DAS-Scan
Function Call Driver

USER'S GUIDE

DAS-Scan
Function Call Driver
User's Guide

Revision A – July 1996
Part Number: 94890

New Contact Information

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

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

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

Keithley MetraByte Division

Keithley Instruments, Inc.

440 Myles Standish Blvd. Taunton, MA 02780

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

Preface

The *DAS-Scan Function Call Driver User's Guide* provides information to help you write application programs for the DAS-Scan system using the DAS-Scan Function Call Driver. The DAS-Scan Function Call Driver supports the following Windows™-based languages:

- Microsoft® Visual C++™ (up to Version 1.52)
- Borland® C/C++ (Version 4.0 and 4.5)
- Microsoft Visual Basic® for Windows (Version 3.0 and Version 4.0)

The manual is intended for application programmers using a DAS-Scan system with an IBM® PC AT® or compatible computer. It is assumed that you have read the *DAS-Scan User's Guide* to familiarize yourself with the system's features, and that you have completed the appropriate hardware installation and configuration. It is also assumed that you are experienced in programming in your selected language and that you are familiar with data acquisition principles.

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

- Chapter 1 contains the information needed to install the DAS-Scan Function Call Driver, a summary of the DAS-Scan Function Call Driver functions typically used when programming a DAS-Scan system, a series of flow diagrams illustrating the procedures typically used when programming a DAS-Scan system, and information on how to get help.
- Chapter 2 contains conceptual information about the DAS-Scan Function Call Driver functions typically used when programming a DAS-Scan system.

- Chapter 3 contains conceptual information about additional DAS-Scan Function Call Driver functions you can use when programming a DAS-Scan system.
- Appendix A contains instructions for converting counts to voltage and for converting counts to temperature.

An index completes this manual.

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

- References to Windows apply to Windows 3.1, Windows 3.11 for Workgroups, and Windows 95. When a feature applies to a specific Windows version, the complete version name is used.
- Keyboard keys are represented in bold.

Table of Contents

List of Figures

List of Tables

1

Getting Started

This chapter contains the following sections:

- **Overview** - a description of the DAS-Scan Function Call Driver.
- **Setup and Installation** - a list of the tasks you should perform before using the DAS-Scan Function Call Driver.
- **Summary of Functions** - a brief description of the DAS-Scan Function Call Driver functions that are typically used when programming a DAS-Scan system.
- **Programming Flow Diagrams** - an illustration of the procedures you will typically follow when programming a DAS-Scan system using the DAS-Scan Function Call Driver.
- **Getting Help** - information on how to get help when installing or using the DAS-Scan Function Call Driver.

Overview

The DAS-Scan Function Call Driver is a library of data acquisition and control functions that is part of the ASO-SCAN software package. The ASO-SCAN software package includes the following:

- Dynamic Link Libraries (DLLs) of Function Call Driver functions for Microsoft Visual C++, Borland C/C++, and Microsoft Visual Basic for Windows.
- Support files, containing program elements, such as function prototypes and definitions of variable types, that are required by the Function Call Driver functions.
- Utility programs that allow you to configure, calibrate, and test the functions of the DAS-Scan system.
- Language-specific example programs.

Setup and Installation

Before you use the DAS-Scan Function Call Driver to program your DAS-Scan system, make sure that you have performed the following tasks. Refer to the *DAS-Scan User's Guide* for more information.

1. Unpack and inspect the components of your DAS-Scan system.
2. Create a channel map indicating the input signals you want to attach to each of the channels in your DAS-Scan system.
3. Configure the hardware components, as follows:
   - On the SCAN-AD-HR board, set the base I/O address switch (S1) to indicate the appropriate base I/O address, and, if appropriate, set the J6 jumper to indicate an external pacer clock or an external digital trigger/hardware gate.
   - On each SCAN-BRD assembly, set the address thumbwheels to indicate the appropriate assembly address.
   - If you are using SCAN-STP-TC screw terminal panels, set the J2 jumpers on each panel to indicate whether you are using cold junction compensation (CJC).

4. Mount the SCAN-STP screw terminal panels on DIN rails.
5. Attach the input signals to the appropriate SCAN-STP screw terminal panels.
6. Mount the SCAN-CH chassis in a 19-inch rack.
7. Attach one end of each SCAN-CAB-64 cable to the appropriate position in the SCAN-CH chassis. (Do not attach the other end of the SCAN-CAB-64 cables at this point.)
8. Attach a power cord to the power-supply panel of each SCAN-CH chassis.
9. Check the operation of the power supply in each SCAN-CH chassis.
10. Install the SCAN-BRD assemblies in the appropriate SCAN-CH chassis.
11. Make sure that power to the computer is turned OFF.
12. Install the SCAN-AD-HR board in your computer.
13. Attach the SCAN-AD-HR board to your first SCAN-CH chassis using the S-1802/M cable.
14. Daisy-chain multiple SCAN-CH chassis, if required, using S-1802/MM cables.
15. Attach the other end of each SCAN-CAB-64 cable to the appropriate SCAN-STP screw terminal panel.
16. Connect an external pacer clock, external digital trigger, or hardware gate, if required, to the J5 connector on the SCAN-AD-HR board using a CAB-SMA-BNC cable.
17. Plug the power cords into the wall, and then power up your computer, the SCAN-CH chassis, and any equipment attached to the SCAN-STP screw terminal panels.
18. Install the ASO-SCAN software package.
19. Specify the configuration options for your DAS-Scan system in the DAS-Scan Configuration Utility.
20. Test the functions of the DAS-Scan system using the DAS-Scan Control Panel.
21. Look at the example programs provided with the ASO-SCAN software package. Refer to the FILES.TXT file in the installation directory for a list and description of the example programs.

Summary of Functions

Table 1-1 describes the functions in the DAS-Scan Function Call Driver that are typically used to program a DAS-Scan system. For more detailed information about the functions, refer to Chapter 2 of this manual and to the DAS-Scan Function Call Driver online help file (SCANFCD.HLP).

Table 1-1. Summary of Functions

Type of Function	Name of Function	Description
Initialization	K_OpenDriver	Initializes a Function Call Driver.
	K_CloseDriver	Closes a Function Call Driver.
	K_GetDevHandle	Initializes a virtual board.
	K_FreeDevHandle	Frees a device handle.
	K_DASDevInit	Reinitializes a virtual board.
Operation	K_ADRead	Reads a single analog input value.
	K_DMAStart	Starts a DMA-mode operation.
	K_DMAStatus	Gets the status of a DMA-mode operation.
	K_DMAStop	Stops a DMA-mode operation.
	DASSCAN_EventEnable	Enables the generation of a Windows event (C/C++).
	DASSCAN_EventDisable	Disables the generation of a Windows event (C/C++).
Frame management	K_GetADFrame	Accesses a frame for an analog input operation.
	K_FreeFrame	Frees a frame.
	K_ClearFrame	Sets all frame elements to their default values.

Table 1-1. Summary of Functions (cont.)

Type of Function	Name of Function	Description
Memory management	K_DMAAlloc	Dynamically allocates a memory buffer for a DMA-mode operation.
	K_DMAFree	Frees a memory buffer that was dynamically allocated for a DMA-mode operation.
	K_MoveBufToArray	Transfers data from a dynamically allocated memory buffer to a local array (Visual Basic for Windows).
Buffer address	K_SetDMABuf	Specifies the address of a dynamically allocated memory buffer.
Channel and gain	K_SetChn	Specifies a single logical channel.
	K_SetStartStopChn	Specifies the first and last logical channels in a group of consecutive logical channels.
	K_SetG	Specifies the gain for a group of consecutive logical channels.
	K_SetStartStopG	Specifies the first and last logical channels in a group of consecutive logical channels and the gain for all channels in the group.
	K_SetChnGAry	Specifies the starting address of a channel-gain queue.
	K_FormatChnGAry	Converts the format of a channel-gain queue (Visual Basic for Windows).
	K_RestoreChnGAry	Restores a converted channel-gain queue (Visual Basic for Windows).
	K_SetADMode	Specifies the input range type (bipolar or unipolar).
	K_GetADMode	Gets the input range type (bipolar or unipolar).
Clock	K_SetClk	Specifies the pacer clock source.
	K_SetClkRate	Specifies the clock rate for the internal pacer clock.
	K_GetClkRate	Gets the clock rate for the internal pacer clock.
	K_SetExtClkEdge	Specifies the active edge of an external pacer clock.

Table 1-1. Summary of Functions (cont.)

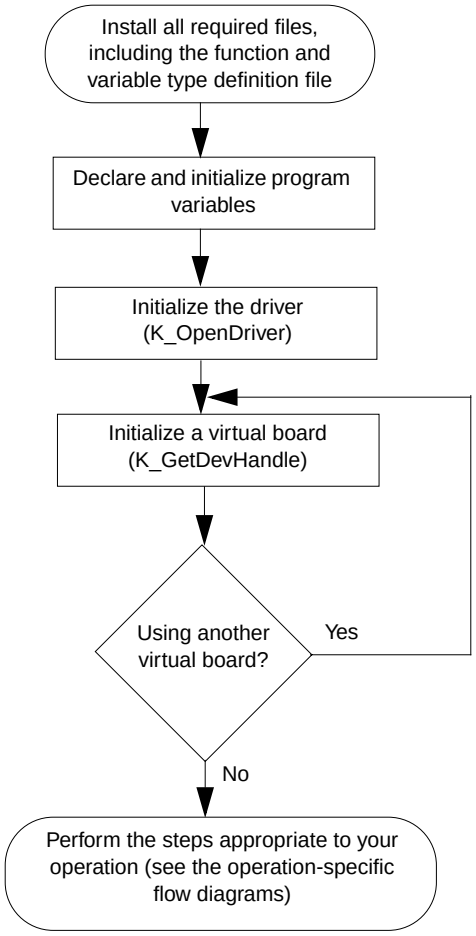
Type of Function	Name of Function	Description
Trigger	K_SetTrig	Specifies the trigger source.
	K_SetDITrig	Sets up a digital start trigger.
Gate	K_SetGate	Specifies the status of a hardware gate.
Miscellaneous	K_GetErrMsg	Gets the address of an error message string (C/C++).
	K_GetVer	Gets revision numbers.
	K_GetShellVer	Gets the current DAS shell version.

Programming Flow Diagrams

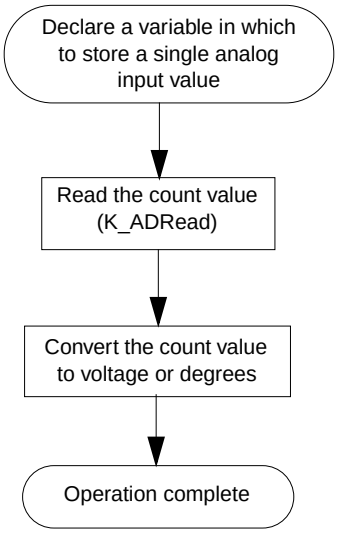
This section contains a series of programming flow diagrams illustrating the procedures you will typically follow when programming a DAS-Scan system using the DAS-Scan Function Call Driver. For more detailed information about the programming procedures, refer to the DAS-Scan Function Call Driver online help file (SCANFCD.HLP).

Although error checking is not shown in the flow diagrams, it is recommended that you check the error/status code returned by each function used in your application program.

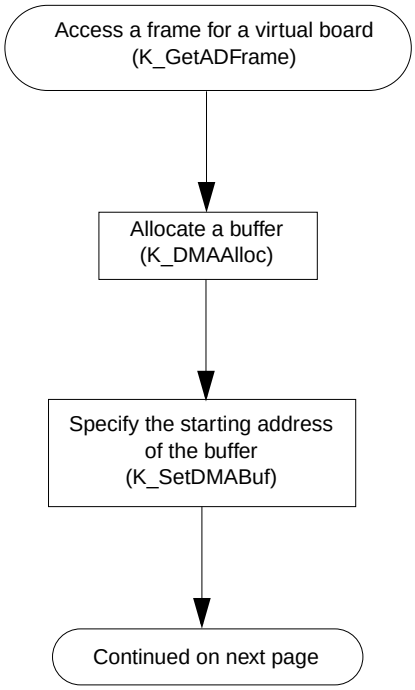
Preliminary Steps for All Analog Input Operations



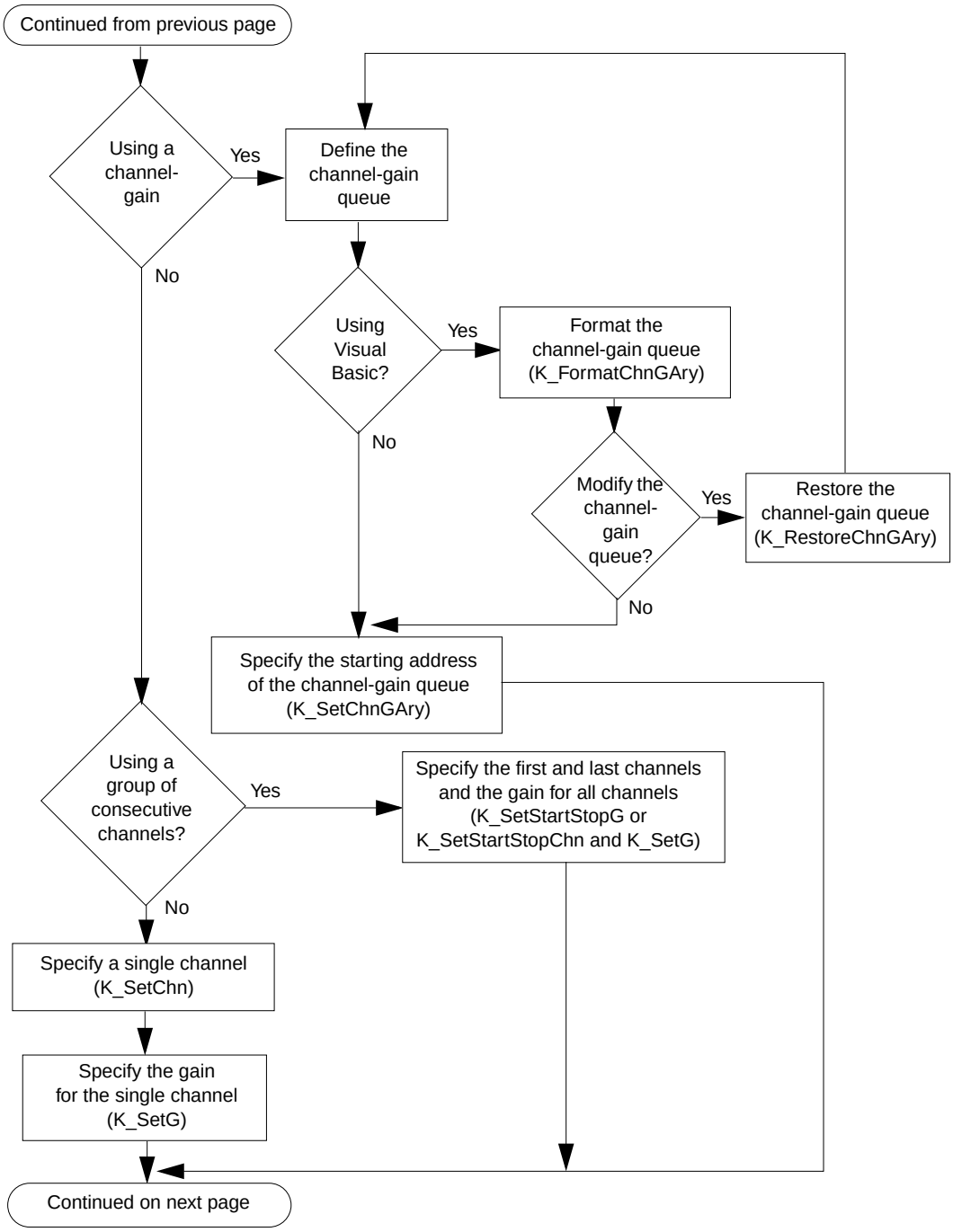
Steps for a Single-Mode Analog Input Operation



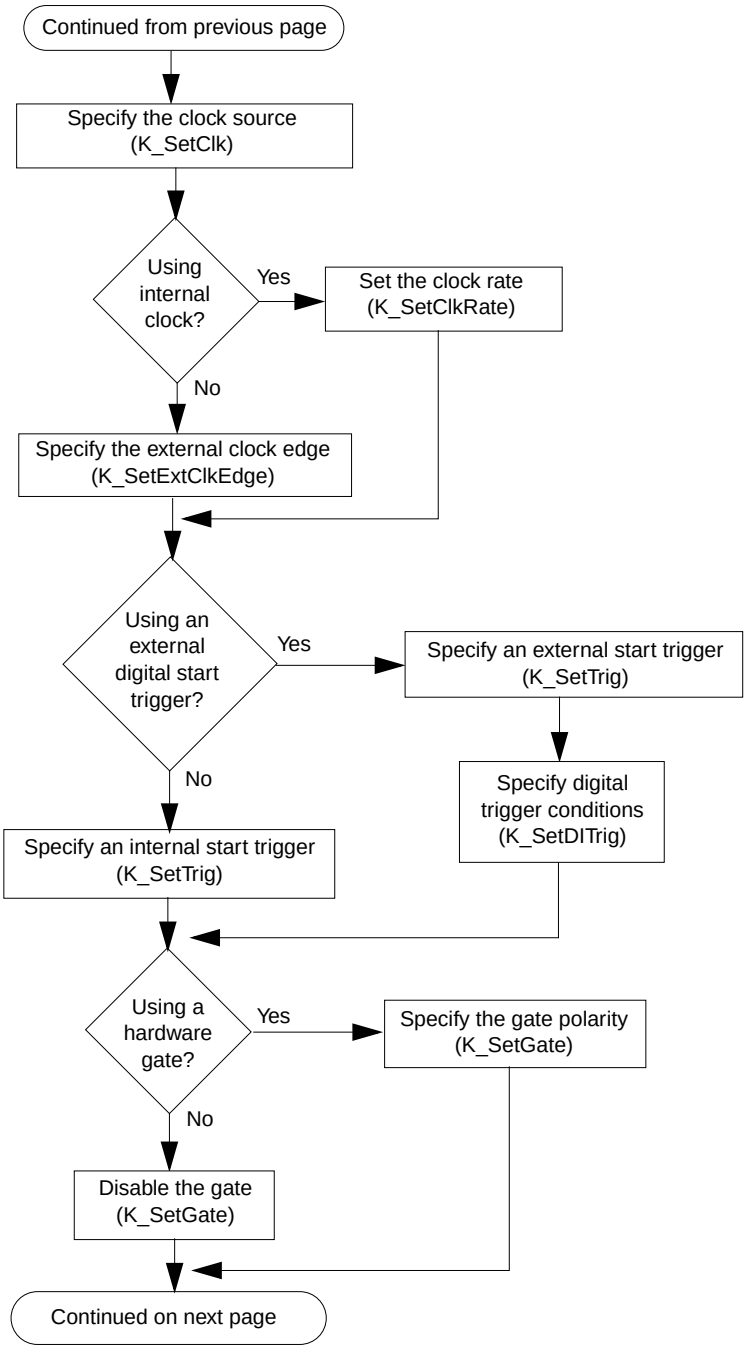
Steps for a DMA-Mode Analog Input Operation



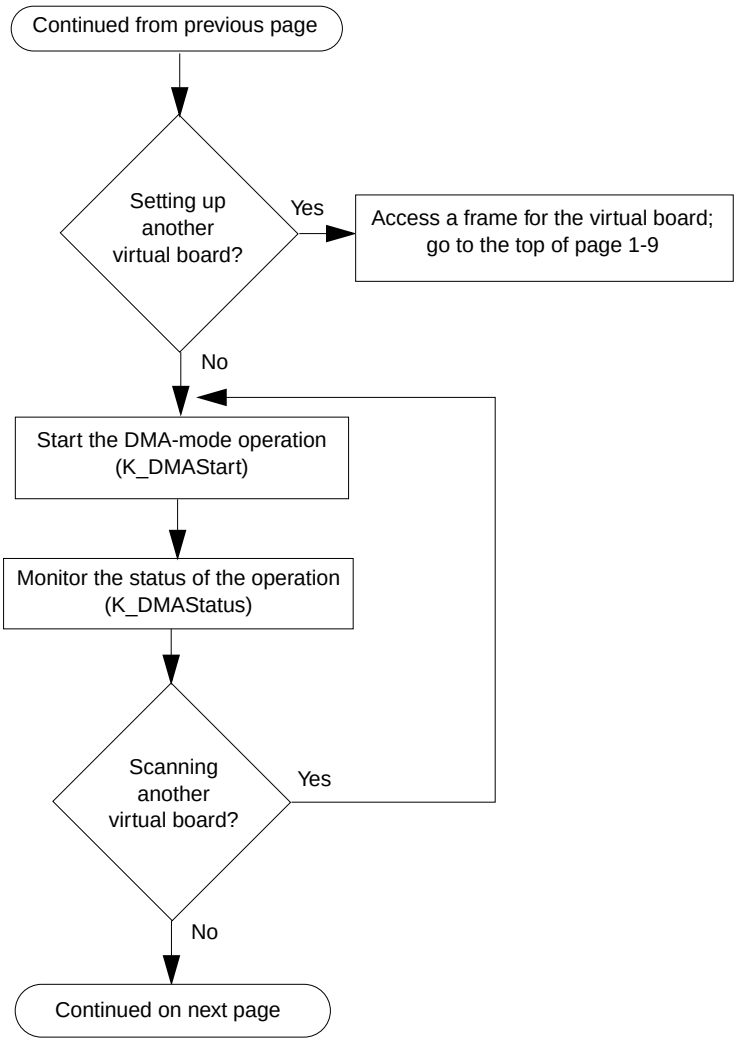
Steps for a DMA-Mode Analog Input Operation (cont.)



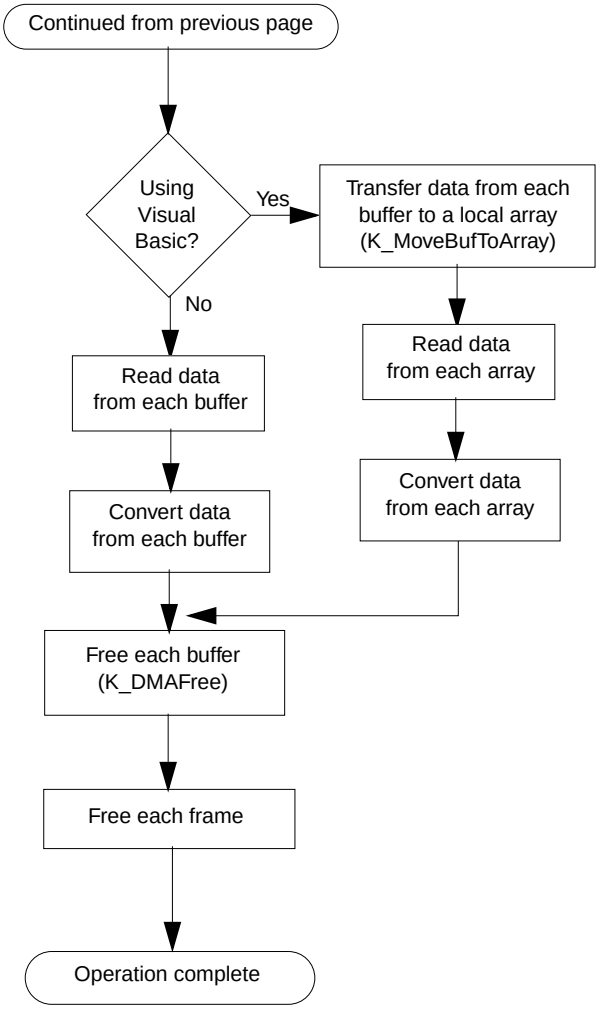
Steps for a DMA-Mode Analog Input Operation (cont.)



Steps for a DMA-Mode Analog Input Operation (cont.)



Steps for a DMA-Mode Analog Input Operation (cont.)



Getting Help

If you need help installing or using the DAS-Scan Function Call Driver, call your local sales office or call the following number for technical support:

(508) 880-3000

Monday - Friday, 8:00 A.M. - 6:00 P.M., Eastern Time

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

SCAN-AD-HR board configuration	Serial #	_____
	Revision code	_____
	Base address setting	_____
	Interrupt level setting	_____
	DMA channel(s)	_____
	External clock, trigger, or gate	_____
SCAN-BRD assembly configuration	Model	_____
	Serial #	_____
	Revision code	_____
	Address setting (0h to 3Fh)	_____
	Unipolar or bipolar mode	_____
SCAN-BRD assembly configuration	Model	_____
	Serial #	_____
	Revision code	_____
	Address setting (0h to 3Fh)	_____
	Unipolar or bipolar mode	_____
SCAN-BRD assembly configuration	Model	_____
	Serial #	_____
	Revision code	_____
	Address setting (0h to 3Fh)	_____
	Unipolar or bipolar mode	_____

SCAN-BRD assembly configuration

Model	________________
Serial #	________________
Revision code	________________
Address setting (0h to 3Fh)	________________
Unipolar or bipolar mode	________________

SCAN-BRD assembly configuration

Model	________________
Serial #	________________
Revision code	________________
Address setting (0h to 3Fh)	________________
Unipolar or bipolar mode	________________

SCAN-BRD assembly configuration

Model	________________
Serial #	________________
Revision code	________________
Address setting (0h to 3Fh)	________________
Unipolar or bipolar mode	________________

SCAN-BRD assembly configuration

Model	________________
Serial #	________________
Revision code	________________
Address setting (0h to 3Fh)	________________
Unipolar or bipolar mode	________________

SCAN-BRD assembly configuration

Model	________________
Serial #	________________
Revision code	________________
Address setting (0h to 3Fh)	________________
Unipolar or bipolar mode	________________

SCAN-BRD assembly configuration

Model	________________
Serial #	________________
Revision code	________________
Address setting (0h to 3Fh)	________________
Unipolar or bipolar mode	________________

SCAN-BRD assembly configuration

Model	________________
Serial #	________________
Revision code	________________
Address setting (0h to 3Fh)	________________
Unipolar or bipolar mode	________________

SCAN-BRD assembly configuration

Model	________________
Serial #	________________
Revision code	________________
Address setting (0h to 3Fh)	________________
Unipolar or bipolar mode	________________

SCAN-BRD assembly configuration	Model	____________
	Serial #	____________
	Revision code	____________
	Address setting (0h to 3Fh)	____________
	Unipolar or bipolar mode	____________
SCAN-BRD assembly configuration	Model	____________
	Serial #	____________
	Revision code	____________
	Address setting (0h to 3Fh)	____________
	Unipolar or bipolar mode	____________
SCAN-BRD assembly configuration	Model	____________
	Serial #	____________
	Revision code	____________
	Address setting (0h to 3Fh)	____________
	Unipolar or bipolar mode	____________
SCAN-BRD assembly configuration	Model	____________
	Serial #	____________
	Revision code	____________
	Address setting (0h to 3Fh)	____________
	Unipolar or bipolar mode	____________
SCAN-BRD assembly configuration	Model	____________
	Serial #	____________
	Revision code	____________
	Address setting (0h to 3Fh)	____________
	Unipolar or bipolar mode	____________
Computer	Manufacturer	____________
	CPU type	____________
	Clock speed (MHz)	____________
	KB of RAM	____________
	Video system	____________
	BIOS type	____________
Operating system	Windows version	____________
Compiler	Language	____________
	Manufacturer	____________
	Version	____________

2

Available Operations

This chapter contains conceptual information about the DAS-Scan Function Call Driver functions typically used when programming a DAS-Scan system. It includes the following sections:

- **System Operations** - information on initializing the Function Call Driver, initializing a virtual board, generating a Windows event, retrieving revision levels, and handling errors.
- **Analog Input Operations** - information on operation modes, accessing a frame, memory allocation and management, gains, channels, the pacer clock, and triggers.

System Operations

This section describes the miscellaneous operations and general maintenance operations that apply to the entire DAS-Scan system and to the DAS-Scan Function Call Driver. It includes information on initializing the Function Call Driver, initializing a virtual board, generating a Windows event, retrieving revision levels, and handling errors.

Initializing the Driver

You must initialize the DAS-Scan Function Call Driver and any other Keithley DAS Function Call Drivers you are using in your application program. To initialize the drivers, use the **K_OpenDriver** function. Specify the driver you are using and the configuration file that defines the use of the driver. The driver returns a unique identifier for the driver; this identifier is called the driver handle.

You can specify a maximum of 30 driver handles for all the Keithley MetraByte drivers initialized from all your application programs. If you no longer require a driver and you want to free some memory or if you have used all 30 driver handles, use the **K_CloseDriver** function to free a driver handle and close the associated driver.

If the driver handle you free is the last driver handle specified for a Function Call Driver, the driver is shut down. The DLLs associated with the Function Call Driver are shut down and unloaded from memory.

Initializing a Board

Each DAS-Scan system contains one physical SCAN-AD-HR board that you plug into your computer. You can attach up to 64 physical SCAN-BRD assemblies to the SCAN-AD-HR. Each SCAN-BRD assembly contains 64 physical channels.

The DAS-Scan Function Call Driver treats each group of four SCAN-BRD assemblies as a virtual board. Each virtual board contains 256 logical channels (four SCAN-BRD assemblies multiplied by 64 physical channels on each). Refer to page 2-12 for information on how the logical channels map to the virtual board.

The driver supports 16 virtual boards for a total of 4,096 logical channels (16 virtual boards multiplied by 256 logical channels on each). You must use the **K_GetDevHandle** function to initialize each virtual board you want to use. Specify the driver handle and the virtual board number (0 to 15). **K_GetDevHandle** verifies that the SCAN-AD-HR board is present at the base address specified in the configuration file, stops any operations currently in progress, initializes the SCAN-AD-HR board to its default state, and calibrates the analog-to-digital converter on the SCAN-AD-HR board.

K_GetDevHandle returns a unique identifier for each virtual board; this identifier is called the device handle. Device handles allow you to communicate with more than one virtual board. Use the device handle returned by **K_GetDevHandle** in subsequent function calls related to the virtual board.

You can specify a maximum of 30 device handles for all the Keithley DAS products accessed from all your application programs. If you are no longer using a Keithley DAS product and you want to free some memory

or if you have used all 30 device handles, use the **K_FreeDevHandle** function to free a device handle.

Use **K_GetDevHandle** the first time you initialize a board only. To reinitialize a virtual board after you have called **K_GetDevHandle**, use the **K_DASDevInit** function.

Generating a Windows Event

If you are performing a DMA-mode operation and programming in C/C++, you can program your DAS-Scan system to generate an interrupt when the operation stops (either because the specified number of samples was read or because an error, such as an overflow, occurred). You can then use the interrupt to generate a Windows event.

By default, the generation of Windows events is disabled. Use the **DASSCAN_EventEnable** function to enable the generation of Windows events. When an interrupt is generated, a Windows event is generated and a message is sent to your application program. Your application program can then take the appropriate action. Use the **DASSCAN_EventDisable** function to disable the generation of Windows events.

Notes: Always disable the generation of Windows events (using **DASSCAN_EventDisable**) before closing your application program.

The DAS-Scan Function Call Driver does not support the generation of Windows events when programming in Visual Basic for Windows.

If a Windows event is generated, the *wParam* parameter of the message structure specifies the status of the operation as returned in the *pStatus* variable of **K_DMAStatus**. Refer to the DAS-Scan Function Call Driver online help file (SCANFCD.HLP) for more information about **K_DMAStatus**.

You can generate a Windows event only if you are performing an operation in single-cycle buffering mode (the default buffering mode). If you change the buffering mode to continuous mode, the generation of Windows events is no longer supported. Refer to page 3-23 for information on when you can use continuous buffering mode.

Retrieving Revision Levels

If you are using functions from different Keithley DAS Function Call Drivers in the same application program or if you are having problems with your application program, you may want to verify which versions of the Function Call Driver, Keithley DAS Driver Specification, and Keithley DAS Shell are used by the DAS-Scan system.

The **K_GetVer** function allows you to get both the revision number of the DAS-Scan Function Call Driver and the revision number of the Keithley DAS Driver Specification to which the driver conforms.

The **K_GetShellVer** function allows you to get the revision number of the Keithley DAS Shell (the Keithley DAS Shell is a group of functions that are shared by all Keithley DAS products).

Handling Errors

Each FCD function returns a code indicating the status of the function. To ensure that your application program runs successfully, it is recommended that you check the returned code after the execution of each function. If the status code equals 0, the function executed successfully and your program can proceed. If the status code does not equal 0, an error occurred; ensure that your application program takes the appropriate action. Refer to the DAS-Scan Function Call Driver online help file (SCANFCD.HLP) for a complete list of error codes.

Each supported programming language uses a different procedure for error checking. Refer to the DAS-Scan Function Call Driver online help file (SCANFCD.HLP) for language-specific information.

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

Analog Input Operations

This section describes analog input operations. It includes information on the operation modes available, how to access a frame, how to allocate and manage memory, and how to specify channels and gains, the pacer clock source, and the trigger source for an analog input operation.

Operation Modes

The operation mode determines which attributes you can specify for an analog input operation and how data is transferred from the SCAN-AD-HR board to computer memory. You can perform analog input operations in single mode or DMA mode, as described in the following sections.

Single Mode

In single mode, the SCAN-AD-HR board acquires a single sample from a single logical channel. The driver initiates the conversion; you cannot perform any other operation until the single-mode operation is complete.

Use the **K_ADRead** function to perform an analog input operation in single mode. You specify the virtual board that the logical channel is associated with, the logical channel, the gain at which you want to read the signal, and the variable in which to store the converted data.

The data in the variable is stored as a count value. Refer to Appendix A for information on converting the count value to voltage or degrees.

DMA Mode

DMA mode is the recommended way to transfer data. In DMA mode, the SCAN-AD-HR board acquires a single sample or multiple samples from one or more logical channels (up to a maximum of 256) on the same virtual board. A hardware clock initiates conversions. Once the analog input operation begins, control returns to your application program. The hardware temporarily stores the acquired data in the FIFO (first-in, first-out data buffer) on the SCAN-AD-HR board and then transfers the data to a user-defined DMA buffer in the computer.

Use the **K_DMAStart** function to start the DMA-mode operation on the first virtual board, specifying the handle of the frame that defines the first operation. After the SCAN-AD-HR board reads a sample from each logical channel associated with the virtual board and stores the samples in the user-defined buffer, the operation stops. (Use the **K_DMAStatus** function to monitor the status of the operation and determine when the operation stops.) Then, use **K_DMAStart** again to start the DMA-mode operation on the next virtual board, specifying the handle of the frame that defines the next operation.

Refer to page 1-9 for the procedure to follow when performing a DMA-mode operation. In addition, example programs 1 and 3, provided in the ASO-SCAN software package, illustrate how to acquire data in DMA mode.

The data in the user-defined DMA buffer is stored as count values. Refer to Appendix A for information on converting the count value to voltage or degrees.

Notes: Because of the overhead required to stop one DMA-mode operation and start another, a slight delay occurs before the DAS-Scan system begins acquiring data on each subsequent virtual board.

If DMA resources are not available, you can also perform an analog input operation in interrupt mode, using interrupts to indicate when data should be transferred from the FIFO. Refer to page 3-19 for more information.

Typically, you want the operation to stop automatically after one sample from each logical channel on the virtual board has been read; this is referred to as single-cycle buffering mode. However, if you are acquiring data from 256 logical channels or fewer, you can also use continuous buffering mode; this allows you to continuously acquire data, overwriting any values already stored in memory. Refer to page 3-23 for more information.

You can perform an analog input operation in single-DMA mode or dual-DMA mode, depending on whether you specified one or two DMA channels in your configuration file. Refer to the *DAS-Scan User's Guide* for more information.

Accessing a Frame

A frame is a data structure whose elements define the attributes of a DMA-mode analog input operation. For each virtual board in your DAS-Scan system, use the **K_GetADFrame** function to access an A/D (analog-to-digital) frame. The driver returns a unique identifier for each frame; this identifier is called the frame handle.

Specify the attributes of each operation by using a separate setup function to define each element of the A/D frame. Use the frame handle returned by the driver in each setup function to ensure that you always define the operation for the same virtual board. For example, assume that you access an A/D frame with the frame handle ADFrame1. To specify the logical channel on which to perform the operation, use the **K_SetChn** setup function, referencing the frame handle ADFrame1; to specify the gain at which to read the logical channel, use the **K_SetG** setup function, also referencing the frame handle ADFrame1.

When you are ready to perform the operation you set up, use the **K_DMAStart** function to start the operation, again referencing the appropriate frame handle. Figure 2-1 illustrates the use of an A/D frame where the frame handle is ADFrame1.

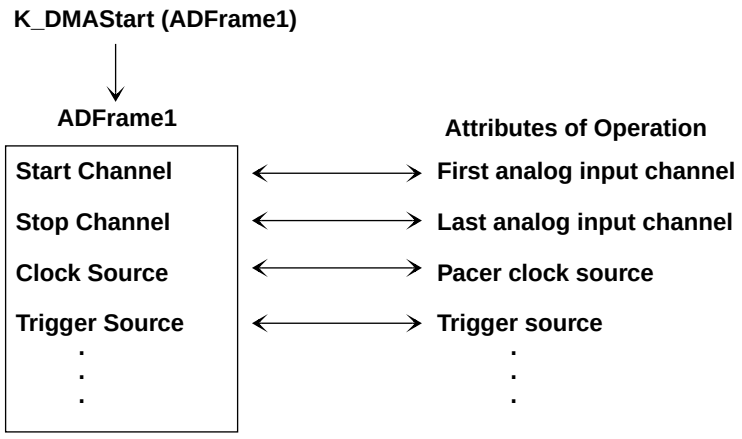


Figure 2-1. Frame-Based Operation

Frames help you create structured application programs. They are useful for operations that have many defining attributes, since providing a separate argument for each attribute could make a function's argument list unmanageably long. In addition, some attributes, such as the clock source and trigger source, are only available for operations that use frames.

You must use as least as many frames as you have virtual boards. For example, if you are using 16 virtual boards (the maximum number), you must use at least 16 frames. The maximum number of frames that you can use in your application program is 32 (two for each virtual board).

Note: To acquire data from all the virtual boards in your DAS-Scan system as smoothly as possible, make sure that you define the elements of each frame consistently. For example, to use the same pacer clock rate for all the virtual boards in your DAS-Scan system, specify the same number of clock ticks for each frame.

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

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

Table 2-1 lists the elements of an A/D frame that are typically defined for a DAS-Scan system. This table also lists the default value of each element, the setup functions used to define each element, and the page(s) in this manual on which to find additional information. Refer to the DAS-Scan Function Call Driver online help file (SCANFCD.HLP) for more detailed information about the setup functions.

Table 2-1. A/D Frame Elements

Element	Default Value	Setup Function	Page Number
Buffer[1]	0 (NULL)	K_SetDMABuf	page 2-9
Number of Samples	0	K_SetDMABuf	page 2-9
Start Channel	0	K_SetChn	page 2-17
		K_SetStartStopChn	page 2-17
		K_SetStartStopG	page 2-17
Stop Channel	0	K_SetStartStopChn	page 2-17
		K_SetStartStopG	page 2-17
Gain	0 (gain of 1)	K_SetG	page 2-17
		K_SetStartStopG	page 2-17
Channel-Gain Queue	0 (NULL)	K_SetChnGAry	page 2-18
Clock Source	Internal	K_SetClk	page 2-20, page 2-21
Pacer Clock Rate[1]	0	K_SetClkRate	page 2-20
External Clock Edge	Negative	K_SetExtClkEdge	page 2-21
Trigger Source	Internal	K_SetTrig	page 2-23
Trigger Polarity	Positive	K_SetDITrig	page 2-23
Hardware Gate	Disabled	K_SetGate	page 2-24

Notes
[1] This element must be set.

Memory Allocation and Management

A DMA-mode analog input operation requires memory in which to store the acquired data.

To reserve memory, use the **K_DMAAlloc** function to dynamically allocate a memory buffer for each virtual board. Specify the number of samples to store in the buffer (typically, one sample for each logical channel associated with the virtual board). The driver returns the starting

address of the buffer and a unique identifier for the buffer (this identifier is called the memory handle).

When you no longer require the buffer, you can free the buffer for another use by specifying the memory handle in the **K_DMAFree** function.

After you allocate your buffer, use the **K_SetDMABuf** function to assign the starting address of the buffer and the number of samples to store in the buffer.

Notes: For Visual Basic for Windows, data in a dynamically allocated buffer is not directly accessible to your program. You must use the **K_MoveBufToArray** function to move the data from the dynamically allocated buffer to the program's local array.

If you are writing Windows 95, 32-bit programs, you must install the Keithley Memory Manager. Refer to your board user's guide for information.

Typically, a single buffer that can hold one sample for each logical channel associated with the virtual board is sufficient. However, if you are acquiring data from 256 logical channels or fewer, you can also allocate a larger buffer or multiple buffers, allowing you to sample each logical channel multiple times. Refer to page 3-24 for more information.

The area used for dynamically allocated memory buffers is referred to as the global heap. This area of memory is left unoccupied as your application program and other programs run. The **K_DMAAlloc** function calls the **GlobalAlloc** API function to allocate the desired buffer size from the global heap. Dynamically allocated memory is guaranteed to be fixed and locked in memory.

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

Gains

Each logical channel of your DAS-Scan system can measure analog input signals in one of eight, software-selectable unipolar or bipolar analog input ranges. You initially set the input range type (unipolar or bipolar) for each virtual board in your configuration file; refer to the *DAS-Scan User's Guide* for more information. To reset the input range type for a virtual board in your application program, use the **K_SetADMode** function.

Table 2-2 lists the analog input ranges supported by the DAS-Scan system and the gain and gain code associated with each range. (The gain code is used by the Function Call Driver to represent the gain.)

Table 2-2. Analog Input Ranges

Analog Input Range		Gain	Gain Code
Bipolar	**Unipolar**		
±10 V	0 to 10 V	1	0
±5 V	0 to 5 V	2	1
±2.5 V	0 to 2.5 V	4	2
±1.25 V	0 to 1.25 V	8	3
±0.2 V	0 to 0.2 V	50	4
±0.1 V	0 to 0.1 V	100	5
±50 mV	0 to 50 mV	200	6
±25 mV	0 to 25 mV	400	7

For a single-mode operation, you specify the gain code in the **K_ADRead** function.

For a DMA-mode operation, you specify the gain code in the **K_SetG** or **K_SetStartStopG** function; the function you use depends on how you specify the logical channels, as described in the following section.

Channels

A DAS-Scan system can contain a maximum of 4,096 analog input channels. You can perform an analog input operation on a single channel or on a group of multiple channels called a scan. The following sections describe how to address and specify the channels you are using.

Addressing Channels

As mentioned previously, each DAS-Scan system contains one physical SCAN-AD-HR board that you plug into your computer. You can attach up to 64 physical SCAN-BRD assemblies to the SCAN-AD-HR. You give each SCAN-BRD assembly a unique address (from 0h to 3Fh, 0 to 63 decimal) by setting a thumbwheel on the board. Each SCAN-BRD assembly contains 64 physical channels, numbered from 0 to 63 on each assembly.

The DAS-Scan Function Call Driver treats each group of four SCAN-BRD assemblies as a virtual board. Because duplicate physical channel numbers can be associated with a virtual board, the driver uses a logical channel number (from 0 to 255) to uniquely identify each physical channel. The channel number that you specify in Function Call Driver functions is always a logical channel number.

Table 2-3 illustrates the mapping of logical channels to virtual boards for the maximum configuration of 4,096 channels. Table 2-3 assumes that when you set the address thumbwheels on the SCAN-BRD assemblies and specified the addresses in the configuration file, you assigned the SCAN-BRD assemblies sequential addresses, as recommended. However, you can assign any address to any SCAN-BRD assembly, as long as no two SCAN-BRD assemblies have the same address.

Table 2-3. Virtual Boards and Logical Channels

Virtual Board	SCAN-BRD Assemblies	Logical Channels
0	SCAN-BRD0 (Address 0h)	0 to 63
	SCAN-BRD1 (Address 1h)	64 to 127
	SCAN-BRD2 (Address 2h)	128 to 191
	SCAN-BRD3 (Address 3h)	192 to 255
1	SCAN-BRD0 (Address 4h)	0 to 63
	SCAN-BRD1 (Address 5h)	64 to 127
	SCAN-BRD2 (Address 6h)	128 to 191
	SCAN-BRD3 (Address 7h)	192 to 255
2	SCAN-BRD0 (Address 8h)	0 to 63
	SCAN-BRD1 (Address 9h)	64 to 127
	SCAN-BRD2 (Address Ah)	128 to 191
	SCAN-BRD3 (Address Bh)	192 to 255
3	SCAN-BRD0 (Address Ch)	0 to 63
	SCAN-BRD1 (Address Dh)	64 to 127
	SCAN-BRD2 (Address Eh)	128 to 191
	SCAN-BRD3 (Address Fh)	192 to 255
4	SCAN-BRD0 (Address 10h)	0 to 63
	SCAN-BRD1 (Address 11h)	64 to 127
	SCAN-BRD2 (Address 12h)	128 to 191
	SCAN-BRD3 (Address 13h)	192 to 255
5	SCAN-BRD0 (Address 14h)	0 to 63
	SCAN-BRD1 (Address 15h)	64 to 127
	SCAN-BRD2 (Address 16h)	128 to 191
	SCAN-BRD3 (Address 17h)	192 to 255

Table 2-3. Virtual Boards and Logical Channels (cont.)

Virtual Board	SCAN-BRD Assemblies	Logical Channels
6	SCAN-BRD0 (Address 18h)	0 to 63
	SCAN-BRD1 (Address 19h)	64 to 127
	SCAN-BRD2 (Address 1Ah)	128 to 191
	SCAN-BRD3 (Address 1Bh)	192 to 255
7	SCAN-BRD0 (Address 1Ch)	0 to 63
	SCAN-BRD1 (Address 1Dh)	64 to 127
	SCAN-BRD2 (Address 1Eh)	128 to 191
	SCAN-BRD3 (Address 1Fh)	192 to 255
8	SCAN-BRD0 (Address 20h)	0 to 63
	SCAN-BRD1 (Address 21h)	64 to 127
	SCAN-BRD2 (Address 22h)	128 to 191
	SCAN-BRD3 (Address 23h)	192 to 255
9	SCAN-BRD0 (Address 24h)	0 to 63
	SCAN-BRD1 (Address 25h)	64 to 127
	SCAN-BRD2 (Address 26h)	128 to 191
	SCAN-BRD3 (Address 27h)	192 to 255
10	SCAN-BRD0 (Address 28h)	0 to 63
	SCAN-BRD1 (Address 29h)	64 to 127
	SCAN-BRD2 (Address 2Ah)	128 to 191
	SCAN-BRD3 (Address 2Bh)	192 to 255
11	SCAN-BRD0 (Address 2Ch)	0 to 63
	SCAN-BRD1 (Address 2Dh)	64 to 127
	SCAN-BRD2 (Address 2Eh)	128 to 191
	SCAN-BRD3 (Address 2Fh)	192 to 255

Table 2-3. Virtual Boards and Logical Channels (cont.)

Virtual Board	SCAN-BRD Assemblies	Logical Channels
12	SCAN-BRD0 (Address 30h)	0 to 63
	SCAN-BRD1 (Address 31h)	64 to 127
	SCAN-BRD2 (Address 32h)	128 to 191
	SCAN-BRD3 (Address 33h)	192 to 255
13	SCAN-BRD0 (Address 34h)	0 to 63
	SCAN-BRD1 (Address 35h)	64 to 127
	SCAN-BRD2 (Address 36h)	128 to 191
	SCAN-BRD3 (Address 37h)	192 to 255
14	SCAN-BRD0 (Address 38h)	0 to 63
	SCAN-BRD1 (Address 39h)	64 to 127
	SCAN-BRD2 (Address 3Ah)	128 to 191
	SCAN-BRD3 (Address 3Bh)	192 to 255
15	SCAN-BRD0 (Address 3Ch)	0 to 63
	SCAN-BRD1 (Address 3Dh)	64 to 127
	SCAN-BRD2 (Address 3Eh)	128 to 191
	SCAN-BRD3 (Address 3Fh)	192 to 255

For each virtual board, you can determine the logical channel number corresponding to a particular physical channel number by using the following equation:

$$\text{LogicalChn\#} = \text{PhysicalChan\#} + (64 \times \text{SCAN-BRD\#})$$

where

PhysicalChan# is an integer from 0 to 63 that indicates the physical channel number on the SCAN-BRD assembly specified by *SCAN-BRD#*, and

SCAN-BRD# is an integer from 0 to 3 that indicates on which SCAN-BRD assembly the physical channel is located (0 indicates the first SCAN-BRD assembly associated with the virtual board, 1 indicates the second SCAN-BRD assembly associated with the virtual board, 2 indicates the third SCAN-BRD assembly associated with the virtual board, and 3 indicates the fourth SCAN-BRD assembly associated with the virtual board).

For example, the logical channel that identifies physical channel 15 on SCAN-BRD2 (the third SCAN-BRD assembly associated with the virtual board) is determined as follows:

$$\text{LogicalChan\#} = 15 + (64 \times 2) = 15 + 128 = 143$$

By using a combination of the virtual board number and the logical channel number in software, you can uniquely identify each channel in your DAS-Scan system. For example, assume you have eight SCAN-BRD assemblies connected to your SCAN-AD-HR board. Physical channel number 51 on the fourth SCAN-BRD assembly (SCAN-BRD3, Address 3) is identified as virtual board 0, logical channel 243 (51 + (64 x 3)); physical channel 51 on the eighth SCAN-BRD assembly (SCAN-BRD3, Address 7) is identified as virtual board 1, logical channel 243.

Note: In the configuration file, the SCAN-BRD assemblies are addressed sequentially from 0h to 3Fh (0 to 63 decimal), as shown in Table 2-3. However, for the purposes of determining the logical channel number, always use the number of the SCAN-BRD assembly in relation to a virtual board (0, 1, 2, or 3).

Specifying a Single Channel

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

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

Refer to Table 2-2 on page 2-11 for a list of the analog input ranges supported by the DAS-Scan system and the gain code associated with each range.

Note: If you are measuring steady-state signals, you can reduce the effects of noise by performing a separate DMA-mode operation on each logical channel and then averaging the results of each operation. For example, you can perform one DMA-mode operation in which you acquire 1,000 samples from logical channel 0 at a gain of 1 and another DMA-mode operation in which you acquire 2,000 samples from logical channel 1 at a gain of 200.

Specifying a Group of Consecutive Logical Channels

For a DMA-mode analog input operation, you can acquire samples from a group of up to 256 consecutive logical channels associated with the same virtual board. Use the **K_SetStartStopChn** function to specify the first and last logical channels. The channels are sampled in order from first to last.

The first logical channel can be higher than the last logical channel. For example, if the first logical channel is 255 and the last logical channel is 2, your program reads data first from logical channel 255, then from logical channels 0, 1, and 2.

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

Refer to Table 2-2 on page 2-11 for a list of the analog input ranges supported by the DAS-Scan system and the gain code associated with each range.

Note: If you are measuring temperature with a SCAN-BRD-TC or SCAN-BRD-TC-ISO assembly, you can use physical channels 31 and 63 as CJC sensors to measure the temperature of the screw terminals. If you enable a CJC channel, the CJC channel should be read at a gain of 1. Unless you intend to read all your channels at a gain of 1, you must either disable the CJC channels or specify your measurement channels in a channel-gain queue. Refer to the next section for information about channel-gain queues.

Specifying Channels in a Channel-Gain Queue

For a DMA-mode analog input operation, you can use a hardware channel-gain queue to acquire samples from up to 256 logical channels associated with the same virtual board. You specify the logical channels you want to sample, the order in which you want to sample them, and a gain code for each logical channel.

You can set up the logical channels in a channel-gain queue either in consecutive order or in nonconsecutive order. You can also specify the same logical channel more than once. The channels are sampled in order from the first logical channel in the queue to the last logical channel in the queue.

Refer to Table 2-2 on page 2-11 for a list of the analog input ranges supported by the DAS-Scan system and the gain code associated with each range.

The way that you specify the logical channels and gains in a channel-gain queue depends on the language you are using. Refer to the DAS-Scan Function Call Driver online help file (SCANFCD.HLP) for language-specific information.

After you create the channel-gain queue in your program, use the **K_SetChnGAry** function to specify the staring address of the channel-gain queue.

Notes: The throughput of the DAS-Scan system is reduced when using a channel-gain queue, particularly when the SCAN-AD-HR board must switch the gain between each logical channel. Therefore, it is recommended that you keep all logical channels you are reading at a particular gain together, even if you must arrange the channels out of sequence.

For example, assume that you are measuring temperature on a SCAN-BRD-TC or SCAN-BRD-TC-ISO assembly and you are using physical channels 31 and 63 as CJC sensors to measure the temperature of the screw terminals. Since the CJC channels should be read at a gain of 1, arrange the channel-gain queue so that you read all the measurement channels first and then read the CJC channels. Example program 4, provided in the ASO-SCAN software package, illustrates the use of a channel-gain queue with the CJC channels enabled.

If you are not using all 256 logical channels on a virtual board, you can allow for settling time by specifying the same logical channel twice and then ignoring the reading from the first logical channel.

If you are not using all 256 logical channels on a virtual board, you can reduce the effects of noise by specifying the same logical channel multiple times and then averaging the readings from each channel. For example, you can specify logical channel 0 for the first five entries in the channel-gain queue, logical channel 1 for the next five entries, and so on.

Pacer Clock

For a DMA-mode operation, you can use a pacer clock to determine the period between the conversion of one logical channel and the conversion of the next logical channel in a scan of multiple logical channels. You can specify an internal or an external pacer clock, as described in the following sections.

Notes: The rate at which the computer can reliably read data from the SCAN-AD-HR board depends on a number of factors, including your computer, the operating system/environment, the types of SCAN-BRD assemblies you are using, the gains of the channels, and other software issues. If your data appears to be invalid, you may have to use a slower clock rate. Refer to the *DAS-Scan User's Guide* for the maximum throughput rates supported for each SCAN-BRD assembly at each gain.

Typically, using the pacer clock to control the period between conversions of individual logical channels in a scan is sufficient; this is referred to as paced conversion mode. However, if you are acquiring data from 256 logical channels or fewer, you can use both the pacer clock (to control the period between individual logical channels in a scan) and the burst mode conversion clock (to control the period between conversions of the entire scan); this is referred to as burst conversion mode. Refer to page 3-25 for more information about specifying burst conversion mode. Refer to page 3-26 for more information about using the burst mode conversion clock.

Internal Pacer Clock

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

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

Because the counters use a 5 MHz time base, each count represents 0.2 μs. Use the **K_SetClkRate** function to specify the number of counts (clock ticks) between conversions. For example, if you specify a count of 300, the period between conversions is 60 μs (16.67 ksamples/s); if you specify a count of 87,654, the period between conversions is 17.53 ms (57 samples/s).

You can specify a count value from 50 (100 kHz) to 4,294,836,255 (0.0012 Hz). The period between conversions ranges from 10 μs to 14.3 minutes.

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

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

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

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

If you want to use the same conversion rate for all the virtual boards in your DAS-Scan system, call **K_SetClkRate** once for each virtual board and specify the same number of counts for each.

The internal pacer clock is the default pacer clock. To reset the pacer clock source to the internal pacer clock, use the **K_SetClk** function. (If you want to reset all the virtual boards in your DAS-Scan system to use the internal pacer clock, call **K_SetClk** once for each virtual board and specify the internal pacer clock for each.)

External Pacer Clock

You connect an external pacer clock to the J5 connector on the SCAN-AD-HR board. When you start an analog input operation (using **K_DMAStart**), conversions are armed. At the next active edge of the external pacer clock (and at every subsequent active edge of the external pacer clock), a conversion is initiated.

Use the **K_SetClk** function to specify an external pacer clock. Then, use the **K_SetExtClkEdge** function to specify the active edge (rising or falling) of the external pacer clock. A falling edge is the default active edge.

If you are using an external pacer clock, make sure that the clock initiates conversions at a rate that the analog-to-digital converter can handle.

If you want to use an external pacer clock source for all the virtual boards in your DAS-Scan system, call **K_SetClk** once for each virtual board and specify an external pacer clock for each. If you want to use the same active edge for all the virtual boards in your DAS-Scan system, call **K_SetExtClkEdge** once for each virtual board and specify the same active edge for each.

Note: You cannot use an external pacer clock if you are using either an external digital trigger or a hardware gate.

Triggers

A trigger is an event that occurs based on a specified set of conditions. If you are performing DMA-mode analog input operations on a series of virtual boards, you can specify a start trigger for the first virtual board to determine when acquisitions start.

The start trigger can be an internal trigger or an external digital trigger, as described in the following sections.

The trigger event is not significant until the operation has been started (using **K_DMAStart**). The point at which conversions begin relative to the trigger event depends on the pacer clock; refer to page 2-19 for more information.

Note: Typically, you want to acquire data after the trigger event occurs; this is referred to as post-trigger acquisition. However, if you are performing a DMA-mode operation and acquiring data from 256 logical channels or fewer, you can also use an optional second trigger, the about trigger, to acquire data before the trigger event occurs (referred to as pre-trigger acquisition) or both before and after the trigger event occurs (referred to as about-trigger acquisition). Refer to page 3-26 for more information about specifying pre-trigger acquisition. Refer to page 3-28 for more information about specifying about-trigger acquisition.

Internal Trigger

An internal trigger is a software trigger. The trigger event occurs when you start the operation. Note that a slight delay occurs between the time you start the operation and the time the trigger event occurs.

If you want to use an internal start trigger, specify an internal trigger for all the virtual boards in your DAS-Scan system.

The internal trigger is the default trigger source. To reset the trigger source to an internal trigger, use the **K_SetTrig** function.

External Digital Trigger

An external digital trigger event occurs when an active edge is detected on the digital trigger signal connected to the J5 connector on the SCAN-AD-HR board.

Use the **K_SetTrig** function to specify an external trigger. Then, use the **K_SetDITrig** function to specify whether you want the trigger event to occur on a rising edge (positive-edge trigger) or on a falling edge (negative-edge trigger). The trigger conditions are illustrated in Figure 2-2.

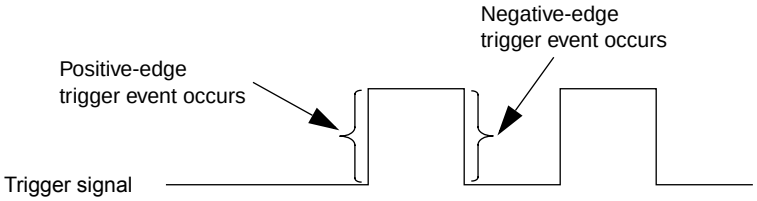


Figure 2-2. Digital Trigger Conditions

If you want to use an external digital start trigger, specify an external trigger (using **K_SetTrig**) and call **K_SetDITrig** for the first virtual board only. For all subsequent virtual boards, use **K_SetTrig** to specify an internal trigger and do not call **K_SetDITrig**.

Note: You cannot use an external digital trigger if you are using either an external pacer clock or a hardware gate.

Hardware Gate

A hardware gate is an externally applied digital signal that determines when conversions can occur. You connect the gate signal to the J5 connector on the SCAN-AD-HR board.

If you start a DMA-mode analog input operation (using **K_DMAStart**) and the hardware gate is enabled, the state of the gate signal determines whether conversions occur, as follows:

- If you specify a positive gate, conversions occur only if the signal to the J5 connector is high; if the signal to the J5 connector is low, conversions are inhibited.
- If you specify a negative gate, conversions occur only if the signal to the J5 connector is low; if the signal to the J5 connector is high, conversions are inhibited.

Use the **K_SetGate** function to enable and disable a hardware gate and to specify the gate polarity (positive or negative).

If you want to use a hardware gate for all the virtual boards in your DAS-Scan system, call **K_SetGate** once for each virtual board and specify the same polarity for each.

The default state of the hardware gate is disabled. If you enable a hardware gate for all the virtual boards in your DAS-Scan system and later want to disable the hardware gate, call **K_SetGate** once for each virtual board and specify disabled for each.

Note: You cannot use a hardware gate if you are using either an external pacer clock or an external digital trigger.

3

Additional Features

This chapter contains conceptual information about additional features of the DAS-Scan Function Call Driver. It includes the following sections:

- **Summary of Additional Functions** - a brief description of the additional DAS-Scan Function Call Driver functions that you can use when reading 256 logical channels or fewer or when performing an interrupt-mode analog input operation.
- **Additional Programming Flow Diagrams** - an illustration of the procedures to follow when using all of the DAS-Scan Function Call Driver functions.
- **Performing an Interrupt-Mode Operation** - information on performing an interrupt-mode analog input operation.
- **Specifying Continuous Mode** - information on specifying continuous buffering mode.
- **Reserving Large or Multiple Memory Buffers** - information on dynamically allocating a large memory buffer or dynamically allocating multiple memory buffers.
- **Specifying Burst Mode** - information on specifying burst conversion mode.
- **Using the Burst Mode Conversion Clock** - information on using the burst mode conversion clock to acquire data in burst mode.
- **Specifying Pre-Trigger Acquisition** - information on using the about trigger to acquire data before a trigger event occurs.
- **Specifying About-Trigger Acquisition** - information on using the about trigger to acquire data both before and after a trigger event occurs.

Summary of Additional Functions

Table 3-1 describes the additional functions in the DAS-Scan Function Call Driver that you can use when reading 256 logical channels or fewer or when performing an interrupt-mode analog input operation. For more detailed information about the functions, refer to the information in this chapter beginning on page 3-19 and to the DAS-Scan Function Call Driver online help file (SCANFCD.HLP).

Table 3-1. Summary of Additional Functions

Type of Function	Name of Function	Description
Operation	K_IntStart	Starts an interrupt-mode operation.
	K_IntStatus	Gets the status of an interrupt-mode operation.
	K_IntStop	Stops an interrupt-mode operation.
Memory management	K_IntAlloc	Dynamically allocates a memory buffer for an interrupt-mode operation.
	K_IntFree	Frees a memory buffer that was dynamically allocated for an interrupt-mode operation.
Buffer address	K_SetBuf	Specifies a local array (C/C++) or a dynamically allocated memory buffer (C/C++ or Visual Basic for Windows) for an interrupt-mode operation.
	K_SetBufI	Specifies a local array (Visual Basic for Windows) for an interrupt-mode operation.
	K_BufListAdd	Adds a buffer or array to the list of multiple buffers or arrays.
	K_BufListReset	Clears the list of multiple buffers or arrays.
Buffering mode	K_SetContRun	Specifies continuous mode.
	K_ClrContRun	Specifies single-cycle mode.
Conversion mode	K_SetADFreeRun	Specifies burst mode.
	K_ClrADFreeRun	Specifies paced mode.

Table 3-1. Summary of Additional Functions (cont.)

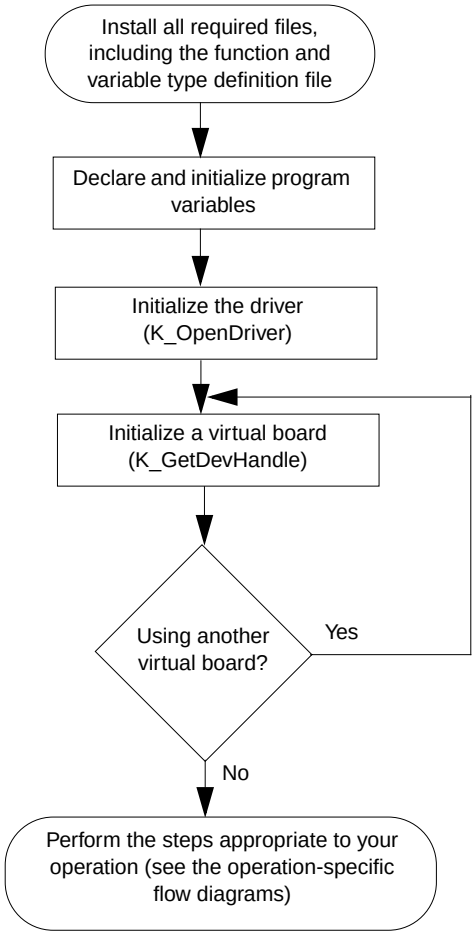
Type of Function	Name of Function	Description
Clock	K_SetBurstTicks	Specifies the burst mode conversion rate.
	K_GetBurstTicks	Gets the burst mode conversion rate.
Trigger	K_SetAboutTrig	Enables the about trigger and specifies the number of post-trigger samples.
	K_ClrAboutTrig	Disables the about trigger.

Additional Programming Flow Diagrams

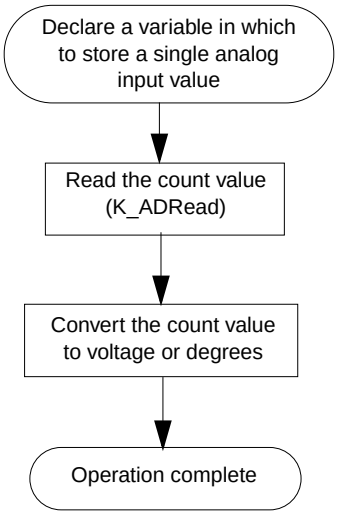
This section contains a series of programming flow diagrams illustrating the procedures to follow when using all of the DAS-Scan Function Call Driver functions. For more detailed information about the programming procedures, refer to the DAS-Scan Function Call Driver online help file (SCANFCD.HLP).

Although error checking is not shown in the flow diagrams, it is recommended that you check the error/status code returned by each function used in your application program.

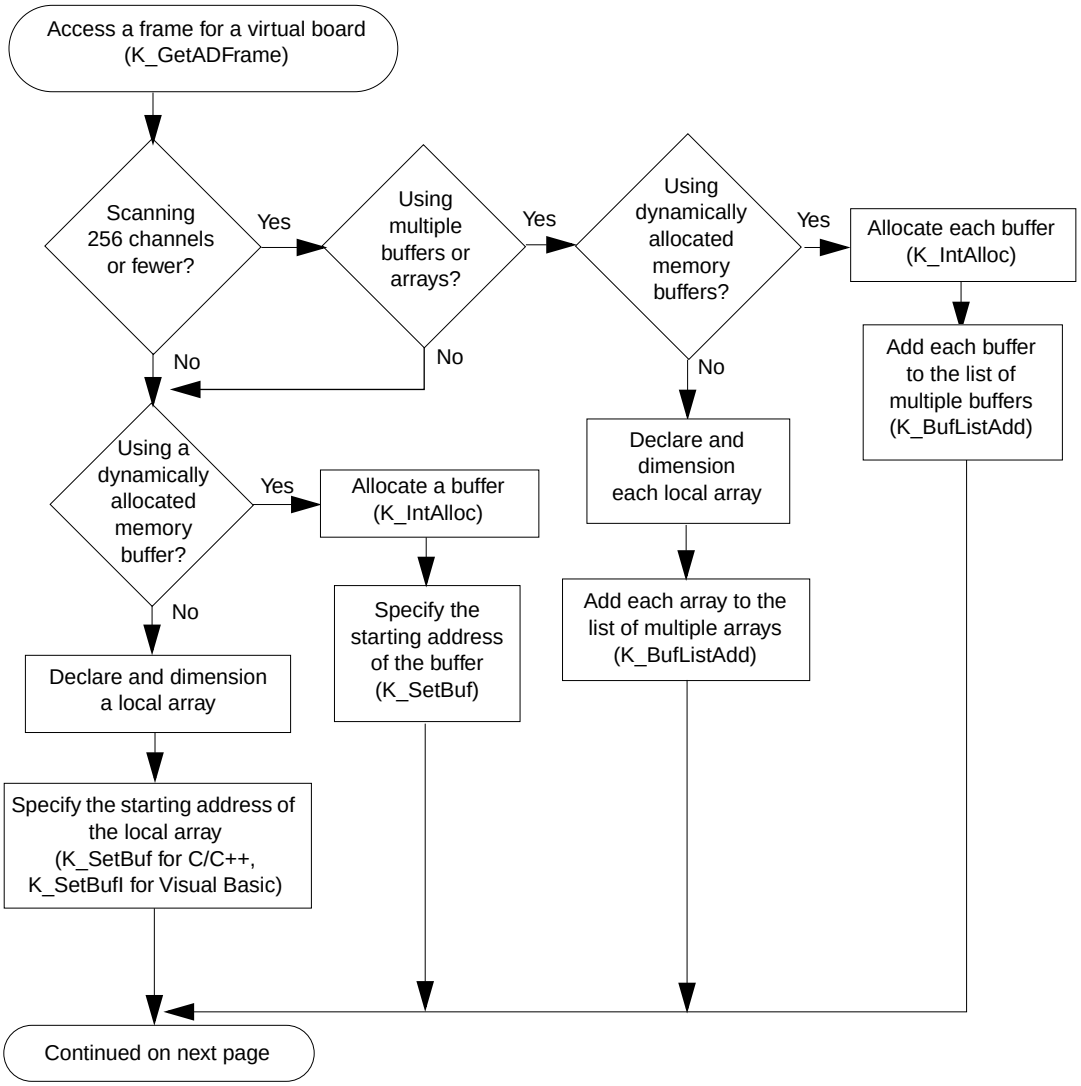
Preliminary Steps for All Analog Input Operations



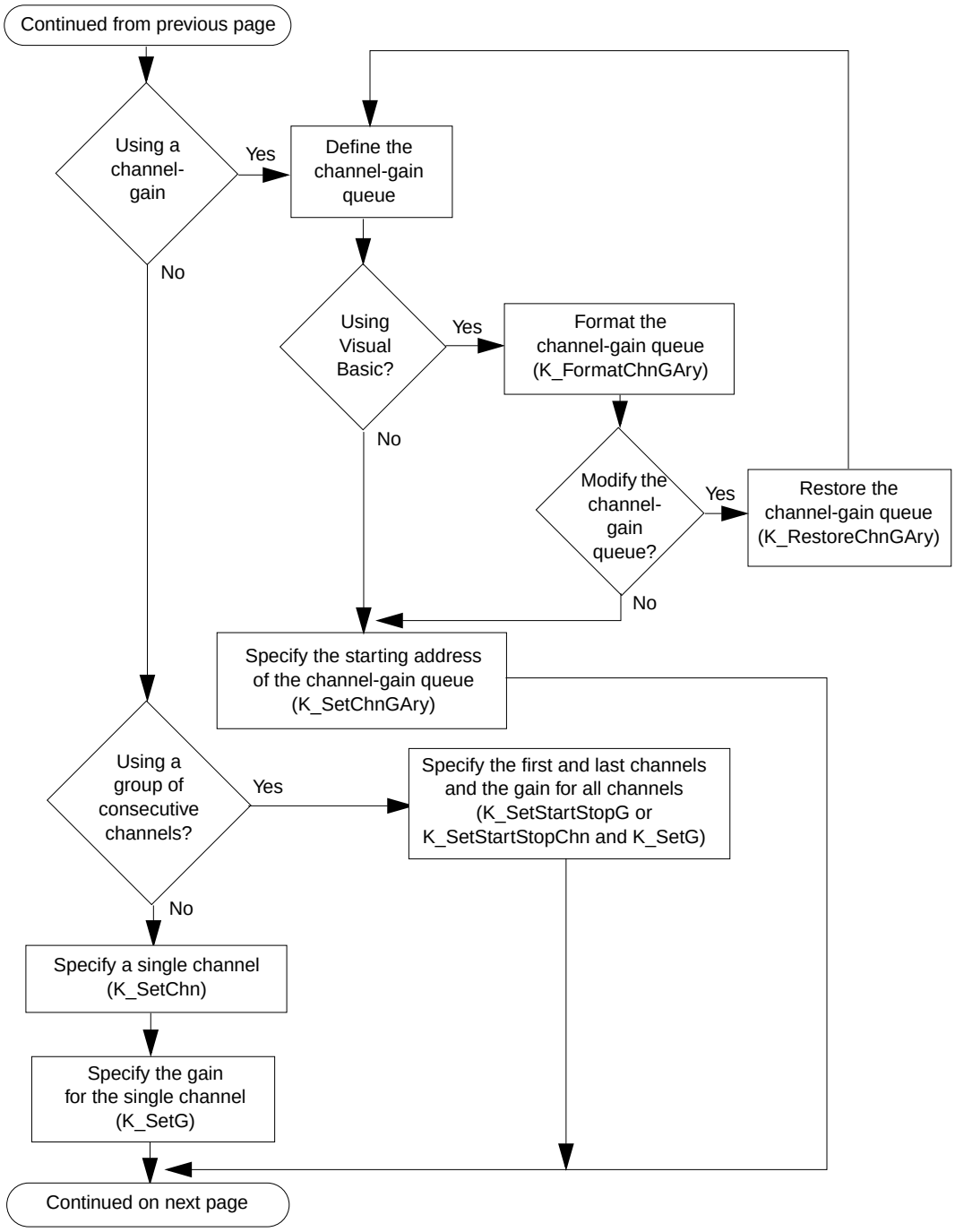
Steps for a Single-Mode Analog Input Operation



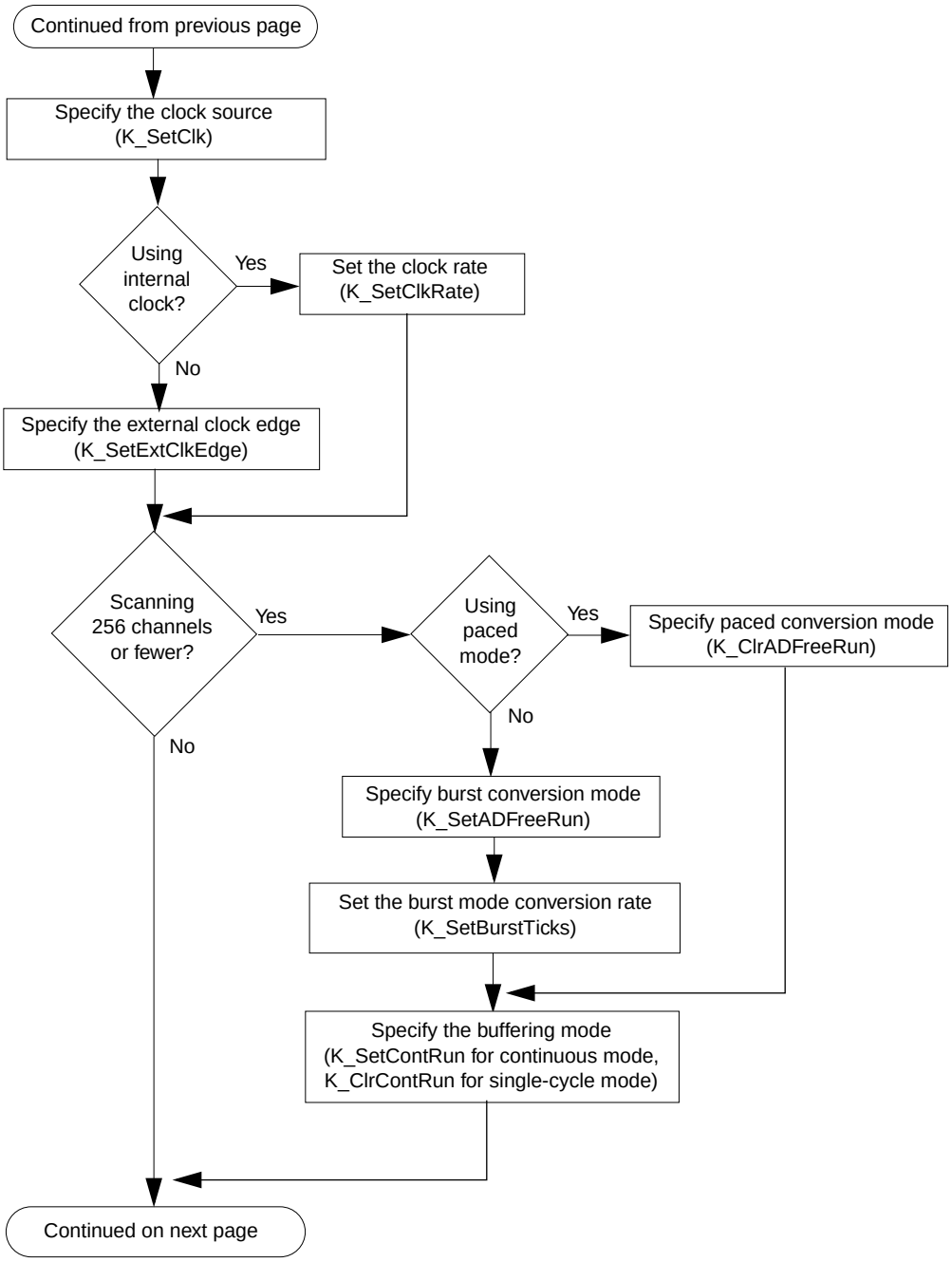
Steps for an Interrupt-Mode Analog Input Operation



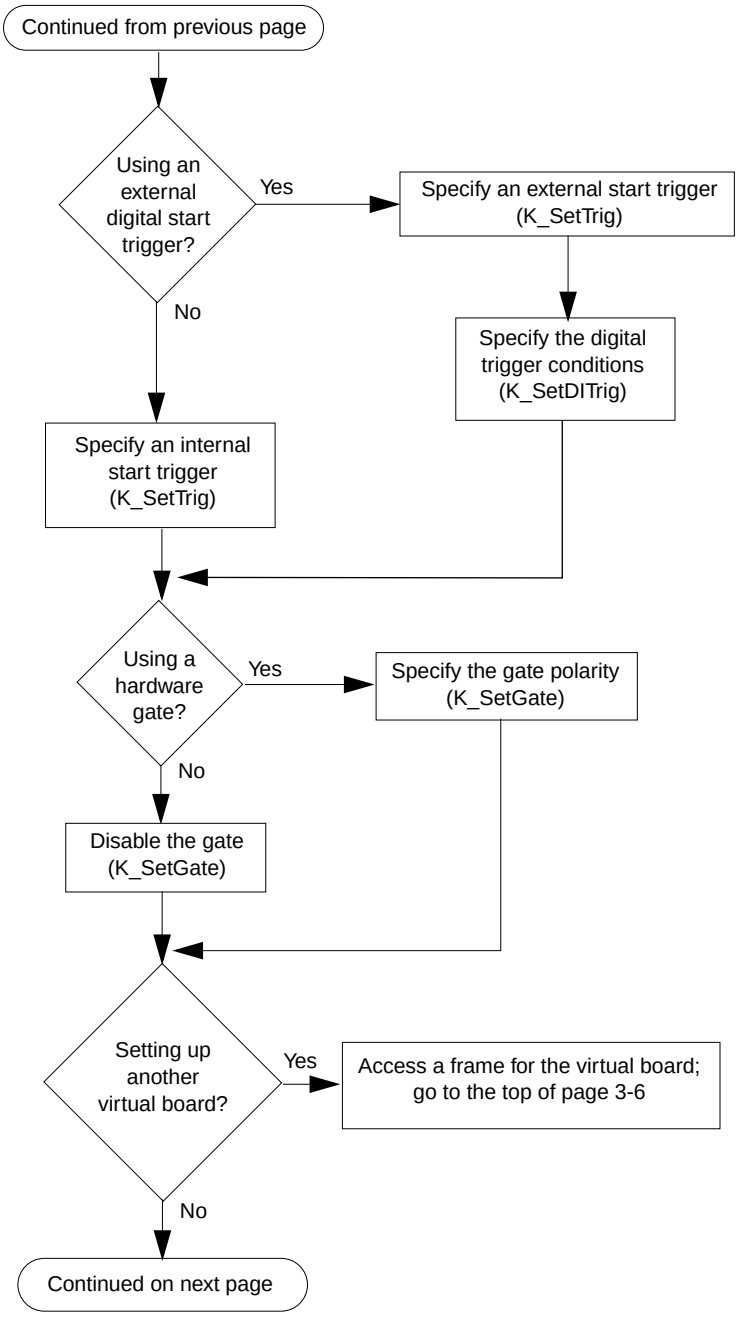
Steps for an Interrupt-Mode Analog Input Operation (cont.)



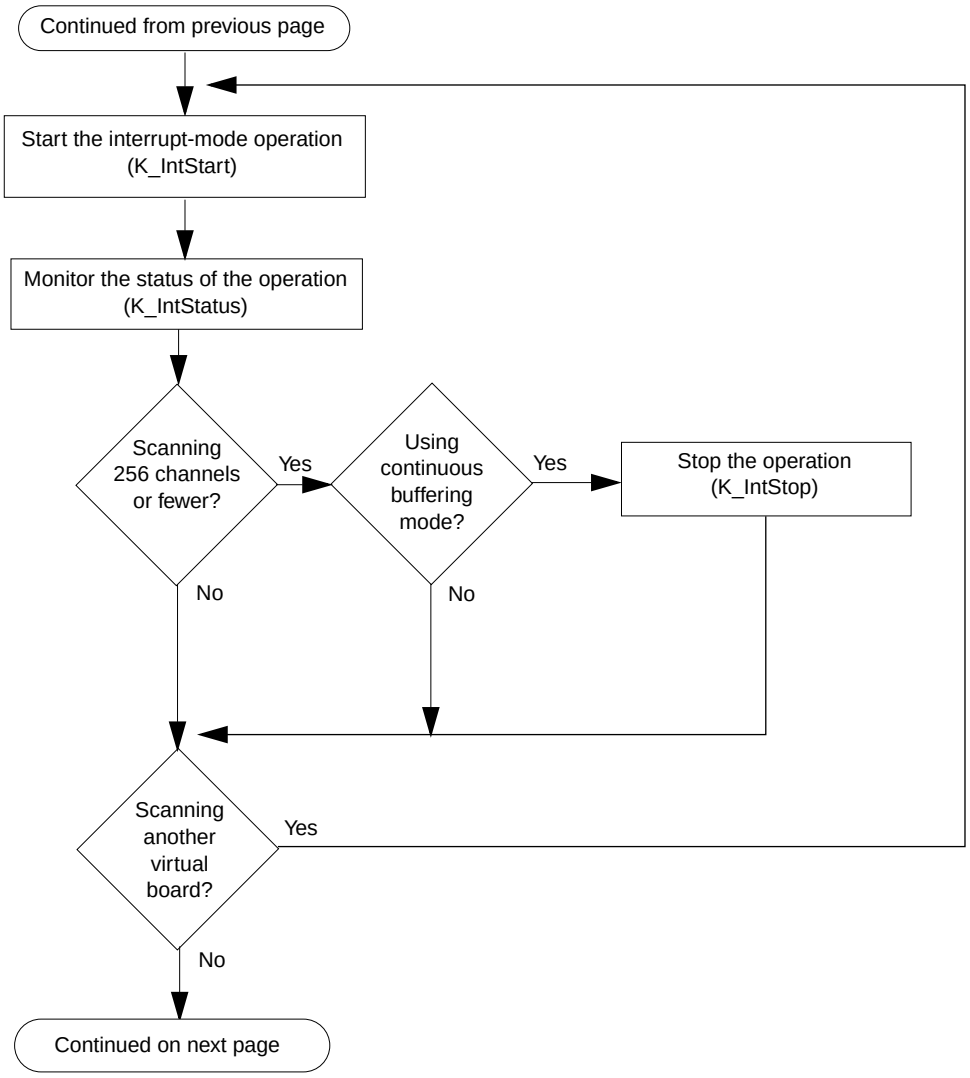
Steps for an Interrupt-Mode Analog Input Operation (cont.)



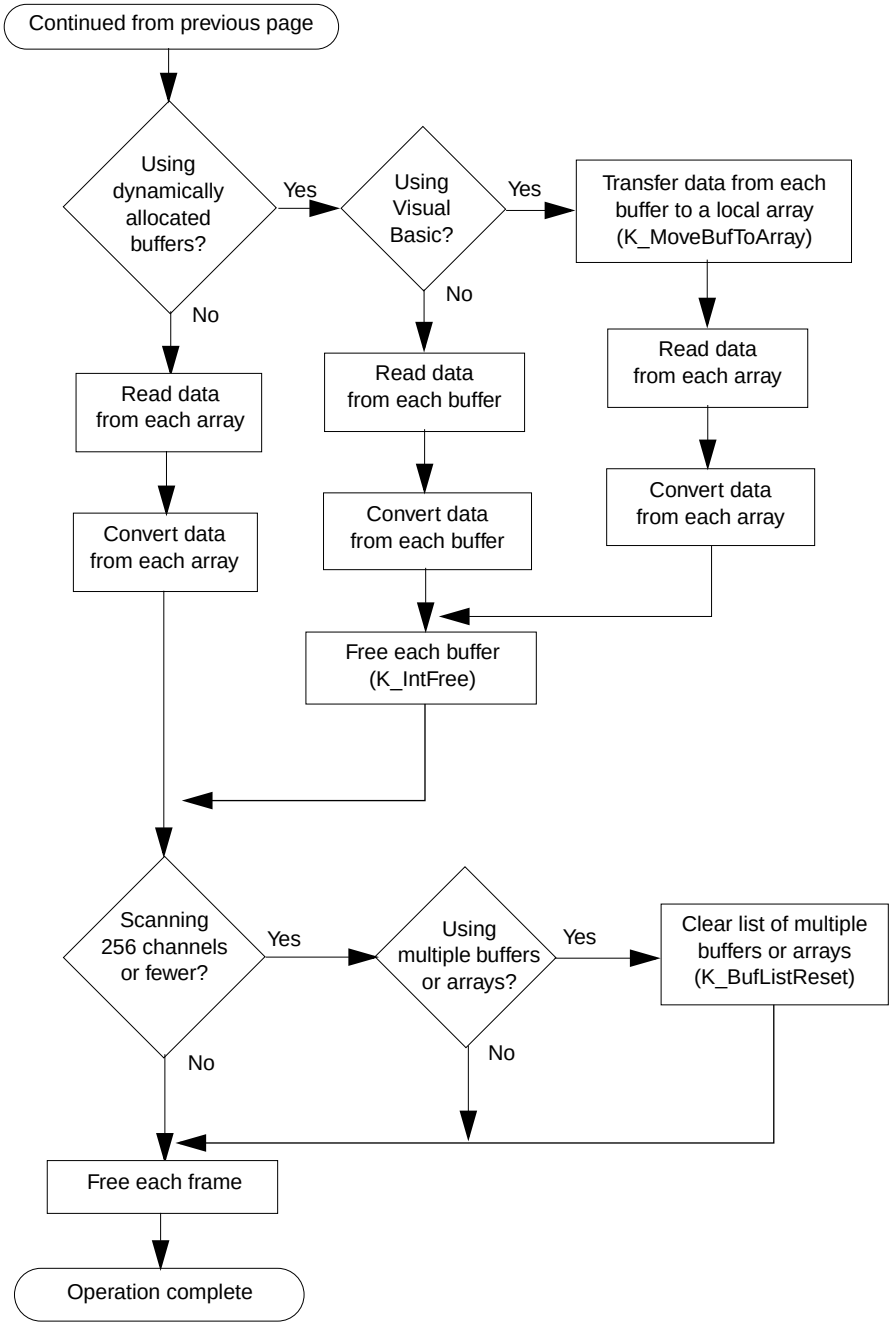
Steps for an Interrupt-Mode Analog Input Operation (cont.)



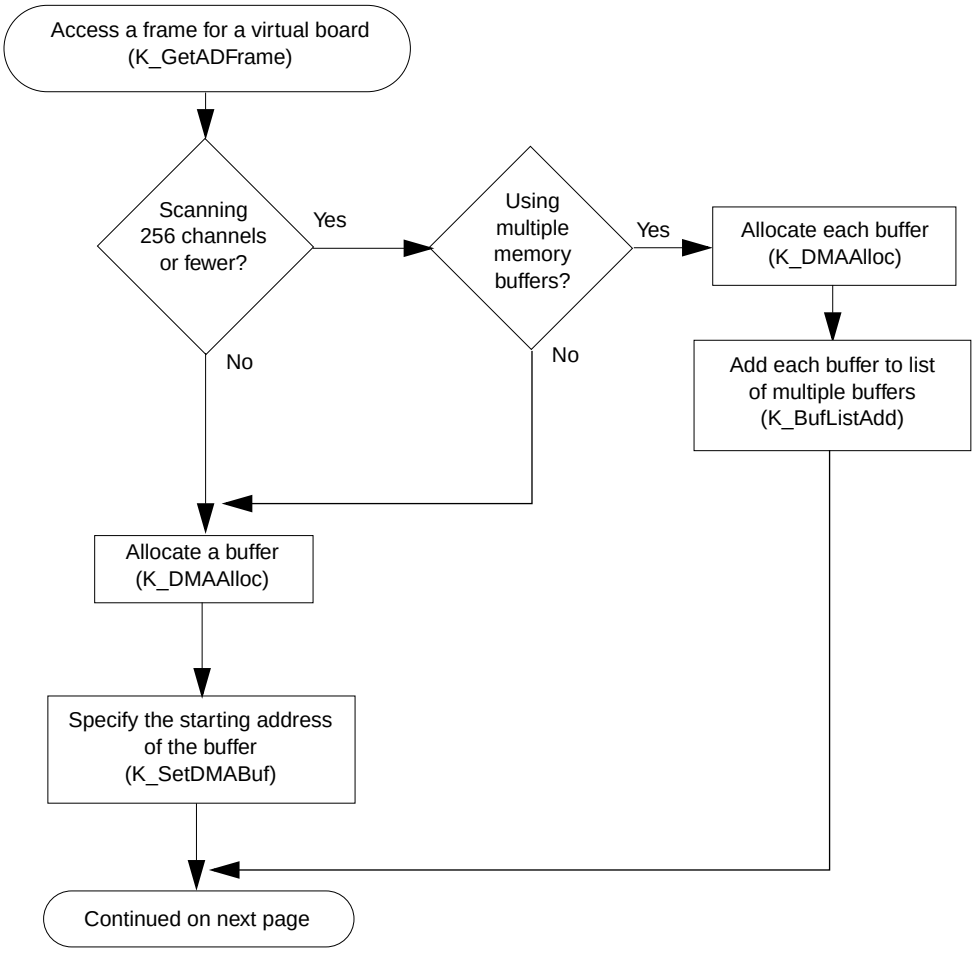
Steps for an Interrupt-Mode Analog Input Operation (cont.)



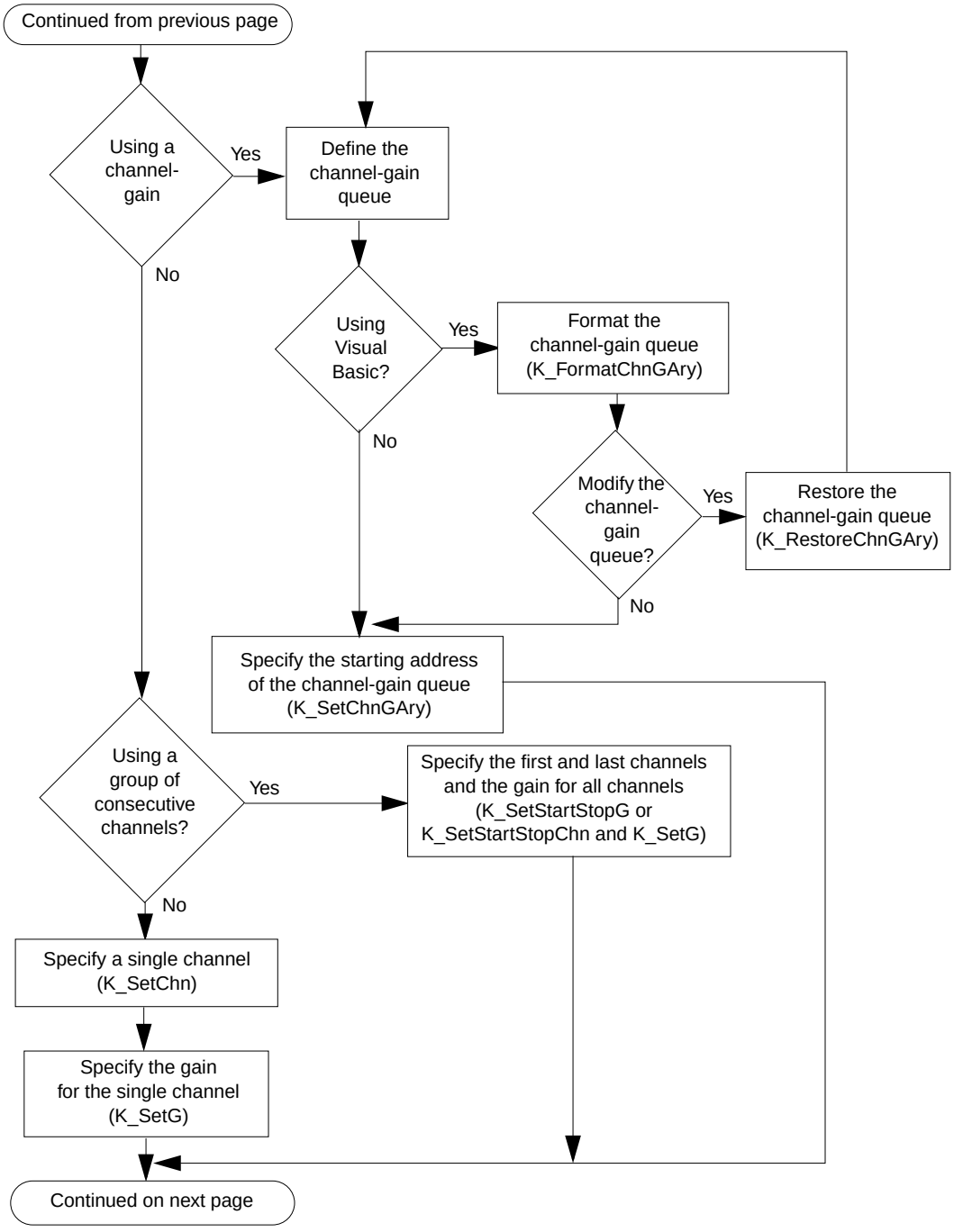
Steps for an Interrupt-Mode Analog Input Operation (cont.)



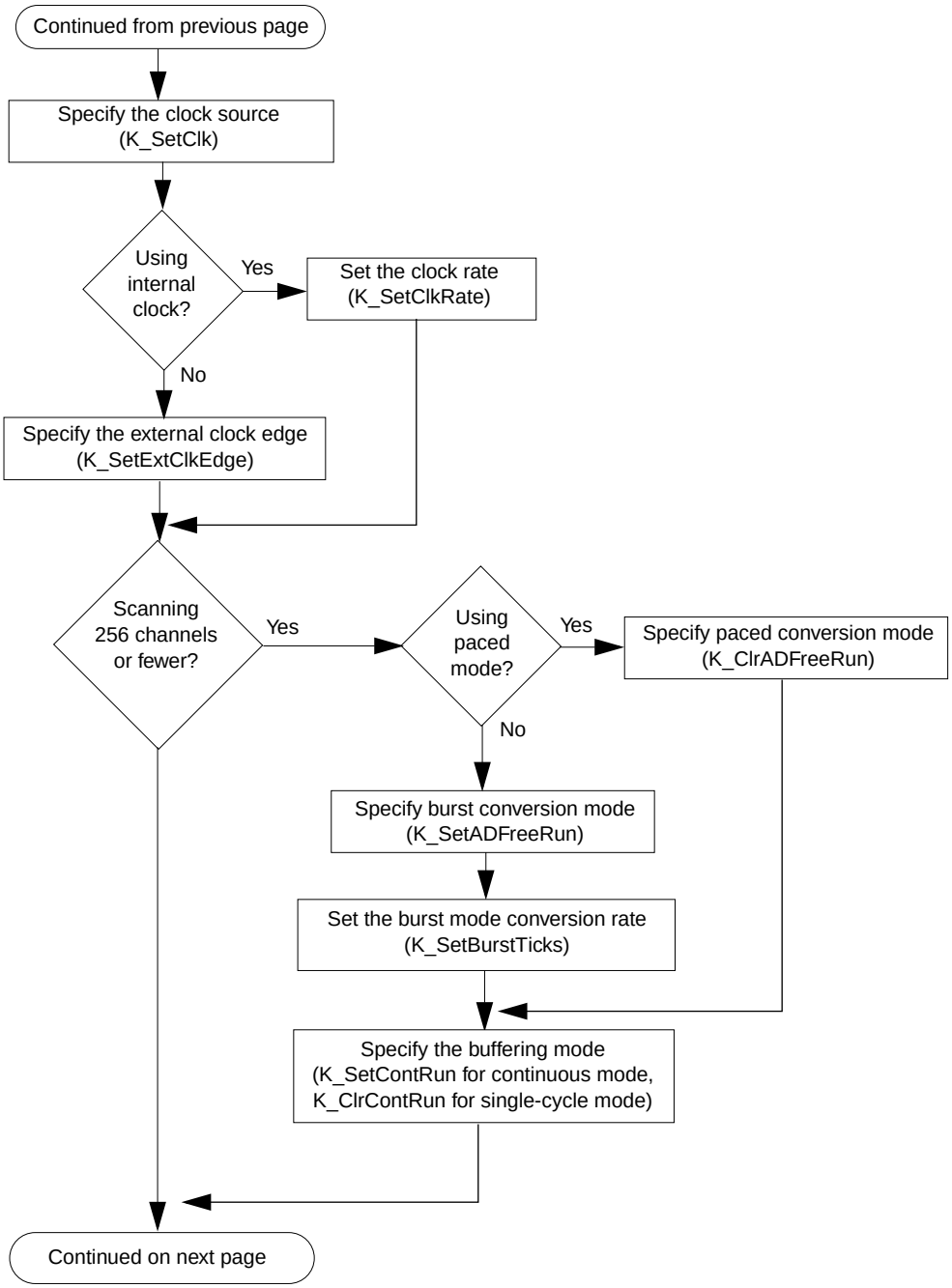
Steps for a DMA-Mode Analog Input Operation



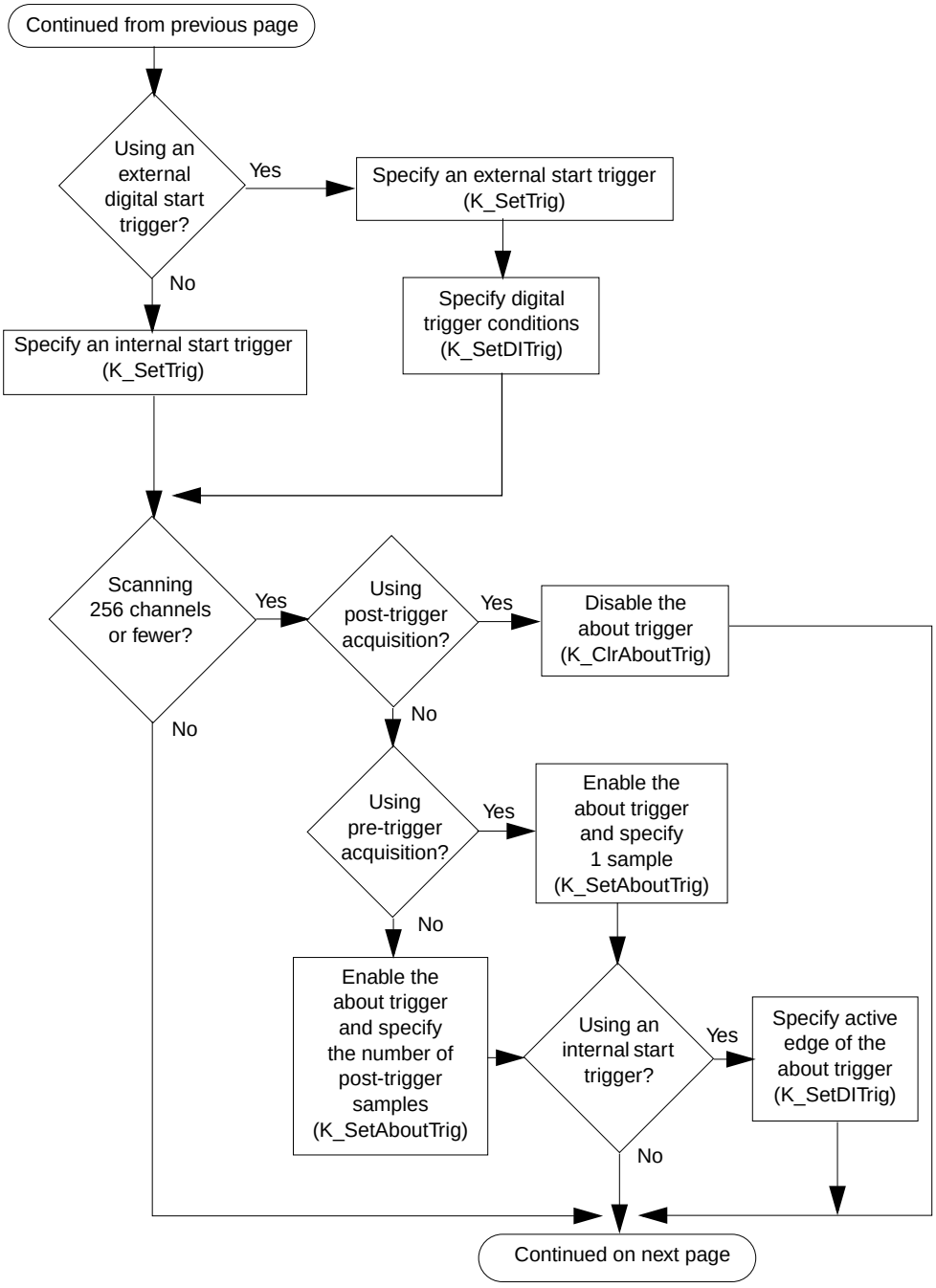
Steps for a DMA-Mode Analog Input Operation (cont.)



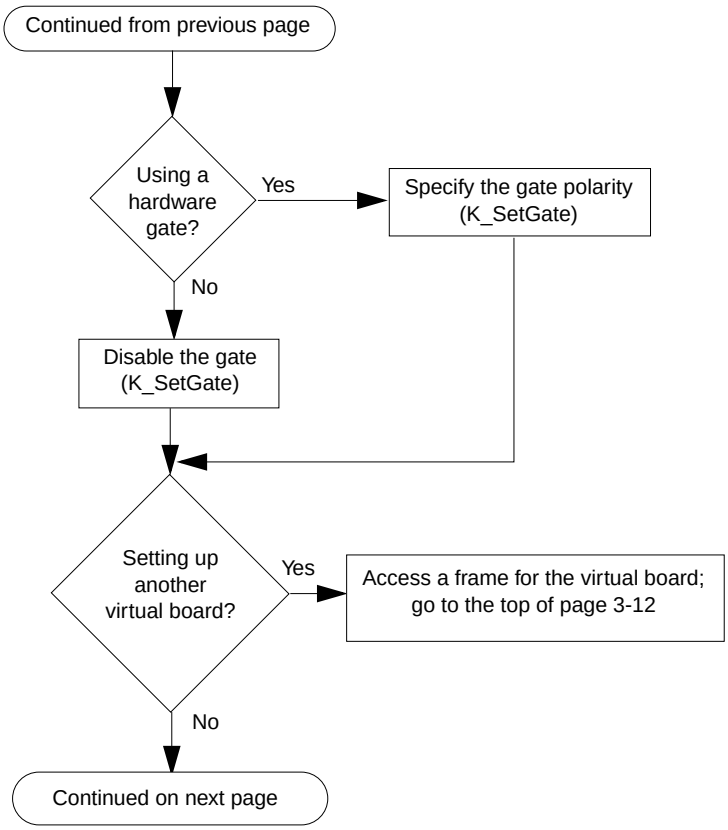
Steps for a DMA-Mode Analog Input Operation (cont.)



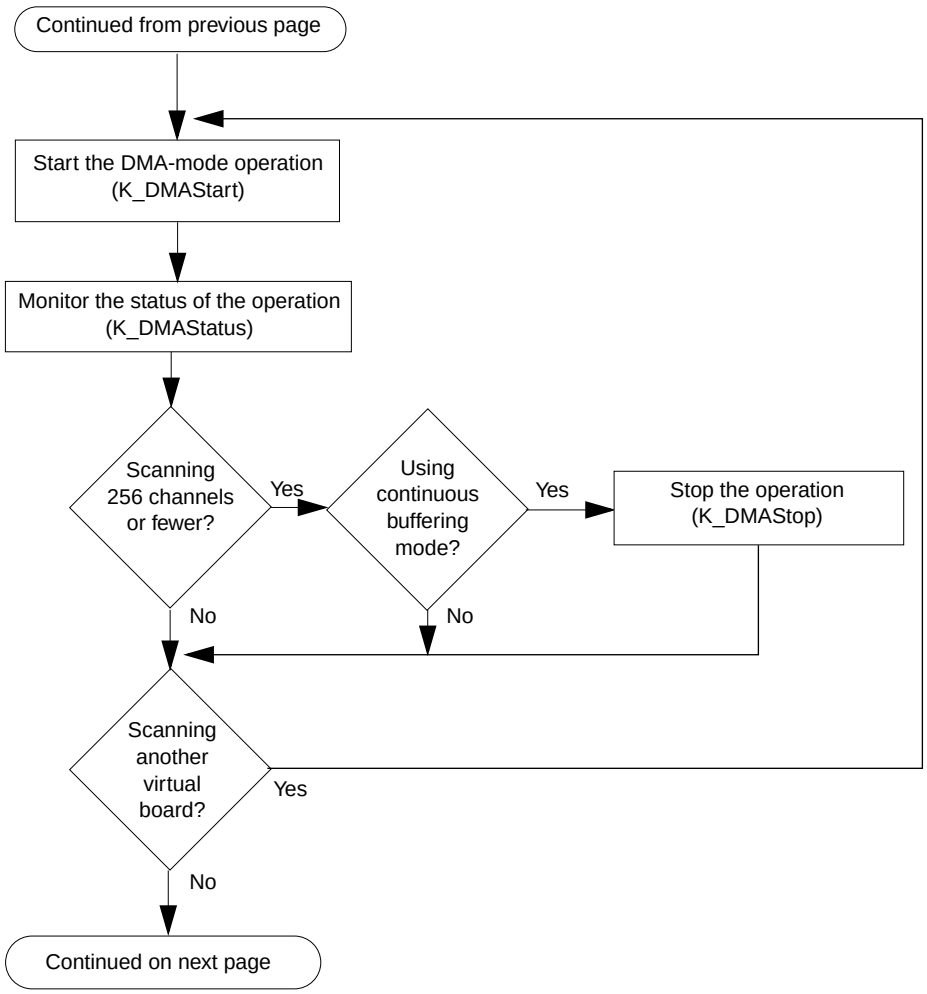
Steps for a DMA-Mode Analog Input Operation (cont.)



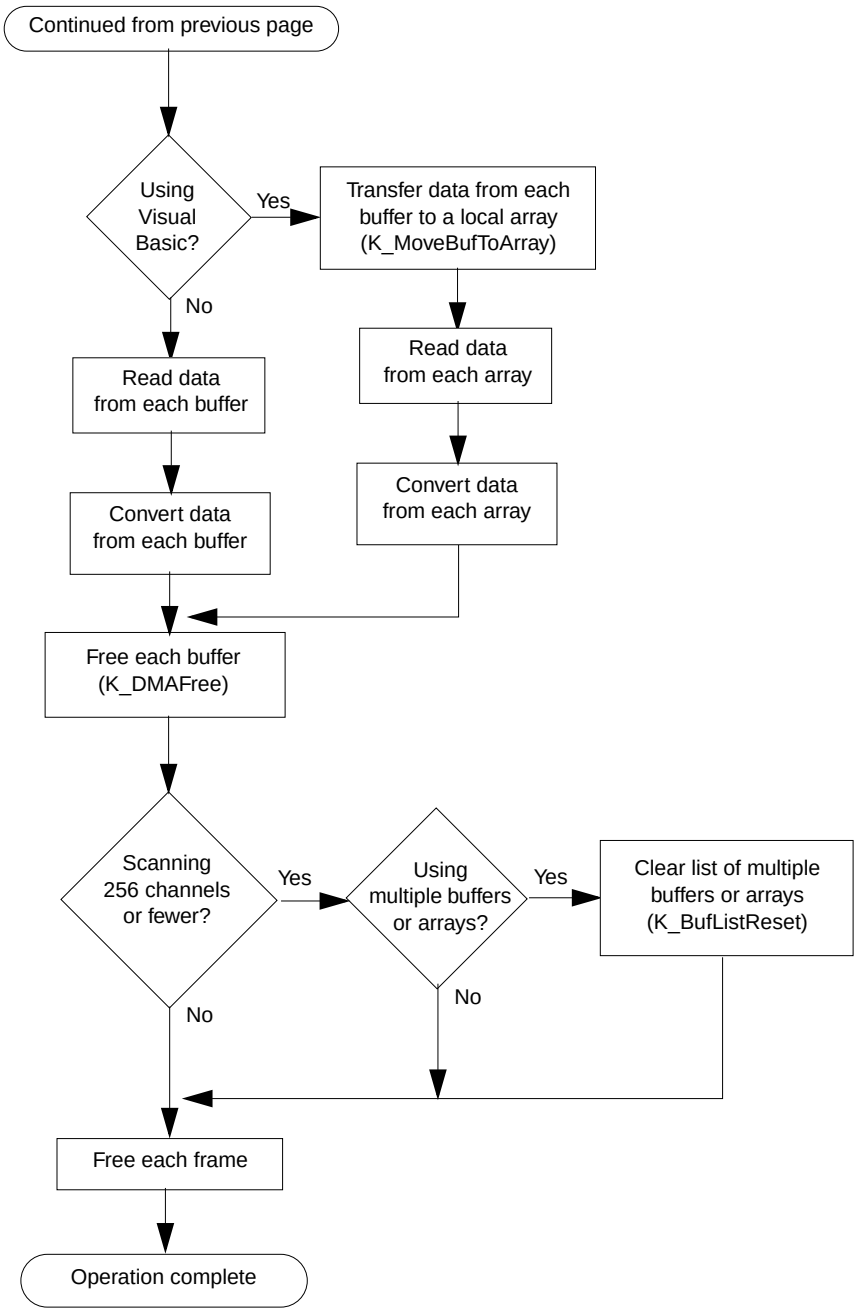
Steps for a DMA-Mode Analog Input Operation (cont.)



Steps for a DMA-Mode Analog Input Operation (cont.)



Steps for a DMA-Mode Analog Input Operation (cont.)



Performing an Interrupt-Mode Operation

If DMA resources are not available, you can perform an analog input operation in interrupt mode. In interrupt mode, the SCAN-AD-HR board acquires a single sample or multiple samples from one or more logical channels (up to a maximum of 256) on the same virtual board. A hardware clock initiates conversions. Once the analog input operation begins, control returns to your application program. The hardware temporarily stores the acquired data in the FIFO on the SCAN-AD-HR board and then transfers the data to a user-defined buffer in computer memory using an interrupt service routine.

Use the **K_IntStart** function to start the interrupt-mode operation on the first virtual board, specifying the handle of the frame that defines the first operation. After the SCAN-AD-HR board reads a sample from each logical channel associated with the virtual board and stores the samples in the user-defined buffer, the operation stops. (Use the **K_IntStatus** function to monitor the status of the operation and determine when the operation stops.) Then, use **K_IntStart** again to start the interrupt-mode operation on the next virtual board, specifying the handle of the frame that defines the next operation.

Refer to page 3-6 for the procedure to follow when performing an interrupt-mode operation. In addition, example program 1, provided in the ASO-SCAN software package, illustrates how to acquire data in interrupt mode.

The data in the user-defined buffer is stored as count values. Refer to Appendix A for information on converting the count value to voltage or degrees.

Notes: Because of the overhead required to stop one interrupt-mode operation and start another, a slight delay occurs before the DAS-Scan system begins acquiring data on each subsequent virtual board.

Typically, you want your interrupt-mode operation to stop automatically after one sample from each logical channel has been read; this is referred to as single-cycle buffering mode. However, like DMA mode, if you are acquiring data from 256 logical channels or fewer, you can also perform the operation in continuous buffering mode. Continuous buffering mode allows you to continuously acquire data, overwriting any values already stored in memory. Refer to page 3-23 for more information.

In most cases, the information in Chapter 2 that applies to a DMA-mode operation also applies to an interrupt-mode operation. The following sections describe additional considerations and exceptions to keep in mind when performing an interrupt-mode operation.

Accessing a Frame

Like a DMA-mode operation, an interrupt-mode operation requires an A/D frame. Refer to page 2-7 for information about accessing a frame.

For interrupt mode, use the **K_IntStart** function (instead of the **K_DMAStart** function) to start the operation you set up.

Reserving Memory

Like a DMA-mode operation, an interrupt-mode operation requires memory in which to store the acquired data. The recommended way to reserve memory for an interrupt-mode operation is to dynamically allocate a memory buffer for each virtual board. Refer to page 2-9 for more information.

For interrupt mode, use the **K_IntAlloc** function (instead of the **K_DMAAlloc** function) to allocate the buffer. When you no longer require the buffer, use the **K_IntFree** function (instead of the **K_DMAFree** function) to free the buffer for another use.

Notes: If you dynamically allocate a buffer of 256 samples (the typical size), an interrupt is generated after each sample and the samples are transferred one-by-one from the FIFO to the user-defined buffer in computer memory. If your computer cannot service interrupts fast enough, a FIFO overflow error may occur. If you dynamically allocate a buffer of 512 samples (half of a FIFO), or any multiple of 512 samples, an interrupt is generated after 512 samples are acquired and the entire block of 512 samples is transferred from the FIFO to the user-defined buffer in computer memory. To optimize an interrupt-mode operation, make sure that the size of your buffer is a multiple of 512 samples.

If you are writing Windows 95, 32-bit programs, you must install the Keithley Memory Manager. Refer to the *DAS-Scan User's Guide* for information.

For interrupt mode, in addition to reserving memory by dynamically allocating a buffer, you can reserve memory by dimensioning an array within your application program's memory area for each virtual board. A local array is directly accessible to your application program; for Visual Basic For Windows, you do not have to use the **K_MoveBufToArray** function to move the data from the dynamically allocated buffer to the program's local array. However, unlike dynamically allocated buffers, which can be freed to make them available to other programs or processes, local arrays occupy permanent memory areas and cannot be freed. In addition, you cannot use local arrays with Windows 95, 32-bit programs.

Typically, a single array or buffer that can hold one sample for each logical channel associated with the virtual board is sufficient. However, like DMA mode, if you are acquiring data from 256 logical channels or fewer, you can dimension a larger array/buffer or multiple arrays/buffers (up to a maximum of 150), allowing you to sample each logical channel multiple times. Use the **K_IntAlloc** function to dynamically allocate a memory buffer of up to 5,000,000 samples.

If you are using large or multiple dynamically allocated memory buffers, you may be limited in the amount of memory you can allocate. It is recommended that you install the Keithley Memory Manager before you begin programming to ensure that you can allocate a large enough buffer or buffers. Refer to the *DAS-Scan User's Guide* for more information about the Keithley Memory Manager.

Table 3-2 lists the functions used in interrupt mode to assign the starting addresses of local arrays or dynamically allocated buffers and to specify the number of samples to store in the buffers or arrays.

Table 3-2. Functions Used to Assign Starting Addresses in Interrupt Mode

Language	Type of Arrays or Buffers	Function
C/C++	Single local array	K_SetBuf
	Multiple local arrays	K_BufListAdd
	Single dynamically allocated buffer	K_SetBuf
	Multiple dynamically allocated buffers	K_BufListAdd
Visual Basic for Windows	Single local array	K_SetBufI
	Multiple local arrays	K_BufListAdd
	Single dynamically allocated buffer	K_SetBuf
	Multiple dynamically allocated buffers	K_BufListAdd

Specifying Channels and Gains

Like a DMA-mode operation, for an interrupt-mode operation, you can specify the logical channels on which to perform the operation and the gain at which to read the logical channels. Refer to page 2-11 for information about specifying the gain; refer to page 2-12 for information about specifying the logical channels.

Specifying a Pacer Clock

Like a DMA-mode operation, for an interrupt-mode operation, you can use a pacer clock to determine the period between the conversion of one logical channel and the conversion of the next logical channel in a scan of multiple logical channels. Refer to page 2-19 for more information.

Typically, using the pacer clock alone is sufficient. However, like DMA mode, if you are acquiring data from 256 logical channels or fewer, you can use both the pacer clock and the burst mode conversion clock to perform your operation in burst conversion mode. Refer to page 3-25 for more information about specifying burst conversion mode. Refer to page 3-26 for more information about using the burst mode conversion clock.

Specifying a Trigger

Like a DMA-mode operation, for an interrupt-mode operation, you can specify a start trigger for the first virtual board to determine when acquisitions start. Refer to page 2-22 for more information.

Note that unlike DMA mode, in interrupt mode, you cannot use the about trigger to perform pre-trigger or about-trigger acquisitions.

Enabling a Hardware Gate

Like a DMA-mode operation, for an interrupt-mode operation, you can enable a hardware gate to determine when conversions can occur. Refer to page 2-24 for more information.

Specifying Continuous Mode

If you are reading 256 logical channels or fewer, you can specify continuous buffering mode. In continuous mode, the SCAN-AD-HR board continuously converts samples and stores them in memory until it receives a stop function. Any values already stored in memory are overwritten. Use the **K_SetContRun** function to specify continuous buffering mode.

If you specify the logical channels you want to read in either a group of consecutive logical channels or in a channel-gain queue, the specified channels are continuously sampled until the specified number of samples is read. For example, in a group of consecutive logical channels, the first logical channel is 14, the last logical channel is 17, and you want to acquire five samples. Your program reads data first from logical channel 14, then from logical channels 15, 16, and 17, and finally from logical channel 14 again.

Notes: If you specify continuous buffering mode for an operation on a virtual board and later want to reset the buffering mode to single-cycle mode, use the **K_ClrContRun** function. In single-cycle mode, after the SCAN-AD-HR board converts one sample from each logical channel on the virtual board and stores the samples in memory, the operation stops automatically.

You cannot use continuous mode if you enable the generation of Windows events. Refer to page 2-3 for more information about generating a Windows event.

Reserving Large or Multiple Memory Buffers

Reserving a large memory buffer or multiple memory buffers is useful if you want to read each logical channel on a virtual board multiple times to average the data read from the channels.

If you are performing a DMA-mode analog input operation and you are reading 256 logical channels or fewer, you can use the **K_DMAAlloc** function to dynamically allocate a memory buffer of up to 65,536 samples. You can also dynamically allocate multiple memory buffers (up to a maximum of 150).

If you are using large or multiple dynamically allocated memory buffers, you may be limited in the amount of memory you can allocate. It is recommended that you install the Keithley Memory Manager before you begin programming to ensure that you can allocate a large enough buffer or buffers. Refer to the *DAS-Scan User's Guide* for more information about the Keithley Memory Manager.

Use the **K_SetDMABuf** function to assign the starting address of a single dynamically allocated buffer.

Use the **K_BufListAdd** function to assign the starting addresses of multiple dynamically allocated buffers. Use the **K_BufListReset** function to clear the list of multiple buffers or arrays when you are through with them.

Specifying Burst Mode

If you are reading 256 logical channels or fewer, you can specify burst conversion mode. Burst mode is useful if you want a single external event (such as the pulse of an external pacer clock) to start conversions of all the channels in a scan.

In burst mode, you use both the pacer clock and the burst mode conversion clock. The pacer clock determines the period between the conversions of one scan and the conversions of the next scan; the burst mode conversion clock determines the period between the conversion of one logical channel in a scan and the conversion of the next logical channel in the scan. Refer to page 2-19 for information about the pacer clock; refer to the next section for information about the burst mode conversion clock. Use the **K_SetADFreeRun** function to specify burst mode.

Notes: It is recommended that you do not use burst mode if a virtual board includes SCAN-BRD-V-ISO or SCAN-BRD-TC-ISO isolated assemblies.

If you specify burst conversion mode for an operation on a virtual board and later want to reset the conversion mode to paced mode, use the **K_ClrADFreeRun** function. In paced mode, the pacer clock determines the period between the conversion of one logical channel in a scan and the conversion of the next logical channel in the scan.

Using the Burst Mode Conversion Clock

If you are reading 256 logical channels or fewer in burst conversion mode, you use the burst mode conversion clock to determine the period between the conversion of one logical channel in a scan and the conversion of the next logical channel in the scan.

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

You can specify from 10 clock ticks (100 kHz) to 64 clock ticks (15.625 kHz). The period between conversions ranges from 10 µs to 64 µs.

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

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

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

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

Specifying Pre-Trigger Acquisition

If you are performing a DMA-mode analog input operation and you are reading 256 logical channels or fewer, you can use pre-trigger acquisition to acquire data before a specific trigger event.

You specify both a start trigger and an about trigger. The start trigger determines when acquisitions start on a virtual board and can be either an internal trigger or an external digital trigger. The about trigger is always an external digital trigger; acquisitions stop on the virtual board when the about-trigger event occurs. Refer to page 2-23 for information about an internal trigger; refer to page 2-23 for information about an external digital trigger.

To specify pre-trigger acquisition, perform the following steps:

1. Specify the start trigger.
   - Use **K_SetTrig** to specify an internal or an external trigger source.
   - If you specify an external start trigger in **K_SetTrig**, use **K_SetDITrig** to specify the active edge for the digital trigger.
2. Use **K_SetAboutTrig** to enable the about trigger and to set the number of samples to 1.

   Note: The minimum number of samples that you can specify in **K_SetAboutTrig** is 1.

   ---

3. Specify the active edge for the about trigger, if necessary.
   - If the start trigger is an internal trigger, use **K_SetDITrig** to specify the active edge for the about trigger.
   - If the start trigger is an external digital trigger, the active edge used for the start trigger is also used for the about trigger. It is not necessary to use **K_SetDITrig** again.

Notes: If you specify pre-trigger acquisition for an operation on a virtual board and later want to reset the operation for post-trigger acquisition, use the **K_ClrAboutTrig** function to disable the about trigger.

Pre-trigger acquisition is not available with interrupt-mode operations.

Specifying About-Trigger Acquisition

If you are performing a DMA-mode analog input operation and you are reading 256 logical channels or fewer, you can use about-trigger acquisition to acquire data both before and after a specific trigger event.

You specify both a start trigger and an about trigger. The start trigger determines when acquisitions start on a virtual board and can be either an internal trigger or an external digital trigger. The about trigger is always an external digital trigger; acquisitions stop after a specified number of samples has been acquired on the virtual board after the about-trigger event occurs. Refer to page 2-23 for information about an internal trigger; refer to page 2-23 for information about an external digital trigger.

To specify about-trigger acquisition, perform the following steps:

1. Specify the start trigger.
   - Use **K_SetTrig** to specify an internal or an external trigger source.
   - If you specify an external start trigger in **K_SetTrig**, use **K_SetDITrig** to specify the active edge for the digital trigger.
2. Use **K_SetAboutTrig** to enable the about trigger and to specify the desired number of post-trigger samples.
3. Specify the active edge for the about trigger, if necessary.
   - If the start trigger is an internal trigger, use **K_SetDITrig** to specify the active edge for the about trigger.
   - If the start trigger is an external digital trigger, the active edge used for the start trigger is also used for the about trigger. It is not necessary to use **K_SetDITrig** again.

Notes: If you specify about-trigger acquisition for an operation on a virtual board and later want to reset the operation for post-trigger acquisition, use the **K_ClrAboutTrig** function to disable the about trigger.

About-trigger acquisition is not available with interrupt-mode operations.

A

Data Formats

This appendix contains the following sections:

- **Converting Counts to Voltage** - instructions for converting a count value returned by the DAS-Scan Function Call Driver to a voltage value.
- **Converting Counts to Temperature** - instructions for converting a count value returned by the DAS-Scan Function Call Driver to a temperature value.

Converting Counts to Voltage

The DAS-Scan Function Call Driver can read count values only. When reading an analog input value (as in **K_ADRead**), you can convert the count value returned by the DAS-Scan Function Call Driver to a voltage value. If you are using a SCAN-BRD-TC or SCAN-BRD-TC-ISO assembly and want to express the value in degrees, you can then convert the voltage value to a temperature value; refer to the next section for information.

To convert an analog input value to a voltage, use the following equation, where *count* is the count value and *span* is the span in volts (refer to Table A-1):

$$\text{Voltage} = \frac{\text{count} \times \text{span}}{65536}$$

Table A-1. Span Values For Data Conversion Equations

Input Range Type	Gain	Analog Input Range	Span (V)
Unipolar	1	0 to 10 V	10
	2	0 to 5 V	5
	4	0 to 2.5 V	2.5
	8	0 to 1.25 V	1.25
	50	0 to 0.2 V	0.2
	100	0 to 0.1 V	0.1
	200	0 to 50 mV	0.05
	400	0 to 25 mV	0.025
Bipolar	1	–10 to 10 V	20
	2	–5 to 5 V	10
	4	–2.5 to 2.5 V	5
	8	–1.25 to 1.25 V	2.5
	50	–0.2 V to 0.2 V	0.4
	100	–0.1 V to 0.1 V	0.2
	200	–50 mV to 50 mV	0.1
	400	–25 mV to 25 mV	0.05

For example, assume that you want to read analog input data from a channel configured for a bipolar input range type; the channel collects the data at a gain of 2. The count value is 1024. The voltage is determined as follows:

$$\frac{1024 \times 10\ \text{V}}{65536} = 0.156\ \text{V}$$

Converting Counts to Temperature

If you are using a SCAN-BRD-TC or SCAN-BRD-TC-ISO assembly, you can convert the count value returned by the DAS-Scan Function Call Driver to a temperature value. Refer to example program 4, provided in the ASO-SCAN software package, as you perform the procedure described in this section.

To convert a count value to a temperature value, perform the following steps:

1. Convert the thermocouple count value returned by the DAS-Scan Function Call Driver to thermocouple voltage. Refer to page A-1 for information.
2. If you are using the CJC channels, convert the CJC count value returned by the DAS-Scan Function Call Driver to CJC voltage, using a gain of 1. Refer to page A-1 for information.
3. Convert the CJC voltage to °K by dividing the CJC voltage by 0.010, and then convert the °K to °C.

Note: Because the CJC sensors used on the DAS-Scan system generate 10 mV per °K, you must divide the CJC voltage by 0.010.

---

4. Adjust the thermocouple voltage to take into account the CJC temperature value, using the appropriate intercept and slope values.

Notes: The thermocouple voltage is a relative value, based on the difference between the voltage at the end of the wire and the voltage at the screw terminal panel; the CJC temperature is an absolute value. You must first convert the CJC temperature to mV, based on the thermocouple type you are using, and then add the two voltage values together.

The intercept and slope values used in the example program assume that the temperature of the CJC sensor is approximately 25°C. If the temperature of your CJC sensor is different, use the appropriate N.I.S.T thermocouple table to determine the appropriate intercept value, and substitute an appropriate slope value.

---

5. Calculate the index into the thermocouple lookup table, using the voltage step interval (mV) and starting voltage (mV) from the [header] section of the appropriate table. Lookup tables for J, K, T, E, R, S, and B type thermocouples are provided in the ASO-SCAN software package.
6. Interpolate between the two nearest entries in the lookup table to determine the true reading in °C.

Index

F

G

H

I

K

L

M

O

P

R

S

T

U

V

W