ADC-16
Analog Input Board
User Guide

KEITHLEY

User Guide

for the

ADC-16

Analog Input

Board

Revision B - December 1992

Part Number: 24446

KEITHLEY DATA ACQUISITION - Keithley MetraByte/Asyst

Warranty Information

All products manufactured by Keithley Data Acquisition are warranted against defective materials and workmanship for a period of one year from the date of delivery to the original purchaser. Any product that is found to be defective within the warranty period will, at the option of the manufacturer, be repaired or replaced. This warranty does not apply to products damaged by improper use.

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 suitable for use in life support or critical applications.

Disclaimer

Information furnished by Keithley Data Acquisition is believed to be accurate and reliable. However, Keithley Data Acquisition assumes no responsibility for the use of such information nor for any infringements of 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.

Note:

Keithley MetraByte™ is a trademark of Keithley Instruments.

Basic™ is a trademark of Dartmouth College.

IBM® is a registered trademark of International Business Machines Corporation.

PC, XT, AT, PS/2, and **Micro Channel Architecture**® are trademarks of International Business Machines Corporation.

Microsoft® is a registered trademark of Microsoft Corporation.

Turbo C® is a registered trademark of Borland International.

Contents

Contents

■ ■ ■

CHAPTER 1

INTRODUCTION

1.1 FUNCTIONAL OVERVIEW

General

The ADC-16 is a low-cost, high-resolution (16 bit), analog input board for ISA and EISA bus computers (IBM PC/XT/AT or compatible). An integrating A/D converter performs up to 16 conversions/sec while ensuring repeatability in noisy environments. Eight differential input channels offer 16-bit resolution (15 bit + sign). Figure 1-1 is a block diagram of the ADC-16.

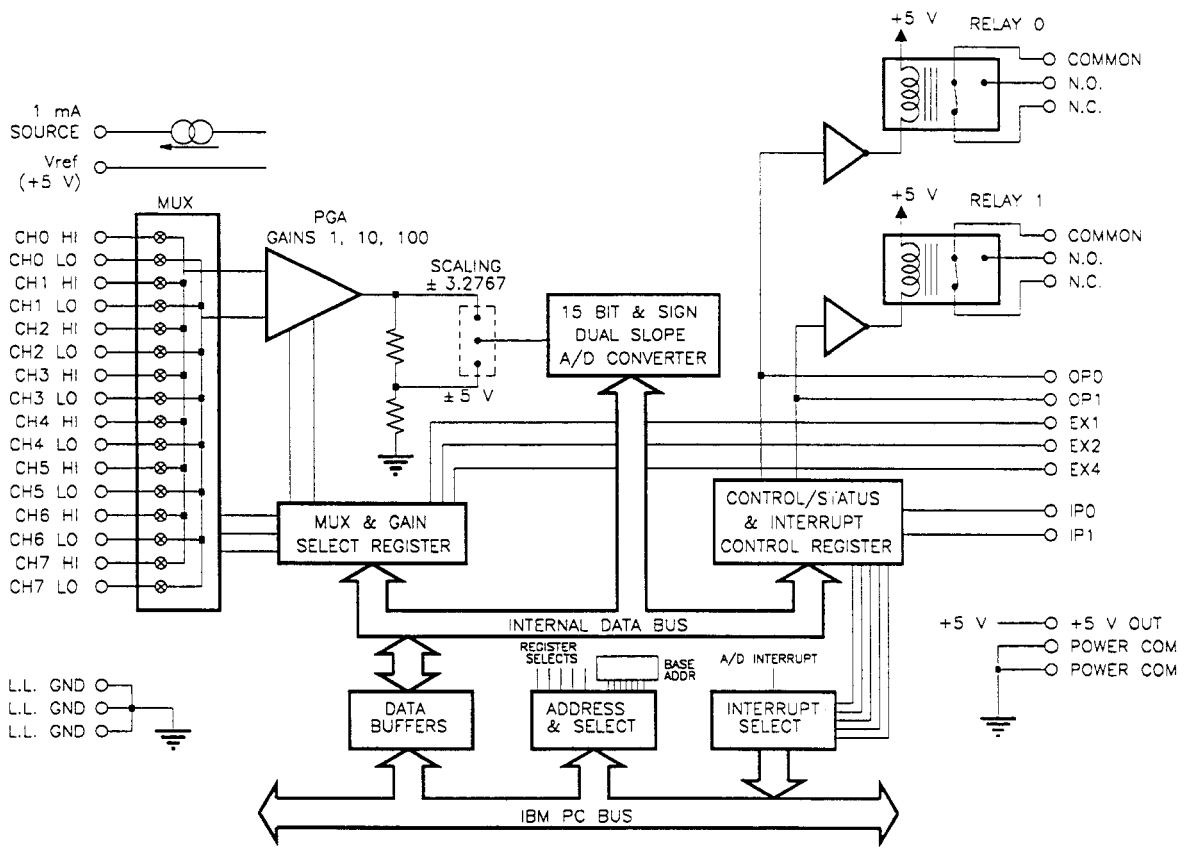


Figure 1-1. Block diagram of the ADC-16.

Gains of 1, 10, and 100 are software-programmable. A single gain adjustment sets the full scale of every range. No additional adjustments are required since the zero of every range is automatically corrected.

Additional features include two TTL compatible, general-purpose digital outputs and two general-purpose TTL/CMOS-compatible digital inputs. The digital outputs also drive two internal Form C relays on the ADC-16. These relays are available for switching and expansion applications.

Three more digital outputs become available when the optional STA-EX8 expansion multiplexer(s) is not used. A 1 mA, high-precision current source with a compliance of -10 V to +4 V allows excitation of resistance-based transducers such as RTDs. All connections to the ADC-16 pass through a standard 37-pin D connector using the manufacturer's C-1800 cable.

Optional equipment for the ADC-16 includes two screw-termination boxes: the STA-U and the STA-EX8. The STA-U provides access to all ADC-16 interface signals via miniature screw-terminal connectors. The STA-EX8 provides access to all ADC-16 interface signals via miniature screw-terminal connectors, and it provides eight additional input channels. Up to eight STA-EX8s may be added to an ADC-16, resulting in 64 fully differential channels.

Operating Modes

A/D Transfers

The ADC-16 conducts A/D conversions on a self-timing basis; the conversions progress as quickly as the A/D circuitry can operate. When a previous conversion ends, the Driver starts another. This process continues until the scan is complete.

When a STart/Stop array is in use, the number of conversions is equal to the number of channels specified in the array. This number can be up to 64 if eight STA-EX8s are in the system.

When a Channel/Gain array is in use, the number of conversions is equal to the number of entries in the array. This number can be up to 256.

There are three modes of A/D operations:

SYNCHRONOUS MODE - where the transfers are occur in the foreground, forcing the user to wait for the board to finish. This mode is invoked by the K_SyncStart call (see Chapters 6 and 7).

INTERRUPT MODE - where the transfers occur in the background, allowing the user to execute code until the board finishes. This mode is invoked by the K_IntStart call and tested for completion by the K_IntStatus call (see Chapters 6 and 7).

These two modes require that channel and gain information be set into Frames using the K_SetBuf and K_SetChanGary or K_SetStartStopG calls. For more information on Frames, refer to Section 6.6.

IMMEDIATE MODE - where when a single A/D operation takes place, all information is passed via the Call parameters rather than by Frames, as in the two previous modes.

Digital Operations

Digital Input and Output operate *only* in the Immediate Mode, as described above for *Immediate Mode*

1.2 SOFTWARE OVERVIEW

This manual refers to the ADC-16 software as the *Distribution Software.* The Distribution Software contains a software-installation program, driver files, utility files, calibration files, and programming example files. For a list of these files, with descriptions, refer to the ASCII file *FILES.DOC.*

Because the Distribution Software arrives in compressed form, it is important that you follow the installation instructions in Section 2.4. You must use the installation program to uncompress these files before you can access FILES.DOC.

The driver software supports the *Pop-Up Control Panel* and the *Call Driver*, which are two of the three interface options available for controlling the ADC-16. The third control interface is *Register-Level I/O* programming. Chapter 3 describes of each of these interfaces.

Additional programming options are available in the *Advanced Software Option (ASO)*, which includes the File I/O Command Driver, the Call Driver for C and Pascal, and the Dynamic Link Library for Windows 3.X languages. Contact the manufacturer for information on this option.

NOTE: The *README.DOC* file (in the Distribution Software), contains last-minute information not included in this manual. README.DOC is a readable ASCII file.

1.3 ACCESSORIES

STA-EX8	8-channel expansion multiplexer.
STA-U	Standard Screw Terminal Board.
C-1800	Cable to connect ADC-16 to STA-EX8 or STA-U.
ASO-ADC-16	Advanced Software Option for the ADC-16. This option includes Call Driver for C and Pascal, the Dynamic Link Library for Windows 3.X, and the File I/O Driver. A user manual is also included.

1.4 SPECIFICATIONS

ADC-16 Board

A/D

Channels: 8 differential, expandable to 64 with STA-EX8 Boards.

Input Resolution: 16 bits (15 plus a sign bit).

Coding: Sign plus Magnitude (binary).

Input Ranges: ±5 V or ±3.2768 V Full Scale (jumper selectable).

Input Gains: 1, 10, or 100 (software- or jumper-selectable).

Sample Rate: 16 samples/second.

Input Settling Time: 50 µs.

Input Offset: Auto-zeroed; ±10 µV for Gain = 100, ±1 LSB for Gain = 1 or 10.

Absolute Accuracy

Gain = 1:	±0.01% of range typical	±0.03% maximum.
Gain = 10:	±0.05% of range typical	±0.10% maximum.
Gain = 100:	±0.05% of range typical	±0.15% maximum.

Relative Accuracy

	Gain = 1,10, and 100 maximum.	±0.005% of range typical ±0.012%

Noise (Typical)

Gain = 1:	< ±1 bit rms.
Gain = 10:	< ±1 bit rms.
Gain = 100:	< ±3 bits rms.

Input Impedance: Greater than 100 MegOhms.

Input Bias Current: 50 nA max.

Common Mode Rejection: Gain = 1 - 100 dB typ, 80 dB min.
Gain = 10 - 110 dB typ, 86 dB min.
Gain = 100 - 120 dB typ, 92 dB min.

Common Mode Range: ±6 V.

Max Input Volts w/o Damage: Power On: ±35 VDC.

Power Off: ±20 VDC.

DIGITAL I/O

Digital Inputs

Number of Inputs: 2, TTL/CMOS compatible.
Logic Type: High input returns a 1.
Logic Levels: Vil = .8 V, Vih = 2.0 V.
Iil = -.2 mA, Iih = 20 uA.

Digital Outputs

Number of Outputs: 5 TTL compatible.
Logic Type: Vol = .5 V max at 8.5 mA.
Voh = 2.7 V min at -0.4 mA.

Relay Outputs

Number of Channels: 2 Form C.
Max Current: 2.0 A at 28 Vrms (resistive).

POWER REQUIREMENTS

+5 V:	800 mA typ, 1 A max.
+12 V:	25 mA max.
-12 V:	15 mA max.

ENVIRONMENTAL

Operating Temperature: 0 to 70 °C.
Storage Temperature: -25 to 85 °C.
Humidity: 0 to 95% noncondensing.

PHYSICAL

Dimensions: 9.0 x 4.2 in. (22.9 x 10.7 cm)
Weight: 10 oz (284 g).

STA-EX8 Board

Number of Inputs: 8 differential. Each STA-EX8 board uses one ADC-16 channel; up to 8 STA-EX boards will work with a single ADC-16.

Input Offset: Auto-zeroed; ±10 µV for Gain = 100, ±1 LSB for Gain = 1 or 10.

ACCURACY

Gain = 1: ±.01% of Full Scale.
Gain = 10: ±.1% typ.
Gain = 100: ±.1% typ.

NOISE (TYPICAL)

Gain = 1: <±1 bit rms.
Gain = 10: <±2 bits rms.
Gain = 100: <±15 bits rms.

Input Impedance: Greater than 100 Megohms.

Input Bias Current: 50 nA max.

Common Mode Rejection: Gain = 1 - 100 dB typ, 80 dB min.
Gain = 10 - 110 dB typ, 86 dB min.
Gain = 100 - 120 dB typ, 92 dB min.

Common Mode Range: ±6 V.

Max Input Volts w/o Damage: Power On: ±35 VDC.
Power Off: ±20 VDC.

POWER REQUIREMENTS

+5 V: 10 mA typ, 100 mA max.

+12 V: Not used.

-12 V: Not used.

ENVIRONMENTAL

Operating Temperature: 0 to 70 °C.

Storage Temperature: -25 to 85 °C.

Humidity: 0 to 95% noncondensing.

PHYSICAL

Dimensions: 6.7 x 5.4 x 2.3 in. (17.0 x 13.7 x 5.8 cm)

Weight: 11 oz (312 g).

STA-U Board

This board is simply a wiring feed-through for facilitating connections to the ADC-16. As such, the STA-U has no meaningful specifications beyond the following.

ENVIRONMENTAL

Dimensions: 6.7 x 5.1 x 2.4 in. (17.0 x 13.0 x 6.0 cm)

Operating Temperature: 0 to 70 °C.

Storage Temperature: -25 to 85 °C.

Humidity: 0 to 95% noncondensing.

■ ■ ■

CHAPTER 2

INSPECTION, CONFIGURATION, & INSTALLATION

2.1 INSPECTION

After removing the wrapped Board (ADC-16) from its outer shipping carton, proceed as follows:

1. Before unwrapping the Board, place one hand firmly on a metal portion of the computer chassis to discharge static electricity from yourself and the Board (the computer must be turned Off but grounded).
2. Carefully remove the Board from its anti-static wrapping material.
3. Inspect the Board for signs of damage. If any damage is apparent, return the Board to the factory.
4. Check the remaining contents of your package against the packing list to be sure your order is complete. Report any missing items to the factory immediately.
5. When you are satisfied with preliminary inspection, you are ready to configure the Board. Refer to the next section for configuration options.

2.2 CONFIGURATION

Switch & Jumper Settings

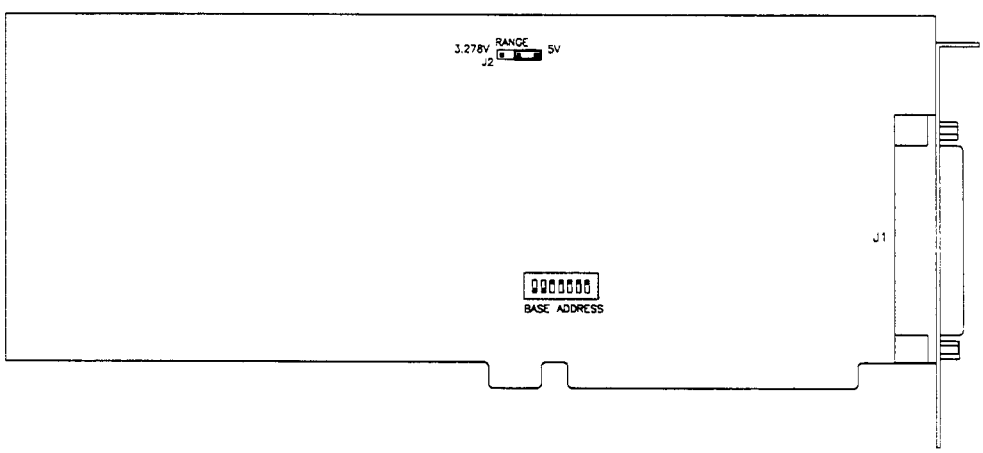


Figure 2-1. Location of Base Address switch and Input Range jumper.

The Base Address Switch

The factory-set Base Address is 300h (768 decimal). If this address is already in use, change the setting of the Base Address Switch (refer to Figure 2-2 for the address-setting arrangement of the Switch). Any new Base Address setting must be within the range of 000h to 3FFh (0 to 1023 Decimal) on an 8-byte boundary. Use the Table 2-1 as an aid to selecting a Base Address.

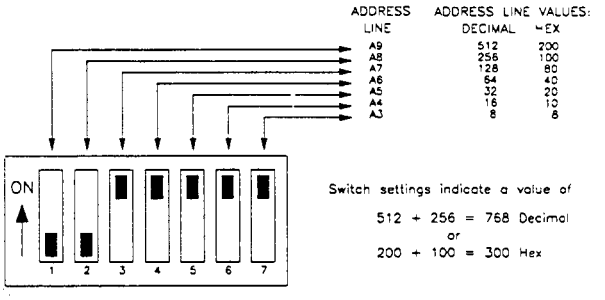


Figure 2-2. Base Address Switch

Table 2-1. PC/XT/AT I/O Address Space

HEX RANGE	USAGE	HEX RANGE	USAGE
000-00F	8237 DMA #1	2C0-2CF	EGA
020-021	8259 PIC #1	2D0-2DF	EGA
040-043	8253 Timer	2E0-2E7	GPIB (AT)
060-063	8255 PPI (XT)	2E8-2EF	Serial Port
060-064	8742 Controller (AT)	2F8-2FF	Serial Port
070-071	CMOS RAM & NMI Mask Reg. (AT)	300-30F	Prototype Card
080-08F	DMA Page Registers	310-31F	Prototype Card
0A0-0A1	8259 PIC #2 (AT)	320-32F	Hard Disk (AT)
0A0-0AF	NMI Mask Register (XT)	378-37F	Parallel Printer
0C0-0DF	8237 DMA #2 (AT)	380-38F	SDLC
0F0-0FF	Coprocessor	3A0-3AF	SDLC
1F0-1FF	Hard Disk (AT)	3B0-3BB	MDA
200-20F	Game/Control	3BC-3BF	Parallel Printer
210-21F	Expansion Unit (XT)	3C0-3CF	EGA
238-23B	Bus Mouse	3D0-3DF	CGA
23C-23F	Alt. Bus Mouse	3E8-3EF	Serial Port
278-27F	Parallel Printer	3F0-3F7	Floppy Disk
2B0-2BF	EGA	3F8-3FF	Serial Port

The Jumper

The only Board jumper is a 2-position arrangement for setting the A/D Full Scale Range. This jumper is set and calibrated by the factory for a Full Scale Range of ±3.2768 V, as shown in Figure 2-3. To change to the ±5 V Range, set the jumper accordingly and recalibrate the Board. See Chapter 9 for calibration instructions.

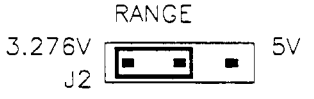


Figure 2-3. J2 jumper shown with factory setting.

The ±3.2768 V scaling has the advantage of round-number bit resolution (100, 10, or 1 uV/bit), which facilitates scaling from the binary output of the converter. The ±5 V scaling has the advantage of matching most commercial signal sources and transducers (10V with 2/1 attenuator). Table 2-2 lists the Full Scale/Gain ranges and their corresponding bit resolutions.

Table 2-2. Input Range vs. Bit Resolutions

	3.276 V INPUT RANGE		5 V INPUT RANGE	
GAIN	RANGE	RESOLUTION	RANGE	RESOLUTION
1	±3.2768 V	100 uV	±5 V	152.6 uV
10	±327.68 mV	10 uV	±500 mV	15.26 uV
100	±32.768 mV	1 uV	±50 mV	1.526 uV

2.3 THE MAIN I/O CONNECTOR J1

Analog and Digital I/O use a 37-pin, D-type connector accessible from the rear of the computer. Figure 2-4 shows the connector and its pinouts.

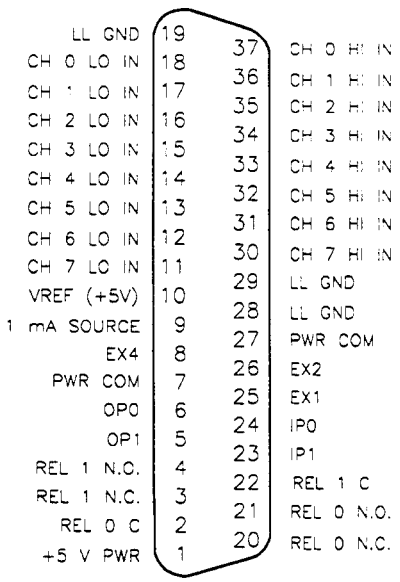


Figure 2-4. Main I/O Connector

2.4 SOFTWARE INSTALLATION

Backing Up The Distribution Software

As soon as possible, make a back-up copy of your Distribution Software. For the back-up copy, be sure to have one (or more,as needed) formatted diskettes on hand. First, place your Distribution Software diskette in your PC's A Drive and log to that drive by typing **A:** . Then, use the DOS *COPY* or *DISKCOPY* command, as described in your DOS reference manual (DISKCOPY is preferred because it copies diskette identification, too).

Installing The Distribution Software

Your Distribution Software is usable only after it is installed on the hard drive of your PC. For installation, insert the Distribution Software diskette (Diskette #1) in the A Drive, and type **INSTALL <Enter>** . Then follow the prompts. The software requires approximately 620 KB of hard-drive space.

The README.DOC File

To learn of last-minute changes, be sure to read the ASCII file *README.DOC* . This is an ASCII text file that is readable with any text editor (word processor) or with the DOS *TYPE* command.

The FILES.DOC File

To learn the contents of your Distribution Software, refer to the ASCII file *FILES.DOC* . This file lists and describes each of the files in the Distribution Software. This is an ASCII text file that is readable with any text editor (word processor) or with the DOS *TYPE* command. FILES.DOC lists and briefly describes the contents of the Distribution Software.

2.5 THE CONFIGURATION PROGRAM

Overview

NOTE: Before you can run the Configuration Program, you must install the Distribution Software (Section 2.4).

Your ADC-16 Distribution Software includes the Configuration Program *ADC16CFG.EXE* , whose purpose is to assist you in setting up Board parameters and saving them to a new or existing configuration file. The default configuration file in your Distribution Software is *ADC16.CFG* ; however, the Configuration Program allows you to save a set of parameters to any configuration file you specify.

A configuration file is a necessary reference for Call Driver programs (Chapters 6 and 7) and for the Pop Up Control Panel. The file contains information such as the number of ADC-16s in use, Base Address, Range, Interrupt Level, and the number of STA-EX8s linked to each Board. If you bypass the configuration program, the Call Driver refers to the ADC16.CFG default file.

Factory Defaults

Factory settings for the ADC16.CFG file are as follows:

Board Number	0
Board Name	ADC16
Base Address	&H300
Range	3.2768
Number Type	SignMagnitude
Interrupt Level	7
Installed STA-EX8s	0

Running the Configuration Program

To start the configuration program, log to its directory, and type **ADC16CFG** *filename* . Then follow the on-screen instructions.

filename is the name of the configuration file to be modified; it may be an existing filename, or it may be a new filename that complies with DOS file-naming conventions. The default file is ADC16.CFG.

2.6 HARDWARE INSTALLATION

Installation Of The Board

To install the ADC-16 in a PC, proceed as follows:

1. Turn Off power to the PC and all attached equipment.

   WARNING!
   ANY ATTEMPT TO INSERT OR REMOVE ANY ADAPTER BOARD WITH COMPUTER POWER ON COULD DAMAGE YOUR COMPUTER!

2. Remove the cover of the PC.
3. Choose an available option slot. Loosen and remove the retainer screw at the top of the blank adapter plate. Then slide the plate up and out to remove.
4. Before touching the Board, place one hand on any metallic part of the PC/AT chassis (but not on any components) to discharge any static electricity from your body.
5. Make sure the Board switches have been properly set (refer to the configuration sections).
6. Align the Board connector with the desired accessory slot and with the corresponding rear-panel slot. Gently press the Board into the socket. Secure the Board in place with retainer screw for the rear-panel adapter-plate.
7. Replace the computer's cover.
8. Plug in all cords and cables. Turn the power to the computer back on. You are now ready to make any necessary system connections.

Installation Of The Expander Boxes

All connections to the ADC-16 are made through the Main I/O connector. Two interface options are the STA-U and STA-EX8 accessory boxes. If your application requires access to ADC-16 interface signals via screw terminals and eight or fewer channels of analog input use the STA-U. If your application requires additional channels (more than eight), use the STA-EX8. This section describes the installation of each. Note that the ADC-16 must already be installed for these procedures.

STA-U Installation

The STA-U allows access to ADC-16 interface signals via screw terminals. To connect the STA-U to the ADC-16, obtain a C-1800 cable. Then, proceed as follows:

1. Turn your computer off.
2. If the STA-U is enclosed, loosen the four corner screws and remove the top cover.
3. Plug one end of the C-1800 cable into the Main I/O Connector of the ADC-16.

4. Plug the other end of the C-1800 cable into one of the two 37-pin connectors on the STA-U.
5. Make any other connections the STA-U, as required by your application.
6. Replace the cover on the STA-U and tighten the screws, if desired.

STA-EX8 Installation

OVERVIEW

The STA-EX8 is a multiplexed expander box with screw connections to the ADC-16 interface signals and the eight input channels. Up to eight STA-EX8 Boards connect to a single ADC-16 to offer up to 64 fully differential channels. The STA-EX8 has two 12-screw terminal blocks (TB1 and TB2) and two 24-screw terminal blocks (TB5 and TB6) allowing access to the ADC-16 interface signals.

TB1 and TB2 provide access for digital input and power signals. TB2 provides two TTL-compatible digital outputs, two TTL/CMOS-compatible digital inputs, and access for power signals. TB1 allows direct connection to the two internal Form C relays on the ADC-16. Note that the signals through TB1 and TB2 run in parallel across the C-1800 cable; this is significant when accessing the power signals. Figure 2-5 shows the signal assignments for TB1 and TB2.

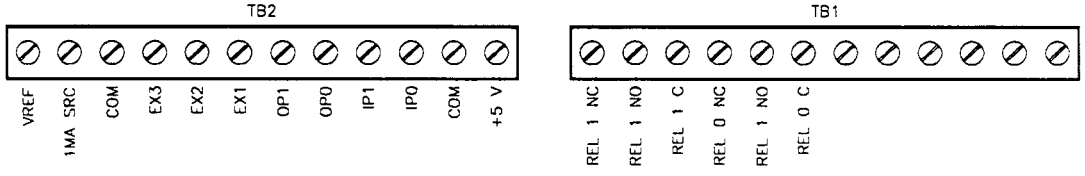


Figure 2-5. TB1 and TB2 signals.

TB5 and TB6 accommodate connections to the input channels. In addition, these two terminal blocks provide extra input channels (see section on input-channel assignments, ahead). Figure 2-6 shows the signal assignments for TB5 and TB6.

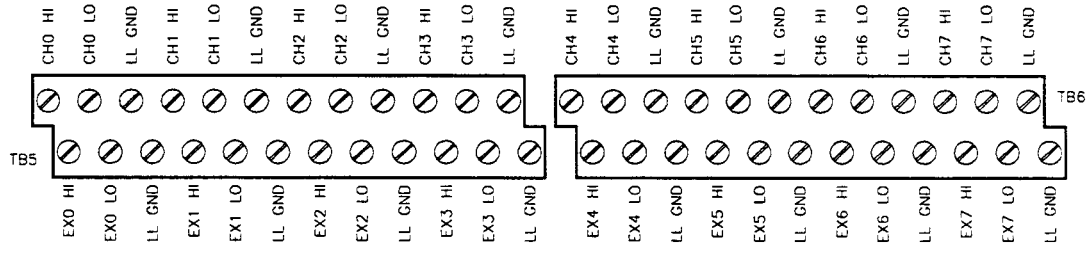


Figure 2-6. TB5 and TB6 signal assignments.

Two 37-pin connectors share pin assignments with the ADC-16 Main I/O Connector. These two connectors may be used to *daisy-chain* multiple STA-EX8s. Refer to Figure 2-4 for pinouts of these connectors.

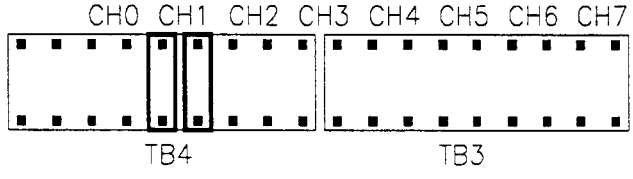


Figure 2-7. Channel Select Jumper

Finally, the Channel Select Jumper (TB3 and TB4) determines which ADC-16 input channel the STA-EX8 will use for communication. You may have up to eight STA-EX8s connected to one ADC-16. Each STA-EX8 must have its own ADC-16 input channel. For example, if your application requires two STA-EX8s, one might be assigned to Channel 0, the other to Channel 1. Figure 2-7 shows the Channel Select Jumper configured for Channel 1.

INSTALLATION PROCEDURE

Obtain the required C-1800 cables, and connect STA-EX8(s) to an ADC-16 as follows:

1. Be sure your computer is off.
2. Remove the top cover of the STA-EX8 (loosen the four corner screws and remove; then, lift the cover).
3. Set the Channel Select Jumper for an available ADC-16 input channel.
4. Plug one end of the C-1800 cable into the ADC-16's I/O Connector.
5. Plug the other end of the C-1800 cable into the first 37-pin connector (J1) on STA-EX8 #1.
6. If you are connecting more than one STA-EX8, daisy-chain the first STA-EX8 to the second. Plug one end of a second C-1800 cable into the second 37-pin connector (J2) of STA-EX8 #1. Then, connect the C-1800 to J1 on STA-EX8 #2. Repeat this step for successive STA-EX8s. (Remember, you cannot install more than eight STA-EX8s.)
7. Make all other STA-EX8 connections, as required by your application.
8. Replace the cover(s) on the STA-EX8(s).

INPUT CHANNEL ASSIGNMENTS

When programming the ADC-16 to accept the data presented on the channels added by the STA-EX8 multiplexer, you must understand the assignment of the input channels. First, the Channel Select Jumper on each STA-EX8 determines which ADC-16 input channel (CH0, CH1, CH2, CH3, etc.) the STA-EX8 will use for communication. The multiplexed channels (EX0, EX1, etc.) are then allocated consecutively from the assigned ADC-16 channel. For example, if your application has two STA-EX8s, you will assign the first STA-EX8 to ADC-16 Channel CH0. The multiplexed channels will then be allocated as follows: EX0 = Channel 0, EX1 = Channel 1, EX2 = Channel 2, ... EX7 = Channel 7. Then, you will assign the second STA-EX8 to ADC-16 Channel CH1. The multiplexed channels of the second STA-EX8 will then be EX0 = Channel 8, EX1 = Channel 9, ... EX7 = Channel 15. The remaining ADC-16 input channels will then follow EX7 = Channel 15; that is, CH2 = Channel 16, CH3 = Channel 17, ... CH7 = Channel 21. Be aware that the ADC-16 input channels run in parallel across the cable. For example, the CH2 output is associated with channel 16 on both STA-EX8s. Figure 2-8 illustrates this arrangement.

Note that the STA-EX8 multiplexed inputs are not usable by the Pop Up Control Panel.

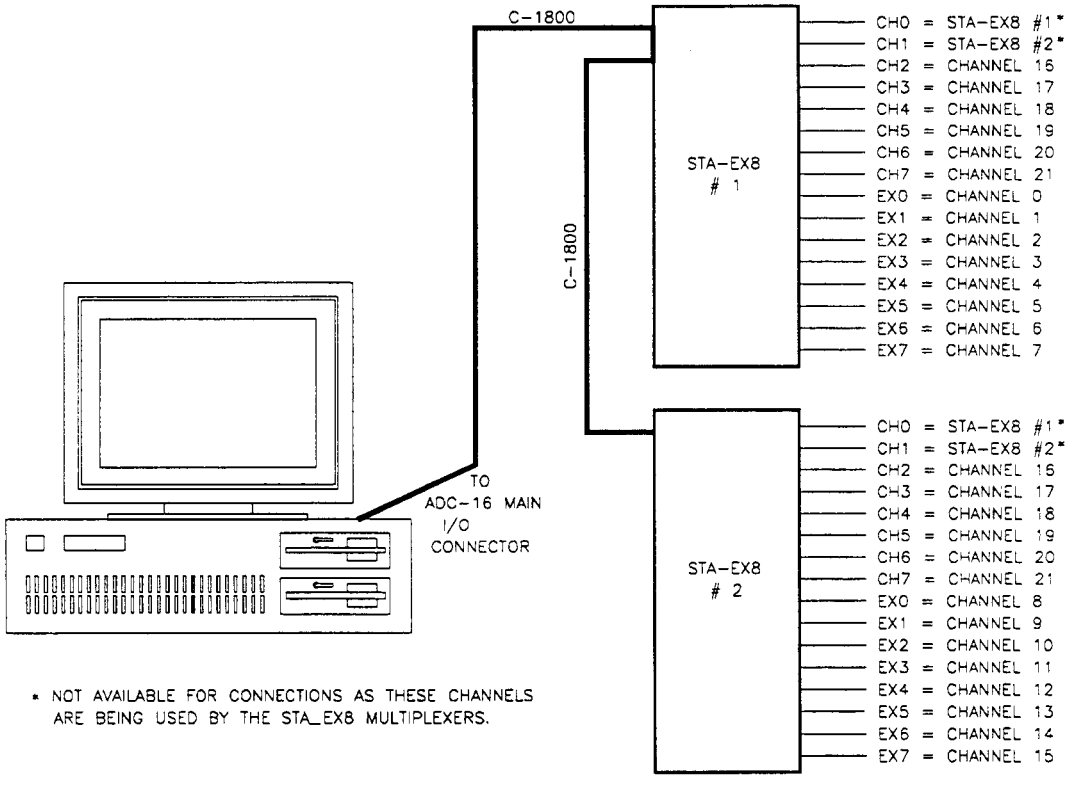


Figure 2-8. Example of Input Channel Assignments.

■ ■ ■

CHAPTER 3

OPERATION & PROGRAMMING OVERVIEW

3.1 GENERAL

The ADC-16 Distribution Software Package provides the following options:

- The Pop Up Control Panel
- The Call Driver for BASIC
- Low-level-Register I/O Programming

Additional programming options, PASCAL and C support, and Windows 3.x languages support are available in the Advanced Software Option.

3.2 THE POP UP CONTROL PANEL

The Pop Up Control Panel allows you direct control of ADC-16 operation without programming. You may configure the board to perform an analog or digital operation and to store the resultant data in a disk file. The Pop-Up uses two control panels that you may pop up with a keyboard sequence and control using either the keyboard or mouse while under DOS or inside an applications package. This Interface provides a quick way to test your board as well as to debug/monitor operation. By selecting the data logging menu, you may turn the ADC-16 into an automatic data logging system.

Chapter 4 contains instructions for loading the Pop Up Control Panel drivers. Chapter 5 covers operation.

3.3 THE CALL DRIVER

The Call Driver is a collection of functions (Calls) for use in programs written in Interpreted BASIC, QuickBASIC, or QBASIC. The Calls allow you to write control programs without using register-level programming, and they perform the most commonly used set-up and operating functions. The Advanced Software Option provides Call Drivers for PASCAL and C and Windows 3.x languages.

Chapter 6 lists and briefly describes each of the Call drivers and each of the Calls. Chapter 7 covers the use of each Call and provides examples for Interpreted BASIC, QuickBASIC, and QBASIC.

3.4 LOW-LEVEL-REGISTER I/O PROGRAMMING

You may also program the ADC-16 by writing directly to the on-board registers. Chapter 8 supplies ADC-16 register maps and corresponding bit functions.

■ ■ ■

CHAPTER 4

THE POP UP CONTROL PANEL

4.1 OVERVIEW

The Pop Up Control Panel is a software tool for setting up and monitoring the operation of your ADC-16 board. In its monitoring capacity, the Pop-Up is also useful for verifying settings and for isolating incorrect settings or problems.

Supporting software for the Pop-Up consists of the two drivers VI.EXE and PADC16.EXE; both drivers are in the Distribution Software. For descriptions, refer to Section 4.2.

To use the Pop Up Control Panel, you must first load VI.EXE and PADC16.EXE into DOS memory. Loading may be via the DOS command line or via batch file. For complete loading instructions, refer to Sections 4.4 and 4.5.

4.2 DRIVER DESCRIPTIONS

VI.EXE	Virtual Instrument driver program that supports the graphics, functions, and background operations of the Pop Up Control Panel for the ADC-16 and other products from the manufacturer. This driver occupies approximately 55K of DOS memory, and it must be loaded **before** PADC16.EXE.
PADC16.EXE	The driver program that supplies the graphics, functions, and background operations for the ADC-16 Pop Up Control Panel. This program loads from the DOS command line, and it must be loaded **after** VI.EXE.

After loading these drivers, you may display or hide the Pop Up Control Panel panels using *hot keys*, either the default hot keys or those you assign during the PADC16.EXE loading. Pop Up Control Panel panels occupy the top eight character lines of the computer display; you operate their controls by either mouse or keyboard. For complete Pop Up Control Panel operating instructions, refer to Chapter 5.

4.3 DRIVER-FILE LOADING/UNLOADING OPTIONS

General

Your options for loading and unloading driver files are as follows:

- Via the AUTOEXEC.BAT file (for loading only).
- Via the command line.

- Via a batch file.
- Via the SHOW.BAT utility.

With each of these options, the load/unload order must be as follows:

Proper loading order:

First, load **VI.EXE**
Then, load **PADC16.EXE**

Proper unloading order:

First, unload **PADC16.EXE**
Then, unload **VI.EXE**

Loading Via AUTOEXEC.BAT

When you load via the AUTOEXEC.BAT file, your driver files load automatically with each computer boot-up. The files remain active until you unload them or shut down the computer.

Modify the AUTOEXEC.BAT file to include driver-loading instructions in either of two ways, as follows:

1. Use a text editor. Edit AUTOEXEC.BAT to include

   ```
   VI
   PADC16
   ```

   Note that using these instructions without switch options gives you default settings (see Chapter 2 for default settings). To change the default settings, use the switch options described in Section 4.5.

2. Use the ADCSETUP.EXE setup utility (from the Distribution Software) to automatically install the basic driver-loading instructions in AUTOEXEC.BAT. To use this utility, log to the directory containing the Distribution Software and type **ADCSETUP** < **Enter** > .

Loading/Unloading From The Command Line

This method allows you to load and unload the driver files on an as-needed basis. You may also change your switch parameters with each loading. Unfortunately, this method can be tedious, especially when your entry contains an error. Drivers loaded from the DOS command line may be unloaded from the command line, or they may be unloaded by rebooting the PC.

Loading/Unloading Via Batch Files

Batch files also allows you to load and unload your driver files on an as-needed basis. However, you are confined to a single set of switch parameters unless you modify your start-up file or you have additional batch files. You must also use separate batch files for loading and unloading. Use a text editor to create your batch files, and be sure you insert your instructions in the correct order (refer to *Loading/Unloading Order*).

Loading Via SHOW.BAT

The Distribution Software contains the file *SHOW.BAT* for the purpose of fast and easy batch-file loading. SHOW.BAT loads VI.EXE and PADC16.EXE in the required order, leaving you with the Pop Up Control Panel on the screen.

Note that loading PADC16 without switch options gives you the board's default settings. See Chapter 2 for a list of default settings.

4.4 SYNTAX CONVENTIONS

Unless otherwise noted, the syntax for the driver load/unload sections Sections 4.5 and 4.6) use the following character-formatting rules:

Bold = Indicates a mandatory entry, as follows:
- **BOLD UPPER-CASE** : enter exactly what is shown.
- **bold lower-case** : substitute a name, string, or value.

Italic = Indicates optional entry, as follows:
- *ITALIC UPPER-CASE* : enter exactly what is shown.
- *italic lower-case* : substitute a name, string, or value.

Plain = Indicates characters not required for execution of the command. Plain characters appear in the syntax only for clarity; whether used or not, they have no effect.

4.5 VI.EXE: LOADING & UNLOADING

Syntax for loading/unloading VI.EXE is as follows:

NOTE: Refer to Section 4.4 for syntax conventions.

path **VI**.exe *MONO* */HK = **x*** */MK = **m*** */SK = **s*** */U*

where

path — DOS path to VI.EXE. Example: **C:\ADC16**

MONO — Optional command for computers with monochromatic displays. If *MONO* is not specified, *COLOR* is assumed.

*/HK = **x*** — Help Key switch: to specify your choice of key(s) for bringing up the Help Screen for the Pop Up Control Panel. The default Help Keys are < **Alt** > + < **H** >.

x is one, two, or three keys, as follows:

(1) One key: any choice of < **A** > through < **Z** >, < **F1** > through < **F10** >, < **0** > through < **9** >, or < **Tab** >, < **Esc** >, or < **?** >;

(2) Two or three keys: it can be a combination of the < **Ctrl** > and/or < **Alt** > key(s) plus any choice of < **A** > through < **Z** >, < **F1** > through < **F10** >, < **0** > through < **9** >, or < **Tab** >, < **Esc** >, or < **?** >.

Note that your entry must be spelled out. For example, entering **/HK = F1** assigns the function key < **F1** > to be the Help Key. Or entering **/HK = Ctrl D** specifies that the < **Ctrl** > and < **D** > keys must be pressed at the same time to bring up the Help screen.

*/MK = **m*** — Mode Select Key switch: to specify whether the Pop-Up Menu is to be Keyboard or a Mouse operated. The default Mode Select keys are < **Alt** > + < **M** >. Using the Mode Select key combination switches the Pop-Up Menu into Keyboard mode; from there, you press < **Esc** > to enter Mouse mode.

m is one, two, or three keys, as follows:

(1) One key: any choice of < **A** > through < **Z** >, < **F1** > through < **F10** >, < **0** > through < **9** >, or < **Tab** >, < **Esc** >, or < **?** >;

(2) Two or three keys: it can be a combination of the < **Ctrl** > and/or < **Alt** > key(s) plus any choice of < **A** > through < **Z** >, < **F1** > through < **F10** >, < **0** > through < **9** >, or < **Tab** >, < **Esc** >, or < **?** >.

Note that your entry must be spelled out. For example, entering /MK = F8 assigns the function key < F8 > for Mode Select. Or entering /MK = Ctrl T specifies that the < Ctrl > and < T > keys must be pressed at the same time.

/SK = s Instrument Select Key switch: if you have other ADC-16 boards installed in your computer, this switch enables you to toggle between Pop-Up Panels of each instrument for the HELP function or for Mode selection of Keyboard input. The default Instrument Select key is < Alt > + < Tab > .

s is one, two, or three keys, as follows:

(1) One key: any choice of < A > through < Z > , < F1 > through < F10 > , < 0 > through < 9 > , or < Tab > , < Esc > , or < ? > ;

(2) Two or three keys: it can be a combination of the < Ctrl > and/or < Alt > key(s) plus any choice of < A > through < Z > , < F1 > through < F10 > , < 0 > through < 9 > , or < Tab > , < Esc > , or < ? > .

Note that your entry must be spelled out. For example, entering /SK = F3 assigns the function key < F3 > as the Mode Select key. Or entering /SK = Ctrl S specifies that the < Ctrl > and < S > keys must be pressed at the same time.

/U Unload switch: when VI.EXE is already loaded and you wish to unload it from memory, you use only this switch in your instruction. For example, using the example path given above your unload instruction will be

```
C:\ADC16\VI /U
```

NOTE that before you can unload VI.EXE, you must first unload PADC16.EXE.

An example instruction for VI.EXE is

```
C:\VI.EXE /HK = Alt X  /MK = Alt Y  /SK = Alt Z
```

You would enter this instruction at the DOS command line or into the AUTOEXEC.BAT file. This instruction specifies the following:

- VI.EXE is in the root directory of the C Drive
- Color monitor
- Help keys are < Alt > + < X >
- Mode Select keys are < Alt > + < Y >
- Instrument Select keys are < Alt > + < Z > .

When this instruction for loading VI.EXE is executed, the computer will respond with the following display:

```
******      VI.EXE  Rev 3.04
                    o HELP KEY is ALT X
                    o INSTRUMENT SELECT KEY is ALT Z
                    o MODE SELECT KEY is ALT Y
```

4.6 PADC16.EXE: LOADING & UNLOADING

Before you load PADC16.EXE, you must load VI.EXE. Syntax for loading/unloading PADC16.EXE is as follows:

NOTE: Refer to Section 4.4 for syntax conventions.

path **PADC16.exe** */BA = **b*** */F = **cfgfile*** */PK = **p*** */NAME = devname* */Help* */U*

The elements of this model are explained as follows (note that if you do not use the switch options, their default values remain in effect):

path DOS path to PADC16.EXE. Example: **C:\ADC16**

*/BA = **b*** Base Address: The Base Address setting for the board being loaded (refer to base-address selection, in Chapter 2). The default Base Address is &H300 (768 decimal).

b is the Base Address value. Either hex (range = &H200-&H3F0) or decimal values are acceptable; however, if they are given in hex they must be preceded by an ampersand (&) and an H (for example, **&H300**). Make certain the Base Address you enter has not been already assigned to another device.

*/F = **cfgfile*** Default Override: allows you to change any or all of the default parameters discussed in the configuration section (Section 2.2) of Chapter 2.

cfgfile the name of a configuration file created with the ADC16CFG.EXE utility discussed in Chapter 2. The example file used for discussion in Chapter 2 is *ADC16.CFG* .

*/PK = **p*** Pop Up Control Panel hot key switch: when the number of installed boards is greater than one, each instrument should have its own Pop-Up key assignment. The default Pop Up Control Panel hot key is **< Alt > < F5 >** .

p is one, two, or three keys, as follows:

(1) One key: any choice of **< A >** through **< Z >** , **< F1 >** through **< F10 >** , **< 0 >** through **< 9 >** , or **< Tab >** , **< Esc >** , or **< ? >** . Note that in working with a single key, the key you choose will be unusable for any other function while CALL.EXE is loaded; you might therefore want to confine your single-key choice to one of the function keys (**< F1 >** through **< F10 >**).

(2) Two or three keys: it can be a combination of the < Ctrl > and/or < Alt > key(s) plus any choice of < A > through < Z >, < F1 > through < F10 >, < 0 > through < 9 >, or < Tab >, < Esc >, or < ? >.

Note that your entry must be spelled out. For example, entering /PK = F3 assigns the function key < F3 > for Mode Select. Or entering /PK = Ctrl P specifies that the < Ctrl > and < P > keys must be pressed at the same time.

/NAME = n Use when the number of installed boards is greater than one and you wish to display the Pop Up Control Panel for any two of the boards simultaneously. **You must enter an PADC16.EXE instruction for both boards--to give each board unique hot keys, Base Address, and name.**

n is the name you enter for one board; it must be 1-8 characters. Any of the following characters are acceptable: < A > through < Z >, < 0 > through < 9 >, < $ >, < & >, < # >, < @ >, < ! >, < % >, < (>, <) >, < - >, < { >, < } >, < _ >. Note that /, \, and * are illegal.

For example, **NAME=ADC16**

/Help Use this switch to list all command-line options for the ADC16.EXE driver. When you use this switch, you will need no other switches, because the driver is not installed.

/U Unload switch: when you wish to unload PADC16.EXE from memory, use only this switch in your instruction. For example, using the path given above your unload instruction will be

```
C:\ADC16\PADC16 /U
```

NOTE that you must unload PADC16.EXE before VI.EXE.

An example instruction for PADC16.EXE is as follows:

```
C:\ADC16\PADC16
```

You would enter this instruction at the DOS command line or into the AUTOEXEC.BAT file. This instruction specifies the following:

- PADC16.EXE is in the ADC16 directory of the C Drive
- With no switches, the software uses the factory-preset parameters.

When this instruction for loading PADC16.EXE is executed, the computer responds with the following display.

```
******   PADC16.EXE popUP Driver   Rev 1.00
                  o popUP Hot Key is ALT F6
                  o Factory Configuration assumed

Number Of Bytes Used By Device : 57968
At Memory Segment  [11B3]
```

4.7 LOADING ANSI.SYS

To obtain optimum computer performance while working with the VI.EXE and PADC16.EXE files, your system may require access to the DOS ANSI.SYS file. ANSI.SYS is generally included with the files in your computer's DOS software. To activate ANSI.SYS, you must modify your computer's CONFIG.SYS file to include an ANSI.SYS loading instruction. Your instruction must use the syntax of the following model:

DEVICE = *path* ANSI.SYS

For example, if ANSI.SYS is in a directory called *DOS* on the D Drive, your instruction would be

```
DEVICE=D:\DOS\ANSI.SYS
```

Remember that when you alter a CONFIG.SYS file, you must re-boot before the new changes can take effect.

■ ■ ■

CHAPTER 5

POP UP CONTROL PANEL OPERATION

5.1 PRELIMINARY REQUIREMENTS

Before you can use the Pop Up Control Panel, your ADC-16 board must be properly installed and your VI.EXE and PADC16.EXE drivers properly loaded. For board-installation instructions, refer to Chapter 2. For driver-loading instructions, refer to Chapter 4.

5.2 GETTING STARTED

To Get Up & Running Quickly

NOTE: This procedure is valid only when ADC-16 factory configuration is in effect for both hardware and software and when you are logged to the directory containing the Distribution Software.

1. With the board installed, type **SHOW** to put the Analog Panel at the top of your display.
2. Click your mouse on **ADSTART**, located at the top of the Panel. You are now acquiring samples from channels at 16 Hz. The data from this first channel scan should be showing on the Panel.
3. Connect known voltages to Analog Input channels and repeat Step 2.

To Get Up & Running Via Hot Keys

1. With the board installed and drivers loaded, use the hot keys to access the Pop Up Control Panel. The default hot keys are **< Alt > < F6 >**; press these keys simultaneously to put the Analog Panel at the top of your display.
2. Click your mouse on **ADSTART**, located at the top right of the Panel. You are now acquiring samples from channels at 16 Hz. The data from this first channel scan should be showing on the Panel.
3. Connect known voltages to Analog Input channels and repeat Step 2.

5.3 IMPORTANT HOT KEY COMBINATIONS

Familiarity with the hot-key combinations can speed up operation. The following hot keys are the default combinations; you may change any or all of them to keys of your choice. To change any of these keys, refer to the switch options in the driver-loading instructions of Sections 4.4 and 4.5.

< Alt > < F6 >**Pop Up/Down Keys.** Use these keys to show and hide the Pop Up Control Panel panels.

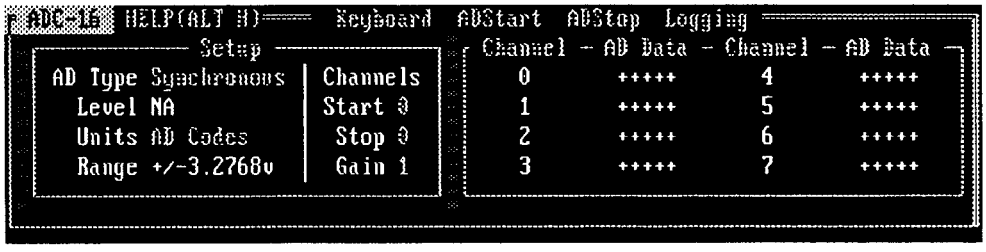
< Alt > < H > **Help Keys.** Use these keys to show the Help Panel whenever a Pop Up Control Panel Panel is showing.

< Alt > < Tab >**Next Pop Up Keys.** Use these keys to move from Pop Up to Pop Up when panels for multiple boards are showing.

< Alt > < M > **Keyboard Mode Entry Keys.** Use these keys to change to the Keyboard Entry Mode. Use the < **Esc** > key to exit this mode.

5.4 THE CONTROL PANEL

The Control Panel of the Pop Up Control Panel appears as follows:



Control Panel Menu

This menu appears along the top of the Panel and contains three options, as follows:

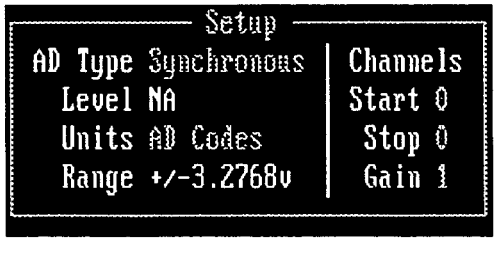
Keyboard This option toggles between the keyboard and mouse control of the Pop Up Control Panel.

ADStart This option starts an A/D conversion takes you to the Digital Panel. However, before starting a conversion, make any necessary changes to the Setup controls.

ADStop This option stops an A/D conversion.

Setup Controls

These controls set up parameters for the A/D conversion. The Setup Menu appears as follows:



AD Type sets the A/D conversion mode. Choices are **Synchronous** and **Interrupt**.

Level is active only when the AD Type is Interrupt. Choices are **2, 3, 4, 5, 7, 10, 11,** and **15**.

Units selects **AD Codes** or **Volts**.

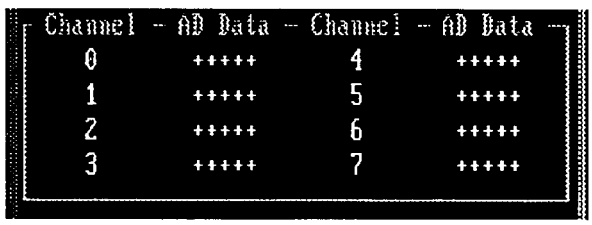
Start selects the Start Channel for the A/D conversion. The Start Channel may range from 0-7; however, it can be no greater than the number specified for the Stop Channel.

Stop selects the Stop Channel for the A/D conversion.

Gain Selects the gain for the A/D conversion. Choices are **1, 10,** and **100**.

Channel - A/D Data

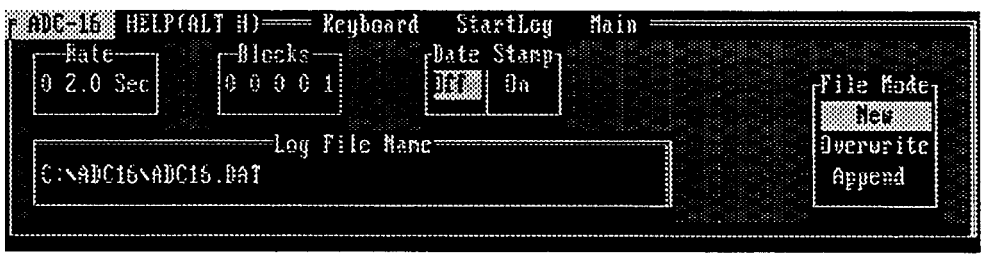
The monitor panel for the A/D conversion(s) appears as follows:



Only the channels specified by Start and Stop (under Setup) will show results. These results update with each conversion cycle.

5.5 THE DATA LOGGING PANEL

The Data Logging Panel appears as follows:



Data Logging Menu

This menu appears along the top of the Logging Panel and contains 3 options as follows:

Keyboard This option toggles between the keyboard and mouse control of the Control Panel.

StartLog This option starts the Logging function and StopLog is displayed. If StopLog is selected then StartLog is displayed and logging is stopped.

Main Displays the Main Control Panel Screen.

■ ■ ■

CHAPTER 6

THE CALL DRIVER

6.1 GENERAL

The Call Driver is a comprehensive set of functions you may incorporate into your own application programs. These functions provide a high-level interface to the ADC-16 and perform all required register-level reads and writes.

The Call Driver is compatible with most BASIC programming languages. Specifically, the Driver supports the following:

- Interpreted BASIC (GWBASIC, BASICA, BASIC) - Section 6.2.
- QuickBASIC Versions 4.0 and 4.5 - Section 6.3.
- Quick/Professional BASIC Version 7.0 and above - Section 6.3.
- QBASIC - Section 6.4

Example programs for each of the supported languages are included in the Distribution Software. You may find it helpful to refer to one or more of these example programs while reading this chapter.

Since the syntax and application of the BASIC languages are different, this chapter describes programming in each language separately.

Section 6.5 includes a listing of Calls available in the ADC-16 driver. This section is a quick-reference only. Detailed descriptions of the Calls are in Chapter 7.

6.2 GWBASIC, BASICA, & BASIC

Program Flow

A typical programming sequence is as follows:

1. Initialize the entry pointers to the driver. *
2. Load the driver. *
3. Declare and dimension all variables to be used in the program. *

   NOTE: Failure to declare all variables before opening the driver will cause program pointers to be shifted and the program to fail.
4. Load the ADC-16 hardware configuration, and open the driver. *
5. Initialize the driver and board, and establish communications between the driver and the program. *

6. EITHER

   a. Execute a simple ADC-16 operation.

   OR

   b. Select the desired group of ADC-16 operations, referred to as the group's *Frame* . Execute these operations, taking some data.

      * Indicates that this function is performed in the "Quick-Start" programs. (see next section).

Choose Step 6a for simple applications such as performing a single A/D conversion or reading a digital input port. Choose 6b for more complex operations such as an interrupt-based scan of the multiple channels.

The following sections describe the various parts of the program flow diagram. However, many programmers may learn faster simply by referring to the example programs in the Distribution Software.

Quick-Start Program

To aide in the creation of programs, the Distribution Software includes a *Quick-Start* program, which includes all driver loading, pointer definition, and ADC-16 initialization. For interpreted BASIC, the Quick-Start program is **ADC16.BAS** . The program is in standard BASIC source code with an *Insert Your Code Here* section.

When you begin a program, it will be easiest to begin with the Quick-Start program and add new sections to the existing code. The alternative is to struggle through rewriting the driver, pointer, and initialization code.

Initializing Pointers And Loading The Driver

The last part of Section 6.2 contains a listing of an Interpreted BASIC example program. You may wish to refer to this example program during the discussions in this section.

In Interpreted BASIC, the driver and a number of pointers to the driver must be loaded within the program itself. The entry-point variables must be loaded at this time (each command has a unique entry point). This is performed by a subroutine in ADC16.BAS. Since there are several entry points, it is recommended that the ADC16.BAS Quick-Start program (or one of the example programs) be used as the starting point for new programs. This will ensure the correct loading of pointers. The following statement calls the subroutine that loads the driver entry points.

```
260  GOSUB 710
       .
       .

-----------

       .
710  ADC16DEVOPEN%=0
720  ADC16GETDEVHANDLE%=3
       .
       .
       .
xxx  RETURN
```

The following statements then load the driver.

```
270  DEF SEG = &H9000
280  BLOAD "ADC16.BIN",0
```

This code loads the driver at 9000h (at the 576K segment) and is usually a good choice. However, computers with 512 KB or less memory will require loading the driver at a lower address (for example, 4000h or 5000h).

Initializing The Variables

You must declare all your program variables before opening the driver. Using an undeclared variable during the program will move the entry points and cause the program to fail. Examples of declarations are as follows:

```
340  ADC16 = 0                                  'declare ADC16 as real
370  DIM ADValue%(2)                            'declare an integer array
390  NumOfBrds%=0: ErrFlag%=0: BrdNum%=0        'declare integers
400  GAIN%=0: CHAN%=0
410  N$=""                                      'declare N$ string
402  VEL=0:DIRECT=0:TEMP=0                      'declare application
                                                'dependent variables
```

Opening The Driver

The next step opens the driver, and reads the current configuration file (see Section 2.6) for the ADC-16 hardware (Base Address, Interrupt Level, input configuration). The ADC16DevOpen command performs this function, which is demonstrated (along with other required initialization functions) in the example program in Section 6.3. The syntax for the ADC16DevOpen function is shown below:

```
500  N$ = "ADC16.CFG" + CHR$(0)      'Setup filename
510  CALL ADC16DevOpen%(N$, NumOfBrds%, ErrFlag%)   'Read ADC-16
520                                                 'configuration file
```

Establish Communications Between The Driver & Program

To allow the use of multiple boards, the Calls require a board identifier for each board. The board identifier is called a *Device Handle*, or simply *Handle*. Without Handles, a **KADRead** command would not know which board (in a multi-board system) to call.

The Handle for each board is returned by the **ADC16GetDevHandle** command after a successful call to **ADC16DevOpen**, which is included in the Quick-Start programs. Additional information on the **ADC16GetDevHandle** function (and concept) is as follows:

SINGLE BOARD SYSTEMS

In single-board systems, obtain the Handle via the ADC16GetDevHandle function. The following examples show use of the Handle.

First, get the Handle, as follows:

```
539  BrdNum% = 0                '0 since only 1 board in system
540  CALL ADC16GetDevHandle%(BrdNum%,DevHandle,ErrFlag%)
```

Subsequent calls use the variable *DevHandle* to select the installed ADC-16.

You may change the variable name *DevHandle* to make the program easier to read. For example, the following program lines reflect a variable change from *DevHandle* to *ADC16* .

```
539  BrdNum% = 0                '0 since only 1 board in system
540  CALL ADC16GetDevHandle%(BrdNum%,ADC16,ErrFlag%)
```

MULTIPLE BOARD SYSTEMS

In systems with more than one board, you must use more than one Handle. The following example shows how to obtain and use Handles in a system with two ADC-16s.

```
530  BrdNum% = 0                'First ADC-16 board
540  CALL ADC16GetDevHandle%(BrdNum%,ADC16A,ErrFlag%)

550  BrdNum% = 1                'Second ADC-16 board
560  CALL ADC16GetDevHandle%(BrdNum%,ADC16B,ErrFlag%)
```

NOTE: In this example, declaration of variables ADC16A and ADC16B should be in the variable-declaration section.

Subsequent function calls will use either the ADC16A or ADC16B Handle to determine the selected board.

Immediate Execution Commands & Frames

Two types of Call commands may be used to control the ADC-16 once the driver is installed and the Board initialized. These are *Immediate Operation* commands and *Setup* or *Frame* commands.

IMMEDIATE OPERATION COMMANDS

A variety of single-function commands may be executed without any prior setup. The simplest of these is **KADRead** , which specifies an input channel, an input gain, and then performs a single A/D conversion. The syntax for this command is as follows:

```
620  Gain%=0:Chan%=0
630  CALL KADRead%(ADC16A, Chan%, Gain%, ADValue%(0), ErrFlag%)
640  PRINT "Board 1 returned",ADValue%(0)
```

Note that the *ADValue%(0)* must be defined as an integer array. The ADC16A Handle is a single-precision, real variable and must be declared in the variable declaration section. The actual numeric value of the Handle is simply a pointer within the driver and should be passed where required.

MULTIPLE BOARD SYSTEMS

Systems with more than one board must use more than one Handle. The following example opens the Handles for two ADC-16s and then takes an A/D reading from Channel 0 of the first Board and from Channel 5 of the second.

```
530  BrdNum% = 0                  'First ADC-16 board
540  CALL ADC16GetDevHandle%(BrdNum%,ADC16A,ErrFlag%)

550  BrdNum% = 1                  'Second ADC-16 board
560  CALL ADC16GetDevHandle%(BrdNum%,ADC16B,ErrFlag%)

570  Chan% = 0: Gain% = 0
580  CALL KADRead(ADC16A, Chan%, Gain%, ADValue%(0), ErrFlag%)
590  PRINT "The first ADC-16 returned",ADValue%(0)

600  Chan% = 5: Gain% = 0
610  CALL KADRead(ADC16B, Chan%, Gain%, ADValue%(0), ErrFlag%)
620  PRINT "The second ADC-16 returned",ADValue%(0)
```

Note that line 580 uses the first ADC-16 Handle as set-up in line 540, while line 610 uses the second Handle.

Other Immediate operation commands include

KDIRead	Read the digital inputs.
KDOWrite	Write to the digital outputs.

FRAMES

Analog Input and Digital Output allow for operation using Frames. For Digital Output the Frame is a means of returning the value of the Digital Output port. For Analog Input the Frame is the group of all setup parameters used in a synchronous or interrupt driven operation. For example, an analog input *Frame* requires setting up the following parameters:

- Input Channels to scan (Start and Stop channels).
- Input Gain.
- Number of samples.
- Data buffer location.

Program flow is similar in each type of Frame. The standard program flow using Frames is as follows:

First:	Open or get a Frame with a **KGetADFrame** or **KGetDOFrame** command.
Next:	Setup any parameters required within the Frame.
Then:	Execute the desired Frame function.
Optionally:	Check the status of the operation initiated.

The following program segment is similar to the GWBEX7.BAS example program included in the Distribution Software. It may be helpful to refer to this example program during the following discussion.

Example using Analog Input Frame

```
10 ' Synchronous A/D scan using Start / Stop / Gain parameters
20 ' STEP 1 : The following is the start up code required to link to the
30 ' DAS Driver for all Interpreted BASIC programs. A subroutine (The GOSUB
40 '  800 below)  initializes a set of variables critical to the Driver.
80 '
90    GOSUB 800              ' Initialize entry variables
100   DEF SEG = &H9000       ' Suggested address where to load the Driver
110   BLOAD "ADC16.BIN",0    ' Load the driver
120 '
130 '-------------------------------------------------------------------
140 ' STEP 2 : Dimension and initialize ALL basic variables first.
150 ' Declare ALL additional program variables in this step.  You must
160 ' avoid declaring and using variables on the fly.
170 '
180   DEVHANDLE=0              ' Declare all handles as Reals
190   ADSET=0                  ' Analog Input Frame
200   A$=""
210   NUMOFBRDS%=0: ERRFLAG%=0: BRDNUM%=0
220   DIM BUFFER%(20), SAMPLES%(2)
230   STARTCHAN%=0
240   STOPCHAN%=0
250   GAINCHAN%=0
260   X%=0
270   BUFFEROFFSET% = 0
280 '
290 '-------------------------------------------------------------------
300 ' STEP 3 : Initialize the internal data tables with the information
310 ' contained in the configuration file ADC16.CFG.
320 '
330   A$="ADC16.CFG" + CHR$(0)
340   CALL ADC16DEVOPEN%(A$, NUMOFBRDS%, ERRFLAG%)
350   IF ERRFLAG% <> 0 THEN BEEP: PRINT "ERROR "; HEX$(ERRFLAG%): STOP
360 '
360 '-------------------------------------------------------------------
370 ' STEP 4 : Establish communication with the driver through the
380 ' Device Handle
390 '
400   BRDNUM%=0
410   CALL ADC16GETDEVHANDLE%(BRDNUM%, DEVHANDLE, ERRFLAG%)
420   IF ERRFLAG% <> 0 THEN PRINT "ERROR "; HEX$(ERRFLAG%): STOP
430 '
440 '-------------------------------------------------------------------
440 ' STEP 5 : To perform a FRAME based Analog Input function, first
450 ' get a FRAME handle and then initialize the parameters for the
460 ' FRAME.
470 '
480   CALL KGETADFRAME%(DEVHANDLE, ADSET, ERRFLAG%)
490   IF ERRFLAG% <> 0 THEN PRINT "ERROR "; HEX$(ERRFLAG%): STOP
500 '
510   STARTCHAN%=0:STOPCHAN%=7:GAIN%=0
520   CALL KSETSTARTSTOPG%(ADSET, STARTCHAN%, STOPCHAN%, GAIN%, ERRFLAG%)
530   IF ERRFLAG% <> 0 THEN PRINT "ERROR "; HEX$(ERRFLAG%): STOP
540 '
550   SAMPLES%(0) = 0        ' SAMPLES parameter ignored
560   BUFFEROFFSET% = VARPTR(BUFFER%(0))
570   CALL KSETBUF%(ADSET, DATASEG%, BUFFEROFFSET%, SAMPLES%(0), ERRFLAG%)
580   IF ERRFLAG% <> 0 THEN PRINT "ERROR "; HEX$(ERRFLAG%): STOP
590 '
600   CALL KSYNCSTART%(ADSET, ERRFLAG%)
610   IF ERRFLAG% <> 0 THEN PRINT "ERROR "; HEX$(ERRFLAG%): STOP
620 ' Display Data
630   FOR X% = 0 TO (STOPCHAN%-STARTCHAN%)
640     IF BUFFER%(X%) AND &H8000 THEN PRINT BUFFER%(X%) AND &H7FFF
650     IF (BUFFER%(X%) AND &H8000) = 0 THEN PRINT BUFFER%(X%) * -1
660   NEXT
```

```
670
800 ' Include the following in all your BASIC programs.
810 ' Initialize Driver entry point variables.
820 '
830   ADC16DEVOPEN%        =  0
840   ADC16GETDEVHANDLE%   =  3
850   KGETADFRAME%         =  6
860   KGETDOFRAME%         =  15
870   KGETVER%             =  30
880   KDASDEVINIT%         =  33
890   KSETIRQMAP%          =  36
900   KFREEFRAME%          =  39
910   KSETBUF%             =  42
920   KSETSTARTSTOPG%      =  54
930   KGETSTARTSTOPG%      =  57
940   KSETCHN%             =  60
950   KGETCHN%             =  63
960   KSETSTARTSTOPCHN%    =  66
970   KGETSTARTSTOPCHN%    =  69
980   KSETG%               =  72
990   KGETG%               =  75
1000  KSETCHNGARY%         =  78
1010  KCHKFRAME%           =  129
1020  KINITFRAME%          =  132
1030  KSYNCSTART%          =  135
1040  KINTSTART%           =  147
1050  KINTSTATUS%          =  150
1060  KINTSTOP%            =  153
1070  KGETDOCURVAL%        =  156
1080  KMOVEDATABUF%        =  201
1090  KADREAD%             =  213
1100  KDIREAD%             =  219
1110  KDOWRITE%            =  222
1120  KFORMATCHNGARY%      =  225
1130  KRESTORECHNGARY%     =  228
1140  KCLEARFRAME%         =  231
1150  DATASEG%             =  &HFFFF
1160 RETURN
```

The **KGETADFRAME** call is used to open the frame with the variable name ADSET for the ADC-16 board referred to by variable DEVHANDLE. The FRAME is filled with the start-stop channels and gain using the call **KSETSTARTSTOPG**. The data location for the acquisition is established by **KSETBUF**. The number of samples taken in this example is determined by the start and stop channels and the SAMPLES parameter in the **KSETBUF** call is ignored. With all the FRAME parameters completed in ADSET, the acquisition is initiated by a call to **KSYNCSTART**.

A program can define multiple frames. For example, an application may define a FRAME with different start-stop channels or use **KSETCHNGARY** to define a FRAME with the gain and channel sequence specified in a channel gain array.

6.3 QUICKBASIC 4.0 +, INCLUDING PROFESSIONAL BASIC 7.0 +

Program Flow

A typical programming sequence is as follows:

1. Load the driver (from the QB command line).
2. $INCLUDE: the driver function declarations. *
3. Declare and dimension all variables that will be used in the program. *

NOTE: Failure to declare all variables before opening the driver will cause the program pointers to be shifted and will cause the program to fail!!!

4. Load the ADC-16's hardware configuration, and open the driver. *
5. Initialize the driver and board, and establish communications between the driver and program.*
6. EITHER
   a. Execute a simple ADC-16 operation.

   OR

   b. Select the desired group of ADC-16 operations, referred to as the group's *Frame*. Execute these operations, taking some data.

   * Indicates that this function is performed in the "Quick-Start" programs. (see next section)

Choose Step 6a for simple applications such as performing a single A/D conversion or reading a digital input port. Choose Step 6b for more complex operations (an interrupt-based scan of the multiple channels).

The following sections describe the various parts of the flow diagram. However, many programmers will learn faster by simply referring to the various example programs supplied with the ADC-16.

Quick-Start Program

To aide in the creation of programs, the Distribution Software includes a *Quick-Start* program. This program includes all driver loading, pointer definition, and ADC-16 initialization; it is provided as standard QuickBASIC source code with an *Insert Your Code Here* section. The Quick-Start program for QuickBASIC versions is **QUICKBAS.BAS** QuickBASIC Version 4.0 or greater and/or Professional BASIC Version 7.0 or greater.

In your initial programming efforts, start with this program and add new sections to the existing code. This should prove simpler than going through the effort to rewrite the driver, pointer and initialization code.

Loading & Including The Driver

In QuickBASIC, the driver needs to be loaded at the QB command line. This procedure is as follows:

```
QB /L ADC16QB.QLB
```

In Quick/Professional BASIC Version 7.0 and greater, use

```
QBX /L ADC16QBX.QLB
```

The program itself will also need to contain the following INCLUDE statement.

QuickBASIC Versions 4.0 and 4.5

```
' $INCLUDE: 'Q4IFACE.BI'
```

Quick/Professional BASIC Version 7.0 and greater

```
' $INCLUDE: 'Q7IFACE.BI'
```

Initializing The Variables

Declare all variables before opening the driver. Examples of declarations include:

```
DIM GAIN, CHAN AS INTEGER      - declare a variable type
DIM NumOfBrds AS INTEGER
DIM ADC16 AS LONG              - declare long (32-bit) integers
DIM ADValue AS LONG
DIM A$                         - declare A as a string
VEL=0:DIRECT=0:TEMP=0          - declare application variables
```

Don't Forget!! Declare all variables before opening the driver. Using an undeclared variable in the program body (even the **I** in a simple **FOR I= 1 to 10:NEXT I** loop) causes shifting of device pointers and results in program failures!!

Opening The Driver

The next step opens the driver, and reads the current configuration file for the ADC-16 hardware (Base Address, Interrupt Level, input configuration). (This file is created using the ADC16CFG.EXE file (described in Chapter 2).) The **ADC16DevOpen** command performs this function. This function, as well as the other required initialization functions are shown in the example program at the end of this section. The syntax for this call is as follows:

```
A$ = "ADC16.CFG" + CHR$(0)          'read ADC-16 config file
Derr = ADC16DEVOPEN%(SSEGADD(A$), NumOfBrds)   'execute the call
```

The SSEGADD command tells the driver where to look for the ADC-16 configuration information. SSEGADD is a standard command in Prof./QuickBASIC Version 7.0 and above; but it is not defined in standard QB Version 4.5 or lower. However, our Version 4.0 and 4.5 drivers define the SSEGADD function.

Establish Communications Between The Driver & Program

To allow the use of multiple boards, the Calls require a board identifier for each board. The board identifier is called a *Device Handle*, or simply *Handle*. Without Handles, a **KADRead** command would not know which board (in a multi-board system) to call. The board Handle is the first parameter in each of the ADC-16 function calls.

The Handle for each board is returned by the **ADC16GetDevHandle** command after a successful call to **ADC16DevOpen**, which is included in the Quick-Start programs. Additional information on the **ADC16GetDevHandle** function (and concept) follows.

SINGLE BOARD SYSTEMS

In single-board systems, the Handle is straightforward. The Handle is obtained via the ADC-16 GetDevHandle function. The following examples describe how the Handle is used.

First, get the handle, as follows:

```
BrdNum = 0              '0 since only 1 board in system
Derr = ADC16GetDevHandle%(BrdNum,DevHandle)
```

(where Derr contains the error status)

Subsequent calls use the variable *DevHandle* to select the installed ADC-16.

The variable name *DevHandle* may be changed to make the program easier to read. For example, a variable change from *DevHandle* to *ADC16* is reflected in the following program lines.

```
BrdNum = 0              '0 since only 1 board in system
Derr = ADC16GetDevHandle%(BrdNum,ADC16)
```

MULTIPLE BOARD SYSTEMS

In systems with more than one board, you must use more than one Handle. The following example shows how the Handles are obtained and used in a system with two ADC-16s.

```
BrdNum = 0              'First ADC-16 board
Derr = ADC16GetDevHandle%(BrdNum,ADC16A)

BrdNum = 1              'Second ADC-16 board
Derr = ADC16GetDevHandle%(BrdNum,ADC16B)
```

Subsequent function calls will use the ADC16A or ADC16B Handle for either Board to determine the selected board.

Immediate Execution Commands & Frames

Two types of Call commands will control the ADC-16 once the board and the driver are installed and initialized. These are Immediate Operation commands and Setup or Frame commands.

IMMEDIATE OPERATION COMMANDS

Several single-function commands may be executed without any prior set-up. The simplest of these is the **KADRead** command that selects an input channel, an input gain, and then performs a single A/D conversion. The syntax for this command is as follows:

```
Chan = 0: Gain = 0
Derr = KADRead%(DevHandle, Chan, Gain, ADValue)
PRINT "The A/D returned",ADValue
```

Note that the *ADValue* and the *DevHandle* are *Long Integers* and must be declared in the variable declaration section. The ADC-16 data is returned in the least significant fifteen bits of ADValue, if bit sixteen is zero multiply the data by minus one.

MULTIPLE BOARD SYSTEMS

In systems with more than one board, more than one Handle must be used. The following example opens the handles for two ADC-16s and then takes an A/D reading from Channel 0 of the Board 1 and from Channel 5 of Board 2.

```
BrdNum = 0              'First board
Derr = ADC16GetDevHandle%(BrdNum,ADC16A)

BrdNum = 1              'Second board
Derr = ADC16GetDevHandle%(BrdNum,ADC16B)

Chan = 0: Gain = 0
Derr = KADRead%(ADC16A, Chan, Gain, ADValue)
PRINT "The first Board returned",ADValue
Chan = 5: Gain = 0
Derr = KADRead%(ADC16B, Chan, Gain, ADValue)
PRINT "The second Board returned",ADValue
```

Note that the first **KADread** uses the Handle of the first Board as Set up with BrdNum = 0, while the second **KADRead** uses the second Handle.

Other Immediate Operation commands include

KDIRead	Read the digital inputs
KDOWrite	Write the digital outputs

FRAMES

Analog Input and Digital Output allow for operation using Frames. For Digital Output the Frame is a means of returning the value of the Digital Output port. For Analog Input the Frame is the group of all setup parameters used in a synchronous or interrupt driven operation. For example, an A/D function requires setting up the following parameters:

- Input Channels to scan (Start and Stop channels).
- Input Gain.
- Number of samples.
- Data buffer location.

A complete set of selections is a *Frame*. A complete definition of the above parameters would be included in an A/D frame. Two types of Frames are available for the ADC-16, as follows: A/D Frames and Digital Output Frames.

Program flow is similar in each type of Frame, as follows:

First:	Open or get a Frame with **KGetADFrame** or **KGetDOFrame**.
Next:	Setup the parameters within the Frame.
Then:	Execute the desired Frame function.
Optionally:	Check the status of the operation initiated.

The following program example is similar to the QBEXAMP7.BAS example in the Distribution Software. It may be helpful to refer to the example during the following discussion.

Example Program Analog Input Frame

```
' STEP 1 : Include the supplied Q4IFACE.BI (or Q7IFACE.BI).  The
'          INCLUDE file  contains all function declarations supported by
'          the driver.
'
'  $INCLUDE: 'Q4IFACE.BI'

'--------------------------------------------------------------------------
' STEP 2 : Define ALL local variables required by the program here.  NOTE
' that you must avoid declaring and using QuickBASIC variables on the
' fly.

DIM BUFFA(20) AS INTEGER
DIM NumOfBoards AS INTEGER
DIM DERR AS INTEGER
DIM DEVHANDLE AS LONG
DIM ADSET AS LONG
DIM STATUS AS INTEGER
DIM COUNT AS LONG
DIM LONGADD AS LONG
DIM StopChan AS INTEGER
DIM StartChan AS INTEGER
DIM GGAIN AS INTEGER
DIM BoardNum AS INTEGER
DIM Samples AS INTEGER
I% = 0

'--------------------------------------------------------------------------
' STEP 3: Initialize the internal data tables with the information
'         contained in the configuration file ADC16.CFG.

    A$ = "ADC16.CFG" + CHR$(0)
    DERR = ADC16DEVOPEN%(SSEGADD(A$), NumOfBoards)
    IF DERR <> 0  PRINT "ERROR "; HEX$(DERR) ; 'IN..DEVOPEN'": STOP

'--------------------------------------------------------------------------
' STEP 4 : Establish communication with the driver through the
'          Device Handle.

    BoardNum = 0
    DERR = ADC16GETDEVHANDLE%(BoardNum, DEVHANDLE)
    IF DERR <> 0 PRINT "ERROR, DEVICE HANDLE IS null. . .": STOP

 '-------------------------------------------------------------------------
' STEP 5 : To perform a FRAME based Analog Input function, first
'          get a FRAME handle and then initialize the parameters for
'          the FRAME.

   DERR = KGetADFrame%(DEVHANDLE, ADSET)
   IF DERR <> 0 THEN PRINT "ERROR "; DERR: STOP

   SAMPLES=0 ' ADC16 does not use the SAMPLES parameter
   DERR = KSetBuf%(ADSET, BUFFA(0), Samples)
   IF DERR <> 0 THEN PRINT "ERROR "; HEX$(DERR): STOP

   DERR = KSetChnGAry%(ADSET, CHANGAINARRAY(0))
   IF DERR <> 0 THEN PRINT "ERROR "; HEX$(DERR): STOP

   StartChan=0:StopChan=7:GGAIN=0
   DERR = KSetStartStopG%(ADSET, StartChan, StopChan, GGAIN)
   IF DERR <> 0 THEN "ERROR "; HEX$(DERR): STOP
   DERR = KSyncStart%(ADSET)
   IF DERR <> 0 THEN PRINT "ERROR "; HEX$(DERR): STOP
```

```
DERR = KSyncStart%(ADSET)
IF DERR <> 0 THEN PRINT "ERROR "; HEX$(DERR): STOP

DERR = KFreeFrame%(ADSET)
IF DERR <> 0 THEN BEEP: PRINT "ERROR "; HEX$(DERR):  STOP

FOR I% = 0 TO (STOPCHAN-STARTCHAN)
     PRINT HEX$(BUFFA(I%))
NEXT I

END
```

The **KGETADFRAME** call is used to open the frame with the variable name ADSET for the ADC-16 board referred to by variable DEVHANDLE. The FRAME is filled with the start-stop channels and gain using the call **KSETSTARTSTOPG**. The data location for the acquisition is established by **KSETBUF**. The number of samples taken in this example is determined by the start and stop channels and the SAMPLES parameter in the **KSETBUF** call is ignored. With all the FRAME parameters completed in ADSET, the acquisition is initiated by a call to **KSYNCSTART**.

A program can define multiple frames. For example, an application may define a FRAME with different start-stop channels or use **KSetChnGAry** to define a FRAME with the gain and channel sequence specified in a channel-gain array.

6.4 QBASIC

Program Flow

A typical programming sequence is as follows:

1. Initialize the pointers to the driver. *
2. Load the driver. *
3. Declare and dimension all variables that will be used in the program. **Failure to declare all variables before opening the driver will cause the program pointers to be shifted and will cause the program to fail!!!** *
4. Load the ADC-16's hardware configuration and open the driver. *
5. Initialize the driver and board and establish communications between the driver and the programming language. *
6. EITHER
   a. Execute a simple ADC-16 operation.

   OR

   b. Select the desired group of ADC-16 operations referred to as the group's *Frame*. Execute the operations, taking some data.

* Indicates that this function is performed in the "Quick-Start" programs. (see next section)

Choose Step 6a for simple applications such as performing a conversion or reading a digital input port. Choose Step 6b for more complex operations.

The following sections will describe the various parts of the flow diagram. However, many programmers will learn faster by referring to the example programs supplied with the ADC-16 Distribution Software.

Quick-Start Program

To aide in the creation of programs, the Distribution Software includes a *Quick-Start* program, which includes all driver loading, pointer definition, and ADC-16 initialization. For interpreted BASIC, the Quick-Start program is **QBASIC.BAS** . The program is in standard BASIC source code with an *Insert Your Code Here* section.

When you program the ADC-16, begin with the Quick-Start program and add new sections to the existing code. This alternative should be easier than rewriting the driver, pointer, and initialization code.

Declaring The Variables

You must declare all program variables before opening the driver. Declare each of the driver functions as integer variables. The end of this section contains a listing of an Interpreted BASIC example program. It may be helpful to refer to this example program during the discussions in this section.

Neglecting to declare all variables will result in a program failure. Declaration examples are as follows:

```
'Declare the driver functions
   DIM DriverArray(22000) AS INTEGER     'Dimension an array to
                                         'hold the driver

   DIM ADC16DEVOPEN AS INTEGER           'Declare the driver
        .                .               'function names
        .                .               'as integer variables
   DIM KRESTORECHNGARY AS INTEGER


'Declare all variables used in the program
   DIM NumOfBrds AS INTEGER  - declare integer variables
   DIM Derr AS INTEGER
   DIM Gain AS INTEGER
   DIM Chan AS INTEGER
   DIM BrdNum AS INTEGER
   DIM A$                    - declare A as a string
   DIM ADValue AS LONG       - declare long variables
   DIM ADC16 AS LONG
   VEL=0:DIRECT=0:TEMP=0     - declare standard real variables
```

Do not Forget!! Declare all variables prior to opening the driver. Using an undeclared variable in the program body (even the **I** in a simple **FOR I= 1 to 10:NEXT I** loop) will cause device pointers to be shifted, and will result in program failures!!

Initializing Pointers & Loading The Driver

In QBASIC, the driver and a number of pointers to the driver are loaded within the program itself. The first task is to load the entry point variables (each command has a unique entry point), using a subroutine in QBASIC.BAS. Since there are several entry points, it is recommended that the QBASIC.BAS Quick-Start program (or one of the example programs) be used as the starting point for new programs. This will ensure that the pointers are loaded correctly. The following statement calls the subroutine which loads the driver entry points.

```
CallSetup    'call subroutine to load driver pointers
```

Next, load the driver.

```
DEF SEG = VARSEG(DriverArray(0))
BLOAD "ADC16Q.BIN",0
```

Opening The Driver

The next step opens the driver and reads the current configuration file for the ADC-16 hardware (Base Address, Interrupt Level, input configuration). (This file is created using the ADC16CFG.EXE file described in Chapter 2.) **ADC16DevOpen** performs this function. **ADC16DevOpen** and the other required initialization functions are shown in the example program at the end of this section. The syntax for this function is as follows:

```
A$ = "ADC16.CFG" + CHR$(0)          'read ADC-16 config file
CALL ABSOULUTE(A$, NumOfBrds, DERR, ADC16DEVOPEN)  'execute the call
```

Establish Communications Between The Driver & Program

To allow the use of multiple boards, the function calls require an identifier for each board. A board identifier is called a *Device Handle* or simply *Handle*. Without the Handle, a **KADRead** command in a multiboard system would not know which board to call. The board Handle is the last parameter in each of the ADC-16 QBASIC calls.

The Handle for each board is returned by the **ADC16GetDevHandle** command after a successful call to **ADC16DevOpen**. This command is included in the Quick-Start programs. Additional information on this function (and concept) is provided in the following paragraphs.

SINGLE BOARD SYSTEMS

In single-board systems the Handle is very straightforward. The Handle is obtained via the **ADC16GetDevHandle** command. The following examples describe how the Handle is used.

First, Get the handle.

```
NumOfBrds% = 0                  '0 since only 1 board in system
CALL ABSOLUTE(NumOfBrds,DevHandle,Derr,ADC16GetDevHandle)
```

(The variable Derr contains the error status)

Subsequent calls use the variable *DevHandle* to select the installed ADC-16.

The variable name *DevHandle* may be changed to make the program easier to read. For example, a variable change from *DevHandle* to *ADC16* is reflected in the following program.

```
BrdNum = 0              '0 since only 1 board in system
CALL ABSOLUTE(BrdNum,ADC16,Derr,ADC16GetDevHandle)
```

MULTIPLE BOARD SYSTEMS

Systems with more than one board must use more than one Handle. The following example shows how to obtain and use the Handles in a system with two ADC-16s.

```
BrdNum = 0          'First board
CALL ABSOLUTE(BrdNum,ADC16A,Derr,ADC16GetDevHandle)

BrdNum = 1           'Second board
CALL ABSOLUTE(BrdNum,ADC16B,Derr,ADC16GetDevHandle)
```

Subsequent calls would use the Handles of either Board (ADC16A or ADC16B) to determine the selected board.

TYPES OF CALL COMMANDS

Two types of Call commands are available to control the ADC-16 once the board and the driver are installed and initialized. These are Immediate Operation commands and Frame based commands.

IMMEDIATE OPERATION COMMANDS

Several single-function commands may be executed without any prior setup. The simplest of these is **KADRead** , which selects an input channel, an input gain, and then performs a single A/D conversion. The syntax for this command is as follows:

For a single A/D conversion on Channel 0 with Gain Range 0, the call would be

```
Chan = 0: Gain = 0
CALL ABSOLUTE(DevHandle, Chan, Gain, ADValue, Derr, KADRead)
PRINT "The A/D returned",ADValue
```

Note that the *ADValue* and the *DevHandle* are *Long Integers* and must be declared in the variable declaration section. The ADC-16 data is returned in the least significant fifteen bits of ADValue, if bit sixteen is zero multiply the data by minus one.

MULTIPLE BOARD SYSTEMS

Systems with more than one board must use more than one Handle. The following example takes an A/D reading from Channel 0 of the first Board and Channel 5 of the second.

```
BrdNum = 0          'First board
CALL ABSOLUTE(BrdNum,ADC16A,Derr,ADC16GetDevHandle)

BrdNum = 1           'Second board
CALL ABSOLUTE(BrdNum,ADC16B,Derr,ADC16GetDevHandle)

Chan = 0: Gain = 0
CALL ABSOLUTE(ADC16A, Chan, Gain, ADValue, Derr, KADRead)
PRINT "The first Board returned",ADValue

Chan = 5: Gain = 0
CALL ABSOLUTE(ADC16B, Chan, Gain, ADValue, Derr, KADRead)
PRINT "The second Board returned",ADValue
```

Note that the first **KADread** uses the first Board Handle as set up with *BrdNum = 0* while the second **KADRead** uses the second Handle.

Other Immediate Operation commands include

KDIRead	Read the digital inputs.
KDOWrite	Write to the digital outputs.

FRAMES

Analog Input and Digital Output allow for operation using Frames. For Digital Output the Frame is a means of returning the value of the Digital Output port. For Analog Input the Frame is the group of all setup parameters used in a synchronous or interrupt driven operation. For example, an A/D function requires setting up the following parameters:

- Input Channels to scan (Start and Stop channels).
- Input Gain.
- Number of samples.
- Data buffer location.

Program flow is similar in each type of Frame, as follows:

First:	Open or get a Frame with **KGetADFrame** or **KGetDOFrame**.
Next:	Setup the parameters within the Frame.
Then:	Execute the desired Frame function.
Optionally:	Check the status of the operation initiated.

The following program segment is similar to the QBASEX7.BAS example in the Distribution Software. It may be helpful to refer to the during the following discussion.

Example using Analog Input Frame

```
' Synchronous A/D scan using Start / Stop / Gain parameters
'-------------------------------------------------------------
' STEP 1 : ALWAYS INCLUDE AND DECLARE THE SUB CallSetup().
'          It is SUPPLIED TO YOU WITHIN THE EXAMPLE PROGRAMS AND
'          MUST BE INCLUDED IN ALL OF YOUR OWN QBASIC PROGRAMS.

   DECLARE SUB CallSetup ()

'-------------------------------------------------------------
' STEP 2 : ALWAYS DIMENSION AN ARRAY TO LOAD THE DRIVER INTO.
'          THE DRIVER WILL OCCUPY APPROXIMATELY 42000 BYTES.

   DIM DriverAry(22000) AS INTEGER ' Array to hold driver Code

'-------------------------------------------------------------
' STEP 3 : ALWAYS INCLUDE THE FOLLOWING DEFINITIONS IN YOUR PROGRAM

   DIM SHARED ADC16DEVOPEN AS INTEGER, ADC16GETDEVHANDLE AS INTEGER
   DIM SHARED KGETADFRAME AS INTEGER, KCLEARFRAME AS INTEGER
   DIM SHARED KGETVER AS INTEGER, KDASDEVINIT AS INTEGER
```

```
   DIM SHARED KFREEFRAME AS INTEGER, KSETBUF AS INTEGER
   DIM SHARED KSETSTARTSTOPG AS INTEGER, KGETSTARTSTOPG AS INTEGER
   DIM SHARED KSETCHN AS INTEGER, KGETCHN AS INTEGER
   DIM SHARED KSETSTARTSTOPCHN AS INTEGER
   DIM SHARED KGETSTARTSTOPCHN AS INTEGER
   DIM SHARED KSETG AS INTEGER, KGETG AS INTEGER
   DIM SHARED KSETCHNGARY AS INTEGER
   DIM SHARED KCHKFRAME AS INTEGER
   DIM SHARED KINITFRAME AS INTEGER, KSYNCSTART AS INTEGER
   DIM SHARED KINTSTART AS INTEGER
   DIM SHARED KINTSTATUS AS INTEGER, KINTSTOP AS INTEGER
   DIM SHARED KGETDOFRAME AS INTEGER, KGETDOCURVAL AS INTEGER
   DIM SHARED KMOVEDATABUF AS INTEGER
   DIM SHARED KADREAD AS INTEGER
   DIM SHARED KDIREAD AS INTEGER, KDOWRITE AS INTEGER
   DIM SHARED KFORMATCHNGARY AS INTEGER
   DIM SHARED KRESTORECHNGARY AS INTEGER

'-------------------------------------------------------------------------
-
' STEP 4 : Define ALL local variables required by the program.
'          Avoid declaring and using QBASIC variables on the fly.

   DIM NUMOFBRDS AS INTEGER      ' rds defined in .CFG file
   DIM ERRFLAG AS INTEGER        ' Holds function error returns
   DIM DEVHANDLE AS LONG         ' Device Handle - all handles are 4 bytes
   DIM ADFRAME AS LONG
   DIM STARTCH AS INTEGER
   DIM STOPCH AS INTEGER
   DIM GAIN AS INTEGER
   DIM X AS INTEGER
   DIM DATABUFFER(20) AS INTEGER
   DIM Samples AS LONG
   DIM BRDNUM AS INTEGER


'-------------------------------------------------------------------------
' STEP 5 : Initialize the function call offsets and load the driver

   CallSetup                          ' Initialize entry variables
   DEF SEG = VARSEG(DriverAry(0))     ' Get location of user array, and
   BLOAD "ADC16Q.BIN", 0              ' Load the driver into user array

'-------------------------------------------------------------------------
' STEP 6 : Initialize the internal data tables with the information
'          contained in the configuration file ADC16.CFG.

   A$ = "ADC16.CFG" + CHR$(0)
   CALL ABSOLUTE(A$, NUMOFBRDS, ERRFLAG, ADC16DEVOPEN)
   IF ERRFLAG <> 0  PRINT "ERROR "; HEX$(ERRFLAG) : STOP

'-------------------------------------------------------------------------
' STEP 7 : Establish communication with the driver through the Device
'          Handle.

   BRDNUM = 0
   CALL ABSOLUTE(BRDNUM, DEVHANDLE, ERRFLAG, ADC16GETDEVHANDLE)
   IF (ERRFLAG <> 0) THEN PRINT "ERROR "; HEX$(ERRFLAG): STOP
'-------------------------------------------------------------------------
' STEP 8 : To perform a FRAME based Analog Input function, first
'          get a FRAME handle and then initialize the parameters for
'          the FRAME.

   CALL ABSOLUTE(DEVHANDLE, ADSET, ERRFLAG, KGETADFRAME)
   IF ERRFLAG <> 0 THEN PRINT "ERROR "; ERRFLAG: STOP
```

```
   STARTCH = 0: STOPCH = 7: GAIN = 0
   CALL ABSOLUTE(ADSET, STARTCH, STOPCH, GAIN, ERRFLAG, KSETSTARTSTOPG)
   IF ERRFLAG <> 0 THEN  PRINT "Error "; ERRFLAG : STOP

   SAMPLES = 0 ' ADC-16 does not use the SAMPLES parameter
   CALL ABSOLUTE(ADSET, DATABUFFER(), SAMPLES, ERRFLAG, KSETBUF)
   IF ERRFLAG <> 0 THEN PRINT "Error "; ERRFLAG : STOP

   CALL ABSOLUTE(ADSET, ERRFLAG, KSYNCSTART)
   IF ERRFLAG <> 0 THEN  PRINT "Error # "; ERRFLAG;  STOP

   CALL ABSOLUTE(ADSET, ERRFLAG, KFREEFRAME)
   IF ERRFLAG <> 0 THEN PRINT "Error "; ERRFLAG;  STOP

   FOR X = 0 TO (STOPCH-STARTCH)
      PRINT HEX$(DATABUFFER(X))
   NEXT

   END

SUB CallSetup

' Initialize entry point Variables

   ADC16DEVOPEN = 0
   ADC16GETDEVHANDLE = 3
   KGETADFRAME = 6
   KGETDOFRAME = 15
   KGETVER = 30
   KDASDEVINIT = 33
   KSETIRQMAP = 36
   KFREEFRAME = 39
   KSETBUF = 42
   KSETSTARTSTOPG = 54
   KGETSTARTSTOPG = 57
   KSETCHN = 60
   KGETCHN = 63
   KSETSTARTSTOPCHN = 66
   KGETSTARTSTOPCHN = 69
   KSETG = 72
   KGETG = 75
   KSETCHNGARY = 78
   KCHKFRAME = 129
   KINITFRAME = 132
   KSYNCSTART = 135
   KINTSTART = 147
   KINTSTATUS = 150
   KINTSTOP = 153
   KGETDOCURVAL = 156
   KMOVEDATABUF = 201
   KADREAD = 213
   KDIREAD = 219
   KDOWRITE = 222
   KFORMATCHNGARY = 225
   KRESTORECHNGARY = 228
   KCLEARFRAME = 231
END SUB
```

The **KGETADFRAME** call is used to open the frame with the variable name ADSET for the ADC-16 board referred to by variable DEVHANDLE. The FRAME is filled with the start-stop channels and gain using the call **KSETSTARTSTOPG**. The data location for the acquisition is established by **KSETBUF**. The number of samples taken in this example is determined by the start and stop channels and the SAMPLES parameter in the **KSETBUF** call is ignored. With all the FRAME parameters completed in ADSET, the acquisition is initiated by a call to **KSYNCSTART**.

A program can define multiple frames. For example, an application may define a FRAME with different start-stop channels or use **KSETCHNGARY** to define a FRAME with the gain and channel sequence specified in a channel-gain array.

6.5 LIST OF CALLS

This list groups the Calls according to the categories of *Driver Initialization* , *General Driver Functions* , *Frame Management*, *Operating Functions* , and *Frame Parameters* . For definitions of Call Driver terms, refer to Section 6.6.

Driver Initialization

These functions are mandatory and are typically called once, at the beginning of the application program, before all other driver function calls. The Driver Initialization functions first verify that the Board is at the selected base I/O address, and then they initialize both the hardware and software. ADC-16 Driver Initialization functions are as follows:

ADC16DevOpen	Initialize a DAS driver using a configuration file.
ADC16GetDevHandle	Get a handle to the logical device associated with an adapter card.
KDASDevInit	Initialize a DAS driver and all aspects of the Board to a defined state.

General Function

KGetVer	Get driver specification and driver revision levels.

Frame Management

To perform multiple-sample Analog I/O or Digital Output, you must first get a Frame from the driver through a Frame Management call. After obtaining a particular frame, you set it up using both the Frame Management and Parameter Functions.

KGetADFrame	Get a logical Frame for Analog to Digital (A/D) operations.
KGetDOFrame	Get a logical Frame for Digital Output operations.
KChkFrame	Check the parameters in a frame for validity.
KFreeFrame	Release a no longer used frame to the driver.
KInitFrame	Initializes the specified frame to a pre-defined state.
KClearFrame	Clears the specified frame.

Operating Functions

IMMEDIATE OPERATIONS

The following functions do not require a Frame.

KADRead	Read a single A/D input value.

KDIRead	Read a single Digital Input value.
KDOWrite	Write a single Digital Output value.

FRAME OPERATIONS

Once a Frame is set up for a particular operation, that operation is ready to be initiated by a Call to one of the following functions.

KSyncStart	Perform a synchronous input operation.
KIntStart	Start an interrupt-driven operation. You should specify Start & Stop Channels, buffering, number of conversions, gain, etc. prior to this call.
KIntStatus	Monitor the interrupt-driven operation associated with a particular frame.
KIntStop	Halt the interrupt-driven operation associated with a particular frame.
KGetDOCurVal	Retrieve the digital output value assigned to a Digital Output frame.

Frame Parameters

After using a Frame Management function to obtain a Frame, you use a Frame Parameter Function to set up its parameters. For example, after using KGetADFrame to obtain an A/D Frame, you would use Frame Parameter functions to set up the analog input channels to be sampled, the gain to apply to these channels, the A/D sampling rate, etc.

KSetChn	Specify a channel for single channel Digital/Analog Input/Output operation.
KGetChn	Obtain the single channel associated with a particular frame. If a multiple channel frame is specified, the start channel is returned.
KSetStartStopChn	Set the Start and Stop channels for A/D input associated with a particular frame.
KGetStartStopChn	Get the Start and Stop channels for A/D input associated with a particular frame.
KSetG	Set the Gain code.
KGetG	Get the Gain code.
KSetStartStopG	Specify the start channel, stop channel for A/D input, and overall gain for a particular frame.
KGetStartStopG	Obtain the start channel, stop channel for A/D input, and overall Gain code associated with a particular frame.KSetChnGAry Insert an A/D channel gain list into a frame.
KFormatChnGAry	Changes format of channel gain array data from integer to unsigned bytes.
KRestoreChnGAry	Changes format of channel gain array data from unsigned bytes to integer.

6.6 GLOSSARY OF CALL TERMS

Frame

A Frame is a board-configuration template for a particular operation; it contains all the parameter information needed to configure the board for that operation. For example, an Analog-to-Digital (A/D) Frame contains parameters for configuring a board to perform an A/D acquisition. A/D Frame parameters include the channel(s) to scan, the gain code, the buffer location for acquired data, and the code for determining whether the acquisition is to be Interrupt driven, etc. A/D operations, Digital Output (DO) operations, etc. each require a separate Frame.

Logical Device

A Device is a grouping of several frames. Thus, a Device may include an A/D Frame, a Digital Input Frame, a Digital Output Frame, etc. If a Device also contains all the fundamental configuration data (Base Address, Interrupt Level, etc.) for a particular board, it is unique to that board and is therefore a Logical Device.

Device Handle

A Device Handle is an identifier for a Logical Device. Each board in a system is assigned its own Device Handle. To obtain a Device Handle for a particular board, use **ADC16GetDevHandle** with the board's physical ID number. This Call returns a Device Handle unique to that board and its Device.

Frame Handle

Once a Logical Device has a Device Handle, you may specify a Frame by a Frame Handle. A Frame Handle is an identifier for a particular Frame and is acquired using **KGetxxFrame** (the xx specifies AD or DO) for A/D input or Digital Output frames.

Memory Handle

A Memory Handle identifies a block of memory assigned for a Frame. Calls that operate on memory require a Memory Handle to designate the particular block of memory to be affected.

■ ■ ■

CHAPTER 7

INDIVIDUAL CALL DESCRIPTIONS

This chapter provides a full description of each Call as used for the BASIC languages (Interpreted BASIC, QuickBASIC, and QBASIC). Note that throughout this manual,

Interpreted BASIC = BASIC , BASICA , and GW BASIC .

Descriptions are in the alphabetical order of the Call names.

Each of the descriptions discusses the following subjects: *Purpose , format , in parameters , out parameters , return values , see also , comments ,* and *examples .*

In each of the Call descriptions, ***format*** is in the form

ReturnValue = **FunctionName(** *Parameter 1 , ••• , Parameter n* **)**

and does not reflect the syntax for any language. Under this scheme, ***examples*** reflect the syntaxes for their respective languages and takes the forms

```
Interpreted BASIC   xx10  CALL FunctionName(Parameter1,•••,ParameterN,ReturnValue)

     QuickBASIC     ReturnValue = FunctionName(Parameter1,•••,ParameterN)

         QBASIC     CALL ABSOLUTE(Parameter1,•••,ParameterN,ReturnValue,FunctionName)
```

♦ ♦ ♦

Note On Handle-Type Variables!

Many of the functions described in this chapter require the use of Handles. In Interpreted BASIC, this variable is a single-precision floating-point value (the default variable type!); however, in QuickBASIC and QBASIC, this value may also be declared as a long integer, as shown in the example programs of this chapter. In either case, a Handle is a 4-byte user-declared variable used strictly by the software driver.

♦ ♦ ♦

ADC16DevOpen

Purpose Driver Initialization. Initialize a DAS driver via a configuration file.

format `DASErr = ADC16DevOpen( CfgFile, NumberOfBoards );`

entry parameters `CfgFile` = you have three options:

1. Specify the name of the file created with the Configuration Program (see Chapter 2 Section 2.5). The address of the ASCII string containing configuration file name is what is actually passed.
2. Specify *-1* to tell the driver you are using the default factory configuration.
3. Specify *0* (NULL pointer) to tell the driver to use the default configuration file name (ADC16.CFG).

`NumberOfBoards` = an integer value that determines the number of boards in the system (*1* or *2*); see *comments* for restrictions.

exit parameters None.

return value `DASErr` = error code number; 0 if no error.

see also ADC16GetDevHandle.

comments The ADC16 driver supports up to two physical boards simultaneously, except from Interpreted BASIC where only one board is supported. If the configuration file defines two boards, Interpreted BASIC supports only the first.

NOTE: RELAYS ARE SET TO NORMALLY CLOSED.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx10  A$ = "ADC16.CFG"+CHR$(0)
xxx20  CALL ADC16DevOpen%(A$,NumberOfBoards%,DASErr%)
```

QuickBASIC

```
A$ = "ADC16.CFG"+CHR$(0)
DASErr% = ADC16DEVOPEN%(SSEGADD(A$), NUMOFBOARDS%)
IF DASErr% <> 0 THEN BEEP: STOP:
```

QBASIC

```
CfgName$ = "ADC16.CFG"+CHR$(0)
CALL ABSOLUTE(CfgName$,NumberOfBoards,DASErr,ADC16DevOpen)
```

♦ ♦ ♦

ADC16GetDevHandle

Purpose Driver Initialization. Get a handle to the logical device associated with an adapter card.

format `DasErr = ADC16GetDevHandle( BoardNum, DevHandle );`

entry parameters `BoardNum` = a board ID number. The first board is *BoardNum = 0*, the last is *BoardNum = NumberOfBoards-1*, where *NumberOfBoards* is obtained from **ADC16DevOpen**.

exit parameters `DevHandle` = a unique handle identifier for a logical DAS device.

return value `DASErr` = error code number; 0 if no error.

see also ADC16DevOpen.

comments **ADC16GetDevHandle** returns a unique device handle for identifying a specific ADC-16 board. Multiple-board applications know how many devices are available for the current session as a result of the previous Call using **ADC16DevOpen**. Associating subsequent Calls with specific boards requires a handle.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx10  BoardNum% = 0
xxx20  CALL ADC16GetDevHandle%(BoardNum%,DevHandle, DASErr%)
```

QuickBASIC

```
DASErr = ADC16GETDEVHANDLE%(0, DevHandle)          'Board #0
IF (DasErr <> 0) THEN BEEP: STOP:
```

QBASIC

```
BoardNum = 0
DIM DevHandle AS LONG
CALL ABSOLUTE(BoardNum,DevHandle,DASErr, ADC16GetDevHandle)
```

♦ ♦ ♦

KADRead

Purpose Operating Function. Read a single A/D input value.

format `DASErr = KADRead(DevHandle, Chan, Gain, ADValue );`

entry parameters *DevHandle* = a unique identifier for a logical DAS device.

Chan = an integer variable that contains the number of the channel from which data is to be read. For A/D frames, use *Chan* in the range of 0 to topchannel, where topchannel = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1.

Gain = analog Gain code selection, according to the following table:

CODE	GAIN
0	1
1	10
2	100

exit parameters *ADValue* = a variable containing the data. In Interpreted BASIC, this value must be previously DIMensioned as an array of two integers. For QBASIC and QuickBASIC, it is DIMensioned as a single long integer.

return value *DASErr* = integer variable (0 = No Error).

see also **KDIRead, KDOWrite.**

comments **KADRead** converts and reads a single A/D value according to the Chan and Gain code parameters. The value returned is the absolute value of the fifteen least significant bits of ADValue. If bit sixteen of ADValue is zero then multiply ADValue by minus one.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx10  DIM ADValue%(2)
xxx20  CALL KADRead%(DevHandle,Chan%,Gain%,ADValue%(0),DASErr%)
```

QuickBASIC

```
DIM ADVALUE AS LONG
DASErr% = KADRead%(DevHandle,Chan%,Gain%,ADVALUE)
IF DASErr% <> 0 THEN BEEP: STOP:
```

QBASIC

```
DIM ADVALUE AS LONG
CALL ABSOLUTE(DevHandle,Chan,Gain,ADValue,DASErr,KADRead)
```

♦ ♦ ♦

KChkFrame

Purpose Frame Management. Check the parameters in a frame for validity.

format `DasErr = KChkFrame( FrameHandle, Type );`

entry parameters *FrameHandle* = frame handle obtained from a previous **KGetADFrame** or **KGetDOFrame** call.

Type = a word-parameter flag specifying the type of operation, as follows:

Type = 0 if checking a frame to be used in Sync Mode.
Type = 1 if checking a frame to be used in Interrupt Mode.

exit parameters None.

return value *DasErr* = integer variable (0 = No Error).

see also **KGetADFrame, KGetDOFrame, KFreeFrame.**

comments **KChkFrame** ensures that frame parameters are within established bounds for the specified type of operation. Specified operation may be Synchronous (initiated by **KSyncStart**) or Interrupt driven (initiated by **KIntStart**). **KChkFrame** is often used during development as a diagnostic tool; it is removed later.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx20  CALL KChkFrame%(FrameHandle,Type%,DASErr%)
```

QuickBASIC

```
DASErr = KChkFrame%(FrameHandle, 0)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC

```
CALL ABSOLUTE(FrameHandle,0,DASErr,KChkFrame)
```

♦ ♦ ♦

KClearFrame

Purpose Frame Management. Clear all frame parameters to their default states.

format `DASErr = KClearFrame( FrameHandle );`

entry parameters *FrameHandle* = frame handle obtained from a previous KGetADFrame call.

exit parameters None.

return value *DASErr* = error number (0 = No Error).

see also KInitFrame.

comments KClearFrame performs a software re-initialization of all parameters within the frame.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xx10 Call KClearFrame%(ADHandle, DASErr%)
```

QuickBASIC
```
DASErr = KClearFrame%(ADHandle)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE (ADHandle, DasErr, KClearFrame)
```

♦ ♦ ♦

KDASDevInit

Purpose General Driver Function. Initialize a DAS driver to a well-defined state.

format `DASErr = KDASDevInit( DevHandle );`

entry parameters `DevHandle` = a unique identifier for a logical DAS device obtained by a previous call to ADC16GetDevHandle.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also ADC16DevOpen, KInitFrame.

comments Initially, **KDASDevInit** is not required since the mandatory call to **ADC16DevOpen** effectively resets the Device parameters to their initial state. Later, **KDASDevInit** may be called as required; for example, when your program has just finished a synchronous acquisition and you want to do an Interrupt operation. By calling **KDASDevInit** you effectively cancel all setups associated with the Synchronous Mode. Calling **KDASDevInit** also stops all currently active operations.

Note that **KDASDevInit** initializes all driver parameters while **KInitFrame** initialize the specified frame.

NOTE: RELAYS ARE SET TO NORMALLY CLOSED.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KDASDevInit%(DevHandle,DASErr%)
```

QuickBASIC
```
DASErr = KDASDevInit%(DevHandle)
IF DASErr% <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(DevHandle,DASErr,KDASDevInit)
```

♦ ♦ ♦

KDIRead

Purpose Operating Function. Read a single Digital Input value.

format `DASErr = KDIRead( DevHandle, Chan, DIValue );`

entry parameters *DevHandle* = a unique identifier for a logical DAS device obtained by a previous call to **ADC16GetDevHandle**.

Chan = number for channel from which data is to be read (always 0 for ADC-16).

exit parameters *DIValue* = a variable that contains the data sample. In Interpreted BASIC, this value must be previously DIMensioned as an array of two integers. For QBASIC and QuickBASIC, it is DIMensioned as a single LONG integer. Bit0 = IP0; Bit1 = IP1.

return value *DASErr* = integer variable (0 = No Error).

see also KDOWrite.

comments **KDIRead** performs a single read operation from the digital input port.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx10  DIM DIValue%(2)
xxx20  CALL KDIRead%(DevHandle,Chan%,DIValue%(0),DASErr%)
```

QuickBASIC

```
DIM DIValue AS LONG
DASErr = KDIRead%(DevHandle, Chan, DIValue)
IF DASErr% <> 0 THEN BEEP: STOP
PRINT "Digital Input value is : "; HEX$(DIValue):
```

QBASIC

```
DIM DIValue AS LONG
CALL ABSOLUTE(DevHandle,Chan,DIValue,DASErr,KDIRead)
```

♦ ♦ ♦

KDOWrite

Purpose Operating Function. Write a single Digital Output value.

format `DASErr = KDOWrite( DevHandle , Chan , DOValue );`

entry parameters *DevHandle* = a unique identifier for a logical DAS device obtained from a previous call to **ADC16GetHandle**.

Chan = number of channel from which data is to be read (always 0 for ADC-16).

DOValue = location of a variable which contains the data sample. In Interpreted BASIC, this value must be previously DIMensioned as an array of two integers. For QBASIC and QuickBASIC, it is DIMensioned as a single LONG integer. Bit0 = OP0; Bit1 = OP1; Bit2 = EX1; Bit3 = EX2; Bit4 = EX4. Digital Output on EX1, EX2, and EX4 is disabled if the ADC-16 is configured (see Chapter 2 Section 2.5) with STA-EX8 boards.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also KGetDOCurVal.

comments **KDOWrite** outputs the specified *DOValue* contents.
KDOWrite writes all the digital outputs at the same time. All data is held in a single long integer in QBASIC and QuickBASIC, or a two element integer array in interpreted BASIC.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx20  CALL KDOWrite%(DevHandle,Chan%,DOValue%(0),DASErr%)
```

QuickBASIC

```
DOValue = 12
DASErr = KDOWrite%(DevHandle, 0, DOValue)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC

```
CALL ABSOLUTE(FrameHandle,Chan,DOValue,DASErr,KDOWrite)
```

♦ ♦ ♦

KFormatChnGAry

Purpose Frame parameter. Reformats a user channel/gain array for use by the ADC-16 driver.

format *DASErr* = **KFormatChnGAry(** *ChanGainArray* **);**

entry parameters *ChanGainArray* = Channel Gain Array in the following form:

OFFSET	TYPE	DESCRIPTION
0	Integer	Number Of Entries In This Table (N/2)
1	Integer	Channel #1
2	Integer	Gain Code For Channel #1
3	Integer	Channel #2
4	Integer	Gain Code For Channel #2
.	.	
.	.	
.	Integer	Channel # (N/2)
N	Integer	Gain Code For Channel # (N/2)

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also **KRestoreChnGAry** .

comments For certain A/D acquisition modes, the user can specify different gains for different input channels using a channel/gain array. This array is passed to the driver using **KSetChnGAry**. For the driver to accept this array, it must be in specific format; this format is not achievable directly from BASIC.

KFormatChnGAry operates on the array that is actually passed to your BASIC program, making it unreadable. To restore the array so that it is readable from BASIC, use the complementary function **KRestoreChnGAry**.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx20  CALL KFormatChnGAry%(ChanGainArray(0),DASErr%)
```

QuickBASIC

```
DIM ChanGainArray(20) AS INTEGER
    :
ChanGainArray(0) = 4                        ' Number of Chan/Gain pairs
ChanGainArray(1) = 0: ChanGainArray(2) = 0     ' Chan 0 Gain x1
ChanGainArray(3) = 1: ChanGainArray(4) = 1     ' Chan 1 Gain x10
ChanGainArray(5) = 2: ChanGainArray(6) = 1     ' Chan 2 Gain x10
ChanGainArray(7) = 3: ChanGainArray(8) = 0     ' Chan 3 Gain x1
DASErr = KFormatChnGAry%(ChanGainArray)
IF DASErr <> 0 THEN BEEP: STOP
    :
```

QBASIC

```
CALL ABSOLUTE(ChanGainArray(),DASErr,KFormatChnGAry)
```

♦ ♦ ♦

KFreeFrame

Purpose Frame Management. Release a frame no longer needed.

format *DASErr* = **KFreeFrame(** *FrameHandle* **);**

entry parameters *FrameHandle* = frame handle obtained from a previous KGetADFrame or KGetDOFrame call.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also KGetADFrame, KGetDOFrame, KChkFrame.

comments None.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KFreeFrame%(FrameHandle,DASErr%)
```

QuickBASIC
```
DASErr = KFreeFrame%(FrameHandle)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,DASErr,KFreeFrame)
```

♦ ♦ ♦

KGetADFrame

Purpose Frame Management. Get a logical Frame for Analog Input (A/D) operations.

format *DASErr* = **KGetADFrame(** *DevHandle, FrameHandle* **);**

entry parameters *DevHandle* = device handle obtained from previous **ADC16GetDevHandle** call.

exit parameters *FrameHandle* = frame identification parameter for use when referencing parameters in the frame or by operations that use the parameters in the frame.

return value *DASErr* = integer variable (0 = No Error).

see also **KChkFrame, KFreeFrame.**

comments **KGetADFrame** returns a handle to a parameter list referred to as a *frame*. Frame parameters describe all aspects of a data acquisition/conversion process. Functions are available to set, select, or enable these parameters. For a given device, each frame is specific to a certain type of operation.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KGetADFrame%(DevHandle,FrameHandle, DASErr%)
```

QuickBASIC
```
DASErr = KGetADFrame%(DevHandle, FrameHandle)
IF (DasErr <> 0) THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(DevHandle,FrameHandle,DASErr, KGetADFrame)
```

♦ ♦ ♦

KGetChn

Purpose Frame Parameter. Obtain the Start Channel associated with a particular frame.

format *DASErr* = **KGetChn(** *FrameHandle, Chan* **);**

entry parameters *FrameHandle* = frame handle obtained from previous KGetADFrame call.

exit parameters *Chan* = integer variable to receive the Start Channel associated with *FrameHandle.*

return value *DASErr* = integer variable (0 = No Error).

see also KSetChn.

comments If the current frame is set up for multiple channels, the Start channel is returned.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx20  CALL KGetChn%(FrameHandle,Chan%,DASErr%)
```

QuickBASIC

```
DASErr = KGetChn(FrameHandle, Chan)
IF DASErr <> 0 THEN BEEP: STOP
PRINT "Analog Input channel is: "; Chan:
```

QBASIC

```
CALL ABSOLUTE(FrameHandle,Chan,DASErr,KGetChn)
```

♦ ♦ ♦

KGetDOCurVal

Purpose Operating Function. Retrieve the Digital Output value currently assigned to the output port of a Digital Output frame.

format `DASErr = KGetDOCurVal( FrameHandle, Value );`

entry parameters *FrameHandle* = frame handle obtained from previous **KGetDOFrame** call.

exit parameters *Value* = long integer containing the current Digital Output value. Bit0 = OP0; Bit1 = OP1; Bit2 = EX1; Bit3 = EX2; Bit4 = EX4.

return value *DASErr* = integer variable (0 = No Error).

see also KSetDOFrame, KDOWrite .

comments **KGetDOCurVal** reads a value up to 5 bits that is currently assigned to the digital output port. This operation can take place while other background A/D operations are in progress.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KGetDOCurVal%(FrameHandle,Value%(0),DASErr%)
```

QuickBASIC
```
DASErr = KGetDOCurVal%(FrameHandle, Value)
IF DASErr <> 0 THEN BEEP: STOP
PRINT "Current Digital Output value is: "; HEX$(Value):
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,Value,DASErr,KGetDOCurVal)
```

♦ ♦ ♦

KGetDOFrame

Purpose Frame Management. Get a logical frame for Digital Output operations.

format `DASErr = KGetDOFrame( DevHandle, FrameHandle );`

entry parameters *DevHandle* = device handle obtained from previous ADC16GetDevHandle call.

exit parameters *FrameHandle* = frame identification parameter for use when referencing parameters in the frame or by operations that use the parameters in the frame. This parameter equals Null (0) if an error condition occurs.

return value *DASErr* = integer variable (0 = No Error).

see also KChkFrame, KFreeFrame.

comments KGetDOFrame returns a frame handle, which identifies a parameter list referred to as a *frame*. Frame parameters describe all aspects of a data acquisition/conversion process. Functions are available to set, select, or enable these parameters. Each frame is specific to a certain type of operation for a specific device.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KGetDOFrame%(DevHandle,FrameHandle,DASErr%)
```

QuickBASIC
```
DASErr = KGetDOFrame%(DevHandle, FrameHandle)
IF FrameHandle = 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(DevHandle,FrameHandle,DASErr, KGetDOFrame)
```

♦ ♦ ♦

KGetG

Purpose Frame Parameter. Get the Gain code for the frame.

format `DASErr = KGetG( FrameHandle, Gain );`

entry parameters *FrameHandle* = frame handle obtained from previous KGetADFrame call.

exit parameters *Gain* = integer variable containing the overall (applied to all Analog Input channels) Gain code associated with the frame .

CODE	GAIN
0	1
1	10
2	100

return value *DASErr* = integer variable (0 = No Error).

see also KSetG.

comments KGetG returns the global Gain code currently applied to all Analog Input channels.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KGetG%(FrameHandle,Gain%,DASErr%)
```

QuickBASIC
```
DASErr = KGetG%(FrameHandle, GainCode)
IF DASErr <> 0 THEN BEEP: STOP
PRINT "The Global Gain Code is: "; GainCode:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,Gain,DASErr,KGetG)
```

♦ ♦ ♦

KGetStartStopChn

Purpose Frame Parameter. Get the Start and Stop channels associated with a particular frame.

format `DASErr = KGetStartStopChn( FrameHandle, StartChn, StopChn );`

entry parameters *FrameHandle* = frame handle obtained from previous KGetADFrame call.

exit parameters *StartChn* = integer variable containing the start channel associated with FrameHandle. Limits = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1.

StopChn = integer variable containing the stop channel associated with FrameHandle. Limits = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1.

return value *DASErr* = integer variable (0 = No Error).

see also KSetStartStopChn.

comments None.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KGetStartStopChn%(FrameHandle,StartChn%,StopChn%,DASErr%)
```

QuickBASIC
```
DASErr = KGetStartStopChn%(FrameHandle, StartChn, StopChn)
IF DASErr <> 0 THEN BEEP: STOP
PRINT "The Channel Scan is "; StartChn; " to "; StopChn:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,StartChn,StopChn,DASErr,KGetStartStopChn)
```

♦ ♦ ♦

KGetStartStopG

Purpose Frame Parameter. Obtain the start channel, stop channel and overall gain code associated with a particular frame.

format `DASErr = KGetStartStopG( FrameHandle, StartChn, StopChn, Gain );`

entry parameters *FrameHandle* = frame handle obtained from previous KGetADFrame call.

exit parameters *StartChn* = integer variable containing the start channel associated with *FrameHandle*. Limits = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1.

StopChn = integer variable containing the stop channel associated with *FrameHandle*. limits = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1.

Gain = integer variable containing the overall gain code associated with *FrameHandle*.

CODE	GAIN
0	1
1	10
2	100

return value *DASErr* = integer variable (0 = No Error).

see also KSetStartStopG.

comments None.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KGetStartStopG%(FrameHandle,StartChn%,StopChn%,Gain%,DASErr%)
```

QuickBASIC
```
DASErr = KGetStartStopG%(FrameHandle, StartChn, StopChn, GainCode)
IF DASErr <> 0 THEN BEEP: STOP
PRINT "Start: "; StartChn; " Stop: "; StopChn; " GainCode: "; GainCode:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,StartChn,StopChn,Gain,DASErr,KGetStartStopG)
```

♦ ♦ ♦

KGetVer

Purpose General Driver Function. Get driver specification and driver revision levels.

format `DASErr = KGetVer( DevHandle, pSpecVer, pDriverVer );`

entry parameters *DevHandle* = device handle obtained from previous ADC16GetDevHandle call.

exit parameters *pSpecVer* = an integer containing the version of driver specification to which a particular driver adheres.

pDriverVer = an integer containing the revision level of a particular driver (that is, 1.00 being initial release).

return value *DASErr* = integer variable (0 = No Error).

see also None.

comments Both the *pSpecVer* and *pDriverVer* integers are returned in the same format. The version number is represented in the form *y.xx*. The *y* value (major Rev. #) is in the high byte of the integer, and the *xx* value (minor Rev. #) is contained in the low byte. The major version number is the result of dividing the returned integer by 256, while the minor version number is the result of taking the modulus of the returned integer (pSpecVer MOD 256).

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KGetVer%(DevHandle,pSpecVer%,pDriverVer%,DASErr%)
```

QuickBASIC
```
DASErr = KGetVer%(DevHandle, SpecVer, DrvVer)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(DevHandle,pSpecVer,pDriverVer,DASErr,KGetVer)
```

♦ ♦ ♦

KInitFrame

Purpose Frame Management. Initializes the specified frame.

format `DASErr = KInitFrame( FrameHandle );`

entry parameters *FrameHandle* = frame handle obtained from previous **KGetADFrame** or **KGetDOFrame** call.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also **KDASDevInit**.

comments **KInitFrame** initializes a frame to its initial default state. When a device is newly opened, this Call is not required because **ADC16DevOpen** opens and reads the configuration file, setting all the frame parameters to their initial state. **KInitFrame** may be used any time an operation is not in process to reset the parameters to that state.

KDASDevInit initializes all device and driver functions while **KInitFrame** initializes only the current frame.

KinitFrame must be preceded by **KGetADFrame** or **KGetDOFrame** to obtain *FrameHandle* .

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KInitFrame%(FrameHandle,DASErr%)
```

QuickBASIC
```
DASErr = KInitFrame%(FrameHandle)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,DASErr,KInitFrame%)
```

♦ ♦ ♦

KIntStart

Purpose Operating Functions. Start an operation and transfer samples using Interrupts.

format `DASErr = KIntStart( FrameHandle );`

entry parameters *FrameHandle* = frame handle obtained from previous **KGetADFrame** call.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also **KIntStatus, KIntStop.**

comments **KIntStart** starts an A/D operation according to various parameters previously setup through other appropriate functions. This function uses the PC and ADC-16 Interrupt capabilities to transfer the acquired data to the PC memory at speeds limited to approximately 16 Hz. This limitation occurs because of the maximum sampling rate of the integrating A/D converter. Because these transfers occur in the background, other DAS function may be performed simultaneously (for example, **KDIRead** or **KDOWrite**).

Once this operation is started, use **KIntStatus** to monitor the progress of the operation and **KIntStop** to terminate the operation prior to its normal finish (after the requested number of samples is reached).

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KIntStart%(FrameHandle,DASErr%)
```

QuickBASIC
```
DASErr = KIntStart%(FrameHandle)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,DASErr,KIntStart)
```

♦ ♦ ♦

KIntStatus

Purpose Operating function. Monitor the status and progress of an Interrupt operation associated with a particular frame.

format `DASErr = KIntStatus( FrameHandle, Status, Count );`

entry parameters *FrameHandle* = frame handle obtained from previous **KGetADFrame** call.

exit parameters *Status* = integer containing status information (1 = Interrupt operation active, 0 = Interrupt operation idle).

Count = long integer to receive current sample transfer count (samples transferred so far).

return value *DASErr* = integer variable (0 = No Error).

see also **KIntStart, KIntStop.**

comments **KIntStatus** monitors the status of a previously started Interrupt operation using **KIntStart.** Use **KIntStop** to terminate the Interrupt operation prior to its normal completion (when the requested number of samples is reached).

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx10  CALL KIntStatus%(FrameHandle,Status%,Index%(0),DASErr%)
```

QuickBASIC
```
DO
  DASErr = KIntStatus%(FrameHandle, Status, Count)
  IF DASErr <> 0 THEN BEEP: STOP
LOOP WHILE STATUS <> 0   ' Wait until status is false:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,Status,Count,DASErr,KIntStatus)
```

♦ ♦ ♦

KIntStop

Purpose Operating function. Terminate a background Interrupt operation.

format `DASErr = KIntStop( FrameHandle, Status, Count );`

entry parameters *FrameHandle* = frame handle obtained from previous **KGetADFrame** call.

exit parameters *Status* = integer to receive status information (1 = active, 0 = idle).

Count = user long integer to receive current sample transfer count (samples transferred so far).

return value *DASErr* = integer variable (0 = No Error).

see also **KIntStart, KIntStatus.**

comments **KIntStop** is used whenever you wish to prematurely terminate an Interrupt operation initiated using **KIntStart.** If an Interrupt operation is in process, it is stopped and the number of transferred samples is returned in *Count* ; otherwise this function does nothing.

◆ ◆ ◆

Examples

Interpreted BASIC
```
xxx10  CALL KIntStop%(FrameHandle,Status%,Count,DASErr%)
```

QuickBASIC
```
DASErr = KIntStop%(FrameHandle, Status, Count)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,Status,Count,DASErr,KIntStop)
```

◆ ◆ ◆

KMoveDataBuf

Purpose Memory Management. Move N bytes from one memory region to another.

format `DASErr = KMoveDataBuf( DestSeg, DestOff, SrcSeg, ScrOff, Samples );`

entry parameters *DestSeg* = integer Segment of the destination address.

DestOff = integer Offset of the destination address.

SrcSeg = integer Segment of the source address.

SrcOff = integer Offset of the source address.

Samples = integer for the number of bytes to transfer.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also KSetBuf.

comments **KMoveDataBuf** allows Interpreted BASIC users to transfer data to/from an internal buffer (not easily accessible from BASIC) to/from a BASIC user array. You would normally use this Call after a **KIntStart** operation ends.

Refer to the examples below for steps on how to transfer the first 100 A/D samples from an internal buffer to a BASIC array. Note that the internal buffer at &H8000:0 was used to acquire data in a previous A/D operation.

If the Destination or Source Buffer are DIMensioned arrays, the Segment value should be set to &HFFFF.

♦ ♦ ♦

Examples

Interpreted BASIC

```
DIM BUF%(100)
     :
BUFSEG% = VARSEG(BUF%(0))      'Segment of Basic array
BUFOFS% = VARPTR(BUF%(0))      'Offset of Basic Array
            CALL KMoveDataBuf%(BUFSEG%,BUFOFS%,&H8000,0,100,DASErr%)
IF DASErr% <> 0 THEN BEEP: STOP
   :
```

♦ ♦ ♦

KRestoreChnGAry

Purpose Frame parameter. Restore a user channel/gain array previously modified by **KFormatChnGAry** .

format `DASErr = KRestoreChnGAry( ChanGain(0) );`

entry parameters *ChanGain(0)* = array of Channel/Gain pairs to be restored.

exit parameters *ChanGain(0)* = restored array of Channel/Gain pairs.

return value *DASErr* = integer variable (0 = No Error).

see also KFormatChnGAry.

comments **KRestoreChnGAry** restores the user's channel/gain array previously modified by **KFormatChnGAry** such that it is readable again from BASIC.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx20  CALL KRestoreChnGAry%(ChanGain%(0),DASErr%)
```

QuickBASIC

```
DIM ChanGain(20) AS INTEGER
    :
DASErr = KRestoreChnGAry%(ChanGain(0))
IF DASErr% <> 0 THEN BEEP: STOP
    :
```

QBASIC

```
CALL ABSOLUTE(ChanGain(),DASErr,KRestoreChnGAry)
```

♦ ♦ ♦

KSetBuf

Purpose Memory Management. Assigns a data buffer and number of samples to a particular frame.

format **Interpreted BASIC:**

```
DASErr = KSetBuf( FrameHandle, BufSeg, BufOff, Samples );
```

QuickBASIC & QBASIC:

```
DASErr = KSetBuf( FrameHandle, BufAddr, Samples );
```

entry parameters *FrameHandle* = frame handle obtained from previous KGetADFrame call.

BufSeg = integer Segment value of a memory area to be assigned to the frame (for Interpreted BASIC only). If using an array created by DIMension, use the parameter *DataSeg%* instead of *BufSeg*. *DataSeg* is a variable declared in the initialization subroutine; it indicates that BASIC's local data-segment should be used.

BufOff = integer Offset value of a memory area to be assigned to the frame (for Interpreted BASIC only).

BufAddr = long integer address value of memory area to be assigned to the frame (for QuickBASIC and QBASIC only).

Samples = long integer specifying the number of samples associated with the data buffer. The Samples parameter is not used in the ADC-16. The number of samples is determined by the start-stop channels or a channel-gain array.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also KSetChnGAry

comments **KSetBuf** assigns the address of a memory location and the number of samples to the specified frame. For Interpreted BASIC, the memory address is passed as absolute Segment and Offset value (for example, &H8000:0).

♦ ♦ ♦

Examples

Interpreted BASIC

```
BuffOff%=VARPTR(Buffer%(0)) : Count%=0     'Not used.
xx20  CALL KSetBuf%(FrameHandle,DataSeg%,BuffOff%,Count%,DasErr%)
xx30  IF DASErr% <> 0 THEN BEEP: STOP
```

QuickBASIC

```
DIM Buffer(100) AS INTEGER
    :
DASErr = KSetBuf%(FrameHandle,Buffer(0),Address,Samples(0))
IF DASErr <> 0 THEN BEEP: STOP
    :
```

QBASIC

```
CALL ABSOLUTE(FrameHandle,Buffer(),Samples,DASErr,KSetBuf)
```

♦ ♦ ♦

KSetChn

Purpose Frame Parameter. Specify a channel for single channel operations.

format `DASErr = KSetChn( FrameHandle, Chan );`

entry parameters *FrameHandle* = frame handle obtained from previous **KGetADFrame** call.

Chan = integer variable containing a channel number. For A/D frames, use *Chan* in the range of 0 to topchannel, where topchannel = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1. For Digital I/O, *Chan* is always 0.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also KSetStartStopChn.

comments **KSetStartStopChn** is used to specify a range of channels for **KSyncStart** or **KIntStart** operations. For instance, several A/D channels may be monitored sequentially by specifying independent Start and Stop Channels. If StartChn = 2 and StopChn = 4, then samples will be taken from Channels 2, 3, 4.

KSetChn may be used to reset the start channel number after **KSetStartStopChn** has established a channel range.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KSetChn%(FrameHandle,Chan%,DASErr%)
```

QuickBASIC
```
DASErr = KSetChn%(FrameHandle, Chan)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,Chan,DASErr,KSetChn)
```

♦ ♦ ♦

KSetChnGAry

Purpose Frame Parameter. Insert a channel gain list into a frame.

format *DASErr* = **KSetChnGAry(** *ChanGainArray* **);**

entry parameters *ChanGainArray* = Channel Gain Array in the following form:

OFFSET	TYPE	DESCRIPTION
0	Integer	Number Of Entries In This Table N/2)
1	Integer	Channel #1
2	Integer	Gain Code For Channel #1
3	Integer	Channel #2
4	Integer	Gain Code For Channel #2
.	.	
.	.	
.	Integer	Channel # (N/2)
N	Integer	Gain Code For Channel # (N2)

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also **KGetChnGAry.**

comments For certain A/D acquisition modes, you may specify different gains for different input channels using a channel/gain array. This array is passed to the driver using **KSetChnGAry**. The driver accepts this array only when it is in the format shown above, which is not achievable from BASIC; BASIC does not support byte-sized variables. From BASIC, always use **KFormatChnGAry** before **KSetChnGAry**, as shown in the examples below. When no longer needed by the driver, this array may restored to its BASIC format using **KRestoreChnGAry**. Using **KSetChnGAry**, you may specify an arbitrary sequence of channel/gain pairs. Any random order of channels and gains is allowable, including sampling the same channel sequentially at different gains. The channel-gain array may contain up to 256 channel/gain entries.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KSetChnGAry%(ChanGainArray%(0),DASErr%)
```

QuickBASIC
```
DIM ChanGain(20) AS INTEGER
    :
ChanGain(0) = 4          ' Number of Chan/Gain pairs
ChanGain(1) = 0: ChanGain(2) = 0     ' Chan 0 Gain Code 1
ChanGain(3) = 1: ChanGain(4) = 1     ' Chan 1 Gain Code 2
ChanGain(5) = 2: ChanGain(6) = 1     ' Chan 2 Gain Code 2
ChanGain(7) = 3: ChanGain(8) = 0     ' Chan 3 Gain Code 1
DASErr = KSetChnGAry%(FrameHandle, ChanGain(0))
IF DASErr <> 0 THEN BEEP: STOP
    :
```

QBASIC
```
CALL ABSOLUTE(ChanGain(),DASErr,KSetChnGAry)
```

♦ ♦ ♦

KSetG

Purpose Frame Parameter. Set the overall analog input gain.

format `DASErr = KSetG( FrameHandle, Gain );`

entry parameters *FrameHandle* = frame handle obtained from previous KGetADFrame call.

Gain = integer which sets the overall gain code (0, 1, or 2) associated with *FrameHandle*.

CODE	GAIN
0	1
1	10
2	100

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also KGetG, KStartStopG, KSetChnGAry.

comments KSetG sets the analog gain for a specified frame.

KSetStartStopG sets the start and stop channel numbers and the gain for a specified frame. This function is an expedient way to combine KSetStartStopChn and KSetG.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx20  CALL KSetG%(FrameHandle,Gain%,DASErr%)
```

QuickBASIC

```
DASErr = KSetG%(FrameHandle, 0)        ' Set Global Gain to x1
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC

```
CALL ABSOLUTE(FrameHandle,Gain,DASErr,KSetG)
```

♦ ♦ ♦

KSetStartStopChn

Purpose Frame Parameter. Set the Start and Stop channels associated with a particular frame.

format `DASErr = KSetStartStopChn( FrameHandle, StartChan, StopChan );`

entry parameters *FrameHandle* = frame handle obtained from previous **KGetADFrame** call.

StartChan = integer start channel to be associated with FrameHandle. Limit = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1.

StopChan = integer stop channel to be associated with FrameHandle. Limit = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also **KGetStartStopChn, KSetChn.**

comments **KSetChn** is used to specify a channel for a single channel operation. As an example, **KSetChn** would typically be used to specify the channel for **KADRead** .

KSetStartStopChn is used to specify a range of channels for **KSyncStart** or **KIntStart** operations. For instance, several A/D channels may be monitored sequentially by specifying independent Start and Stop Channels. If StartChn = 2 and StopChn = 4, then samples will be taken from Channels 2, 3, 4.

KSetChn may also be used to reset the start channel number after **KSetStartStopChn** has established a channel range.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KSetStartStopChn%(FrameHandle,StartChan%,StopChan%,DASErr%)
```

QuickBASIC
```
DASErr = KSetStartStopChn%(FrameHandle, 0, 3)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,StartChan,StopChan,DASErr,KSetStartStopChn)
```

♦ ♦ ♦

KSetStartStopG

Purpose Frame Parameter. Specify the start channel, stop channel and overall gain for a particular frame.

format `DASErr = KSetStartStopG( FrameHandle, StartChan, StopChan, Gain );`

entry parameters *FrameHandle* = frame handle obtained from previous **KGetADFrame** call.

StartChan = the start channel associated with FrameHandle. Limit = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1.

StopChan = the stop channel associated with FrameHandle. Limit = ((# of STA-EX8s * 8) + (8 - # of STA-EX8s)) - 1.

Gain = the overall gain code associated with an operation.

CODE	GAIN
0	1
1	10
2	100

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also **KGetStartStopG.**

comments **KSetG** sets the analog gain for a specified frame.

KSetStartStopG sets the start and stop channel numbers and the gain for a specified frame. This function is an expedient way to combine **KSetStartStopChn** and **KSetG** .

KSetChnGAry selects an arbitrary sequence of channel sampling with selectable gain settings. The Call receives an array of channel numbers and gain settings, and it scans through the list sequentially. Therefore, any random order of channels and gain settings is allowable, including sampling the same channel sequentially at different gain settings. All the conceptual rules that combined with **KSetStartStopChn** apply to this Call.

♦ ♦ ♦

Examples

Interpreted BASIC

```
xxx20  CALL KSetStartStopG%(FrameHandle,StartChan%,StopChan%,Gain%,DASErr%)
```

QuickBASIC

```
DASErr = KSetStartStopG%(FrameHandle, 0, 3, 0)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC

```
CALL ABSOLUTE(FrameHandle,StartChan,StopChan,Gain,DASErr,KSetStartStopG)
```

♦ ♦ ♦

KSyncStart

Purpose Operating Function. Starts a synchronous A/D operation.

format `DASErr = KSyncStart( FrameHandle );`

entry parameters *FrameHandle* = frame handle obtained from previous KGetADFrame call.

exit parameters None.

return value *DASErr* = integer variable (0 = No Error).

see also KIntStart.

comments Starts a synchronous A/D operation. **KSyncStart** takes a frame handle as an input parameter. This handle is obtained by a **KGetADFrame** function and therefore defines the type of operation.

KSyncStart is a foreground operation and does not return until the specified number of acquisitions/conversions are complete. Therefore, no complimentary status or stop functions are required.

Prior to making this call, the user should have already specified the start and stop channels, buffering, gain, etc.

♦ ♦ ♦

Examples

Interpreted BASIC
```
xxx20  CALL KSyncStart%(FrameHandle,DASErr%)
```

QuickBASIC
```
DASErr = KSyncStart%(FrameHandle)
IF DASErr <> 0 THEN BEEP: STOP:
```

QBASIC
```
CALL ABSOLUTE(FrameHandle,DASErr,KSyncStart)
```

■ ■ ■

CHAPTER 8

REGISTER - LEVEL I/O MAPS

8.1 INTRODUCTORY INFORMATION

The ADC-16 is programmable at the register level using I/O (Input/Output) instructions. In BASIC, the I/O instructions are **`INP(X)`** and **`OUT X,Y`**. In Assembly and most other high-level languages, the I/O instructions are similar; for example, the Assembly Language equivalents are **`IN AL,DX`** and **`OUT DX,AL`**.

As an aid to register-level programming, this chapter describes each ADC-16 register in terms of function, address, bit structure, and bit functions. The chapter does not go into any programming detail since it is likely to vary too greatly from person to person.

8.2 I/O REGISTER ADDRESS MAP

The ADC-16 uses four consecutive Base Addresses in the computer I/O space, as shown in the following table. Note that in the table R = read, and W = write.

LOCATION	FUNCTION	TYPE
Base Address +0	A/D Data High Byte	R
	Start A/D	W
Base Address +1	A/D Data Low Byte	R
Base Address +2	Mux & Gain Register	R/W*
Base Address +3	Status Register	W
	Control Register	R*

* Cleared at power-up.

8.3 A/D REGISTERS (BASE ADDRESS +0 & +1)

While writing to Base Address +0 initiates an A/D conversion, neither Base Address +0 nor Base Address +1 will accept data. Both registers are read-only. However, writing to Base Address +0 initiates an A/D conversion. Data format of the two A/D Data Byte Registers at Base Address +1 and +2 is as follows:

Base Address +0

BIT:	D7	D6	D5	D4	D3	D2	D1	D0
WRITE:	X	X	X	X	X	X	X	X

A write does not register, but it initiates an A/D conversion.

BIT:	D7	D6	D5	D4	D3	D2	D1	D0
READ:	O/P	B14	B13	B12	B11	B10	B9	B8

O/P = Overrange/Polarity bit; is controlled by Bit D6 of the Control Register.

When D6 of the Control Register = 1: O/P = 1 input is Overrange; O/P = 0 input is In Scale.

When D6 of the Control Register = 0: O/P = 1 data is Positive; O/P = 0 data is Negative.

B8-B14 = Data bits.

Base Address +1

BIT:	D7	D6	D5	D4	D3	D2	D1	D0
READ:	B7	B6	B5	B4	B3	B2	B1	B0

B0-B7 = Data bits.

Data is readable from the A/D converter only when an A/D conversion is not in process. Always check the A/D BUSY Bit in the Status Register (Base Address +3) before initiating a conversion or reading the result.

8.4 MUX & GAIN REGISTER (BASE ADDRESS +2)

The MUX & Gain Register is a read/write register that clears to zeroes on power-up or whenever RESET is asserted. Format of the data is as follows:

Base Address +2

BIT:	D7	D6	D5	D4	D3	D2	D1	D0
READ/WRITE:	G1	G0	CH4	CH2	CH1	EX4	EX2	EX1

G0 & G1 These bits select the scaled gain of the programmable input amplifier as follows:

G1	G0	GAIN
0	0	1
0	1	10
1	0	100

CH1-CH4 These bits control the multiplexer, and they determine the channel to be connected to the A/D input amplifier as follows:

CH4	CH2	CH1	CHANNEL
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

EX1-EX4 These bits select the STA-EX8(s) multiplexer channel. (See Chapter 2 for more details.) If the ADC-16 is not used with an STA-EX8, these bits may be used as general purpose digital outputs.

NOTE: To allow the input amplifier time to settle, select the channel and gain about 50 microseconds before an A/D conversion is initiated. If you are programming in Interpreted BASIC, the software delays will more than exceed the required amplifier settling time. However, if you are using a compiled language or Assembly, program a small delay loop into the code for maximum accuracy.

8.5 CONTROL REGISTER (BASE ADDRESS +3)

The Control Register is a write register; it controls the operating modes of the board. The register clears to zeroes on power-up (reset). Bits function as follows:

Base Address +3

BIT:	D7	D6	D5	D4	D3	D2	D1	D0
WRITE:	INTE	O/P	OP1	OP0	INT4	INT2	INT1	INT0

INTE = Interrupt Enable. Enables or disables an ADC-16 interrupt request to the computer system's bus; where INTE = 1 = enabled and INTE = 0 = disabled.

O/P = Overrange/Polarity Bit Select. If set high (1), overrange data is returned in D7 of the A/D High Register (Base Address + 0). If this bit is cleared (0), polarity data is be returned in D7 of the A/D High Register.

OP0 & OP1 = Digital Output. These bits also drive ADC-16 relays REL1 (OP1) and REL0 (OP0).

INT0-INT4 = Interrupt Level Select. Avoid using a level already assigned to another I/O device. Interrupts are chosen as follows:

INT4	INT2	INT1	INT0	INTERRUPT LEVEL
0	0	0	0	Disabled
0	0	0	1	Disabled
0	0	1	0	Level 2 (XT, AT)
0	0	1	1	Level 3 (XT, AT)
0	1	0	0	Level 4 (XT, AT)
0	1	0	1	Level 5 (AT)
0	1	1	0	Disabled
0	1	1	1	Level 7 (XT, AT)
1	0	0	0	Disabled
1	0	0	1	Level 9 (AT)
1	0	1	0	Level 10 (AT)
1	0	1	1	Level 11 (AT)
1	1	0	0	Level 12 (AT)
1	1	0	1	Disabled
1	1	1	0	Disabled
1	1	1	1	Level 15 (AT)

NOTE: INTE, OP0, and OP1 bits are write only. It would be prudent to keep a variable that contains the status of these bits. These bits could then be read to determine present status; they should be updated before they are written to the control register. Also, set OP0 and OP1 to 0 during initialization, to ensure that the variable reflects the true state of the bit.

8.6 STATUS REGISTER (BASE ADDRESS +3)

The Status Register is a read-only register that provides information on the operation and configuration of the A/D in the ADC-16. Writing to the Status Register address clears the ADC-16 interrupt request and provides the means of acknowledging the ADC-16 interrupt and re-enabling it. The format is as follows:

Base Address +3

BIT:	D7	D6	D5	D4	D3	D2	D1	D0
READ:	BUSY	IRQ	IP1	IP0	INT4	INT2	INT1	INT0

BUSY = A/D BUSY. This bit reflects the state of the A/D converter. If BUSY = 1, then the A/D is performing a conversion. If BUSY = 0, the A/D is ready to begin an A/D conversion or to read data.

IRQ = Interrupt Request. Generated even if bus interrupts are disabled (INTE bit of the Control Register = 0). If this bit equals 1, the A/D is finished and data is ready. Read the Status Register to clear the IRQ bit.

IP0 & IP1 = Digital Inputs.

INT0-INT4 = Interrupt Level Select. Reflects the Interrupt Level as specified in the Control Register.

8.7 TYPICAL PROGRAMMING SEQUENCE

A sample programming sequence for performing an A/D conversion on the ADC-16 is as follows:

1. Write to MUX/Gain Register (Base Address +2) to select desired channel and gain.
2. Poll the Status Register (read Base Address +3) to check that the D7 (BUSY) bit is zero.
3. Start A/D conversion by writing any data to Base Address +0.
4. Poll the Status Register (read Base Address +3) until D7 (BUSY) bit is zero.
5. Set D6 in the Control Register (write Base Address +3) to 1 to read overrange.
6. Read the A/D high byte (read Base Address +0).
7. If Overrange Bit D7 is high, an input-overload condition exists.
8. Clear D6 in the Control Register (read Base Address +3) to 0 to read polarity.
9. Read the A/D high byte and sign (read Base Address +0).
10. Read the A/D low byte (read Base Address +1).
11. Assemble high/low byte data into one signed data word (16-bit integer).
12. Repeat cycle for another conversion, gain, channel, etc.

Since the A/D converter performs a conversion between Steps 3 and 4 of the above sequence, there is always a delay of about 50 milliseconds between these two steps before BUSY goes low and the conversion is completed. An example BASIC program that performs these steps is in the following section.

8.8 BASIC EXAMPLE PROGRAM

```
10 'REG.BAS", a 'neat save trick: type EDIT 10, F4, CR
20 FALSE=0:TRUE=NOT FALSE
30 CHAN%=0  ' start on channel0; gain = 1
40 DONE%=FALSE
50 WHILE ((INP(&H313) AND &H80) = &H80  'check for not busy
60 WEND
70 WHILE (NOT DONE%)
80  OUT &H312, CHAN%
90  OUT &H310, 0  ' start conv
100   WHILE ((INP(&H313) AND &H80) = &H80)
110   WEND
120   OUT &H313, &H40  'set overrange ON
130   OVR%=INP(&H310)  'read overrange bit
140   IF(OVR% AND &H80) = &H80 THEN PRINT "OVER RANGE on Channel  "; CHAN%
150   OUT &H313, 0 'reset overrange bit to polarity
160   LOW=INP(&H311)
170   HI = INP(&H310)
```

```
180   HI = (HI AND &H7F)
190   RDG = (256 * HI) OR LOW
200   PRINT "reading = "; (RDG / 32767) * 5
210   FOR DELAY = 0 TO 5000; NEXT
220   CHAN%=CHAN%+8
230  IF CHAN% > 65 THEN DONE%=TRUE
240  WEND
```

■ ■ ■

CHAPTER 9

CALIBRATION

9.1 CALIBRATION INTERVAL

Periodic re-calibration of the ADC-16 is necessary for accuracy, with the interval of recalibration based on type of service. For example, an environment with frequent large changes of temperature and/or vibration would call for a 3-month re-calibration interval, while laboratory or office conditions would call for six months to one year.

9.2 CALIBRATION PROGRAM

Your Distribution Software contains the calibration program *ADC16CAL.EXE* is provided to step you through the calibration process. To run ADC16CAL.EXE, log to the ADC-16 directory and type ADC16CAL . From the opening menu, select as follows:

- ***Change Base Address*** - To find out how to reset the Base Address Switch. The default setting is 300h (768 decimal).
- ***Main Menu*** - To access the calibration routines. These routines consist of calibrating the 1 mA source (**Set 1 mA Source**) and calibrating the A/D converter (**Cal A/D**). There is also a utility program (select **A/D Utility**); this program allows a check of the ADC-16 configuration, and it performs A/D measurement, and controls the DI and DO.
- ***ADC-16 Info*** - To find out more about the ADC-16.

9.3 REQUIRED TEST EQUIPMENT

Calibration of the ADC-16 requires the following equipment:

Set 1 mA Source Procedure

STA-EX8 and C-1800 cable

DMM (Keithley Instruments 196 or equivalent)

1K, 0.1%, 1/4 watt (min) resistor (optional)

A/D Calibration Procedure

STA-EX8 and C-1800 cable

DC Voltage Calibrator, EDC type or equivalent

More test equipment specifications are given in the ADC-16 Info screen.

■ ■ ■

CHAPTER 10

FACTORY RETURNS

Before returning any equipment for repair, please call 508/880-3000 to notify Keithley MetraByte's technical service personnel. If possible, a technical representative will diagnose and resolve your problem by telephone. If a telephone resolution is not possible, the technical representative will issue you a Return Material Authorization (RMA) number and ask you to return the equipment. Please reference the RMA number in any documentation regarding the equipment and on the outside of the shipping container.

Note that if you are submitting your equipment for repair under warranty, you must furnish the invoice number and date of purchase.

When returning equipment for repair, please include the following information:

1. Your name, address, and telephone number.
2. The invoice number and date of equipment purchase.
3. A description of the problem or its symptoms.

Repackage the equipment. Handle it with ground protection; use its original anti-static wrapping, if possible.

Be sure to reference the RMA number on the outside of the package!

■ ■ ■

Appendix A

Summary Of Error Codes

The following list contains the ADC-16 Error Codes with corresponding numbers and definitions.

DAS Shell Codes

The following code numbers signify the errors you might encounter after making DAS Shell Calls.

Error 6000: Bad Configuration File

Returned if the Device Configuration File has an undefined word or file is corrupt.

Error 6001: Bad Base Address

Returned if the specified Base Address is out of range. This error is typically returned during the initialization process.

Error 6004: Error Opening Configuration File.

Returned when Device Configuration does not open.

Error 6005: Illegal Channel Number.

Returned when the specified channel number is out of range.

Error 6006: Illegal Gain

Returned if the specified Gain is out of range.

Error 6009: Wrong Version

Returned in response to an attempt to initialize a device with an incorrect version number.

Error 600A: Configuration Not Found

Returned when the specified Device Configuration File could not be found.

Error 600D: Bad Handle

Illegal handle for a frame.

Error 7000: No Board Name

No Board name was specified.

Error 7001: Bad Board Name

An illegal board name was specified. Legal name is ADC16.

Error 7002: Bad Board Number

An illegal board number was given. Specify 0 or 1.

Error 7003: Bad Base Address

An illegal base address was specified. Valid base addresses range from 200 Hex to 3F8 Hex.

Error 7005: Bad Interrupt Level

An illegal Interrupt level is specified. Valid interrupt levels are 2,3,4,5,7,9,10,11, or 15.

Error 7006: Bad Number of EXP's

Specify 1 through 8 EXP's.

Error 7013: Bad Exp Number

Specify a number between 1 and 8.

Error 7015: Bad Number of EXP GP's

Specify a number between 1 and 8.

Error 7018: No Board Name

Specify a number between 0 and 7.

Core Codes

The following code numbers signify the errors you might encounter after making Core Driver Calls. No explanations appear where the error title is self-explanatory.

Error 00: No Error

No error.

Error 8001: Not Supported

Indicates that the specified function is not supported.

Error 8002: No Such Function

Indicates that the specified function is out of bounds.

Error 8003: Illegal Card Number

Indicates that the specified board number is not valid.

Error 8004: Bad Error

Indicates that the error number is not valid.

Error 8005: No Board

Indicates that the specified board is not at the configured address.

Error 8006: A/D Not Initialized

Indicates that the A/D Converter is not initialized.

Error 8008: Digital Input Not Initialized

Error 8009: Digital Output Not Initialized

Error 801A: Interrupts Active

Error 8020: Bad Revision

Specified DAS revision number is not valid.

Error 8021: Error Resource Busy

Illegal handle for frame.

Error 8022: Unknown Error Number

■ ■ ■