KEITHLEY

KDAC500 Data Acquisition and Control Software

Compiler Version for IBM, COMPAQ, and 100% Compatible Personal Computers

Publication Date: October 1991
Document Number: 501-915-01 Rev. C

Copyright Notice

Keithley Metrabyte/Asyst/DAC
Data Acquisition Division
440 Myles Standish Blvd.
Taunton, MA 02780
1-508-880-3000

Table of Contents

KEITHLEY

Upgrade Notes For KDAC500 Version 1.4

Changes in KDAC500 Version 1.4

The following changes have been made in KDAC500 Version 1.4. This information collectively covers interpreter and compiler versions of the software as noted.

Enhanced language support in KDAC500/M - The following Microsoft languages are supported by KDAC500/M:

Microsoft QuickBASIC 4.0	Microsoft Fortran 5.0
Microsoft QuickBASIC 4.5	Microsoft BASIC P.D.S. 7.0
Microsoft QuickPascal 1.0	Microsoft BASIC P.D.S. 7.1
Microsoft C 5.0, 5.1, 6.0	

Enhanced language support in KDAC500/B - The following Borland languages are now supported by KDAC500/B:

Turbo Pascal 5.0	Turbo C 1.0
Turbo Pascal 5.5	Turbo C 1.5
Turbo Pascal 6.0	Turbo C++ 1.0

Documentation changes - The contents of previous KDAC500/B and /M addenda have been integrated into the KDAC500 Compiler Manual. There is no additional documentation other than this manual.

NOTES ON TURBO PASCAL:

The Turbo Pascal Unit for Turbo Pascal 5.0 is named KDAC500.T50.
The Turbo Pascal Unit for Turbo Pascal 5.5 is named KDAC500.T55.
The Turbo Pascal Unit for Turbo Pascal 6.0 is named KDAC500.TPU.

In order for Turbo Pascal to find the KDAC500 TPU files they MUST have a .TPU extension. If you are using Turbo Pascal 5.x you should delete KDAC500.TPU from your installed version of the software and rename KDAC500.T5x to KDAC500.TPU. DO NOT delete KDAC500.TPU from the distribution disks! If you use the wrong version of TPU file the Turbo Pascal compiler will generate an error message and will not be able to compile your program.

The functional interface between all versions of Turbo Pascal is exactly the same. The format of the TPU file is different, however.

NOTES ON TURBO C++:

Turbo C++ 1.0 is supported by KDAC500/B. However, the KDAC500 interface library is a C style library and not a C++ library. In order to prevent Turbo C++ from "mangling" the KDAC500 function names you will need to include the following statements when including the KDAC500 header file:

```
#ifdef __cplusplus
extern "C" {
#endif
#include "kdac500.h"
#ifdef __cplusplus
}
#endif
```

Since Turbo C++ is also a C compiler, you can create C programs using KDAC500 with the same library file. The Turbo C++ library file for KDAC500 is named TURBO500.LIB. The file TURBO500.LIB is used for all versions of Turbo C as well as Turbo C++.

NOTES ON MICROSOFT BASIC PDS:

KDAC500 Version 1.4 is compatible with the Microsoft BASIC Professional Development System Version 7.0 and 7.1 for operation under DOS only. KDAC500 will not run under OS/2 or Microsoft Windows. Libraries are included on the KDAC500/M diskettes to facilitate running KDAC500 under the BASIC 7 QBX environment:

KDACQBX.LIB
KDACQBX.QLB

The file KDACQBX.LIB contains the object modules that make up the KDAC500 interface. The file KDACQBX.QLB is a quick library for the QuickBASIC Extended (QBX) environment. These files are NOT compatible with Microsoft QuickBASIC.

All BASIC 7.x modules that are compiled for use with KDAC500 must use FAR STRINGS. This means that they must be compiled using the /Fs option. According to the BASIC PDS Programmer's Guide, "If you are linking new code containing far strings with older code containing near strings, you must recompile the old code using the /Fs option. Otherwise the program will return an error message "LOW LEVEL INITIALIZATION and terminate."

NOTES ON MICROSOFT QuickPascal:

KDAC500 Version 1.4 is compatible with the Microsoft Quick Pascal Version 1.0. Every place that Turbo Pascal is referred to in the Kdac500/M manual also applies to QuickPascal. QuickPascal is functionally equivalent to Turbo Pascal. All the Language Syntax statements for Turbo Pascal also apply to QuickPascal.

The support unit for QuickPascal is named KDAC500.QPU. This file should either be in your working directory (it will automatically be copied there during installation) or in the QuickPascal UNIT directory.

CHANGES TO KDAC500/I

The only significant changes in KDAC500 Version 1.4 apply to Borland Turbo Pascal support for KDAC500/B, and Microsoft Professional BASIC support for KDAC500/M. The KDAC500/I Version 1.3 manual, Revision C, also covers KDAC500/I Version 1.4.

Changes in KDAC500 Version 1.3

The following changes have been made in KDAC500 Version 1.3. This information collectively covers interpreter and compiler versions of the software as noted. The contents of the addenda for KDAC500/I and KDAC500/B/M differ slightly.

SUPPORT FOR THE WAV1 MODULE (all versions) - KDAC500 V1.3 includes new commands for programming the WAV1 waveform generator module.

REVISED DOCUMENTATION -

Installation Section:

NEW INSTALL SECTION (all versions) - Previous addenda concerning the installation of KDAC500 have been incorporated into a revised installation section for KDAC500 V1.3. Replace your existing installation information with this new section.

"Methods..." Sections:

GENERAL NOTES, PLUS NEW INFORMATION FOR USING FORTRAN (KDAC500/B/M only) - The "Methods of Creating a KDAC500 Program" and "Methods of Running a KDAC500 Program" sections have been updated. Replace your existing pages with these new pages.

Command Section:

INTON (all versions) - Revised pages are included for the INTON command. Replace your existing INTON pages with the new pages.

KDCLOCK (KDAC500/B/M only) - A revised page for the KDCLCOK command is included for compiler versions of KDAC500. The page includes an example for KDAC500/M and QuickBASIC. Replace your existing KDCLOCK page with this new page.

WAV and WAVSETUP (all versions) - New pages are included for the WAV and WAVSETUP commands. Insert these pages at the end of the KDAC500 command section.

KDAC500/M EXAMPLE PROGRAMS NOW INCLUDED (KDAC500/B/M only) - An application note and examples covering programming with KDAC500/M and Microsoft QuickBASIC 4.5 have been added to the KDAC500/B/M (compiler version) manual. This information also discusses running KDAC500/M under the QuickBASIC environment. Add these pages to the end of the KDAC500 command section.

ENHANCEMENTS IN CONFIG.EXE (all versions) - The WAV1 now appears in CONFIG's list of available modules.

MICROSOFT BASIC 7 IS NOT SUPPORTED (KDAC500/B/M only) - None of the Microsoft Professional Development System BASIC Versions (6.0, 7.0, 7.1) are not supported for use with KDAC500/M. You can, however, use Microsoft Professional BASIC to perform low-level access to the Keithley hardware by PEEKing and POKEing the data acquisition system command registers. In this case, KDAC500 is not used.

TRIGGER/KDTIMER BUG FIXED (all versions) - A bug which prevented KDTIMER from waiting for a background trigger has been fixed.

ANINQ BUG FIXED (all versions) - For multi-channel acquisition, the ANINQ command sometimes returned incorrect data to the first width in the data array. This has been fixed.

PASSING PARAMETERS TO BASICA WORKS AS DOCUMENTED (KDAC500/I only) - KDAC500/I V1.3 now permits passing the BASIC interpreter start-up parameters such as number of files open, buffer size, etc. These modifiers are added to the batch command used to start KDAC500, e.g. "KDAC500 /F:5 /S:32".

Changes in KDAC500 Version 1.2

The following changes were made in KDAC500 Version 1.2. This information covers interpreter and compiler versions of the software as noted.

KDAC500/I EXAMPLE PROGRAMS NOW INCLUDED (KDAC500/I only) - The KDAC500/I distribution diskettes include several example programs which exercise most KDAC500/I commands. The programs are stored on both the 5 " KDAC500/I Library disk and the 3 " combined disk under the directory "EXAMPLES". The programs are also listed in the KDAC500/I manual (Rev. B or later) at the end of the command section.

SIMPLIFIED INSTALLATION (all versions) - The installation process now uses a single INSTALL.EXE file to copy files, test the system environment, and generate the necessary KDAC500 files. Previous KDAC500 versions used INSTALL.BAT and KINSTALL.EXE to perform these functions.

NOTE: If you choose NMI as the interrupt method, you must select and execute the NMI Interlock Test. On 286-based (AT) computers, the NMI Interlock Test will cycle the NMI while displaying a window showing test counts and remaining time. On 8088- and 80386-based computers, INSTALL will set the interrupt mask information, and then exit the test without exercising the interrupt test.

CHANGES IN THE INSTALL SCREEN (all versions) - The installation environment screen now shows the speed determined for the master IBIN interface, rather than the system bus or clock speed. This information is for reference, and will not affect how you perform the installation. If the installation environment screen shows a negative amount for system memory, you have insufficient memory to do an installation. 640k of system RAM is recommended. Abort INSTALL by pressing <Escape>. Check your DOS AUTOEXEC.BAT and CONFIG.SYS files for resident programs or other memory users, and eliminate as many as possible. Rerun INSTALL.

MORE ACCURATE ASSESSMENT OF PROCESSOR SPEED (all versions) - The new installation program more accurately determines the computer processor speed, particularly in faster 286 and 386-based machines. KDAC500 has been tested on 25MHz 386-based computers, and should be compatible with 33Mhz systems as well.

INTERNAL CHANGES IN KINSTALL.KIF AND CLKSPD.COM (all versions) - The installation ".KIF" file includes new parameters which record additional diagnostic information concerning system timing. The new installation is not compatible with .KIF files from earlier KDAC500 versions. If you have an earlier version of the software, do not install KDAC500 1.2 in the same directory or you may mix old and new files.

The structure of the CLKSPD.COM file in KDAC500 V1.2 differs from CLKSPD.COM in older versions. Information concerning the location of various operating parameters and variables within older CLKSPD.COM files is not valid for the V1.2 CLKSPD.COM.

ENHANCEMENTS IN CONFIG.EXE (all versions) - The CONFIG.EXE file has been re-compiled for greater speed. The handling of some modules has been improved. Minor problems in CONFIG have been corrected.

Additional modules appear in the hardware set-up screen.
The AMM1 module selection provides choices for AMM1 or AMM1A.
The AIM3 module selection provides choices for AIM3 and AIM3A.
The single module designation "DIO1" covers both the DIO1 and DIO1A.

IMPROVED HANDLING OF THE AIM3A IN CONFIG (all versions) - Previous versions of CONFIG permitted the AIM3A to be assigned a cold-junction reference IONAME even though the AIM3A has no CJR or channel 32. This could result in problems at run time. The V1.2 CONFIG program will no longer accept a reference junction IONAME for the AIM3A. You may use an older CONFIG.TBL file with KDAC500 V1.2. However, any IONAMEs which reference channel 32 on an AIM3A will be ignored.

A bug affecting KDAC500's recognition of IONAMEs for the AIM3A has been fixed.

CHANGES IN THE HANDLING OF THE TRG1 MODULE IN CONFIG (all versions) - Two changes have been made in how CONFIG handles the TRG1 module:

First, KDAC500 V1.2 treats the TRG1 module as an analog module. Previous KDAC500 versions treated the TRG1 as a digital module because of the TRG1's digital trigger output signal.

If you are using a TRG1 with KDAC500 V1.2, you must reinstall the module in CONFIG, and recreate the TRG1 IONAMEs. You may use an older CONFIG.TBL file with KDAC500 V1.2. However, any IONAMEs which reference a TRG1 will cause an error message when KDINIT is executed.

Second, the TRIG1 module can now be used as a single channel of differential analog input. Set up an IONAME which references the TRIG1 input channel, and then include the IONAME in an analog input command. The IONAME can include the following set-up parameters:

Local gain of x1 or x10. Default = x1.
Global gain of x1, x2, x5, or x10. Default = x1.
AC or DC coupling. Default = DC.
Filter of 300Hz - 1Mhz. Default = 1MHz.

ENHANCEMENTS TO INTON (all versions) - The INTON command now accepts Hertz (HZ) and millihertz (MILHZ) as time units. The legal ranges are 0-65,535 Hertz and 0-65,535 millihertz. The maximum achievable rate on a 386-based computer is approximately 6,000Hz; slower on 286 and XT type systems. Specifying too high an interrupt rate may lock up the computer.

IMPROVED ACCURACY FOR TEMPERATURE CONVERSIONS (all versions) - In KDAC500 V1.2, temperature EUF routines dynamically select temperature conversion constants based on thermocouple and reference junction output voltages.

FIX IN GRAPHIC OVERLAYS (all versions) - When two identical graphics commands were issued consecutively in a program, the second graphics command might not produce a graph. For example, if HGRAPHRT commands were called in two consecutive program lines, only the first HGRAPHRT would plot a graph. This problem has been corrected.

BACKING UP YOUR KDAC500 DISKETTES (all versions) - If you need to make back-up copies of the original KDAC500 diskettes, do so with the DOS DISKCOPY command. Using a simple DOS COPY or XCOPY command produces backups which will not function correctly when INSTALL.EXE is executed.

Updates and Errata

Please observe the following notes, errors, and updates pertaining to the KDAC500 manual.

Chapter 1, Getting Started:

1. * New Installation Section (all versions) - integrates all previous changes to the installation procedure into a new replacement installation section for the manual.

2. * Methods of Creating a KDAC500 Program (KDAC500/B/M only) - additional instructions have been included for programming with Microsoft FORTRAN. These are contained in a new replacement section for the manual.

3. * Methods of Running a KDAC500 Program (KDAC500/B/M only) - notes and changes have been included in a new replacement section for the manual.

Chapter 2, "Engineering Units Conversion" (KDAC500/I only):

1. The Engineering Unit Flag for LVDT/RVDT sensors under KDAC500/I is "C.AIM9.D" rather than "C.AIM9D" which is shown in the KDAC500/I manual.

Chapter 4, "KDAC500 Commands" (all versions):

1. ARLOAD - A dummy string (up to 255 characters) must be created for the array name prior to running ARLOAD. If the actual array name is shorter than the dummy string, the resulting array name will be padded by spaces, with length equal to that of the dummy string.

For example, if the dummy name is defined as ARN$ = SPACE$(12) and the actual name is "DATA%", ARN$ will equal "DATA%" plus 7 spaces ("DATA% ") after the ARLOAD.

2. ARGET/ARPUT - The ION$ parameter takes precedence over the WID% parameter. To identify a particular data set by its width parameter, set the ION$ parameter to "" (two adjacent double quotes).

3. ARSAVE - The legal time units for ARSAVE now include HZ and MILHZ. If the file is saved in one of the non-KDAC data formats using HZ or MILHZ as the time base, the time units column in the output file will be represented as fractions of a second.

4. ARSTATUS - A dummy string (up to 255 characters) must be created for the array label prior to running ARLOAD. If the actual array label is shorter than the associated dummy string, the resulting array label will be padded by spaces, with length equal to that of the dummy string. See ARLOAD command information listed above for similar requirements concerning the array name.

5. INTON - The first data point acquired after INTON is issued will be taken after the specified interrupt period has elapsed.

6. KDCLOCK - Data returned for the year consists of four digits, e.g. "1990" rather than "90".

7. KDINIT - a DIO1A module and the digital section of a Model 575 will be initialized as follows when KDINIT is called: Ports A and B to logic 0; ports C and D to logic 1.

8. BGREAD and ARGET - If you intend to apply engineering units conversion, you should not mix IONAMEs for different types of modules (e.g. AIM7 and AIM8) in the same BGREAD command. This will cause the data from both types of modules to be stored in one KDAC500 array. When you issue an ARGET command with engineering units conversion, an error will be generated. Separate BGREAD commands should be used. If you plan to work exclusively in A/D counts or voltage, this limitation does not apply.

9. TRIGGER - The CHM parameter in the KDAC500 manual shows the values "OFF" and "ON" for digital input. For interpreter BASIC and QuickBASIC only, these should be "OFF." and "ON."

10. KDCLOCK - The last line of the TIME%() parameter shows "Time%(50 year....". This should read "Time%(5) year...".

11. FGREAD - The RANGE$ parameter shows "P2.READ_RESET" and "P2.READ_ONLY" for pulse reading modes with PIM1 and PIM2. Correct syntax for BASICA and QuickBASIC is "P2.READ.RESET" and "P2.READ.ONLY".

Appendix A, "Installation of KDAC500/I on a Dual 360K System" (KDAC500/I only):

1. Under Step 3, Item 3 - KINSTALL.EXE is now INSTALL.EXE.

2. Under Step 4, Item 2 - KINSTALL.EXE is now INSTALL.EXE.

Introduction

Welcome to the world of workstation data acquisition and control with Keithley's KDAC500, and thank you for selecting a Keithley product.

WHAT IS KDAC500?

KDAC500 is a family of software products for data acquisition and control using Keithley's 500A, 500P, 575 and 570 Measurement and Control Systems and IBM, Compaq, or 100% compatible personal computers. The KDAC500 family consists of the following individual packages:

KDAC500/I — an interpreter-based version which runs under Microsoft GWBASIC, IBM Advanced BASIC (BASICA) and Compaq Advanced BASIC. KDAC500/I is bundled with certain Keithley hardware.

KDAC500/B — a compiler version of KDAC500 which runs under Borland's Turbo C and Turbo Pascal. KDAC500/B is available as an option.

KDAC500/M — a compiler version of KDAC500 which runs under Microsoft's C, QuickC, Quick Pascal, QuickBASIC, and FORTRAN. KDAC500/M is available as an option.

Each package adds a number of new commands to its respective compatible language. Since KDAC500/I runs under BASICA, it provides an easy, entry-level approach to data acquisition programs under a language that is well-known and widely used. KDAC500/B and KDAC500/M provide an easy migration path for users who wish to upgrade from BASICA to faster, more structured compiler languages.

USING THIS MANUAL

In this manual you will find a complete description of the KDAC500 software package. This manual does not duplicate information in the computer hardware documentation or programming manuals except where necessary to explain specific features of KDAC500. This manual will not teach you how to program. If you need more help with programming, consult your local bookstore. There are numerous books available on programming languages and personal computers.

Many users of KDAC500/B and KDAC500/M will have had experience programming in Soft500, Quick500, or KDAC500/I. While these products are similar to KDAC500, there are important differences. This manual will make occasional reference to older software packages in order to call out some of those differences, and, hopefully, eliminate points of confusion for the user who is converting to KDAC500/B or KDAC500/M. If this is your first experience in programming for Keithley's data acquisition systems, these comments will not concern you.

The first three chapters of this manual provide a quick introduction to the KDAC500 software, and cover topics such as installation, creating the hardware configuration table, running KDAC500, etc. Chapter 4 includes a description of KDAC500 commands and an explanation of how each command is used. The commands are referenced alphabetically, and a list of all KDAC500 commands is included in the Appendix section. We recommend that you read your compiler manual and the Keithley KDAC500 manual before you begin installation and programming. Later, you can return to specific sections of the appropriate manual for more careful study. After a while, most users of KDAC500 will only need to use the command reference section for finding the command, format, and the parameters to use with KDAC500 commands.

Some Typographical Conventions

1. In this manual, all KDAC500 command parameters and reserved words are given in upper case characters. Quoted strings, remarks and data statements will remain in the form input by the programmer.
2. Any items enclosed by square brackets [...] are optional. However, none of the KDAC500 commands contain options within their parameter lists.
3. The characters "<" and ">" delimit keystroke combinations, or file names which you must include as part of a keyboard entry. The "<" and ">" marks themselves must not be entered. Examples include:

   <Ctrl-C>

   <Enter>

   <Ctrl-Alt-Del>

   <filename.ext>

   <configuration table file name>

4. Any other punctuation that appears in the KDAC500 command format line must be entered as shown. However, there is some flexibility in the delimiters used to separate parameters in a KDAC500 list. Valid delimiters are commas, spaces and tabs.

CHAPTER 1
Getting Started

What You Will Need to Run KDAC500

You will require the following items to install and run KDAC500:

DOS

You should have a DOS version which is approved by the manufacturer for use with your computer. Keithley recommends IBM PC-DOS 3.1 or later for IBM computers, Compaq DOS 3.0 or later for Compaq computers, and a manufacturer-approved version of MS-DOS 3.0 or later for compatibles. All versions of MS-DOS are not alike. A version which is optimized for use on one compatible may not operate properly on another. Versions of DOS earlier than the recommended versions may cause problems and should not be used.

COMPUTER

The KDAC500 Software System was developed to run on IBM PC, XT, AT, and PS/2 computers, Compaq computers, and most compatibles. The INSTALL program used with KDAC500 performs an in-depth analysis of the computer, DOS, RAM workspace, and other hardware parameters. INSTALL then creates batch files named KLOAD.BAT and KRUN.BAT, and a CLKSPD.COM file customized for the hardware environment. If INSTALL cannot accommodate the hardware, it will abort the installation process and issue an error message. If this happens, contact the Keithley DAC applications department for assistance.

Your computer should have 640K of random access memory (RAM). A smaller RAM space will detract from the amount of data that can be acquired. Since you will also be running a compiler, make sure that your RAM space is also sufficient for that purpose. KDAC500 will not run on a system which has less than 512K of RAM. KDAC500 will not take advantage of LIM/EMS expanded memory or extended memory.

Your computer should have a fixed disk KDAC500 Compiler versions must be installed with, and have ready access to, the compiler you've chosen. The total storage space for a typical installation of KDAC500 plus compiler may be three megabytes or more. It is not realistic to operate these packages from floppy-based systems.

To take advantage of the graphics capabilities of KDAC500, your computer should have a suitable graphics adapter. This adapter may be an IBM Color Graphics Adapter, Hercules Color Card, IBM Enhanced Graphics Adapter, IBM Video Graphics Array, or 100% equivalent. If the adapter is non-IBM, it must be 100% compatible with its IBM counterpart. Compaq portable computers and some Compaq desktop machines have a built-in combined text-and-graphics adapter. This is also suitable for displaying KDAC500 graphics.

Some standard color graphics adapters have an NTSC composite color output which will display graphics (without color) on a composite input monochrome monitor. These adapter/monitor combinations are also suitable, although some adapters may translate certain color combinations into invisible or unreadable monochrome shades. Monochrome adapters can be used with KDAC500, although they will not support graphics. This includes Hercules (monochrome) Graphics cards and similar monochrome graphics cards.

Graphics adapters which require special driver programs, or which make non-standard use of the computer's non-maskable interrupt (NMI) will cause problems when KDAC500 is run.

A math coprocesser chip for the computer is optional. It will improve the speed of execution of some KDAC500 commands, notably the commands which manipulate KDAC500 arrays and do engineering units conversions. Compilers often take advantage of coprocessors, and will deliver various performance improvements.

Installing KDAC500

Follow these instructions to install the KDAC500 software on your computer system. These steps cover all versions of KDAC500, including the interpreter and compiler versions. The references to BASICA or GWBASIC concern installation of the interpreter version only, and are noted as such where they appear. Reference to compilers concern only the /B and /M versions.

Install the Keithley Data Acquisition Interface Card

Installation of the KDAC500 software is a dynamic process which actively checks various parameters within the computer. If you have not done so already, open your computer and install your IBIN-A, IBIN-PS/2, or System 570 interface card. Be sure to note all CAUTIONS and WARNINGS for relevant safety information. If necessary, get a technician to assist you.

⚠ **WARNING: UNPLUG ALL POWER CORDS TO THE COMPUTER AND DATA ACQUISITION HARDWARE BEFORE YOU ATTEMPT INSTALLATION.**

KDAC500 is normally shipped as two 5-1/4", 360K diskettes and one 3-1/2" 720K floppy diskette. One of the 5-1/4" diskettes is labeled "Program Disk" and the other is labeled "Library Disk". Both volumes are combined on the single 3-1/2" disk. The KDAC500 diskettes and software manual contain all the support needed to make full use of your Keithley data acquisition hardware.

The following instructions presume the use of 5-1/4" KDAC500 distribution diskettes unless otherwise noted. References to 720K or 1.4M diskettes as drives A: or B: presume that your computer's standard A: or B: drives are 3-1/2" units.

You should not use the original KDAC500 diskettes for day-to-day operations. Install KDAC500 on to another floppy diskette or hard disk as described below. If you anticipate doing several installations, make and use a backup copy of the originals. In any case, store the originals in a safe place.

You should use the DOS version recommended for your computer, but no version before DOS 3.1 for IBM systems, DOS 3.0 for Compaq, or DOS 3.0 for compatibles running Microsoft DOS.

The Installation Files

Installing KDAC500 involves running "INSTALL.EXE". INSTALL.EXE may use other support files on the KDAC500 diskette set. The INSTALL.EXE file copies the necessary files to the destination disk, then configures the software.

Pre-installation

Before you begin the actual installation of KDAC500, make sure that any other computer configuration or memory-resident software has been installed and is operating properly. KDAC500 can coexist with many other types of programs, but compatibility problems may occur with some programs.

When you install KDAC500, you will specify an amount of system memory to be used for data. This memory will be allocated when you run KDAC500. If you add memory resident software to your operating environment after you have installed KDAC500, you may have insufficient memory left to run KDAC500 with the designated array space. Keep track of memory usage if you add memory-resident programs to your system, and reinstall KDAC500 if necessary.

Turn on your computer and load DOS.

Installations on Computers with Multiple Processing Speeds

Some computers have switchable CPU clock speeds. If your computer has a switchable CPU clock speed, perform data acquisition only at the CPU clock speed which was set at the time of installation.

Advanced BASIC (KDAC/I Only)

During installation of KDAC500/I, you will be prompted for the name and location of your Advanced or GW-BASIC interpreter file. At that time, enter the disk drive letter, path (if any), and complete filename of the BASIC interpreter. During installation, BASIC must be where you specify or you will receive an error message.

Approved Compilers (KDAC/M and /B Only)

The KDAC500/B and KDAC500/M packages are designed to run under any of several approved compilers. The "/B" package is designed for Borland Turbo C and Turbo Pascal, while the "/M" package is intended for Microsoft C, QuickC, QuickBASIC, Quick Pascal, and FORTRAN. The supported revisions include:

KDAC500/B

Borland Turbo C	Version 2.0
Borland Turbo Pascal	Versions 5.0 and 5.5

KDAC500/M

Microsoft C:	Versions 5.0 and 5.1
Microsoft QuickC:	Versions 1.0 and 2.0

Microsoft QuickBASIC: Versions 4.0 and 4.5
Microsoft QuickPascal: Version 1.0
Microsoft FORTRAN: Version 5.0

Earlier versions of these products are not supported and should not be used. Borland Turbo BASIC is also not supported.

Support for other compilers will be added from time to time. If you are considering a compiler or version other than those listed, please call Keithley DAC for more information. The KDAC500 packages do not include the compiler. You are free to purchase an approved compiler package for the language you desire.

Note that the installation for each of the various compilers may involve specialized path parameters, environment variables, library directories, or other special configurations needed by the compiler. Make sure that your compiler is installed and operating properly before you attempt to run KDAC500. Ideally, your system path, directories, and environment variables should be organized such that you can run your compiler while working in the KDAC500 directory. See your compiler manual for more information on installing the compiler.

Disk Preparation

Most personal computers share one of a relatively few disk drive configurations. The drive configuration will dictate which drive receives the installation, and which drive serves as the source.

Installing KDAC500 requires two disk drives. The drive receiving the installation should be a fixed disk, or a 720K, 1.2M, or 1.4M floppy drive. The diskette drive holding the KDAC500 product diskette may be any type.

Depending on the compiler, KDAC500 may be run from a fixed disk or a high-capacity floppy disk drive such as a 720K, 1.2M, or 1.4M drive.

Running KDAC500/I from two 360K floppy diskette is not recommended. However, an installation for dual 360K floppies is outlined in the Appendix section of the KDAC500/I manual. A single 360K floppy does not have enough room for a working copy of KDAC500.

For KDAC500/B or /M, note that some compilers require a fixed disk because floppy disks do not have enough room

Installing to a High-Capacity Floppy Diskette (KDAC500/I, only)

These instructions assume that you have a 1.2M or 1.4M diskette drive as A: and any type of drive as B:. If your high capacity diskette drive is Drive B: rather than A:, switch the A: and B: designations in the following procedure.

These instructions also assume that you are installing KDAC500/I. Compiler versions plus the required compiler will not fit on one floppy disk.

The HC diskette which receives KDAC500 must be formatted and should also be bootable. KDAC500 may also be installed to a non-bootable floppy diskette. Doing so will save the space on the diskette normally taken by the DOS system files at the expense of operating convenience. To format without copying the system files, omit the "/S" option.

To format a HC floppy diskette, place the blank HC diskette in the HC drive A: and your DOS diskette in drive B:. Make B: the default drive and run FORMAT:

B: <Enter>
FORMAT A: /S <Enter>

Unless told otherwise, FORMAT assumes that a diskette formatted in a HC drive is a high-capacity diskette. The system will format the diskette in drive A: and copy over the DOS system files.

Install KDAC500/I in its own directory on the HC diskette. When you run INSTALL, the desired target directory will be created automatically if it does not already exist.

Replace your DOS diskette in Drive B: with the KDAC500/I Program Disk and continue with the section "The INSTALL Environment".

Installing KDAC500 on a Fixed Disk

Normally, the bootable fixed disk on a personal computer is drive C:. If you have two fixed disks, or if your disk is divided into "logical" drives, you may also have drives D:, E:, F:, etc. KDAC500 may be installed on any drive.

Fixed disk installation is recommended for all versions of KDAC, and provides maximum convenience.

You should install KDAC500 in a separate directory on the fixed disk. When you run INSTALL, the desired target directory will be created automatically if it does not already exist.

Place the KDAC500 Program Disk in Drive A: and continue with the section "The INSTALL Environment".

The INSTALL Environment

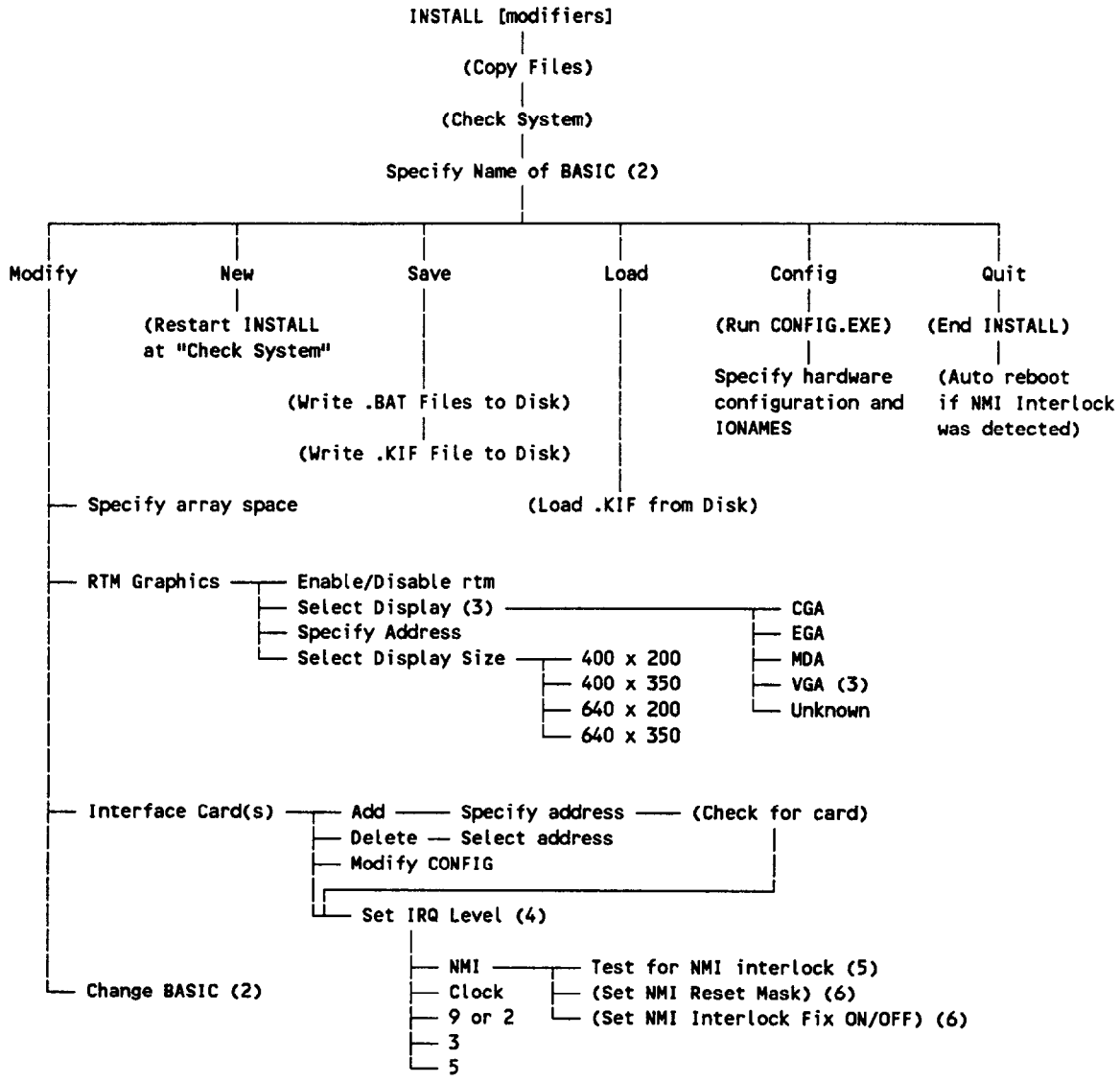
The INSTALL program initially checks the computer hardware and then generates an information screen with pull-down menus (see Figure 1-1). The INSTALL environment will

prompt you for information. Some responses will require that you enter alpha or numeric characters, while other responses will be simple menu selections. In both cases, the available entries will be obvious or self explanatory. Figure 1-2 is a map of the installation process. Interpretive BASIC will appear only for KDAC500/I.

```
Modify  New  Save  Load  Config  Quit
Allows user to modify displayed parameters.
```

```
Keithley DAC KDAC500 Installation

  Array Space / Maximum Size:      64K / 182K
  Master IBIN Timer Speed:         1.00 MHz
  Machine Type:   IBM AT or compatible

  Processor Type: 80286
  RTMDS Graphics: Disabled

  Interface Board(s):
  IBIN Address      CONFIG File Name
     cff8H     -*-      CONFIG          (int lev = NMI)
                                        (nmi mask = 08)
                                        (NMI interlock fix OFF)

  KDAC500 Working Directory:
  C:\KEITHLEY\
  Interpretive BASIC:
  C:\DOS\BASICA.COM
```

Figure 1-1. Main Installation Screen



NOTES:

1. Processes in parentheses ( ) are performed by the installation software.
2. KDAC500/I interpreter version, only. BASIC interpreter name may be respecified through the "Change BASIC" option.
3. Installation automatically determines type of primary display, and will show "UNKNOWN" if display type is not recognized. VGA is not compatible with RTM card.
4. IRQ is set only for the first interface card. IRQ9 on an AT = IRQ2 on a PC/XT.
5. Must be performed by user if NMI has been selected. See manual for details.
6. May also be modified manually.

Figure 1-2. Main Install Map

Running Install

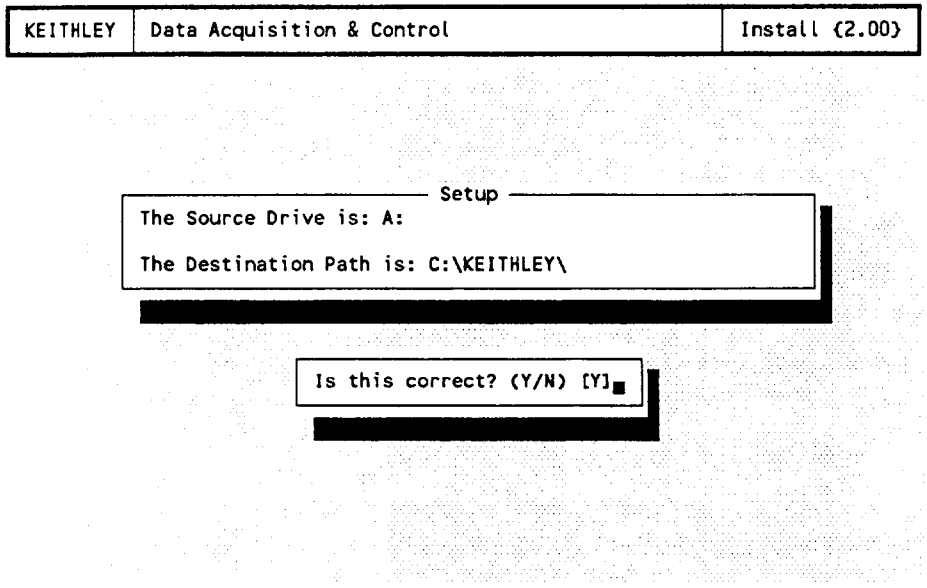
The simplest method for running INSTALL is to make the floppy diskette holding the KDAC500 distribution diskette the default, and then execute "INSTALL" at the DOS command line.

If you are using the 5-1/4" diskettes, insert the diskette marked "Program" into the disk drive. Later, INSTALL will prompt you to change to the "Library" diskette. The 3-1/2" diskette contains all the KDAC500 Program and Library files, and will not require a disk change.

The floppy disk volumne will typically be A: or B:. Change to the appropriate drive and run INSTALL:

A: <Enter>
INSTALL <Enter>

The INSTALL program will respond with the following screen. If the source and destination are correct, press <Enter>. If not, chang "Y" to "N" and press <Enter>. You will then have an opportunity to change the source and destination.



KDAC500 Installation Screen

When you accept the indicated setup, the INSTALL program will copy the files from the KDAC500 diskette(s) to the target, and then run the remainder of the installation program. Proceed to the next section "Modifying the Installation Parameters".

You may specify the source and destination for the installation as part of the INSTALL command line. If you elect to use this method, you should specify source, destination, and directory on the INSTALL command line. The following optional syntax may be used:

NOTE: You must leave a space between all items entered in the INSTALL command line or the installation will not be completed properly.

Make the drive holding the source diskette the default drive.

A: <Enter>

Enter the INSTALL command. The optional format is:

INSTALL <Source Drive> [Destination Drive [Destination Path]]

where:

<Source Drive> is the drive containing the KDAC500 diskette (A:, B:, etc.). Source Drive is required.

[Destination Drive] is the drive to which KDAC500 will be installed (C:, D:, etc.) Destination Drive is optional. If it is omitted, "C:" will be assumed.

[Destination Path] is the path to KDAC500. The path name must begin with the "\" backslash. Destination Path is optional. If it is omitted, KDAC500 will be installed to directory KEITHLEY off the root directory of the destination drive.

If you want to install to a root directory , use "\" as the destination path. This is not recommended for fixed disks, but may be desirable for a floppy disk installation.

If this is a floppy disk installation with the destination in drive A: and the source in Drive B:, you can start the installation with the command:

INSTALL B: A:

If this is an installation to the root directory of a floppy disk in drive B: and the source is Drive A:, start the installation with the command:

INSTALL A: B: \

Entering the Name and Path of Interpreter BASIC (KDAC/I, only)

For an installation of KDAC500/I, INSTALL will ask you for the name and location of your BASIC interpreter. Typical names are BASICA.COM, BASICA.EXE, GWBASIC.EXE and GW-

BASIC.EXE. Specify the drive letter, complete path, and complete BASIC filename, then press <Enter>. The installation will check your version of BASIC, during which the computer screen will briefly go blank.

Modifying the Installation Parameters

SELECT "MODIFY" FROM THE PULL-DOWN MENU: After you have specified the name and location of your BASIC interpreter, you will see the screen in Figure 2-1. If the menu is not visible, it can be invoked by pressing the <Esc> key.

Once visible, any menu choice can be invoked by using the cursor keys to move the cursor to the desired choice, and then pressing <Enter>. You may also press the first letter of the menu word.

Select MODIFY — The cursor will drop down into the information screen and let you change selected installation parameters. You may change any or all of the following items.

ARRAY SPACE — The Array Space line shows the following information:

Array Space / Maximum Size

"Array Space" shows what you have specified for the desired array workspace. "Maximum RAM" is a suggested maximum amount of RAM available for data arrays.

NOTE: For KDAC500/I, the suggested maximum is based on the sizes of IBM PC-DOS 3.3, GWBASIC 3.2, and the KDAC500 software. Your DOS and BASIC versions may require slight adjustment of the suggested maximum. Under some circumstances, you may enter more than the suggested RAM size.

NOTE: If you specify too large an amount of RAM, you will receive an error message when you run KDAC500, or run into other problems. For a first installation, it is best to select a small array memory size, e.g. 64k for KDAC500/I, or 8-16k for a compiler version. After you become familiar with the software, you can change the memory size.

Select ARRAY SPACE and press <Enter>. A box will appear on the screen. Type the size of the array space you want and press <Enter>.

RTM GRAPHICS — If you have an RTM (Dataq WFS-200) waveform scroller card installed in your computer and would like to use it for KDAC500 graphics, select RTM GRAPHICS and press <Enter>.

ENABLE/DISABLE — to enable use of the RTM card, move the cursor to this choice and press <Enter>. The feature will toggle on or off. RTM cannot be enabled if the detected monitor adapter is VGA.

SET DISPLAY TYPE — move the cursor to this option and Press <Enter>. Select the type of monitor you will be using with the RTM (Dataq) WFS-200 waveform scroller card. VGA video is not compatible with the WFS-200.

ADDRESS — select this option and press <Enter>. Type the hardware address of the RTM card as installed in your system (see the RTM manual for more information)

DISPLAY SIZE — select this option and press <Enter>. Move the cursor to the desired display size and press <Enter>. The only valid choice if you are using a CGA card with the RTM card is 640x200. Note messages on the INSTALL screen for more information.

INTERFACE CARD(S) —You must specify the address of each Keithley hardware interface board installed in your computer. If you do not specify a hardware interface, the software will not be able to access the data acquisition system connected to that interface card. Select INTERFACE CARD(S) and press <Enter>.

NOTE: If you are installing to a PS/2 with Microchannel bus (PS/2 Model 50, 60, 70, and 80), the only parameter you may change under INTERFACE CARD(S) is "MODIFY CONFIG". The interrupt is set as part of the hardware installation using the PS/2 Reference Diskette.

ADD — Selecting "INTERFACE CARD(S)" when there are no cards shown will automatically move to "ADD" and prompt you for an address. For additional interfaces, select ADD and press <Enter>.

You will be prompted for the first two characters (in hexadecimal) of the board's address. For example, if a board is installed at address CFF80 (hex), you would enter "CF".

After you add the first card, the menu moves directly to the "SET INTERRUPT LEVEL" option.

Repeat the ADD step for each interface installed in your computer.

DELETE — to remove a board from the configuration, select DELETE and press <Enter>. Use the cursor keys to select the board for deletion and press <Enter>. Repeat this step for each interface you want to remove from the configuration process.

MODIFY CONFIG — The default name of the configuration table file for the first interface you identify is "CONFIG.TBL". Additional boards will be named "CONFIG1.TBL", "CONFIG2.TBL", etc. To specify a different name, select MODIFY CONFIG and press <Enter>. Use the cursor keys to select the name to be changed and press <Enter>. Type the desired name and press <Enter>.

SET INTERRUPT LEVEL — This option selects a hardware interrupt within the PC which will be used to control background (timed) data acquisition. The choices, in order of system priority, are "NMI", "CLOCK", 9, 3, and 5. NMI is the computer's non-maskable (highest priority) interrupt. CLOCK is the computer system clock. If you are using an 8088-based PC or XT, Level 9 corresponds to IRQ2 in the PC.

NMI is the suggested method, but may not be compatible with all types of PC hardware. Specifically, the NMI is used by some video adapters and by 80386 systems operating in virtual 8086 mode. These systems will conflict with KDAC500 if it also attempts to use the NMI.

NOTE: For non-Microchannel computers, you must select an interrupt method as part of the installation. If you do not select an interrupt system, operating errors will result when you run KDAC500.

Move the cursor over the desired choice and press <Enter>. If you select "NMI", you will receive a submenu with the following choices:

TEST FOR NMI INTERLOCK — This choice will check the computer microprocessor for compatibility with use of NMI and Keithley interface, and then install the NMI as the interrupt system for KDAC500.

SET NMI RESET MASK — The previous option "TEST FOR NMI INTERLOCK" will normally set this parameter automatically if it is required. This option should not normally be selected unless an installation problem requires you to manually adjust an installation's parameters. The NMI RESET MASK sets up an installation parameter which helps determine the software's use of the computer's non-maskable interrupt.

NMI INTERLOCK FIX ON/OFF — The option "TEST FOR NMI INTERLOCK" will normally set this parameter automatically if it is required. Some early 80286 microprocessors have an internal problem which causes background data acquisition to spontaneously abort. NMI FIX will correct the problem through software. NMI fix also disables the system clock, so BASIC commands such as "TIMER" will not work when the NMI fix is in place.

CAUTION: You should not change the NMI RESET MASK or NMI FIX unless you have experienced trouble with the installation, and have been instructed by Keithley DAC technical support to do so. The following chart is for reference only.

NOTE: If the NMI interlock is detected on your computer, you may also choose CLOCK or Level 9, 3, or 5.

COMPUTER	MASK	NMI FIX
Later AT *(80286 CPU)	08	OFF
Later AT&T 6300 Plus	10	OFF
AT&T 6300, PC, XT (8088 CPU)	20	OFF
Older AT (80286 CPU)*	08	ON
Older AT&T 6300 Plus *	10	ON

* Exhibits NMI Interlock, a defect in some early 80286 processor chips can cause the computer system to spontaneously lose interrupts when executing background programs. If this occurs, contact Keithley DAC technical support.

RUN "CONFIG" FROM THE PULL-DOWN MENU: The "CONFIG" option will run the CONFIG.EXE program which enables you to set up a hardware configuration table for each system connected to an interface in your computer. The CONFIG option defaults to the first interface in the INTERFACE CARD(S) list. To set up configuration tables for other interfaces, select the proper modules and other information, and save the table under the name "CONFIG1", "CONFIG2", etc. See the section of this manual covering CONFIG.EXE for more information of running CONFIG.

SAVE YOUR INSTALLATION: The "SAVE" option will create one or more of the following files: KDAC500.BAT, KINSTALL.KIF, KLOAD.BAT, and KRUN.BAT. KDAC500.BAT is used to start KDAC500/I in non-resident mode. KINSTALL.KIF is a record of the all the information which has been entered in the installation screen.

KLOAD.EXE is used to make the KDAC500 kernel memory resident. KRUN.BAT is used to run compiled programs generated under compiler versions of KDAC500, and does not apply to KDAC500/I.

SELECT "QUIT" FROM THE PULL-DOWN MENU: The "QUIT" option will close the installation screen. If the NMI interlock problem was detected, the computer will automatically reboot when you quit.

Other MENU Options

NEW — This option will reset all parameters on the installation screen. The current data shown on the screen will be lost. This does not affect the information stored to disk unless you execute SAVE.

LOAD — This option loads the last KINSTALL.KIF file saved to disk. It enables you to make minor changes to an existing installation without having to re-enter all the information.

For Future Installations

Once KDAC500 has been installed, you can change setup parameters by re-running INSTALL.EXE on the destination, or by running the INSTALL program directly.

The syntax for the INSTALL command is:

INSTALL [–f] [–mono] [–rtm<display_area><wfs–200_address>]

Any or all the modifiers may be used.

Details on the INSTALL command line modifiers are as follows:

–f — The "–f" modifier tells KINSTALL to automatically load and use the set-up information in the KINSTALL.KIF file, rather than rechecking the system. This option will permit you to modify parts of an installation without having to re-enter all the information..

–mono — The "–mono" modifier selects monochrome output. Normally, the program is able to sense what type of display adaptor the computer has and adjust the display accordingly. Where a composite monochrome monitor is being driven by a CGA, the color pallet is translated into various shades and patterns which may be unreadable under some circumstances. In these cases the –mono modifier overrides the sensing of a display adaptor and improves the readability of the screen.

–RTM — The "–RTM" modifier can be used to specify all the set up information for the RTM (Dataq WFS-200) waveform scroller card. The full syntax for the RTM modifier is:

–rtm<display_area><wfs–200_address>

There are four possible values for the display area:

LL = 400 pixels horizontal x 200 pixels vertical
HL = 640 pixels horizontal x 200 pixels vertical
LH = 400 pixels horizontal x 350 pixels vertical
HH = 640 pixels horizontal x 350 pixels vertical

When operating on a CGA the only valid display area specification is HL. When no display area is specified a default of HH (for EGA or MONO) is used.

The scroller card address is a number from 0 to 3f8 (hex) which must be evenly divisible by 8. The default address is 308.

NOTE: in order to specify an alternate address, a display area must also be specified.

The KINSTALL.KIF File

When you write an installation to disk with SAVE, a file named "KINSTALL.KIF" is also written to disk. This file is a record of the information entered in the installation screen. This information is valuable when you contact the applications department for help.

Installations Involving the 500GPIB Module

If your system includes a 500GPIB module, note that KDAC500 does not perform installation steps for this module. The 500GPIB driver installation is outlined in the 500GPIB manual. It requires that the user manually edit the DOS AUTOEXEC.BAT file to insert the required command line.

If You Have Problems Installing KDAC500

If you experience any problems with INSTALL, first review your work. If you must call Keithley DAC for assistance, have available a printout of your KDAC500.BAT file (KDAC500/I only) and KINSTALL.KIF (if they exist).

You should also have available the make and model of your computer and type of monitor adapter. Also print out copies of your AUTOEXEC.BAT file and CONFIG.SYS file. If possible, be near your computer so that you can supply additional information if it is requested.

Running the CONFIG Program

This section of the KDAC500 manual covers the creation of hardware configuration table "CONFIG.TBL" with the KDAC500 program "CONFIG.EXE". The following paragraphs discuss the purpose of the "CONFIG.TBL" file, explain the important run-time features of CONFIG, and provide a tutorial session using CONFIG.

The Function of CONFIG.TBL

Keithley data acquisition and control systems are modular and expandable. The possible combinations of modules, ranges, gains, and I/O configurations are practically unlimited. KDAC500 must know the module names, switch settings, and related information in order to control the hardware and acquire data. During installation, the INSTALL program can create up to four configuration table filenames -- one for each interface card located in the computer system. These names are CONFIG.TBL, CONFIG1.TBL,...CONFIG4.TBL. You must run CONFIG to create a configuration table file for each of the file names.

Setting Up Channel Information (IONAMEs)

The CONFIG.TBL file must also include the channel information for each channel accessed by KDAC500. This information establishes a name for a channel, as well as the slot, channel number, gain, and other parameters needed to fully describe the channel for input or output, hence the term "IONAME". KDAC500 itself does not provide for an IONAME command to be used within the programming environment.

An important function of IONAMEs in the CONFIG.TBL file is that they set programmable functions in many modules which otherwise lack hardware switches. An important example is the programmable local gain on the AMM1A and AMM2 modules. In this case, IONAME enables you to set the both local gain and global gain for any channel input of the AMM module.

Menu Screens, Cursor Control, and Special Function Keys

All the functions of CONFIG are performed menu-style using tabular screens and the computer's cursor and special function keys. The screens are "HARDWARE SETUP" and "CHANNEL SETUP" (see figures 1–3 and figure 1–4). Each screen contains several columns or windows which show instructions and menu choices for configuration.

Selections are made with various special function keys F1-F10. Each key controls one or more functions depending on which screen or window is active. Some keys are not used.

The cursor appears as flashing text in reverse video block. Cursor keys on the right side of the keyboard control the movement of the cursor around the screen. The left and right arrow

keys function only where choices are arranged in two or more columns. The up arrow and down arrow keys control vertical movement, and will automatically move the cursor through multiple columns.

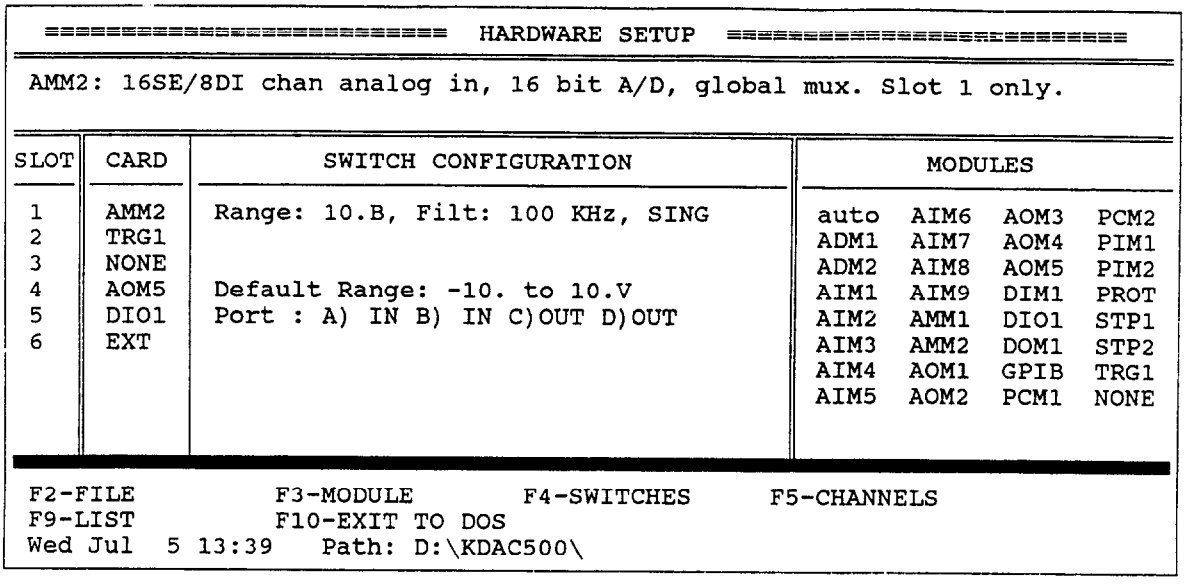


Figure 1-3 Hardware Setup

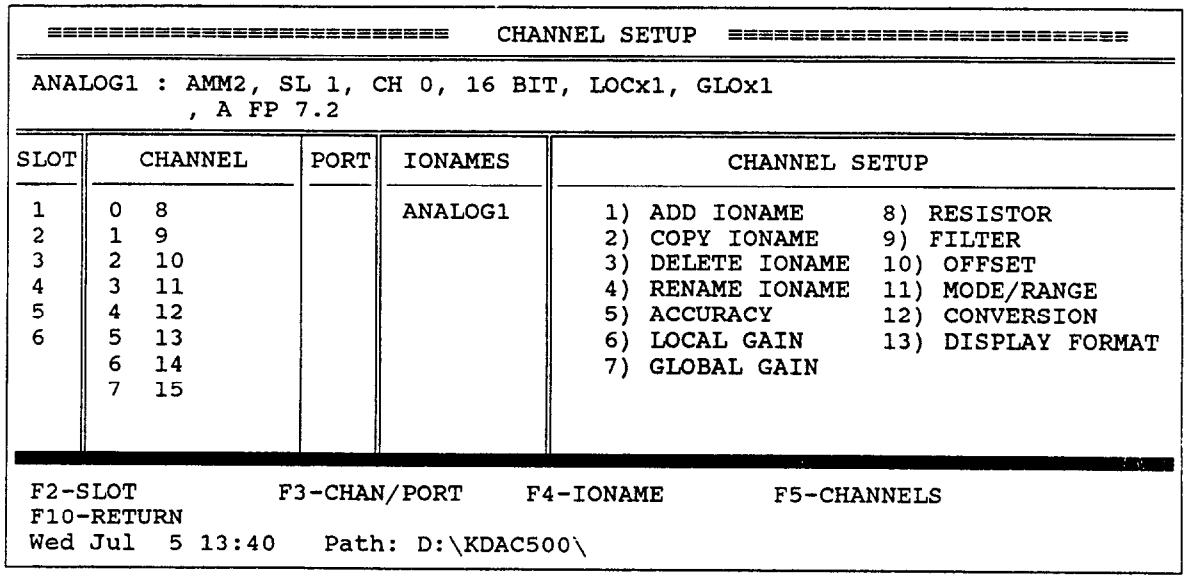


Figure 1-4. Channel Setup

Special Considerations for the Model 575 and System 570

The CONFIG program contains an option which automatically creates a configuration table for the Model 575-1, Model 575-2, or System 570. This configuration table contains the factory default module assignments, ranges, gains, and other setup information for these systems.

To automatically create a Model 575-1 configuration table, type:

CONFIG 575-1

To automatically create a Model 575-2 configuration table, type:

CONFIG 575-2

To automatically create a System 570 configuration table, type:

CONFIG 570

Press <Enter>. CONFIG loads a default configuration table. Refer to the specific hardware manual for more details on setting up the system hardware.

The modules in the Model 575 and System 570 cannot be changed, except for the option slot. However, you can change hardware switches (if present), and other set-up information for the virtual slots. If you add an option module or change the hardware configuration, run CONFIG and update the configuration table to show these changes.

The CONFIG commands shown above should be run only once to create the desired configuration table. Make subsequent changes to the table by running CONFIG. Then LOAD CONFIG.TBL, or the appropriate configuration table.

Running A Sample Configuration Session

The following instructions are a step-by-step tutorial through a typical configuration of AMM2, AIM3, and DIO1 modules installed in a 500A System. This exercise does not cover all possible menu selections, and may not cover the particular hardware installed in your system. However, it will provide enough familiarity with CONFIG that you can create any configuration table that you will need.

This tutorial has five objectives:

1. Set up an AMM2 module in slot 1 with single-ended input, and ±10V A/D range.
2. Set up an AIM3 module in slot 6 with 16 differential inputs and a hardware gain of x100.
3. Set up a DIO1 in slot 8 with channels 0-15 configured as outputs and channels 16-31 configured as inputs.
4. Create several IONAMES within the configuration table, including a digital output IONAME for the DIO1 port A and a digital input IONAME for channel 31.

This exercise will create a configuration table starting with a blank table, as is normally done for a 500A.

Assigning Modules to Slots

To begin a practice session running CONFIG, make sure you have booted the system from a KDAC500 Working Disk or fixed disk. From the DOS prompt, type:

CONFIG <Enter>

CONFIG will run and present the HARDWARE SETUP screen without any modules.

1. The cursor should be over 1 in the SLOT column. If necessary, use the cursor keys to move the cursor to 1.
2. Press the special function F3 key to SELECT MODULE. The cursor will move to the MODULES window.
3. Use the cursor keys to move the cursor over AMM2.
4. Press <Enter>. The cursor moves back to 1 in the SLOT column.
5. Move the cursor to the 6 position in the SLOT column.
6. Press F3 to SELECT MODULE. The cursor moves to the MODULES window.
7. Use the cursor keys to move the cursor over AIM3.
8. Press <Enter>. The cursor moves back to 6 in the SLOT column.
9. Move the cursor to the 8 position in the SLOT column.
10. Press F3 to SELECT MODULE. The cursor moves to the MODULES window.
11. Use the cursor keys to move the cursor over DIO1.
12. Press <Enter>. The cursor moves back to 8 in the SLOT column.

This completes the selection of modules for the configuration table.

Setting Configuration Switches and Options

The following instructions describe the entry of switch information for the modules already entered into the configuration table.

1. This step begins the assignment of the ±10V A/D range for the AMM2 analog inputs. The cursor should be over 1 in the SLOT column. If necessary move the cursor to 1.
2. Press special function key F4 to SET SWITCHES. GAIN information replaces the MODULES window.
3. Use the cursor keys to move through the available range settings to –10. to +10.V. The AMM2 module is software-programmable for gain and range, so there are no hardware switches to be set. Gain is set in the IONAME commands.
4. Press <Enter>. The filter selection window will appear. Move the cursor to 100kHz.
5. Press <Enter>. The SINGLE-ENDED/DIFFERENTIAL choices appear. Use the cursor keys to toggle the cursor to SINGLE-ENDED.
6. Press <Enter>. This concludes setting for AMM2.
7. This step begins the setting of switches on the AIM3. Configuration of the AIM3A module is similar. Move the cursor to the 6 position in the SLOT column.

8. Press special function key F4 to SET SWITCHES. GAIN information replaces the MODULES window. Switch setting information appears in the AIM3 manual.
9. Use cursor keys to move through the available gain settings to x 100. Set the gain switches on the AIM3 according to the switch information at the top of the screen.
10. Press <Enter>. The SINGLE-ENDED/DIFFERENTIAL choices appear in the GAIN window. Use the cursor keys to toggle the cursor to DIFFERENTIAL. Set the input mode switches on the AIM3 for differential operation (both switches up).
11. Press <Enter>. The window shows a choice between AIM3 and AIM3A. Move the cursor to AIM3 and press <Enter>. This concludes set-up of the AIM3 switches.
12. This step begins the setting of switches on the DIO1. Move the cursor to the 8 position in the SLOT column.
13. Press F4 to SET SWITCHES. PORT information replaces the MODULES window. (A port is 8 input or output channels.)
14. The cursor will rest on either INPUT or OUTPUT according to the present setting for port A. The port or channel being affected is listed at the top of the PORT window. Use the cursor keys to toggle to OUTPUT, and then press <Enter>.
15. Repeat step 17 to set port B to OUTPUT. Repeat step 17, except with the cursor on "INPUT", for ports C and D. After you configure the last port, the cursor will move back to 8 in the SLOT column.

The DIO1 has a 4-bank DIP switch which is labeled "IN" and "OUT". Set the DIP switches for channels 0-15 to "OUT", and channels 16-31 to "IN". This information is not repeated at the top of the screen. The DIO1A module is software-programmable for I/O and does not have switches.

This concludes switch settings for the system.

Programming IONAMES as Part of the Configuration Table

The next instructions explain how to program IONAMEs directly in the configuration table. KDAC500 does not provide for an IONAME statement, so all channel information must be set up through CONFIG.

Three IONAME's will be set up:

- One IONAME for channel 2 of the AIM3 in slot 6 with differential input and global gain (GA%) of 10.
- One IONAME for DIO1 port A. KDAC500 will program these 8 lines for output as individual bits in a byte.
- One IONAME for the individual DIO1 input channel 31.

During this operation, new options will appear in the CHANNEL SETUP window. These options deserve special note since some of their functions are not immediately obvious.

ADD IONAME — adds an IONAME for a slot/channel. Make sure the cursor is on ADD IONAME. Press <Enter>. CONFIG prompts for the new IONAME. Type the name and press <Enter>.

COPY IONAME — copies the contents of an existing IONAME to the IONAME which is highlighted in the IONAME column. To select an IONAME for copying, press F4-SELECT IONAME. Move the cursor to the desired IONAME. Press F5-CHANNEL SETUP to return to the CHANNEL SETUP window. Move the cursor to COPY IONAME and press <Enter>. CONFIG will prompt for the IONAME to be copied to the highlighted IONAME. Type the name and press <Enter>.

DELETE IONAME — deletes the IONAME highlighted in the IONAME column. To select an IONAME for deletion, press F4-SELECT IONAME. Move cursor to the desired IONAME. Press F5-CHANNEL SETUP to return to the CHANNEL SETUP window. Move the cursor to DELETE IONAME and press <Enter>. CONFIG will prompt for a confirmation before it deletes the selected IONAME.

RENAME IONAME — renames the IONAME highlighted in the IONAME column. To select an IONAME for renaming, press F4-SELECT IONAME. Move cursor to the desired IONAME. Press F5-CHANNEL SETUP to return to the CHANNEL SETUP window. Move the cursor to RENAME IONAME and press <Enter>. CONFIG will prompt for the new name. Type the name and press <Enter>.

ACCURACY — specifies A/D resolution - 12, 14, or 16 bits. Move the cursor to ACCURACY and press <Enter>. Toggle to 12, 14, or 16 with the cursor keys and press <Enter>.

LOCAL GAIN — specifies hardware gain selected for analog input modules which have on-board gain amplifiers. For gains which are set by hardware switches, CONFIG reads the switch information you previously entered and updates LOCAL GAIN automatically. For modules with software-programmable gain, LOCAL GAIN shows legal gain values. Move the cursor to LOCAL GAIN and press <Enter>. Move the cursor to the desired gain and press <Enter>.

GLOBAL GAIN — Gain applied by the master analog module global gain amplifier, but programmed on a slot–and–channel basis for any analog input modules. Legal values are x1, x2, x5, and x10. Move the cursor to GLOBAL GAIN and press <Enter>. Move the cursor to the desired gain and press <Enter>.

RESISTOR — The value of a resistor which you have installed on an analog input module for current-to-voltage conversion. Move the cursor to RESISTOR and press <Enter>. CONFIG will prompt for the resistor value in ohms. Type in the value (integer) and press <Enter>.

FILTER — For modules with software-programmable filters. Move the cursor to FILTER and press <Enter>. Move the cursor to the desired filter and press <Enter>.

OFFSET — Enables or disables adjustable input offset feature of the AIM8. Move the cursor to OFFSET and press <Enter>. Move the cursor to ENABLED or DISABLED as desired, and press <Enter>.

MODE/RANGE — This menu selection set the mode of the PIM2 as either READ only, or READ and RESET. This will either allow totalizing or not, depending on your application. This command also sets the range of the AOM5 module.

CONVERSION — In general, this function allows you to define the type of conversion that should be performed on the input signal. It currently does not cause the conversion to take place but can be used for documentation of the type of signal or tranducer that you are using. For AIM8 and AIM9 you can set the calibration factor and calibration measurement units used to calibrate the card and transducer. Move cursor to CONVERSION and press <Enter>. Then move to SPECIAL and press <Enter>. Then select either LVDT/RVDT or STRAIN GAGE. CONFIG then prompts for the calibration factor in millivolts per volt excitation. Type in factor to three decimal places and press <Enter>. CONFIG prompts for calibrating units of measure. Type in units (integer) and press <Enter>.

DISPLAY FORMAT — This function does nothing for KDAC500. It will be used in future products to allow you to define the data formats for hard copy printouts of data, and for communication with analysis packages.

NOTE: The AIM6 module is supported in a different fashion from previous versions of CONFIG. To select the proper mode of operation for the AIM6, simply use the LOCAL gain menu selection.

1. Before continuing, make sure the cursor is in the SLOT column of the initial HARDWARE SETUP screen. Press Esc several times if necessary to return to the HARDWARE SETUP screen.
2. Use the cursor keys to move the cursor over 6 in the SLOT column. Press special function key F5 for the CHANNEL SETUP screen.
3. Press F3 to SELECT CHAN/PORT. The cursor will move to 0 in the CHANNEL column.
4. Use the cursor keys to move the cursor to 2 in the CHANNEL column.
5. Press <Enter>. The cursor will move to ADD IONAME in the CHANNEL SETUP window.
6. Press <Enter>. CONFIG will prompt for an IONAME. This name can contain up to 8 letters and numbers. In this case, type in "TEST" and press <Enter>.
7. Move the cursor to ACCURACY in the CHANNEL SETUP window.
8. Press <Enter>. A/D menu choices appear in the CHANNEL SETUP window. Use the cursor keys to select 16 BIT and press <Enter>. The CHANNEL SETUP choices will reappear.
9. Use the cursor keys to move the cursor to GLOBAL GAIN.
10. Press <Enter>. Global Gain choices will appear in the CHANNEL SETUP window. Use the cursor keys to move the cursor to the x10 position. Press <Enter>.
11. This completes the configuration of IONAME parameters for the AIM3 module. Go directly to creating an IONAME for another slot by pressing F2 for SELECT SLOT.
12. The following instructions set up IONAME's for the DIO1 module port A and Channel 15. Use the cursor keys to move the cursor to 8 in the SLOT column.
13. Press F3 for SELECT CHAN/PORT. The cursor will move to the CHANNEL column. Press F3 again to move to the PORT column.
14. Use the cursor keys to move the cursor to A in the PORT column.
15. Press <Enter>. The cursor will move to ADD IONAME in the CHANNEL SETUP column.
16. Press <Enter>. CONFIG will prompt for an IONAME. In this case, type in "OUTA" and press <Enter>. (The only parameter you can enter for digital I/O is the name.)
17. Press F3 for SELECT CHAN/PORT. The cursor will return to the PORT column.
18. Press F3 for SELECT CHAN/PORT again. The cursor will move to the CHANNEL column.
19. Use the cursor keys to move the cursor to 31.
20. Press <Enter>. The cursor moves to ADD IONAME in the CHANNEL SETUP column.
21. Press <Enter>. CONFIG will prompt for an IONAME. In this case, type in "IN15" and press <Enter>. (The only parameter you can enter for digital I/O is the name.)

22. This completes the assignment of IONAMES for the DIO1. Press F10 to return to the HARDWARE SETUP screen.

File I/O — Saving the Configuration Table

These steps will save the configuration table to disk.

1. Press F2 for FILE operations. The MODULE window will be replaced by a FILE I/O menu. It includes choices for loading, saving, and deleting files, and for changing default drives and directory.
2. Use the cursor keys to move the cursor to SAVE FILE.
3. Press <Enter>. The FILE I/O window will show the drive, directory, and one or more configuration file names. If no CONFIG filename exist, only "new file" will be shown. Use the cursor keys to move the cursor to CONFIG or "new file".
4. Press <Enter>. If you indicated "new file", you will be prompted for a filename. If so, enter "CONFIG". The file will be saved as CONFIG.TBL. The filename extension ".TBL" is added automatically.

To save the table under another existing filename, move the cursor to that filename in place of "CONFIG". If there are more filenames than will fit in the window, press the cursor control PgUp or PgDn keys to call up the additional filenames. Move the cursor to the desired filename and press <Enter>.

To specify a completely new filename, select the "new file" menu option. CONFIG will prompt for a filename prefix, up to 8 characters. These characters must be legal DOS filename characters. Enter only the filename prefix. Do not enter ,"TBL".

This concludes a typical configuration.

File I/O — Loading a Configuration File

These instructions presume that CONFIG.EXE is running, and that you want to modify a configuration table which resides on the disk. The desired file can be called back into the configuration table selecting the LOAD FILE option under F2-FILE.

1. Press F2 for FILE I/O. The MODULES window will be replaced by the FILE I/O.
2. Use the cursor keys to move the cursor to LOAD FILE.
3. Press <Enter>. The FILE I/O window will show the drive, directory, and one or more configuration file names. Use the cursor keys to move the cursor to the desired filename. In this case select "CONFIG". If there are more filenames that will first in the window, press the cursor control PgUp or PgDn keys to call up the additional filenames. Move the cursor to the desired filename.
4. Press <Enter>. The file will be loaded into the CONFIG program workspace. Make whatever changes you desire.
5. To preserve the changes refer to the instructions above for saving a configuration table file. You can save the configuration table under the name CONFIG, or its original name, or as a new name.

The configuration file thus loaded into memory will overwrite any configuration table that may currently be in memory.

File I/O — Deleting a Configuration Table

These steps will delete a configuration file from the disk.

1. Press F2 for FILE operations. The MODULE window will be replaced by the FILE I/O menu.
2. Use the cursor keys to move the cursor to DELETE FILE and press <Enter>.
3. CONFIG will list the configuration table file in the FILE I/O window. Move the cursor to the filename to be deleted. If there are more filenames than will fit in the window, press the cursor control PgUp or PgDn keys to call up the additional filenames. Move the cursor to the desired filename.
4. Press <Enter>. CONFIG will prompt for a confirmation. Type "Y" and press <Enter>.

File I/O — Changing the Default Drive or Directory

1. Press F2 for FILE I/O.
2. To change the default drive, select CHANGE DRIVE from the FILE I/O window and press <Enter>. CONFIG will prompt for the drive ID. Type only the letter of the drive (no colon) and press <Enter>.
3. To select a different directory, select CHANGE DIRECTORY from the FILE I/O window and press <Enter>. CONFIG will prompt for the directory name. Type an existing DOS directory name and press <Enter>.

If the new directory name is on the current default drive, subsequent disk I/O will address that directory. If the directory is on a drive other than the current directory drive, you must also CHANGE DRIVE to make the new drive the default. Attempting a change to a nonexistent drive or directory will produce an error message.

Generating a Configuration Table File Report

CONFIG can generate a report of the configuration table and IONAME's. The report can be listed to the screen, written to a disk file, or sent to the printer.

1. Press F9 to LIST. The MODULES window changes to show the output destinations FILE, PRINTER, or SCREEN.
2. Use the cursor key to select the desired destination for the report and press <Enter>. If you select SCREEN, CONFIG will show a summary report of the configuration and IONAMES. Press any key to return to the FILE, PRINTER, or SCREEN menu. If you select FILE, CONFIG will prompt for the filename. Type a legal DOS filename and press <Enter>. The filename will overwrite any existing file with the same name. If you select PRINTER, CONFIG will prompt for confirmation that the printer is ready. Be sure that the printer is connected and turned on. Type "Y" and press <Enter>.
3. Press Esc to return to the HARDWARE SETUP screen.

Leaving the Config Program

1. If necessary, press Esc several times or <F10> to return to the HARDWARE SETUP screen.
2. Press <F10> to exit. The program will remind you to save the changes. If you have saved them and want to end CONFIG, enter "Y" and press <Enter>. If you need to save the configuration table and haven't done so, press "N" and <Enter>. CONFIG will move directly to the SAVE menu. Save the table using the F2-FILE instructions before leaving CONFIG.

Aborting a Configuration Session

You may abort a configuration session by repeatedly pressing the escape ("Esc") key until the HARDWARE SETUP screen reappears. Then, press <F10> to exit to DOS and answer the prompt "Y" for Yes. If you do not save the configuration table, all configuration files stored on the disk will remain unchanged, and any changes made to the configuration table in memory will be lost.

You may also abort a CONFIG session from anywhere in CONFIG by pressing <Alt><F10>. No prompt will be given for exit verification.

Methods of Creating a KDAC500 Program

Before you begin with KDAC500, you should become completely familiar with the programming language package you have chosen and how to write, debug, and compile a program. The KDAC500 documentation does not replace the compiler reference manual.

There are quite a few ways of creating and running KDAC500 programs. The following paragraphs describe these methods. In each case it is assumed that access to the interface routines is available. Before the K500.EXE kernel is loaded, KDAC500 programs can be compiled but cannot be run.

General Method for Creating Executable Test Programs

If you have experience in using Keithley's Soft500, Quick500, or KDAC500/I software, you will note some major changes in how you approach programming with KDAC500/B and KDAC500/M. The most important of these is that, under KDAC500/B and /M, you may not be able to run and debug your program from within the compiler's editor/environment (provided it has one). For example, Soft500 and KDAC500/I programs are written, run, and debugged entirely from within the BASIC interpreter environment.

In contrast, KDAC500/B and KDAC500/M operate with several compilers. The memory requirements of these compilers leave very little free memory for arrays. Therefore, it is more realistic to create your program source code with a text editor, and then compile the program from the DOS prompt. Note that the QuickBASIC, Quick C, Microsoft C, and Borland Turbo compilers have an editor/environment which enables you to write and edit source programs and then proceed directly to the compile/link step. These environments automate many of the steps which must be completed manually when compiling from the DOS prompt.

A second consideration for KDAC500 programming is that you must have a clear design for your test program before you begin programming. This includes identifying the various input and output channels, and then running CONFIG.EXE to set up the necessary IONAMEs in the hardware configuration table. You will also need to identify the amount of data that will be generated when the program runs. While some packages permit programming "on the fly", KDAC500 requires a more organized approach. In the long run, this approach will result in better, more easily maintained test programs.

The general approach to creating executable files with KDAC500/B and KDAC500/M is as follows:

3. Install your compiler and the KDAC500 software on your computer system. Confirm that the compiler can "find" any header or library files, and that it runs properly before you attempt to write a KDAC500 programs.
4. Design your test program.
5. Run the KDAC500 CONFIG.EXE program to create the necessary hardware configuration table for your DAC hardware. Decide on the channels of input and output, and set this

information up as IONAME data in the table. Note that there is no provision for setting up IONAMEs directly within your test programs.

6. Run the compiler's editor/environment and write or edit your program source code. Note that you must "include" the appropriate KDAC500 langauge support file in your program: KDAC500.BI for QuickBASIC programs, KDAC500.H for Microsoft C, Quick C, and Borland Turbo C, "include 'KDAC500.FI' " and "include 'KDAC500.FD' " for Microsoft FORTRAN, and the statement "uses KDAC500" for Borland Turbo Pascal.
7. Compile, link, etc. to produce an executable file. You can usually do this most conveniently from within the compiler environment. Most environments will aid in the debugging process by identifying compile or link errors. You can normally compile and link from the DOS command line as well, but without some of the benefits of the environment (see your compiler manual for the correct DOS command line syntax and other procedural details).
8. Run the successfully compiled program with the KDAC500 "KRUN" utilities. See the next section "Methods of Running a KDAC500 Progam" for details.
9. Test the various program functions to make sure it does what you expect.
10. Repeat 4-8 until the program functions as desired.

QuickBASIC Programming

OBJECTIVE: Run the QuickBASIC editor/enviroment and create an executable (.EXE) program. Generally speaking there is not enough RAM left with QuickBASIC, K500, your program and a Quick Library all resident in memory at the same time, so it isn't feasible to run KDAC500 programs from within the QuickBASIC environment. However, the environment is an excellent place for editing, debugging, and compiling your KDAC500 programs.

METHOD: First you must create a Quick Library that QuickBASIC can use. The library file KDAC500.LIB is the file that will be used to make it. This need only be done one time. Once the Quick Library has been created it can be used for all your QuickBASIC programs.

At the DOS prompt type:

link /QU KDAC500.LIB,,nul,BQLB45<enter>

The file BQLB45 is a library file that is supplied with version 4.5 of QuickBASIC. It is used to create Quick libraries for version 4.5 of QuickBASIC. Refer to the section on Quick Libraries in the QuickBASIC manual for more information.

Next, start QuickBASIC and load the KDAC500 Quick Library.

At the DOS prompt type:

qb /l KDAC500<enter>

Write and edit your program using QuickBASIC's smart editor and syntax checker. The interfaces to the KDAC500 library are defined in the file KDAC500.BI which must be included in your QuickBasic program with the statement:

'$INCLUDE: 'KDAC500.BI'.

When the program is satisfactory, use the QuickBASIC "ALT R" command to access the "RUN" pull-down menu.

Select the "Make EXE" option and complete the remaining QuickBASIC steps to produce a compiled test program file on disk. Because the KDAC500 Quick Library was loaded when QuickBASIC started, QuickBASIC "knows" to look for the KDAC500.LIB file automatically when it creates the .EXE file. Leave QuickBASIC and execute the program with the KDAC500 "KRUN" utility; e.g. "KRUN <your program>".

C Programming

OBJECTIVE: Run the Turbo C or QuickC editor/environment and create an executable program. As with QuickBASIC, there is not enough RAM to execute your KDAC500 program from within the environment. However, the benefits of the environment's debugging and editing facilities outweigh the drawbacks.

METHOD: First a file that defines the modules that make up your program must be created. In Turbo C this is called a project file. In QuickC it is called a program list. You can use the Turbo C editor to create a project file for your program. QuickC has a facility for creating program lists. Let's assume that our program is named TEST1.C.

The project file for TEST1.EXE (Turbo C) will contain two lines:

```
TEST1.C
TURBO500.LIB
```

If TEST1.EXE was made of multiple modules each module would be entered into the project file. See the Turbo C documentation for more information on project files. The important thing to remember about project files used to create KDAC500 programs is the inclusion of the TURBO500.LIB line. If it is not present, Turbo C does not know how to resolve the references to KDAC500 functions. The interfaces to KDAC500 library are defined in the KDAC500.H header file. This file must be included at the top of your program with any other include's that are needed.

The program list for TEST1.EXE (QuickC) will contain these two files:

```
TEST1.C
KDAC500.LIB
```

If TEST1.EXE was made of multiple modules each module would be entered into the program list. See the QuickC documentation for more information on program lists. The important thing to remember about program lists used to create KDAC500 programs is the inclusion of KDAC500.LIB. If it is not present, QuickC does not know how to resolve the references to KDAC500 functions.

Complete the remaining steps to produce a compiled program on disk. Refer to the documentation for your compiler. The completed.EXE file can be executed from the DOS command line using the KDAC500 "KRUN" utility, e.g. "KRUN <your program>".

Pascal Programming

OBJECTIVE: Run the Turbo Pascal or Quick Pascal editor/environment and create an executable program. As with QuickBASIC, there is not enough RAM to execute your KDAC500 program from within the environment.

METHOD: KDAC500 is very easy to implement with Turbo Pascal or Quick Pascal. The only requirement is the one statement:

```
uses KDAC500;
```

This command must be included within your Pascal program if you wish to access the KDAC500 functions. Standard Pascal Units can be used in conjunction with KDAC500 (i.e. uses Crt, Dos, KDAC500;). Refer to the Turbo Pascal or Quick Pascal documentation for more information on Pascal Units. Run the completed .EXE file from the DOS command line with the KDAC500 "KRUN" utility, e.g. "RUN <your program>".

FORTRAN Programming

OBJECTIVE: Create a program using the Microsoft FORTRAN 5.0 package.

METHOD: Creating an executable file using FORTRAN is quite easy. First, create the source code with an ASCII-compatible word processor or notepad editor. The program must contain the following lines at its beginning:

```
include 'KDAC500.FI'
include 'KDAC500.FD
```

When the source code is complete, save it as an ASCII file. Compile the source by executing the following command at the DOS prompt:

```
fl / c / AL<filename.ext>
```

The filename must be complete, including the filename extension. (The extension typically used for FORTRAN program is ".FOR").

Next, link the object module by executing the following command at the DOS prompt:

```
link <filename>,<filename>,nul,KDAC500
```

The first <filename> refers to the name of the .OBJ module created during the compile step. The second <filename> refers to the name of the resulting .EXE file. Normally, both names will be the same as the original FORTRAN source. The filename extensions are not used here.

Run the completed .EXE file from the DOS command line with the KDAC500 "KRUN" utility, e.g. "KRUN <your program>".

Make Files

OBJECTIVE: Create an executable program using a program build utility such as MAKE.

METHOD: The only requirement for using a MAKE utility is to be sure to link with the correct library or there will be unresolved references. The stand alone Turbo Pascal and Quick Pascal compilers are exceptions to this rule because the "Uses KDAC500;" statement tells the compiler what to use. Unfortunately there is no way to tell the other languages which library to use that makes linking as easy as with Pascal. You must specify the name of the library at link time.

```
TEST1.EXE : TEST1.C                          ; Turbo C example
    tcc –c –ms test1.c                       ; small memory model
    tlink c0s+test1,test1,nul,cs + turbo500

TEST1.EXE : TEST1.C                          ; QuickC example
    qcl –c test1.c
    link test1,test1,nul,KDAC500

TEST1.EXE : TEST1.BAS                        ; QuickBASIC example
    bc test1;
    link test1,test1,nul,KDAC500

TEST1.EXE : TEST1.PAS                        ; Turbo Pascal example
    tpc test1.pas

TEST1.EXE : TEST1.PAS                        ; Quick Pascal example
    qpl test1.pas


TEST1.EXE : TEST1.FOR                        ; Microsoft FORTRAN example
    FL/C/AL Test1.FOR
    LINK TEST1, TEST1, NUL,KDAC500
```

Methods of Running a KDAC500 Program

Before you continue with this section, you must have installed KDAC500 and your compiler on your computer, and also created a configuration table file for each data acquisition system installed in the computer.

KDAC500 can be started and run in a number of different operating modes. To understand these modes, it is important to note the functions of several files in the KDAC500 package.

KDAC500 FILES:

KDAC500.LIB — A library file which resolves the KDAC500 functions you include in a KDAC500 program. KDAC500.LIB is the library of routines that is used with Microsoft C, QuickC, QuickBASIC v4.x, and Microsoft FORTRAN 5.0. These statements communicate with the data acquisition hardware. KDAC500.LIB must be on the disk and accessible when you link your programs or error messages will result.

TURBO500.LIB — Same as KDAC500.LIB but used with Borland Turbo C 2.0

KDAC500.TPU — Same as KDAC500.LIB but used with Borland Turbo Pascal 6.0

KDAC500.T55 — Same as KDAC500.LIB but used with Borland Turbo Pascal 5.5

KDAC500.T50 — Same as KDAC500.LIB but used with Borland Turbo Pascal 5.0

KDAC500.QPU — Same as KDAC500.LIB but used with Microsoft Quick Pascal 1.0

K500.EXE — This is the memory-resident kernel which controls the data acquisition hardware. K500 must be in memory in order for you to run any KDAC500 programs. KRUN.BAT and KLOAD.BAT load K500.EXE into memory (see KRUN.BAT and KLOAD.BAT descriptions below). K500.EXE must be on the disk and accessible when you run KDAC500 programs.

KRUN.BAT — KRUN.BAT runs KDAC500 programs that have been compiled into DOS executable form (.EXE). KRUN will return to DOS when the program terminates. The KRUN file loads the K500.EXE kernel into memory, and then runs the specified KDAC500 program (.EXE file). KRUN loads the KDAC500 KERNEL (K500.EXE) temporarily. Once the KDAC500 program terminates, K500 removes itself from memory before returning to DOS.

The syntax for this command is:

KRUN <your program name> <arg1> <arg2>....

If there are any arguments to be appended to your program, enter them after the name of your program.

KLOAD.BAT — This batch file loads the K500.EXE kernel and makes the kernel permanently resident in memory. After K500.EXE has been made memory resident, KLOAD will return to DOS where you can run your compiled programs by simply entering the program name at the DOS prompt.

The syntax for this command is:

```
KLOAD
```

Operating more than One Data Acquisition System on a Single Computer

KDAC500 has the ability to run up to four data acquisition systems using one computer as a host.

Set-up Procedure

You must adhere to the following setup procedure to operate more than one data acquisition system on a single computer:

Each System Must Have its own Interface

Each data acquisition system must have its own interface plugged into the computer. (This requires that the computer have enough expansion slots available.) The KDAC500 INSTALL program will assign the first interface entered in the installation screen as the "master". The interrupts of the master interface card become the master interrupts and control the background data acquisition of all the interface cards. This is the case even though all of the interfaces are capable of generating interrupts for data acquisition.

Each Interface Must Have A Unique Address

Each interface you install must be set to a different address. Generally, the address ranges CAF80-CFF80 and DOF80-DFF80 can be used successfully on both XT- and AT-class computers.

Each System Must Have A Separate Configuration Table File

For multiple acquisition systems, KDAC500 must have a method of accessing the specific hardware in each system. This task is achieved by using a separate, unique configuration table (".TBL") file for each interface. Accordingly, you must run CONFIG and create a hardware configuration table file for each interface address. Each file must have a unique file name. Normally, you should use the names provided for by the INSTALL program (CONFIG, CONFIG1, CONFIG2, CONFIG3).

Each configuration table file may be a duplicate of the others in terms of the hardware modules and placement within the slots. However, all IONAME's which are set up within the configuration table files must be different. No two tables may have duplicate IONAMEs. See the section of running the configuration program CONFIG for more information.

The KDAC500 Installation Must Identify all the Interface Addresses and Config Filenames

If you have done an installation for one data acquisition system, and later want to add more systems to one computer, you must do a re-installation with the addresses and the configuration table filenames for each interface plugged into the computer.

Running Multiple Systems

To run KDAC500 with multiple systems, start KDAC500 as you normally would with a single system. Write your test program. Observe that the memory in the computer is now serving multiple acquisition systems. You may have to adjust the number or size of the data arrays in order to fit all data into the available memory.

Initializing Hardware at Power-up with HARDINIT

(NOTE: This information describes how to set up the HARDINIT command. Please refer to any other safety-related information concerning specific cards in your data acquisition system.)

KDAC500 includes a utility program "HARDINIT.EXE" which can initialize all the digital and analog output modules to a known state when the system is booted.

When you turn on the data acquisition system, some output modules may power up in a "scrambled" state. This condition would normally exist until you load and run a data acquisition program which initializes the hardware. By inserting HARDINIT with the necessary parameters into the AUTOEXEC.BAT file, system initialization will occur when the computer boots and executes AUTOEXEC.BAT. This occurs when the computer is turned on, or when you do a warm reboot (CTRL-ALT-DEL).

You must observe a few precautions when you use HARDINIT:

1. The data acquisition system(s) must be turned on at the time the AUTOEXEC.BAT file with HARDINIT is executed. HARDINIT will not initialize hardware which is turned off.
2. HARDINIT uses the configuration table files to identify and initialize output modules. The configuration table filenames must include the drive letter and path corresponding to their location on the fixed disk or floppy diskettes. Failure to do so will produce the error message "Configuration file not found", and the hardware will not be initialized.
3. The KDAC500 utility HARDINIT.EXE must be accessible to AUTOEXEC.BAT, either directly, or in the search path. Install copies HARDINIT.EXE to the working diskette or fixed disk directory along with the rest of KDAC500. For a fixed disk, it is easiest to simply recopy HARDINIT.EXE to the fixed disk root directory.

Format of the HARDINIT command:

HARDINIT –c 0xAAAA <CONFIG table filename> –p

Where:

–c specifies that the following parameter is an interface address and configuration filename.

0xAAAA is the address of the interface. AAAA represents the four most significant digits of the address in hexadecimal notation.

<filename> is the complete configuration table filename (with full path spec.) corresponding to the interface at AAAA.

–p is a "pause" option. Ending the HARDINIT command line with "–p" will type a reminder to the screen that the data acquisition systems must be turned on. You will have to press the <Enter> key before the initialization takes place.

The sequence "–c 0xAAAA filename" must be entered for each data acquisition system that is to be initialized. The –p option is used, it should be inserted at the end of the HARDINIT command line and used only once per HARDINIT command line.

An example of an AUTOEXE.BAT containing one HARDINIT is as follows:

```
ECHO OFF
CLS
HARDINIT –c 0xAFF8 C:\K500\CONFIG.TBL –c 0xCFF8 C:\K500\CONFIG1.TBL –p
DATE
TIME
```

This AUTOEXEC.BAT file will initialize two systems simultaneously: one at address AFF80 and one at address CFF80. It will also pause for a press of the <Enter> key.

An alternate technique for using HARDINIT is to enter a separate HARDINIT command for each system. An example AUTOEXEC.BAT file containing two HARDINIT's is as follows:

```
ECHO OFF
CLS
HARDINIT –c 0xAFF8 C:\K500\CONFIG.TBL –p
HARDINIT –c 0xCFF8 C:\K500\CONFIG1.TBL –p
DATE
TIME
```

This AUTOEXEC.BAT will initialize systems at address AFF80 and at address CFF80. The systems will be initialized sequentially. HARDINIT will pause for a press of the <Enter> key for each system.

CHAPTER 2
KDAC500 System Features

KDAC500 Memory Management

Background/Foreground

Triggering

Engineering Units Conversion

KDAC500 Memory Management

The KDAC500 Memory Management System provides an extremely flexible and sophisticated memory management system which is well-suited for data acquisition and control applications.

Its main purposes are:

- To provide very large data arrays.
- To provide data structures with special features which make them particularly useful for data acquisition and control applications.
- To free your program's data space for your own applications.
- To allow you to implement two memory management systems at once (KDAC500's and your compiler), each with its own advantages, and to be able to make the best use of each.

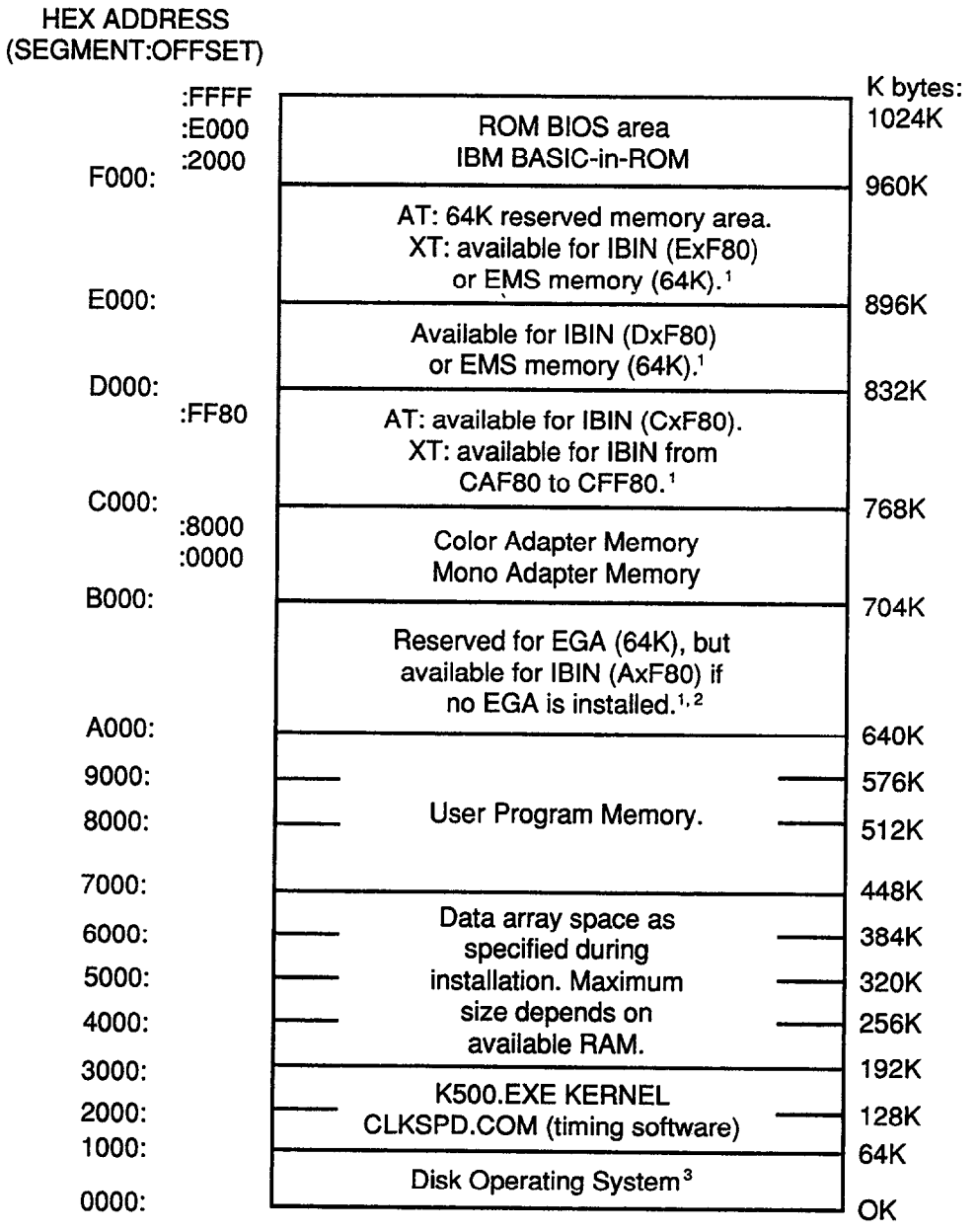
To understand the KDAC500 Memory Management System, a memory map is needed. The following diagram represents the typical memory usage for IBM, Compaq, and MS-DOS Versions 3.0 or later.

The exact boundaries of the blocks containing the Disk Operating System, your program, and K500.EXE may vary depending on their versions. The hardware shown in memory above A000:0000 will vary depending on the options installed in the computer.

When you install KDAC500, you must consider the information in the memory maps, including the memory size of your computer. You must have a clear understanding of your ultimate memory requirements when you run KDAC500's INSTALL program. If you select too large an array space, you will receive a variety of error messages from KDAC500, your application, or both. Numbers for your system depend on the rev level and size of DOS, the size the K500.EXE kernel, and any other memory-resident programs you may be running. These are approximations, and should be verified for your particular hardware, software, and application program.

The KDAC500 startup batch files KLOAD.BAT and KRUN.BAT contain a "–m" parameter that specifies the amount of memory to be reserved for arrays. This parameter represents the number of 16-byte memory segments that KDAC500 will have for data arrays. The KDAC500 INSTALL process sets the –m parameter according to your input at the time of installation. The easiest way to alter the memory reserved for arrays is to rerun INSTALL. Alternately, you can use an ASCII word processor or DOS's EDLIN utility to modify the KLOAD or KRUN batch files directly.

MEMORY MAP - IBM XT or AT WITH 640K AND FIXED DISK



1. Any address in this block ending in "F80" can be chosen provided there is no conflict with other hardware. Factory-suggested address for IBIN is CFF80.

2. ON IBM PC's with 256K mother board, and on some compatible computers, this space can also be populated with RAM to increase addressable system RAM to 704K. (Not applicable for IBM Portable PC, XT, and AT, or systems with EGA).

3. Varies in size depending on amount of RAM specified for array during installation. A small portion of the DOS file COMMAND. COM resides at the top of system RAM. It is not shown in this diagram.

Table 2-1. Approximate Array Space for KDAC500/B and KDAC500/M

Total System RAM	RAM Available for Arrays
640K	~300K
576K	~236K
512K	~172K

How Data Arrays are Created for KDAC500

NOTE: The following programming commands and examples are shown in QuickBASIC format, as would be used with KDAC500/M. Coding conventions and punctuation (such as single or double quotes, underscores, and periods), vary with from language to language. However, actual KDAC500 command names, general usage, and functionality are the same, regardless of KDAC500 version.

KDAC500 input commands (BGREAD & ANINQ) create KDAC500 arrays automatically when they are executed. KDAC500 arrays can also be created with the KDAC500 command ARMAKE (ARray MAKE).

For example, the statement:

```
call armake("array1", 1000, "ION1 ION2")
```

creates an array named **array1** with 1000 elements and a width of 2 elements. The IONAMEs after the array depth determine the array type. If ION1 is an analog channel, each element will occupy two bytes of RAM. The name given to the array enables other KDAC500 commands to access the array.

Using the array name, you may put values in the array, retrieve values from it, save/retrieve the array to/from disk, and perform several other tasks on the array.

In the above example, the array type was determined by the IONAME. There are four types of KDAC500 arrays:

- BIT ARRAY — Packed bit arrays. Values stored as 0's and 1's. Used with digital input and output commands.
- BYTE ARRAY — Elements of eight bits each. Values stored as unsigned bytes, 0 to 255. Used especially with commands that access digital I/O ports.
- INTEGER ARRAY — Elements two bytes long. Values stored as unsigned integers, 0 to 65535. Used especially with analog input and output commands.
- LONG ARRAY — Elements four bytes long. Values stored as unsigned long integers, 0 to 4,294,967,295. Long arrays are used with the 32-bit pulse input commands.

Every time you create a KDAC500 array, the array type is determined by the IONAME list. This information tells KDAC500 how much memory should be allocated or accessed for each data point in the array, as well as the intended use for the array.

KDAC500 can retrieve any value stored in a KDAC500 array as raw binary (i.e A/D counts for analog data) or engineering units. Even though values are stored as integers, they may be returned as volts, milliamperes, degrees C, or other engineering units according to the module, transducer, and EUF parameter being used.

Aside from making arrays and accessing the data, KDAC500 provides many commands to increase the usefulness of KDAC500 arrays. Arrays can be deleted (deallocated from RAM) with the ARDEL command. A descriptive string may be associated with the array using ARLABEL, so that if it is saved to disk with ARSAVE, then loaded back into RAM with ARLOAD, any information in the string will remain associated with the array. ARSTATUS will return information to you regarding the array. ARLASTP will return the depth index of the most recently (in real-time) sampled or output value in a KDAC500 array (see the command reference section for more details).

Finally, KDAC500 graphics commands support KDAC500 arrays (GRAPH, GRAPHRT, HGRAPHRT). In addition, these commands all support internal conversion to volts and engineering units.

Background / Foreground

One of KDAC500's most powerful features is its background/foreground processing. Background/foreground takes KDAC500 far beyond the abilities of simple programs, allowing the Keithley data acquisition system to manage independent tasks at machine language speeds -- without machine-language programming.

The concept of KDAC500's background/foreground is easier to understand if one examines the interface cards included with the Keithley data acquisition systems. The IBIN cards contain interrupt-generating circuitry which works in conjunction with the computer's interrupt handling abilities. The KDAC500 function INTON (INTerrrupts ON), allows you to specify the frequency with which you want interrupts to be generated. The command for generating an interrupt every 100 milliseconds looks like this:

```
CALL INTON( 100, MIL)
```

When the INTON function is called, the programmable interval timers on the Keithley interface installed in the computer begin to generate interrupts at the specified interval. Each time an interrupt occurs, the processor jumps from executing foreground tasks (your program or foreground KDAC500 commands) to executing background tasks. Each background task is checked to see if it requires handling on a given interrupt, and is executed appropriately. When all the background tasks have been worked through, control is returned to the foreground until the next interrupt.

The background acquisition tasks should have the highest priority. The KDAC500 installation program permits you to specify NMI, CLOCK, or IRQ 9, 3, or 5 as the interrupt controlling background acquisition. All the time-critical data acquisition and control tasks are thus handled at regular intervals. In order to maximize accuracy, special care is taken by KDAC500's background controller to assure that samples and updates are not time-skewed. Most non-background KDAC500 commands and all ordinary BASIC commands are classified as "foreground" commands, and will be executed during free time when the system is not servicing the interrupt-driven commands.

KDAC500's background/foreground operation is a form of multitasking in that it allows the computer to perform one high-priority task (background) and one low-priority task (foreground) in a time-sharing fashion.

It is important to remember that although the foreground and the background appear to operate simultaneously, the computer is really switching its attention back and forth between two tasks. Foreground/background commands also operate independently. After a background command is issued, it is only possible for the foreground to know how the background commands are progressing through the use of various status commands.

Background/foreground processing gives the programmer considerable flexibility. While data acquisition and control sequences execute in the background, the foreground can be used to monitor the status of background tasks, to communicate with the operator or with

other peripheral devices, or to perform analysis on previously collected data. In fact, it is possible to have the KDAC500 program set up data acquisition sequences in the background, then terminate and pass control back to DOS. Depending on the speed of acquisition and the amount of free RAM, you may be able to run other application programs while the background tasks continue. Later, another KDAC500 program can be run which harvests the data from the arrays filled by the first program.

Foreground and Background Communication

So far, we have discussed one basic way that the foreground communicates with the background: setting up task sequences with KDAC500 background commands. There are a number of other ways that the foreground can act on and track the background's operations.

To begin with, KDAC500 offers a background triggering capability. This feature is covered thoroughly elsewhere in this manual. However, we will examine it briefly here in the general context of background operation. As we have discussed, once background commands have been started by the foreground, they can proceed independently of the controlling program. Background triggering allows acquisition and control sequences to be started independently of the foreground.

The KDAC500 command, TRIGGER, can act as a background trigger. This command allows the background to react directly to real-world events such as digital inputs or threshold values on analog channels.

Background commands such as BGREAD or BGWRITE can be linked to TRIGGER with a special parameter, the "trigger mode". When KDAC500 encounters this parameter, the task is set up in the background but not started. It remains latent until triggered, at which time the task begins without foreground intervention.

One of the background's most useful features is that once sequences are set up, they do not require the attention of the controlling program. However, there are many instances when it is useful for the foreground to be able to monitor ongoing background tasks. For example, after acquiring data with BGREAD (BackGround READ), it is often desirable to save that data to disk. However, before the foreground program can issue the ARSAVE command, it must determine that the acquisition sequence is complete. Similarly, when alarm conditions are detected, orderly shutdown procedures may need to know which sequences are active, and which are waiting for triggers.

For situations like these, KDAC500 includes commands which allow the foreground to check the progress of the background commands. In both of the above examples, the BGSTATUS command could be used to poll the background for the necessary information.

BGSTATUS assesses whether a background task is executing, waiting for a trigger, or finished. To facilitate this, all background commands have an optional parameter, the "background function name" (BFN) which allows tasks to be easily identified. A background task's parameter list should always include this parameter if it will be monitored from the foreground.

There are also situations where it is necessary to know exactly how much of a data array has been filled or output, or it may be desirable to synchronize computations in the foreground with acquisitions in the background. In both cases, ARLASTP (ARray LAST Point) can provide the necessary data.

ARLASTP returns the depth index of the last point in an array which has been accessed by a KDAC500 background command. Remember that each width is associated with a particular channel, and that depth levels are associated with the number of samples acquired. ARLASTP is particularly useful when you wish to perform some analysis on part of an array, while the rest of the array is still being acquired. It can be used in conjunction with ARGET (ARray GET) to retrieve one or more of the most recently acquired data points for computation. In process control applications, ARLASTP and ARGET can be used to synchronize data logging and the acquisition of data for control loops.

The foreground program can also act to stop, clear, and restart the background, as well as to stop ongoing tasks selectively. INTON (INTerrupts ON) starts the background, and can be issued either before or after background commands have been set up. This allows it to start any number of tasks simultaneously so that they are synchronized. Similarly, INTOFF (INTerrupts OFF) stops the background, whether or not tasks are currently running. When INTOFF is issued, no tasks are cleared, so that the background can be temporarily halted and then restarted from the same place (note, however, that all timers are cleared by this procedure).

If desired, the background can be cleared of all active and latent tasks with the BGCLEAR (BackGround CLEAR) command. The BGHALT command can be used to stop and clear background tasks selectively.

The Singleground Mode

Foreground/background processing is extremely useful for most measurement and control tasks. In some circumstances, however, it may be preferable to operate in KDAC500's singleground mode. Singleground operation is very similar to foreground operation with the exception that background functions are not allowed. This allows KDAC500 to operate as fast as possible. In particular, the very high speed sampling provided by ANINQ (ANalog INput Quick) and ANOUTQ (ANalog OUTput Quick) can only be achieved by suspending background operation.

KDAC500's Timers

Along with data acquisition and control functions, KDAC500's timers also operate in the background. Each time an interrupt occurs, all of the timers which are currently on will "tick". These timers require that the background be on in order to operate. If the background is suspended (with INTOFF), updating of the software timers will stop and they will retain their last values until they are restarted or interrupts are re-enabled. Turning interrupts on at any time (with INTON) will clear all timers, and they will begin counting from zero.

Disk Access and Interrupts

KDAC500 permits disk access with ARSAVE and ARLOAD while interrupts are enabled. Due to the way the PCs perform disk access, processing may be slowed during disk access. If a background task is executing at this point, data sampling may be time-skewed, but the system will not hang up.

Triggering

Conditional triggering is a very useful tool in data acquisition and control systems. It provides a means for external (environmental) and internal (program) conditions to elicit a programmed response.

An example of external triggering would be a case in which the opening or closing of a switch activates a background command which then begins outputting a voltage to a piece of equipment. In this case, the environment influences a program which in turn influences the environment. This kind of bi-directional communication between the computer and the world is extremely useful in process control applications.

An example of an internal trigger would be a case in which the completion of an analog input task triggered the start of a digital output sequence. This kind of trigger is useful when two background commands must operate in sequence.

In order to implement these examples and others, KDAC500 provides five types of conditional triggers. Each has advantages and disadvantages, but together they offer a powerful means of supporting every need for conditional triggering and response. The five types are discussed below.

Polling

Polling will suspend execution of certain statements until a desired condition has been reached.

One method of polling is shown in the following QuickBASIC example below. First, the variable to which the reading will be returned is defined as outside the desired range:

```
Temp! = 0.0
```

Next, the conditional expression is set up and evaluated. When the temperature (Temp!) reaches 100.0, processing will move on to the next statements.

```
WHILE Temp! < 100.0
CALL FGREAD("tempinput",NONE,VARSEG(Temp!),VARPTR(Temp!),C.THCU.J,NT)
WEND
```

Finally, the alternate decision path is taken when Temp! reaches 100.0.

```
OutVal! = (Temp! + 312.0) / Temp!
CALL FGWRITE("channel0", VARSEG(OUTVAL!), VARPTR(OUTVAL!),C.VOLTS, NT)
```

Advantages of Polling:

Any programming or KDAC500 statements may be used to acquire the information on which the decision will be made.

Any conditional expression(s) within your program may be used as the conditional expression on which the decision is made.

Any KDAC500 statement(s) or any statement(s) within your program may make up the alternate path.

Disadvantages of Polling:

The speed of the loop in which the condition is tested and the response time of the trigger may be slower than corresponding triggering operations handled exclusively by KDAC500 commands.

Execution of the foreground is suspended until the condition is met, wasting the microprocessor's time.

Foreground Triggering

The purpose of the foreground trigger is to suspend execution of your program, after the KDAC500 foreground trigger command, until a desired condition is true.

Foreground triggering is implemented in two steps. First the KDAC500 trigger command is called. This command's parameter list specifies the condition for which the command will wait. For example, the following example of the TRIGGER command will loop internally, constantly checking digital input values, until the specified channel is "on" (high). The statements which are to be executed when the condition is met are included after the TRIGGER call, as shown in the following example:

```
CALL TRIGGER("digch0",0.0,0.0,ON,C.RAW.FLOAT,NT,"",1)
PRINT "ALARM"
```

Advantages of Foreground Triggering:

The polling rate at which the condition is tested is very fast as it is written entirely in assembly language.

Any program statement or KDAC500 command may follow the trigger command.

Disadvantages of Foreground Triggering:

The information on which the decision will be made is limited to KDAC500 foreground trigger commands, i.e. digital input, analog input and time delay.

Only certain conditional expressions may be implemented as they are limited to the capabilities of the KDAC500 foreground trigger.

Execution of the foreground is suspended until the condition is met, wasting the microprocessor's time.

Singleground Triggering

The purpose of the singleground trigger is to suspend execution of a KDAC500 command that must react immediately to a condition, until that condition is true.

Singleground triggering is implemented in two steps. First, a KDAC500 singleground trigger command is called. This call sets up the information about the trigger (what kind of trigger and what condition). Second, a KDAC500 command which uses a WST parameter (Wait on Singleground Trigger) is issued. The WST command may be separated from the ST (Singleground Trigger) command by any number of program lines (however, no other ST command should intervene).

A section of a QuickBASIC program which implements singleground triggers might be:

```
CALL TRIGGER("anch0",5,6,BETW,C.VOLTS,ST,"S_Trig",1)
.
.
.
CALL ANINQ("speedy",10000,"anch0",0,WST)
```

When the program is run, the ST task will not be activated when the command is first encountered. Instead, the task is set up for execution and control is passed directly to the next command. Only when the WST command is encountered does the ST task begin checking for the specified condition. When this condition is met, the WST task is triggered and the ST task turns itself off. If interrupts are on when an ST task is set up, a warning message will be issued. If interrupts are on when the WST task is called, background processing will be suspended.

There are six KDAC500 commands which have wait-on-singleground-trigger capability:

ANINQ	FGREAD	FGWRITE
HREAD	HWRITE	ANOUTQ

Advantages of Singleground Triggering:

The response time of the WST command to the trigger is relatively fast, a matter of a few milliseconds. (This is especially critical for ANINQ and AOUTQ.).

The speed of the loop in which the condition is tested is very fast as it is written entirely in assembly language.

Disadvantages of Singleground Triggering:

The information on which the decision will be made is limited to KDAC500 singleground trigger commands.

Only certain conditional expressions may be implemented as they are limited to the capabilities of KDAC500 singleground trigger commands.

The statements which respond immediately to the trigger are limited to KDAC500 commands implementing the WST parameter.

Execution of the foreground is suspended until the condition is met, wasting the microprocessor's time.

Hardware Triggering

The purpose of the hardware trigger is to suspend execution of a KDAC500 command, that must react immediately to a condition, until that condition is true.

Hardware triggering is implemented in two steps. First, a KDAC500 hardware trigger command is called. This call sets up the information about the trigger (what kind of trigger and what condition). Second, a KDAC500 command which uses a **WHT** parameter (Wait on Hardware Trigger) is issued. The WHT command may be separated from the hardware trigger command by any number of program lines (however, no other hardware trigger command should intervene).

A section of a QuickBASIC program which implements hardware triggers might be:

```
CALL ANTRIG("trig1",1.5,TRG.LATCH,TRG.ABOVE)
.
.
.
CALL ANINQ'("speedy",100000,"anch0",0,WHT)
```

When the program is run, the trigger task will not be activated when the command is first encountered. Instead, the task is set up for execution and control is passed directly to the next command. Only when the WHT command is encountered does the trigger task begin checking for the specified condition. When this condition is met, the WHT task is triggered and the trigger task turns itself off. If interrupts are on when a hardware trigger is set up, a warning message will be issued. If interrupts are on when the WHT task is called, background processing will be suspended.

There is only one KDAC500 command with wait-on-hardware-trigger capability:

ANINQ

Advantages of Hardware Triggering:

The response time of the WHT command to the trigger is very fast, a matter of a few microseconds. (This is especially critical for ANINQ).

The speed in which the condition is tested is extremely fast as it is implemented in hardware.

Disadvantages of Hardware Triggering:

The information on which the decision will be made is limited to the capabilities of the hardware.

The statements which respond immediately to the trigger are limited to the KDAC500 command ANINQ.

Execution of the foreground is suspended until the condition is met, wasting the microprocessor's time.

Background Triggering

The purpose of background triggering is to watch for a given condition without holding up processing in the foreground. When the condition is met, the task to be triggered is activated and the trigger task becomes inactive.

Background triggering is implemented in two steps. First, a KDAC500 trigger command is called with the trigger mode set to "BT" (background trigger). This sets up information about the trigger (what kind of trigger, what condition). Second, a KDAC500 command which uses a **WBT** (Wait on Background Trigger) parameter is issued. The WBT command may be separated from the **BT** (Background Trigger) command by any number of program lines, but no other BT may intervene.

A section of a program in QuickBASIC which implements background triggers might be:

```
CALL TRIGGER("switch",0.0,0.0,ON,C.RAW.FLOAT,BT,"B_Trig",1)
CALL BGWRITE("outarray","ch0",3,FOREVER,WBT,"BGFunc" )
```

When the program is run, the BT command will not be activated when it is first encountered. The task is set up but not communicated to the background, and control is passed directly to the next command. When the WBT command is encountered, the BT and WBT tasks are sent to the background together, and the BT task will begin checking for the trigger condition as of the next interrupt. It will check for the condition once every interrupt. When the condition is met, the WBT task is triggered and the BT task turns itself off.

There are three KDAC500 commands which have background trigger capabilities:

TRIGGER	BGREAD	BGWRITE

There are six KDAC500 commands which can wait on background trigger:

BGREAD	BGWRITE	KDTIMER
BGGO	BGHALT	

Interrupts need not be on when either the BT or WBT command is called. However, for the BT and WBT tasks to execute, interrupts must be turned on. Interrupts may be turned on at any time: before the BT and WBT commands, between them, or after them.

Advantages of Background Triggering:

The foreground is not held up by polling for a condition, as the condition is tested every interrupt in the background.

The response time of the WBT command to the trigger is exceedingly fast, because the WBT task immediately follows the BT task in the background execution sequence.

Disadvantages of Background Triggering:

The trigger decision can be made only on the basis of information available to KDAC500 commands. However, certain KDAC500 background tasks can be configured to trigger upon completion.

The statements which respond immediately to the trigger are limited to KDAC500 commands implementing the WBT parameter.

BGGO Triggering

The KDAC500 BGGO command provides a means to trigger up to 16 background tasks simultaneously. Because BGGO itself can be triggered on a background trigger or on a foreground trigger, the command provides very powerful capabilities.

BGGO triggering is implemented in two steps. First, from 1 to 16 WGO (Wait on BGGO) commands are set up as background tasks. However, their status is not "on", so that they are not actually executed in the background.

Second, the BGGO command is called. The BGGO may be foreground, in which case it will execute immediately, or it may be a WBT in the background, executing later in the background when triggered by its BT.

If BGGO is executed in the foreground it will turn on all previous WGO tasks immediately. However, it must turn them all on without being interrupted, otherwise some would execute,

others not. Therefore, the BGGO command needs enough time between interrupts to turn on all the pending tasks.

If the BGGO command is called with WBT as its trigger mode, it will execute in the background. In this case, all tasks that BGGO activates will start execution on the interrupt subsequent to the one in which the BGGO executed.

An example of foreground BGGO triggering is:

```
CALL BGREAD("ar1",1000,"temp",2,NONE,1,WGO,"t1")
CALL BGWRITE("ar2","volts1",4,1,WGO,"t2")
CALL BGGO(NT,"")
```

Advantages of BGGO Triggering:

More than one event (task) can be triggered simultaneously.

When BGGO is triggered by a background trigger, it will have all the advantages of a background trigger. If BGGO is triggered by a foreground trigger, it will have all the advantages of that type of trigger.

Disadvantages of BGGO Triggering:

When BGGO itself is triggered by another trigger, it assumes the disadvantages of that trigger.

Engineering Units Conversion

One of the unique features of KDAC500 is its ability to convert raw binary values to engineering units. This allows KDAC500 to express analog data in volts, millivolts, milliamperes, degrees centigrade, percent, or others as the case requires. KDAC500 supports many popular transducer types including various types of thermocouples, resistance temperature detectors (RTD's), strain gages, displacement transducers, and industrial current loop signals.

The automatic conversion of raw values to engineering units is a great advantage since the programmer no longer has to approach analog data in terms of binary values returned by analog to digital conversion. The conversion of these raw values is performed internally by KDAC500 which takes into account the type of transducer, the signal range and accuracy, and the applied signal conditioning. Because KDAC500 maintains a record of the complete current hardware configuration of the system, including the placement of modules and the setting of all switches, the conversion to engineering units takes into account all the hardware attributes of the system.

In this way, DC voltages may be read directly and expressed in volts, with KDAC500 considering the module configuration and all relevant hardware and software selected attributes. Or, a thermocouple may be connected directly to a Series 500 module and the measurement results expressed in degrees centigrade. All the calculation for this conversion (amplification, linearization, and compensation for the temperature of the cold reference junction) is carried out by KDAC500.

The Engineering Units Flag

The conversion of raw values to engineering units is controlled by the EUF (engineering units flag) parameter. This parameter is used by certain KDAC500 commands to indicate a particular conversion algorithm. When a KDAC500 command makes use of the engineering units flag, the number assigned to the EUF parameter will determine what type of conversion is required. The value C.RAW.INT indicates that no conversion will take place and that the value will be in raw A/D counts. Some EUF flags specify a proportional conversion from raw values to voltages, while other EUF flags may specify non-linear conversion algorithms for special devices (thermocouples, for instance).

Direct and Indirect Conversion to Engineering Units

Certain KDAC500 commands support the use of engineering units without making use of the EUF parameter in their own parameter lists. These commands support indirect conversion to engineering units.

One example is the BGREAD command, which samples one or more channels of analog input and stores the measured values in a KDAC500 array. When these values are retrieved from the array with the ARGET command, an engineering units flag may be set in the ARGET parameter list, causing the raw values from the array to be converted and expressed in engi-

neering units. The conversion of the data acquired by BGREAD occurs after the fact, hence, indirectly. Because conversion is not performed in the time-critical background, no time is wasted.

We can illustrate this use of indirect conversion by writing a short sample QuickBASIC program. This program invokes the BGREAD command to take 1000 samples from a type J thermocouple. These samples are acquired in real-time from the background and stored in the array created by BGREAD. When the acquired data is retrieved from the array, ARGET converts the raw values into results expressed in degrees Celsius. These results are then printed on the screen.

```
REM $INCLUDE: 'KDAC500.BI'
CALL KDINIT (BASIC.)
CALL BGREAD("temp",1000,"coldjunc,thermo",1,NONE,1,NT,"task1")
CALL INTON( 100, MIL)
DO
    CALL BGSTATUS( "task1", stat%)
LOOP UNTIL stat% = ST.DONE
CALL INTOFF
FOR i% = 1 to 1000
    CALL ARGET("temp", i%, i%, "thermo", -1, VARSEG(degrees!),_
    VARPTR(degrees!), C.THCU.J)
    PRINT "Temperature at measurement "; i%; "was "; degrees!
NEXT i%
```

In the example above, the KDAC500 BGSTATUS command is used to check the status of the background task accomplished by BGREAD. When BGREAD is done sampling, ARGET is called to retrieve the data from the array "temp". In the parameter list of ARGET, the engineering units flag is assigned the value C.THCU.J, indicating a conversion appropriate for type J thermocouples. Thus, the measurement values are returned in degrees Celsius.

Note that the cold junction reference channel ("coldjunc") must be read separately in the BGREAD command, and must be the first channel read. Thus,"coldjunc" is the first name in the IONAME list. ARGET uses the cold junction reference value to calculate the readings of the other channels in degrees Celsius.

The FGREAD command is an example of a KDAC500 command that supports direct conversion to engineering units. FGREAD is a foreground command used to sample analog input in the foreground.The measurement result is returned as a variable. Depending on the value assigned to the EUF parameter, FGREAD will return the measurement result as a raw value, or express the result in volts or engineering units. Note that the conversion occurs as part of the execution of FGREAD, so no separate conversion command is needed.

If we use FGREAD to take a measurement from the same type J thermocouple as before, a simple QuickBASIC program would be:

```
REM $INCLUDE: 'KDAC500.BI'
CALL KDINIT (BASIC.)
CALL FGREAD("thermo", NONE, VARSEG(degrees!), VARPTR(degrees!),_
C.THCU.J, NT)
PRINT "The temperature is "; degrees!;"C"
```

Note that FGREAD, unlike BGREAD, performs an automatic cold junction reading without having the cold junction reference specified in the parameter list.

The use of the engineering units flag with BGWRITE and FGWRITE is completely analogous to the use of that flag with BGREAD and FGREAD.

BGWRITE, in conjunction with ARPUT, supports indirect conversion to volts. BGWRITE is used to output values from a KDAC500 array to some specified number of output channels. The values can be put into a KDAC500 array using the ARPUT command. If the value C.VOLTS is assigned to the EUF parameter in ARPUT's parameter list, the programmer specifies values in volts and has them converted to raw binary values by ARPUT. These raw values will then be used by BGWRITE to send output to the various output channels specified.

The following QuickBASIC program illustrates how BGWRITE and ARPUT may be used for indirect engineering units conversion:

```
REM $INCLUDE: 'KDAC500.BI'
DIM VOLTS AS SINGLE, I AS INTEGER
CALL KDINIT (BASIC.)
CALL ARMAKE("sawtooth", 250, "chanout")
I = 0
FOR VOLTS = 0 TO 2.49 STEP 0.01
    I=I+1
    CALL ARPUT("sawtooth", I, I, "chanout", -1, VARSEG (VOLTS),_
    VARPTR(VOLTS), C.VOLTS)
NEXT VOLTS
CALL BGWRITE("sawtooth","chanout",1,FOREVER,NT,"ramp")
CALL INTON(10, MIL)
END
```

This program creates a sawtooth voltage output to an analog output channel named "chanout", with an amplitude of 0 to 2.5 volts and a period of 2.5 seconds. Note that the EUF parameter for ARPUT has been assigned the value C.VOLTS, indicating a conversion from volts to raw binary values.

The use of FGWRITE is analogous to the use of FGREAD, however the FGWRITE command performs a conversion from volts directly, as part of its own execution. The same C.VOLTS EUF flag is used to indicated that a conversion from volts to raw binary values is desired.

In addition to input and output commands, several other KDAC500 commands make direct use of the EUF parameter. These commands include GRAPH and GRAPHRT, which allow the user to graph values expressed in volts or engineering units.

Engineering Units Flags

When connecting voltage and current sources and common transducers to the Series 500, certain requirements must be met to ensure correct operation. These requirements are summarized below according to the value of the engineering units flag to which they apply.

NOTE: The syntax of Engineering Units Flags differs slightly for QuickBASIC and the other languages. QuickBASIC EUFs contain periods while C, FORTRAN, and Pascal EUFs contain underscores. If you are a KDAC500/I user converting to KDAC500/B or KDAC500/M note that EUFs must not be bounded by quotation marks. Be sure to use the correct format for your programming language or you will receive errors.

Raw Values

QuickBASIC: EUF = C.RAW.INT or C.RAW.FLOAT
C, FORTRAN, and Pascal: EUF = C_RAW_INT or C_RAW_FLOAT

Raw values are always positive integers ranging from 0 to 4095 for 12-bit converters, 0 to 16383 for 14-bit converters, or 0 to 65535 for 16-bit converters. When using raw values, KDAC500 does not take into account any gain or range information, because the data values are simply proportional to whatever range is being used. C.RAW.INT and C_RAW_INT specify the data in integer format and C.RAW.FLOAT and C_RAW_FLOAT specify the data in single-precision float format.

Voltage Inputs and Outputs

QuickBASIC: EUF = C.VOLTS, C.MILVLT, or C.MICVLT
C, FORTRAN, and Pascal: EUF = C_VOLTS, C_MILVLT, or C_MICVLT

With these EUF parameters, the results of an A/D conversion can be read directly in voltage. Similarly, voltage values can be written to analog output channels during D/A conversion. The system automatically accounts for converter ranges, amplifications, software-selected signal conditioning attributes, and converter resolutions.

Percent Full Scale

QuickBASIC: EUF = C.PERCENT
C, FORTRAN, and Pascal: EUF = C_PERCENT

KDAC500 allows data to be returned as a percentage of full scale reading ranging from –100 to +100 percent.

Thermocouples

QuickBASIC: EUF = C.THCU.J, C.THCU.K, C.THCU.S, C.THCU.T, C.THCU.E, C.THCU.B, or C.THCU.R
C, FORTRAN, and Pascal: EUF = C_THCU_J, C_THCU_K, C_THU_S, C_THCU_T, C_THCU_E, C_THCU_B, or C_THCU_R

Seven popular thermocouple types (J, K, S, T, E, B, and R) are directly supported by KDAC500. Thermocouples should be connected to the AIM7. If the hardware configuration of the system indicates that a thermocouple is connected to a module which does not support thermocouples, an error will be issued. The system automatically linearizes the output volt-

age using polynomial curve-fitting, while compensating for the cold junction temperature. The result is returned in degrees Celsius.

While the A/D converter (AMM2, AMM1A) can be set to any of the available ranges, the voltage output range of some thermocouples includes negative values. If an A/D unipolar voltage range is set, an error may result if the thermocouple output goes negative.

Resistance Temperature Detectors (RTD's)

QuickBASIC: EUF = C.RTD3175 or C.RTD3212
C, FORTRAN, and Pascal: EUF = C_RTD3175 or C_RTD3212

Two types of popular RTD's can be used with the KDAC500 engineering units capability. RTD's with an alpha of 0.00385 and RTDs with an alpha of 0.00392 are specified by the two values for the RTD flag. The result is returned or specified in degrees Celsius.

The system software uses a polynomial linearization technique appropriate for the device. RTD's must be used only with the AIM6 module; the use of any other module will result in an error message.

Current excitation is provided by the AIM6 module directly, and both 2-wire and 3-wire RTDs can be interfaced to the module.

For correct operation, the AIM6 should be set to the RTD mode and 50 should be chosen for the special AIM6 gain. The A/D converter may be set to any bipolar range that accommodates the transducer.

If the temperature being measured exceeds the specified range for RTD'S (–200 degrees C to +700 degrees C), a warning message will be given, and the temperature returned will be the last temperature within the range.

Strain Gages and Load Cells - AIM8

QuickBASIC: EUF = C.AIM8.C or C.AIM8.D
C, FORTRAN, and Pascal: EUF = C_AIM8_C or C_AIM8_D

KDAC500 is capable of returning values directly in measurement units when using an AIM8. If a calibration factor has been entered into the CONFIG.TBL via CONFIG.EXE, specifying C.AIM8.D or C_AIM8_D as the EUF parameter will return load values in terms of the measuring units (grams for example). If on the other hand you wish to calibrate the AIM8 from within your program, specifying C.AIM8.C or C_AIM8_C as the EUF will calibrate the AIM8 to a known force. The value of the known force can be entered into the CONFIG.TBL via the MODE function in CONFIG. Once the AIM8 has been calibrated, specifying C.AIM8.D or C_AIM8_D as the EUF parameter will return values in units of the calibrating force.

LVDT/RVDT

QuickBASIC: EUF = C.AIM9.D
C, FORTRAN, and Pascal: EUF = C_AIM9_D

KDAC500 is capable of returning values directly in units of measure corresponding to the calibration factor stored in the configuration table via CONFIG.EXE.

Semiconductor Temperature Sensors AD590/AC2626

QuickBASIC: EUF = C.AD590
C, FORTRAN, and Pascal: EUF = C_AD590

The popular and versatile semiconductor temperature sensors Analog Devices AD590 and Analog Devices AC2626 are supported by KDAC500 with readings in degrees Celsius. These transducers can be connected to the AMM1A, AMM2, AIM3, AIM5 or AIM6 modules. Using the AD590/AC2626 EUF with other modules will result in a warning error message. A 1000 ohm resistor must be used with the AMM modules, AIM3, and AIM5. To provide for individual channel calibration, a 950 ohm resistor can be used in series with a 100 ohm potentiometer, allowing a 5% calibration adjustment. With the AIM6, a 210 ohm resistor must be used. KDAC500 assumes that these resistor values are used in order to convert the transducer's current output signal into degrees Celsius. Excitation can be provided by the data acquisition hardware. With the AIM6, the +10 volt excitation source can be used with AD590's. The level of the excitation voltage will have no effect on calibration with these devices.

Current Loop Inputs

QuickBASIC: EUF = C.MA420 (4-20mA only), or C.MILLIA
C, FORTRAN, and Pascal: EUF = C_MA420 (4-20mA only), or C_MILLIA

Industrial control signals with standard 4-20 mA outputs can be sensed by KDAC500, with the result expressed in milliamps ranging from 4 to 20. This conversion is supported for backward compatibility only. You may use the C.MILLIA or C_MILLIA conversion for 4-20mA measurements, which has the added advantage that the external resistor value can be defined from within the CONFIG program on a channel by channel basis. For this type of conversion the following resistor values are assumed.

Typically, a 250Ω resistor will be used with current inputs. However, when it is essential to maintain a high compliance voltage (the voltage in the loop), it is necessary to minimize the value of the shunt resistor, thereby lowering the voltage across the shunt resistor. A 250Ω resistor will drop five volts at full scale. However, a 25Ω resistor will drop only 0.5 volt at full scale in the loop. When the current transmitter is a 2-wire device that is powered from the loop, it is important to observe the minimum voltage required to properly power the transmitter.

Table 2. Shunt Resistor Values with Current Loop Inputs

Resistor Value	Total Gain	Notes
250 ohms	x1	default
25 ohms	x10	AIM3, AIM4 only
2.5 ohms	x100	AIM3, AIM4 only

Pulse Input

QuickBASIC: EUF = C.RAW.INT, C.RAW.FLOAT, C.COUNT
C, FORTRAN, and Pascal: EUF = C_RAW_INT, C_RAW_FLOAT, C_COUNT

Two types of pulse input modules are supported by KDAC500. If you are using a PIM1 or PIM2 module (16-bit mode) for event counting, use C.RAW.INT, C.RAW.FLOAT, C_RAW_INT, or C_RAW_FLOAT to retrieve data. If you are using a PIM2 in 32-bit mode, use C.COUNT or C_COUNT to retrieve data.

Frequency

QuickBASIC: EUF = C.FREQ
C, FORTRAN, and Pascal: EUF = C_FREQ

This Engineering Unit Flag allows data collected with the PIM1 module in frequency mode to be converted directly to frequency values expressed as Hertz.

CHAPTER 3
KDAC500 Functions

Brief Listing of KDAC500 Functions

General Considerations

Command Format

Brief Listing of KDAC500 Functions

FOREGROUND FUNCTIONS

aninq	– fast analog read
anoutq	– fast analog write
fgread	– foreground read
fgwrite	– foreground write
gethandle	– return a pointer to an ioname
hread	– foreground read with handles
hwrite	– foreground write with handles
kdclock	– read the real-time clock
kdinit	– initialize the system
kdpause	– real-time delay
kdtimerrd	– read the software timer values
kdwarn	– turn warning messages on or off
softinit	– initialize system software only. Hardware, any running program, and any existing data arrays are not disturbed.
antrig	– setup a hardware trigger

BACKGROUND FUNCTIONS

bgclear	– clear all background tasks
bggo	– start a background task waiting for GO
bghalt	– stop a background task
bgread	– background read
bgstatus	– get the status of a background task
bgtime	– compute time of background task
bgwrite	– background write
intoff	– stop interrupts
inton	– turn interrupts on
kdtimer	– set up software timers
trigger	– set up a background trigger

ARRAY MANAGEMENT

aravail	– get available array space
ardel	– delete an array
arget	– get data from an array
arptr	– return the segment and offset of a KDAC500 array.
arput	– put data into an array
arlabel	– put a description on an array
arlastp	– get the last point accessed
arload	– load an array from disk
armake	– create an array
arstatus	– get information about an array
arsave	– save an array to disk

STEPPER MOTOR FUNCTIONS

stpabsloc	– define motors position
stpmaxsp	– set stepper motor's max speed
stpmoveabs	– move to an absolute position
stpmoverel	– move to a relative position
stpreset	– reset a stepper motor
stpset	– configure stepper controller
stpspeed	– sets stepper motor speed
stpstatus	– get stepper motor status

GRAPHICS FUNCTIONS

graph	– graph data after acquisition
graphrt	– graph data during acquisition
grlabel	– put a label on a graph
hgraphrt	– high-res graphing during acquisition

USER-DEFINED ERROR HANDLING

glerror	– KDAC500 error handler
GetKDACMsg	– return error message text
SetErrHandler	– set user-defined error handler (Turbo Pascal and Quick Pascal only)

General Considerations

As we have mentioned earlier, KDAC500 commands are given specific information and instructions by means of command parameters. Most KDAC500 commands expect to be given certain parameters when the command is issued. These parameters are included in a KDAC500 parameter list, which is enclosed by parentheses.

All parameters are identified by parameter names, short mnemonic labels that indicate to the programmer the function of that particular parameter. In this manual, we use the parameter name to refer to the parameter itself, as, for example , when we speak of the ARRNM parameter (meaning the parameter that specifies the ARRay NaMe). It is a matter of convenience to use the parameter name in this way, as it allows us to describe in a general way the properties of a given parameter.

KDAC500 parameters are subject to certain conventions and restrictions, summarized below:

1. KDAC500 input parameters may be specified by variables or by constants. KDAC500 output parameters must be variables or arrays which must be passed by reference, i.e. a pointer. In QuickBASIC this is achieved by the QuickBASIC Functions VARSEG() and VARPTR() which can be embedded in the call statements to the KDAC functions. For more information on VARSEG() or VARPTR() please refer to your QuickBASIC Reference Manual. In FORTRAN, the LOCFAR Function is used to pass parameters by reference. Please refer to your FORTRAN users manual for more information.
2. Parameters must be of a specific type, indicated in the description of the parameter name. There are six parameter types used by KDAC500.

   string — variables of type string differ from language to language. It is imperative that the correct language type is specified when the KDAC500 command KDINIT or SOFTINIT is called. QuickBASIC handles strings by way of a string descriptor, C strings are terminated by a null byte, and Pascal strings store the length of a string in the first byte of the string. If the wrong language is specified, KDAC500 will have no way of determining what the string's value is. FORTRAN must use C-style strings or KDAC500 will not be able to handle the strings (example: 'STRING'C)

   integer — variables of type integer are 16–bits long. They may be signed or unsigned depending on usage. This corresponds to the FORTRAN type INTEGER*2. The default integer type in FORTRAN is INTEGER*4 so INTEGER without the size should not be used. See the description under Special Considerations for more information.

   long — variables of type long are 32 bit integers. The FORTRAN type is INTEGER*4. They may signed or unsigned depending on usage.

   single precision real — variables of this type conform to the IEEE standard for single precision floating point numbers. The Pascal REAL type is **not** supported. Any single precision real variables must be of type SINGLE.

   double precision real — variables of this type conform to the IEEE standard for double precision floating point numbers.

   pointer — a pointer is a 32 bit far address stored as segment:offset. All the supported languages have facilities for passing pointers. In QuickBASIC, the VARPTR and VAR-

SEG commands are used. In Pascal and C the address operators (@ and & respectively) are used. In FORTRAN the LOCFAR command is used.

3. Depending on the type of the variable or constant, a certain format must be observed. The following list summarizes the correct form for each parameter type. Exceptions to these rules are described in the individual reference sections for each parameter.

STRING: A string may contain between 1 and 255 significant characters. This limitation is imposed by KDAC500, not necessarily the language your programs are written in. QuickBASIC, for instance, can handle strings up to 32767 characters. Since strings only reference IONAMEs and arrays, this limitation should never be a problem. All FORTRAN strings must be defined as C-style strings. This is accomplished by following the string with the letter C. Refer to the Microsoft FORTRAN manual for more information.

Special considerations apply to KDAC500 parameters, depending on their usage:

A. NAMES: The ION (IOName) and BFN (Background Function Name) parameters. Only the first 8 characters are significant.

   The ARRNM (ARRay NaMe) parameter has a minimum length of 1 character, and a maximum of 255 characters, although only the first 8 characters are significant.

   KDAC500 names may not contain any delimiter characters (comma, blank, tab, right parenthesis, double quote). With the exception of array names, KDAC500 names may not contain symbols ($,%,!, &, @, #).

B. LISTS: A KDAC500 list is a string consisting of one or more names (ioname list, background function name list) separated by delimiters. Valid delimiters for KDAC500 name lists include commas, spaces, tabs, or any combination of these characters.

C. INTEGERS:

   KDAC500 integer arrays are treated as arrays of unsigned integers (not signed integers). Thus, although the range for standard integer values are –32768 to 32767 (depending on the application), the range of values stored in KDAC500 integer arrays is 0 to 65535. In Pascal the unsigned integer type is WORD. QuickBASIC and FORTRAN do not support unsigned integers, and it is the job of the programmer to recognize and handle this limitation.

   The command ARPUT (with C_RAW_INT as the engineering units flag) will translate all integers to unsigned integers for word arrays, and similarly, ARGET (with C_RAW_INT as the engineering units flag) will only return unsigned values from these arrays. Since the primary use of KDAC500 word arrays is the storage of raw values for analog input and output, the unsigned interpretation is appropriate.

D. REALS: Reals refer to single precision real numbers, not to the Pascal REAL type.

E. DOUBLE-PRECISION REALS: Double-precision reals refer to floating point number values stored in 8 bytes of memory (real number values are stored in 4 bytes).

It is crucial to know exactly which parameter should fall where in the parameter list. Memorizing the parameter list for many KDAC500 commands would be a considerable inconven-

ience, therefore this manual uses a notation which allows the programmer to determine at a glance the parameter list for any KDAC500 command.

For Quick Basic Programmers: You must include the KDAC500 interface header at the top of your program file prior to any other BASIC commands. The file KDAC500.BI contains all KDAC function and constant declarations and is included by the following statement:

```
'$INCLUDE: 'KDAC500.BI' or REM $INCLUDE: 'KDAC500.BI'
```

For C Programmers: You must include the KDAC500 header file (KDAC500.H) at the top of your C program. This file contains all KDAC function and constant declarations which are necessary to correctly use KDAC500. KDAC500.H is included by the statement:

```
#include "kdac500.h"
```

For Pascal Programmers: To correctly interface to KDAC500 functions you must include this statement at the beginning of your program:

```
uses KDAC500;
```

The files KDAC500.TPU for Turbo Pascal or KDAC500.QPU for Quick Pascal provides all the necessary definitions for KDAC500 procedures and constants.

For FORTRAN Programmers: There are two header files which you must include to correctly interface to the KDAC library. The first is the INTERFACE file (KDAC500.FI). This file must be included at the top of each FORTRAN module. It provides interface information to the compiler for each KDAC500 function. The second file that must be referenced is the DEFINITION file (KDAC500.FD). This file provides definitions for all the constants needed by the KDAC function. KDAC500.FD must be included by the main FORTRAN program as well as any subroutines that require access to the KDAC500 library functions. Use the FORTRAN include statement to include the contents of these header files in your program.

```
INCLUDE 'KDAC500.FI'
INCLUDE 'KDAC500.FD'
```

Each command will be described using the format as shown on the next page.

COMMAND FORMAT

Purpose A description of the command and its intended use.

Language Syntax

BASIC: call command(parameter 1, parameter 2,...)

C: command(parameter 1, parameter 2,...);

FORTRAN: call command (parameter 1, parameter 2, ...)

Pascal: command(parameter 1, parameter 2,...);

Parameter 1 description

Parameter 2 description

Notes Additional information of a general nature

The text of the parameter describes the parameters function. ARRNM means "Array name", IONL means "I/O name list", etc. We use this kind of notation because many of the parameter types are repeated in several different KDAC500 functions. For example, all of the array management commands require the name of the array to be supplied. You will note that they all require a string to specify this name, all the trigger modes are integers, etc. The type of the parameter is specified under its description. For more specific language interface information refer to the header files for the language you are interested in.

KDAC500.h	C header file
KDAC500.bi	QuickBASIC header file
KDAC500.doc	Pascal Unit documentation
KDAC500.FI	FORTRAN Function Interface File
KDAC500.FD	FORTRAN Function and constant Definition header File

Pascal in the language syntax above describes both the Turbo Pascal and Microsoft Quick Pascal calling sequence.

CHAPTER 4
KDAC500 Commands

ANINQ

Purpose

ANINQ is a "quick" version of BGREAD, providing highspeed sampling of up to 64 channels of analog input. The ANINQ command is designed for singleground execution only, and for this reason, is able to achieve very high sampling rates for a single channel (50 KHz with an AT-class or 386 computer). When executed, ANINQ causes measurements to be taken at a specified interval from a specified number of channels, and creates a KDAC500 array in which to store the data from these measurements.

ANINQ is best used with the AMM1A, AMM2, AIM2, and AIM3 analog input modules. Generally, the AIM4, AIM5, AIM6, AIM7, AIM8, and AIM9 modules require longer settling times, especially when higher gains are programmed. Thus, the readings returned from these modules by ANINQ may be incorrect.

Language Syntax

QuickBASIC: call aninq (arrynm, numsamp, ionl, sintv, tm)

C: aninq(arrynm, num_samp, ionl, sintv, tm);

FORTRAN: call aninq (arrynm, numsamp, ionl, sintv, tm)

Pascal: aninq(arrynm, num_samp, ionl, sintv, tm);

ARRYNM — (string) The ARRYNM parameter is a standard KDAC500 array name given to the array created by the execution of the ANINQ command. KDAC500 arrays are accessible through KDAC500 array management commands only. (See ARGET).

NUMSAMP — (long) The NUMber of SAMPles parameter indicates the depth of the array created by ANINQ. This value corresponds to the number of times each channel is sampled when ANINQ is executed.

IONL — (string) The IOName List parameter may include from 1 to 64 IONAMES.

SINTV — (integer) By assigning a value to the SINTV parameter, the user specifies the interval between samples as a multiple of 10 microseconds. The SINTV parameter has legal values from 0-6400. Not all these values are appropriate, however, and the user must determine whether a specified interval will allow enough time between A/D conversions. The required interval depends on the class of computer which is used (AT-class vs PC-class).

SPECIAL USE OF ANINQ: SINTV may be given the value 0, and ANINQ will sample as quickly as possible for as many channels as are included in the IONAME list.

TM (integer) The Trigger Mode parameter allows ANINQ to be triggered by the execution of some other KDAC500 command. The only valid values for this parameter are:

WST	Wait for Singleground Trigger. When TM is given this value the ANINQ function will be associated with the most recent KDAC500 command with an ST in its parameter list.
WHT	Wait for Hardware Trigger. When TM is given this value the ANINQ function will be associated with the most recent KDAC500 hardware trigger command.
NT	No Trigger.

The singleground and hardware triggers for ANINQ are very exact. There is very little delay between the recognition of the trigger condition and the execution of the high–speed ANINQ command.

Notes

ANINQ should be used to access the AMM1A, AMM2, AIM2, and AIM3 modules unless you are sure that settling times are of no consequence with other modules. When used with the AIM3, the switch-selectable gains must be set to x1. If the external mode is selected and a gain-programming resistor installed on these modules, ANINQ may not be able to compensate for the settling time of the components.

High-Speed Acquisition Mode with the AMM1A and AMM2

The ANINQ command can operate the AMM1A and AMM2 module in a high-speed "auto–acquire" mode at an aggregate throughput of up to 50 KHz. Auto-acquire applies to single or multiple channels. For multiple channels, the per-channel scan rate equals 50 KHz divided by the number of channels.

The analog input modules AIM2 and AIM3 can also provide up to 50 KHz throughput when these modules are used in a system containing an AMM2 or AMM1A.

To operate the AMM1A and AMM2 in auto-acquire mode, you must adhere to the following requirements:

- The analog input channels sampled by ANINQ may be on an AMM2, AMM1A, AIM2, or AIM3.
- All the channels sampled by the ANINQ command must be on one module.
- If the input channels are on an AIM3, the AIM3 must be set to x1 gain.
- If the input channels are on an AMM2 or AMM1A, the input filter must be set to 100 KHz.

If any of these conditions cannot be met, the speed of the ANINQ command will revert to the speed of a normal BGREAD command. Under these circumstances, it is better to use BGREAD to take advantage of background/foreground mode.

Consult the following chart for approximate ANINQ speeds for various computer and input configurations.

	Single Channel	**Multi-channel on 1 card**	**Multi-channel on >1 card**
6MHz AT	50KHz	50KHz	6KHz
8MHz PC	40KHz	30KHz	4KHz
4.77 MHz PC	33KHz	25KHz	3KHz

CONDITIONS:

1. All channels requested must be on the same card.
2. Module and channels must be capable of listed speed; filters and higher gains may slow acquistion.
3. Selected gains must allow for listed speeds. Higher gains require longer settling times.

ANOUTQ

Purpose

ANOUTQ is a "quick" version of BGWRITE, providing highspeed output of up to 64 analog channels. The ANOUTQ command is designed for singleground execution only, and for this reason, is able to achieve very high output rates for a single channel (>50 KHz). When executed, ANOUTQ causes output to be generated at a specified interval from a specified number of channels, and reads from a KDAC500 array to get the data for these outputs.

ANOUTQ can be used with any of the AOM modules.

Language Syntax

QuickBASIC: call anoutq (arrynm, ionl, sintv, cycle, tm)

C: anoutq(arrynm, ionl, sintv, cycle, tm);

FORTRAN: call anoutq (arrynm, ionl, sintv, cycle, tm)

Pascal: anoutq(arrynm, ionl, sintv, cycle, tm);

ARRYNM — (string) The ARRYNM parameter is a standard KDAC500 array name given to the array created by the execution of a BGREAD, ANINQ, or ARMAKE command. KDAC500 arrays are accessible through KDAC500 array management commands only. (See ARGET/ARPUT).

IONL — (string) The IOName List parameter may include from 1 to 64 IONAMES. The IONAMES are mapped 1 to 1 to the widths in ARRYNM. THE IONAMES SPECIFIED DO NOT HAVE TO BE THE IONAMES THAT CREATED THE ARRAY. In this way data acquired with BGREAD or ANINQ can be directly output from the same array. (AOM5 data must be converted.)

SINTV — (integer) By assigning a value to the SINTV parameter, the user specifies the interval between samples as a multiple of 10 microseconds. The SINTV parameter has legal values from 0-6400. Not all these values are appropriate, however, and the user must determine whether a specified interval will allow enough time between D/A conversions. The required interval depends on the class of computer which is used (AT-class vs PC-class).

SPECIAL USE OF ANOUTQ: SINTV may be given the value 0, and ANOUTQ will output as quickly as possible for as many channels as are included in the IONAME list.

CYCLE (unsigned integer) The cycle parameter will allow multiple output of the same KDAC500 array. Legal values are 0 to 65535. Specifying 0 for the cycle will cause the same array to be output 65536 times. To output the data only one time cycle should be set to 1.

NOTE: When cycle is any value other than 1, a maximum of 32767 data points per channel can be output. Only when cycle is 1 can the entire KDAC500 array be output.

TM (integer) The Trigger Mode parameter allows ANOUTQ to be triggered by the execution of some other KDAC500 command. The only valid values for this parameter are:

WST Wait for Singleground Trigger. When TM is given this value the ANOUTQ function will be associated with the most recent KDAC500 command with an ST in its parameter list.

NT No Trigger.

The singleground trigger for ANOUTQ is very exact. There is very little delay between the recognition of the trigger condition and the execution of the high-speed ANOUTQ command.

ANTRIG

Purpose

ANTRIG is the hardware trigger setup command, providing hardware triggering for the ANINQ command.

Language Syntax

QuickBASIC: call antrig (ion, threshold, action, mode)

C: antrig(ion, threshold, action, mode);

FORTRAN: call antrig (ion, threshold, action, mode)

Pascal: antrig(ion, threshold, action, mode);

ION

(string) The IOName parameter is the name of the channel which is to be used as the trigger input signal. This may be the name of any analog input channel provided the associate ANINQ command is a single channel acquisition and the IOName is the same in both the ANTRIG and the ANINQ commands, or the IOName that of the external trigger channel assigned to the TRG1 module in the slot designated to this ANTRIG function call.

The ANTRIG command may be issued once for each TRG1 module in the system, thus allowing a maximum of two hardware triggers to be set. The first ANTRIG command is associated with the TRG1 module in slot 2 and the second to the TRG1 module in slot 3.

THRESHOLD

(double) The threshold parameter indicates at what voltage level the trigger condition will become true. Valid thresholds are between –10 and +10 volts.

ACTION

(integer) The action parameter determines whether the trigger condition must be met after each sample or only once.

Valid trigger actions are:

TRG_LATCH — once the trigger condition becomes true, it stays true until all data is collected for the associated ANINQ.

TRG_FOLLOW — the trigger condition must be met for each sample acquired by the associated ANINQ.

LANGUAGE	ACTION PARAMETERS
QuickBASIC	TRG.LATCH TRG.FOLLOW
C/Pascal & FORTRAN	TRG_LATCH TRG_FOLLOW

MODE (integer) The mode parameter is a bit mask which establishes filtering, coupling, and level sensitivity. This mask can be achieved by logically oring one selection from each of the three following categories together.

LANGUAGE	FILTER	COUPLING	SENSITIVITY
QuickBASIC	TRG.1M TRG.300K TRG.100K TRG.30K TRG.10K TRG.3K TRG.1K TRG.300	TRG.AC TRG.DC	TRG.ABOVE TRG.BELOW
C/Pascal & FORTRAN	TRG_1M TRG_300K TRG_100K TRG_30K TRG_10K TRG_3K TRG_1K TRG_300	TRG_AC TRG_DC	TRG_ABOVE TRG_BELOW

Notes

The ANTRIG command will have no effect on acquisition of the ANINQ command if all conditions for HIGH SPEED ACQUISITION are not satisfied. (See ANINQ for HIGH SPEED ACQUISITION considerations.)

For accurate triggering, hardware jumpers must be set properly, see hardware documentation for specifics of jumper positions. To abort the ANINQ command while it is waiting on a hardware trigger condition to occur, press any key on the keyboard.

ARAVAIL

Purpose

The ARAVAIL function returns the size of the largest free block of contiguous memory that currently exists in the KDAC500 memory-management system. The size will be expressed as a number of bytes.

Language Syntax

QuickBASIC: call aravail(size)

C: aravail(&size);

FORTRAN: call aravail(size)

Pascal: aravail(size);

SIZE (long) The SIZE of the largest free block of contiguous memory will be returned to the SIZE parameter by the ARAVAIL command. The value assigned to SIZE by ARAVAIL will be given as some number of bytes.

Notes

For multi-channel acquisitions, the largest number of samples that will fit an available block of memory can be calculated as:

$$\text{Samples} = \frac{\text{SIZE}-(\text{No. of Channels} - 1) \times 64)}{\text{No. of channels} \times \text{bytes per sample}}$$

The number "samples" corresponds to the number of scans that can be made through the complete channel list.

ARDEL

Purpose

The ARDEL command deletes (de-allocates) a KDAC500 array from KDAC500 memory. ARDEL allows the user to free blocks of memory previously occupied by KDAC500 arrays.

Language Syntax

QuickBASIC: call ardel (arrnm)

C: ardel(arrnm);

FORTRAN: call ardel (arrnm)

Pascal: ardel(arrnm);

ARRNM (string) The ARRay NaMe parameter identifies the KDAC500 array to be deleted from system memory. The ARRNM parameter must refer to a KDAC500 array created by the ARMAKE command or by a KDAC500 input routine.

If the user tries to delete an array which does not exist, KDAC500 will return an error message.

Notes

The ARDEL command is used for de-allocating KDAC500 arrays only. ARDEL cannot be used to delete arrays within your programs.

There is a link between KDAC500 arrays and the background that will cause the background task that is acquiring data into an array to be cleared if the corresponding array is deleted. This fact can be used to clear specific groups of tasks from the background. Keep in mind, however, that if a task is removed from the background, the relative timing of the tasks may change.

ARGET

Purpose

The ARGET command allows you to quickly transfer blocks of data from a KDAC500 array to a local array. ARGET internally loops through the algorithm that converts KDAC500 data values into engineering units. The data blocks must be expressed as raw binary values, or in terms of engineering units.

Language Syntax

QuickBASIC: call arget(arrnm,dep1,dep2,ion,wid,varseg(extarry(0)), varptr(extarry(0)),euf)

C: arget(arrnm,dep1,dep2,ion,wid,&extarry[0],euf);

FORTRAN: call arget(arrnm,dep1,dep2,ion,wid, LOCFAR (extarry(1)),euf)

Pascal: arget(arrnm,dep1,dep2,ion,wid, @extarry[0], euf);

ARRNM — (string) The ARRay NaMe parameter identifies the array from which ARGET gets the specified data. The array name must refer to a KDAC500 array, created previously by ARMAKE or by a KDAC500 input command (ANINQ or BGREAD).

DEP1,DEP2 — (long) The DEPth1 and DEPth2 parameters delimit a subset of the named KDAC500 array. ARGET will retrieve values from the indicated area. The subset extends from the lower depth index (given by DEP1) to the higher index (DEP2). Setting DEP1 to a value higher than DEP2 will result in an error.

ION — (string) The IOName parameter is used to get a specific channel's data. The ION parameter refers to the IOName that was used when the array was created.

WID — (integer) The width index of a specific channel's data. Set ION to be an empty string ("") to use the WID parameter.

EXTARRY — (Pointer to a float, integer, or long) When ARGET is executed, the block of values in the specified KDAC500 array will be transferred to the array designated by EXTARRAY. The array must be of type integer if an EUF flag of C_RAW_INT is specified, type long if C_COUNT is specified and type float otherwise. Depending on the value assigned to the engineering units flag, the values will be returned as raw binary, volts, or engineering units.

NOTE: Because the Engineering Units Flag determines the type of returned data, there is no way for the compiler to insure that the type of the array specified will match the type of the returned data. Therefore it is up to the user to make sure the EXTARRY is of the correct type. If it is the wrong type, at best the data will be incorrect and at worst the system may hang.

EUF — (integer) The engineering units conversion flag is used to specify that raw values be converted to volts, frequency, or engineering units. See the EUF section for a complete description of the engineering units flag.

ARPTR

Purpose

ARPTR returns the segment and offset address of the first byte of a KDAC500 array, making it possible to access the array directly instead of through ARGET and ARPUT. ARPTR is useful for word and byte arrays. Bit arrays require more complex calculations, and are better left to the ARGET and ARPUT commands.

Language Syntax

QuickBASIC: call arptr (arrnm, ioname, wid, arrptr)

C: arptr (arrnm, ioname,, wid, &arrptr);

FORTRAN: call arptr (arrnm, ioname, wid, arrptr)

Pascal: arptr (arrnm, ioname, wid, @arrptr);

ARRNM — (string) ARRay NaMe is a standard KDAC500 array name whose address is to be returned by ARPTR. The array must have been previously created by ARMAKE, BGREAD, or ANINQ.

IONAME — (string) The IONAME parameter identifies a specific channel's data according to the IONAME that was used when the array was created. If IONAME is used, WID must be set to –1.

WID — (integer) The WID parameter indicates the index into a KDAC500 Array of a particular channel's data. WID starts at 1 and can be set to a maximum of the number of channels in the array. If WID is used, IONAME must be set to a null string ("") ("C in FORTRAN).

ARRPTR — (far pointer, long) The ARRay PoinTeR holds the far address to the KDAC500 array. The result returned will occupy 4 bytes.

Notes

The ARRPTR parameter should be defined and used as follows:

QuickBASIC:

```
DIM ARRPTR AS LONG
SEGMENT = ARRPTR / 65536
OFFSET = ARRPTR AND 65535
DEF SEG = SEGMENT
VALUE = PEEK (OFFSET)
```

C:

```
unsigned int (or char) far *arrptr;
value = *arrptr;
    or
value = arrptr[i]; where i is an index variable
```

Pascal:

```
arrptr : ^word (or ^byte)
value := arrptr^;
```

FORTRAN:

Does not support pointers or peek/poke operations.

ARPUT

Purpose

The ARPUT command allows you to quickly transfer blocks of data from a local array to a KDAC500 array. ARPUT internally loops through the algorithm that converts data values into a format usable by KDAC500. The data blocks may be expressed as raw binary values or in terms of engineering units.

Language Syntax

QuickBASIC:

```
call arput(arrnm,dep1,dep2,ion,wid,varseg(extarry(0)),
varptr(extarry(0)),euf)
```

C:

```
arput(arrnm,dep1,dep2,ion,wid,&extarry[0],euf);
```

FORTRAN:

```
call arput(arrnm,dep1,dep2,ion,wid,LOCFAR(exarry(1)),euf)
```

Pascal:

```
arput (arrnm, dep1, dep2, ion, wid, @extarry[0], euf);
```

ARRNM — (string) The ARRay NaMe parameter identifies the array to which ARPUT puts the specified data. The array name must refer to a KDAC500 array, created previously by ARMAKE or by a KDAC500 input command (ANINQ or BGREAD).

DEP1,DEP2 — (long) The DEPth1 and DEPth2 parameters delimit a subset of the named KDAC500 array. ARPUT will store values in the indicated area. The subset extends from the lower depth index (given by DEP1) to the higher index (DEP2). Setting DEP1 to a value higher than DEP2 will result in an error.

ION — (string) The IONname parameter is used to put a specific channel's data. The ION parameter refers to the IONname that was used when the array was created.

WID — (integer) The width index of a specific channel's data. Set ION to be an empty string ("") to use the WID parameter.

EXTARRY — (Pointer to a float, integer, or long) When ARPUT is executed, the block of values in the EXTARRY will be transferred to the specified KDAC500 array. The array must be of type integer if an EUF flag of C_RAW_INT is specified, type long if C_COUNT is specified and type float otherwise. Depending on the value assigned to the engineering units flag, the values can be sent as raw binary, volts, or engineering units.

NOTE: Because the Engineering Units Flag determines the type of expected data, there is no way for the compiler to insure that the type of the array specified will match the type of the data being sent. Therefore it is up to the user to make sure the EXTARRY is of the correct type. If it is the wrong type the data stored in the KDAC500 array will be incorrect.

EUF — (integer) The engineering units conversion flag is used to specify that raw values be converted to volts, frequency, or engineering units. See the EUF section for a complete description of the engineering units flag.

ARLABEL

Purpose

The ARLABEL command associates a descriptive string with a specified KDAC500 array. This string will be saved with the array (ARSAVE) and loaded with the array (ARLOAD). ARLABEL is very useful for storing miscellaneous information about an array with the array itself.

Language Syntax

QuickBASIC: call arlabel (arrnm,labl)

C: arlabel(arrnm, labl);

FORTRAN: call arlabel (arrnm,labl)

Pascal: arlabel(arrnm, labl);

ARRNM — (string) The ARRay NaMe parameter indicates the array which will receive the label when the ARLABEL command is executed. The array name must refer to a KDAC500 array that currently resides in memory.

LABL — (string) The LABeL parameter should be assigned a descriptive string, identifying the array and providing any other miscellaneous information that the user would like included. The string assigned to the LABL parameter will be stored with the array when the array is saved with the ARSAVE command. The label may be 1-255 characters in length.

ARLASTP

Purpose

The ARLASTP command returns the depth index of the last point accessed by a KDAC500 background routine. Note that ARLASTP does not return the data value of this point, but only its depth in the array.

If ARLASTP is executed on an array that has not been used by a KDAC500 background I/O routine (BGREAD, ANINQ, BGWRITE), the depth index returned (the "last point") will be 0. Only the routines mentioned above will set the pointer that is used to the depth of the last point accessed. (If the array has been filled by ANINQ, however, ARLASTP will return the index of the maximum depth of the array, since ANINQ must always finish before any other function is called.)

Language Syntax

QuickBASIC: call arlastp (arrnm, lp)

C: arlastp(arrnm, &lp);

FORTRAN: call arlastp (arrnm, lp)

Pascal: arlastp(arrnm, lp);

ARRNM (string) The ARRay NaMe parameter is given the name of the KDAC500 array from which ARLASTP finds the depth of the last point accessed. The array name must refer to a KDAC500 array that currently resides in memory.

LP (long) ARLASTP will find the depth index of the last point in the array accessed by a KDAC500 background routine and return the depth to the LP parameter. Note that LP is not assigned the value of the last data point (that could be found easily with ARGET), but only the depth of that point in the array.

Notes

ARLASTP duplicates one part of the ARSTATUS command, but is easier to use. ARLASTP is particularly useful in process control applications.

ARLOAD

Purpose

The ARLOAD command loads a specified KDAC500 array from disk into memory. The array accessed must have been saved earlier with the ARSAVE command.

ARLOAD is a foreground routine. It can be executed when interrupts are on, although time skewing of the data may result at the faster interrupt rates.

Language Syntax

QuickBASIC: call arload (arrnm, DOSfile)

C: arload(arrnm, DOSfile);

FORTRAN: call arload (arrnm, DOSfile)

Pascal: arload(arrnm, DOSfile);

ARRNM

(string) The ARRay NaMe parameter identifies the KDAC500 array loaded into memory when ARLOAD is executed. Note that ARRNM is assigned a value by ARLOAD. When the ARLOAD command is given, the name of the array will be returned to the ARRNM parameter. This means that ARRNM must be initialized to a length that will be long enough to hold the returned string. In QuickBASIC this can be accomplished by:

ARRNM$ = SPACE$(255)

In C or Pascal declare ARRNM to be a 256 element array of char. In FORTRAN declare ARRNM as CHARACTER*256 ARRNM.

DOSFILE

(string) The DOSFILE parameter should be assigned the standard DOS file name given to the file created when the array was first saved with ARSAVE. This file name will appear in the directory of the disk.

Notes

When ARLOAD is executed, all information pertaining to the specified array is loaded with the array into memory. This information includes associated IONAMEs, the array label, the internal pointers, and all other information saved by the ARSAVE command.

ARSAVE and ARLOAD conform to DOS conventions for save and load operations. There are several errors that may result from improper disk access. The following table summarizes the conditions that will cause KDAC500 to return an error message:

4. File does not exist
5. Filename not completely specified
6. Invalid drive designator
7. Invalid filename
8. Filename specified is not a KDAC500 file

Other errors that may occur:

9. Insufficient memory (allocation is necessary, since ARLOAD essentially recreates the array in memory).
10. Array is still in memory, and must be deleted before reloading.

The actual ARRNM string in FORTRAN can be retrieved using the scan() function and FORTRAN's substring command. For example:

```
ARRNM (1:SCAN(ARRNM,CHAR(0)))
```

In QuickBASIC, the RTRIM$() command can be used.

```
RTRIM$(ARRNM)
```

Pascal & C do not need special treatment

ARMAKE

Purpose

The ARMAKE command creates a KDAC500 array in the KDAC500 memory.

Every KDAC500 array has two dimensions; width and depth. The depth of the array refers to the number of values associated with each width. An array with depth 1000 might be used to send 1000 consecutive bit values to a specified number of digital output channels.

The width is related to the number of channels accessed by KDAC500 commands that make use of the array. For example, an array created with 5 channels has a width of 5.

Language Syntax

QuickBASIC: call armake (arrnm, dep, ionl)

C: armake(arrnm, dep, ionl);

FORTRAN: call armake (arrnm, dep, ionl)

Pascal: armake(arrnm, dep, ionl);

ARRNM — (string) The ARRay NaMe parameter allows the user to assign a name to the array created by ARMAKE. Certain conventions must be followed when assigning an array name:

1. The name can have up to 255 characters however only the first 8 characters will be significant. Hence, the given name "ABCDEFGHIJKLMN%" will be truncated to "ABCDEFGH".
2. The array name may not include spaces, commas, tabs or control characters.

DEP — (long) The DEPth parameter defines the depth of the array created by ARMAKE. The DEP parameter may be assigned values of 1 or greater. Note, however, that the maximum depth for any array is always limited by the largest free block of contiguous memory in the system. Note also that the amount of memory taken by an array will depend both on the depth and on the type of the array, that is, a byte array with depth 1000 will take up 8 times as much memory as a bit array with depth 1000. The array type is determined by the IOName List parameter.

IONL — (string) The IONname List parameter is used to set the array width. All the IONAMEs specified must be of the same type (i.e. analog output, analog input, digital port, digital channel, etc.). Any references to the array can then be made by IONAME. (See ARGET/ARPUT).

Notes

If an array is to be used for analog output, the IONAME's used to create the array do not have to be the IONAME's actually used when the output is done. This allows an array created by BGREAD, ARMAKE or ANINQ to be used for output without having to copy the contents of the array into another array.

ARSTATUS

Purpose

The ARSTATUS command returns various information about a given KDAC500 array, including the depth of the array, the width, the last point accessed by a KDAC500 background routine, and the descriptive label (if any) associated with the array.

Language Syntax

QuickBASIC: call arstatus (arrnm, dep, wid, lp, labl)

C: arstatus(arrnm, &dep, &wid, &lp, labl);

FORTRAN: call arstatus (arrnm, dep, wid, lp, labl)

Pascal: arstatus(arrnm,dep,wid,lp,labl);

ARRNM — (string) The ARRay NaMe parameter is assigned the name of a KDAC500 array; this name identifies the array whose status will be returned by ARSTATUS. The array name must refer to a KDAC500 array that is currently in memory.

DEP — (long) The DEPth of the array is returned by ARSTATUS to the DEP parameter.

WID — (integer) The WIDth of the array is returned to the WID parameter.

LP — (long) ARSTATUS returns the depth of the last point accessed by a KDAC500 background routine to the LP parameter. The LP parameter is especially useful for determining the depth of the last point of input taken by input routines (BGREAD), or the last point output by KDAC500 output routines (BGWRITE).

If the user wants to know the depth of the last point accessed and doesn't need any additional information, the ARLASTP command will return the value of the LP parameter and nothing else (see ARLASTP).

LABL — (string) The LABeL parameter will hold the descriptive label given to the array with the ARLABEL command. This string must be initialized to a length long enough to hold the returned string. If LABL is longer than the descriptive label then LABL will be padded with spaces.

NOTE to C and FORTRAN programmers: LABL is assumed to be an array of char and is long enough to hold the returned string. The entire string is always returned to LABL when C or FORTRAN is the language type.

Notes

ARSTATUS is particularly useful for determining information about an array that has just been loaded into memory from disk.

ARSAVE

Purpose

The ARSAVE command saves a KDAC500 array on floppy or fixed disk in a number of different formats. ARSAVE will save a memory array of any size; the only limitation is the amount of space on the disk.

ARSAVE makes it easier to export data to spreadsheets, word processors, data bases, and analysis programs. The data can be saved in ASCII, DADiSP, Lotus 123, Asyst, or Keithley format.

Language Syntax

QuickBASIC: call arsave (arrnm, DOSfile, euf, ftype, samprate, tunits)

C: arsave (arrnm, DOSfile, euf, ftype, samprate, tunits);

FORTRAN: call arsave (arrnm, DOSfile, euf, ftype, samprate, tunits)

Pascal: arsave (arrnm, DOSfile, euf, ftype, samprate, tunits);

ARRNM — (string) The ARRay NaMe identifies the KDAC500 array that will be saved to disk when this command is executed. The ARRNM parameter must refer to a KDAC500 array which is resident in memory. If the specified array is not found, KDAC500 will return an error.

DOSFILE — (string) The DOSFILE parameter allows the user to assign a standard DOS filename to the data file stored to disk. The filename will be added to the disk directory and treated like any other DOS file.

EUF — (integer) The Engineering Units Flag parameter causes the engineering units conversion to be applied to the data before it is saved to the disk. If EUF is set to C_RAW_INT or C_RAW_FLOAT, the data in the array will be saved in a raw A/D count data format.

FTYPE — (integer) The File TYPE parameter indicates which filetype is desired:

QuickBASIC	C, Pascal, or FORTRAN	
FT.KDAC	FT_KDAC	Keithley binary format
FT.ASCII	FT_ASCII	ASCII file
FT.BIN16	FT_BIN16	Binary file with 16-byte ASCII header

FT.DADiSP	FT_DADiSP	DADiSP binary format
FT.LOT123	FT_LOT123	ASCII importable into Lotus 123 as a .PRN file
FT.ASYST	FT_ASYST	ASYST binary format

SAMPRATE (integer) The SAMPle RATE parameter is the rate at which data was collected. Set SAMPRATE to equal IR x BINTV (interrupt rate times background interval).

TUNITS (integer) The Time UNITS parameter that was specified for collecting data in the INTON command:

HMIC	Hundereds of microseconds
MIL	Milliseconds
SEC	Seconds
MIN	Minutes

Notes

ARSAVE is a foreground routine. If it is executed when interrupts are on and operating at the fastest rates, time skewing of the data acquisition may occur.

There are several errors that may result from improper disk access with ARSAVE. The following table summarizes the conditions that will cause KDAC500 to return an error message.

1. No space in file directory.
2. Filename not completely specified
3. Invalid filename.
4. Disk full.

The above errors will interrupt the command.

If the filename already exists, it will be overwritten by the new data.

Only files saved with FT.KDAC or FT_KDAC can be loaded by the ARLOAD command.

BGCLEAR

Purpose

BGCLEAR kills all current background functions. Any background functions that may have been set up earlier will be cleared by this command.

Language Syntax

QuickBASIC: call bgclear

C: bgclear();

FORTRAN: call bgclear()

Pascal: bgclear;

Notes

If a BGTIME command is executed after BGCLEAR has been called, BGTIME will not return 0, as might be expected, but a small number that represents the time taken by the interrupt processing routines.

BGGO

Purpose

The BGGO command allows the user to trigger up to 16 background routines that have been assigned the trigger mode WGO – Waiting for BGGO. When BGGO is executed, all such KDAC500 background routines will be started simultaneously.

When the TM parameter is assigned the value WBT, BGGO can itself be triggered by another KDAC500 background function configured as background trigger (BT).

Language Syntax

QuickBASIC: call bggo(tm, bfn)

C: bggo(tm, bfn);

FORTRAN: call bggo(tm, bfn)

Pascal: bggo(tm, bfn);

TM (integer) The Trigger Mode parameter can be used to set up a background trigger for the BGGO command. There are two valid strings for this parameter:

WBT	Wait for Background Trigger
NT	No Trigger

Setting TM to NT will execute BGGO in the foreground.

BFN (string) The Background Function Name is a standard KDAC500 parameter that allows the user to name the background task or tasks accomplished by the BGGO command.

Note that the BFN parameter may be used only when the BGGO command is set up as a background function (when the TM parameter is set to WBT). When not used, set BFN equal to the null string ("").

Notes

BGGO is a particularly useful function that can be used to start many functions simultaneously on a single trigger. Routines with WGO in the parameter list will be started by the first BGGO that

follows. This will be true whether that particular BGGO command is executed immediately, or is itself waiting for a background trigger. In the latter case, the WGO routines in the background will only be triggered by BGGO when BGGO itself is triggered in the background by a BT Background Trigger command.

When BGGO is used as a background routine, the functions that are Waiting for BGGO will not be started until the interrupt after the execution of the BGGO command. This means that there will be a delay between the trigger and the routine that is triggered. The delay will be approximately equal to the time between interrupts.

BGHALT

Purpose

The BGHALT command halts execution of up to 16 specified background tasks with assigned background function names. BGHALT can be triggered by another background function or by the BGGO command.

Depending on the value assigned to the TM (trigger mode) parameter, BGHALT can be a foreground or a background function.

Language Syntax

QuickBASIC: call bghalt (bfnl ,tm ,bfn)

C: bghalt(bfnl, tm, bfn);

FORTRAN: call bghalt (bfnl ,tm ,bfn)

Pascal: bghalt(bfnl, tm, bfn);

BFNL — (string) The Background Function Name List parameter is used by BGHALT to halt execution of the named tasks. This parameter may include up to 16 background function names, created earlier by the commands that set up the background tasks.

TM — (integer) The Trigger Mode parameter can be given three valid values in this command:

WBT	Wait for Background Trigger
WGO	Wait for BGGO
NT	No Trigger

If TM is set equal to NT, BGHALT will act as a foreground command, immediately halting the specified background tasks. If the TM parameter is equal to WBT or WGO, BGHALT will be set up as a background command.

BFN — (string) The Background Function Name is a standard KDAC500 parameter that allows the user to name the task accomplished by the BGHALT command. Note that this parameter may be used only when BGHALT is set up as a background routine, i.e., when the TM parameter is set to WBT or WGO. If a BGHALT command will be accessed by a BGSTATUS command or by another BGHALT, the BFN parameter must be included, when not used, set BFN equal to a null string (""), or ('') for Turbo Pascal.

Notes

BGHALT is essentially a command to "kill" background execution, irrevocably halting the background tasks identified in the name list.

BGHALT can be extremely useful when set up as a background function waiting for a BT. When used in this way, BGHALT will wait for some specified background trigger until the triggering condition occurs. BGHALT will then halt execution of the specified background tasks.

Note that one BGHALT may halt another.

When BGHALT is used as a background routine, the functions specified will not be halted until the interrupt after the execution of the BGHALT command. This means that there will be a delay between the trigger that executes the BGHALT command and the actual halting of the routine. The delay will be approximately equal to the time between interrupts.

BGREAD

Purpose

BGREAD is a background routine for sampling up to 64 channels of input. The BGREAD command causes measurements to be taken at a specified interval from a specified number of channels, and creates a KDAC500 array in which to store the acquired data. BGREAD can be triggered or can act as a trigger for another KDAC500 command.

Language Syntax

QuickBASIC: call bgread(arrynm,numsamp,ionl,bintv,range,cyc,tm,bfn)

C: bgread(arrynm,num_samp,ionl,bintv,range,cyc,tm,bfn);

FORTRAN: call bgread(arrynm,numsamp,ionl,bintv,range,cyc,tm,bfn)

Pascal: bgread(arrynm,num_samp,ionl,bintv,range,cyc,tm,bfn);

ARRYNM — (string) The ARRaY NaMe parameter is a standard KDAC500 array name given to the array created by the execution of the BGREAD command. KDAC500 arrays are accessible through KDAC500 array management commands only (see ARGET).

NUMSAMP — (long) The NUMber of SAMPles parameter indicates the number of samples to be taken from each channel.

IONL — (string) The IOName List parameter includes 1-64 names, where each name refers to a specific channel of input. The programmer can achieve a burst effect by using the same name several times in the name list. This will cause BGREAD to sample the channel several times on every execution of the command; once for every occurrence of the name. Note that the number of names in the IONAME list specifies the width of the array being created.

BINTV — (integer) The Background INTerVal parameter determines the time interval between each sample in terms of the interrupt period. The integer value given to the BINTV parameter multiplied by the interrupt period will yield the interval between each sample as some number of milliseconds. For example, if BINTV is set at 5, each channel will be sampled on every 5th interrupt. If the interrupt period is 10 milliseconds, the interval between samples will be 50 milliseconds.

RANGE — (integer) The RANGE parameter is only used when reading pulse or frequency channels. When the input channel is not a pulse or frequency channel set RANGE = NONE.

When measuring frequency, the RANGE parameter indicates the maximum frequency expected on the channels being measured. RANGE should be assigned a value according to the table on the next page. Note that the value given to RANGE affects the length of time taken by BGREAD to complete the measurement of each channel (gate time). In turn, the gate time affects the accuracy (resolution) of the measurement. Closely matching RANGE to your expected maximum frequency will result in the highest resolution possible.

When counting pulses RANGE specifies the type of pulse counting that will be used. The PIM1 can be configured to count in the normal mode or in gated mode. In normal mode all events occurring on a specified input channel will be monitored. In gated mode, two channels are monitored on the PIM1: channel n and channel n+4. Channel n represents the pulse input channel as specified in the IONL parameter. Channel n+4 acts as a gate for channel n. When channel n+4 is high BGREAD will count events occurring on channel n. When channel n+4 is low, events will not be counted. In this way the counting of events on channel n is made conditional on some other event or condition as defined by the state of channel n+4. When using the gated mode, the pulse input channel (channel n) must be channel 0, 1, 2, or 3. For more information on the PIM1 refer to the PIM1 reference manual.

The counters on the PIM2 can be read and left alone or read and reset back to 0. RANGE specifies which mode to use.

Range				
QuickBASIC	C, Pascal, or FORTRAN	MAX FREQ	Gate Time	Resolution
F.62K	F_62K	62.5 KHz	1028.576 mS	1.0 Hz
F.125K	F_125K	125 KHz	524.288 mS	1.9 Hz
F.250K	F_250K	250 KHz	262.144 mS	3.7 Hz
F.500K	F_500K	500 KHz	131.072 mS	7.5 Hz
F.1M	F_1M	1 MHz	65.536 mS	15.0 Hz
F.2M	F_2M	2 MHz	32.768 mS	30.0 Hz
F.4M	F_4M	4 MHz	16.384 mS	61.0 Hz
F.8M	F_8M	8 MHz	8.192 mS	122.0 Hz

Range		
QuickBASIC	C, Pascal or FORTRAN	Pulse Mode
P1.NORMAL	P1_NORMAL	Normal pulse read on PIM1
P1.GATED	P1_GATED	Gated pulse input on PIM1
P2.DEFAULT	P2_DEFAULT	Default mode for PIM2
P2.READ.RESET	P2_READ_RESET	Read and Reset for PIM2
P2.READ.ONLY	P2_READ_ONLY	Read only mode for PIM2

Range	
NONE	any mode that is not pulse or frequency

CYC (integer) The CYCling parameter determines the number of times the BGREAD task will cycle. If CYC is given the value 50, the BGREAD task will execute for 50 cycles. If CYC is given the value FOREVER, BGREAD will be executed repeatedly until halted by a BGHALT, BGCLEAR, or KDINIT command. The cycling BGREAD command is very useful for repeatedly reading analog input in the background without having to store vast arrays of information. Hence, it is well adapted for process control applications and real-time graphing.

TM (integer) The Trigger Mode parameter indicates whether the BGREAD command is associated with some other command, either as a trigger for that command, or as a function to be triggered. There are four valid values for this parameter:

BT	Background Trigger
WBT	Wait for Background Trigger
WGO	Wait for BGGO
NT	No Trigger

BFN (string) The Background Function Name parameter is used to name the background task being performed by BGREAD.

If a background function name is not used, you must set this parameter equal to a string of zero length (i.e. "" or '').

Notes

When BGREAD is used to measure signals produced by thermocouples, it will be necessary to give an I/O name to Channel 32 of the TC module to which the thermocouples are connected. Channel 32 accesses the cold junction compensation circuitry. This channel must be read whenever thermocouple inputs are connected. The I/O name given to Channel 32 must be the **first** name in the I/O name list. The values returned from reading Channel 32 will be stored along with all other data collected with the BGREAD command. When ARGET retrieves data from the KDAC500 array, it will assume the cold junction compensation values are the first width of the array and apply them to the values taken from the thermocouple inputs. In this way, the ARGET command is able to return an accurate measurement of the thermocouple signal, by converting the raw values to volts or degrees celsius.

BGSTATUS

Purpose

BGSTATUS is a foreground function that returns the current status of a specified background routine. The BGSTATUS command returns a value to the STAT parameter, indicating whether the background function is off, on, waiting for a background trigger (WBT), or waiting for BGGO (WGO).

Language Syntax

QuickBASIC: call bgstatus (bfn, stat)

C: bgstatus(bfn, &stat);

FORTRAN: call bgstatus (bfn, stat)

Pascal: bgstatus(bfn, stat);

BFN — (string) The Background Function Name parameter indicates the background function whose status is returned by the BGSTATUS command. If the background function name does not match the names of any background tasks currently set up by KDAC500, the BGSTATUS command will return a value indicating that the function is not active.

STAT — (integer) The STATus of the specified background function will be returned in the STAT parameter. This parameter will be assigned one of four values:

ST_DONE ST.DONE	The function is off or not found. (QuickBASIC)
ST_EXECUTE ST.EXECUTE	The function is on (executing). (QuickBASIC)
ST_WBT ST.WBT	The function is waiting for a background trigger. (QuickBASIC)
ST_WGO ST.WGO	The function is waiting for a BGGO command. (QuickBASIC)

BGTIME

Purpose

The BGTIME function returns the amount of time taken up by all the KDAC500 routines set up in the background. BGTIME measures the time taken from the moment a Series 500 interrupt occurs to the moment when control is returned to normal foreground execution. This time includes the time taken by the interrupt processing routine and other internal tasks.

Language Syntax

QuickBASIC: call bgtime (time)

C: bgtime(&time);

FORTRAN: call bgtime (time)

Pascal: bgtime(time);

TIME (float) The amount of time taken by the background is returned to the TIME parameter as a variable expressed in milliseconds. This value will have three significant decimal places. For example, BGTIME might return the value 6.391362; this should be interpreted as 6.391 milliseconds.

BGWRITE

Purpose

BGWRITE is the background routine that writes specified values to digital, or analog, channels or ports. If the output channels are analog the user can specify the output as raw values or in engineering units. BGWRITE gets its values from a KDAC500 array.

Language Syntax

QuickBASIC: call bgwrite(arrynm,ionl,bintv,cyc,tm,bfn)

C: bgwrite(arrynm,ionl,bintv,cyc,tm,bfn);

FORTRAN: call bgwrite(arrynm,ionl,bintv,cyc,tm,bfn)

Pascal: bgwrite(arrynm,ionl,bintv,cyc,tm,bfn);

ARRYNM — (string) The ARRaY NaMe parameter is a standard KDAC500 array name given to the array created by the execution of a BGWRITE, ANINQ, or ARMAKE command. KDAC500 arrays are accessible through KDAC500 array management commands only (see ARGET, ARPUT).

IONL — (string) The IOName List parameter may include from 1 to 64 IONAMES. The IONAMES are mapped 1 to 1 to the widths in ARRYNM. The IONAMES specified do not have to be the IONAMES that created the array. In this way data acquired with BGWRITE or ANINQ can be directly output from the same array. (AOM5 data must be converted.)

BINTV — (integer) The Background INTerVal parameter determines the time interval between each sampie in terms of the interrupt period. The integer value given to the BINTV parameter multiplied by the interrupt period will yield the interval between each sample as some number of milliseconds. For example, if BINTV is set at 5, each channel will be sampled on every 5th interrupt. If the interrupt period is 10 milliseconds, the interval between samples will be 50 milliseconds.

CYC — (integer) The CYCling parameter determines the number of times the BGWRITE task will cycle. If CYC is given the value 50, the BGWRITE task will execute for 50 cycles. If CYC is given the value FOREVER, BGWRITE will be executed repeatedly until halted by a BGHALT, BGCLEAR, or KDINIT command. The cycling BGWRITE command is very useful for repeatedly outputting data in the background without having to store vast arrays of informa-

tion. Hence, it is well adapted for process control applications and real-time graphing.

TM — (integer) The Trigger Mode parameter indicates whether the BGWRITE command is associated with some other command, either as a trigger for that command, or as a function to be triggered. There are four valid values for this parameter:

BT	Background Trigger
WBT	Wait for Background Trigger
WGO	Wait for BGGO
NT	No Trigger

BFN — (string) The Background Function Name parameter is used to name the background task being performed by BGWRITE.

If a background function name is not used, you must set this parameter equal to a null string (i.e. "" or '').

BLANK

Purpose

BLANK is a foreground graphics command that is used to clear the screen. It supports the DATAQ waveform scroller card, and will also clear the screen when doing IBM CGA graphics. If the waveform scroller is used, BLANK will only clear the graphics produced by the scroller, not the text produced by the program.

Language Syntax

QuickBASIC: call blank

C: blank();

FORTRAN: call blank()

Pascal: blank;

FGREAD

Purpose

FGREAD is the foreground routine that samples digital, analog, pulse, and frequency inputs. FGREAD returns values directly to the calling language for easy access.

Language Syntax

QuickBASIC: call fgread(ionl, range, varseg(vl(0)), varptr(vl(0)), euf, tm)

C: fgread (ionl, range, &vl[0], euf, tm);

FORTRAN: call fgread(ionl, range, LOCFAR(vl(1)), euf, tm)

Pascal: fgread (ionl, range, @vl[0], euf, tm);

IONL — (string) The Ioname list is made up of IONAMES separated by commas, spaces or both. An ioname is a standard parameter used to indicate which channel or port of input will be read by the FGREAD command. These names must have been created earlier with the CONFIG program.

RANGE — (integer) The RANGE parameter is only used when reading pulse or frequency channels. When the input channel is not a pulse or frequency channel set RANGE = NONE.

When measuring frequency, the RANGE parameter indicates the maximum frequency expected on the channels being measured. RANGE should be assigned a value according to the table on the next page. Note that the value given to RANGE affects the length of time taken by FGREAD to complete the measurement of each channel (gate time). In turn, the gate time affects the accuracy (resolution) of the measurement. Closely matching RANGE to your expected maximum frequency will result in the highest resolution possible.

When counting pulses RANGE specifies the type of pulse counting that will be used. The PIM1 can be configured to count in the normal mode or in gated mode. In normal mode all events occurring on a specified input channel will be monitored. In gated mode, two channels are monitored on the PIM1: channel n and channel n+4. Channel n represents the pulse input channel as specified in the IONL parameter. Channel n+4 acts as a gate for channel n. When channel n+4 is high FGREAD will count events occurring on channel n. When channel n+4 is low, events will not be counted. In this way the counting of events on channel n is made conditional on

some other event or condition as defined by the state of channel n+4. When using the gated mode, the pulse input channel (channel n) must be channel 0, 1, 2, or 3. For more information on the PIM1 refer to the PIM1 reference manual.

The counters on the PIM2 can be read, or read and reset back to 0. RANGE specifies which mode to use.

Range				
QuickBASIC	C, Pascal or FORTRAN	MAX FREQ	Gate Time	Resolution
F.62K	F_62K	62.5 KHz	1028.576 mS	1.0 Hz
F.125K	F_125K	125 KHz	524.288 mS	1.9 Hz
F.250K	F_250K	250 KHz	262.144 mS	3.7 Hz
F.500K	F_500K	500 KHz	131.072 mS	7.5 Hz
F.1M	F_1M	1 MHz	65.536 mS	15.0 Hz
F.2M	F_2M	2 MHz	32.768 mS	30.0 Hz
F.4M	F_4M	4 MHz	16.384 mS	61.0 Hz
F.8M	F_8M	8 MHz	8.192 mS	122.0 Hz

Range		
QuickBASIC	C, Pascal or FORTRAN	Pulse Mode
P1.NORMAL	P1_NORMAL	Normal pulse read on PIM1
P1.GATED	P1_GATED	Gated pulse input on PIM1
P2.DEFAULT	P2_DEFAULT	Default mode for PIM2
P2.READ.RESET	P2_READ_RESET	Read and Reset for PIM2
P2.READ.ONLY	P2_READ_ONLY	Read only mode for PIM2

Range	
NONE	any mode that is not pulse or frequency

VL () (Pointer to a float, single, or long) This variable (or array) holds the results of the readings. If more than one ioname is specified in IONL then VL must be an array with a depth at least as large as the number of entries in the IONL parameter. The values will appear in the array in the same order as the channels or ports were listed in the ioname list.

ANALOG CHANNELS – When the EUF parameter is set to C_RAW_INT or C_RAW_FLOAT, VL will hold raw values from 0-4095 (for systems with 12 bit A/D's), 0-16383 (for systems with 14 bit A/D's), or 0-65535 (for systems with 16 bit A/D's). The EUF parameter can also be set to return values expressed as volts or engineering units.

DIGITAL CHANNELS or PORTS – When FGREAD samples a digital channel the status of that channel will be returned as a 0 or a 1(off or on; TTL low or high). If FGREAD samples a port, the status of the port will be returned as a decimal number from 0 to 255. This number can be looked at as an 8 bit binary value where the status of each bit represents the status of one channel.

Consider the following example:

FGREAD reads a port of digital input from a DIO1 module and returns the decimal value 100. When this value is converted to a binary number (01100100), the status of each bit corresponds to the status of one channel:

Ch7	Ch6	Ch5	Ch4	Ch3	Ch2	Ch1	Ch0
0	1	1	0	0	1	0	0
off	on	on	off	off	on	off	off

FREQUENCY CHANNELS – When EUF is set to C.RAW.INT, C.RAW.FLOAT, C_RAW_INT, or C_RAW_FLOAT, FGREAD will return raw values from 0 to 65535 (a reading of 65535 indicates that the frequency is above the limit specified by RANGE). By setting the engineering flag to C.FREQ or C_FREQ, values can be returned as some number of Hertz.

PULSE CHANNELS – Each entry in VL will contain an unsigned long integer which holds the input channels count value. The count will be in the range 0-4294967295. EUF should be set to C_COUNT or C.COUNT for pulse channels.

EUF (integer) The engineering units conversion flag is used to specify that raw values be converted to volts, frequency, or engineering units. See the EUF section for a complete description of the engineering units flag.

TM (integer) The trigger mode parameter allows FGREAD to be triggered by the execution of another KDAC500 command. There are only two valid trigger modes for this command:

WST	Wait for Singleground Trigger
NT	No Trigger

Note that when WST is used, FGREAD is used as a singleground function. If the user specifies WST when the background is on, a warning error will be given. If NT is specified FGREAD will operate in the foreground.

Notes

FGREAD may be called as a foreground routine if a background routine is active, however, if there is not enough time left in the foreground to do the sampling without being interrupted, an interrupt will be skipped.

Cold junction compensation is automatically provided when the IONL parameter specifies a channel on a thermocouple module and the EUF parameter is given a value that indicates a thermocouple is being used. Excitation is provided automatically when the IONL parameter specifies a channel on an AIM8 and the EUF parameter is set to C_AIM8_C, C_AIM8_D, C.AIM8.C, or C.AIM8.D.

The IONAME list may include a single channel mentioned multiple times. This would allow the user to take a burst of readings from a single channel during the execution of one foreground routine.

FGWRITE

Purpose

FGWRITE is the foreground routine that writes specified values to digital, or analog, channels or ports. If the output channels are analog the user can specify the output as raw values or in engineering units. FGWRITE gets its values from a variable or array passed to it from the calling language.

Language Syntax

QuickBASIC: call fgwrite(ionl, varseg(vl(0)), varptr(vl(0)), euf, tm)

C: fgwrite (ionl, &vl[0], euf, tm);

FORTRAN: call fgwrite(ionl, LOCFAR(vl(1)), euf, tm)

Pascal: fgwrite (ionl, @vl[0], euf, tm);

IONL (string) The IOName list is made up of IONAMES separated by commas, spaces or both. An IONAME is a standard parameter used to indicate which channel or port will be written to by the FGWRITE command. These names must have been created earlier with the CONFIG program.

VL (Pointer to a float, or integer) This variable (or array) holds the value(s) to be written. If more than one ioname is specified in IONL then VL must be an array with a depth at least as large as the number of entries in the IONL parameter. The values should appear in the array in the same order as the channels or ports were listed in the IONAME list.

ANALOG CHANNELS – When the EUF parameter is set to C.RAW.INT, C.RAW.FLOAT, C_RAW_INT, or C_RAW_FLOAT, VL should hold raw values from 0-4095 (for systems with 12 bit D/A's) or 0-65535 (for systems with 16 bit D/A's). The EUF parameter can also be set so values can be expressed as volts or engineering units.

DIGITAL CHANNELS or PORTS – When the elements of VL are not integer numbers, they will be rounded to the nearest whole number when the values are transmitted. When FGWRITE accesses digital channels the elements of VL should be assigned the values 1 or 0, indicating the on and off (high and low) status of that channel. When accessing ports, the value assigned the elements of VL must be from 0 to 255. When these numbers are converted to 8-bit binary values, the status of each bit will correspond to the status of one of the 8 channels of the digital output port.

Suppose the user wishes to to write to port B on a DIO1 module, turning channels 8–14 on and turning channel 15 off. the corresponding element of VL is assigned the value 127.0 (01111111B), representing the status of the channels as follows:

Ch15	Ch14	Ch13	Ch12	Ch11	Ch10	Ch9	Ch8
0	1	1	1	1	1	1	1
off	on	on	on	on	on	on	on

EUF (integer) The engineering units conversion flag is used to specify that input values are specified in volts or engineering units. The EUF parameter is only used for analog channels. Set EUF = C.RAW.INT or C_RAW_INT if the outputs channels are digital or if the output values are raw D/A values. See the EUF section for a complete description of the engineering units flag.

TM (integer) The trigger mode parameter allows FGWRITE to be triggered by the execution of another KDAC500 command. There are only two valid trigger modes for this command:

WST	Wait for Singleground Trigger
NT	No Trigger

Note that when WST is used, FGWRITE will be executed immediately following the execution of a command with ST (Singleground Trigger) in its parameter list. If the user specifies WST when the background is on, a warning error will be given. If NT is specified FGWRITE will operate in the foreground.

Notes

FGWRITE may be called as a foreground routine when a background routine is active. However, if there is not enough time left in the foreground to do the sampling without being interrupted, an interrupt will be skipped.

GETHANDLE

Purpose

The GETHANDLE command transfers the KDAC500 "handle" associated with an IONAME to an integer variable. GETHANDLE is a support command for the "handle-using" foreground functions HREAD and HWRITE.

Internally KDAC500 identifies every IONAME with an integer value called a "handle" Everytime a foreground read or write command is executed (FGREAD, FGWRITE) KDAC500 searches its list of IONAMES for a string that matches the one found in the IONL parameter of the command. Each IONAME has a unique "handle" value assigned to it by KDAC500 when it is defined by CONFIG. This value remains unchanged throughout the execution of a KDAC500 program.

If the same channel is being accessed multiple times from the foreground, the IONAME string search will occur every time. This multiple string search can be avoided by acquiring the "handle" value one time using GETHANDLE at the start of the program. Once acquired, the "handle" can be used in the "handle-using" foreground commands. Depending on how often the channel is accessed, and the total number of IONAMES in the system, the performance increase can be appreciable.

Language Syntax

QuickBASIC: call gethandle(ionl,varseg(handles(0)),varptr(handles(0)))

C: gethandle (ionl, &handles[0]);

FORTRAN: call gethandle(ionl,handles)

Pascal: gethandle (ionl, handles[0]);

IONL (string) The Ioname list is made up of IONAMES separated by commas, spaces or both. An ioname is a standard parameter used to indicate which channel or port of input will be accessed by a KDAC500 command. These names must have been created earlier with the CONFIG program.

HANDLES (Pointer to an integer array) The GETHANDLE command will return the values of the "handles" associated with the given IONL into the HANDLES array. If a particular IONAME is not found, a value of 0 will be returned in the corresponding location in HandleList. After the last IONAME, a value of 0 is placed in handles.

Therefore, handles should be dimensioned to 1 larger than the number of IONAMES.

Notes

The performance increase yielded with the "handle-using" commands can only be realized when the interrupts are off. If the interrupts are on, the normal foreground read and write commands have the same performance characteristics as the "handle-using" read and write commands. The "handle-using" read and write commands yield the greatest performance improvement over the normal foreground read and write commands when three situations are coincident: the interrupts are off, the number of IONAMES is large, and a large number of foreground reads and writes are desired that deal with the same channel or group of channels.

GRAPH

Purpose

GRAPH is a foreground routine that provides horizontal, linear graphing of data values stored in KDAC500 arrays. The GRAPH command will support several sets of data, graphing each set in a specified color.

GRAPH may be used to graph analog or digital values from any KDAC500 array, or any part of a KDAC500 array specified by DEP1 and DEP2. By setting the engineering units flag, the user may specify an internal conversion of raw binary values to volts or engineering units.

GRAPH also has the option of data magnification or reduction, by which a set of data points may be magnified or reduced on the screen.

Language Syntax

QuickBASIC:

```
call graph (arrnm, varseg(widl(0)), varptr(widl(0)), varseg(coll(0)),
varptr(coll(0)), displm, miny, maxy, mrm, res, dep1, dep2, euf)
```

C:

```
graph( arrnm, &widl[0], &coll[0], displm, miny, maxy, mrm, res,dep1,
dep2, euf );
```

FORTRAN:

```
call graph (arrnm, LOCFAR(widl(1)), LOCFAR(coll(1)), displm,
miny, maxy, mrm, res, dep1, dep2, euf)
```

Pascal:

```
graph(arrnm, widl[0], coll[0], displm, miny, maxy, mrm, res,dep1,
dep2, euf );
```

ARRNM — (string) The ARRay NaMe parameter should be given the name of the array to be accessed by the GRAPH command. From this array, the user may specify several sub–arrays (given as widths in the array) for graphing. The number of sub–arrays to be graphed will be determined by the number of widths included in the width list (WIDL) parameter.

All sub-arrays will be of the same type as the array indicated by the ARRNM parameter.

WIDL — (Pointer to a 16 element integer array) The WIDth List parameter will specify those sub-arrays to be graphed by the GRAPH command.

The WIDL parameter is an array of 16 elements. Each element of the array should be assigned the integer values of the widths to be accessed by GRAPH. After the entry of the last channel to be graphed, the WIDL should be set to –1 to indicate the end of the list. The following C statements specify widths one, three, and five:

```
WIDL[0] = 1;      /* assign width 1 */
WIDL[1] = 3;      /* assign width 3 */
WIDL[2] = 5;      /* assign width 4 */
WIDL[3] = –1;     /* terminate the list */
```

The GRAPH command will treat each width in the width list as a separate sub-array.

Since different widths in an array correspond to different channels from which samples were taken, the WIDL parameter is a useful way of specifying only those channels that the user wants to graph. These widths may be specified in any order.

The number of widths that may be graphed at one time is determined by the type of the array. The maximum widths for each type are summarized below:

Type of Array	Number of Widths
bit arrays	16
byte arrays	2*
integer arrays	8
long arrays	8

*Note that a byte array will be graphed as 8 individual bits. Thus, two sub-byte arrays will be graphed as 16 bit sub-arrays.

NOTE: The WIDth List MUST be dimensioned to 16 elements even if only one channel is being graphed.

COLL (Pointer to a 16 element integer array) The nth width in the width list will be graphed using the nth color in the COLor List. If there are more colors than widths, the extra colors will be ignored. If there are more widths than colors, then colors will be assigned to widths until the color list has been exhausted. The next width will be graphed using the first color again, the following width with the second color, and so on, until every width has been assigned a color. After the entry of the last color, the COLor List should be set to –1 to indicate the end of the list. The following table shows the legal color values:

IBM CGA Graphics

Pallette 0		Pallette 1
0	background	background
1	green	cyan
2	red	magenta
3	brown	white

DATAQ Waveform Scroller Card

0	black
1	blue
2	green
3	cyan
4	light red
5	light magenta
6	yellow
7	bright yellow

NOTE: The COLor List MUST be dimensioned to 16 elements even if only one channel is being graphed.

DISPLM (integer) The DISPLay Mode parameter specifies how the graph will be presented on the screen, whether scrolled, paged, overlaid (IBM style) left to right or right to left (DATAQ style).

IBM CGA Graphics

SCROLL Scroll; the graph will be scrolled across the window.

PAGEC Page Clear; the graph will be paged, i.e. the window will be cleared after each window is filled.

PAGEO Page Overlay; The graph will be paged, but the window will not be cleared after the window is filled nor when the command is first given.

Note that when DISPLM is assigned either PAGEC or SCROLL the screen will be cleared on the very first execution of GRAPH, before new data is graphed. If DISPLM is set to PAGEO, the screen will not be cleared at all.

DATAQ Waveform Scroller Card

L_SCROLL Scroll left to right
L.SCROLL (QuickBASIC)

R_SCROLL	Scroll right to left
R.SCROLL (QuickBASIC)	

MINY (double) The MINimum value of the Y coordinate parameter sets the minimum value of the Y coordinate of the graph.

When accessing a bit or byte array (created for digital values), MINY must be set to 0 and MAXY set to 1. If they are not set to these values, a fatal error will be given.

When accessing a word array (created for analog values), MINY may be any number less than or equal to MAXY. In the special case when MINY is set equal to MAXY (only legal for word arrays), KDAC500 will provide automatic range finding based on the minimum and maximum data values stored in the sub-array (or segment of the sub-array) being graphed.

When accessing an array filled with analog data, MINY will be interpreted in terms of engineering units. See EUF for more information.

MAXY (double) The MAXimum value of the Y coordinate parameter sets the maximum value of the Y coordinate of the graph.

When accessing a bit or byte array (created for digital values), MAXY must be set to 1. When accessing a word or real array, MAXY may be any number greater than or equal to MINY. In the special case when MAXY equals MINY, KDAC500 will provide automatic range finding based on the minimum and maximum data values stored in the sub-array (or segment of the sub-array) being graphed.

When accessing an array filled with analog data, MAXY will be interpreted in terms of engineering units. See EUF for more information.

MRM (integer) The Magnification/Reduction Mode must be used with the RES parameter. The MRM parameter gives the user the option of magnifying or reducing the number of data points plotted by GRAPH.

There are three valid values for this parameter:

MAGNIFY	Magnify data by some factor
REDUCE	Reduce data by some factor
NORMAL	Plot data as-is

The factor of magnification or reduction is given by the RES parameter.

When MRM equals MAGNIFY the GRAPH command will pull the data points apart and draw straight lines between them, thereby extending the graph so that it has a greater expanse along the x axis. When MRM equals REDUCE the GRAPH command will take groups of data points and average them. These averages will be plotted rather than the original data results. In this way, the graph will be reduced by some factor.

The factor of magnification or reduction is given by the RES parameter, which indicates how many data points to average to produce a single new result (reduction mode), or how many screen coordinates to add between data points (magnification mode).

RES — (integer) The RESolution must be used with the MRM parameter. The RES parameter should be assigned a value to specify the factor by which the graph will be magnified or reduced. If, for example, MRM is set for REDUCE and RES is assigned the value 10, the graph will be reduced by a factor of 10.

The RES parameter may be assigned any integer value from 1-100, or –1. When RES is given the special value –1, the graph will automatically be modified to fit a single window.

If MRM is set to NORMAL then RES is not used and should be set to 1.

DEP1, DEP2 — (long) The DEPth 1 and DEPth 2 parameters must be used together. By assigning values to DEP1 and DEP2, the user may specify that only some segment of the sub–array will be graphed. DEP1 and DEP2 cause GRAPH to plot only that segment of the array beginning at depth 1 and ending at depth 2.

To plot all the points, DEP1 should equal 1 and DEP2 should equal the depth of the array.

EUF — (integer) The Engineering Units Flag parameter determines how to interpret the MINY and MAXY values. A value of C.RAW.FLOAT or C_RAW_FLOAT indicates that MINY and MAXY are expressed as raw binary values. Refer to the EUF section for the complete list of values that may be assigned to this parameter.

NOTE: The EUF value C.RAW.INT or C_RAW_INT is interpreted the same as FLOAT for MINY and MAXY becaue MINY and MAXY are always single precision real values.

Notes

To terminate graphing at any time, press the escape key (ESC). This will completely halt the GRAPH command and cause the program to continue with the next program line. To re-create the graph, a new GRAPH command must be given. The command will also wait at the end of a disply of one full screen of data for a keyboard input before displaying the next page of data.

GRAPHRT

Purpose

GRAPHRT is a foreground routine that provides realtime, horizontal graphing of one to several channels accessed by the following background routines: BGREAD and BGWRITE. With analog channels, the user may set the engineering units flag, specifying an internal conversion of raw binary values to volts or engineering units.

Language Syntax

QuickBASIC: call graphrt (arrnm, varseg(widl(0)), varptr(widl(0)), varseg(coll(0)), varptr(coll(0)), displm, miny, maxy, npts, euf)

C: graphrt(arrnm, &widl[0], &coll[0], displm, miny, maxy, npts, euf);

FORTRAN: call graphrt (arrnm, LOCFAR(widl(1)), LOCFAR(coll(1)), displm, miny, maxy, npts, euf)

Pascal: graphrt(arrnm, widl[0], coll[0], displm, miny, maxy, npts, euf);

ARRNM — (string) The ARRay NaMe parameter should be given the name of the array to be accessed by the GRAPHRT command. From this array, the user may specify several sub-arrays (given as widths in the array) for graphing. The number of sub-arrays to be graphed will be determined by the number of widths included in the width list (WIDL) parameter.

All sub-arrays will be of the same type as the array indicated by the ARRNM parameter.

WIDL — (Pointer to a 16 element integer array) The WIDth List parameter will specify those sub-arrays to be graphed by the GRAPHRT command.

The WIDL parameter is an array of 16 elements. Each element of the array should be assigned the integer values of the widths to be accessed by GRAPHRT. After the entry of the last channel to be graphed, the WIDL should be set to –1 to indicate the end of the list. The following C statements specify widths one, three, and five:

```
WIDL[0] = 1;      /* assign width 1 */
WIDL[1] = 3;      /* assign width 3 */
WIDL[2] = 5;      /* assign width 4 */
WIDL[3] = -1;     /* terminate the list */
```

Since different widths in an array correspond to different channels from which samples were taken, the WIDL parameter is a useful way of specifying only those channels that the user wants to graph. These widths may be specified in any order.

The number of widths that may be graphed at one time is determined by the type of the array. The maximum widths for each type are summarized below:

Type of Array	Number of Widths
bit arrays	16
byte arrays	2*
integer arrays	8
long arrays	8

*Note that a byte array will be graphed as 8 individual bits. Thus, two sub-byte arrays will be graphed as 16 bit sub-arrays.

NOTE: The WIDth List MUST be dimensioned to 16 elements even if only one channel is being graphed.

COLL (Point to a 16 element integer array) The nth width in the width list will be graphed using the nth color in the COLor List. If there are more colors than widths, the extra colors will be ignored. If there are more widths than colors, then colors will be assigned to widths until the color list has been exhausted. The next width will be graphed using the first color again, the following width with the second color, and so on, until every width has been assigned a color. After the entry of the last color, the COLor List should be set to –1 to indicate the end of the list. The following table shows the legal color values:

IBM CGA Graphics

Pallette 0		Pallette 1
0	background	background
1	green	cyan
2	red	magenta
3	brown	white

DATAQ Waveform Scroller Card

0	black
1	blue
2	green
3	cyan
4	light red
5	light magenta
6	yellow
7	bright yellow

NOTE: The COLor List MUST be dimensioned to 16 elements even if only one channel is being graphed.

DISPLM (integer) The DISPLay Mode parameter specifies how the graph will be presented on the screen, whether scrolled, paged, overlaid (IBM style) left to right or right to left (DATAQ style).

IBM CGA Graphics

SCROLL Scroll; the graph will be scrolled across the window.

PAGEC Page Clear; the graph will be paged, i.e. the window will be cleared after each window is filled.

PAGEO Page Overlay; The graph will be paged, but the window will not be cleared after the window is filled nor when the command is first given.

Note that when DISPLM is assigned either PAGEC or SCROLL the screen will be cleared on the very first execution of GRAPHRT, before new data is graphed. If DISPLM is set to PAGEO, the screen will not be cleared at all.

DATAQ Waveform Scroller Card

L_SCROLL Scroll left to right
L.SCROLL (QuickBASIC)

R_SCROLL Scroll right to left
R.SCROLL (QuickBASIC)

MINY (double) The MINimum value of the Y coordinate parameter sets the minimum value of the Y coordinate of the graph.

When accessing a bit or byte array (created for digital values), MINY must be set to 0 and MAXY set to 1. If they are not set to these values, a fatal error will be given.

When accessing a word array (created for analog values), MINY may be any number less than MAXY. Note that MINY **may not** be set equal to MAXY in GRAPHRT. If they are set equal, a fatal error will be given.

When accessing an array filled with analog data, MINY will be interpreted in terms of engineering units. See EUF for more information.

MAXY — (double) The MAXimum value of the Y coordinate parameter sets the maximum value of the Y coordinate of the graph.

When accessing a bit or byte array (created for digital values), MAXY must be set to 1. When accessing a word or real array, MAXY may be any number greater than MINY. Note that MINY **may not** be set equal to MAXY in GRAPHRT. If they are set equal, a fatal error will be given.

When accessing an array filled with analog data, MAXY will be interpreted in terms of engineering units. See EUF for more information.

NPTS — (long) The Number of PoinTS parameter is used to specify the number of points which will be graphed before the GRAPHRT command is halted. By assigning a number to NPTS, the user can determine in advance how many points will be graphed.

The NPTS parameter may be given legal values from 1 on up, or FOREVER, which indicates that GRAPHRT will continue to graph data points until the background task is halted.

EUF — (integer) The Engineering Units Flag parameter determines how to interpret the MINY and MAXY values. A value of C.RAW.FLOAT or C_RAW_FLOAT indicates that MINY and MAXY are expressed as raw binary values. Refer to the EUF section for the complete list of values that may be assigned to this parameter.

NOTE: The EUF value C.RAW.INT or C_RAW_INT is interpreted the same as FLOAT for MINY and MAXY becaue MINY and MAXY are always single precision real values.

Notes

To terminate graphing at any time, press the escape key (ESC). This will completely halt the GRAPHRT command and cause the program to continue with the next program line. To re-create the graph, a new GRAPHRT command must be given.

Since GRAPHRT is a foreground routine, it is interrupted by the function it is graphing (and any others in the background). Thus, if the proportion of background time to foreground time is high, GRAPHRT will not run as smoothly as it would if there were more time for it in the foreground (i.e., more data points will be skipped over in graphing).

The speed of the real-time graphing for analog data is affected by EUF as follows :

1. EUF is set to C.RAW. FLOAT, C.RAW.INT, C_RAW_FLOAT or C_RAW_INT, raw binary (fastest).
2. EUF set to anything but thermocouple or RTD types (fast).
3. EUF set to thermcouple or RTD types (slow).

There are two side effects you may notice when running GRAPHRT at optimal speed. They are generally noticeable only when graphing a signal that is changing rapidly:

1. If SCROLL is specified, once scrolling begins the graphed signal may appear fragmented. This is because the data points sampled while scrolling are missed by the graphing routine. This apparent fragmentation can be avoided by specifying PAGEC (page clear) for the DISPLM parameter.
2. If the interrupt rate is not sufficiently fast, the graphing routine may "extend" the last data point sampled into a short horizontal line while waiting for a new sample to plot. This can be eliminated by increasing the interrupt rate (decreasing the interval).

GRLABEL

Purpose

The GRLABEL command prints the text statements when the high-resolution graphics screen is on, allowing you to label graphs created with the HGRAPHRT command. The label can be for the horizontal or the vertical axis.

Language Syntax

QuickBASIC: call grlabel (labl, win, nwin, loc1, loc2)

C: grlabel(labl, win, nwin, loc1, loc2);

FORTRAN: call grlabel (labl, win, nwin, loc1, loc2)

Pascal: grlabel(labl, win, nwin, loc1, loc2);

LABL — (string) The LABeL parameter is the string that will be displayed on the graphics screen. When labeling the horizontal axis LABL can be up to 64 characters long. The maximum number of vertical characters is 21 with one window displayed, 10 with two windows, and 6 with three windows. Any printable character is legal.

WIN — (integer) The WINdow position (integer); input. Assign a value to the WIN parameter in order to specify the window that should be labelled with the string of the LABL parameter. There are three legal values for this parameter:

3. Window at the bottom of the screen.
4. Middle window (top window if only 2).
5. Window at top of screen.

The number assigned to WIN should be less than or equal to the total number of windows being graphed in (i.e., less than or equal to the NWIN parameter).

NWIN — (integer) The Number of WINdows parameter indicates the total number of windows being used (1, 2, or 3). This number should be the same as the WND parameter in the HGRAPHRT command list.

LOC1 — (integer) The LOCation 1 parameter allows you to position the text string either at the top, bottom, or to the left of the desired window. The legal values for this parameter are:

LEFT — Position the text to the left of the window (aligned on the vertical plane).

TOP Position the text at the top of the window (aligned on the horizontal plane).

BOTTOM Position the text at the bottom of the window (aligned on the horizontal plane).

The BOTTOM location for window 2 is the same as the TOP location for window 1. The BOTTOM location for window 3 is the same as the TOP of window 2. The line last written to a location will prevail, and will overwrite any label previously written to that location.

LOC2 (integer) The LOCation 2 parameter gives you the option of justifying the text string. If the text is at the top or bottom of the window, use one of the following strings:

RIGHT Right justify the text

LEFT Left justify the text

CTR Center the text on the horizontal plane

If the text is to the left of the window, use:

TOP Top justify the text

BOTTOM Bottom justify the text

CTR Center the text on the vertical plane

The LOC1 and LOC2 parameters combine to allow you to position the text string exactly where you want it on the screen, without the trial-and-error. Text positioned with parameters will not be overwritten when points are plotted on the desired graph.

Notes

The GRLABEL routine does not put the display adapter into graphics mode. The user must do this within his own program.

HGRAPHRT

Purpose

HGRAPHRT is a foreground routine that utilizes the high-resolution graphics page to provide realtime, horizontal graphing of one or more channels accessed by the background acquisition routines (ANIN, BGREAD) and the output command BGWRITE.

With the ANIN and ANOUT, you may set a list of engineering units flags, specifying an internal conversion of raw binary values to voltage, temperature or other engineering unit values.

HGRAPHRT also allows you to specify up to three windows, with one of four display modes and an optional grid. Up to 12 channels of analog or 16 channels of digital data may be displayed.

Language Syntax

QuickBASIC:

```
call hgraphrt(arrnm , varseg(widl(0)),varptr(widl(0)), displm,
varseg(minyl(0)), varptr(minyl(0)), varseg(maxyl(0)), varptr
(maxyl(0)), varseg(eufl(0)), varptr(eufl(0)), npts, wnd, grid)
```

C:

```
hgraphrt( arrnm, &widl[0], displm, &minyl[0], &maxyl[0], &eufl[0],
npts, wnd, grid);
```

FORTRAN:

```
call hgraphrt(arrnm , LOCFAR(widl(1)), displm, LOCFAR(minyl(1)),
LOCFAR(maxyl(1)),  LOCFAR(eufl(1)), npts, wnd, grid)
```

Pascal:

```
hgraphrt( arrnm, widl[0], displm, minyl[0], maxyl[0], eufl[0], npts,
wnd, grid);
```

ARRNM

(string) The ARRay NaMe parameter should be given the name of the array to be accessed by the HGRAPHRT command. From this array, the user may specify several sub-arrays (given as widths in the array) for graphing. The number of subarrays to be graphed will be determined by the number of widths included in the width list (WIDL) parameter.

WIDL

(Point to a 16 element integer array) The WIDth List parameter specifies those channels to be graphed by the HGRAPHRT command.

The WIDL parameter is an array of 16 elements. Each element of the array should be assigned the integer values of the widths to be accessed by HGRAPHRT. After the entry of the last channel to be graphed, the WIDL should be set to –1 to indicate the end of the list. The following C statements specify widths one, three, and five:

```
WIDL[0] = 1;      /* assign width 1 */
WIDL[1] = 3;      /* assign width 3 */
WIDL[2] = 5;      /* assign width 4 */
WIDL[3] = -1;     /* terminate the list */
```

Since different widths in an array correspond to the different channels from which samples were taken, the WIDL parameter is a useful way of specifying only those channels that the user wants to graph. These widths may be specified in any order.

NOTE: The WIDth List MUST be dimensioned to 16 elements even if only one channel is being graphed.

NOTE: The number of channels that can be graphed is limited by the number of windows you use (see the WND parameter, below). The channels will be limited as follow:

IBM CGA Graphics

No. of Windows Displayed	No. of Digital Channels	No. of Analog Channels
1	16	12
2	16	12
3	12	12

DATAQ Waveform Scroller Graphics

No. of Windows Displayed	No. of Digital Channels	No. of Analog Channels
1	1	1
2	2	2
3	2	2
4	4	4
5	2	2
6	4	4
7	4	4
8	8	8
9	4	4
10	8	8

DISPLM — (integer) The DISPLay Mode parameter specifies how the graph will be presented on the screen, whether scrolled, paged, overlaid (IBM style), scrolled right to left or left to right (DATAQ style). The valid values for this parameter are:

IBM CGA Graphics

SCROLL	Screen is scrolled as data is added to the display
FAST or PAGEC	The old data is cleared off the screen at the end of each screenful.
PAGEO	The screen is not cleared, and new data is graphed over the old.

DATAQ Waveform Scroller Graphics

R_SCROLL Scroll screen right to left
R.SCROLL (QuickBASIC)

L_SCROLL Scroll screen left to right
L.SCROLL (QuickBASIC)

MINYL (16 element array of float) The MINimum Y-coordinate List sets a minimum value of the Y-coordinate of the graph for each width (channel) being graphed. The minimum value for each channel may be assigned any number, as long as it is smaller than the maximum Y-coordinate value for that width.

NOTE: All 16 elements of the MINYL parameter should be assigned values even if only one channel is being graphed.

MAXYL (16 element array of float) The MAXimum Y-coordinate List sets a maximum value of the Y-coordinate of the graph for each width (channel) being graphed. The maximum value for each channel may be assigned any number, as long as it is larger than the minimum Y-coordinate value for that width.

NOTE: All 16 elements of the MAXYL parameter should be assigned values even if only one channel is being graphed.

The minimum and maximum y-coordinate values for each width are used together to scale the analog values. If, for example, values in an array range from 0 to 4095 (12-bit resolution), then set the minimum Y-coordinate value to 0 and the maximum to 4095. If most of the values lie between 1000 and 2000, then set the minimum Y value to 1000 and the maximum to 2000.

Similarly, MINYL and MAXYL for each width together scale digital values. Since these values can only be 0 or 1, the minimum Y value must be set to 0, and the maximum to 1.

EUFL (16 element integer array) The Engineering Units Flag List determines whether each channel to be graphed will be raw binary, voltage, or other engineering unit value. A separate engineering units flag must be set up for each channel, so that the first one graphed could be in volts, the second in A/D counts, and so on.

NOTE: All 16 elements of the EUFL parameter should be assigned values even if only one channel is being graphed.

See the EUF section for a summary of engineering units flag values.

NPTS (long) The Number of PoinTS parameter specifies the number of points to be graphed. When HGRAPHRT has charted this number of points, graphing will stop. Note that only one number of points can be specified; HGRAPHRT will graph the same number of points for each channel indicated.

Legal values for the NPTS parameter include any positive non-zero whole number, and **FOREVER**. Giving NPTS a value of FOREVER means that HGRAPHRT will continue graphing points until the background function is halted.

WND (integer) The number of WiNDows Flag parameter determines the number of windows to be displayed. The legal values are as follows:

IBM CGA Graphics

You may assign an integer value between one and three to the WND parameter, indicating the number of windows you wish to have displayed. The channels being graphed will be divided between the selected number of windows.

For example, if you are graphing four channels in three windows, two channels will be graphed in window 1, and one each in windows 2 and 3.

DATAQ Waveform Scroller Graphics

You may assign an integer value between one and ten to the WND parameter, indicating the window format you wish to have displayed. Refer to the following page for display organizations.

GRID (integer) The GRID parameter to specifies whether or not a grid should be drawn in the displayed window(s). The legal values are:

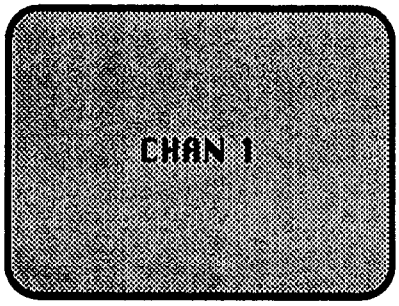
GRID	Apply grids to the display
NOGRID	Do not apply grids to the display

Notes

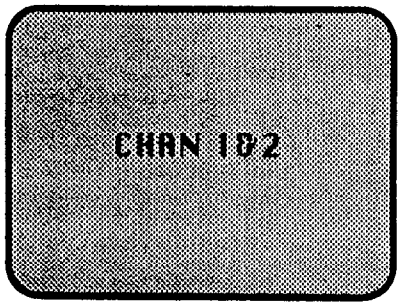
The HGRAPHRT command uses the high resolution graphics screen. HGRAPHRT does not turn on the graphics screen, so before the command is given, you must put the screen into graphics mode. After graphing is complete you can return the screen to text mode.

DATAQ SCROLLER MODULE DISPLAY ORGANIZATION

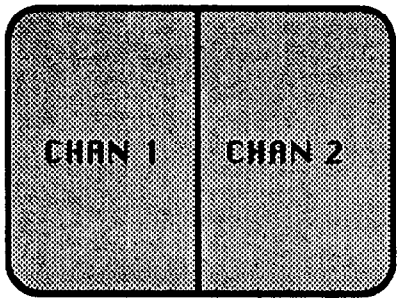
DISPLAY FORMAT 1



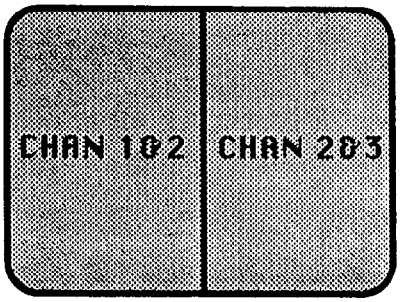
DISPLAY FORMAT 2



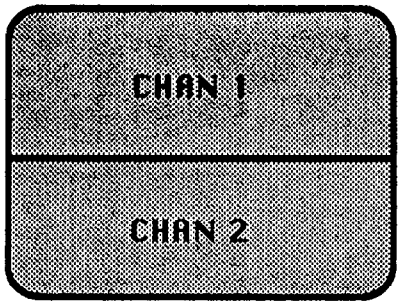
DISPLAY FORMAT 3



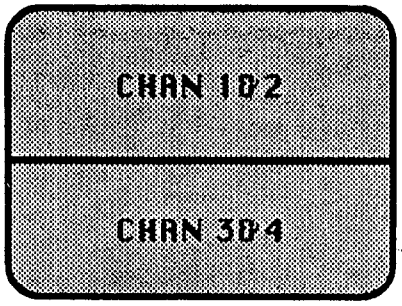
DISPLAY FORMAT 4



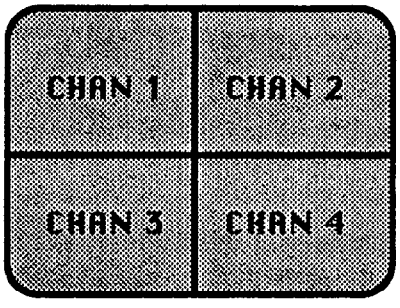
DISPLAY FORMAT 5



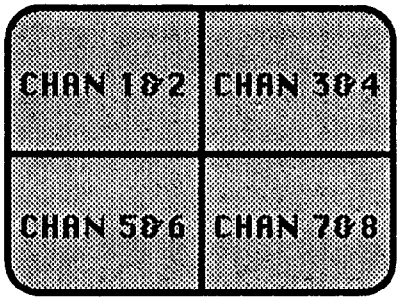
DISPLAY FORMAT 6



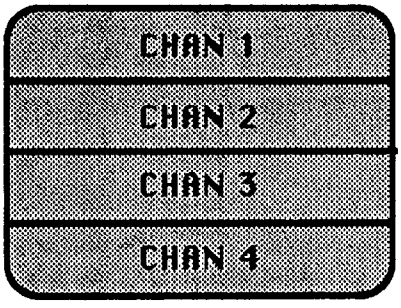
DISPLAY FORMAT 7



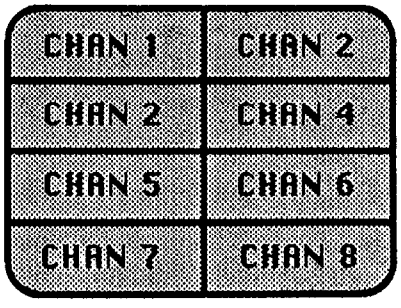
DISPLAY FORMAT 8



DISPLAY FORMAT 9



DISPLAY FORMAT 10



HREAD

Purpose

HREAD is the "handle-using" foreground routine that samples digital, analog, pulse, and frequency inputs. HREAD returns values directly to the calling language for easy access.

Language Syntax

QuickBASIC:

```
call hread(varseg(handles(0)), varptr(handles(0)),range,
varseg(vl(0)), varptr(vl(0)), euf, tm)
```

C:

```
hread (&handles[0], range, &vl[0], euf, tm);
```

FORTRAN:

```
call hread (handles, range, LOCFAR(vl(1)), euf, tm)
```

Pascal:

```
hread (handles[0], range, @vl[0], euf, tm);
```

HANDLES

(integer array) HANDLES is the array of integer values returned by the GETHANDLE command and is used to indicate which channel or port of input will be read by the HREAD command. Following the last channel "handle" in HANDLES the value 0 should be entered to mark the end of the array.

RANGE

(integer) The RANGE parameter is only used when reading pulse or frequency channels. When the input channel is not a pulse or frequency channel set RANGE = NONE.

When measuring frequency, the RANGE parameter indicates the maximum frequency expected on the channels being measured. RANGE should be assigned a value according to the table on the next page. Note that the value given to RANGE affects the length of time taken by HREAD to complete the measurement of each channel (gate time). In turn, the gate time affects the accuracy (resolution) of the measurement. Closely matching RANGE to your expected maximum frequency will result in the highest resolution possible.

When counting pulses RANGE specifies the type of pulse counting that will be used. The PIM1 can be configured to count in the normal mode or in gated mode. In normal mode all events occurring on a specified input channel will be monitored. In gated mode, two channels are monitored on the PIM1: channel n and channel n+4. Channel n represents the pulse input channel as specified in the IONL parameter. Channel n+4 acts as a gate for channel n. When channel n+4 is high HREAD will count events occurring on channel

n. When channel n+4 is low, events will not be counted. In this way the counting of events on channel n is made conditional on some other event or condition as defined by the state of channel n+4. When using the gated mode, the pulse input channel (channel n) must be channel 0, 1, 2, or 3. For more information on the PIM1 refer to the PIM1 reference manual.

The counters on the PIM2 can be read and left alone or read and reset back to 0. RANGE specifies which mode to use.

Range				
QuickBASIC	C, Pascal or FORTRAN	MAX FREQ	Gate Time	Resolution
F.62K	F_62K	62.5 KHz	1028.576 mS	1.0 Hz
F.125K	F_125K	125 KHz	524.288 mS	1.9 Hz
F.250K	F_250K	250 KHz	262.144 mS	3.7 Hz
F.500K	F_500K	500 KHz	131.072 mS	7.5 Hz
F.1M	F_1M	1 MHz	65.536 mS	15.0 Hz
F.2M	F_2M	2 MHz	32.768 mS	30.0 Hz
F.4M	F_4M	4 MHz	16.384 mS	61.0 Hz
F.8M	F_8M	8 MHz	8.192 mS	122.0 Hz

Range		
QuickBASIC	C, Pascal or FORTRAN	Pulse Mode
P1.NORMAL	P1_NORMAL	Normal pulse read on PIM1
P1.GATED	P1_GATED	Gated pulse input on PIM1
P2.DEFAULT	P2_DEFAULT	Default mode for PIM2
P2.READ.RESET	P2_READ_RESET	Read and Reset for PIM2
P2.READ.ONLY	P2_READ_ONLY	Read only mode for PIM2

Range	
NONE	any mode that is not pulse or frequency

VL (Pointer to a float, integer, or long) This variable (or array) holds the results of the readings. If more than one ioname is specified in IONL then VL must be an array with a depth at least as large as the number of entries in the IONL parameter. The values will appear in the array in the same order as the channels or ports were listed in the ioname list.

ANALOG CHANNELS – When the EUF parameter is set to C.RAW.INT or C_RAW_INT, VL will hold raw values from 0-4095 (for systems with 12 bit A/D's), 0-16383 (for systems with 14 bit A/D's), or 0-65535 (for systems with 16 bit A/D's). The EUF parameter can also be set to return values expressed as volts or engineering units.

DIGITAL CHANNELS or PORTS – When FGREAD samples a digital channel the status of that channel will be returned as a 0 or a 1(off or on; TTL low or high). If FGREAD samples a port, the status of the port will be returned as a decimal number from 0 to 255. This number can be looked at as an 8 bit binary value where the status of each bit represents the status of one channel.

Consider the following example:

FGREAD reads a port of digital input from a DIO1 module and returns the decimal value 100. When this value is converted to a binary number (01100100), the status of each bit corresponds to the status of one channel:

Ch7	Ch6	Ch5	Ch4	Ch3	Ch2	Ch1	Ch0
0	1	1	0	0	1	0	0
off	on	on	off	off	on	off	off

FREQUENCY CHANNELS – When EUF is set to C.RAW.INT, C.RAW.FLOAT, C_RAW_INT, or C_RAW_FLOAT, HREAD will return raw values from 0 to 65535 (a reading of 65535 indicates that the frequency is above the limit specified by RANGE). By setting the engineering flag to C.FREQ or C_FREQ, values can be returned as Hertz.

PULSE CHANNELS – Each entry in VL will contain an unsigned long integer which holds the input channels count value. The count will be in the range 0-4294967295. EUF should be set to C_COUNT or C.COUNT for pulse input on a PIM2 in 32-bit mode.

EUF (integer) The engineering units conversion flag is used to specify that raw values be converted to volts, frequency, or engineering units. See Chapter 2 for a complete description of the engineering units flag.

TM (integer) The trigger mode parameter allows HREAD to be triggered by the execution of another KDAC500 command. There are only two valid trigger modes for this command:

WST Wait for Singleground Trigger

NT No Trigger

Note that when WST is used, HREAD is used as a singleground function. If the user specifies WST when the background is on, a

warning error will be given. If NT is specified HREAD will operate in the foreground.

Notes

HREAD may be called as a foreground routine if a background routine is active, however, if there is not enough time left in the foreground to do the sampling without being interrupted, an interrupt will be skipped.

Cold junction compensation is automatically provided when the IONL parameter specifies a channel on an AIM3, AIM5, or AIM7 (preferred for TC's) and the EUF parameter is given a value that indicates a thermocouple is being used.

The IONAME list may include a single channel mentioned multiple times. This would allow the user to take a burst of readings from a single channel during the execution of one foreground routine.

HWRITE

Purpose

HWRITE is the "handle-using" foreground routine that writes specified values to digital, or analog, channels or ports. If the output channels are analog the user can specify the output as raw values or in engineering units. HWRITE gets its values from a variable or array passed to it from the calling language.

Language Syntax

QuickBASIC: call hwrite(varseg(handles(0)), varptr(handles(0)),varseg(vl(0)), varptr(vl(0)), euf, tm)

C: hwrite (&handles[0], &vl[0], euf, tm);

FORTRAN: call hwrite(handles, LOCFAR(vl(1)), euf, tm)

Pascal: hwrite (handles[0], @vl[0], euf, tm);

HANDLES — (Pointer to an integer array) HANDLES is the array of integer values returned by the GETHANDLE command and is used to indicate which channel or port will be accessed by the HWRITE command. Following the last channel "handle" the value 0 should be placed to mark the end of the array.

VL — (Pointer to a float, integer) This variable (or array) holds the value(s) to be written. If more than one handle is specified in HANDLES then VL must be an array with a depth at least as large as the number of entries in the HANDLES parameter. The values should appear in the array in the same order as the channels or ports were listed in the handle array.

ANALOG CHANNELS – When the EUF parameter is set to C.RAW.INT or C_RAW_INT, VL should hold raw values from 0-4095 (for systems with 12 bit D/A's) or 0-65535 (for systems with 16 bit D/A's). The EUF parameter can also be set so values can be expressed as volts or engineering units.

DIGITAL CHANNELS or PORTS – When the elements of VL are not integer numbers, they will be rounded to the nearest whole number when the values are transmitted. When HWRITE accesses digital channels the elements of VL should be assigned the values 1 or 0, indicating the on and off (high and low) status of that channel. When accessing ports, the value assigned the elements of VL must be from 0 to 255. When these numbers are converted to 8-bit binary values, the status of each bit will correspond to the status of one of the 8 channels of the digital output port.

Suppose the user wishes to to write to port B on a DIO1 corresponding element of VL is assigned the value 127.0 (01111111B), representing the status of the channels as follows:

Ch15	Ch14	Ch13	Ch12	Ch11	Ch10	Ch9	Ch8
0	1	1	1	1	1	1	1
off	on	on	on	on	on	on	on

EUF (integer) The engineering units conversion flag is used to specify that input values are specified in volts or engineering units. The EUF parameter is only used for analog channels. Set EUF = C.RAW.INT or C_RAW_INT if the outputs channels are digital or if the output values are raw D/A values. See the EUF section for a complete description of the engineering units flag.

TM (integer) The trigger mode parameter allows HWRITE to be triggered by the execution of another KDAC500 command. There are only two valid trigger modes for this command:

WST Wait for Singleground Trigger

NT No Trigger

Note that when WST is used, HWRITE will be executed immediately following the execution of a command with ST (Singleground Trigger) in its parameter list. If the user specifies WST when the background is on, a warning error will be given. If NT is specified HWRITE will operate in the foreground.

Notes

HWRITE may be called as a foreground routine while a background routine is active. However, if there is not enough time in the foreground to do the sampling without being interrupted, an interrupt will be skipped.

INTOFF

Purpose

INTOFF disables interrupts, halting execution of all background functions. This command does not clear the list of background functions. Note, however, that when interrupts are enabled again with the INTON command, INTON will reset all software and hardware timers, hence, the phase of signal input and output will be lost.

Language Syntax

QuickBASIC: call intoff

C: intoff();

FORTRAN: call intoff()

Pascal: intoff;

Notes

INTOFF disables interrupts but does not clear the list of background tasks. Clearing the background can be accomplished by the BGCLEAR command, or by KDINIT.

INTON

Purpose

INTON turns on interrupts and sets the interrupt rate according to a value specified by the user. The INTON command must be issued to implement foreground/background execution. Note, however, that the background may be set up before interrupts are enabled. In this case, background routines will not run until the INTON command is given.

Language Syntax

Quick BASIC: **call inton(ir, tu)**

C: **call inton(ir, tu)**

FORTRAN: **call inton(ir, tu)**

Pascal: **inton(ir, tu)**

IR — (integer) The Interrupt Rate parameter is used to specify the period of the interrupt. The range of legal values for IRrate is dependent on the value assigned to TUunit (see below).

TU — (integer) The Time Unit has four valid values, indicating the time units to use when interpreting the value assigned to the IRrate parameter:

HMIC	hundreds of microseconds; IR = 1–32767.
MIL	milliseconds; IR = 1–32767.
SEC	seconds; IR = 1–4400.
MIN	minutes; IR = 1–74
HZ	hertz; IR = 0–65535
MILHZ	millihertz; IR = 0–65535

Notes

INTON can be used at any time to enable interrupts (assuming that interrupts are not already enabled). By enabling interrupts after the background has been set up, INTON can be used somewhat like a trigger.

HMIC is only applicable to AT-class (ie. high speed) computers.

The maximum interrupt rate, when specified through the HZ time unit, is approximately 6000Hz for a 386-class computer, or slower on XT and AT systems.

KDCLOCK

Purpose

The KDCLOCK command allows the user to access the time and date from within a KDAC500 program. KDCLOCK will read the date from the battery-backed clock on the mother board of an AT- or PS/2 type computer, from the older 500-IBIN interface, or from a "slotless" SMARTWATCH clock/calendar module (Dallas semiconductor p.n. DS1216E).

Language Syntax

Quick BASIC: **call kdclock(time)**

C: **kdclock (&time);**

FORTRAN: **call kdclock(time)**

Pascal: **kdclock (time);**

TIME

(Pointer to time_struc) is a structure (user defined type in BASIC) that holds the results of the KDCLOCKd call. Data is returned to TIME in the following structure:

```
struct time_struc {
    int     hour        /*0-12 or 0-23 hours */
    int     minute      /*0-59 minutes */
    int     second      /*0-59 seconds */
    int     day         /*1-31 day of the month */
    int     month       /*1-12 month */
    int     year        /*00-99 last two digits of the year*/
}
```

The structure is defined in the appropriate include file for the language being used.

Programming Example

```
'$INCLUDE: 'KDAC500.BI'
CALL kdinit(BASIC.)
CALL softinit (BASIC.)
CLS
'Set up array to hold time/date info
DIM timdat AS timestruc
CALL kdclock (timdat)

PRINT timdat.hour
PRINT timdat.minute
PRINT timdat.second
PRINT timdat.month
PRINT timdat.day
PRINT timdat.year
END
```

KDINIT

Purpose

The KDINIT command initializes the KDAC500 environment by setting up memory management and other system functions. The KDINIT command returns the Series 500 hardware and the KDAC500 operating environment to a known state (its original starting state).

Language Syntax

QuickBASIC: call kdinit(BASIC.)

C: kdinit(_C_);

FORTRAN: call kdinit(FORTRAN_)

Pascal: kdinit(PASCAL_);

Language

(Integer constant) The particular value used for language depends on the programming language used to interface to KDAC500. Each language has its own include file. Within each include file all the language types are defined. It is entirely up to the user to determine which is appropriate for his needs. The purpose of telling KDAC500 the type of language it is interfacing with determines KDAC500's method of interpreting string variables. BASIC uses a string descriptor. C strings are terminated with a null byte. Pascal strings are either fixed length or the first byte of the string determines its length.

user's language	Language
QuickBASIC ver 4.x	BASIC.
Turbo Pascal ver 5.0	PASCAL_
Microsoft C (QuickC)	_C_
Borland Int'l Turbo C	_C_
Microsoft FORTRAN	FORTRAN_
Quick Pascal ver 1.0	PASCAL_

Notes

KDINIT is the command for system initialization. When executed, KDINIT resets the KDAC500 memory management system, so that access to all KDAC500 arrays created within the program are lost. KDINIT also resets all 8 software timers.

KDINIT calls an internal INTOFF and an internal BGCLEAR, completely re-initializing the background. Warning messages are turned on. KDINIT initializes the hardware according to the following:

1. All analog outputs are set to 0 volts.
2. All digital outputs are set to LOGIC 0.
3. If a DIO1 or DIO1A is in the system and port C or D is set for output, the outputs for those ports are set to logic 1

Calling KDINIT at the beginning of a program will ensure that the system is at a known, reset state at the start of every program. If more than one program is written to perform independent tasks within the same KDAC500 context (same background functions, same arrays, etc.), the KDINIT command should only be issued in the first program that will be executed. SOFTINIT may be used in the second program. Proceeding programs can check the status of previously initiated background functions, gather data from previously declared KDAC500 arrays, etc. A call to KDINIT removes all references to any KDAC500 command operation previously executed, making program-to-program interaction impossible.

AMM1A/AMM2 A/D-calibration During KDINIT

The AMM2 module performs a calibration of the A/D gain and range each time KDINIT is called. KDAC500 executes the KDINIT function automatically each time it is loaded. KDAC500 will expect an AMM module in the system if the configuration file (CONFIG.TBL) shows an AMM2 in slot 1. If the software cannot complete the calibration, it will issue an error message such as "Unable to calibrate A/D module". If this occurs, check that:

1. The data acquisition hardware is turned on.
2. The cable between the hardware and the host computer is connected.
3. An AMM module is mounted in slot 1 of the data acquisition system.

KDPAUSE

Purpose

The KDPAUSE command delays execution of the program for a specified amount of time. This time is specified as a number of seconds or minutes, depending on the value assigned to the TU (Time Units) parameter.

KDPAUSE creates a waiting loop by accessing one of the three timers on the Series 500 interface card. In this way, KDPAUSE determines the amount of time to wait before continuing with program execution.

Note that KDPAUSE is a singleground routine, i.e. it may not be used when interrupts are on.

Language Syntax

QuickBASIC: call kdpause(time, tu)

C: kdpause(time, tu);

FORTRAN: call kdpause(time, tu)

Pascal: kdpause(time, tu);

TIME — (integer) The TIME parameter specifies the number of time units to wait before continuing with foreground execution. The actual time taken by the KDPAUSE command will depend upon this parameter and upon the TU parameter that specifies the time units.

The TIME parameter can be given values of 1-59 when specifying minutes (TU = MIN), or values 1-3000 when specifying seconds (TU = SEC).

TU — (integer) The Time Units parameter specifies the units of time used to determine the actual time paused when the KDPAUSE command is executed. There are two valid values for this parameter:

SEC	Seconds
MIN	Minutes

The actual time paused is determined by the TIME parameter and the TU parameter.

Notes

The KDPAUSE command creates a waiting loop in the foreground that holds up execution of other foreground functions for a specified amount of time. KDPAUSE may not be used when interrupts are on. If the KDPAUSE command is given when interrupts are on, KDAC500 will give a warning message.

Note that once KDPAUSE becomes active, the user cannot break its execution. The system will be "locked up" until the specified time period expires.

KDTIMER

Purpose

KDTIMER enables the user to initialize (set to 0) any or all of the 8 KDAC500 timers; the timers are incremented at the occurrence of every interrupt, hence, the resolution of the timers is determined by the interrupt rate. When interrupts occur every millisecond, timers will have millisecond resolution.

Language Syntax

QuickBASIC: call kdtimer(varseg(tim(0)), varptr(tim(0)), tm, bfn)

C: kdtimer(&tim[0], tm, bfn);

FORTRAN: call kdtimer(tim, tm, bfn)

Pascal: kdtimer(tim[0], tm, bfn);

TIM (Pointer to an integer array) The TIMer Array parameter is a one-dimensional, 8-element integer array, corresponding to timers 0-7 in the KDAC500 system. Placing a specified value into an element of TIM affects the corresponding timer. The valid values and their affects on a timer are as follows:

QuickBASIC	C, PASCAL or FORTRAN	
TIMER.STOP	TIMER_STOP	Stop the corresponding timer and hold the present value.
TIMER.NUL	TIMER_NUL	Do nothing to the corresponding timer.
TIMER.START	TIMER_START	Reset the corresponding timer and enable counting.

TM (integer) The Trigger Mode allows the KDTIMER command to be triggered by another background command, or by the KDAC500 BGGO command. There are three valid strings for this parameter:

WBT	Wait on Background Trigger
WGO	Wait on BGGO
NT	No Trigger

BFN (string) The Background Function Name parameter allows the user to assign a name to the background task performed by KDTIMER. This name is necessary if the task is to be accessed by BGHALT or BGSTATUS. If not used, set BFN equal to a null string ("").

Notes

The KDTIMER command accessess 8 software timers built into the KDAC500 software system. Since these timers are incremented upon the occurrence of an interrupt, their operation depends on interrupts being enabled. For this reason, the KDTIMER and KDTIMERRD functions will not execute properly when the background is not on. If interrupts are disabled at any time after execution of a KDTIMER command, the timers will no longer be incremented.

KDTIMERRD

Purpose

KDTIMERRD enables the user to read the 8 KDAC500 timers. Once started, the timers are incremented at the occurrence of every interrupt, hence, the resolution of the timers is determined by the interrupt rate. When interrupts occur every millisecond, timers will have millisecond resolution.

Language Syntax

QuickBASIC: call kdtimerrd(varseg(timr(0)),varptr(timr(0)))

C: kdtimerrd(&timr[0]);

FORTRAN: call kdtimerrd(timr)

Pascal: kdtimerrd(timr[0]);

TIMR (Pointer to an 8-element long array) The TIMR parameter is a one-dimensional, 8-element unsigned long integer array, representing the values associated with timers 0-7. The values returned in the elements of the array will represent the elapsed time of each timer in terms of the interrupt period. To find the total elasped time for any one timer, multiply the value by the interrupt period. If interrupts have been set to occur every 10 milliseconds, and the TIMR[7] parameter returns the value 16785, the total elapsed time is the product of these two values:

16785 * 10 mS = 167850 milliseconds, or 167.85 seconds.

The timer number corresponds to the index into the TIMR array. For example, TIMR[0] holds the value of timer 0, and TIMR[1] holds the value of timer 1.

Note also that the maximum value returned to the elements of TIMR is 232 – 1(which equals 4,294,967,295), hence, the capacity of any one timer is (232 – 1) times the maximum interrupt period. When interrupts occur every millisecond, the maximum capacity of one timer will be about 50 days. If timers do overflow, they will simply start over, beginning the count from 0.

Notes

The KDTIMERRD command accesses 8 software timers built into the KDAC500 software system. Since these timers are incremented upon the occurrence of an interrupt, their operation depends on interrupts being enabled. For this reason, the KDTIMER and KDTIMERRD functions will not execute properly when the back-

ground is not on. If KDTIMERRD is executed when the background is not on, KDAC500 will return a warning error message. If interrupts are disabled at any time after execution of a KDTIMER command, the timers will no longer be incremented.

Because they are incremented once every interrupt, the KDAC500 timers do not measure absolute time, but rather indicate the number of interrupts that have occurred since the timer was started.

KDWARN

Purpose

The KDWARN command is used to suppress/reactivate KDAC500 warning error messages. Warning errors are non-fatal errors that do not halt normal program execution.

Language Syntax

QuickBASIC: call kdwarn(warnlevel)

C: kdwarn(warnlevel);

FORTRAN: call kdwarn(warnlevel)

Pascal: kdwarn(warnlevel);

WARNLEVEL (integer) The WARNing LEVEL specifies whether warning messages are to be turned on or off. There are two legal values:

WARNON enables warning messages
WARNOFF disables warning messages

Notes

The user may want to suppress non-fatal error messages in cases when errors have been identified and deemed unimportant. The ability to turn error messages on and off may be useful where a non-critical situation might produce an error. The user can suppress warning errors after that segment of the program has been executed. When the KDINIT command is executed (at the beginning of each program), warning error messages are enabled.

SOFTINIT

Purpose

The SOFTINIT command initializes KDAC500's internal software system pointers for the programming language so that KDAC500 can communicate with the language. This function is also performed by KDINIT, with the difference that KDINIT also initializes all hardware, resets any background tasks, and clears memory.

Language Syntax

QuickBASIC: call softinit (BASIC.)

C: softinit (_C_);

FORTRAN: call softinit (FORTRAN_)

Pascal: softinit (PASCAL_);

Language

(Integer constant) The particular value used for language depends on the programming language used to interface to KDAC500. Each language has its own include file. Within each include file all the language types are defined. It is entirely up to the user to determine which is appropriate for his needs. The purpose of telling KDAC500 the type of language it is interfacing with determines KDAC500's method of interpreting string variables. BASIC uses a string descriptor. C strings are terminated with a null byte. Pascal strings are either fixed length or the first byte of the string determines its length.

user's language	Language
QuickBASIC ver 4.x	BASIC.
Turbo Pascal ver 5.0	PASCAL_
Microsoft C (QuickC)	_C_
Borland Int'l Turbo C	_C_
Microsoft Quick Pascal	PASCAL_
Microsoft FORTRAN	FORTRAN_

Notes

The SOFTINIT command does not effect the hardware, nor does it have any effect on user-declared KDAC500 parameters such as IONAMEs, arrays, or background functions. SOFTINIT must be used if you re-enter KDAC500 from DOS while it is acquiring data in the background.

STPABSLOC

Purpose

The STPABSLOC command associates the specified motor's present position to an absolute location. This is an initialization command for absolute positioning that tells the motor controller "you are here".

Language Syntax

QuickBASIC: call stpabsloc(step2ion, loc)

C: stpabsloc(step2ion, loc);

FORTRAN: call stpabsloc(step2ion, loc)

Pascal: stpabsloc(step2ion, loc);

STEP2ION (string) The STEP2 IOName parameter indicates the stepper motor channel that will be used by STPABSLOC.

LOC (integer) The LOCation within the 65536 step motion space that the selected motor will be assigned to. Valid integers are –32768 to +32767.

Note

When using the STPMOVEABS command, the new location that the motor will move to will be relative to that value assigned by this command.

STPMAXSP

Purpose

This command sets the maximum speed the specified motor can achieve while processing positioning commands (to prevent motor stalling). This command does not affect the speed control command (STPSPEED).

Language Syntax

QuickBASIC: call stpmaxsp(step2ion, maxspeed)

C: stpmaxsp(step2ion, maxspeed);

FORTRAN: call stpmaxsp(step2ion, maxspeed)

Pascal: stpmaxsp(step2ion, maxspeed);

STEP2ION (string) The STEP2 IOName parameter indicates the stepper motor channel that will be used by STPMAXSP.

MAXSPEED (integer) The MAXimum SPEED a motor will be permitted to attain during a positioning operation, expressed in steps per second. Valid integers are 1 to 16000.

Notes

The maximum speed set by this command does not limit the steady-state motor speed during speed control. This command only affects the speed attainable when doing position control. Maximum speed must be set at least once, and before any positioning commands.

STPMOVEABS

Purpose

This command moves the selected motor to the specified position. The motor is presently located at the position that it was last moved to by a previous STPMOVEABS command or at the location assigned by the STPABSLOC command.

Language Syntax

QuickBASIC: call stpmoveabs(step2ion,position)

C: stpmoveabs(step2ion, position);

FORTRAN: call stpmoveabs(step2ion,position)

Pascal: stpmoveabs(step2ion, position);

STEP2ION (string) The STEP2 IOName parameter indicates the stepper motor channel that will be used by STPMOVEABS.

POSITION (integer). This integer designates the position the motor will go to. Valid integers are –32768 to +32767.

Notes

If the motor was assigned a position of 0 by a CALL STPABSLOC command and then directed to move absolute by a CALL STPMOVEABS command to position 10, the motor would move 10 steps in the CW direction.

STPMOVEREL

Purpose

This command moves the selected motor the specified number of steps in the specified direction.

Language Syntax

QuickBASIC: call stpmoverel (step2ion, steps, direction)

C: stpmoverel(step2ion, steps, direction);

FORTRAN: call stpmoverel (step2ion, steps, direction)

Pascal: stpmoverel(step2ion, steps, direction);

STEP2ION — (string) The STEP2 IOName parameter indicates the stepper motor channel that will be used by STPMOVEREL.

STEPS — (long) The STEPS parameter designates the number of steps to be taken in the specified direction. Valid values are 0 to 65535.

DIRECTION — (integer) The DIRECTION parameter designates the motor direction. Valid values are:

CW	clockwise
CCW	counter-clockwise

STPRESET

Purpose

The STPRESET command resets the specified motor to a known state.

Langauge Syntax

QuickBASIC: call stpreset (step2ion)

C: stpreset(step2ion);

FORTRAN: call stpreset (step2ion)

Pascal: stpreset(step2ion);

STEP2ION (string) The STEP2 IOName parameter indicates the stepper motor channel that will be used by STPRESET.

Notes

This command is used under two circumstances...

1. If a motor has tripped a limit condition, STPRESET purges any unexecuted commands for this motor, clears the limit condition, and reenables the motor even if the STEP2 LIMIT input is still active.
2. If STPRESET is called while the motor is running, the motor is halted immediately, without deceleration ramping, and all unexecuted commands for this motor are purged.

STPSET

Purpose

The STPSET command sets up the ramp rate on the specified STEP1 module. This function affects the acceleration/deceleration rates for all of the STEP2's associated with the specified STEP1 module.

Language Syntax

QuickBASIC: call stpset (step1ion, ramprate)

C: stpset(step1ion, ramprate);

FORTRAN: call stpset (step1ion, ramprate)

Pascal: stpset(step1ion, ramprate);

STEP1ION (string) The STEP1 IOName parameter indicates the stepper motor controller that will be accessed by this function.

RAMPRATE (integer) The RAMP RATE parameter designates the desired ramp rate used for accelerating and decelerating all motors associated with the specified STEP1 module. Valid values are:

Ramp Rate		
Quick BASIC	C, PASCAL or FORTRAN	Acceleration in Steps/Second
RR.4096	RR_4096	4096 spss
RR.4369	RR_4369	4369 spss
RR.4681	RR_4681	4681 spss
RR.5041	RR_5041	5041 spss
RR.5461	RR_5461	5461 spss
RR.5957	RR_5957	5957 spss
RR.6553	RR_6553	6553 spss
RR.7281	RR_7281	7281 spss
RR.8192	RR_8192	8192spss
RR.9362	RR_9362	9362 spss
RR.10922	RR_10922	10922 spss
RR.13107	RR_13107	13107 spss
RR.16384	RR_16384	16384 spss
RR.21845	RR_21845	21845 spss
RR.32768	RR_32768	32768 spss

Notes

This command is directed to the STEP1. It affects all STEP2's associated with the STEP1 module.

Ramp rate is an initializing function. It must be set at least once in a program, and before any stepper commands.

STPSPEED

Purpose

This command performs speed control on the specified motor channel. STPSPEED sets a selected motor to a specified speed.

Language Syntax

QuickBASIC: call stpspeed (step2ion,speed,direction)

C: stpspeed(step2ion, speed, direction);

FORTRAN: call stpspeed (step2ion,speed,direction)

Pascal: stpspeed(step2ion, speed, direction);

STEP2ION — (string) The STEP2 IOName parameter indicates the stepper motor channel that will be used by STPSPEED.

SPEED — (long) The motor SPEED parameter specifies the desired speed which the motor achieves, expressed in steps per second. Valid values are 0 to 65535.

DIRECTION — (integer) The DIRECTION parameter designates the motor direction. Valid values are:

CW	clockwise
CCW	counter-clockwise

STPSTATUS

Purpose

The STPSTATUS command returns the status of the specified motor.

Language Syntax

QuickBASIC: call stpstatus (step2ion,mocomp,limit,dir,posit)

C: stpstatus(step2ion, &mocomp, &limit, &dir, &posit);

FORTRAN: call stpstatus (step2ion,mocomp,limit,dir,posit)

Pascal: stpstatus(step2ion, mocomp, limit, dir, posit);

STEP2ION (string) The STEP2 IOName parameter indicates the stepper motor channel that will be used by STPSTATUS.

MOCOMP (Pointer to an integer) The MOtion COMPlete boolean. The value returned is:

1	if motion is complete
0	if still processing.

LIMIT (Pointer to an integer) The LIMIT has been reached boolean. The value returned is:

1	if in limit condition
0	if not in limit condition.

DIR (Pointer to an integer) The motor DIRection. The value returned is:

1	if CW
0	if CCW

POSIT (Pointer to an integer) The POSITioning mode The value returned is:

1	if in relative positioning mode
0	if in absolute positioning mode.

TRIGGER

Purpose

The TRIGGER command reads a single channel of input and determines whether the signal has met a specified condition, for example, whether it has exceeded a threshold level. TRIGGER will sample continuously until the specified condition is true; when the condition is met, TRIGGER will cause the execution of some other process associated with the TRIGGER via the trigger mode parameter. TRIGGER will operate as a foreground, background, or singleground trigger. TRIGGER supports all analog and digital input modules.

Language Syntax

QuickBASIC: call trigger (ion, thrl, thrh, chm, euf, tm, bfn, cyc)

C: trigger(ion, thrl, thrh, chm, euf, bfn, cyc);

FORTRAN: call trigger (ion, thrl, thrh, chm, euf, tm, bfn, cyc)

Pascal: trigger(ion, thrl, thrh, chm, euf, bfn, cyc);

ION (string) The IONAME parameter indicates the channel or port of input to be read by TRIGGER. The name must refer to a channel or port of analog or digital input.

THRL (double) The Threshold Low parameter sets the lower threshold level of the **analog** signal being monitored. This parameter specifies the first part of the trigger condition. If the input being monitored is a **digital port** then triggering occurs when the port value is equal to THRL. If the input being monitored is a **digital channel** then THRL is not used and should be set to 0.

THRH (double) The Threshold High parameter sets the higher threshold level of the **analog** signal being monitored. THRH must be set higher than THRL. THRH is not used by digital inputs and should be set to 0.

CHM (integer) The Change Mode parameter allows the user to specify the state of the signal being monitored that is to cause the trigger to be implemented. There are four valid values for this parameter with analog inputs:

ABOVE The signal is above or equal to THRH.

BELOW	The signal is below or equal to THRL.
BETW	The signal is between THRH and THRL but not equal to either.
NOTBETW	The signal is below or equal to THRL, or above or equal to THRH.

There are three valid values for digital input:

OFF	triggering occurs when the **channel** goes low.
ON	triggering occurs when the **channel** goes high
PORT	triggering occurs when the **port** value is equal to THRL described above.

TRIGGER will sample continuously until the specified signal condition is met, and then will trigger its associated KDAC500 command.

EUF (integer) The Engineering Units Flag can be used to pass the THRL and THRH parameters in engineering units values; these values include raw binary, voltage, and all other engineering units except temperature. For A/D counts, EUF should be set equal to C.RAW.FLOAT or C_RAW_FLOAT.

TM (integer) The Trigger Mode parameter indicates whether the TRIGGER will be used in the foreground, background, or singleground mode. There are three valid strings for this parameter:

BT	Background Trigger
ST	Singleground Trigger
NT	No Trigger

Note that if TM is set to NT, TRIGGER will function as a foreground command, halting program execution until the condition is met.

Warning: When TRIGGER is used as a foreground or singleground trigger, the expected trigger condition must be met before the program can continue or break. There is no way to escape once the TRIGGER begins, except by finding the trigger or rebooting the system.

BFN (string) The Background Function Name allows the user to name the background task being done by TRIGGER. This parameter may be used only when TRIGGER is set up as a background function, that is when the TM parameter is assigned the value: BT. If not used, set BFN equal to a null string ("").

CYC (integer) The CYCling boolean parameter. When TRIGGER monitors a signal, it is deactivated once the signal reaches the specified

threshold state. If the signal is constantly changing you may want to trigger another command only when the trigger signal meets the specified threshold requirements, i.e., you may not want the TRIGGER deactivated after the first occurence of the threshold state. If this is the case, assign the CYC parameter the value 1; this initializes the TRIGGER as a cycling trigger. Note that 1 is the only valid parameter for CYC. If you do not want a cycling TRIGGER, set CYC equal to 0.

The CYCling parameter in TRIGGER is different from the cycling parameter in the other background functions. CYC is a boolean in TRIGGER. This means it can be either TRUE (1) or FALSE (0). TRIGGER **cannot** be cycled a specific number of times as with standard background functions. TRIGGER either cycles forever or just once.

Notes

A Schmitt trigger refers to a system with two thresholds, where state transitions are measured in a way that eliminates problems associated with noise or small variations in the signal. With a standard Schmitt trigger, once a signal exceeds the high threshold it is considered high until it drops below the low threshold; once it drops below the low threshold, it is considered low until it exceeds the high threshold. This allows you to accurately assess the transition history of a signal.

WAV

Purpose

The WAV command causes a WAV1 module to output a waveform with user-specified frequency, amplitude, offset, and duty cycle. The mode (wave ON, OFF, or haver waveform) can also be specified. A WAVSETUP command must be issued before WAV.

Language Syntax

Quick BASIC: **CALL WAV(FREQ#, AMPL#, OFFS#, DUTY#, MODE%)**

C: **CALL WAV(FREQ, AMPL, OFFS, DUTY, MODE)**

FORTRAN: **CALL WAV(FREQ, AMPL, OFFS, DUTY, MODE)**

Pascal: **CALL WAV(FREQ, AMPL, OFFS, DUTY, MODE)**

FREQ — (double). The FREQ parameter specifies the desired output frequency. The frequency must lie within the limits of the FRNG parameter specified in the WAVSETUP command. If frequency autoranging is used in the WAVSETUP command, then FREQ may be any value from 0 to 200000 (duty cycle permitting). Frequency cannot be set to a negative value or an error will result.

AMPL — (double). The AMPL parameter specifies the desired peak-to-peak amplitude of the waveform. Note that the range of maximum output of the WAV1 is controlled by a switch on the module. Settings of 1V and 10V give a nominal 2V or 20V peak-to-peak. The actual maximum is 1 bit less, or 1.9951V and 19.951V, respectively. Amplitude cannot be set to a negative value or an error will result.

OFFS! — (double). The OFFS parameter specifies the desired offset of the WAV1 module waveform. When KDAC500 is initialized, the WAV1 output assumes 0V. The OFFS parameter permits this base line to be shifted up to 1V or 10V (nominal) in a positive or negative direction, depending on the position of the WAV1 output range switch.

DUTY! — (double). The DUTY parameter specifies the duty cycle of the output waveform. The legal range for DUTY is between 0 to 100, corresponding to 0-100%.

MODE — (integer). Sets the output condition of the WAV1 module:

C, Pascal	QuickBASIC	Function
WV_OUTPUT_OFF	WV.OUTPUT.OFF	sets output to 0.
WV_OUTPUT_ON	WV.OUTPUT.ON	WAV1 outputs the waveform defined by the WAVSETUP and WAV parameters.
WV_OUTPUT_HAVER	WV.OUTPUT.HAVER	WAV1 generates one cycle of the defined waveform.

Notes

The WAV command must be used in conjunction with the WAVSETUP command, Typically, a WAVSETUP command will be issued immediately before a corresponding WAV command. If only one WAVSETUP is needed in a program, the command can be placed at the beginning of the program. However, it is not possible to cluster several WAVSETUP commands for different WAV modules, frequency ranges, functions, etc. at the beginning of a program. The last WAVSETUP command will take priority.

Duty cycle and frequency place constraints on each other within any given frequency range. It is possible to request a frequency and duty cycle which are mutually exclusive, in which case an error message results. If you are using duty cycles other than 50%, AUTORANGING may be used to provide the optimum selection of frequency range for a desired duty cycle. With AUTORANGING, the maximum frequency for a 5% or 95% duty cycle is 20kHz set on the 200kHz range. Higher frequencies will require duty cycles progressively nearer 50%.

When programming haver waveforms, note that the first haver pulse will start at 0V and end at the minimum amplitude of the waveform. Subsequent haver pulses will start and end to the minimum amplitude. If may be desirable to fire one haver pulse at the beginning of a program to set the WAV1 output to a minimum.

It is possible to specify an offset and amplitude which result in the waveform being clipped. For instance, specifying 15Vp-p amplitude and 10V offset would result in a waveform with maxima and minima of 17.5V and 2.5V, respectively. The WAV module would clip at the maximum at 9.9951V. No error message results

WAVSETUP

Purpose

The WAVSETUP command configures a WAV1 module for a desired frequency range and output function. The WAVSETUP command includes an IONAME which identifies a WAV1 module in a given system slot.

Language Syntax

Quick BASIC: **CALL WAVSETUP(ION, FRNG, FUNC, 0,0)**

C: **CALL WAVSETUP(ION, FRNG, FUNC, (void far)*0)**

FORTRAN: **CALL WAVSETUP(ION, FRNG, FUNC, 0)**

Pascal: **CALL WAVSETUP(ION, FRNG, FUNC, nil)**

ION$ (string). The ION (IONAME) parameter refers to a channel name previously set up in the KDAC500 CONFIG table. For the WAV1 module, the ION parameter can consist of only one IONAME at a time.

FRNG$ (integer). The FRNG (FREQUENCY RANGE) parameter identifies the desired frequency range of the WAV1 module:

C, Pascal	QuickBASIC	Function
WV_FRNG_AUTO	WV.FRNG.AUTO	Autorange
WV_FRNG_2	WV.FRNG.2	2Hz range
WV_FRNG_20	WV.FRNG.20	20Hz range
WV_FRNG_200	WV.FRNG.200	200Hz range
WV_FRNG_2K	WV.FRNG.2K	2kHz range
WV_FRNG_20K	WV.FRNG.20K	20kHz range
WV_FRNG_200K	WV.FRNG.200K	200kHz range

FUNC$ (integer). The FUNC (FUNCTION) parameter identifies the desired waveform to be output by the WAV1 module:

C, Pascal	QuickBASIC	Function
WV_NONE	WV.NONE	DC output
WV_SINE	WV.SINE	Sine wave output
WV_SQUARE	WV.SQUARE	Square wave output
WV_TRIANGLE	WV.TRIANGLE	Triangle wave output.

Notes

The WAVSETUP command must be used in conjunction with the WAV command, and will specify the hardware setup by the WAV command. Typically, a WAVSETUP command will be issued immediately before a corresponding WAV command. If only one WAVSETUP is needed in a program, the command can be placed at the beginning of the program. However, it is not possible to cluster several WAVSETUP commands for different WAV modules, frequency ranges, functions, etc. at the beginning of a program. The last WAVSETUP command will take priority.

The autorange option for FRNG causes the software to automatically choose the best range to provide a desired frequency. (The frequency is specified as part of the WAV command.)

The parameter(s) after FUNC in the KDAC500 V1.3 WAVSETUP command references a dummy array which is reserved for future use.

User Defined Error Handling

GLERROR

GetKDACMsg

SetErrHandler

GLERROR

Purpose

User defined error handling in the KDAC500 system varies depending on the language being used. However the underlying method of implementing your own handler is straight foreward and relatively simple.

KDAC500 includes a function called GLERROR. This function displays warning and error messages and terminates your program if an error occurs. Many times it may not be necessary to terminate a program if the error can be corrected and the operation tried again. If you want to do all the error handling yourself all you need to do is write a replacement routine for GLERROR. The new GLERROR is linked in with your program so that the GLERROR in the KDAC500 library is not used. (Turbo Pascal is an exception to this. Refer to the Turbo Pascal specific section for more information).

GLERROR is never called directly. It is called by the interface functions when an error or warning has been detected.

Language Syntax

QuickBASIC: declare sub glerror (byval ernum, byval fcnum)

C: void pascal far glerror(int ernum, int fcnum);

FORTRAN: subroutine glerror (ernum,fcnum)

Pascal: procedure glerror (ernum, fcnum : integer); external;

ERNUM (Integer) This is the actual error code. Error codes range from 1 to 211. If the error code is greater than 1000 , then ernum is a warning code. Warning codes range from 1025 to 1236. Refer to the appendix for a description of the error codes. Warnings are potential errors. The meaning of the warning code is the same as the meaning of the error code. By default, however, errors cause immediate termination of your program and warnings only print a message and continue.

FCNUM (Integer) This is a code that indicates which function the error or warning occured in. The valid values for fcnum are 0 to 50. Refer to the appendix for a listing of the functions and their codes.

GLERROR in QuickBASIC

A QuickBASIC program does not directly support variables passed by value. It expects everything to be passed by reference. The bad news is the GLERROR routine is passed the ernum and fcnum by value. The good news is the VARPTR function can be used to get their values. A simple GLERROR replacement for QuickBASIC might look like this:

```
SUB GLERROR( ernum AS INTEGER, fcnum AS INTEGER)
DIM ercode AS INTEGER
DIM fcode AS INTEGER
        ercode = VARPTR( ernum )
        fcode = VARPTR( fcnum )
        ermsg$ = SPACE$( 255 )
        fcn$ = SPACE$( 255 )
        IF ercode > 1000 THEN
            ercode = ercode – 1024
            PRINT "Warning in ";
            CALL GetKDACMsg(ercode, fcode, ermsg$, fcn$)
            PRINT RTRIM$( fcn$ ); ": "; RTRIM$( ermsg$ )
        ELSE
            PRINT "Error in ";
            CALL GetKDACMsg(ercode, fcode, ermsg$, fcn$)
            PRINT RTRIM$( fcn$ ); ": ";
            BEEP
            PRINT RTRIM$( ermsg$ )
            END
        END IF
END SUB
```

GLERROR in C

A simple GLERROR replacement for C might look like this:

```
void pascal far GLERROR( int ercode, int fcode)
{
        char    ermsg[255],
                fcn[15];

        if( ercode > 1000 ) {
                ercode -= 1024;
                GetKDACMsg(ercode, fcode, ermsg, fcn);
                printf( "Warning in %s: %s\n", fcn, ermsg);
        }
        else {
                GetKDACMsg(ercode, fcode, ermsg, fcn);
                printf( "Error in %s:\007 %s\n", fcn, ermsg);
                exit( 1 );
        }
}
```

GLERROR in Pascal

Turbo or Quick Pascal will not support a direct replacement of GLERROR due to the way the Pascal Unit was created. The GLERROR function itself, however remains the same. A special function called **SetErrHandler** is used to tell the Pascal interface to use a different error handler than the default one supplied. A simple GLERROR replacement for Pascal might look like this:

```
{$F+} {Force FAR calls}
procedure GLERROR( ercode, fcode : integer );
var
        ermsg, fcn : string;
begin
        if ercode > 1000 then
        begin
            ercode := ercode - 1024;
            GetKDACMsg(ercode, fcode, ermsg, fcn);
            writeln( 'Warning in ', fcn, ': ', ermsg);
        end
        else
        begin
            GetKDACMsg(ercode, fcode, ermsg, fcn);
            writeln( 'Error in ', fcn, ': ', ermsg);
            halt( 1 );
        end;
end;
```

GLERROR in FORTRAN

```
      SUBROUTINE GLERROR [PASCAL] (ercode, fcode)
      INTEGER*2 ercode
      INTEGER*2 fcode
      include 'kdac500.fd'

      CHARACTER*256     ermsg
      CHARACTER*20      fname
      CHARACTER*1       BEEP, NULL
      PARAMETER ( BEEP = CHAR(7))
      PARAMETER ( NULL = CHAR(0))

      IF ( ercode .GT. 1000 ) THEN
         ercode = ercode - 1024
         CALL GetKdacMsg ( er code, fcode, ermsg, fname )
         WRITE (*, 100) fname(1:SCAN(fname,NULL)-1),
     +                  ermsg(1:SCAN(ermsg,NULL)-1)
      ELSE
         CALL GetKdacMsg( ercode, fcode, ermsg, fname )
         WRITE (*, 200) fname(1:SCAN(fname,NULL)-1), BEEP,
     +                  ermsg(1:SCAN(ermsg,NULL)-1)
         STOP
      END IF

100   FORMAT ( ' Warning in ', A', ': ', A )
200   FORMAT (' Error in ', A, ':', A1, A )
      END
```

GetKDACMsg

Purpos The purpose of this function is to translate error numbers into text. Once the numbers have been translated, the error message can be displayed.

Language Syntax:

QuickBASIC: call GetKDACMsg(ernum, fcnum, ermsg, fcnmsg)

C: GetKDACMsg(ernum, fcnum, &ermsg[0], &fcnmsg[0]);

FORTRAN: call GetKDACMsg(ernum, fcnum, ermsg, fcnmsg)

Pascal: GetKDACMsg(ernum, fcnum, ermsg, fcnmsg);

ERNUM (integer) The ERror NUMber parameter is the error code returned from the KDAC500 kernel. There are 211 different error codes that can be returned from the kernel. Refer to the appendix for an error code summary.

FCNUM (integer) The FunCtion NUMber parameter is the code number of the function that the error occured in. Refer to the appendix for a summary of the function codes.

ERMSG (string) The ERror MeSsaGe parameter is a string that holds the returned error text. ERMSG must be long enough to hold the returned string.

In QuickBASIC, the string should be initialized with spaces as follows:

ERMSG$ = SPACE$(255)

The string can then be trimmed with the RTRIM$ command.

In C, ermsg should be defined:

```
char        ermsg[256];
```

In FORTRAN, ermsg should be defined:

CHARACTER*256 ermsg

the string can be obtained by locating the null byte and using the substring command:

ermsg(1:SCAN(ermsg,CHAR(0))–1)

see the glerror example for FORTRAN.

And in Pascal:

ermsg : string;

FCNMSG — (string) The FunCtioN MeSsaGe parameter is a string that holds the name of the function the error occured in. FCNMSG must be long enough to hold the returned string.

Notes

The KDAC500 kernel treats strings differently depending on how the KDINIT or SOFTINIT function is called. If the language type is C or FORTRAN the returned string will be terminated by a null byte. If the language type is Pascal, the length of the string is determined by the first byte of the string. And if the language is BASIC, the string length is not modified and the user should remove trailing spaces.

KDINIT or SOFTINIT must be called with a valid language type or GetKDACMsg will return unpredictable messages, or result in a system lock-up.

SetErrHandler

Purpose

The purpose of this function is to setup the KDAC500 error handling routine in Turbo or Quick Pascal.

THIS IS A PASCAL FUNCTION ONLY!

Language Syntax

SetErrHandler(@GLERROR);

GLERROR — (far pointer to a function) This parameter points to the GLERROR routine that was described earlier.

Notes

If GLERROR is defined in an external module, for example, gloerror.tpu, the call would be:

SetErrHandler (@GLOERROR.GLERROR)

APPENDIX A
Summary of KDAC500 Commands and Parameters

KDAC500 COMMANDS

The following KDAC500 commands are shown in a generic format. See the Command Section for complete descriptions.

FOREGROUND

aninq	(arrynm, numsamp, ionl, sintv, tm)
anoutq	(arrynm, ionl, sintv, cycle, tm)
antrig	(ion, theshold, action, mode)
fgread	(ionl, range, vl(), euf, tm)
fgwrite	(ionl, vl(), euf, tm)
gethandle	(ionl, handles())
hread	(handles(), range, vl(), euf, tm)
hwrite	(handles(), vl(), euf, tm)
kdclock	(time())
kdinit	(language)
kdpause	(time, tu)
kdtimerrd	(tim())
kdwarn	(warnlevel)
softinit	(language)

BACKGROUND

bgclear	( )
bggo	(tm, bfn)
bghalt	(bfnl, tm, bfn)
bgread	(arrynm, numsamp, ionl, bintv, range, cycle,tm, bfn)
bgstatus	(bfn, stat)
bgtime	(time)
bgwrite	(arrynm, ionl, bintv, cycle, tm, bfn)
intoff	( )
inton	(ir, tu)
kdtimer	(tim(), tm, bfn)
trigger	(ion, thrl, thrh, chm, euf, tm, bfn, cycle)

ARRAY

aravail	(size)
ardel	(arrynm)
arget	(arrynm, dep1, dep2, ion, wid, extarray(), euf)
arlabel	(arrynm, labl)
arlastp	(arrynm, lp)
arload	(arrynm, dosfile)
armake	(arrynm, dep, ionl)
arptr	(arrynm, ion, wid, arptr)
arput	(arrynm, dep1, dep2, ion, wid, extarray(), euf)
arsave	(arrynm, dosfile, euf, ftype, srate, tunits)
arstatus	(arrynm, dep, wid, lp, labl)

STEPPER

stpabsloc	(stp2ion, loc)
stpmaxsp	(stp2ion, maxspeed)
stpmoveabs	(stp2ion, position)
stpmoverel	(stp2ion, steps, direc)
stpreset	(stp2ion)
stpset	(stp1ion, rmprate)
stpspeed	(stp2ion, speed, direc)
stpstatus	(stp2ion, mocomp, limit, direc, posit)

GRAPHICS

blank	
graph	(arrynm, widl(), coll(), displm, miny, maxy, mrm, res, dep1, dep2, euf)
graphrt	(arrynm, widl(), coll(), displm, miny, maxy, npts, euf)
grlabel	(labl, wind, nwin, loc1, loc2)
hgraphrt	(arrynm, widl(), displm, minyl(), maxyl(), eufl, npts, wind, grid)

KDAC500 PARAMETERS

For QuickBASIC, replace underscores (_) with periods (.). For example C_RAW_INT is C.RAW.INT in QuickBASIC.

ENGINEERING UNIT FLAGS

C_RAW_INT	C_THCU_J	C_THCU_R	C_MA420
C_RAW_FLOAT	C_THCU_K	C_RTD3175	C_FREQ
C_VOLTS	C_THCU_S	C_RTD3212	C_AIM8_C
C_MILVLT	C_THCU_T	C_STGA30	C_AIM8_D

C_MILLIA	C_THCU_B	C_AD590	C_COUNT
C_PERCENT			

TRIGGER MODES

BT	ST	WGO	NT
WBT	WST	WHT	

GRAPHICS MOVEMENTS

SCROLL	PAGEC	PAGEO	FAST
R_SCROLL	L_SCROLL		

GRAPHICS LOCATIONS

LEFT	TOP	RIGHT	CTR
BOTTOM			

GRAPHICS MODES

GRID	MAGNIFY	REDUCE	NORMAL
NOGRID			

TIME UNITS

HMIC	MIL	SEC	MIN
MIC			

STEPPER MOTOR DIRECTIONS

CCW	CW

TRIGGER DEFINITIONS

BELOW	ABOVE	BETW	NOTBETW
ON	OFF	EQUAL	PORT

PIM1 / PIM2 RANGES

NONE	F_62K	F_125K	F_250K
F_500K	F_1M	F_2M	F_4M
F_8M	P1_NORMAL	P1_GATED	P2_DEFAULT
P2_READ_RESET	P2_READ_ONLY		

STEPPER RAMP RATES

RR_4096	RR_4369	RR_4681	RR_5041
RR_5461	RR_5957	RR_6553	RR_7281
RR_8192	RR_9362	RR_10922	RR_13107
RR_16384	RR_21845	RR_32768	

ARSAVE FILE FORMATS

FT_KDAC	FT_ASCII	FT_BIN16	FT_DADISP
FT_LOT123	FT_ASYST		

WARNING LEVELS

WARNON	WARNOFF

TRG1 CONSTANTS

TRG_1M	TRG_300K	TRG_100K	TRG_30K
TRG_10K	TRG_3K	TRG_1K	TRG_300
TRG_DC	TRG_AC	TRG_BELOW	TRG_ABOVE
TRG_LATCH	TRG_FOLLOW	TRG_EVENT	

APPENDIX B
KDAC500 Error Messages

```
;*    Data Tables For Error Text
;*
;* (c) Copyright Keithley Data Acquisition and Control 1989 ;
```

Error 000	system error – unknown error code
Error 001	
Error 002	ADM1 and/or ADM2 must be removed first.
Error 003	ADM1 or 2 must be in this slot
Error 004	ADMn must go in slot 2 or 3
Error 005	AIM1 or AMMn must be defined in slot 1 before ADMn
Error 006	AMMn or AIM1 must go in slot 1
Error 007	ANREAD – system error during the read
Error 008	cannot open configuration file
Error 009	unable to open the file
Error 010	channel records deleted
Error 011	configuration information saved to disc
Error 012	directory changed
Error 013	drive changed
Error 014	drive not installed
Error 015	drive not ready
Error 016	enter a value less than 5000.0
Error 017	file deleted
Error 018	file not deleted
Error 019	file saved
Error 020	filter not allowed for this module
Error 021	for deleting channels only
Error 022	invalid engineering units flag
Error 023	illegal resistor value (64900 ohms max.)
Error 024	insert an a/d into the system
Error 025	invalid channel/port for this module
Error 026	invalid command for this module
Error 027	invalid drive specification
Error 028	invalid filter selected
Error 029	invalid gain combination
Error 030	invalid global gain – use 1,2,5,10
Error 031	invalid hardware segment in RTMDS.BIN
Error 032	invalid ioclass was detected
Error 033	invalid module specified
Error 034	invalid offset selected
Error 035	invalid range

Error 036	invalid range/gain for this module
Error 037	invalid RTMDS.BIN file
Error 038	invalid slot number
Error 039	ioname not found
Error 040	limit of 4 STP2s per STP1 reached
Error 041	limit of eight ionames
Error 042	max of channels reached
Error 043	maximum version of datafile reached
Error 044	module must be in slot 2 to 10
Error 045	module types are not the same.
Error 046	A/D module in slot 3 must be removed first
Error 047	must install correct A/D module
Error 048	must select a channel name first.
Error 049	no channel names found
Error 050	no channels on this module
Error 051	no datafile selected
Error 052	no files found in this directory
Error 053	no RTMDS channels selected
Error 054	no switch settings on this module
Error 055	not a legal conversion
Error 056	not valid for selected module
Error 05	only one ADMn allowed per system
Error 058	other error
Error 059	path not found
Error 060	printer not ready
Error 061	RTMDS.BIN file not found
Error 062	select a channel first
Error 063	select a channel to be named first
Error 064	selected file is not compatible
Error 065	setting up i/o
Error 066	STP1 must be defined before STP2
Error 067	STP2 must be placed below STP1/STP2
Error 068	STP2(s) in lower slot(s) must be removed first.
Error 069	STP2(s) must be removed before a module can be inserted in this slot
Error 070	system error allocating memory
Error 071	system error freeing memory
Error 072	system error invalid i/o call
Error 073	that accuracy not available
Error 074	this ioname is already being used, try another
Error 075	this is a major goof
Error 076	this module may not be removed
Error 077	unable to read data from disc
Error 078	unable to write data to disc
Error 079	used for analog input modules only
Error 080	used for aomn and ADMn modules only
Error 081	used for digital modules only
Error 082	loading overlay
Error 084	invalid hardware segment
Error 085	unable to calibrate A/D module
Error 100	a trigger must be set up first
Error 101	a trigger target has not been declared
Error 102	array must have a width > 1
Error 103	array width must be even to convert
Error 104	background operation no longer supported

Error 105	cannot perform function on one point
Error 106	channel list not supported
Error 107	convert – illegal range specified
Error 108	depth values out of bounds of basic array
Error 109	depth values out of bounds of source array
Error 110	engineering units conversion not allowed
Error 111	error in setting system clock.
Error 112	error while setting up clock read
Error 113	first depth value exceeds second value
Error 114	first depth value is out of range
Error 115	GRAPHRT not supported by DATAQ card
Error 116	in width or channel list
Error 117	interrupts should be disabled
Error 118	interrupts should be enabled
Error 119	invalid accuracy specified
Error 120	invalid array – unable to setup i/o
Error 121	invalid array type
Error 122	invalid call to backclear
Error 123	invalid channel record pointer
Error 124	invalid date.
Error 125	invalid digital module type
Error 126	invalid display format
Error 127	invalid display mode
Error 128	invalid gate time detected
Error 129	invalid hardware segment.
Error 130	invalid hour.
Error 131	invalid loc1 parameter
Error 132	invalid loc2 parameter
Error 133	invalid minutes.
Error 13	invalid mode selected
Error 135	invalid month.
Error 136	invalid of channels
Error 137	invalid of windows
Error 138	invalid option.
Error 139	invalid resolution
Error 140	invalid time units specified.
Error 141	invalid trigger mode selected
Error 142	invalid type in get_data.
Error 143	invalid window
Error 144	invalid windows parameter
Error 145	invalid year.
Error 146	ioname has been previously defined
Error 147	miny must be less than maxy
Error 148	must use channel 0 –> 3 in gate mode
Error 149	second depth value is out of range
Error 150	specified ioname does not match array
Error 151	system error during i/o setup
Error 152	system error during read
Error 153	system error during write
Error 154	this function is no longer supported
Error 155	too many options for this module
Error 156	unable to compute the backtime
Error 157	unable to convert from real to integer
Error 158	unable to convert integer to real
Error 159	unable to execute setclock.

Error 160	width value is out of range
Error 161	wst can be used with quick mode only
Error 162	not installed for RTM graphics
Error 163	long counter values cannot be truncated
Error 164	trigger threshold out of bounds
Error 165	timeout specified out of range
Error 166	invalid trigger channel specified
Error 167	invalid trigger channel for multi-channel aninq
Error 168	invalid trigger action
Error 169	invalid antrig call
Error 170	WHT can be used with quick mode only
Error 171	invalid trigger mode for aninq setup
Error 172	invalid number of samples requested
Error 173	KDINIT or SOFTINIT must be the first function called
Error 200	array already exists, cant be created again
Error 201	mixed iotypes or ioname not foun
Error 202	trigger mode unrecognize
Error 20	unrecognized array name
Error 204	unrecognized display mode
Error 205	unrecognized magnify/reduce mode
Error 206	unrecognized location string
Error 207	unrecognized grid mode
Error 20	unrecognized time units
Error 209	unrecognized direction
Error 210	System Kernel must be in memory
Error 211	unrecognized language – not supported

APPENDIX C
KDAC500 Function List

```
;*    Data Tables For Function Name Text
;*
;* (c) Copyright Keithley Data Acquisition and Control 1989 ;*
;*
```

Function 00	KDCLOCK
Function 01	FGREAD
Function 02	FGWRITE
Function 03	GETHANDLE
Function 04	HREAD
Function 05	HWRITE
Function 06	KDINIT
Function 07	KDPAUSE
Function 08	BGSTATUS
Function 09	KDTIMERRD
Function 10	KDWARN
Function 11	ANINQ
Function 12	BGCLEAR
Function 13	BGTIME
Function 14	BGREAD
Function 15	BGWRITE
Function 16	BGGO
Function 17	BGHALT
Function 18	INTOFF
Function 19	INTON
Function 20	KDTIMER
Function 21	TRIGGER
Function 22	ARDEL
Function 23	ARGET
Function 24	ARPUT
Function 25	ARLABEL
Function 26	ARLASTP
Function 27	ARLOAD
Function 28	ARMAKE
Function 29	ARSTATUS
Function 30	ARSAVE
Function 31	ARAVAIL
Function 32	STPABSLOC
Function 33	STPMAXSP
Function 34	STPMOVEABS
Function 35	STPMOVEREL

Function 36	STPRESET
Function 37	STPSET
Function 38	STPSPEED
Function 39	STPSTATUS
Function 44	BLANK
Function 45	GRAPH
Function 46	GRAPHRT
Function 47	HGRAPHRT
Function 48	GRLABEL
Function 49	ANOUTQ
Function 50	ANTRIG
Function 51	SOFTINIT

APPENDIX D
Engineering Unit Conversions

*EUFs shown in C, Pascal, and FORTRAN format. For QuickBASIC, underscores (_) must be replaced by periods (.). Example: C_RAW_FLOAT becomes C.RAW.FLOAT.

Transducer	EUF*	Modules Supported	Voltage Range Supported	Units	Interfacing Requirements
None	C_RAW_INT C_RAW_FLOAT	AIM1-7, AMM1,2 AOM1-4,PIM1	all	Raw integer Raw float	None, all gains supported
None	C_VOLTS	AIM1-7, AMM1,2, AOM1,2,4	all	Raw>Volts Volts>Raw	None, all gains supported
None	C_MILVLT	AIM1-7,AMM1,2, AOM1-4	all	Raw>mV mV>RAW	None, all gains supported
None	C_MICVLT	AIM1-7,AMM1,2, AOM1-4	all	Raw>μ V μ V>Raw	None, all gains supported
None	C_MILLIA	AIM1-4,AMM1,2, AOM3	all	Raw>ma ma>Raw	For current measurement, install path-to-ground or input high-to-low shunt resistor and enter resistor value in KDAC500 config table. Typical value 1K ohm.
None	C_PERCENT	AIM1-7,AMM1,2, AIM1-4	all	% of full scale of A/D	None, all gains supported
Thermocouple J	C_THCU_J	AIM3,5,7	all	° C	Path-to-Ground Resistor and x100 local gain required with AIM3.
Thermocouple K	C_THCU_K	AIM3,5,7	all	° C	Path-to-Ground Resistor and x100 local gain required with AIM3.
Thermocouple S	C_THCU_S	AIM3,5,7	all	° C	Path-to-Ground Resistor and x100 local gain required with AIM3.
Thermocouple T	C_THCU_T	AIM3,5,7	all	° C	Path-to-Ground Resistor and x100 local gain required with AIM3.
Thermocouple E	C_THCU_E	AIM3,5,7	all	° C	Path-to-Ground Resistor and x100 local gain required with AIM3.
Thermocouple B	C_THCU_B	AIM3,5,7	all	° C	Path-to-Ground Resistor and x100 local gain required with AIM3.
Thermocouple R	C_THCU_R	AIM3,5,7	all	° C	Path-to-Ground Resistor and x100 local gain required with AIM3.
RTD	C_RTD3175	AIM6	±2.5V ±5.0V ±10.0V	° C	IONAME must invoke alpha=0.00385 RTD mode with local gain of x50 (STRG%=1)

Transducer	EUF%	Modules Supported	Voltage Range Supported	Units	Interfacing Requirements
RTD	C_RTD3212	AIM6	±2.5V ±2.5V ±10.0V	°C	IONAME must invoke alpha=0.00392 mode with local gain x50 (STRG%=1)
Strain Gage, ±30mV	C_STGA30	AIM6	±2.5V ±5.0V ±10.0V	-100 to +100%	IONAME must invoke Strain Gage mode with gain of x166 (STRG%=2)
Strain Gage, +100mV	B_STGA100	AIM6	±2.5V ±5.0V ±10.0V	-100 to +100%	IONAME must invoke Strain Gage mode with gain of x50 (STRG%=3)
AD590/AC2626 Sensors	C_AD590	AIM1,3,5,6, AMM1,2	all	°C	1000 ohms shunt resistor (210 for AIM6)
4-20mA Current Loop Inputs	C_MA420	AIM1-4, AMM1,2	0-5.0V	Raw>mA	Local Gain of 1 with a 250 ohm shunt typical
4-20mA Current LoopOutput	C_MA420	AOM3	±10.0V 0-5.0V 0-10.0V	mA>Raw	
None	C_FREQ	PIM1	all	Hz	8MHz maximum frequency input
Strain Gage	C_AIM8_C	AIM8	all	None	Sets calibration factor for a strain gage
Strain Gage	C_AIM8_D	AIM8	all	Units of Measure parameters or cal factor in IONAME	Specifies that data retrieve will be in units of measure corresponding to units of the strain gage cal factor
LVDT/RVDT	C_AIM9_D	AIM9	all	cal factor in the IONAME	Specifies that data retrieved will be in units of measure corresponding to units of the transducer cal factor.
None	C_COUNT	PIM2	32 Bit		Specify to read 32-bit data value from PIM2 counters in 32-bit mode.

Note: All engineering units conversion supports the 12-bit, 14-bit, and 16-bit resolution.

APPENDIX E
Running KDAC500/M Under the Microsoft QuickBASIC Environment

Keithley's KDAC500/I software for BASICA and GWBASIC interpreters is currently bundled with the Models 500A, 500P, and 575 data acquisition systems. KDAC500 is also available as an option for various Microsoft and Borland compilers as KDAC500/M and KDAC500/B, respectively. KDAC500/M is compatible with Microsoft BASIC PDS, Quick BASIC, Quick C, Quick Pascal, C, and Fortran, while KDAC500/B is compatible with Borland Turbo C and Turbo Pascal.

The similarities between KDAC500 and its predecessors, Soft500 and Quick500, are evident in the structure and use of the software. KDAC500/M and Quick500 are perhaps most alike, but do differ in an important area. Quick500 was originally designed so that programs could be written, executed, and compiled under the QuickBASIC environment. Conversely, the KDAC500 manual instructs that when KDAC500/M is used with QuickBASIC, programs may be written and compiled under the QuickBASIC environment, but should be executed from the DOS command line using KDAC500's KRUN or KLOAD utilities.

The reason for not running KDAC500/M programs under the QuickBASIC environment is that QuickBASIC and KDAC500 consume nearly all the available system memory, leaving only a few kbytes for data arrays. During an installation of KDAC500, the user specifies the desired array memory size. While 300k or more may be shown as available, only 16-20k may actually be available when QuickBASIC and KDAC500 are both loaded (this was also the case with Quick500). Further, some types of errors can cause KDAC500 and QuickBASIC to abort and return to the DOS prompt. This is not a true "crash" since the user can rerun QuickBASIC and KDAC500. However, it does behoove the user to save work often.

Despite these limitations, it can be useful to write and execute KDAC500/M programs under the QuickBASIC environment. One example is developing and debugging programs where a small array memory is sufficient to prove out a technique. A second example is a KDAC500 program which uses only foreground commands, and thus requires no KDAC500 array memory at all.
The following steps will aid you in installing KDAC500/M and QuickBASIC 4.5 so that programs can be run under the QuickBASIC environment. They presume the use of Microsoft QuickBASIC 4.5. Refer to the QuickBASIC documentation for more information on installing and running QuickBASIC.

1. Using the installation instructions in the KDAC500 manual, install KDAC500/M for a small array memory, e.g. about 16k for a 640k system. Also indicate the address and desired interrupt method for your interface card. SAVE the installation and QUIT installation.

2. From the DOS comamnd line, run CONFIG.EXE. If you have a Model 570, issue the command "CONFIG 570". If you are using a Model 575, issue the command "CONFIG 575-1" or "CONFIG 575-2" according to the A/D module (AMM1A or AMM2) in slot 1. If you have a Model 500A or 500P, simply issue "CONFIG", and then indicate your modules and their slot locations.

3. If you have a specific application in mind at this point, create IONAMEs for the channels you will be using in your application program. If not, skip this step for now.

4. Save the CONFIG table.

5. Install QuickBASIC 4.5 to the same directory containing KDAC500/M. This is the easiest way to assure that all the necessary QuickBASIC files will be accessible to KDAC500/M. You may also have separate directories for KDAC500/M and QuickBASIC as long as the DOS path, environment variables, etc. are set up properly. See the QuickBASIC documentation for details.

6. Make the necessary KDAC500/M Quick library. Enter the following command at the DOS command line:

```
LINK /QU KDAC500.LIB,,NUL,BQLB45
```

7. Start KDAC500/M. Enter the following at the DOS command line:

```
KRUN [drive:\path\]QB /L KDAC500.QLB
```

If QB.EXE is in another directory, include [drive:\path\] as part of the QB file name. You may want to create a batch file containing this command to simplify running KDAC500 and QuickBASIC. After a few moments, you should see the QuickBASIC environmnent on-screen.

8. Enter the following test program. Note the first three lines. These must be included in all KDAC500/QuickBASIC programs.

```
'$INCLUDE: 'KDAC500.BI'
CALL KDINIT(BASIC.)
CALL SOFTINIT(BASIC.)
CALL INTON(100, MIL)
T!=TIMER: WHILE TIMER-T!<3: WEND
CALL INTOFF
```

9. To run the program, press <Alt R> to invoke the QuickBASIC "Run" menu, and then "S" to start the program. The the "ON LINE" lamp on the data acquisition hardware should flash briefly, after which the program will terminate.

10. To compile the program (.EXE file), press <Alt R> to invoke the QuickBASIC "Run" menu, and then "X" to compile the program. Follow the various QB prompts and supply a file name. Compile the file as ".EXE requiring BRUN45.EXE".

11. To run the .EXE file from the DOS command line, first leave QuickBASIC with <Alt F> and then "X". Run the program by issuing "KRUN <your file>. See the KDAC500 manual for information on KRUN and KLOAD.

12. You can now develop your program using the remaining KDAC500 and QuickBASIC commands. Add IONAMEs to the CONFIG table if you have not done so already. Rerun KDAC500 as you did in step 7. Since an error may send you back to DOS, save your work often. If there will be several executable files on one disk, compile your programs as ".EXE requiring BRUN45.EXE". Compiling "Stand-alone" will produce longer .EXE files which ultimately use more much disk space.

Suggestions for Designing KDAC500 Programs

Producing robust test programs is easy with KDAC500/M and QuickBASIC, although there are a few considerations which will aid in the process. These suggestions apply to all versions of KDAC500, not just QuickBASIC and KDAC500/M.

The first suggestion concerns the order of commnads in a program. If a program uses background read (BGREAD) or background write (BGWRITE) commands, these commands should all be placed before the CALL INTON command which starts them. When interrupts are enabled, KDAC500 allots a specific amount of background time to handle the tasks listed before CALL INTON. If interrupts are enabled and then the background calls are issued, KDAC500 may not be able to deal with the added background overhead, and problems will result.

The second suggestion concerns embedding background commands in a loop. Under the right circumstances, the background processing time can increase with each pass through the loop, resulting in foreground routines running more and more slowly. This will cause a noticeable slowing of screen updates, for example. The slowing will become more severe with each pass through the loop until

the system either aborts the interrupts or crashes entirely. If it is absolutely necessary to run background routines in a loop, try to include a CALL KDINIT command at the beginning of the loop, and then reissue the background commands. You will have to pay close attention to the handling of data, since a CALL KDINIT will wipe out any KDAC500 array currently in memory. It is best to avoid this type of programming altogether.

Third, interrupts should not repeatedly be turned on and off in a program. An example is a FOR-NEXT loop containing CALL INTON and CALL INTOFF commands.

Under some circumstances, KDAC500 may fail to trap a an interrupt, which may then get through to DOS. If this occurs, the computer may crash. While it may take thousands of passes through the loop for this to occur, it is best to avoid the practice entirely.

In any of these situations, it is helpful to use KDAC500's BGTIME command to keep track of the background processing time. In a well-designed program, the background time should remain stable for a given INTON rate.

Example Programs for KDAC500/M and Microsoft QuickBASIC

The following programs illustrate the use of KDAC500/M with QuickBASIC Version 4.5. Note that some of the programs involve several commands. In these cases, each command is used in a short block of code which can be used as a model, or lifted for inclusion in your own programs.

HEADER.BAS	Shows a standard header which should be included at the beginning of KDAC500/QuickBASIC programs to initialize the software and hardware.
ANALOGIO.BAS	Shows the use of FGREAD, FGWRITE, BGREAD, and BGWRITE for analog I/O. Each command is demonstrated in a short block of code which can be lifted and used in other programs. BGSTATUS, ARGET, ARMAKE, and ARGET are also demonstrated.
ANINQ.BAS	Shows the use of ANINQ for high-speed acquisition, as well as ARGET, and ARSAVE for inspecting and saving data.
BGREAD.BAS	Shows the use of BGREAD, HGRAPRT, AND GRLABEL for thermocouple measurements and real-time graphing.
DIGIO.BAS	Shows the use of FGREAD, FGWRITE, BGREAD, and BGWRITE for digital I/O. Each command is demonstrated in a short block of code which can be lifted and used in other programs. BGSTATUS, ARGET, ARMAKE, and ARGET are also demonstrated.
KDCLOCK.BAS	Shows the use of KDCLOCK command.
WAV1.BAS	Shows programming of the WAV1 module using WAVSETUP and WAV commands.
AMMPOKE.BAS	Shows how to use the AMM1A or AMM2 modules by PEEKing and POKEing the command registers. Program is useful for writing a low-level driver for the AMM modules.

```
*************************************************************************
*
*   SUMMARY - SET-UP INFORMATION FOR RUNNING KDAC500/M UNDER THE
*   MICROSOFT QUICKBASIC 4.5 ENVIRONMENT.
*
*************************************************************************
```

To run KDAC500/M under the QuickBASIC environment, follow these instructions:

1. Install KDAC500 for a small array memory, e.g. 16k for a 640k system.

2. Run CONFIG and create IONAMES as the application requires.

3. Copy QuickBASIC to the KDAC500/M directory. This is the "easy" way. You may also have separate directories for KDAC500 and QuickBASIC as long as the DOS path, environment variables, etc. are set up properly. This is a bit more complex, and may make it more difficult to discern any QuickBASIC problems from KDAC500 problems which may occur during the initial set-up of KDAC500.

   REQUIRED QUICKBASIC FILES:

BC.EXE	BCOM45.LIB	QB.INI
BQLB45.LIB	BRUN45.EXE	QB.LIB
BRUN45.LIB	LINK.EXE	QB.QLB
QB.BI	QB.EXE	

4. Make the necessary Quick library by executing at the DOS command line:

```
LINK /QU KADC500.LIB,,NUL,BQLB45
```

5. Start KDAC500 by executing at the DOS command line (this command may be placed in a batch file for convenience. Drive and path to QB are optional):

```
KRUN QB /L KDAC500.QLB
```

6. Include as the first executable line of code in the program:

```
'$INCLUDE: 'KDAC500.BI'
```

NOTE: KDAC500 errors encountered in this mode of operation may cause BASIC to abort back to DOS. This is not a "crash". You may restart KDAC500 and reload your program. Save your program often to guard against losing your most recent changes.

NOTE: If you are upgrading from KDAC500/I to a compiler version, you must keep the versions of KDAC500 separate. It is best to install them in different directories.

```
'*************************************************************************
'*
'*  HEADER.BAS
'*
'*  STANDARD PROGRAM HEADER FOR KDAC500/M WITH QUICKBASIC.     10/15/90
'*
'*  If the '$INCLUDE:...." line is ommited, error "Unrecognized Language"
'*  will result.
'*
'*************************************************************************

      '$INCLUDE: 'kdac500.bi'

      CALL kdinit(BASIC.)
      CALL softinit(BASIC.)
      CLS

' Your program goes here.....

' For example ...

      CALL inton(10, MIL)
      FOR t = 1 TO 10000: NEXT
      CALL intoff

END
```

```
'*************************************************************************
'*
'* ANALOGIO.BAS
'*
'* EXAMPLE KDAC500/M AND QUICKBASIC PROGRAM FOR ANALOG I/O.       10/15/90
'*
'* IONAMEs in CONFIG.TBL are SL1_CH0 for input and SL5_CH0 for output.
'*
'*************************************************************************

' HEADER

      '$INCLUDE: 'kdac500.bi'
      CALL kdinit(BASIC.)
      CALL softinit(BASIC.)

      DEFINT A-Z
      CLS

'-------------------------------------------------------------------------

' Example of analog input foreground command (AMM2 in slot 1)

      DIM voltsin!(1): DIM countsin!(1)
      voltsin!(0) = 0: countsin!(0) = 0

      WHILE INKEY$ = ""
            CALL fgread("sl1_ch0", NONE, VARSEG(voltsin!(0)), VARPTR(voltsin!(0)), C.VOLTS, NT)
            LOCATE 1, 1: PRINT "Voltage input from AMM2, channel 0 = "; voltsin!(0)
            CALL fgread("sl1_ch0", NONE, VARSEG(countsin!(0)), VARPTR(countsin!(0)), C.RAW.FLOAT, NT)
            LOCATE 3, 1: PRINT "A/D Counts input from AMM2, channel 0 = "; countsin!(0)
            LOCATE 24, 1: PRINT "Press any key to continue";
      WEND

'-------------------------------------------------------------------------

' Example of foreground commands for analog output (AOM1 in slot 5)

      DIM voltsout!(1)
      voltsout!(0) = 3.3
      CALL fgwrite("sl5_ch0", VARSEG(voltsout!(0)), VARPTR(voltsout!(0)), C.VOLTS, NT)
      LOCATE 5, 1: PRINT "Slot 5 channel 0 output is 3.3V"
      LOCATE 24, 1: PRINT "Press any key to continue";

      WHILE INKEY$ = "": WEND

      voltsout!(0) = -5
      CALL fgwrite("sl5_ch0", VARSEG(voltsout!(0)), VARPTR(voltsout!(0)), C.VOLTS, NT)
      LOCATE 7, 1: PRINT "Slot 5 channel 0 output is -5.0V"
      LOCATE 24, 1: PRINT "Press any key to continue";

      WHILE INKEY$ = "": WEND
      CLS

'-------------------------------------------------------------------------

' Example of background command for analog input. Acquire 10 analog readings
' at 5 readings/second and store them in an array named "inarray%"

      CALL bgread("inarray%", 10!, "sl1_ch0", 1, NONE, 1, NT, "anin")

      PRINT : PRINT "Press any key to start background analog input..."
      WHILE INKEY$ = "": WEND

      CALL inton(200, MIL)
```

```
        PRINT : PRINT "Taking data and checking status..."

        stat% = -1
        WHILE stat% <> ST.DONE
              CALL bgstatus("anin", stat%)
              LOCATE 6, 1: PRINT "Status = "; stat%
        WEND

        CALL intoff

        PRINT : PRINT "Analog input is completed."

' Read voltage values back from array with ARGET

        DIM results!(1)
        results!(0) = 0

        FOR depth& = 1 TO 10
              CALL arget("inarray%", depth&, depth&, "sl1_ch0", NONE, VARSEG(results!(0)), VARPTR(results!(0)), C.VOLTS)
              PRINT "Volts at depth "; depth&; " = "; results!(0)
        NEXT depth&

        PRINT

' Do it again as A/D counts

        FOR depth& = 1 TO 10
              CALL arget("inarray%", depth&, depth&, "sl1_ch0", NONE, VARSEG(results!(0)), VARPTR(results!(0)), C.RAW.FLOAT)
              PRINT "A/D counts at depth "; depth&; " = "; results!(0)
        NEXT depth&

        LOCATE 24, 1: PRINT "Press any key to continue";
        WHILE INKEY$ = "": WEND
        CLS

'-------------------------------------------------------------------------------

' Example of background command for analog output.

' First make a BASIC array containing a 1000-point ramp from 0 to 10

        DIM basarray!(1000)
              FOR T! = 0 TO 999
              basarray!(T!) = T! / 100
        NEXT T!

' Make KDAC500 array...

        CALL armake("outarray%", 1000&, "sl5_ch0")
        CALL arput("outarray%", 1&, 1000&, "sl5_ch0", NONE, VARSEG(basarray!(0)), VARPTR(basarray!(0)), C.VOLTS)

' Set up background write for 10 cycles through the array...

        CALL bgwrite("outarray%", "sl5_ch0", 1, 10, NT, "anout")

        PRINT : PRINT "Press any key to start background output..."
        WHILE INKEY$ = "": WEND

        CALL inton(1, MIL)

        PRINT : PRINT "Outputting 10 sawtooth pulses..."

        stat% = -1: lp& = 0

        WHILE stat% <> ST.DONE
              CALL bgstatus("anout", stat%)
```

```
            CALL arlastp("outarray%", lp&)
            LOCATE 6, 1: PRINT "Status = "; stat%
            LOCATE 8, 1: PRINT "Last point accessed  = "; lp&
      WEND

      CALL intoff

'-------------------------------------------------------------------------

END
```

```
'*****************************************************************************
'*
'* ANINQ.BAS
'*
'* EXAMPLE KDAC500/M AND QUICKBASIC PROGRAM FOR ANINQ.              10/15/90
'*
'* IONAME in CONFIG.TBL is SL1_CH0 for input.
'*
'*****************************************************************************

' HEADER

      '$INCLUDE: 'kdac500.bi'

      CALL kdinit(BASIC.)
      CALL softinit(BASIC.)

      CLS

'-----------------------------------------------------------------------------

' Call ANINQ to take 100 points into an array named "data.array%"

      CALL aninq("data.array%", 100!, "sl1_ch0", 0, NT)

' Look through the array, one point at a time, with ARGET

      LOCATE 1, 1
      DIM value!(1)
      value!(0) = 0

      FOR depth& = 1 TO 100
            CALL arget("data.array%", depth&, depth&, "sl1_ch0", 1, VARSEG(value!(0)), VARPTR(value!(0)), c.volts)
            PRINT "Data value at depth "; depth&; " = "; value!(0)
      NEXT depth&

' Transfer entire KDAC array to a BASIC array with ARGET, and then look
' through the BASIC array

      DIM allvalues!(100)

      CALL arget("data.array%", 1&, 100&, "sl1_ch0", 1, VARSEG(allvalues!(0)), VARPTR(allvalues!(0)), c.volts)

      ' Look through the BASIC array, one point at a time. Depths of 0 to
      ' 99 are used because the BASIC array begins with element 0.

      LOCATE 1, 1

      FOR depth& = 0 TO 99
            PRINT "Data value at depth "; depth&; " = "; allvalues!(depth&)
      NEXT depth&


' Save the KDAC500 array in KDAC, ASCII, and LOTUS formats using ARSAVE

' NOTE: The AMM2 module was run at maximum speed, which is 50kHz (62.5kHz
' for the AMM1A).

      CALL arsave("data.array%", "KDACFRMT.DAT", c.volts, FT.KDAC, 50000!, HZ)
      CALL arsave("data.array%", "ASCIFRMT.DAT", c.volts, FT.ASCII, 50000!, HZ)
      CALL arsave("data.array%", "123FRMT.DAT", c.volts, FT.LOT123, 50000!, HZ)

END
```

```
'*************************************************************************
'*
'* BGREAD.BAS
'*
'* EXAMPLE BGREAD PROGRAM                                10/15/90
'*
'* Purpose: Demonstrates KDAC500 Background Read (BGREAD) for thermocouple
'*          measurement, and graphs data.
'*
'* IONAMEs "J_TC_0", "J_TC_1", and "CJR" are set up in the CONFIG table for
'*          slot 3, channels 0, 1, and 32, respectively.
'*
'*************************************************************************

' HEADER

' Intitialize system

      '$INCLUDE: 'KDAC500.BI'

      CALL kdinit(BASIC.)
      CALL softinit(BASIC.)

      CLS

'-------------------------------------------------------------------------

' Set up some arrays containing parameters needed for the HGRAPHRT command

' Specify array widths to be graphed. We want to look at 2 and 3 because the
' cold junction reference takes position 1 in the KDAC500 array.

      DIM widl(16) AS INTEGER
      widl(0) = 2: widl(1) = 3

' Presuming a test at room temperature...
' Specify the lower limit of the graph as 15 degrees C

      DIM miny(16) AS SINGLE
      minyl(0) = 15: minyl(1) = 15

' Specify the upper limit of the graph as 25 degrees C

      DIM maxyl(16) AS SINGLE
      maxyl(0) = 30: maxyl(1) = 30

' Specify the engineering units flags for the thermocouple types (J)
' Change last letter of each EUF to your TC type

      DIM eufl(16) AS INTEGER
      eufl(0) = c.thcu.j: eufl(1) = c.thcu.j

'-------------------------------------------------------------------------

' Set up the BGREAD command to take 1000 readings.
' (Note that parmameter list is all one line)

      CALL bgread("array1%", 1000&, "cjr j_tc_0, j_tc_1", 1, none, 1, NT, "readstatus")

'-------------------------------------------------------------------------

' Change to 640x200 color graphics screen and set up the graph command

      SCREEN 2
      LOCATE 1, 1: PRINT "Press <Escape> to quit..."
```

```
' Label the graph axis

      CALL GRLABEL("30", 1, 1, left, top)
      CALL GRLABEL("15", 1, 1, left, bottom)

'-------------------------------------------------------------------------

' Turn on interrupts

      CALL inton(100, mil)

      CALL hgraphrt("array1%", VARSEG(widl(0)), VARPTR(widl(0)), scroll, VARSEG(minyl(0)), VARPTR(minyl(0)),
            VARSEG(maxyl(0)), VARPTR(maxyl(0)), VARSEG(eufl(0)), VARPTR(eufl(0)), 1000&, 1, grid)

' The hgraphrt command will execute until the data array has been graphed completely.

'-------------------------------------------------------------------------

' Now that it's all over, turn off interrutpts.

      CALL intoff

END
```

```
'*************************************************************************
'*
'* DIGIO.BAS
'*
'* EXAMPLE KDAC500/M AND QUICKBASIC PROGRAM FOR DIGITAL I/O.       10/15/90
'*
'* IONAMEs in CONFIG.TBL are DIGI_0 for input via channel 0 of a DIO1 module,
'* and DIGO_16 for output via channel 16 of the same module.
'*
'*************************************************************************

' HEADER

      '$INCLUDE: 'kdac500.bi'

      CALL kdinit(BASIC.)
      CALL softinit(BASIC.)

      DEFINT A-Z

      CLS

'-------------------------------------------------------------------------

' Example of foreground command for digital input...

      DIM inval%(1)
      inval%(0) = 0

      WHILE INKEY$ = ""
            CALL fgread("digi_0", NONE, VARSEG(inval%(0)), VARPTR(inval%(0)), C.RAW.INT, NT)
            LOCATE 1, 1: PRINT "Digital input from DIO1, channel 0 = "; inval%(0)
      WEND

'-------------------------------------------------------------------------

' Example of foreground command for digital output ...

      DIM outval%(1)
      outval%(0) = 1

      CALL fgwrite("DIGO_16", VARSEG(outval%(0)), VARPTR(outval%(0)), C.RAW.INT, NT)
      LOCATE 3, 1: PRINT "Output channel is HIGH"
      LOCATE 4, 1: PRINT "Press any key to toggle output..."

      WHILE INKEY$ = "": WEND

      outval%(0) = 0

      CALL fgwrite("DIGO_16", VARSEG(outval%(0)), VARPTR(outval%(0)), C.RAW.INT, NT)
      LOCATE 3, 1: PRINT "Output channel is LOW "
      LOCATE 4, 1: PRINT "Press any key to continue...      "

      WHILE INKEY$ = "": WEND

'-------------------------------------------------------------------------

' Example of background command for digital input. Acquire 10 digital
' readings and store them in an array named "digin%"

      CALL bgread("inarray%", 10!, "digi_0", 1, NONE, 1, NT, "digin.job")

      PRINT : PRINT "Press any key to start background digital input..."
      WHILE INKEY$ = "": WEND

      CALL inton(100, mil)
```

```
        PRINT : PRINT "Taking Data..."

        stat% = -1

        WHILE stat% <> ST.DONE
              CALL bgstatus("digin.job", stat%)
        WEND

        CALL intoff

        PRINT : PRINT "Digital input is completed."

' Read values back from array with ARGET

        DIM results(1) AS INTEGER

        FOR depth& = 1 TO 10
              CALL arget("inarray%", depth&, depth&, "digi_0", NONE, VARSEG(results(0)), VARPTR(results(0)), C.RAW.INT)
              PRINT "Result at depth "; depth&; " = "; results(0)
        NEXT depth&

'-------------------------------------------------------------------------------

' Example of background command for digital output.
' First make a BASIC array containing a "1" and a "0"

        DIM outvals(2) AS INTEGER
        outvals(0) = 0: outvals(1) = 1

' Make KDAC500 array...

        CALL armake("outarray%", 2&, "DIGO_16")
        CALL arput("outarray%", 1&, 2&, "DIGO_16", NONE, VARSEG(outvals(0)), VARPTR(outvals(0)), C.RAW.INT)

' Set up background write for 20 cycles through the array...

        CALL bgwrite("outarray%", "DIGO_16", 1, 20, NT, "digout.job")

        PRINT : PRINT "Press any key to start background output..."
        WHILE INKEY$ = "": WEND

        CALL inton(500, mil)

        PRINT "Outputting 20 square wave pulses..."

        stat% = -1
        WHILE stat% <> ST.DONE
              CALL bgstatus("digout.job", stat%)
        WEND

        CALL intoff

'-------------------------------------------------------------------------------

END
```

```
'*************************************************************************
'*
'* KDCLOCK.BAS
'*
'* EXAMPLE PROGRAM FOR KDCLOCK COMMAND                10/15/90
'*
'*************************************************************************

' HEADER

' Intitialize system

      '$INCLUDE: 'KDAC500.BI'

      CALL kdinit(BASIC.)
      CALL softinit(BASIC.)

      CLS

'-------------------------------------------------------------------------

' Set up array to hold time/date info

      DIM timdat AS timestruc

      CALL kdclock(timdat)

      PRINT timdat.hour
      PRINT timdat.minute
      PRINT timdat.second
      PRINT timdat.month
      PRINT timdat.day
      PRINT timdat.year
END
```

```
'*************************************************************************
'*
'* WAV1 Example Program                                    10/15/90
'*
'* Demonstrates use of the WAVSETUP and WAV calls to controll a WAV1 module.
'* The IONAME "Wav1" has been set up in the CONFIG table.
'*
'* NOTE: The WAV1 requires that either:
'*       a. An AMMx module be installed in slot 1 of the system, or
'*       b. A resistor be added to the WAV1 to supply a reference voltage.
'*
'*       See the WAV1 manual for details.
'*
'*************************************************************************

' HEADER

      '$INCLUDE: 'kdac500.bi'

      CLS
      CALL kdinit(BASIC.)
      CALL softinit(BASIC.)

'-------------------------------------------------------------------------

' SET UP THE WAV1

' Set up the WAV1 for 2kHz range and sine output. The "0, 0" parameters at
' end of the parameter list mark the positions of parameters reserved for
' a future release of KDAC500. These have no effect in KDAC500 V1.3, and
' should be left as 0, 0.

'     CALL WavSetup( ION$ ,  FREQ_RNG%,  FUNCT%, 0, 0)

      CALL WavSetup("wav1", WV.FRNG.2k, wv.sin, 0, 0)


' OUTPUT THE WAVEFORM

'     CALL wav(FRQ, AMPL, OFFS, DUTY, MODE)

      CALL Wav(1000, 5, 0, 0, wv.output.on)


' NOTE: Where two or more WAV1 modules are used in a system, WAV should be
' called immediately after the WAVSETUP associated with it. Any WAV call will
' automatically assume the setup described in the last WAVSETUP command.

END
```

```
'*************************************************************************
'*
'* AMMPOKE.BAS
'*
'* EXAMPLE QUICKBASIC PROGRAM FOR POKING THE AMM1A or AMM2.       10/15/90
'*
'* This program uses POKEs exclusively, and does not require KDAC500. It
'* assumes that the interface card is set to address CFF80(h). The AMM
'* module must be located in slot 1.
'*
'*************************************************************************

' Do a set-up for the AMM module...

      CLS : KEY OFF
      DEF SEG = &HCFF8            ' Set segment address of interface.

' Reset and recalibrate the A/D converter...

      POKE &H0, 0
      POKE &H1, 0
      POKE &H1A, 255

' Wait for conversion done

      WHILE PEEK(&H1B) > 127: WEND

'-------------------------------------------------------------------------

'       Set up a byte for a write to CMDA which will select channel, single-
'       ended mode, local gain of 1, regular acquire mode, and filter
'       See AMM manual for other legal values for each bit.
'
'              Bit pattern =  (MSB)  8 7 6 5 4 3 2 1  (LSB)
'
'       Calculations as BIT POSITION WEIGHT x DESIRED PROGRAMMING VALUE
'       ---------------------------------------\/-----------------
'       Select channel 0 -- 0 * 1               =  0      (bit 1, 2, 3, 4)
'       Select single-ended mode -- 16 * 1      = 16      (bit 5)
'       Select local gain 1 -- 32 * 0           =  0      (bit 6)
'       Regular acquire mode -- 64 * 0          =  0      (bit 7)
'       Filter 100k -- 128 * 0                  =  0      (bit 8)
'                                              ---------
'       Total                                    16
'
'       Set up a byte for write to CMDB which will set up slot, CMDA read
'       mode, range, and global gain
'
'       Select slot -- 1 * 1                    =  1      (bit 1, 2, 3, 4)
'       CMDA read mode -- 16 * 1 for data read  = 16      (bit 5)
'       Range select +/-10  -- 32 * 1           = 32      (bit 6)
'       Global gain is X1 -- 64 * 0 and 128 * 0 =  0      (bit 7, 8)
'                                              ----------
'       Total                                    49


' Do the set-up writes...

           POKE &H0, 16               ' CMDA write
      POKE &H1, 49               ' CMDB write

'-------------------------------------------------------------------------

' Set up a DO loop and do PEEKs and POKEs while keyboard is inactive...

       DO
```

```
' Start an A/D conversion...

              POKE &H1B, 255

' Check status of the AMM to see if conversion is done...

              WHILE PEEK(&H1B) > 127: WEND

' Read registers...

              DHIGH = PEEK(&H1)
              DLOW = PEEK(&H0)

' Reconstruct the data (+/-10V range assumed)...

              DRES! = 256 * DHIGH + DLOW
             DVOLTS! = (20 * DRES! / 65536!) - 10
              LOCATE 1, 1: PRINT "A/D counts = "; DRES!; "          "
              LOCATE 3, 1: PRINT "Voltage = "; DVOLTS!; "           "

' If keyboard is still inactive, loop around and do it again...

       LOOP WHILE INKEY$ = ""

END
```

KEITHLEY

Service Form

Model No. _____ **Serial No.** _____ **Date** _____

Name and Telephone No. _____

Company _____

List all control settings, describe problem and check boxes that apply to problem. _____

_____

_____

- ❑ Intermittent
- ❑ Analog output follows display
- ❑ Particular range or function bad; specify _____
- ❑ IEEE failure
- ❑ Obvious problem on power-up
- ❑ Batteries and fuses are OK
- ❑ Front panel operational
- ❑ All ranges or functions are bad
- ❑ Checked all cables

Display or output (check one)

- ❑ Drifts
- ❑ Unstable
- ❑ Overload
- ❑ Unable to zero
- ❑ Will not read applied input

- ❑ Calibration only
- ❑ Data required
- ❑ Certificate of calibration required

(attach any additional sheets as necessary)

Show a block diagram of your measurement system including all instruments connected (whether power is turned on or not). Also, describe signal source.

Where is the measurement being performed? (factory, controlled laboratory, out-of-doors, etc.)

_____

_____

What power line voltage is used? _____ Ambient temperature? _____ °F

Relative humidity? _____ Other? _____

Any additional information. (If special modifications have been made by the user, please describe.)

_____

_____

Be sure to include your name and phone number on this service form.

KEITHLEY METRABYTE/ASYST/DAC

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

AUSTRIA:	**Keithley Instruments GesmbH** • Vienna Office • Rosenhugelstrasse 12 • A-1120 Wien • 0-222-804-6548 • Fax: 0-222-804-3597
FRANCE:	**Keithley Instruments SARL** • 3 Allee des Garays • B.P. 60 • 91121 Palaiseau Cedex • 0-1-60-11-51-55 • Fax: 0-1-60-11-77-26
GERMANY:	**Keithley Instruments GmbH** • Munich Office • Landsberger Str. 65 • D-8034 Germering • 0-89-849307-0 • Fax: 0-89-84930759
GREAT BRITAIN:	**Keithley Instruments, Ltd.** • The Minster • 58 Portman Road • Reading, Berkshire RG3 1EA • 0-734-575666 • Fax: 0-734-596469
ITALY:	**Keithley Instruments SRL** • Viale S. Gimignano 38 • 20146 Milano • 0-2-48303008 • Fax: 0-2-48302274
JAPAN:	**Keithley Instruments Far East KK** • Sumiyoshi 24 Bldg., Room 201 • 2-24-2 Sumiyoshi-Cho • Naka-Ku, Yokohama 231 • 81-45-201-2246 • Fax: 81-45-201-2247
NETHERLANDS:	**Keithley Instruments BV** • Amsterdam Office • Avelingen West 49 • 4202 MS Gorinchem • Postbus 559 • 4200 AN Gorinchem • 0-1830-35333 • Fax: 0-1830-30821
SWITZERLAND:	**Keithley Instruments SA** • Zurich Office • Kriesbachstrasse 4 • 8600 Dubendorf • 0-1-821-9444 • Fax: 0-1-820-3081
TAIWAN:	**Keithley Instruments Taiwan** • Room 1105, 11th Floor, No. 147 • Section 2, Chien Kuo North Road • Taipei, R.O.C. • 886-2-509-4465 • Fax: 886-2-509-4473