** to dynamically allocate memory outside of your program area for later release with **K_IntFree**. If you are running in Windows standard mode and transferring data using interrupts, you must use a dynamically allocated buffer to receive the acquired data, since your program's memory pointers may shift.

You can use a combination of local and dynamically allocated buffers for storing blocks of acquired samples. The function, **K_MoveDataBuf**, provides a convenient method, particularly in Visual Basic, for moving acquired data from a dynamically allocated buffer into a local buffer.

Buffering Mode

You can specify either SINGLE-CYCLE or CONTINUOUS buffering mode for interrupt operations. In Single-Cycle mode, the specified number of samples is stored in the buffer and the operation stops automatically. Use the **K_ClrContRun** function to specify Single-Cycle buffering mode.

In Continuous mode, the board keeps acquiring the same number of new values, placing the data in the buffer until it receives the stop function, **K_IntStop**. The transfer index and buffer pointers are reset before another transfer cycle is initiated, and acquired values in the buffer are overwritten. Use the **K_SetContRun** function to specify Continuous buffering mode. If you do not specify Continous buffering mode, the DAS-TC defaults to Single-Cycle mode.

Note If you are acquiring data using interrupts and Continuous buffering, as soon as the last block of samples is transferred,

- the transfer count and buffer pointer are reset to zero,
- **K_IntStatus** returns zero instead of the requested sample size in the *index* parameter, and
- the driver begins to overwrite your buffer's data.

If your application requires consecutive blocks of data, you should begin processing your buffer *before* your buffer is full, using **K_IntStatus** to determine how many blocks have been transferred (this function's *index* parameter increments by the block size).

2.2 Overview of programming with the Function Call Driver

The procedure to write a Function Call Driver program is as follows:

1. Define the application's requirements.
2. Write the program code.
3. Compile and link the program.

The subsections that follow describe the details of each of these steps.

Defining the application's requirements

Before you begin writing the program code, you should have a clear idea of the operations you expect your program to execute. Additionally, you should determine the order in which these operations must be executed and the characteristics (number of samples, start and stop channels, and so on) that define each operation. You may find it helpful to review the list of available operations in Section 2.1 and to browse through the short descriptions of the Functions in Section 3.1.

Writing the program code

Several sources of information relate to this step:

- Section 2.3 explains the initial programming tasks that all Function Call Driver programs must execute
- Section 2.4 describes typical frame-based sequences of function calls
- Section 3.2 provides detailed information on individual functions
- The ASO includes several example source code files for Function Call Driver programs. The FILES.DOC file in the ASO-DOS installation directory lists and describes the example programs. The FILES.DOC in the ASO-Windows installation directory lists and discribes the example programs that run in Windows only.

Compiling and linking the program

Refer to Section 2.5 for compile and link instructions and other language-specific considerations for each supported language.

2.3 Board/Driver initialization tasks

Every Function Call Driver program must execute the following programming tasks:

1. Identify a function/variable type definition file
   The method to identify this file is language-specific; refer to Section 2.5 for additional information.
2. Declare/initialize program variables
3. Call **DASTC_DevOpen** to initialize the driver
4. Call **DASTC_GetDevHandle** to initialize the board and get a device handle for the board.

The tasks listed are the minimum tasks your program must complete before it attempts to execute any operation-specific tasks. Your application may require additional board/driver initialization tasks. For example, if your program requires access to two boards, then it must call **DASTC_GetDevHandle** for each board.

Note A *device handle* is a variable whose value identifies an installed board. The purpose of a device handle is to provide a mechanism through which the Function Call Driver can access a board. A device handle is also a convenient method for different function calls to reference the same board. Each board must have a unique device handle.

Each of the programming languages is supported by a file that contains a definition of the DDH (for *DAS Device Handle*) variable type; you should declare all device handles to be of this type.

2.4 Operation-specific programming tasks

After you perform the board/driver initialization tasks, perform the appropriate operation-specific tasks, as follows:

- For Single-Call A/D Operations - The only operation-specific task required is using the appropriate single-call A/D function (**K_ADRead** or **DASTC_GETCJC**).
- For Frame-Based A/D Operations - The operation-specific tasks required for frame-based A/D operations depend on whether you are using synchronous or interrupt mode, whether you are using Start and Stop channels or Channel-Gain arrays, and whether you are using local buffers, dynamically allocated buffers, or both. For the page number that corresponds to the operation you want to perform, see the table shown below.

Operation mode	Method of specifying acquisition channels	Buffer type	See page...
Synchronous	Start/Stop channels	Local	15
Synchronous	Start/Stop channels	Dynamic	15
Synchronous	Channel-Gain array	Local	16
Synchronous	Channel-Gain array	Dynamic	16
Interrupt	Start/Stop channels	Local [1]	17
Interrupt	Start/Stop channels	Dynamic	17
Interrupt	Start/Stop channels	Both	18
Interrupt	Channel-Gain array	Local [1]	18
Interrupt	Channel-Gain array	Dynamic	19
Interrupt	Channel-Gain array	Both	20

[1] Do not use this sequence if you are running in Windows standard mode.

Note If you do not use the functions that set a frame's elements, the Function Call Driver defaults to the values that resulted from frame initialization.

You must pass the address of the buffer that is receiving the data, by calling **K_SetBuf**, **K_SetBufL**, or **K_SetBufR**. The choice of **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** depends on the programming language and buffer type. See Section 3.2 for more information on these functions. No error message occurs if this function is not included; however, the frame element, *BufAddr*, has a default value of zero, and no samples are returned.

Synchronous, Start/Stop channels, local buffer only

Use this calling sequence to perform a synchronous transfer, using Start/stop channels and a local buffer only. Before calling the functions in the sequence, define a local buffer as an array of four-byte elements.

1. Call **K_GetADFrame** to get the handle to an A/D frame.
2. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the buffer address previously obtained to the Buffer Address element in the frame.
3. Call **K_SetStartStopChn** to assign values to the Start and Stop Channel elements in the frame.
4. Call **K_SyncStart** to start the operation. Data is stored in the local buffer.
5. Call **K_FreeFrame** to return the frame to the pool of available frames obtained, unless you are starting another sequence that uses the same frame.

Synchronous Start/Stop channels, dynamically allocated buffer only

Use this calling sequence to perform a synchronous transfer using Start/Stop channels and a dynamically allocated buffer only.

1. Call **K_GetADFrame** to get the handle to an A/D frame.
2. Call **K_IntAlloc** to allocate the buffer into which the driver stores the A/D values outside of the program's memory area.
3. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the buffer address previously obtained to the Buffer Address element in the frame.
4. Call **K_SetStartStopChn** to assign values to the Start and Stop Channel elements in the frame.
5. Call **K_SyncStart** to start the operation. Data is accessed via the pointer returned by **K_IntAlloc**.
6. Call **K_IntFree** to deallocate the buffer.
7. Call **K_FreeFrame** to return the frame to the pool of available frames obtained, unless you are starting another sequence that uses the same frame.

Synchronous, Channel-Gain array, local buffer only

Use this calling sequence to perform a synchronous transfer using a Channel-Gain array and a local buffer only. Before calling the functions in the sequence, define a local buffer as an array of four-byte elements.

1. Call **K_GetADFrame** to get the handle to an A/D frame.
2. Define and assign values to a Channel-Gain array.
3. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the buffer address previously declared to the Buffer Address element in the frame.
4. Call **K_SetChnGAry** to assign the Channel-Gain array to the Channel-Gain Array Address element in the frame.
5. Call **K_SyncStart** to start the operation. Data is stored in the local buffer.
6. Call **K_FreeFrame** to return the frame to the pool of available frames.

Synchronous, Channel-Gain array, dynamically allocated buffer only

Use this calling sequence to perform a synchronous transfer using a Channel-Gain array and a local buffer only.

1. Call **K_GetADFrame** to get the handle to an A/D frame.
2. Define and assign values to a Channel-Gain array.
3. Call **K_IntAlloc** to allocate the buffer into which the driver stores the A/D values outside of the program's memory area.
4. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the address of the buffer previously declared to the Buffer Address element in the frame.
5. Call **K_SetChnGAry** to assign the Channel-Gain array to the Channel-Gain Array Address element in the frame.
6. Call **K_SyncStart** to start the operation. Data is accessed via the pointer returned by **K_IntAlloc**.
7. Call **K_IntFree** to deallocate the buffer.
8. Call **K_FreeFrame** to return the frame to the pool of available frames, unless you are starting another sequence that uses the same frame.

Interrupt, Start/Stop channels, local buffer only

Use this calling sequence to perform an interrupt transfer using Start/Stop channels and a local buffer only. Before calling the functions in the sequence, define a local buffer as an array of four-byte elements.

1. Call **K_GetADFrame** to get the handle to an A/D frame.
2. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the buffer address previously declared to the Buffer Address element in the frame.
3. Call **K_SetStartStopChn** to assign values to the Start and Stop Channel elements in the frame associated with the frame handle previously obtained.
4. Call **K_IntStart** to start the operation.
5. Call **K_IntStatus** to monitor the status of the operation. When completion is detected, the data is available in the local buffer.
6. Call **K_FreeFrame** to return the frame to the pool of available frames, unless you are starting another sequence that uses the same frame.

Interrupt, Start/Stop channels, dynamically allocated buffer only

Use this calling sequence to perform an interrupt transfer using Start/Stop channels and a dynamically allocated buffer only.

1. Call **K_GetADFrame** to get the handle to an A/D frame.
2. Call **K_IntAlloc** to allocate a buffer into which the driver stores the A/D values outside of the program's memory area.
3. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the buffer address previously declared to the Buffer Address element in the frame.
4. Call **K_SetStartStopChn** to assign values to the Start and Stop Channel elements in the frame associated with the frame handle previously obtained.
5. Call **K_IntStart** to start the operation.
6. Call **K_IntStatus** to monitor the status of the operation. When completion is detected, the data is accessed via the pointer returned by **K_IntAlloc**.
7. Call **K_IntFree** to deallocate the buffer.

8. Call **K_FreeFrame** to return the frame to the pool of available frames, unless you are starting another sequence that uses the same frame.

Interrupt, Start/Stop channels, dynamically allocated and local buffers

Use this calling sequence to perform an interrupt transfer using Start/Stop channels and both buffers. Before calling the functions in the sequence, define a local buffer as an array of four-byte elements.

1. Call **K_GetADFrame** to get the handle to an A/D frame.
2. Call **K_IntAlloc** to allocate a buffer into which the driver stores the A/D values outside of the program's memory area.
3. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the buffer address previously declared to the Buffer Address element in the frame.
4. Call **K_SetStartStopChn** to assign values to the Start and Stop Channel elements in the frame associated with the frame handle previously obtained.
5. Call **K_IntStart** to start the operation.
6. Call **K_IntStatus** to monitor the status of the operation. When completion is detected, the data is accessed via the pointer returned by **K_IntAlloc**.
7. Call **K_MoveDataBuf** to transfer the acquired data from a buffer allocated by **K_IntAlloc** to the user-defined array.
8. Call **K_IntFree** to deallocate the buffer.
9. Call **K_FreeFrame** to return the frame to the pool of available frames, unless you are starting another sequence that uses the same frame.

Interrupt, Channel-Gain array, local buffer only

Use this calling sequence to perform an interrupt transfer using a Channel-Gain array and a local buffer only. Before calling the functions in the sequence, define a local buffer as an array of four-byte elements.

1. Call **K_GetADFrame** to get the handle to an A/D frame, unless you are starting another sequence that uses the same frame.
2. Define and assign values to a Channel-Gain array.

3. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the address of the buffer previously declared to the Buffer Address element in the frame.

4. Call **K_SetChnGAry** to assign the Channel-Gain array previously obtained to the Channel-Gain Array Address element in the frame.

5. Call **K_IntStart** to start the operation.

6. Call **K_IntStatus** to monitor the status of the operation. When completion is detected, data is available in the local buffer.

7. Call **K_FreeFrame** to return the frame to the pool of available frames, unless you are starting a another sequence that uses the same frame.

Interrupt, Channel-Gain array, dynamically allocated buffer only

Use this calling sequence to perform an interrupt transfer using a Channel-Gain array and a dynamically allocated buffer only.

1. Call **K_GetADFrame** to get the handle to an A/D frame.

2. Define and assign values to a Channel-Gain array.

3. Call **K_IntAlloc** to allocate the buffer into which the driver stores the A/D values outside of the program's memory area.

4. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the address of the buffer previously declared to the Buffer Address element in the frame.

5. Call **K_SetChnGAry** to assign the Channel-Gain array previously obtained to the Channel-Gain Array Address element in the frame.

6. Call **K_IntStart** to start the operation.

7. Call **K_IntStatus** to monitor the status of the operation. When completion is detected, the data is accessed via the pointer returned by **K_IntAlloc**.

8. Call **K_IntFree** to deallocate the buffer.

9. Call **K_FreeFrame** to return the frame to the pool of available frames, unless you are starting a sequence that uses the same frame.

Interrupt, Channel-Gain array, dynamically allocated and local buffers

Use this calling sequence to perform an interrupt transfer using a channel-Gain array and both a local and a dynamically allocated buffer. Before calling the functions in the sequence, define a local buffer as an array.

1. Call **K_GetADFrame** to get the handle to an A/D frame.
2. Define and assign values to a Channel-Gain array.
3. Call **K_IntAlloc** to allocate a buffer into which the driver stores the A/D values outside of the program's memory area.
4. Call **K_SetBuf**, **K_SetBufL**, or **K_SetBufR** to assign the buffer address previously declared to the Buffer Address element in the frame.
5. Call **K_SetChnGAry** to assign the channel-gain array previously obtained to the Channel-Gain Array Address element in the frame.
6. Call **K_IntStart** to start the operation.
7. Call **K_IntStatus** to monitor the status of the operation. When completion is detected, the data is accessed via the pointer returned by **K_IntAlloc**.
8. Call **K_MoveDataBuf** to transfer data from a buffer you have allocated by **K_IntAlloc** to the array.
9. Call **K_IntFree** to deallocate the buffer.
10. Call **K_FreeFrame** to return the frame to the pool of available frames.

2.5 Language-specific programming notes

This section provides specific programming guidelines for each of the supported languages. Additional programming information is available in the ASO example programs. Refer to the FILES.DOC file for names and descriptions of the ASO example programs.

Note The example programs in this section are not actual programs but are fragments that are designed to illustrate an interrupt-mode A/D input sequence that uses a Channel-Gain array.

Borland C/C++ and Microsoft C/C++

Related Files

DASTC.LIB
DASRFACE.LIB
USERPROT.H
USERPROT.BCP

Compile and Link Instructions

Borland C:

```
BCC -ml filename.c dastc.lib dasrface.lib
```

Borland C++
If you want to compile your program as a Borland C++ program,

1. Use the supplied file USERPROT.BCP instead of USERPROT.H.
2. Specify the C++ compilation in one of the following two ways:
   a. Specify .CPP as the extension for your source file, or
   b. Use the BCC **-P** command line switch.

Microsoft C/C++:

```
CL /AL /c filename.c
LINK filename,,,DASTC+DASRFACE;
```

Code example

This example executes an interrupt-mode A/D sequence using a Channel-Gain array.

```
/*************************************************************/
/* CEXAMP2.C                                          DASTC  */
/*                                                           */
/* 'C' - Interrupt Mode A/D transfer                         */
/*       with Channel/Gain Array                             */
/*                                                           */
/* To create an EXE using Microsoft C:                       */
/*                                                           */
/*     CL /c CEXAMP2.C (use /Tp<FileName> for C++ compile)   */
/*     LINK CEXAMP2,,,DASTC+DASRFACE;                        */
/*                                                           */
/* To create an EXE using Borland C++ (Ver 2.0 and up):      */
/*                                                           */
```

```
/*      BCC -ml -c CEXAMP2.C dastc.lib dasrface.lib       */
/*                                                        */
/**********************************************************/



//use this include file statement for MS C
#include "userprot.h"

//use this include file statement for MS C++
/*
extern "C" {
#include "userprot.h"
}
*/

//use this include file for Borland C++ and use -P switch
//for C++ compile
/*
extern "C" {
#include "userprot.bcp"
}
*/

#include <stdio.h>
#define Samples 16
DWORD LocalBuffer[Samples];

GainChanTable ChanGainArray =
{
   16,
   0,0,
   2,0,
   4,0,
   6,0,
   8,0,
   10,0,
   12,0,
   14,0,
   1,0,
   3,0,
   5,0,
   7,0,
   9,0,
   11,0,
   13,0,
   15,0
} ;

main()
```

```
{
DDH DASTC_brd0 ; // handle for board 0
FRAMEH AD_brd0 ; // frame for board 0 A/D operations

long Index;
short BoardNumber, Err, Status, m;
char  NumberOfBoards;
float CJC=0;

//-------------------------------------------------------------
// initialize board hardware and driver

printf("\n");
printf("Initializing the board - - - PLEASE wait\n");

if (( Err = DASTC_DevOpen( "DASTC.CFG", &NumberOfBoards ))
!=0 )
   {
   printf( " Error %x on Device open ", Err ) ;
   return Err ;
   }

//-------------------------------------------------------------
// The DEVICE Handle must be obtained in order to work with
// a specific board
// It is used subsequently to obtain FRAME Handles

BoardNumber = 0;
if ( ( Err = DASTC_GetDevHandle( BoardNumber, &DASTC_brd0 )
) != 0 )
   {
   printf( "Error getting Device Handle" );
   return 1 ;
   }

if ( ( Err=DASTC_GETCJC(BoardNumber, &CJC) ) != 0 )
   {
   printf( "Error getting CJC Temperature" );
   return 1 ;
   }

printf("CJC Temperature = %f\n", CJC);

//-------------------------------------------------------------
// The FRAME Handle must be obtained using the DEVICE Handle
// in order to make each type of function call,
// in this case, Analog Input.
// The variable is suffixed with a "0" to reference board 0.

if ( (Err = K_GetADFrame( DASTC_brd0, &AD_brd0 ) ) != 0 )
```

```
   {
   printf( "Error getting Frame Handle" );
   return 1 ;
   }

//-------------------------------------------------------------
// The FRAME Handle is now used in Analog Input calls.

printf("\n\nInterrupt Mode with Chan Gain Array\n\n\n" );

if ( ( Err = K_SetBuf( AD_brd0, LocalBuffer, Samples )  ) !=
0 )
   {
   printf("Error %x Occurred during K_SetBuf call. . .\n",
Err);
   return 1;
   }

if ( ( Err = K_SetChnGAry( AD_brd0, &ChanGainArray)  ) != 0
)
   {
   Printf("Error %x Occurred during K_SetChnGAry call. .
.\n", Err);
   return 1;
   }

// un-comment this block of code for continuous run
//printf("Continuous Run Selected.\n");
//if ( (Err = K_SetContRun( AD_brd0 ) ) != 0 )
//   {
//   Printf("Error %x Occurred during K_SetContinRun call. .
//.\n", Err);
//   return 1;
//   }

if ( ( Err = K_IntStart( AD_brd0 )  ) != 0 )
   {
   Printf("Error %x Occurred during K_IntStart call. . .\n",
Err);
   return 1;
   }

printf("TYPE any key to stop\n\n");
do
   {
   if ( ( Err = K_IntStatus( AD_brd0 , &Status, &Index ) )
!= 0 )
       {
       Printf("Error %x Occurred during K_IntStatus call. .
.\n", Err);
```

```
        return 1;
        }
    printf("Conversions completed= %6d\r", Index);
    }
while ( (Status & 1) && !_kbhit() );

if ( ( Err = K_IntStop( AD_brd0 , &Status, &Index ) ) != 0 )
    {
    Printf("Error %x Occurred during K_IntStop call. . .\n",
Err);
    return 1;
    }

printf("\n");

for (m = 0; m < Samples ; m++)
   printf("Sample No. %d    %ld\n", m+1, LocalBuffer[m]);

printf("\n");

//-------------------------------------------------------------
// Release memory used by the frame.
if ( ( Err = K_FreeFrame( AD_brd0 )  ) != 0 )
   {
   Printf("Error %x Occurred during K_FreeFrame call. .
.\n", Err);
   return 1;
   }

}
```

Borland Turbo Pascal

Related Files

DASTC.TPU

Compile and Link Instructions

`TPC/$E+ /$N+ filename.pas`

In the Turbo environment

> Options\Compiler\Numeric Processing:
> [x] 8087/80287
> [x] Emulation

Note If you are using a version of Turbo Pascal higher than 6.0: Before compiling

the example program shown below, you must create a TPU (Turbo Pascal unit) file that is compatible with your version. In FILES.DOC you will find a reference to DASTCTPU.BAT. Run this batch file in order to create the compatible TPU. The file, DASTCTPU.BAT contains the DOS command:

```
tpc DASTC.PAS
```

This file also includes the sources for the TPU, and a description of this procedure.

Code example

This example executes an interrupt-mode A/D sequence using a Channel-Gain array.

```
Program tpexamp2;
{
 Interrupt Mode A/D transfer with Channel/Gain Array

 For this example ONLY;
        the configuration file must specify FLOATING POINT.
}


uses Crt, DASTC;

Type
GainChanTable = Record
        num_of_codes : Integer;
        queue : Array[0..31] of Byte;
end;

Const
ChanGainArray : GainChanTable = (
num_of_codes : (16);
queue        : (0,0,
                1,0,
                2,0,
                3,0,
                4,0,
                5,0,
                6,0,
                7,0,
                8,0,
                9,0,
                10,0,
                11,0,
                12,0,
                13,0,
                14,0,
                15,0)
```

```
);

Var
BufPtr : ^Integer;
BoardNumber, m : Integer;
NumberOfBoards : Integer ;
Status, Ertn : Word;
Samples, Index, DASTC_brd0, AD_brd0 : Longint;
ConfigFile : String;
DataBuffer : Array[0..20] of Real;
CJC : Real;

begin

{-------------------------------------------------------------
 initialize board hardware and driver }

BufPtr := @DataBuffer[0];
ConfigFile :=  'DASTC.CFG' + #0 ;

Ertn := DASTC_DevOpen( ConfigFile[1], NumberOfBoards );

if Ertn <> 0 then
   begin
        writeln( 'Error ', Ertn, 'on Device open' );
        Halt(1);
   end;

{-------------------------------------------------------------
 The DEVICE Handle must be obtained in order to work with a
specific board It is used subsequently to obtain FRAME
Handles }

BoardNumber := 0;
Ertn := DASTC_GetDevHandle( BoardNumber, DASTC_brd0 );

if Ertn <> 0 then
   begin
        writeln( 'Error getting Device Handle' );
        Halt(1);
   end;



Ertn := DASTC_GETCJC( BoardNumber, CJC );

if Ertn <> 0 then
   begin
        writeln( 'Error getting CJC Temperature' );
        Halt(1);
```

```
   end;

writeln( 'CJC Temperature = ', CJC );

{-------------------------------------------------------------
 The FRAME Handle must be obtained using the DEVICE Handle
in order to make each type of function call, in this case,
Analog Input. The variable is suffixed with a "0" to
reference board 0. }

Ertn := K_GetADFrame( DASTC_brd0, AD_brd0 ) ;

if Ertn <> 0 then
   begin
        writeln( 'Error getting Frame Handle' );
        Halt(1);
   end;


{-------------------------------------------------------------
The FRAME Handle is now used in Analog Input calls. }

writeln( 'Interrupt Mode with Chan Gain Array' );

Samples := 20;
Ertn := K_SetBuf( AD_brd0, longint(BufPtr), Samples ) ;

if Ertn <> 0 then
   begin
        writeln( 'Error in K_SetBuf call' );
        Halt(1);
   end;

Ertn := K_SetChnGAry( AD_brd0, ChanGainArray.num_of_codes )
;

if Ertn <> 0 then
   begin
        writeln( 'Error in K_SetChnGAry call' );
        Halt(1);
   end;

Ertn := K_IntStart( AD_brd0 ) ;

if Ertn <> 0 then
   begin
        writeln( 'Error in K_IntStart call' );
        Halt(1);
   end;
```

```
repeat

   Ertn := K_IntStatus( AD_brd0 , Status, Index ) ;

   if Ertn <> 0 then
      begin
        writeln( 'Error in K_IntStatus call' );
        Halt(1);
      end;

   writeln( 'Conversions Completed = ', Index );

until (Status AND 1) = 0;

     writeln('');

     for m := 0 to Samples-1 Do
             writeln( DataBuffer[ m ] );

{-------------------------------------------------------------
Release memory used by the frame. }

Ertn := K_FreeFrame( AD_brd0 ) ;

if Ertn <> 0 then
   begin
        writeln( 'Error in K_FreeFrame call' );
        Halt(1);
   end;


end.
```

Borland Turbo Pascal for Windows

Related files

DASTCTPW.INC
DASTC.DLL

Notes

For Windows use DASTC.DLL. The information presented for Borland Turbo Pascal applies here with the following additions:

- Use the compiler directive {$I ... } to include the supplied include file DASTCTPW.INC.
- Substitute 'WinCrt' for the 'Crt' unit; this is necessary in order that the console I/O procedures (writeln, readln, etc...) operate properly.

The following code fragment illustrates these substitutions:

```
Program TPW_EX;
{ UNITS USED BY THIS PROGRAM }
Uses WinCrt;

{ LOCAL VARIABLES }
Var

{ BEGIN MAIN MODULE }
BEGIN

{ $I DASTCTPW.INC}
```

Code example

This example executes an interrupt-mode A/D sequence using a channel-gain array.

```
Program TPWEX2;
{
*************************************************************
TPWEX2.PAS
DASTC

 Turbo Pascal for Windows :

 The following is an example program that demonstrates the
use of AD interrupt conversions using a channel/gain queue.

*************************************************************
}

{ The WinCrt unit allows Windows to handle 'writeln' and
'readln' the same
  way as in DOS }

uses WinCrt;

Type
GainChanTable = Record
        num_of_codes : Integer;
        queue : Array[0..31] of Byte;
end;

Const
ChanGainArray : GainChanTable = (
num_of_codes : (16);
queue        : (0,0,
               1,0,
               2,0,
               3,0,
```

```
                4,0,
                5,0,
                6,0,
                7,0,
                8,0,
                9,0,
                10,0,
                11,0,
                12,0,
                13,0,
                14,0,
                15,0)
);

Var
BufPtr : ^Integer;
BoardNumber, m : Integer;
NumberOfBoards : Integer ;
Ertn, chan, Status : Word;
Samples, Index, InPort, DASTC_brd0, AD_brd0 : Longint;
ConfigFile : String;
DataBuffer : Array[0..20] of Longint;
CJC : Single;


{ ------------------------------------------------------------}

{$I DASTCTPW.INC}                { DLL function prototypes. }

begin

{-------------------------------------------------------------
initialize board hardware and driver }

BufPtr := @DataBuffer[0];
ConfigFile :=  'DASTC.CFG' + #0 ;

Ertn := DASTC_DevOpen( ConfigFile[1], NumberOfBoards );

if Ertn <> 0 then
   begin
        writeln( 'Error ', Ertn, 'on Device open' );
        Halt(1);
   end;

{-------------------------------------------------------------
 The DEVICE Handle must be obtained in order to work with a
specific board It is used subsequently to obtain FRAME
Handles }
```

```
BoardNumber := 0;
Ertn := DASTC_GetDevHandle( BoardNumber, DASTC_brd0 );

if Ertn <> 0 then
   begin
        writeln( 'Error getting Device Handle' );
        Halt(1);
   end;

{-------------------------------------------------------------
The FRAME Handle must be obtained using the DEVICE Handle in
order to make each type of function call, in this case,
Analog Input. The variable is suffixed with a "0" to
reference board 0. }

Ertn := K_GetADFrame( DASTC_brd0, AD_brd0 ) ;

if Ertn <> 0 then
   begin
        writeln( 'Error getting Frame Handle' );
        Halt(1);
   end;


{-------------------------------------------------------------
The FRAME Handle is now used in Analog Input calls. }

writeln( 'Interrupt Mode with Chan Gain Array' );

Samples := 20;
Ertn := K_SetBuf( AD_brd0, longint(bufptr), Samples ) ;

if Ertn <> 0 then
   begin
        writeln( 'Error in K_SetBuf call' );
        Halt(1);
   end;

Ertn := K_SetChnGAry( AD_brd0, ChanGainArray.num_of_codes )
;

if Ertn <> 0 then
   begin
        writeln( 'Error in K_SetChnGAry call' );
        Halt(1);
   end;

Ertn := K_IntStart( AD_brd0 ) ;

if Ertn <> 0 then
```

```
   begin
        writeln( 'Error in K_IntStart call' );
        Halt(1);
   end;

repeat

Ertn := K_IntStatus( AD_brd0 , Status, Index ) ;

if Ertn <> 0 then
   begin
        writeln( 'Error in K_IntStatus call' );
        Halt(1);
   end;


until (Status AND 1) = 0;

     writeln('');
     writeln( 'Interrupts Completed ');
     writeln('');

     for m := 0 to Samples-1 Do
             writeln( 'Channel [',m,'] = ',DataBuffer[ m ]
);

(-------------------------------------------------------------
Release memory used by the frame. )

Ertn := K_FreeFrame( AD_brd0 ) ;

if Ertn <> 0 then
   begin
        writeln( 'Error in K_FreeFrame call' );
        Halt(1);
   end;


end.
```

Microsoft Quick C for Windows

Related files

DASTC.DLL

Compile and Link Instructions

1. Load *filename*.C into the Quick C for Windows environment if you are editing this file.
2. Create a project file, that includes *filename*.C, *filename*.DEF, *filename*.RC and *filename*.H.
3. Select PROJECT ▸ BUILD to create a stand-alone .EXE that can be executed from within Windows.

Notes

The .DEF file must be included to import functions from DASTC.DLL.

The programming procedure required to call the functions from Quick C for Windows programs is identical to the procedure described for Microsoft C.

Code example

This example executes an interrupt-mode A/D sequence using a Channel-Gain array.

```
/*************************************************************
*
*
*          Keithley/Metrabyte DASTC Example Program for
*
*          Microsoft Windows 3.0 and 3.1
*
*
*
*        This Program Accesses the DASTC functions through
*
*        DASTC.DLL.
*
*        this is fragment taken from the "WINEXAMP.C" program
*
*************************************************************/
#include "winexamp.h"
#include "userprot.h"


long LocalBuffer[20];            // Declare a buffer for out
AD Data

long far *FirstElement;          // Pointer to Interrupt
Buffer
WORD MemHandle;                 // Handle of the above
Pointer
```

```
char NumberOfBoards;            // Number of boards to
configure

short Done = 0;                   // Operation Done Flag

short  Status;                    // Status variable for
Interrupt
long Index;                     // Index variable for
Interrupt
short  Err;                       // Return value from the
functions

DDH  DASTC;                     // Device Handle
FRAMEH AD;                      // Frame Handle

float CJC;



//**** Open the config file and read it...

if((Err=DASTC_DevOpen("DASTC.cfg", &NumberOfBoards)) != 0)

  {
      wsprintf(szErr,"DASTC Error = %4x", Err);
       MessageBox(NULL, szErr," Error ", MB_OK |
          MB_ICONEXCLAMATION);
   return 1;
  }
 //**** Now get a Device Handle
if((Err=DASTC_GetDevHandle(0,&DASTC)) != 0)
  {
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr," Error ", MB_OK |
          MB_ICONEXCLAMATION);
   return 1;
  }

//**** Now get the CJC Temperature
if((Err=DASTC_GETCJC(0,&CJC)) != 0)
  {
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr," Error ", MB_OK |
          MB_ICONEXCLAMATION);
   return 1;
  }

      wsprintf(szData,"CJC Temperature = %f", CJC);
```

```
//**** Setup for INTERRUPT AD Conversions
//**** Get a AD Frame
if((Err = K_GetADFrame(DASTC, &AD )) != 0)
  {
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr," Error ", MB_OK |
          MB_ICONEXCLAMATION);
   return 1;
  }

//**** Allocate a Buffer
if((Err = K_IntAlloc(AD, 16, &FirstElement, &MemHandle))
!=0)
  {
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr," Error ", MB_OK |
          MB_ICONEXCLAMATION);
   return 1;
  }
//**** Tell the Frame about the Buffer
if((Err = K_SetBuf(AD, FirstElement, 16)) != 0)
  {
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr," Error ", MB_OK |
          MB_ICONEXCLAMATION);
   return 1;
  }
//**** Set the Start/Stop Channels and Gain
if((Err = K_SetStartStopChn(AD, 0, 15)) != 0)
  {
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr," Error ", MB_OK |
          MB_ICONEXCLAMATION);
   return 1;
  }

IrqOP = 1;              // Set Operation Flag
Done = 0;              // Clear Done Flag
Status = 0;             // Clear Interrupt Status Flag

UpdateWindow(hWndMain);  // Print Running

//**** Start Interrupt MODE AD
if((Err = K_IntStart(AD)) != 0)
  {
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr," Error ", MB_OK |
          MB_ICONEXCLAMATION);
   return 1;
  }
```

```
//**** Start a 10ms timer to monitor status
if(!SetTimer(hWndMain, ID_TIMER, 10, NULL))
  {
      MessageBox(NULL, "TIMER ERROR...","  Error ", MB_OK |
          MB_ICONEXCLAMATION);
   return 1;
  }

#############################################################
// timer routine that polls for interrupt completion

if((Err = K_IntStatus(AD, &Status, &Index)) != 0)
  {
      KillTimer(hWndMain, ID_TIMER);

      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr,"  Error ", MB_OK |
          MB_ICONEXCLAMATION);

if((Err = K_IntStop(AD, &Status, &Index)) != 0)
  {                                          // Free the frame
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr,"  Error ", MB_OK |
          MB_ICONEXCLAMATION);
  }
      if((Err = K_IntFree(MemHandle)) != 0)
        {                                    // Free the frame
          wsprintf(szErr,"DASTC Error = %4x", Err);
          MessageBox(NULL, szErr,"  Error ", MB_OK |
          MB_ICONEXCLAMATION);
        }
      if((Err = K_FreeFrame(AD)) != 0)
        {                                     // Free the frame
          wsprintf(szErr,"DASTC Error = %4x", Err);
          MessageBox(NULL, szErr,"  Error ", MB_OK |
            MB_ICONEXCLAMATION);
        }
      break;
 }

InvalidateRgn(hWndMain, hRgn, FALSE);  // Update client Area
//                                            with count

if((Status & 1)==0)
  {
      KillTimer(hWndMain, ID_TIMER);
      if((Err = K_IntStop(AD, &Status, &Index)) != 0)
        {                                    // Free the frame
          wsprintf(szErr,"DASTC Error = %4x", Err);
          MessageBox(NULL, szErr,"  Error ", MB_OK |
```

```
                MB_ICONEXCLAMATION);
    }

  // Move Data to our Local Buffer
      K_MoveDataBuf(LocalBuffer, FirstElement, 16);

  if((Err = K_IntFree(MemHandle)) != 0)
    {                                    // Free the frame
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr," Error ", MB_OK |
         MB_ICONEXCLAMATION);
    }
  if((Err = K_FreeFrame(AD)) != 0)
    {                                     // Free the frame
      wsprintf(szErr,"DASTC Error = %4x", Err);
      MessageBox(NULL, szErr," Error ", MB_OK |
         MB_ICONEXCLAMATION);
    }
```

Microsoft Visual Basic for Windows

Related files

DASTC.DLL
DASTCGLB.BAS
Q4IFACE.BI

Notes

Before you begin coding your Visual Basic program, you must copy (from inside the Visual Basic environment) the contents of DASTCGLB.BAS into your application's GLOBAL.BAS. Use the following procedure to add the contents of DASTCGLB.BAS to GLOBAL.BAS (you should make a back-up copy of GLOBAL.BAS before you modify it):

1. Select FILE ▸ ADD FILE... from the Visual Basic main menu.
2. Select DASTCGLB.BAS.
3. Highlight the contents of the entire DASTCGLB.BAS file.
4. Select EDIT ▸ COPY to copy the contents of DASTCGLB.BAS to the Windows clipboard.
5. Double-click on GLOBAL.BAS in the Project window.
6. Select EDIT ▸ PASTE.
7. Select FILE ▸ SAVE PROJECT.

Code example

This example executes an interrupt-mode A/D input operation using a Channel-Gain array.

```
##################################################################
DAS-TC Visual Basic Example:
Interrupt transfer using Gain/Channel array with local
buffer.

This is a code fragment taken from DASTCEX1.FRM.
##################################################################



##################################################################
Channel / Gain Interrupt Event routine

(double click on the "Channel / Gain Queue" START button in
the "Interrupt Mode A/D" frame to see this code)
##################################################################

Sub StartQInt_Click ()

scalemode = 2

timer2.enabled = False          'Disable our Timer

Cls

SSFlag = False

Print
Print "Scan is using Channel / Gian Queue"
Print

MyErr = DASTC_devopen("DASTC.cfg", board%)
If MyErr <> 0 Then
 MsgBox "DASTC_devopen Error", 48, "Error"
  Exit Sub
End If

MyErr = DASTC_getdevhandle(0, DASTC)
If MyErr <> 0 Then
 MsgBox "DASTC_getdevhandle Error", 48, "Error"
  Exit Sub
End If

MyErr = K_GetADFrame(DASTC, ad)
If MyErr <> 0 Then
 MsgBox "K_GetADFrame Error", 48, "Error"
  Exit Sub
End If
```

```
MyErr = k_clearframe(ad)
If MyErr <> 0 Then
 MsgBox "K_ClearFrame Error", 48, "Error"
  Exit Sub
End If

MyErr = K_IntAlloc(ad, samples, GBuffer, HANDLE)
If MyErr <> 0 Then
 MyErr = K_FreeFrame(ad)
 o$ = "K_IntAlloc Error = " + Hex$(MyErr)
 MsgBox o$, 48, "Error"
  Exit Sub
End If

Print "        Buffer Handle = "; Hex$(HANDLE)
Print " AD Interrupt Buffer = "; Hex$(GBuffer)


MyErr = K_SetBuf(ad, ByVal GBuffer, samples)
If MyErr <> 0 Then
 MyErr = K_FreeFrame(ad)
 MyErr = K_IntFree(HANDLE)
 MsgBox "K_SetBuf Error", 48, "Error"
  Exit Sub
End If

MyErr = K_SetChnGAry(ad, ChanGainArray(0))
If MyErr <> 0 Then
 MyErr = K_FreeFrame(ad)
 MsgBox "K_SetChnGAry Error", 48, "Error"
  Exit Sub

End If

MyErr = K_IntStart(ad)
If MyErr <> 0 Then
 MyErr = K_FreeFrame(ad)
 MsgBox "K_IntStart Error", 48, "Error"
  Exit Sub
End If

 Status = 1                    ' Enable Status Flag

timer2.enabled = True           ' Enable our Timer

End Sub
```

```
##################################################################
Timer routine used to detect interrupt completion and then
to transfer data.

(double click on timer icon to see this code)
##################################################################

Sub Timer2_Timer ()

MyErr = K_IntStatus(ad, Status, Index)
If MyErr <> 0 Then
 MyErr = K_IntStop(ad, Status, Index)
 MyErr = K_FreeFrame(ad)
 MsgBox "K_IntStatus Error", 48, "Error"
  Exit Sub
End If

PSet (0, 55)
o$ = "Count = " + Format$(Index, "######")
Print o$

If (Status And 1) = 0 Then

    timer2.enabled = False

    MyErr = K_IntStop(ad, Status, Index)
    MyErr = K_MoveDataBuf(Buffer(0), ByVal GBuffer, samples
* 2)
    MyErr = K_IntFree(HANDLE)
    If MyErr <> 0 Then
        o$ = "K_IntFree Error = " + Hex$(MyErr)
        MsgBox o$, 48, "Error"
        Exit Sub
    End If

    Print : Print "   Interrupt Operation Complete..."

    MyErr = K_FreeFrame(ad)

    Print

    For x = 0 To samples - 1
        Print "     Buffer("; x; ") = "; Buffer(x)
    Next x
End If
End Sub
```

Functions 3

3.1 Functional grouping

The function calls can be classified according to the functionality that each provides. This section lists each function as a member of one of the following groups:

- Initialization
- Memory management
- Frame management
- Frame-element management
- Frame-based operation control
- Single-call I/O
- Miscellaneous operations

This section provides short descriptions of each function; refer to Section 3.2 for additional information on each function.

Initialization

DASTC_DevOpen	Initialize and configure the driver.
DASTC_GetDevHandle	Obtain a device handle.
K_DASDevInit	Reset and initialize the device and driver.

Memory management

K_IntAlloc	Allocate a buffer suitable for an interrupt-mode A/D operation.
K_IntFree	De-allocate an interrupt buffer that was previously allocated with **K_IntAlloc**.
K_MoveDataBuf	Transfer acquired A/D samples between a memory buffer and an array.

Frame management

K_FreeFrame	Free the memory used by a frame and return the frame to the pool of available frames.
K_GetADFrame	Obtain the handle to an A/D frame.

Frame-element management

K_ClearFrame	Clears all the elements of an A/D frame.
K_ClrContRun	Set the value of a frame's Buffering Mode element to SINGLE-CYCLE.
K_FormatChnGAry	Convert a Visual Basic Channel-Gain array into an equivalent Function Call Driver Channel-Gain array (Visual Basic Only).
K_GetBuf	Get the values of an A/D frame's Buffer Address and Number of Samples elements.
K_GetChnGAry	Get the value of an A/D frame's Channel-Gain Array Address element.
K_GetContRun	Get the value of a frame's Buffering Mode element.
K_GetStartStopChn	Get the values of an A/D frame's Start Channel and Stop Channel elements.

K_InitFrame	Initialize a board's A/D circuitry and set an A/D frame's elements to their default values.
K_RestoreChnGAry	Convert a Function Call Driver Channel-Gain array into an equivalent Visual Basic Channel-Gain array (Visual Basic only).
K_SetBuf	Set the values of an A/D frame's Buffer Address and Number of Samples elements (Pascal and C languages only).
K_SetBufL	Set the values of a frame's Buffer Address and Number of Samples elements for user-defined long integer arrays (Visual Basic for Windows only).
K_SetBufR	Set the values of a frame's Buffer Address and Number of Samples elements for user-defined floating-point arrays (Visual Basic for Windows only).
K_SetChnGAry	Set the value of a frame's Channel-Gain Array Address element.
K_SetContRun	Set the value of a frame's Buffering Mode element to CONTINUOUS.
K_SetStartStopChn	Set the values of an A/D frame's Start Channel and Stop Channel elements.

Frame-based operation control

K_IntStart	Start an interrupt-mode A/D operation.
K_IntStatus	Determine the status of an interrupt-mode A/D operation.
K_IntStop	Abort an interrupt-mode A/D operation.
K_SyncStart	Start a synchronous-mode A/D operation.

Single-call I/O

K_ADRead	Read a single A/D value.
DASTC_GETCJC	Returns the value of the CJC on the DAS-TC in degrees Celsius; this value is used to correct temperature input values.

Miscellaneous operations

K_GetErrMsg	Get the address of an error message string (available only as C-language function).
K_GetVer	Determine the driver revision and driver specification.

3.2 Function reference

This section contains reference entries each function. The entries appear in alphabetical order by function name. These reference entries provide the details associated with the use of each function.

The information related to the following topics pertains to several functions:

- the gain codes the driver uses to represent gains and the A/D input ranges that correspond to each gain
- the return value for every call to a Function

These topics are described in the next several paragraphs and referred to throughout the reference entries that follow.

Gains

The ASO drivers use gain codes to represent gains. The valid gain codes are **0**, **1**, **2**, **3**; the table below lists the gains that correspond to these gain codes, as well as the A/D input ranges affected by each gain.

Table 1 DAS-TC gains and A/D voltage gains

gain code	DAS-TC gain	DAS-TC voltage input range
0	1	-2.5 to +10 V
1	125	-20 to 80 mV
2	166.67	-15 to 60 mV
3	400	-6.25 to 25 mV

Note Gains are only available when a channel is configured as a voltage input. You can program the gain only through an A/D input operation that uses a Channel-Gain array. If you are acquiring data by either a **K_ADRead**, or an A/D input operation that uses Start/Stop channels, the gain from the .CFG file is used.

Return values

Strictly speaking, the function return value is of type error. "Returns" is also used to mean that the driver executes the function and stores the result in user-defined variables or allocated buffers. Whether used as placeholder, to pass a value, or to contain results from a function return, a variable must be declared with a type consistent with the corresponding parameter.

Number type

The number type that is returned is either Integer or Floating Point, and is determined by one of the following:

- the built-in default (which is the same as the configuration file at distribution time);
- the default configuration file (DASTC.CFG); or,
- the specified configuration file.

Buffers

When the number type is integer, a twos complement 32-bit number is returned. If a particular channel is configured as a temperature input, the value returned is in .01 degrees. If a particular channel is configured as a voltage input, the value returned in in microvolts. To convert .01 degrees to degrees, divide the value by 100; to convert microvolts to volts, divide the value by 1,000,000.

When the number type is floating point, an IEEE 32-bit real number is returned. The value returned is in volts or degrees.

Declare a user-defined data buffer with a type appropriate to the number type that was configured.

A single sample is four bytes long. Therefore, you should declare a local buffer as an array of four-byte elements, the size of which is at least equal to the number of samples you are requesting.

Declare pointers to buffers allocated by **K_IntAlloc** with a type that is appropriate to the number type that was configured.

Purpose

Initialize and configure the driver.

Prototype

C

DASErr far pascal DASTC_DevOpen(char far * *cfgFile*,
char far * *numDevices*);

Pascal

Function DASTC_DevOpen(Var *cfgFile* : char;
Var *numDevices* : Integer) : Word;

Visual Basic for Windows

DASTC_DevOpen Lib "DASTC.dll" (ByVal *cfgFile*,
numDevices As Integer) As Integer

Parameters

cfgFile — Driver configuration file

numDevices — Number of devices defined in *cfgFile*. Valid values: **1**, **2**

Notes

DASTC_DevOpen initializes the driver according to the information in *cfgFile*. On return, *numDevices* contains the number of devices for which *cfgFile* contains configuration information.

If *cfgfile* is **-1**, the built-in defaults are used. They are identical to the defaults in the DASTC.CFG file when this file is initially distributed. This file specifies that the device is set as follows:

Board Number	0	1
Base Address	300h	308h
Interrupt Level	7	5

The following parameters have the same defaults for both TC boards:

Normal Mode Rejection Frequency	60 Hz
Number Type	Integer
Units	C
Number of Readings to average	1
CJC Correction	ON

Specify **0** for *cfgFile* to cause the driver to search for DASTC.CFG.

Purpose

Returns the value of the CJC on the DAS-TC in degrees Celsius; this value is used to correct temperature input values.

Prototype

C

DASErr far pascal DASTC_GETCJC (int *devNumber*, float far * *CJCtemp*);

Pascal

Function DASTC_GETCJC (*devNumber* : Integer; Var *CJCtemp* : Single) : Word;

Visual BASIC for Windows

DASTC_GETCJC Lib "DASTC.dll" (ByVal *devNumber* As Integer, *CJCtemp* As Single) As Integer

Parameters

devNumber Board number. Valid values: **0**, **1**

CJCtemp CJC sensor temperature value in degrees Celsius.

Notes

This function call reads temperature at the STA-TC or STC-TC terminals.

Upon return, *CJCtemp* contains the CJC (Cold Junction Compensation) temperature associated with the device identified by *devNumber*. The value stored in *CJCtemp* is floating point regardless of the format specified in the configuration file.

In order to obtain a temperature reading from a thermocouple type not recognized by the Driver, you need to perform your own linearization. For a corrected temperature reading, you can call **DASTC_GETCJC** and use the resulting value to correct the linearization.

Depending upon the volatility of the ambient temperature where the CJC resides, the more samples you take, the more often you should call **DASTC_GETCJC**.

This call does not use a frame.

An error is returned if an Interrupt operation is in progress.

Purpose

Obtain a device handle.

Prototype

C

DASErr far pascal DASTC_GetDevHandle (int *devNumber*, void far * far * *devHandle*);

Pascal

Function DASTC_GetDevHandle(*devNumber* : Integer; Var *devHandle* : Longint) : Word;

Visual Basic for Windows

DASTC_GetDevHandle Lib "DASTC.dll" (ByVal *devNumber* As Integer, *devHandle* As Long) As Integer

Parameters

devNumber	Device number. Valid values: **0**, **1**
devHandle	Device handle

Notes

On return, *devHandle* contains the handle associated with the device identified by *devNumber*.

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

The driver supports up to two DAS-TC boards; a unique handle must be associated with each supported board.

In addition to obtaining a device handle, **DASTC_GetDevHandle** performs the following tasks:

- aborts all in-progress A/D operations
- checks if device identified by *devHandle* is present
- checks if settings in configuration file match actual board settings
- initializes the board to its default state

Purpose

Read a single A/D value.

Prototype

C

DASErr far pascal K_ADRead(DDH *devHandle*, unsigned char *chan*, unsigned char *gainCode*, void far * *ADvalue*);

Pascal

Function K_ADRead(*devHandle* : Longint; *chan* : Byte; *gainCode* : Byte; Var *ADvalue* : Longint) : Word;

Visual BASIC for Windows

K_ADRead Lib "DASTC.dll" (ByVal *devHandle* As Long, ByVal *chan* As Integer, ByVal *gainCode* As Integer, *ADvalue* As Long) As Integer

Parameters

devHandle	Device handle
chan	Input channel. Valid values: **0**, **1**,..., **15**
gainCode	Gain code is ignored. A value of **0** must be passed even though it is not used. The reading is returned according to the configured gain.
ADvalue	Storage location of acquired A/D value

Notes

On return, *ADvalue* contains the value read from channel *chan* of the device identified by *devHandle*.

See Table 1, page 47 for the A/D voltage ranges and their corresponding gains.

The return values are in microvolts or .01 degrees for integer types, and are not scaled for floating point.

This function returns an error if an Interrupt operation is in progress.

Purpose

Clears all the elements of an A/D frame.

Prototype

C

DASErr far pascal K_ClearFrame(FRAMEH *frameHandle*);

Pascal

Function K_ClearFrame(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_ClearFrame Lib "DASTC.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Frame handle

Notes

K_ClearFrame initializes to zero all of the elements in the frame identified by *frameHandle*.

Purpose

Set the value of a frame's Buffering Mode element to SINGLE-CYCLE.

Prototype

C

DASErr far pascal K_ClrContRun(FRAMEH *frameHandle*);

Pascal

Function K_ClrContRun(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_ClrContRun Lib "DASTC.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Frame handle

Notes

K_ClrContRun sets the Buffering Mode to SINGLE-CYCLE in the frame identified by *frameHandle*.

Purpose

Reset and initialize the device and driver.

Prototype

C

DASErr far pascal K_DASDevInit(DDH *devHandle*);

Pascal

Function K_DASDevInit(*devHandle* : Longint) : Word;

Visual BASIC for Windows

K_DASDevInit Lib "DASTC.dll" (ByVal *devHandle* As Long)
As Integer

Parameters

devHandle Device handle

Notes

K_DASDevInit performs the following tasks:

- Aborts all in-progress A/D operations
- Checks if device identified by *devHandle* is present
- Checks if settings in configuration file match actual board settings
- Initializes the board to its internal defaults or to the configuration file values.

Purpose

Convert a Visual Basic Channel-Gain array into an equivalent Function Call Driver Channel-Gain array (Visual Basic Only).

Prototype

Visual Basic for Windows

K_FormatChnGAry Lib "DASTC.dll" (*chanGainArray* As Integer) As Integer

Parameters

chanGainArray Storage location for Channel-Gain Array

Notes

A Channel-Gain Array defines two characteristics of an A/D operation:

- the sequence in which the input channels are sampled and,
- the gain applied to each of the channels configured for voltage in that sequence.

A Channel-Gain Array can define up to 16 randomly sequenced channel-gain pairs. Adjacent pairs can specify the same channel (with equal or unequal gains). The following table illustrates the required format of a Channel-Gain array for Visual Basic.

Integer	0	1	2	3	4	...	2N-1	2N
Value	N	chan	gain	chan	gain	...	chan	gain
	# of pairs	pair 1		pair 2		...	pair N	

The gain must be specified as a gain code. Refer to Table 1 on page 47 for the input range affected by each gain.

Gain Code	0	1	2	3
Gain	1	125	166.67	400

K_FormatChnGAry converts the Visual Basic Channel-Gain array identified by *chanGainArray* into an equivalent Channel-Gain array but formatted for use by the Function Call Driver. On return, *chanGainArray* identifies the resulting array, which replaces the Visual Basic array. The function, **K_SetChnArray**, requires you to pass a reference to the resulting array, which is unreadable in Visual Basic. To resore the array so that it is readable from Visual Basic, use the complementary function, **K_RestoreChnGary**.

A Channel-Gain array enables you to specify different gains for different input channels.

Purpose

Free the memory used by a frame and return the frame it to the pool of available frames.

Prototype

C

DASErr far pascal K_FreeFrame(FRAMEH *frameHandle*);

Pascal

Function K_FreeFrame(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_FreeFrame Lib "DASTC.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Frame handle

Notes

K_FreeFrame frees the memory used by the frame identified by *frameHandle*; the frame is then returned to the pool of available frames. The frame elements are automatically cleared to zero.

Do not use this function if you plan to use the same frame for future calls to the driver.

Purpose

Obtain the handle to an A/D frame.

Prototype

C

DASErr far pascal K_GetADFrame(DDH *devHandle*,
FRAMEH far * *frameHandle*);

Pascal

Function K_GetADFrame(*devHandle* : Longint;
Var *frameHandle* : Longint) : Word;

Visual Basic for Windows

K_GetADFrame Lib "DASTC.dll" (ByVal *devHandle* As Long,
frameHandle As Long) As Integer

Parameters

devHandle	Device handle
frameHandle	Handle to A/D frame

Notes

On return, *frameHandle* contains the handle to an A/D frame associated with the device identified by *devHandle*.

Purpose

Get the values of an A/D frame's Buffer Address and Number of Samples elements.

Prototype

C

DASErr far pascal K_GetBuf(FRAMEH *frameHandle*, void far * far * *bufAddr*, long far * *samples*);

Pascal

Function K_GetBuf(*frameHandle* : Longint; Var *bufAddr* : Integer; Var *samples* : Longint) : Word;

Visual Basic for Windows

K_GetBuf Lib "DASTC.dll" (ByVal *frameHandle* As Long, *bufAddr* As Long, *samples* As Long) As Integer

Parameters

frameHandle	Frame handle
bufAddr	Buffer Address
samples	Number of Samples

Notes

On return, the following parameters contain the value of an element in the frame identified by *frameHandle*:

- *bufAddr* contains the value of the Buffer Address element
- *samples* contains the value of the Number of Samples element

Purpose

Get the value of an A/D frame's Channel-Gain Array Address element.

Prototype

C

DASErr far pascal K_GetChnGAry(FRAMEH *frameHandle*,
void far * far * *chanGainArray*);

Pascal

Function K_GetChnGAry(*frameHandle* : Longint;
Var *chanGainArray* : Integer) : Word;

Visual Basic for Windows

K_GetChnGAry Lib "DASTC.dll" (ByVal *frameHandle* As Long,
chanGainArray As Long) As Integer

Parameters

frameHandle	Handle to A/D frame
chanGainArray	Channel-Gain Array Address

Notes

On return, *chanGainArray* contains the value of the Channel-Gain Array Address element in the frame identified by *frameHandle*.

Refer to **K_SetChnGAry** for a description of Channel-Gain arrays.

Purpose

Get the value of a frame's Buffering Mode element.

Prototype

C

DASErr far pascal K_GetContRun(FRAMEH *frameHandle*,
short far * *mode*);

Pascal

Function K_GetContRun(*frameHandle* : Longint;
Var *mode* : Word) : Word;

Visual Basic for Windows

K_GetContRun Lib "DASTC.dll" (ByVal *frameHandle* As Long,
Mode As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
mode	Code that indicates Buffering Mode, **0**=Single-cycle, **1**=Continuous

Notes

On return, *mode* contains a code that indicates the Buffering Mode in the frame identified by *frameHandle*.

Purpose

Get the address of an error message string. This function is available only as a C-language function.

Prototype

C

DASErr far pascal K_GetErrMsg(DDH *devHandle*, short *msgNum*,
char far * far * *errMsg*);

Parameters

devHandle	Device handle
msgNum	Error message number
errMsg	Error message string

Notes

On return, *errMsg* contains a pointer to a string that corresponds to *msgNum* for the device identified by *devHandle*.

Refer to Appendix A for error numbers and error messages.

Purpose

Get the values of an A/D frame's Start Channel and Stop Channel elements.

Prototype

C

DASErr far pascal K_GetStartStopChn(FRAMEH *frameHandle*,
short far * *start*, short far * *stop*);

Pascal

Function K_GetStartStopChn(*frameHandle* : Longint; Var *start* : Word;
Var *stop* : Word) : Word;

Visual Basic for Windows

K_GetStartStopChn Lib "DASTC.dll" (ByVal *frameHandle* As Long,
start As Integer, *stop* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
start	Start Channel. Valid values: **0**, **1**,...,**15**
stop	Stop Channel. Valid values: **0**, **1**,...,**15**

Notes

On return, the following parameters contain the value of an element in the frame identified by *frameHandle*:

- *start* contains the value of the Start Channel element
- *stop* contains the value of the Stop Channel element

Purpose

Determine the driver revision and driver specification.

Prototype

C

DASErr far pascal K_GetVer(DDH *devHandle*, short far * *spec*,
short far * *version*);

Pascal

Function K_GetVer(*devHandle* : Longint; Var *spec* : Word;
Var *version* : Word) : Word;

Visual Basic for Windows

K_GetVer Lib "DASTC.dll" (ByVal *devHandle* As Long, *spec* As Integer,
version As Integer) As Integer

Parameters

devHandle	Device handle
spec	Driver specification
version	Driver version

Notes

On return, *spec* contains the revision number of the Keithley DAS Driver Specification to which the driver conforms; *version* contains the driver's version number.

spec and *version* are two-byte integers; the high byte contains the major revision level and the low byte contains the minor revision level (in the version number **2.1**, for example, the major and minor revision levels are **2** and **1**, respectively).

On return, use the following equations to extract the major and minor revision levels from either *spec* or *version*:

$$\text{major revision level} = \frac{\textit{returned value}}{256}$$

The remainder is dropped.

minor revision level = *returned value* MOD 256

where *returned value* represents either *spec* or *version*.

Purpose

Initialize a board's A/D circuitry and set an A/D frame's elements to their default values.

Prototype

C

DASErr far pascal K_InitFrame(FRAMEH *frameHandle*);

Pascal

Function K_InitFrame(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_InitFrame Lib "DASTC.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Handle to A/D frame

Notes

K_InitFrame initializes the A/D circuitry on the DAS-TC that is associated with the frame identified by *frameHandle*.

If an interrupt-mode A/D operation is not active, **K_InitFrame** checks the validity of the board number associated with the frame identified by *frameHandle* and enables A/D operations.

If an interrupt-mode A/D operation is active, **K_InitFrame** returns an error that indicates that the board is busy.

Purpose

Allocate a buffer suitable for an interrupt-mode A/D operation.

Prototype

C

DASErr far pascal K_IntAlloc(FRAMEH *frameHandle*, DWORD *samples*,
void far * far * *intAddr*, WORD far * *memHandle*);

Pascal

Function K_IntAlloc(*frameHandle* : Longint ; *samples* : LongInt;
Var *intAddr* : Longint ; Var *memHandle* : Word) : Word;

Visual Basic for Windows

K_IntAlloc Lib "DASTC.dll" (ByVal *frameHandle* As Long,
ByVal *samples* As Long, *intAddr* As Long, *memHandle* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
samples	Number of samples. Valid values: **0-65,535**
intAddr	Address of interrupt buffer
memHandle	Handle to interrupt buffer

Notes

On return, *intAddr* contains the address of a buffer that is suitable for an interrupt-mode A/D operation of *samples* samples; *memHandle* contains a handle to the buffer that this function allocates.

Purpose De-allocate an interrupt buffer that was previously allocated with **K_IntAlloc**.

Prototype

C

DASErr far pascal K_IntFree(WORD *memHandle*);

Pascal

Function K_IntFree(*memHandle* : Word) : Integer;

Visual Basic for Windows

K_IntFree Lib "DASTC.dll" (ByVal *memHandle* As Integer) As Integer

Parameters

memHandle	Handle to interrupt buffer

Notes **K_IntFree** de-allocates the interrupt buffer identified by *memHandle*.

Purpose

Start an interrupt-mode A/D operation.

Prototype

C

DASErr far pascal K_IntStart(FRAMEH *frameHandle*);

Pascal

Function K_IntStart(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_IntStart Lib "DASTC.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Handle to A/D frame

Notes

K_IntStart starts the interrupt-mode A/D operation defined in the frame identified by *framehandle*. An error is returned if an Interrupt operation is in progress.

Acquired samples are stored at a location identified by the Buffer Address element of the frame identified by *frameHandle*.

The values acquired are in microvolts or .01 degrees for integer types, and are not scaled for floating point.

Purpose

Determine the status of an interrupt-mode A/D operation.

Prototype

C

DASErr far pascal K_IntStatus(FRAMEH *frameHandle*, short far * *status*, long far * *index*);

Pascal

Function K_IntStatus(*frameHandle* : Longint; Var *status* : Word; Var *index* : Longint) : Word;

Visual Basic for Windows

K_IntStatus Lib "DASTC.dll" (ByVal *frameHandle* As Long, *status* As Integer, *index* As Long) As Integer

Parameters

frameHandle	Handle to A/D frame
status	Code that indicates status of interrupt operation. Valid values: **0** = Interrupt-mode A/D operation idle **1** = Interrupt-mode A/D operation active
index	Buffer array index. Used by this function to store the buffer array index.

Notes

On return, *status* contains a code that indicates the status of the Interrupt operation defined by the frame identified by *frameHandle; index* contains the number of the next buffer element, at the time the function was called, which is to be written with the next sample.

For Continuous buffer mode, *index* is reset to zero when the last block transfer is completed and another acquisition cycle has been initiated.

Purpose

Abort an interrupt-mode A/D operation.

Prototype

C

DASErr far pascal K_IntStop(FRAMEH *frameHandle*, short far * *status*, long far * *index*);

Pascal

Function K_IntStop(*frameHandle* : Longint; Var *status* : Word; Var *index* : Longint) : Word;

Visual Basic for Windows

K_IntStop Lib "DASTC.dll" (ByVal *frameHandle* As Long, *status* As Integer, *index* As Long) As Integer

Parameters

frameHandle	Handle to A/D frame
status	Code that indicates status of interrupt operation. Valid values: **0** = Interrupt operation idle **1** = Interrupt operation active (interrupt was stopped)
index	Buffer array index. Used by this function to store the buffer array index.

Notes

K_IntStop aborts the interrupt operation defined by the frame identified by *frameHandle*. On return, *status* contains a code that indicates what the status was when the function was called; *index* contains the number of the next buffer element, at the time the function was called, which is to be written with the next sample.

K_IntStop does nothing if an interrupt-mode A/D operation is not in progress.

Purpose

Transfer acquired A/D samples between a memory buffer and an array.

Prototype

C

DASErr far pascal K_MoveDataBuf(int far * *dest*, int far * *source*, unsigned int *samples*);

Pascal

Function K_MoveDataBuf(*dest* : Longint; *source* : Longint; *samples* : Word) : Integer;

Visual Basic for Windows

K_MoveDataBuf Lib "DASTC.dll" (*dest* As Any, *source* As Any, ByVal *samples* As Integer) As Integer

Parameters

dest	Address of destination buffer
source	Address of source buffer
samples	Number of samples to transfer

Notes

K_MoveDataBuf moves *samples* samples from the buffer at *source* to the buffer at *dest*.

Although this function is valid for all of the supported languages, it is intended primarily for use with those languages (such as Visual Basic) that do not provide a convenient method of accessing memory directly. This function is also needed in languages that are running in a Windows standard environment, where acquired samples must be initially written into a dynamically allocated buffer before the data can be stored in a local buffer.

Purpose

Convert a Function Call Driver Channel-Gain array into an equivalent Visual Basic Channel-Gain array (Visual Basic Only).

Prototype

Visual Basic for Windows

K_RestoreChnGAry Lib "DASTC.dll" (*chanGainArray* As Integer) As Integer

Parameters

chanGainArray Storage location for Channel-Gain array

Notes

Use this function to restore the Channel-Gain Array in a format readable to Visual Basic.

Do not call this function until a **K_SyncStart** or **K_IntStart** has been called.

Purpose

Set the values of an A/D frame's Buffer Address and Number of Samples elements (Pascal and C languages only).

Prototype

C

DASErr far pascal K_SetBuf(FRAMEH *frameHandle*, void far * *bufAddr*, long *samples*);

Pascal

Function K_SetBuf(*frameHandle* : Longint; *bufAddr* : Longint; *samples* : Longint) : Word;

Parameters

frameHandle	Handle to A/D frame
bufAddr	Buffer Address
samples	Number of Samples (**1-65,535**)

Notes

K_SetBuf assigns values to the following elements in the frame identified by *frameHandle*:

- the Buffer Address element is assigned the value in *bufAddr*
- the Number of Samples element is assigned the value in *samples*

If using Visual Basic for Windows, *bufAddr* must be the address of a dynamically allocated buffer obtained from **K_IntAlloc**. For user-defined arrays, see **K_SetBufL** if integer type is configured, and **K_SetBufR** if floating-point type is configured.

Purpose

Set the values of a frame's Buffer Address and Number of Samples elements for user-defined long integer arrays (Visual Basic for Windows only).

Prototype

Visual Basic for Windows

K_SetBufL Lib "DAS1600.dll" (ByVal *frameHandle* As Long, *bufAddr* As Long, ByVal *samples* As Long) As Integer

Parameters

frameHandle	Frame handle
bufAddr	Address of user-created buffer defined as a long array
samples	Number of samples to be stored in buffer

Notes

K_SetBufL sets the Buffer Address to *bufAddr* and the Number of Samples to *samples* in the frame identified by *frameHandle*.

Purpose

Set the values of a frame's Buffer Address and Number of Samples elements for user-defined floating-point arrays (Visual Basic for Windows only).

Prototype

Visual Basic for Windows

K_SetBufR Lib "DAS1600.dll" (ByVal *frameHandle* As Long, *bufAddr* As Single, ByVal *samples* As Long) As Integer

Parameters

frameHandle	Frame handle
bufAddr	Address of user-created buffer defined as a long array
samples	Number of samples to be stored in buffer

Notes

K_SetBufR sets the Buffer Address to *bufAddr* and the Number of Samples to *samples* in the frame identified by *frameHandle*.

Purpose

Set the value of a frame's Channel-Gain Array Address element.

Prototype

C

DASErr far pascal K_SetChnGAry(FRAMEH *frameHandle*,
void far * *chanGainArray*);

Pascal

Function K_SetChnGAry(*frameHandle* : Longint;
Var *chanGainArray* : Integer) : Word;

Visual Basic for Windows

K_SetChnGAry Lib "DASTC.dll" (ByVal *frameHandle* As Long,
chanGainArray As Integer) As Integer

Parameters

frameHandle Handle to A/D frame

chanGainArray Channel-Gain Array Address

Notes

K_SetChnGAry sets the value of the Channel-Gain Array Address element to *chanGainArray* in the frame identified by frameHandle.

A Channel-Gain Array defines two characteristics of an A/D operation:

- the sequence in which the input channels are sampled and,
- the gain applied to each of the channels configured for voltage in that sequence.

A Channel-Gain Array can define up to 16 randomly sequenced channel-gain pairs. Adjacent pairs can specify the same channel (with equal or unequal gains). The following table illustrates the required format of a Channel-Gain array for the C and Pascal languages.

Byte	0	1	2	3	4	5	...	2N	2N+1
Value	N		chan	gain	chan	gain	...	chan	gain
	# of pairs		pair 1		pair 2		...	pair N	

The following table illustrates the required format of a Channel-Gain array for Visual Basic.

Integer	0	1	2	3	4	...	2N-1	2N
Value	N	chan	gain	chan	gain	...	chan	gain
	# of pairs	pair 1		pair 2		...	pair N	

The gain must be specified as a gain code. Refer to Table 1 on page 47 for the input range affected by each gain.

Gain Code	0	1	2	3
Gain	1	125	166.67	400

A Channel-Gain array enables you to specify different gains for different input channels.

Purpose

Set the value of a frame's Buffering Mode element to CONTINUOUS.

Prototype

C

DASErr far pascal K_SetContRun(FRAMEH *frameHandle*, short);

Pascal

Function K_SetContRun(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_SetContRun Lib "DASTC.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Handle to A/D frame

Notes

K_SetContRun sets the Buffering Mode to CONTINUOUS in the frame identified by *frameHandle*.

The default setting for *buffering mode* is SINGLE-CYCLE

Purpose

Set the values of an A/D frame's Start Channel and Stop Channel elements.

Prototype

C

DASErr far pascal K_SetStartStopChn(FRAMEH *frameHandle*, short *start*, short *stop*);

Pascal

Function K_SetStartStopChn(*frameHandle* : Longint; *start* : Word; *stop* : Word) : Word;

Visual Basic for Windows

K_SetStartStopChn Lib "DASTC.dll" (ByVal *frameHandle* As Long, ByVal *start* As Integer, ByVal *stop* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
start	Start Channel. Valid values: **0**, **1**,...,**15**
stop	Stop Channel. Valid values: **0**, **1**,...,**15**

Notes

K_SetStartStopChn assigns values to the following elements in the frame identified by *frameHandle*:

- the Start Channel element is assigned the value in *start*
- the Stop Channel element is assigned the value in *stop*

During a Start/Stop scan, the gains applied are either the internal defaults or those read from the configuration file at load time.

Use **K_SetChnGAry** to specify a non-sequential channel-scanning sequence and/or to specify channel gains.

If the Stop channel number is greater than the Start channel number, then the scan order is

Start channel number,., ... ,Stop channel number

For example, if Stop=13 and Start=10, the scan order is 10,11,12, and 13.

If the Stop channel number is less than the Start channel number, then scan order is
Start channel number,., ... ,15,0, ... ,Stop channel number

For example, if Start=13 and Stop=10, the scan order is 13-15 (inclusive), then 0-10 (inclusive).

If the Start and Stop channel numbers are the same, a single scan is performed.

Purpose

Start a synchronous-mode A/D operation.

Prototype

C

DASErr far pascal K_SyncStart(FRAMEH *frameHandle*);

Pascal

Function K_SyncStart(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_SyncStart Lib "DASTC.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Handle to A/D frame

Notes

K_SyncStart starts the synchronous-mode A/D operation defined in the frame identified by *framehandle*. An error is returned if an Interrupt operation is in progress.

The values acquired are in microvolts or .01 degrees for integer types, and are not scaled for floating point.

Function Call Driver Error Messages

A

A.1 Error Codes

Error 0000h **No error.**

Error 6000h **Error In Configuration File**

Cause The configuration file supplied to DASTC_DevOpen() is corrupt or does not exist. If file is known to be good, then it probably contains one or more undefined keywords.

Solution Check if the file exists at the specified path. Check for illegal keywords in file; the best way to fix illegal keywords is to let the supplied DASTCCFG.EXE utility do it.

Error 6001h **Illegal Base Address In Configuration File.**

Error 6002h **Illegal IRQ level In Configuration File.**

Error 6004h **Error opening configuration file.**

Error 6005h **Illegal Channel Number**

Cause The specified I/O operation channel is out of range. The legal range is **0-15**.

Solution Specify legal channel number.

Error 6006h **Illegal gain.**

Error 6008h **Bad number in configuration file.**

Cause An illegal specification of a number is detected in the Configuration file. Note that if specifying a hexadecimal number for the Base Address, that number must be proceeded with '&H'.

Solution Check the number following 'Address' in the Configuration file.

Error 6009h **Incorrect version number.**

Error 600Ah **Configuration file not found.**

Cause This error is returned by the DASTC_DevOpen() function whenever the specified configuration file is not found.

Solution Check the configuration file name (spelling!), path, etc...

Error 600Ch **Error returning INT buffer.**

Cause This error occurs during K_IntFree() whenever DOS returns an error in INT 21h function 49H.

Solution Make sure that the parameter passed to K_INTFree() was previously obtained via K_INTAlloc().

Error 600Dh **Bad frame handle.**

Cause This error is usually returned by Frame Management or an Operation Function whenever an illegal Frame handle is passed to one of these functions.

Solution Check the Frame Handle.

Error 600Eh **No more frame handles.**

Error 600Fh **Requested INT buffer too large.**

Error 6010h **Cannot allocate INT buffer.**

Error 6011h **INT buffer already allocated.**

Error 6012h **INT buffer De-Allocation Error.**

Error 6013h **INT buffer never allocated.**

Error 7000h **No board name**

Cause DASTC_DevOpen() function did not find the keyword 'Name'or a name following in the specified configuration file.

Solution Make sure that a board name is specified in your configuration file. The legal DAS-TC name is: DASTC.

Error 7001h **Bad board name**

Cause DASTC_DevOpen() function found the board 'name' in the specified configuration file to be illegal. The legal DASTC name is: DASTC.

Solution Check the name following keyword 'Name' in your configuration file.

Error 7002h **Bad board number**

Cause DASTC_DevOpen() function found the 'Board' number in the specified configuration file to be illegal. The legal board numbers are 0 and 1.

Solution Check the number following 'Board' in your configuration file.

Error 7003h **Bad base address**

Cause DASTC_DevOpen() function found the board's base I/O 'Address' in the specified configuration file to be illegal. The legal address are 200h (512) through 3F0h (1008) in increments of 10h (16) inclusive.

Solution Check the number following 'Address' in your configuration file. NOTE that to specify a Hex number, the number must be preceded by '&H'.

Error 7004h **Bad Interrupt Level.**

Cause DASTC_DevOpen() function found the Interrupt Level in the specified configuration file to be illegal. The legal Interrupt levels are 2, 3, 4, 5, and 7.

Solution Check the number following 'IntLevel' in your configuration file.

Error 7005h **Bad Normal Mode Rejection Frequency.**

Error 7006h **Bad Number Type.**

Error 7007h **Bad Channel Configuration.**

Cause One or more of these conditions exists:

- Channel # is out of range
- Channel argument is illegal

Error 7008h **Check Sum Error.**

Cause Checksum in communication packet failed, resulting in a communication failure.

Error 7009h **Board Not Initialized.**

Cause One or more of the following conditions exists:

- A function was called before K_DASDevInit was called.
- The PC Side Board diagnostics done during board initialization failed.
- Attempt to return the DAS-TC ID failed.
- Wrong Base Address.

Error 700Ah **Initialization Failure.**

Error 700Bh **Protocol Communication Error.**

Error 700Ch **Bad Voltage to Temperature Calculation Error.**

Error 8000h **No error.**

Error 8001h **Function not supported**

Cause A request is made to a function that is not supported by a DAS-TC. This error should not occur in a standard release software.

Solution Insure that the function is one listed in chapter 3. If the problem cannot be resolved, contact the Keithley Technical Support Department.

Error 8002h **Function out of bounds**

Cause Illegal function number is specified. This error should not occur in a standard release software.

Solution Contact the Keithley Technical Support Department.

Error 8003h **Illegal board number**

Cause The driver supports up to two boards: 0 and 1.

Solution Check the board number parameter in your call to DASTC_ GetDevHandle().

Error 8005h **No board**

Cause This error is issued during K_DASDevInit() whenever the board presence test fails. This is normally caused by a conflict in the specified board I/O address and the actual I/O address the board is configured for. Also, this error is issued when the board is not present in the system.

Solution Check the board's base I/O address dip switch and make sure it matches the base address in your configuration file.

Error 8006h **A/D not Initialized**

Cause A function was called before K_DASDevInit was called.

Error 801Ah **Interrupts Already Active.**

Cause An attempt is made to start an Interrupt-based operation while another is already active.

Solution Stop current Interrupt mode first and retry.

A.2 Error Conditions

Voltage/Thermocouple Error Conditions

When a voltage/thermocouple input is under or over the voltage range set for a particular channel, the DAS-TC responds with the following readouts.

For under the voltage/thermocouple range:

- Floating Point is -10,000.00
- Integer is -971,227,136

For over the voltage/thermocouple range:

- Floating Point is +10,000.00
- Integer is +1,176,256,512

KEITHLEY DATA ACQUISITION

440 Myles Standish Boulevard
Taunton, MA 02780
508-880-3000