KM-488-DD

IEEE-488 Interface Board

Keithley MetraByte Corporation

A Subsidiary of Keithley Instruments, Inc.
440 Myles Standish Boulevard
Taunton, Massachusetts 02780

Part Number: 24407

First Printing: June 1991

☞ ***NOTE:*** This equipment has been tested and found to comply with the limits for a Class B Digital Device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does not cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/tv technician for help.

☞ ***NOTE:*** Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate this equipment.

☞ ***NOTE:*** The use of a non-shielded interface cable with the referenced device is prohibited.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 GENERAL INTRODUCTION

Keithley MetraByte's KM-488-DD is an IEEE-488 interface board which allows programs written on an IBM® PC/XT/AT/386™ or IBM PS2 models 25/30 to communicate with the IEEE-488 bus. The KM-488-DD conforms to the 1978 IEEE-488 (GPIB) standard and thus, is compatible with other IEEE-488 products currently offered by many other manufacturers. Up to fourteen other devices may be connected to the IEEE-488 bus, including instruments, printers, and other computers.

The KM-488-DD's design includes a Wait State Generator to adjust the bus timing. This allows the KM-488-DD to perform within the operating specifications of the GPIB controller chip, even on the fastest 386 or 486-based PC's. The KM-488-DD can also be configured to generate interrupts on any of 6 interrupt request lines and DMA transfers on channels 1, 2, or 3. Figure 1-1 provides a functional block diagram of the KM-488-DD.

Keithley MetraByte's software allows the KM-488-DD to be accessed either through the use of the File I/O Commands or from the Call Interface, both of which can be used in programs written in QuickBASIC™, BASICA, Microsoft™ C, TURBO™C, TURBO™ PASCAL, and Microsoft™ PASCAL. The KM-488-DD is also compatible with a variety of third-party software packages, such as Keithley Asyst's Asystant GPIB.

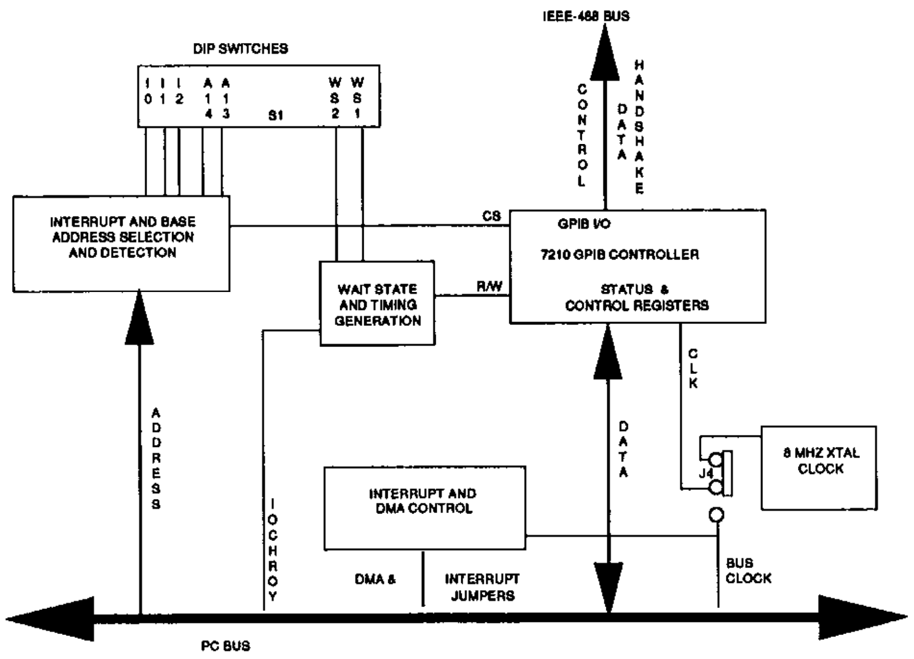


Figure 1-1. KM-488-DD Block Diagram

1.2 PROGRAMMING OVERVIEW

Keithley MetraByte's driver for the KM-488-DD is a Terminate and Stay Resident driver which provides a means of transferring data between your application program and the GPIB bus. Once the driver is installed, the application program can access it either through the DOS file facility or directly by program calls.

Programs which must be optimized for GPIB throughput should employ the CALL interface. If throughput is not a consideration then the choice between the CALL interface and the FILE interface is a matter of personal preference.

Programmers familiar with using file I/O (Opening files and transferring data between the program and files) may be more comfortable with the FILE interface. The File I/O Command interface tends to distance the programmer a bit more from the "mechanics" of the GPIB bus than the CALL interface. Programming can be as simple as writing strings to a file opened for output or reading strings from a file opened for input. The File I/O Commands are "English-Language" like representations of the GPIB commands. Through the use of the *CONFIG* program, you can create series of facts about a particular GPIB device (i.e., bus address and terminators) and refer to it as a **Named Device** through the File I/O

Command Interface. For example, the GPIB address 12 could have the name "SCOPE", which when used implies the GPIB address 12 and whatever terminators were associated with "SCOPE".

With the exception of the fact that Named Devices cannot be referred to directly through the CALL interface, for most of the File I/O Commands there is an a CALL which performs an equivalent function.

The following two sections are provided for those who wish to know a bit about how the driver functions. You do not have to read these sections in order to employ the driver. Most of what is described occurs automatically without the programmer's knowledge.

FILE I/O COMMAND INTERFACE

To employ the File I/O Command Interface, the programmer will use the facilities of his/her particular programming language to open a file for outputting to the driver and a file for inputting from the driver.

Driver Commands/Direct Data Output

Transfers between the application program and the GPIB bus will be initiated by the application program sending a command to the driver. Commands take the form of character strings which are written to the driver output file opened through DOS (Groups of commands and also data may be placed in one string; but the maximum string length is 255 characters.). This has the effect of calling DOS with a "Write to Device" request as shown in Figure 1-2. DOS moves the character string one character at a time from the application program into the command parser of the KM-488-DD device driver. DOS must make two calls to the driver for each character that is transferred. This command transfer process is relatively time consuming.

Once the command is in the driver parser, the driver must interpret the command and jump to a particular set of code within the driver which will carry out the appropriate function.

If the command is to output data to the GPIB bus, the data can be included in the string along with the command and follows the same path as the command through DOS. The amount of data than can be sent in this manner is limited by the string length and must also pass through DOS character by character. The one exception is the "OUTPUT DEVICE; data" command in which data can consists of an infinite number of characters. OUTPUT's of this type are halted when the EOL character sequence is encountered in the data stream of characters.

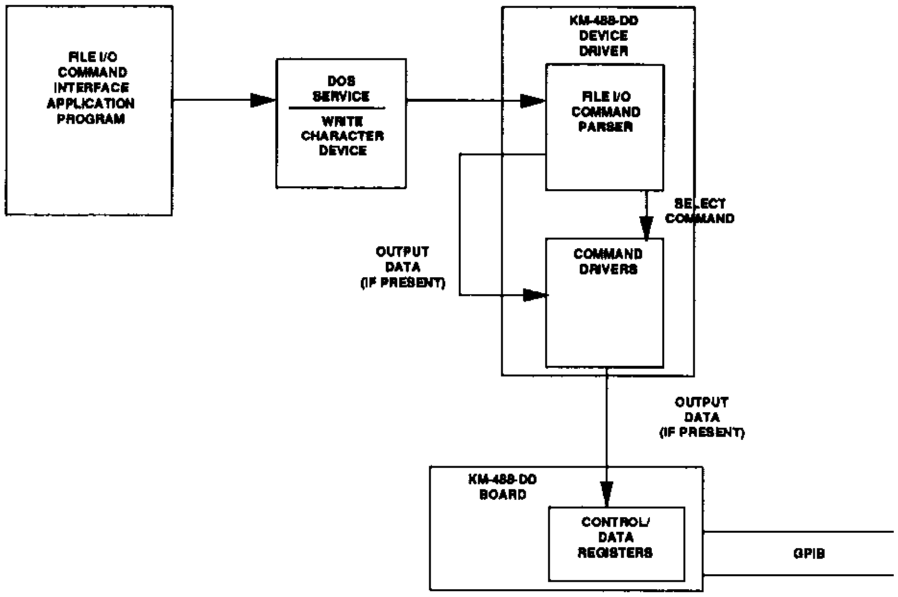


Figure 1-2. File I/O Interface - Direct Output Path

Direct Data Entry

To return information from the driver, the application program sends a command to the driver requesting data in the manner described above and then initiates an input from the driver. For example, the returned data may pertain to the status of the driver or be obtained from the GPIB bus. In either case, the data is buffered in the driver awaiting an INPUT action.

To bring the data into the application program, the program does an input from the file that was opened for driver input as shown in Figure 1-3. This amounts to employing the DOS "read from device" request and data will be returned in the same manner as it was sent, character by character. It is important to input data resulting from a command before issuing a second command which will return data. Otherwise, a driver error will occur.

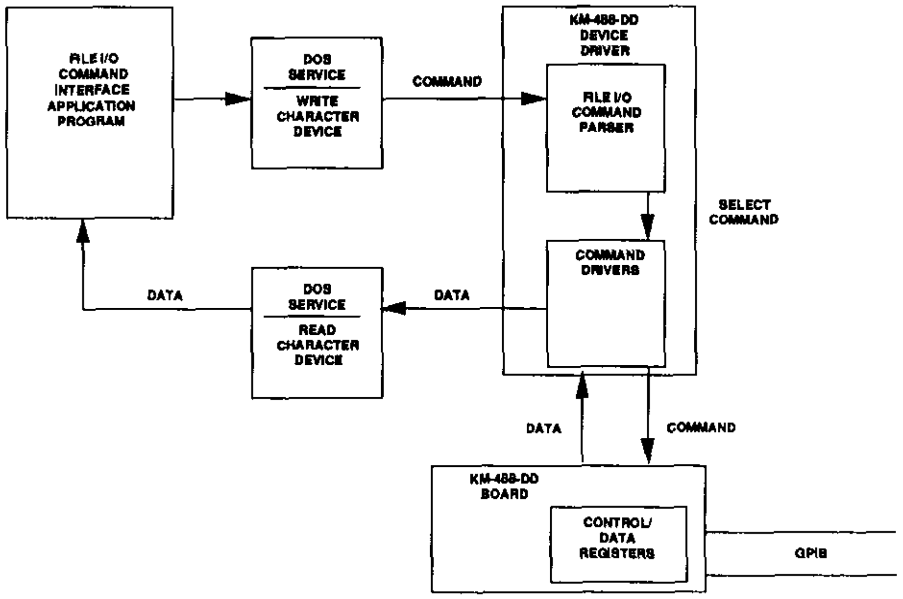


Figure 1-3. File I/O Interface - Direct Input Path

Buffered/DMA Data I/O

The amount of data that can be transferred by Direct I/O is limited (except in the case of "OUTPUT 05; data" and the transfer mechanism is relatively slow. Both these deficiencies can be overcome by using buffered transfers. In this case, the command sent to the driver contains information about the location in memory where data will sent to/received from the GPIB bus. Once set up, the data will move directly between the program memory and the GPIB bus via the KM-488-DD's hardware. Figure 1-4 illustrates the case of data send. A data entry would appear similar with the data passing through an ENTER routine in the opposite direction.

In normal buffered transfers, the driver transfers the data between memory and the GPIB bus via the KM-488-DD's hardware. The PC's DMA facility can be used to make a buffered transfer. Although buffered transfers are generally faster, there will be some overhead in setting them up so direct transfers are probably just as efficient if only small amounts of data are involved. All transfers through the CALL interface are buffered.

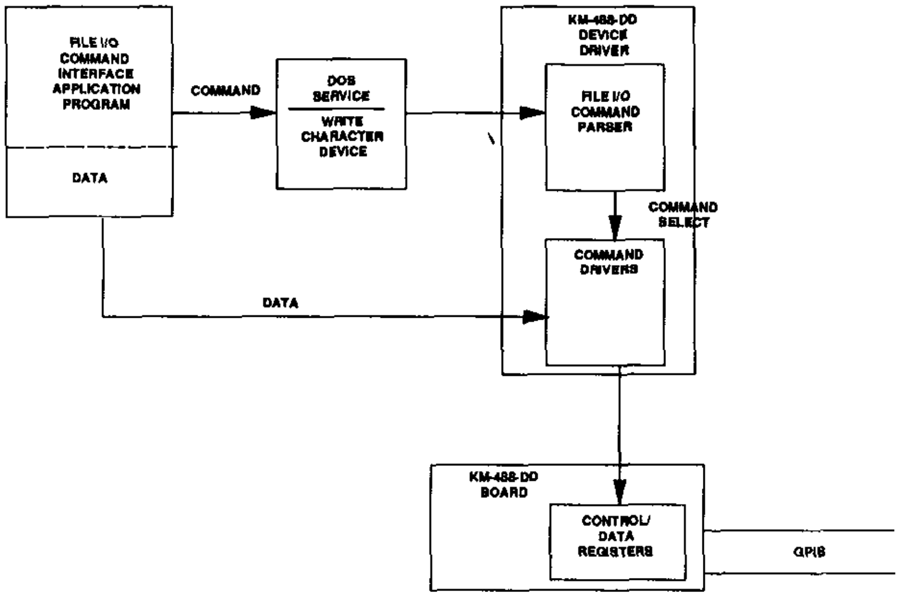


Figure 1-4. File I/O Commands - Buffered Output Path

CALL INTERFACE

To employ the CALL interface, the programmer uses the call facility of his/her particular programming language. Typically, a Keithley MetraByte provided library(ies) or loadable module is incorporated when building the application with a supported language. Appropriate batch files are provided to assist in remaking these libraries/modules.

When a call is made, the appropriate parameters contained in the call are placed on the caller's stack for use by the driver and the program enters the code of the interface module as diagrammed in Figure 1-5. The interface module for the particular language verifies that the driver is installed and jumps directly to the driver command code which performs the desired function. The amount of time required to get to the driver command code from the application by means of the call interface is much less than that by means of the file interface.

If data is to be transferred, then the call parameters include addresses of location(s) in program memory where the data is to be found and/or sent. As opposed to the File I/O Command Interface, the Call Interface always uses the faster buffered transfers (with or without DMA).

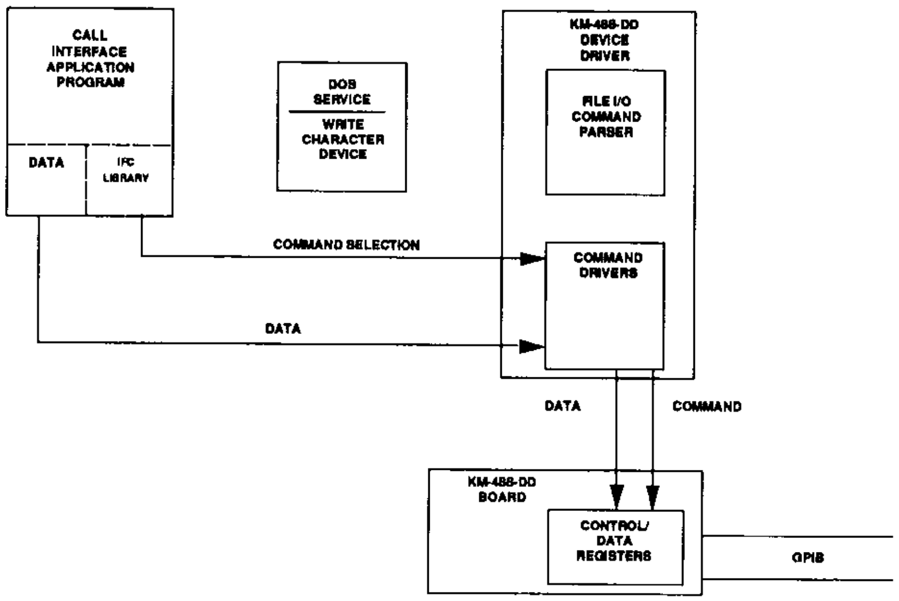


Figure 1-5. Call Interface Path

1.3 SPECIFICATIONS

Specifications for the KM-488-DD are as follows:

Dimensions:	One Short PC Slot size
Weight:	.29 lb.
Data Transfer Rate:†	> 300 KB per second
IEEE Controller Chip:	NEC µPD7210
Power Consumption:	< 650 mA
Operating Temperature:	0° to 50° C
Storage Temperature:	-20° to +70° C
Humidity:	0 to 90% non-condensing

†Governed by the slowest device on the GPIB bus.

Wait States:	Switch Selectable 1,2,3 or 4
I/O Base Address:	Switch Selectable &H02E1, &H22E1, &H42E1, or &H62E1
DMA Capability:	Switch Selectable Channels 1,2, or 3 shareable
Interrupt (IRQ) Address:	Switch Selectable Levels 2 through 7
Device Interface Capabilities Supported	SH1, AH1, T6, TE0, L4, LE0, SR1, PP0, RL0, DC1, DT1, C0, E1/2, Controller Subsets: C1, C2, C3, C4, and C10 Terminator: Software Selectable characters and/or EOI

1.4 ORDERING INFORMATION

Table 1-1 provides additional ordering information, in the event that you need additional KM-488-DD's or appropriate cables.

Table 1-1. Ordering Information

Part Number	Description
KM-488-DD	Includes the KM-488-DD IEEE-488 Interface Board, Software (on 5.25" disks), and appropriate documentation.
KM-488-DD/3.5	Includes the KM-488-DD IEEE-488 Interface Board, Software (on 3.5" disks), and appropriate documentation.
KM-488-DD-BRD	Includes the KM-488-DD IEEE-488 Interface Board and appropriate documentation.
CGPIB-1	1 meter IEEE-488 cable.
CGPIB-2	2 meter IEEE-488 cable.
CGPIB-4	4 meter IEEE-488 cable.

1.5 HOW TO USE THIS MANUAL

This manual provides you with the information necessary to install and program the KM-488-DD. It assumes you are familiar with the language in which you are developing your application program. It also assumes that you are familiar with the GPIB protocol.

Chapter 2, "Installation and Configuration", provides all the necessary information for you to get the KM-488-DD "up and running". It details how to unpack, inspect, and install the KM-488-DD. It also describes how to install and configure the software. Jumper and switch setting information is provided in case you should need to change the factory-selected defaults.

Chapters 3 through 5 describe the use of the File I/O Command Interface. Chapter 3 provides introductory information. Chapter 4 describes how to implement the commands in each supported language: BASICA, QuickBASIC, C, Microsoft PASCAL, and TURBO PASCAL. Chapter 5 describes each command in detail.

Chapters 6 through 11 described the use of the Call Interface. Chapter 6 provides introductory information. Chapters 7 through 11 provide a detailed description of each routine and how it is called from each of the languages supported: BASICA, QuickBASIC, C, Microsoft PASCAL, and TURBO PASCAL.

Chapter 12, "Warranty Information", describes Keithley MetraByte's warranty and gives instructions for returning the board to the factory.

The appendices contain additional useful information. Appendix A contains an ASCII Equivalence Chart. This gives hexadecimal and decimal equivalents for the ASCII 128 Character Set. Appendix B is an IEEE-488 tutorial. Appendices C and D contain supplementary information to the IEEE-488 tutorial (Device Capability Identification Codes and IEEE Multiline Commands). Appendix E describes how to use the KM-488-DD Printer and COM Port Re-Director. Appendix F describes the *CONFIG.DAT* file. Appendix G lists and describes the Error Messages.

CHAPTER 2
INSTALLATION AND CONFIGURATION

This chapter outlines all of the steps necessary to get your KM-488-DD "up and running". The chapter is broken up into four parts;

- ***Part 1 - Hardware Installation.*** This section describes how to unpack and inspect the KM-488-DD, set its jumpers and switches, and install the KM-488-DD board(s).

- ***Part 2 - Initial Software Installation and Configuration.*** This section describes those procedures which are to be used the first time you install and configure the KM-488-DD software.

- ***Part 3 - Driver Installation.*** Once you have configured the software, you will need to load the driver software before you can use the KM-488-DD. This section describes how to load the driver from your *AUTOEXEC.BAT* file or manually from the DOS prompt.

- ***Part 4 - Software Re-Configuration.*** This section describes how to change your system configuration once it has already been made.

Once you have installed the KM-488-DD and configured the software, you may want to run the *KIEEECMD.BAS* program. This program will allow you to send File I/O Commands and see the results. Note, however, that this program can only be run in a single board set-up. To run this program, make sure that BASICA is installed and from the DOS prompt, type: **`BASICA KIEEECMD`**.

PART 1 - HARDWARE INSTALLATION

Before you can use your new KM-488-DD board you must unpack and install it correctly. To do this, follow these steps:

- ***Unpack and Inspect the KM-488-DD.*** This seems like a trivial step; however it is crucial that your board be unpacked correctly. This information is described in section 2.1.

- ***Set the Jumpers and Switches on the KM-488-DD.*** This step is unnecessary if you elect to use the factory selected defaults shown in Table 2-1. If you want to change any of these settings, consult section 2.2.

- ***Install the KM-488-DD.*** The KM-488-DD is installed in the same manner as any other peripheral board. If you are unfamiliar with this process, follow the instructions in section 2.3.

If you are installing multiple KM-488-DD's in the same GPIB system, be sure to read Section 2.4, Multiple Board Installation Notes.

2.1 UNPACKING AND INSPECTION

Confirm that each item on the packing list has been shipped. It is a good idea to retain the outer packing material in the event that the board must be returned to the factory for repair.

1. Then, if you haven't already done so, unpack the board down to its anti-static packaging.

2. Hold the board (in its anti-static bag) in one hand. Place the other hand firmly on a metal portion of the system chassis. This procedure eliminates static electricity build-up, thus preventing any possible damage to the board.

3. After allowing a moment for static electricity discharge, carefully unwrap the board from the anti-static material.

4. Inspect the board for any possible shipping damage. If any sign of damage is detected, return the board to the factory as described in section 12.2.

You are now ready to set the jumpers and switches on the KM-488-DD.

2.2 JUMPERS AND SWITCHES

The KM-488-DD contains one DIP switch, two 12-pin jumper banks, and a single 3-pin jumper. The locations of these are shown in Figure 2-1. The Base Address switch controls the base address, interrupt address, and wait states. The jumper banks are used to select the interrupt level and DMA Channel to be used. The single jumper is used to select the source of the master clock signal. The default switch and jumper selections are listed in Table 2-1.

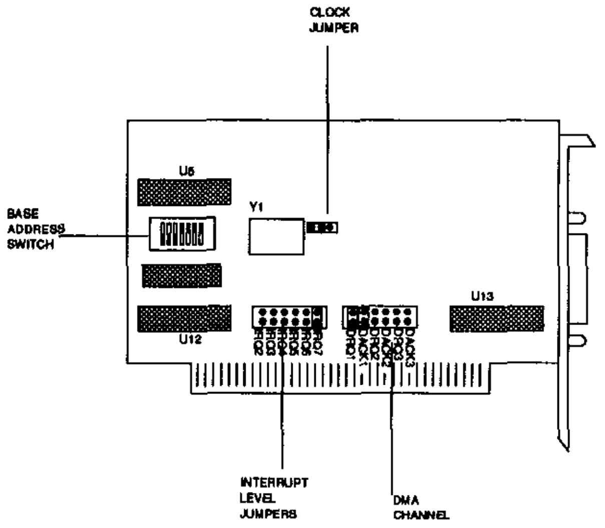


Figure 2-1. Location of Jumpers and Switches

A program, *INSTALL.EXE,* is provided to assist you with setting the switches and jumpers. To run the INSTALL program:

1. Insert Disk #1 "Installation and Setup" into your computer's floppy drive.

2. Then, change the directory and from the DOS prompt, type:

```
INSTALL
```

It is a good idea to record all switch and jumper selections on the GPIB System Configuration Worksheet found in the rear of this manual.

Table 2-1. Default Jumper and Switch Settings

Switch/Jumper	**Default**
Base Address Switch (S1)	
Base Address	02E1 hex
Interrupt Address	02F7 hex
I/O Wait States	1
Interrupt Level (J2)	7
DMA Channel (J3)	1
Clock (J4)	8 MHz. On-board

Switches

There is only one switch on the KM-488-DD board. This is the **Base Address Switch.** Remember to record your switch selections on the GPIB System Configuration Worksheet found in the rear of this manual.

The **Base Address Switch** (S1) is a 7-slide, DIP switch. When a rocker is moved to the ON position, this sends the signal low (logical 0). The 7 slide switches are used to select the **Base Address, Interrupt Address,** and **I/O Wait States.**

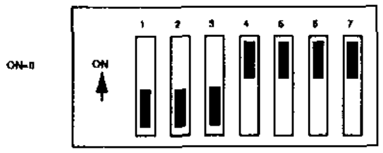


Figure 2-2. Base Address Switch

Interrupt Address. Switches 1, 2, and 3 (I0, I1, and I2) are used to select the Interrupt Response Level Address. This is the address to which the interrupting device must write to be reset. The address is **02F*x***, where *x* is the interrupt level, from 2 to 7. The default selection is 7. To make another choice, configure the slide switches as shown in Figure 2-3. These switches only need to be set if the KM-488-DD is used with software other than that provided.

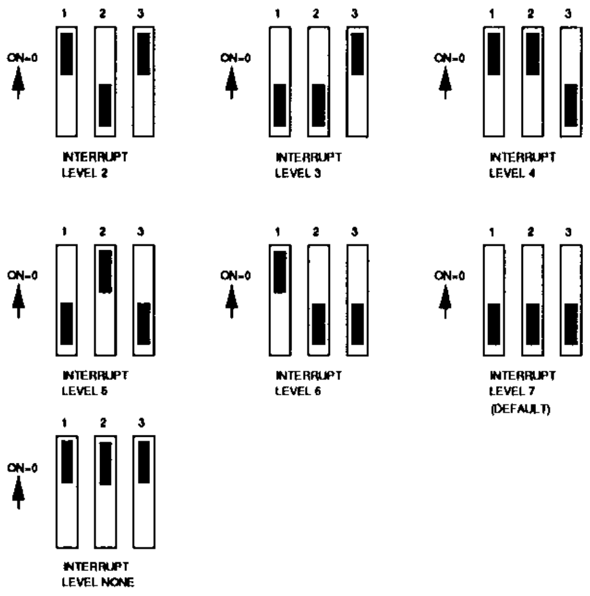


Figure 2-3. Interrupt Selection

Base Address. Switches 4 and 5 (A14 and A13) set the address used by the computer to communicate with the IEEE Controller Chip on the board. Available base addresses (in hex) are: 02E1, 22E1, 42E1, and 62E1. The default base address is 02E1. To select another base address, move the switches to the positions shown in Figure 2-4.

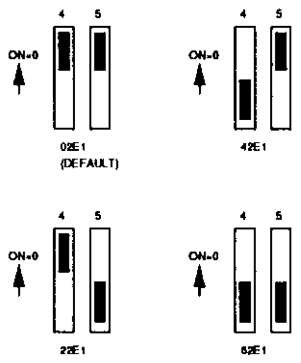


Figure 2-4. Base Address Selection

I/O Wait States. The KM-488-DD is designed with a switch-selectable wait state generator. Selectable wait states are provided to insure optimum performance and reliable operation at the differing bus clocks found in personal computers. The default number of wait states (1) should be correct for most personal computers. However, if you get garbled data or your program crashes, you may need to adjust the number of wait states. Some general guidelines are presented in Table 2-2. The number of wait states is selected by setting switches 6 and 7 (marked WS1 and WS2) on the DIP switch. You can select one, two, three, or four wait states. The default number of wait states is 1. To select a different number, set the switches to one of the positions shown in Figure 2-5.

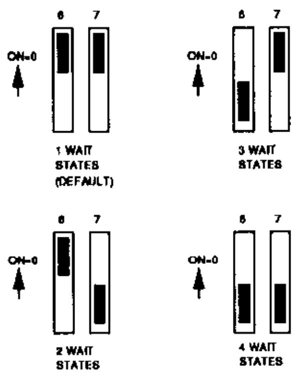


Figure 2-5. I/O Wait State Selections

Table 2-2. Wait States

Bus Clock Frequency	Number of Wait States
≤ 5 MHz.	1 *(default)*
5 MHz < freq ≤ 8 MHz.	2
8 MHz < freq ≤ 10 MHz.	3
10 MHz < freq.	4

Jumpers

There are two jumper blocks and a single 3-pin jumper on the KM-488-DD. These are used to select the **Interrupt Level**, **DMA Level**, and the **Clock Source**. Remember to record your jumper selections on the GPIB System Configuration Worksheet found in the rear of this manual.

Interrupt Level. If you are using third-party software, note that the KM-488-DD is capable of interrupting the PC under some circumstances. The interrupt level jumper (J2) defines the interrupt level to be used. Valid interrupt levels (2 through 7) and the jumper positions used are shown in Figure 2-6.

Be certain that the same Interrupt Level and the Interrupt Response Level (See Base Address Switch.) are selected. If you are programming the KM-488-DD with Keithley MetraByte's KM-488-DD driver software, note that this software does not support these interrupts. Thus, you should place the interrupt level jumper in the disabled position.

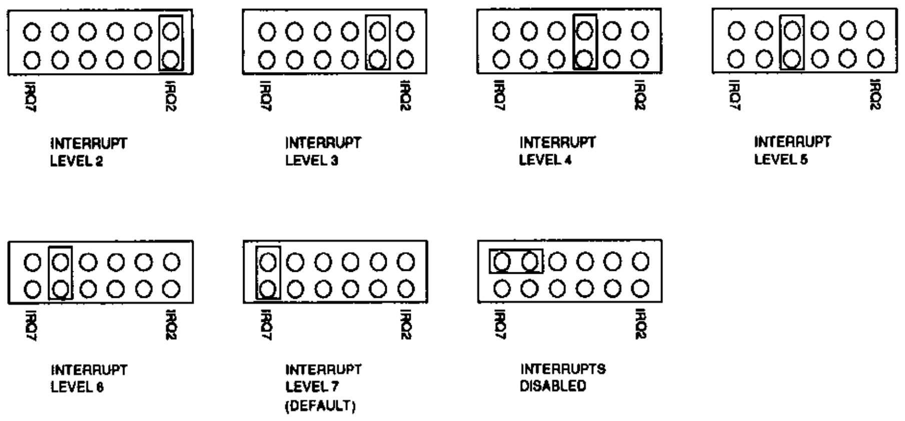


Figure 2-6. Interrupt Level Jumpers

DMA Level. DMA (Direct Memory Access) can be used to/from transfer data quickly from/to a peripheral to the computer. The PC has four DMA channels of which one or three may be selected. However, based on your system configuration they may not all be available. Select an appropriate DMA level using the DMA Level Jumpers. Refer to Figure 2-7 for jumper positions.

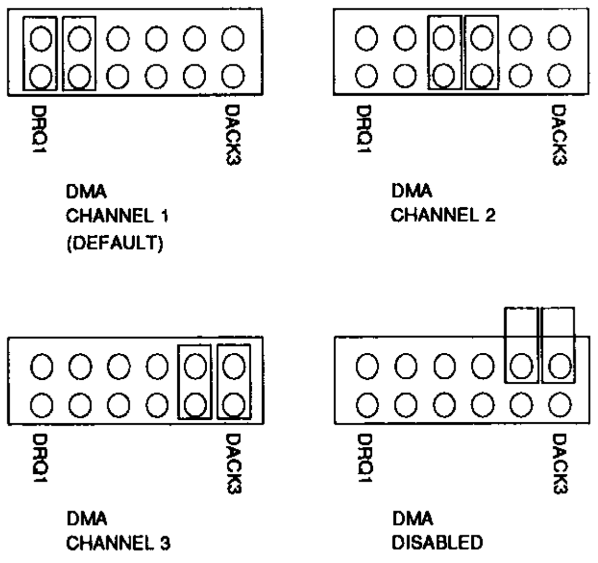


Figure 2-7. DMA Level Jumpers

Clock Source. The IEEE bus interface circuitry on the KM-488-DD board requires a master clock signal. This clock is normally connected to an on-board 8 MHz clock oscillator. You can, however, elect to drive this circuitry from the PC's clock. Be aware that the clock speed will vary according to the model of computer used. To select the clock source, place the jumper block between the appropriate pins. See Figure 2-8.

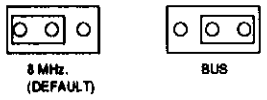


Figure 2-8. Clock Source Jumper

☞ ***NOTE:*** The BUS position may be required by certain third party software packages.

2.3 BOARD INSTALLATION

This section provides general instructions for installing the KM-488-DD Board in an IBM PC/XT/AT/286/386 or compatible. Some things, such as removing the computer's cover, are machine-dependent. Consult the documentation provided with your computer, if necessary.

WARNING

DO NOT ATTEMPT TO INSERT OR REMOVE ANY ADAPTER BOARD WITH THE COMPUTER POWER ON! THIS COULD CAUSE DAMAGE TO THE COMPUTER!

To install the KM-488-DD Board:

1. Turn the power to the PC and all attached options OFF.

2. Unplug the power cords of all attached options from the electrical outlets. Then, make a note of where all the device cables are attached to the rear of the system unit. Disconnect the cables.

3. Remove the cover of the PC. To do this, first remove the cover mounting screws on the rear panel of the computer. Then, slide the cover of the computer about 3/4 of the way forward. Tilt the cover upwards to remove.

4. Choose an available option slot. Loosen and remove the screw at the top of the blank adapter plate. Then slide the plate up and out to remove.

5. Hold the KM-488-DD (in its anti-static bag) in one hand. With the other hand, touch any metallic part of the PC/AT cabinet. This will safely discharge any static electricity which has built-up in your body. Unwrap the KM-488-DD from the anti-static bag.

6. If you haven't already done so, set the switches and jumpers as described in section 2.4.

7. Align the gold edge connector with the edge socket and the back adapter plate with the adapter plate screw opening. Gently press the board downward into the socket. Re-install the adapter plate screw.

8. Replace the computer's cover. Tilt the front of the cover up and slide it onto the system's base, making sure the sides of the cover are under the rail along the sides of the frame. Install the mounting screws.

9. Plug in all cords and cables. Turn the power to the computer back on.

2.4 MULTIPLE BOARD INSTALLATION NOTES

When installing multiple KM-488-DD in the same GPIB system, be sure to:

- Assign a different Base Address to each KM-488-DD
- Assign a different Interrupt Level to each KM-488-DD, if required by your software
- Assign a different DMA Level to each KM-488-DD, if required by your software

PART 2 - INITIAL SOFTWARE INSTALLATION AND CONFIGURATION

The first time you install the KM-488-DD software and configure your KM-488-DD system, you will run the *SETUP* batch file. Subsequent configurations are performed in a different manner as described in Part 4 - Reconfiguring the Software.

Before you install the software, you will need to determine the following:

- ***Which programming style are you going to use?*** Decide whether you want to develop your applications program using the File I/O Commands or Call Interface. If absolutely necessary, you can write your program using both interfaces. If you are having trouble deciding which programming style to use, refer to the Programming Overview in section 1.2 and glance through the reference chapters for each programming style.

  Also, know which language(s) you will be developing your programs in.

- ***Do you want to be able to refer to example programs?*** A full selection of example programs for each language and programming style is provided.

- ***Will you need to recompile the Interface Modules?*** If so, you may want to access the batch files which will do this for you automatically. These are considered source files.

- ***What kinds of terminators do you want to use?*** A discussion of terminator types is provided in Chapters 3 (for the File I/O Commands) and 6 (for the Call Interface). You may want to read these sections prior to configuring your system.

- ***If you are programming via the File I/O Commands, do you want to use Named Devices?*** Named devices are discussed in Chapter 3. If you want to use named devices, it is a good idea to determine what instruments will be part of your system before you begin the *SETUP* program.

Have ready the GPIB System Configuration Worksheet before you begin the configuration process. This worksheet can be found in the rear of this manual.

2.5 INITIAL KM-488-DD SOFTWARE INSTALLATION AND CONFIGURATION

Follow the procedure outlined in this section if you are installing the KM-488-DD software for the first time. (***NOTE:*** If you have already installed the software and simply need to change its parameters, see Part 4- Reconfiguring the Software.)

To begin, insert the KM-488-DD Installation and Setup Disk into the floppy disk drive on your computer. Then:

1. If necessary, return to the DOS prompt. This will look something like:

   ```
   C:\
   ```

2. Change the directory to the drive containing the disk. In most instances, this will be disk drive A: or B:. Thus, at the DOS prompt type:

   A: - for the A: drive

   or

   B: - for the B: drive

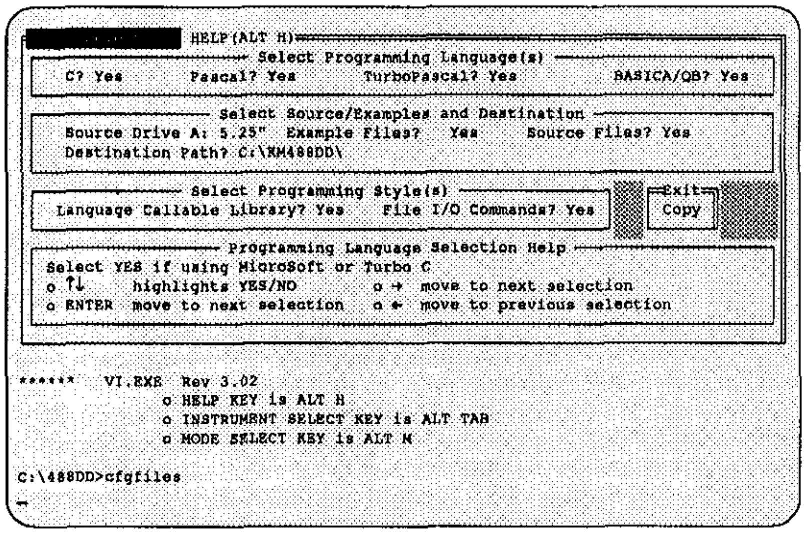
3. Now, type:

   SETUP

   The SETUP program will now run. This program allows you to copy the KM-488-DD software to your computer's hard drive, configure the KM-488-DD(s), select default terminators, and configure the other devices in your GPIB System.

1 - Copying the Software to the Hard Drive

The first step in the SETUP program is to copy the KM-488-DD software to your hard drive. This software consists of the KM-488-DD driver and language support files. (A complete list of the files comprising the KM-488-DD software can be found in the *FILES.DOC* file on the KM-488-DD Utility/Driver disk.) The SETUP program prompts you for the necessary information by displaying the screen shown in Figure 2-9.



```
                HELP(ALT H)
               Select Programming Language(s)
  C? Yes      Pascal? Yes      TurboPascal? Yes        BASICA/QB? Yes

               Select Source/Examples and Destination
  Source Drive A: 5.25"  Example Files?  Yes    Source Files? Yes
  Destination Path? C:\KM488DD\

           Select Programming Style(s)                      Exit
 Language Callable Library? Yes   File I/O Commands? Yes    Copy

               Programming Language Selection Help
 Select YES if using MicroSoft or Turbo C
 o ↑↓     highlights YES/NO        o →  move to next selection
 o ENTER  move to next selection   o ←  move to previous selection

******   VI.EXE  Rev 3.02
                o HELP KEY is ALT H
                o INSTRUMENT SELECT KEY is ALT TAB
                o MODE SELECT KEY is ALT M

C:\488DD>cfgfiles
_
```

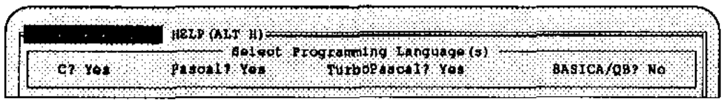
Figure 2-9. Software Installation Screen

At any time, if you wish to abort from the Software Installation Screen, move the highlight to **EXIT**. Select **ABORT**.

Select the Language(s). The first step is to select the language(s) which you are going to develop your application programs in. Choice of languages include: C, Microsoft PASCAL, TURBO PASCAL, BASICA, and QuickBASIC. Any combination of languages may be chosen. You must indicate whether or not you want language support for each language listed. To select/unselect a language:

- Move the highlight to the language desired. (Use ← or → or press Enter.)

- Use the ↑ or ↓ keys to select a YES or NO response. An example is shown in Figure 2-10.

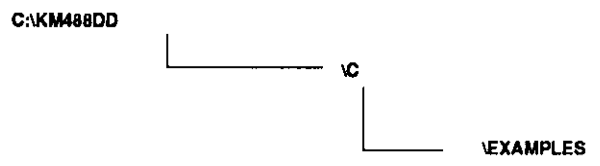


```
                HELP (ALT H)
               Select Programming Language(s)
  C? Yes      Pascal? Yes      TurboPascal? Yes        BASICA/QB? No
```

Figure 2-10. Selecting Language(s)

Choose the Destination. Next, you will need to tell the SETUP program which drive to copy the KM-488-DD software from and where to put the software on your hard drive. The default path is **C:\KM488DD**. You can elect to use this path or select another. To select a different one, type a new pathname in typical DOS format, i.e., **{drive}:\{main directory}\{sub-directory}**. The directory does not have to already exist on your hard drive. The SETUP program will create the directory for you.

Note that when the SETUP program copies the KM-488-DD software to your hard drive, it will automatically create the necessary sub-directories. For example, if you elected to copy the C support files and the example programs, the tree structure that would be created would look like this:

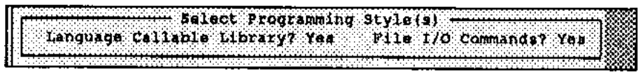


At this time, you will also need to indicate whether or not you want the Example Programs and Source Programs copied. There are example programs available for each language using both the File I/O Commands and the Call Interface. The Source Programs consist of batch files which can be used to produce executable versions of the example programs. The example and source programs are further described in the file *FILES.DOC*.

Select Programming Style. Next, you need to instruct the SETUP program that you will be programming via the File I/O Commands, the Call Interface, or both. To do this:

- Move the highlight to **File I/O** Commands. (Use ← or → or press Enter.)

- Use the ↑ or ↓ keys to select a YES or NO response. Repeat this procedure for **Call Interface**. Your selections will look something like Figure 2-11.



```
Select Programming Style(s)
Language Callable Library? Yes    File I/O Commands? Yes
```

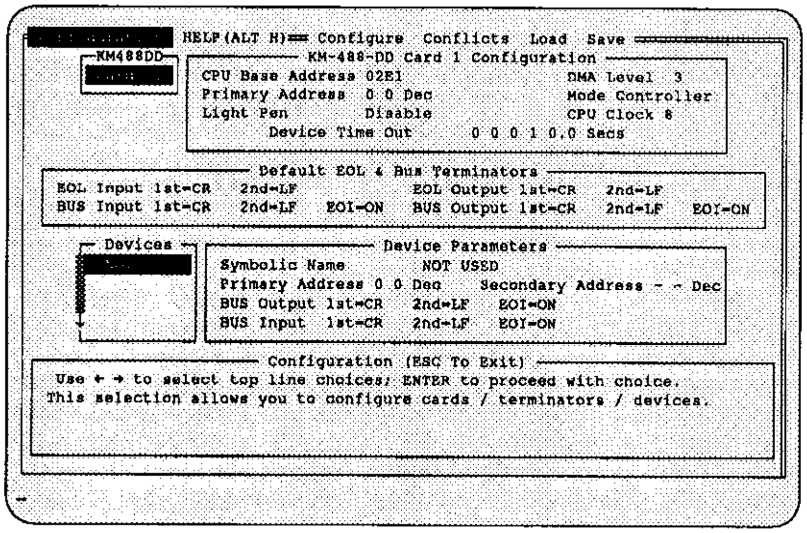
Figure 2-11. Select Command Style

Exit and Copy. If you want to discontinue or abort from the software copying procedure, at **EXIT** select **ABORT**. Otherwise, review your selections:

- If they are all correct, select **COPY**. The program will prompt you to insert the required disks.

- If your selections are incorrect, return to the selection and alter your choice. Then, return to **EXIT**, select **COPY**, and follow the instructions.

Once the disk copying process has been completed, SETUP will advance to the Device Configuration Screen.



```
HELP(ALT H)== Configure  Conflicts  Load  Save
KM488DD          KM-488-DD Card 1 Configuration
        CPU Base Address 02E1                      DMA Level  3
        Primary Address  0 0 Dec                   Mode Controller
        Light Pen        Disable                   CPU Clock 8
              Device Time Out      0 0 0 1 0.0 Secs

                      Default EOL & Bus Terminators
 EOL Input 1st=CR   2nd=LF              EOL Output 1st=CR   2nd=LF
 BUS Input 1st=CR   2nd=LF   EOI=ON     BUS Output 1st=CR   2nd=LF   EOI=ON

 Devices                  Device Parameters
          Symbolic Name          NOT USED
          Primary Address 0 0 Dec    Secondary Address - - Dec
          BUS Output 1st=CR   2nd=LF   EOI=ON
          BUS Input  1st=CR   2nd=LF   EOI=ON

                       Configuration (ESC To Exit)
 Use ← → to select top line choices; ENTER to proceed with choice.
 This selection allows you to configure cards / terminators / devices.
```

Figure 2-12. Device Configuration Screen

In this part of the initial software installation process, you will configure the "devices" which are part of your GPIB application. These devices include the KM-488-DD board(s) and other GPIB instruments such as meters, scopes, and plotters. The configuration parameters needed vary according to your application and whether you are configuring a KM-488-DD or a GPIB instrument. Note that the parameters selected must agree with the actual board switch and jumper settings.

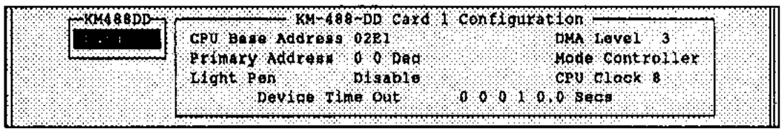
For every KM-488-DD installed (you can have up to 4), you must assign it a CPU Base Address, a GPIB Address, and select various operating parameters. If you are using the File I/O Command Interface, you also have the option of setting other GPIB devices as *named devices.* This means that for every device you can assign it a name, a GPIB address, and the Bus terminators which it will recognize. The use of named devices is especially helpful in applications where you have many devices which recognize different terminator sequences.

A GPIB System Configuration Worksheet is provided in the rear of this manual. You may find it helpful to keep track of your system configuration. If you have more devices than there are spaces allocated, simply photocopy the rear side of the worksheet.

Once you have completed your System Configuration, you will need to save the parameters to the *CONFIG.DAT* file. This file is read into the KM-488-DD driver. The *CONFIG.DAT* file is described in more detail in Appendix F.

2 - Configuring the KM-488-DD(s)

The first step is to configure the KM-488-DD(s). Note that bypassing the configuration forces the KM-488-DD to work with the default settings. To do this, use the ← and → keys to highlight the **CONFIGURE** option at the top of the System Configuration Screen. Then, select **CARD**. The Card Configuration section of the screen will be highlighted. (See Figure 2-13.)



```
KM-488-DD Card 1 Configuration
CPU Base Address 02E1                    DMA Level  3
Primary Address  0 0 Dec                 Mode Controller
Light Pen        Disable                 CPU Clock 8
     Device Time Out      0 0 0 1 0.0 Secs
```

Figure 2-13. Selecting the Card Option

Keys Used Within the System Configuration Screen

The keys listed below have the following general functions. Note that these functions may change according to the option being configured. Be sure to watch the listing at the bottom of the System Configuration Screen.

↑ ↓	Changes the highlighted choice.
→	Moves to the next selection.
←	Moves to the previous selection.

[Enter] Takes action or moves on to the next selection.

If You Cannot Remember Your Switch/Jumper Settings. Refer to your GPIB System Configuration Worksheet.

Adding Board Structures. If you have multiple KM-488-DDs, you will need to add board structures. This is accomplished by moving the cursor to the **Card1** option and pressing [Ins].

You are now ready to begin configuring your KM-488-DD(s).

Select the Board to be Configured. If you have multiple KM-488-DDs, you will need to select the board to be configured. Move the cursor to the list of boards, using the cursor keys. Then press [↓] or [↓] to move the highlight to the board to be configured.

Select a Base Address. The KM-488-DD can be assigned any one of four base addresses: 02E1 hex, 22E1 hex, 42E1 hex, and 62E1 hex. The default base address is 02E1 hex. To select a different base address:

- Press [→] to move to the **CPU Base Address** option. If the base address displayed conflicts with (is also assigned to) another KM-488-DD, it will appear in red.
- Press [↓] or [↑] to toggle through the list of valid base addresses.
- Press [Enter] or [→] to select a base address and advance to the **Primary Address** option..

Assign a GPIB Primary Address. The KM-488-DD must be assigned a unique GPIB Primary Address in the range 00 to 30. To define the GPIB Primary Address, enter each digit separately. Then, press [Enter].

Enable/Disable Lightpen Interrupts. It is possible for lightpen interrupts to occur under certain conditions. However, the **Lightpen** option must be enabled for this to occur. Choose **enable** or **disable** by pressing the [↑] or [↓] keys. Then press [Enter] or [→] to advance to the **DMA Level** option.

Select a DMA Level. The KM-488-DD must be assigned one of the following DMA Levels: 0, 1, 2, 3, or none. If the DMA level selected conflicts with that assigned to another board, it will be displayed in red. To select an DMA level, toggle through the options by pressing ↓. When you reach the desired value, press [Enter] or → to select the value and advance to the **Mode** option.

Choose a GPIB Operating Mode. The KM-488-DD is capable of acting as either a **Controller** (System) or a **Peripheral** (non-System Controller). You must indicate which function the KM-488-DD is to perform. Toggle through the selections by pressing ↑ or ↓. At the appropriate selection, press [Enter] or → to select the mode and advance to the **CPU Clock** option.

Select a CPU Clock Speed. The IEEE bus interface circuitry on the KM-488-DD board requires a master clock signal. This clock is normally connected to an on-board 8 MHz clock oscillator. You can, however, elect to drive this circuitry from the PC's clock. The actual clock speed will vary according to the model of computer used. The **CPU Clock** option is used to select the clock speed. (Remember that source of the clock is selected by Jumper J4. See section 2.2.) Clock speeds range between 1 MHz. and 8 MHz.

To select the clock speed:

- Press ↓ or ↑ to toggle through the list of valid clock speeds. Note that if the clock source is the on-board clock, you must select 8.

- Press [Enter] or → to select the clock speed and advance to the **Device Timeout** option.

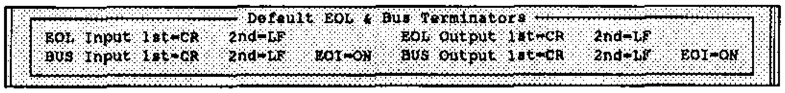
Set a Device Timeout. This parameter sets the maximum amount of time (in milliseconds) which is to elapse before a Device Timeout Error occurs. The default value is 10 seconds. Valid Timeouts range from .1 to 65535.9 seconds. A timeout value of 0.0 will disable timeout checking.

To change the value, enter each digit separately. Then, press [Enter].

Be sure to specify all of the above parameters for each KM-488-DD board.

3 - Choosing the Default Terminators

Next, you will need to select the default terminator sequences to be used. To begin, use the ↑ and ↓ keys to select the **TERMINATORS** option from the **CONFIGURE** menu. The Default EOL and Bus Terminators section of the screen will be highlighted. (See Figure 2-14.)



```
─────────────── Default EOL & Bus Terminators ───────────────
EOL Input 1st=CR  2nd=LF          EOL Output 1st=CR  2nd=LF
BUS Input 1st=CR  2nd=LF  EOI=ON  BUS Output 1st=CR  2nd=LF  EOI=ON
```

Figure 2-14. Selecting the Default EOL and Bus Terminators

The KM-488-DD driver recognizes two types of terminator sequences: Bus Terminators and EOL Terminators. The Bus Terminators are used to delimit data transmitted between the KM-488-DD driver and the GPIB Bus. The EOL terminators mark the end of a character string which is being transferred between your application program and the KM-488-DD driver. The KM-488-DD driver then replaces these EOL terminators with a GPIB Bus Terminator sequence when communicating with the GPIB bus devices. EOL Terminators are only used by the File I/O Commands. (See Chapter 3 for more information.) The default terminators are listed in Table 2-3.

Table 2-3. Default Terminator Sequences

Terminator	EOL		GPIB		
	1ST	2ND	1ST	2ND	EOI
Input	CR	LF	CR	LF	ON
Output	CR	LF	CR	CR	ON

EOL Terminators. You need not change the default EOL terminators if you are using the Call Interface. (KM-488-DD Calls don't recognize EOL's.)

The **EOL Input Terminator(s)** is appended to data sent from the KM-488-DD to the application program. It consists of one or two ASCII characters. (They don't have to be printable.) To change the EOL Input Terminator, press → to the **1st** prompt. Enter the desired ASCII character using the format described in the help screen. Do the same for the second character. If no second character is desired, leave the space blank.

The **EOL Output Terminator(s)** are selected in the same manner as the

EOL Input Terminators.

GPIB Bus Terminators. The **BUS Input Terminator** sequence marks the end of data transferred from the GPIB bus to the KM-488-DD driver. This terminator sequence can be comprised of one or two ASCII characters and may include detection of an EOI signal. Any combination of ASCII characters and EOI may be selected, including no termination or EOI alone.

To change the Bus Input Terminator, press → to the **1st** prompt. Enter the desired ASCII character using the format described in the help screen. Do the same for the second character. If no second character is desired, leave the space blank. Then, press → to advance to the EOI prompt. Press ↓ or ↑ to select **ON** or **OFF**.

The **BUS Output Terminator** sequence delimits the data transferred from the KM-488-DD driver to the GPIB bus. This terminator sequence is comprised of one or two ASCII characters and may include an asserted GPIB EOI signal. Any combination of ASCII characters and EOI may be selected, including no termination or EOI alone. If only EOI is used to signal the end of an output data transmission, the EOI signal will be asserted when the last data character is transmitted. Otherwise, the terminator sequence character(s) is appended to the data characters and EOI, if used, is asserted on the last terminator character sent.

The Bus Output Terminator Sequence is selected in the same manner as the Bus Input Terminator Sequence.

If you are programming via the Call Interface and have completed configuring your KM-488-DD(s) and selecting the default terminator sequences, you are ready to save your configuration parameters to the *CONFIG.DAT* file. Proceed to Step 5. (You can also go to step 5 if you don't want to set-up any Named Device structures.) Otherwise, continue to step 4.

4 - Configuring the Named Devices

Named Devices can be used if you are using the File I/O Command Interface.

Create a New Device and Give It a Name. Each device must be assigned a name. This name will be used to identify the set of characteristics which you have selected. The name can be from 1 to 8 characters long. Press [Enter].

☞ ***NOTE:*** Do not name "NAMED DEVICES" the same as any directory on the current working disk. For example, if you have a directory called **C:\PROGS\DMM**, do not create a named device called DMM.

Assign a GPIB Primary Address. The device must be assigned a GPIB Primary Address in the range 00 to 30. To define the GPIB Primary Address, enter each digit separately. Then, press [Enter].

Assign a GPIB Secondary Address, If desired. You can also assign the device a secondary address. This secondary address must fall in the range 00 to 31. To define the address, enter each digit separately. Then, press [Enter].

GPIB Bus Terminators. As explained above, the **BUS Input Terminator** sequence marks the end of data transferred from the GPIB bus to the KM-488-DD driver. This terminator sequence can be comprised of one or two ASCII characters and may include detection of an EOI signal. Any combination of ASCII characters and EOI may be selected, including EOI alone.

To change the Bus Input Terminator, press [→] to the **1st** prompt. Enter the desired ASCII character using the format described in the help screen. Do the same for the second character. If no second character is desired, leave the space blank. Then, press [→] to advance to the **EOI** prompt. Press [↓] or [↑] to select **ON** or **OFF**.

Remember that the **BUS Output Terminator** sequence delimits the data transferred from the KM-488-DD driver to the GPIB bus. This terminator sequence is comprised of one or two ASCII characters and may include an asserted GPIB EOI signal. Any combination of ASCII characters and EOI may be selected, including EOI alone. If only EOI is used to signal the end of an output data transmission, the EOI signal will be asserted when the last data character is transmitted. Otherwise, the terminator sequence character(s) is appended to the data characters and EOI, if used, is asserted on the last terminator character sent.

The Bus Output Terminator Sequence is selected in the same manner as the Bus Input Terminator Sequence.

5 - Saving Your Configuration Parameters to CONFIG.DAT

When you have checked all of your configuration parameters and are ready to save them to the *CONFIG.DAT* file, select **SAVE**. Then, enter the appropriate filename, when prompted. The contents of this file can be reloaded by selecting the **LOAD** from the top menu.

PART 3 - DRIVER INSTALLATION

The KM-488-DD driver consists of two pieces of loadable/unloadable TSR (Terminate Stay Resident) software. These are the **VI.EXE** (or optionally the **VIPARSE.EXE**) driver and the **KM.EXE** driver.

The **VI.EXE** driver is a support program which provides the graphics display routines required by the *CONFIG.EXE* and *CFGFILES.EXE* program. It also provides other software routines required by the *KM.EXE* driver. If you are not going to use the *CONFIG.EXE* program after the initial software configuration or you find that the *VI.EXE* program takes up too much memory for your application, you can alternatively load the *VIPARSE.EXE* driver, which is much smaller yet adequate for supporting *KM.EXE.*

The **KM.EXE** driver is the software driver for the KM-488-DD. The *KM.EXE* driver reads in the contents of the software configuration file, *CONFIG.DAT,* which was created when the *SETUP* program was run. The *CONFIG.DAT* file is re-generated every time the *CONFIG.EXE* program is run or may be modified manually. The *CONFIG.DAT* program is discussed in more detail in Appendix F.

When the *KM.EXE* driver is loaded, it creates and installs three DOS devices for each KM-488-DD board configured. (It is important that you properly configure every KM-488-DD board using the *CONFIG.EXE* program. This is the only way the *KM.EXE* driver will be able to recognize the number of KM-488-DD's used in your system.) Because BASICA and QuickBASIC require a different device for input and output operations, two devices are created for use with those languages. The third device is one which can be used in both input and output operations. The device names are shown in Table 2-4. The use of these devices is discussed further in Chapter 4.

Table 2-4. Assigned DOS Devices

Board Number	Input/Output Device	BASIC Input Device	BASIC Output Device
0	IEEE	IEEEIN	IEEEOUT
1	IEEE2	IEEEIN2	IEEEOUT2
2	IEEE3	IEEEIN3	IEEEOUT3
3	IEEE4	IEEEIN4	IEEEOUT4

The *VI* and *KM* drivers are TSR programs can be loaded manually from DOS or can be loaded from your *AUTOEXEC.BAT* file or another batch file. One advantage to loading the drivers from a batch file (such as *AUTOEXEC.BAT*) is that the software is automatically available once your computer has been powered-up. This is useful if you are frequently going to run GPIB programs. However, the drivers do occupy memory and may interfere with other programs you use. If this becomes a problem or you are only running GPIB programs once in a while, you will probably want to load the drivers from the DOS prompt. You can also elect to unload the drivers, while using memory intensive software, as described in Section 2.9.

2.7 Loading the KM-488-DD Driver from AUTOEXEC.BAT

When you load the driver via the *AUTOEXEC.BAT* file, your files will remain loaded until you shut down the computer or manually unload the drivers from DOS. (See section 2.9.) Note, however, that if you unload the drivers from DOS and need to re-activate them, you will have to re-load the drivers from the command line as described in section 2.8. If they are loaded via the *AUTOEXEC.BAT* file, they will automatically be re-loaded each time the computer is powered-up.

If you choose to load the drivers directly from the *AUTOEXEC.BAT* file, start by copying your existing *AUTOEXEC.BAT* file to a back-up file (e.g., *AUTOEXEC.BAK*). Then, bring the *AUTOEXEC.BAT* file into EDLIN or some other word processor which will handle raw ASCII text. Your *AUTOEXEC.BAT* file might look something like this:

```
path c:\;c:\dos;
prompt = $p$g
c:\mouse\mouse
```

Then:

1. If you want to use the *CONFIG.EXE* program, add a statement to load the VI or VIPARSE driver.

   Add the line **`[D:] [PATH]VIPARSE`**

   or, in the case of VI

   Add the line **`[D:] [PATH]VI  [MONO]`**

Where:

[D:]	is an optional parameter which indicates the DOS drive where *VI.EXE* or *VIPARSE.EXE* is installed. You must only include this if the VI driver is not installed in the root directory.
[PATH]	is an optional parameter which represents the path to the directory in which *VI.EXE* or *VIPARSE.EXE* is installed. You must only include this if the VI driver is not installed in the root directory.
[MONO]	is an optional parameter which must be included for computers with monochrome displays. If this parameter is not include, it is assumed that a color monitor is being used.

2. Then, install the KM driver.

   Add the line **[D:] [PATH]KM<[D:] [PATH]FILE.EXT**

   Where:

[D:]	is an optional parameter which indicates the DOS drive where *KM.EXE* or *FILE.EXT* exists. You must only include this if the corresponding file is not resident in the same directory as the *AUTOEXEC.BAT* file.
[PATH]	is an optional parameter which represents the path to the directory in which *KM.EXE* or *FILE.EXT* resides. You must only include this if the corresponding file is not installed in the same directory as the *AUTOEX-EC.BAT* file.
<	Instructs DOS to "pipe" the contents of **FILE.EXT** into KM at load time.
FILE.EXT	is the name of the configuration file to be "piped" into the KM-488-DD driver. This file is created by the *CONFIG* program and is generally called *CONFIG.DAT*, unless you have specified otherwise.

Your resulting *AUTOEXEC.BAT* file would look something like this:

```
PATH C:\;C:\DOS;
PROMPT = $P$G
```

```
C:\MOUSE\MOUSE
C:\KM488DD\VIPARSE
C:\KM488DD\KM <C:\KM488DD\CONFIG.DAT
```

2.8 Loading the KM-488-DD Driver from DOS

If you elect to load the drivers manually from DOS, you must remember to load the VI driver first, then the KM driver.

To load the VI driver, at the DOS prompt, type:

```
[D:][PATH]VIPARSE
```

or, in the case of VI

```
[D:][PATH]VI  [MONO]
```

Where:

`[D:]`	is an optional parameter which indicates the DOS drive where *VI.EXE* or *VIPARSE.EXE* is installed. You must only include this if the VI driver is not installed in the current directory.
`[PATH]`	is an optional parameter which represents the path to the directory in which *VI.EXE* or *VIPARSE.EXE* is installed. You must only include this if the VI driver is not installed in the current directory.
`[MONO]`	is an optional parameter which must be included for computers with monochrome displays. If this parameter is not include, it is assumed that a color monitor is being used.

For example:

```
C:\KM488DD\VIPARSE
```

or

```
C:\KM488DD\VI
```

Then, install the KM driver.

Type the line **`[D:] [PATH]KM<[D:] [PATH]FILE.EXT`**

Where:

`[D:]`	is an optional parameter which indicates the DOS drive where *KM.EXE* or *FILE.EXT* exists. You must only include this if the corresponding file is not resident in the current directory.
`[PATH]`	is an optional parameter which represents the path to the directory in which *KM.EXE* or *FILE.EXT* resides. You must only include this if the corresponding file is not installed in the current directory.
<	Instructs DOS to "pipe" the contents of **`FILE.EXT`** into KM at load time.
`FILE.EXT`	is the name of the configuration file to be "piped" into the KM-488-DD driver. This file is created by the *CONFIG* program and is generally called *CONFIG.DAT,* unless you have specified otherwise. Note that if the configuration file is no specified, the software will use the default values.

For example:

```
C:\KM488DD\KM<C:\KM488DD\CONFIG.DAT
```

2.9 Unloading the KM-488-DD Driver from DOS

To unload the KM-488-DD driver software, first unload *KM.EXE* and then the VI software. For example, make sure your path name is correct, then at the DOS prompt, type:

```
KM /U
```

Then:

`VI /U` or **`VIPARSE /U`**

PART 4 - SOFTWARE RE-CONFIGURATION

Before you re-configure the software make sure the *VI* driver has been loaded, this can be done manually (See section 2.7) or via your *AUTOEXEC.BAT* file (See section 2.8). To reconfigure, the software:

1. Switch to the directory where the KM-488-DD software is loaded.

2. At the DOS prompt, type:

   `CONFIG`

3. Load the existing software configuration file (for example, *CONFIG.DAT*) by selecting the **LOAD** option. Then, enter the appropriate filename when prompted.

4. Modify the necessary parameters by using the same method described in Section 2.5, steps 2 through 4.

CHAPTER 3
INTRODUCTION TO PROGRAMMING VIA THE FILE I/O COMMANDS

Chapter 5 contains a detailed description of all of the KM-488-DD commands that are available for use through the DOS file interface. Chapter 4 deals with the idiosyncrasies of employing the commands in the various languages supported (BASICA, C, Quick BASIC, Microsoft PASCAL and TURBO PASCAL). This chapter gives an overview of the file interface and an introduction to the more common commands that will be used in nearly every application.

3.1 THE DOS FILE INTERFACE

When the KM-488-DD driver is installed, it creates three DOS devices for each KM-488-DD board(card) specified in *CONFIG.DAT.* In addition, one file is created for each NAMED GPIB device. Likewise, DOS creates devices associated with other drivers such as disks, COM ports, or printer ports. To keep track of the open devices (or files), DOS makes entries in a **table**. This **table** must be large enough to hold the required number of open files; but if it is larger than required , it is wasting computer memory. DOS is optimized at power-up through entries in the *CONFIG.SYS* file. The entry which controls the number of files which can be opened is: **`FILES`** = *n* where *n* is some number. If errors are encountered about insufficient files, check your *CONFIG.SYS* file . A good compromise is **`FILES=20`**.

Once the driver is installed, it is controlled by commands consisting of strings of characters which are sent through the DOS file interface. The contents of these strings are discussed in detail in Chapter 5. The way these commands are sent and received from the various supported programming languages is discussed in Chapter 4. As a side item, because the KM-488-DD is a DOS device then, commands and data can be sent to the KM-488-DD driver from the DOS command prompt by using the DOS **`ECHO`** command and data returned from the driver by means of the DOS **`TYPE`** command. As an example, consider the RESET command.

RESET

When starting a program, it is a good idea to reset the driver by issuing a REST command to the driver which returns the driver to its default conditions. If the KM-488-DD is also the System Controller, the GPIB IFC line will be asserted for 500 µsecs. From the DOS prompt, the KM-488-DD device can be reset by issuing the following:

ECHO RESET>IEEE Then press (Enter).

If the KM driver has not been previously installed, DOS will write RESET to a disk file called IEEE, creating the file if one does not already exist. The presence of such a file does not cause any problem with the driver installed, because DOS has a hierarchy and will try to send a command to a driver before writing to a file. These erroneous files can always be deleted. It is necessary to delete them before installing the driver, however. Otherwise, DOS thinks you are trying to delete a driver and will not allow it.

For a more complicated example, consider the following monitor display. This is entered from the DOS prompt:

ECHO HELLO>IEEE> Then press (Enter).
TYPE IEEE Then press (Enter).

copyright (c) 1991 Keithley MetraByte Corp will be returned.

HELLO is one of the commands discussed in Chapter 5. Pressing (Enter) in the ECHO line of the above screen display uses the string HELLO followed by CR,LF (Carriage Return, Line Feed) to be sent to the driver. The driver recognizes this as a command places the copyright message followed by a carriage return line feed in a driver buffer associated with the device IEEE. The DOS **TYPE** command reads the message from the IEEE device and displays it on the monitor.

Observe that the above screen display remains on the monitor for a few moments and then scrolls off the top. Depending on the FILL command used, the screen will fill with some character or blanks. It will be necessary to press (Ctrl)(Break) to regain control and return the DOS prompt to the monitor. The reason for this behavior is discussed in the following section.

FILL

The DOS **TYPE** command does not realize that the CR,LF appended to the end of the copyright message means an end of message. In fact, **TYPE** is used to display large files of text with lots of CR, LF's. When left on its own, **TYPE**will continue to ask for characters from a driver until it receives the ASCII character 26 (CTRL Z), the End of File designator. The driver, on the other hand, knows that its buffer has been emptied and considers a further request to be an OUT OF DATA error. The way the driver reacts to an OUT OF DATA error is determined by the configuration and can be modified by means of the FILL command.

The default for the FILL command is to return the NULL character (ASCII character 0). This may be appropriate for a C language program where a NULL signifies the end of the string; but, not for DOS' **TYPE** which sends the NULL to the monitor and asks for another character. Since NULL is displayed as a a blank space, eventually the screen in blanked.

To correct the problem, the FILL character can be changed to CTRL Z, for example from the DOS prompt, type;

ECHO FILL $26>IEEE Then press Enter.
ECHO HELLO>IEEE Then press Enter.
TYPE IEEE Then press Enter.

copyright (c) 1991 Keithley MetraByte Corp will be returned, followed by:

```
C:\_
```

In the above example, the driver has been instructed to respond by returning CTRL Z every time it is asked for data and its buffer is empty. The **TYPE** command recognizes CTRL Z to mean that there is no more data and returns to the DOS prompt. Note there is a blank line between message and the prompt because of the addition of the CR,LF appended to the message by the KM-488-DD driver.

EOL

The optional characters that are appended to the strings sent from the application program to the driver or returned from the driver to the application program are called **End of Line** (EOL) terminators. The terminators send by the application program are called **output** terminators and those returned to the application program are called **input** terminators. The EOL terminators can be set up by *CONFIG.DAT* or altered from within a program by the CONFIG. or EOL commands. The defaults are CR LF which are appropriate to most programming languages.

Obviously, the EOL IN terminators are not appropriate for the DOS **TYPE** command; but a CTRL X would be. Consider the following monitor display (*Note:* Enter this at the DOS prompt.):

ECHO EOL IN $26>IEEE Then press Enter.
ECHO HELLO>IEEE Then press Enter.
TYPE IEEE Then press Enter.

`copyright (c) 1991 Keithley MetraByte Corp` will be returned, followed by:

`C:\>_` Your DOS prompt.

Note in this case **`TYPE`** returns to a DOS prompt on the line immediately following the copyright message because CTRL Z instead of CR,LF was appended to the copyright message. Note also that **`TYPE`** will cease asking for character when CTRL Z is returned so the error condition described in the previous section is never reached and the FILL character is irrelevant.

Refer to Chapter 5 for a discussion of the options available for EOL and Chapter 4 for the choice appropriate to a particular language. Usually if one programming language is being used, EOL can be determined at configuration and never changed.

3.2 SELECTING THE GPIB BUS TERMINATORS

The GPIB BUS Terminators are used to delimit data transmitted between the KM-488-DD driver and the GPIB Bus. There are two types of GPIB BUS Terminators: **Input** and **Output**.

The **GPIB BUS Output Terminator** sequence delimits the data transferred from the KM-488-DD driver to the GPIB bus. This terminator sequence is comprised of one or two ASCII characters and may include an asserted GPIB EOI signal. Any combination of ASCII characters and EOI may be selected, including EOI alone. If only EOI is used to signal the end of an output data transmission, the EOI signal will be asserted when the last data character is transmitted. Otherwise, the terminator sequence character(s) is appended to the data characters and EOI, if used, is asserted on the last terminator character sent.

If the data to be sent to the GPIB bus passes through the File Interface (as opposed to being a buffered transfer), the application program will typically append output EOL terminators to the data. The EOL output terminator is intercepted by the driver and not sent to the GPIB bus. If GPIB output terminator characters are used, they will immediately follow the data as shown in Figure 3-1. Frequently both the GPIB and EOL terminators are CR LF so the switch is not obvious.

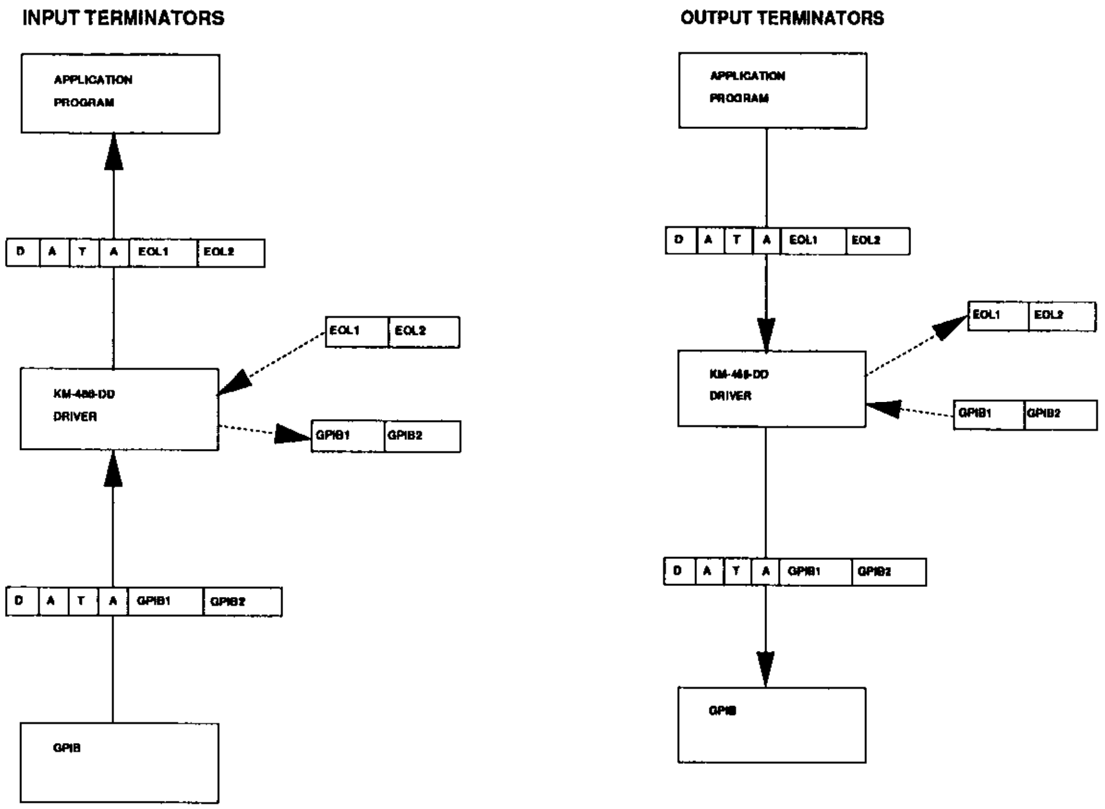


Figure 3-1. Terminators used with the File I/O Commands

The **GPIB BUS Input Terminator** sequence marks the end of data transferred from the GPIB bus to the KM-488-DD driver. This terminator sequence can be comprised of one or two ASCII characters and may include detection of an EOI signal. Any combination of ASCII characters and EOI may be selected, including EOI alone.

If only EOI is chosen to signal the end of input data, then all the characters received from the bus device, including the one on which EOI was asserted will be returned to the application program.

If only ASCII characters are used, then all the characters up to, but not including, the terminator characters, will be returned.

However, if ASCII characters are selected in combination with EOI, the following will occur:

- If EOI is received and the complete ASCII terminator sequence has not been received, all of the received characters are returned to the program.
- If the complete terminator sequence has been received, then all of the data excluding the terminator sequence will be returned.

If the data received from the GPIB bus passes through the File Interface (as opposed to being a buffered transfer) to the application program, the driver will typically strip the GPIB terminators and append the appropriate input EOL terminators to the data as shown in Figure 3-1. Frequently both the GPIB and EOL terminators are CR LF so the switch is not obvious.

Because it is sometime convenient to send data in two "pieces" without terminators between the pieces and because different GPIB devices use different GPIB terminators, there will be cases where it will be necessary or desirable to change the GPIB terminators within a program.

The OUTPUT command allows the same data to be sent to multiple GPIB devices. Because the OUTPUT command can only append one set of terminators, care must be taken that all of the listening devices will respond to the terminators used.

TERM

This command is used to change the default terminators used when transferring data between the KM-488-DD driver and a GPIB bus device. The default input and output GPIB BUS terminators are initially set by the CONFIG program. The sequence CR LF EOI is appropriate for most GPIB bus devices.

3.3 RE-CONFIGURING THE DEVICES

The operating parameters of the KM-488-DD can be re-configured at run-time by issuing a CONFIG command.

CONFIG

This command re-configures the operating parameters of the KM-488-DD. These include:

- GPIB Bus Terminators
- Named Devices
- GPIB Bus Address
- EOL Terminators

3.4 NAMED DEVICES

The KM-488-DD allows a GPIB device to be referred to by a name of up to 8 characters. The device created will contain information about the device's GPIB address and terminators which will be employed automatically by the driver as is appropriate to the command involved. Named devices can be set up through the configuration software an by means of the CONFIG command within the program. For example, from the DOS prompt, typing:

`ECHO CONFIG /N DMM 12 /T CR LF EOI>IEEE` followed by Enter

will allow a programmer to refer to a digital multimeter at GPIB address 12 as DMM. Whenever data is sent or received from the meter, the GPIB terminators of CR, LF (Carriage Return, Line Feed) accompanied by the EOI signal will be used.

3.5 CLEARING DEVICES

Devices can be returned to their power-up initial state (ranges and functions) by issuing a CLEAR command. If the clear command contains addresses then only those devices are cleared by means of the addressed GPIB selected device clear. If no addresses are sent, then all the GPIB bus devices are cleared by means of the universal device clear. For example, from the DOS prompt, type:

`ECHO CLEAR DMM>IEEE` followed by Enter

3.6 TRANSMITTING DATA

Once you have selected the appropriate terminator sequences to be used, you will be able to transmit data. Data is transmitted in one of two ways, either through **Direct Output** or **Buffered Output**.

Direct I/O makes use of the OUTPUT command and the data to be sent on the GPIB bus is sent through the File Interface along with the command. For example, to set a Keithley 196 to the ac volts range, send the following from the DOS prompt:

`ECHO OUTPUT DMM;F1X>IEEE` followed by Enter

In this case, the data to be sent on the GPIB bus (F1X) is sent along with the command (OUTPUT DMM;) through the file interface. This is convenient where a limited number of characters (length of data plus the length of command is not more than 255 characters) and the data are normal printable characters.

DOS interprets certain characters such as CTRL Z in special ways, rather than passing the character and those that might follow on to a device such as the monitor (which is controlled by DOS CON device). This feature of DOS can be disabled by putting DOS into RAW MODE via an INT 21 H DOS call. See *IEEEIO.C* for more details on Raw Mode operations.

A better approach to sending any number of any kind of characters is to use the **OUTPUT #count BUFFER buffer_address** command. In this case, the address of a program data area containing the data and the number of bytes to be transmitted on the GPIB bus are sent to the driver with the command. The data is then moved directly from the program memory to the GPIB bus (without passing through DOS) either via the KM driver or by the PC DMA controller, if the DMA option has been invoked. A further CONTINUE option is available for DMA. In this case, the driver does not monitor the transfer, but returns control to the application program. In this case, the application program can perform tasks while the DMA proceeds in the background. At some point, the application program should use the WAIT command to verify that transfer is complete.

Because their data does not have to pass through DOS, buffered transfers are generally faster than unbuffered transfers when the amounts of data to be transferred are large enough to warrant the time required to set up the buffered transfer. Because setting up the data buffers is language-dependent, no example is given here. Refer to the appropriate section of Chapter 4 and the appropriate example programs on the distribution disks to see how buffered transfers are implemented.

3.7 RECEIVING DATA

Once you have selected the appropriate GPIB terminator sequences to be used, you will be able to receive data. Data is received in one of two ways, either through Direct Input or Buffered Input.

Direct I/O makes use of the ENTER command and the data to be received from the GPIB bus is received through the File Interface. For example, to return a reading from a Keithley 196, the ENTER command is issued which tells the driver to receive data from a GPIB bus device. The driver places the data in a driver buffer where the application program can retrieve it through DOS. For example, from the DOS prompt, type:

ECHO ENTER DMM>IEEE followed by [Enter]
TYPE IEEE followed by [Enter]

NACV + 000.0163E+0 or something like it will be returned.

The same limitations of direct data transfer discussed in the previous section for OUTPUT also apply to ENTER. Again buffered inputs (with or without DMA)overcome these deficiencies. To do a buffered input , use the **`ENTER #count BUFFER buffer_address`** command. Refer to the previous section for general information on buffered transfers and to the appropriate section of Chapter 4 and example programs on the distribution disks for information pertaining to your programming language.

3.8 TIMEOUT

A TIMEOUT check is available to ensure that the commands which perform handshake data transfers on the GPIB bus (i.e., ENTER, ENTER #count BUFFER buffer_address, OUTPUT, OUTPUT #count BUFFER buffer_address, SEND, and SPOLL), complete their data transfer within a limited amount of time. This prevents the application program and/or GPIB bus from hanging because a device was turned off or failed to function. The timeout can be modified by the timeout COMMAND. For example, to set the timeout to 8.6 seconds, from the DOS prompt type:

`ECHO TIMEOUT 8.6 >IEEE` followed by Enter

The timeout can be disabled by sending 0 for the timeout value; but this is not recommended. The timeout should be long enough so that a data transfer can complete. This will depend on the speed of the GPIB device and the amount of data to be transferred. In normal buffered transfers or DMA without CONTINUE, this includes all the characters transferred. In a DMA with CONTINUE, the driver only monitors the first character transferred so the timeout applies only to that character. After the first character, the transfer occurs automatically while the application program perform other tasks. At some point, the application program will want to verify that the transfer is complete by issuing a WAIT command. When the WAIT command is issued, the timeout function will start. Alternatively, the buffered command can be used.

3.9 BUFFERED

The BUFFERED command can be used to determine the number of character transferred in the last buffered transfer. This is particularly useful where a DMA with CONTINUE has been initiated and the application program has worked on other tasks. By issuing the BUFFERED command, the application program can determine the number of bytes transferred thus far. If the number of transferred characters equals the expected number to be transferred, the application program knows the transfer is complete. Otherwise, the application program can check back later or issue a WAIT and wait for the completion or a time-out.

3.10 TRIGGER

GPIB devices can be triggered by a GPIB command to perform some function such as to make a measurement. The Keithley 196 can be configured to make a measurement when triggered and generate an SRQ (See the next section.) when the measurement is complete. The following sequence initializes the 196 to that mode and triggers it to perform the measurement. For example, from the DOS prompt type:

`ECHO OUTPUT DMM;T3M8X>IEEE` followed by Enter
`ECHO TRIGGER DMM>IEEE` followed by Enter

3.11 STATUS and SPOLL

Many GPIB bus devices have a status register which can be read by the controller by means of the SPOLL (Serial poll) mechanism. To eliminate the need for the controller to SPOLL a device when no change of status has occurred, many devices can assert an SRQ (service request) line on a status change. In this way, the controller can be performing other tasks while devices do not need attention. Furthermore, some instruments do not take kindly to being queried while they are in the middle of something. Because all bus devices share the same SRQ line, the controller may have to poll several devices before it arrives at the one requesting service.

In the previous example, the 196 has been triggered to make a measurement and assert SRQ when the measurement is complete. An application program can determine when an SRQ is present by looking at the driver status. When the status reveals the presence of an SRQ, a serial poll can be conducted which returns a status byte from the device polled. For example, from the DOS prompt you would enter:

`ECHO OUTPUT STATUS>IEEE` followed by Enter

`TYPE IEEE` followed by Enter

`C 00 G1 T S1 E00 T0 C0 P0 OK` will be returned.

`TYPE IEEE` followed by Enter

`88` will be returned.

ECHO ENTER DMM>IEEE followed by Enter

TYPE IEEE followed by Enter

NACV+000.0163E+0 or something like it will be returned.

☞ *NOTE:* The commands BUFFERED, ENTER, HELLO, PPOLL, SPOLL, and STATUS cause the driver to generate data to be returned to the application program. It is necessary to return the data from one of these commands (such as by **TYPE IEEE**) before a command from the group is issued again. Otherwise, errors will occur.

A discussion of the returned status string is provided in Chapter 5, in the STATUS command description. The result of the SPOLL indicates the 196 is asserting SRQ as indicated by DIO7 being asserted. The act of serial polling the 196 cause the 196 SRQ to be cleared so a subsequent SPOLL would return 24. The meaning of the 24 is unique to the 196 and indicates that the 196 has completed a measurement and is ready for another command.

3.12 LIGHTPEN INTERRUPTS

The KM-488-DD board is capable of being configured to generate hardware interrupts on receipt of an SRQ. the KM-488-DD driver does not support hardware interrupts, but does implement the lightpen "pseudo interrupt".

To employ the lightpen interrupt, the driver must be configured to accept interrupts at load time via the *CONFIG.DAT* file. The driver will periodically check for the SRQ and set a flag that is normally set when a lightpen is activated, if it detects the presence of an SRQ (It also can flag many other conditions.). The application program must then check for the presence of the lightpen interrupt.

In BASIC and Quick BASIC, if the PEN ON command is issued the lightpen interrupt will be checked at the completion of each BASIC command (A FOR/NEXT loop is considered a single command, so detection does not take place within a FOR/NEXT loop). If an interrupt is detected, the program can jump to a routine which services the interrupt and then return to the main routine. In other languages, it is necessary to decide your own strategy for checking the lightpen interrupt.

3.13 PPOLL and PPC

A few GPIB devices, such as the Sorenson HPD30-10 power supply, support a parallel poll mode of operation. A device which support parallel polling can be configured to assert one particular GPIB data line when the controller conducts the parallel poll. If different devices assert different data lines, the controller can rapidly determine which device needs service by issuing a parallel poll and then interpreting the result.

Many times the device's parallel response is asserted when the device asserts SRQ. In most cases, the controller will generally perform a serial poll of the device(s) which a parallel poll has revealed needed servicing. This is to obtain more complete status information and to cause the device to unassert its SRQ and parallel responses.

The following monitor display results from a parallel poll example for the Sorenson power supply at GPIB address 01. The Sorenson is configured to generate an SRQ when it receives an over-range value. It is also configured to generate a parallel response on the third data line when an SRQ is generated and then sent an out of range value. From the DOS prompt, you would enter:

`ECHO SPOLL 01 >IEEE` followed by [Enter]

`TYPE IEEE`

`66` will be returned.

`ECHO OUTPUT01;MSK 02>IEEE` followed by [Enter]

`ECHO PPC01; &H0A>IEEE` followed by [Enter]

`ECHO OUTPUT01;V 80.5>IEEE` followed by [Enter]

`ECHO STATUS>IEEE` followed by [Enter]

`TYPE IEEE`

`C 00 G1 T S1 E00 T0 C0 P0 OK` will be returned.

`ECHO PPOLL>IEEE` followed by [Enter]

`TYPE IEEE`

4 will be returned.

ECHO SPOLL 01>IEEE followed by Enter

TYPE IEEE

66 will be returned.

ECHO PPOLL>IEEE followed by Enter

TYPE IEEE

0 will be returned.

The power supply generates an SRQ and a parallel poll reveals the third line is asserted by returning the value 4. The SPOLL clears the SRQ and thus, the parallel poll response of the Sorenson.

3.14 REQUEST

The KM-488-DD can function in the non-controller or peripheral mode. In this mode, the KM-488-DD can set its serial poll response which can be read by a controller in charge. The seventh bit, when set, is reserved to generate an SRQ when desired. The significance of the other bits would be defined by the application. An SRQ without an address can be used to determine if the SRQ has been serviced. If the value returned is 64, the SRQ is still pending, if 0 the SRQ has been serviced by the controller and the KM-488-DD has automatically stopped asserting the SRQ.

You must first unload the KM-488-DD driver and then reload it as a peripheral before verifying the following example. At the DOS prompt, type:

ECHO EOL IN $26>IEEE followed by Enter

ECHO STATUS>IEEE followed by Enter

TYPE IEEE

P 00 G1 I S0 E00 T0 C0 P0 OK will be returned.

ECHO REQUEST 64>IEEE followed by Enter

ECHO SPOLL>IEEE followed by [Enter]

TYPE IEEE

64 will be returned.

The P returned by STATUS verifies the KM-488-DD is a peripheral and the I indicates that it is in an idle state (not addressed to talk or listen). Sending a REQUEST with 64 sets the SRQ and reading back a 64 indicates the SRQ has not been serviced.

CHAPTER 4
USING THE FILE I/O COMMANDS

Chapter 3 introduced communicating with the KM-488-DD driver through the FILE interface. Chapter 5 contains a detailed description listed in alphabetical order of all the commands available. This chapter deals with the peculiarities of using the FILE interface in the programming languages supported.

4.1 IN BASICA

Driver Configuration

EOL IN - CR LF

EOL OUT - CR LF

FILL - ERROR

Preparing the Environment

Opening the KM-488-DD driver's I/O devices (IEEEIN, IEEEOUT, etc.) uses up BASIC's limited default number of file handles. Usually, the number of files available for a program is small (i.e. GW-BASIC Version 3.2 allows two user files). This is because BASIC needs at least three file handles just to support the DOS STDERR, STDIN, STDOUT, STDAUX and STDPRN devices. If your program needs several file handles beyond the two normally used for IEEEIN and IEEEOUT, you must invoke the BASIC interpreter with the :

```
BASICA /F:num
```

option, which will allocate data space for additional file opens. Consult your particular BASIC manual for the correct value of **num**.

Opening the Driver

```
290 'Open the driver to receive commands from this
program
300 OPEN "IEEEOUT" FOR INPUT AS #1
310 '
320 'Reset the Driver
330 IOCTL #1, "BREAK"
340 '
350 'Open the driver to return data to this program
360 OPEN "IEEEIN" FOR INPUT AS #2
```

If the KM-488-DD driver is not installed, an error will result in Line 300 and DOS will open a file called *IEEEOUT*. Exit BASICA, delete the file *IEEEOUT* and install the driver. After the driver has been opened for output, the BASICA program can send it commands. The IOCTL will abort any activity previously going on and clear the buffer for data to be returned. RESET resets the driver to its default conditions.

Users with more than one KM-488-DD card installed will have to open other devices to access the drivers for those cards. Consult the following table for the appropriate device names:

Board Number	BASIC Input Device	BASIC Output Device
0	IEEEIN	IEEEOUT
1	IEEEIN2	IEEEOUT2
2	IEEEIN3	IEEEOUT3
3	IEEEIN4	IEEEOUT4

Once the files have been opened properly, the BASICA **PRINT** or **INPUT** command is used to pass through the File I/O Command and communicate with the KM-488-DD board. The **PRINT** command is used to pass commands to the KM-488-DD and the **INPUT** command is used to read data from the KM-488-DD. If an error occurs, the KM-488-DD's driver will return an error code and error description. These error codes are listed in Appendix F.

The EOL Input Terminator delimits the end of data which is passed from the KM-488-DD to the application program, during an ENTER routine. If, however, a character count is specified in the ENTER, only that number of characters will be returned. (An EOL will not be appended to the data.) In this instance, you will not be able to read the data by using the BASICA **INPUT** function. You will have to use the BASICA **INPUT$(count, file)** function to read a specified number of characters, for example:

```
210 PRINT#1,"ENTER 16#10"
220 A$=INPUT$(10,2)
```

Typical Code Sequence

Refer to the example programs on the distribution disks.

Utilities for Buffered Transfers

The KM-488-DD driver supports four utilities to overcome certain BASICA deficiencies. These utilities are required for buffered transfers. The utilities are described in section 4.3. They include *GET.SEGMENT, PACK.BYTES, UNPACK.BYTES*, and *SWAP.BYTES*.

These driver utilities are accessed as CALLs from the BASICA program. BASICA normally works with its own code area; but can jump to routines outside its own environment through a segment and an offset. In this case, the segment will be the location of the beginning of the driver in memory and can be found by using the driver CODE ADDRESS command (See section 4.2). The value returned by this command is used with the BASICA **DEF SEG** command to define the code segment. The particular place for a specific CALL to jump within the driver is defined by an offset from the beginning address. The following code segment illustrates defining the code offsets and finding the driver location:

```
100  ' Define the CALL offsets
120 GET.SEGMENT = 0
130 PACK.BYTES = 5
140 UNPACK.BYTES = 10
150 SWAP.BYTES = 15
200 'The KM-488-DD driver has already been opened
210  PRINT #1, "CODE ADDRESS"
220 'Ask for segment location of driver
230 INPUT #2, IEEESEG%
'Return segment location of driver
240 '
250 'Use BASICA DEF SEG to set code segment to
driver location
260 '
270 DEF SEG = IEEESEG%
280 '
290 'Call routine in driver to return segment
location of
300 'BASICA's data memory.
310 '
320 DSEG% = 0
330 'All arguments of a BASICA call must be allo-
cated before
331 ' before the call DSEG% can be used in
buffered transfers.
340 CALL GET.SEGMENT(DSEG%)
341 'DSEG% can be used in buffered transfers
```

☞ *NOTE:* All arguments of a BASICA CALL must be allocated before the CALL. Setting **DSEG% = 0** in line 320 provides a storage space. Otherwise, the driver may write the returned value to another area.

Buffered Transfers

Buffered transfers are normally used to send large amounts of data and will frequently represent numerical values. Thus, you can use integer arrays. (Refer to section 4.2). It is possible to send characters; but note that if transfers involving strings are used, they are limited to 256 characters in BASICA.

Data buffers are located by a segment and offset in BASICA. The segment location is obtained by means of GET.SEGMENT and the offset is determined by the BASICA function **VARPTR**. Note that **VARPTR** should be called immediately before the File I/O command.

Refer to the example program *BDMAF.BAS* for an example of buffered transfers.

Data Limitations

It is important to note that BASIC is limited to a single 64K segment of data. This data space is shared by all data items (arrays, strings, variables) and the names and internal descriptors used by BASIC to manage its own memory. This means that any BASIC program will normally be limited to 64K of data items. It is possible to input and output data to absolute addresses with commands such as the following:

```
100 PRINT #1, "ENTER 05 #100 BUFFER &H8000:0"
```

and then view the data using the BASICA **DEF SEG** and **PEEK** directives. It should be noted that this method makes no assumptions about the current contents of the memory at address &H8000:0, and that such an operation may write over existing BASICA data, or the code in the BASICA interpreter itself, thereby causing corrupted data and/or program crashes. For applications requiring more than 64K of data space and not using DMA, it is recommended that you perform repeated reads into smaller arrays or switch to a language that supports FAR data areas, such as QuickBASIC.

4.3 ADDITIONAL BASICA ROUTINES

The following routines can be called directly from BASICA and QuickBASIC.

CODE ADDRESS

Purpose Returns the segment address that points to the GET.SEGMENT, PACK.BYTES, and UNPACK.BYTES routines.

Syntax `CODE ADDRESS`

Parameters None.

Returns An integer from 0 to 65535 which represents the segment address.

Programming Notes

1. This routine is useful when defining the buffer address sequence before an **`ENTER #count BUFFER`** or **`OUTPUT #count BUFFER`**.

2. This routine needs only to be called once.

GET.SEGMENT

Purpose	Responds with data segment value of the BASIC variable space.
Offset	0
Syntax	`CALL GET.SEGMENT(dataseg%)`
Parameters	None.
Returns	`dataseg%` is an integer variable which is set to the data space segment value.
Programming Notes	This routine is useful when defining the buffer address sequence before an `ENTER #count BUFFER` or `OUTPUT #count BUFFER`.

IOCTL

Purpose When followed by **BREAK**, it resets the KM-488-DD and its EOL terminators to their defaults.

Syntax `IOCTL#2,"BREAK"`

Returns None.

Programming Notes None.

IOCTL$

Purpose This is a BASIC function which, when used with the **IOCTL Read DOS-** command, can be used to define the current communication status of the KM-488-DD. The **IOCTL$** function can be utilized in other languages by using MS-DOS FUNCTION calls.

☞ *NOTE:* **BREAK** represses lightpen interrupt emulation so that the next lightpen status request returns a "no interrupt status". Then, lightpen emulation is restored.

Syntax **A$=IOCTL$(#2)**

Parameters **A$** is a string variable to which the value is returned.

Returns When executed, this routine returns one of the following values into the **$A** variable:

0	This indicates that the KM-488-DD is ready to receive a command.
1	This indicates that the KM-488-DD has data available for reading by the application program. This must be read before the program sends a new command (except **IOCTL BREAK**) or **SEQUENCE-DATA HAS NOT BEEN READ** error will occur.
2	This indicates that the KM-488-DD is waiting for data to output to the IEEE bus. This data must be sent (via the **PRINT** statement or equivalent) equipped with terminators. If the application program tries to read data from the KM-488-DD before it is ready will cause a **SEQUENCE-NO DATA AVAILABLE** error.
3	This indicates that the KM-488-DD is waiting for the application program to execute a command.

Programming Notes None.

PACK.BYTES

This is a BASIC callable subroutine which packs discrete 8-bit data values (stored in a 16-bit integer array) into sequential bytes. This routine is useful when an instrument can only accept an 8-bit data value.

Syntax `CALL PACK.BYTES(source%(0),dest%(0),count%)`

Offset 5

Parameters **source%** is the name of the integer array which contains the data to be converted.

dest% is the name of the integer array in which the converted data is to be stored.

count% is an integer which represents the number of elements comprising the **source%** array. (This is also the number of elements contained in the **dest%** array.)

Returns Result is stored in **dest%**.

Programming Notes None.

SWAP.BYTES

Purpose Exchanges the bytes contained in the elements of an integer array.

Offset 15

Syntax `CALL SWAP.BYTES(array%(0),count%)`

Parameters **`array%`** is the name of the integer array whose bytes are to be "swapped".

`count%` is an integer representing the number of elements contained in the integer array (**`array%`**).

Returns **`array%`** will contain the newly formatted integer array.

Programming Notes

1. This routine exchanges the bytes which comprise the elements of an array of integer variables. It may be required to perform a "byte swap" when exchanging data between the KM-488-DD and various IEEE-488 bus device.

   "Byte Swaps" are necessary because the KM-488-DD stores integer variables as two consecutive bytes. The lower byte contains the eight least significant bits of the integer. Likewise, the higher byte contains the eight most significant bits. Unfortunately, many IEEE-488 devices will transfer the most significant byte of an integer variable first. Therefore, the bytes which constitute the integer are opposite of the format in which the KM-488-DD stores them. It is necessary to use SWAP.BYTES before sending to, or after receiving from this type of device.

2. SWAP.BYTES must be called before a data transfer between devices which store data in the opposite manner occurs.

UNPACK.BYTES

Purpose Converts an integer array to consecutive 16-bit integers.

Offset 10

Syntax **`CALL UNPACK.BYTES (source%(0),dest%(0),count%)`**

Parameters **`source%`** is the name of the integer array which contains the data to be converted.

`dest%` is the name of the integer array in which the converted data is to be stored.

`count%` is an integer which represents the number of elements comprising the **`source%`** array. (This is also the number of elements contained in the **`dest%`** array.)

Returns **`dest%`** will contain the newly converted data.

Programming Notes None.

4.3 IN QUICKBASIC

Driver Configuration

EOL IN - CR LF

EOL OUT - CR LF

FILL - ERROR

Versions Supported

QuickBASIC 4.0 and higher

Preparing the Environment

To program the KM-488-DD driver from within the QuickBASIC environment, invoke QuickBASIC with the "/l" option (load Quick Library Module) followed by either *KM488QB4.QLB* (QuickBASIC 4.5 and older) or *KM488QB7.QLB* (QuickBASIC 7 and newer) and the name of your program. For example :

```
QB  /lKM488QB4 MYPROG.BAS(For A QuickBASIC 4.0 User)
QBX /lKM488QB7 MYPROG.BAS(For A QuickBASIC 7.0 User)
```

MYPROG.BAS is loaded in conjunction with a .QLB file for two reasons. First, the .QLB file contains KUNPACK, KSWAP and KPACK routines for manipulating byte data stored within integer BASIC arrays. Secondly, *KM488QB4.QLB* contains the function SSEGADD, which is intrinsic to QBX. Therefore, SSEGADD is not included in *KM488QB7.QLB*, as this would result in a multiply defined function. At any rate, the .QLB file format is different for both *QBX.EXE* (BASIC 7.00) and *QB.EXE* (QuickBASIC 4.5 and older) and a working Quick Library for each is provided.

To create an executable *MYPROG.EXE*, link *MYPROG.OBJ* with the appropriate standard .LIB library file -- *KM488QB4.LIB* for QuickBASIC 4.5 and older, *KM488QB7.LIB* for BASIC 7.00 and above users.

Opening the Driver

```
'Open the driver to receive commands from this
program
OPEN "IEEEOUT" FOR OUTPUT AS #1
'
'Reset the driver
IOCTL #1,"BREAK"
PRINT#1, "RESET"
'Open driver to return data to this program
OPEN "IEEEIN" FOR INPUT AS #2
```

If the KM-488-DD driver is not installed, an error will result in the first **OPEN** statement and DOS will open a file called *IEEEOUT*. If this should occur, exit QuickBASIC, delete the file *IEEEOUT* and install the driver. After the driver has been opened for output the program can send it commands. The IOCTL will abort any activity previously going on and clear the buffer for data to be returned. RESET resets the driver to its default conditions.

Users with more than one KM-488-DD card installed will have to open other devices to access the drivers for those cards. Consult the following table for the appropriate device names:

Board Number	QBASIC Input Device	QBASIC Output Device
0	IEEEIN	IEEEOUT
1	IEEEIN2	IEEEOUT2
2	IEEEIN3	IEEEOUT3
3	IEEEIN4	IEEEOUT4

Once the files have been opened properly, the BASICA **PRINT** or **INPUT** command is used to pass through the File I/O Command and communicate with the KM-488-DD board. The **PRINT** command is used to pass commands to the KM-488-DD and the **INPUT** command is used to read data from the KM-488-DD. If an error occurs, the KM-488-DD's driver will return an error code and error description. These error codes are listed in Appendix F.

The EOL Input Terminator delimits the end of data which is passed from the KM-488-DD to the application program, during an ENTER routine. If, however, a character count is specified in the ENTER, only that number of characters will be returned. (An EOL will not be appended to the data.) In this instance, you will not be able to read the data by using the BASICA **INPUT** function. You will have to use the BASICA **INPUT$(count,file)** function to read a specified number of characters, for example:

```
210 PRINT#1,"ENTER 16#10"
220 A$=INPUT$(10,2)
```

Typical Code Sequence

Refer to the example programs on the distribution disks.

Utilities for Buffered Transfers

The KM-488-DD driver supports three utilities to facilitate manipulating data involved in buffered transfers. Buffered transfers move bytes between the GPIB bus and contiguous locations in memory. There are three utilities which aid in manipulating these bytes. They are:

`KPACK(BYVAL source%(0), BYVAL dest%(0),count%)` - This routine packs discrete 8-bit data values (stored in a 16-bit integer array) into sequential bytes.

`KUNPACK(BYVAL source%(0), BYVAL dest%(0), count%)` - This converts an integer array to consecutive 16-bit integers.

`KSWAP (BYVAL array(0), count%)` - This swaps the high and low bytes stored in the integer array.

Buffered Transfers

Buffered transfers are normally used to send large amounts of data and will frequently represent numerical values. Thus, you can use integer arrays. (Refer to section 4.3). It is possible to send characters; but note that if transfers involving strings are used, they are limited to 256 characters in BASICA.

Data buffers are located by a segment and offset in BASICA. The segment location is obtained by means of GET.SEGMENT and the offset is determined by the BASICA function **VARPTR**. Note that **VARPTR** should be called immediately before the File I/O command.

Refer to the example program *BDMAF.BAS* for an example of buffered transfers.

4.3 IN MICROSOFT PASCAL

Driver Configuration

EOL IN - CR LF

EOL OUT - CR LF

FILL - ERROR

Versions Supported

4.0 and later

Preparing the Environment

You must ensure that the driver is installed prior to opening the device IEEE or Microsoft PASCAL will create a disk file called IEEE in the current working directory.

When using the supplied function IEEEOpen, or any of the other functions supplied in the source code files *IEEEIO1.ASM* and *IEEEIO2.PAS*, you should invoke the PASCAL compiler in compile-only fashion, then include the library *KMPASIO.LIB* in the link statement in the following manner:

```
PAS1 MYPROG.PAS ;
       PAS2
       LINK MYPROG ,,, KMPASIO ;
```

Opening the Device

In Microsoft PASCAL, File I/O to and from the KM-488-DD driver is accomplished via the intrinsic PASCAL type TEXT. This type of file handle allows files opened for input and output at the same time. Remember that when you installed the KM-488-DD driver (See Chapter 2.), it created three DOS devices for each installed KM-488-DD board. These devices were IEEE, IEEEIN, IEEEOUT. Because Microsoft PASCAL can use open devices for read and write operations via a single handle, you should only have to use the IEEE device. You can, however, use the IEEEIN and IEEEOUT devices if needed.

The IEEE device can be opened in either of two ways. The first is to open the file via the PASCAL ASSIGN command and assign the IEEE device to a variable of type TEXT. The second is to use the IEEEOpen procedure supplied in the file *IEEEIO2.PAS*. This is accomplished as follows:

```
VAR IEEE : TEXT;
(*  KM-488-DD Device Handle Variable    *)
BEGIN
```

```
(****  Open KM-488-DD Device for Input/Output   ****)
IeeeOpen (IEEE) ;
END.
```

The following is an example of opening the IEEE device yourself, without the aid of the supplied IEEEOpen procedure:

```
 (* Assign Dev. 'IEEE' to Var. IEEE  *)
Assign (IEEE, 'IEEE') ;
(* Set Mode to Read/Write           *)
IEEE.MOD := DIRECT ;
(* Rewind File for Output           *)
Rewrite (IEEE) ;
(* Reset PASCAL's File Pointer      *)
Seek (IEEE, 1) :
```

Programming Notes

1. The *IEEEIO2.PAS* contains several useful procedures which can be called from your application program. These include:

`IEEEWrite ( )`	Write a command string to the selected KM-488-DD device.
`IEEEReadStr()`	Read data from the KM-488-DD driver into a STRING.
`IEEEReadInt()`	Read an integer from the KM-488-DD driver into an INTEGER.
`IEEEOpen()`	Open the device IEEE and assign it to a TEXT variable.

The *IEEEIO1.ASM* file also contains the following:

`IOCTL ( )`	Perform the IOCTL "BREAK" command on the KM-488-DD driver. (See section 4.1., BASIC.)

If your program is to make use of these external procedures, they should be declared either in your main source file, or in an 'include' file so that PASCAL can perform a syntax and argument type-check during the compile process. You should declare these external procedures as follows:

```
PROCEDURE IeeeWrite (VAR Handle:TEXT; CONST Com-
mand: LSTRING) ; EXTERN ;
PROCEDURE IeeeReadStr (VAR Handle:TEXT;  VARS
Buffer: STRING) ; EXTERN;
PROCEDURE IeeeReadInt (VAR Handle:TEXT; VAR Sta-
tus: INTEGER) ; EXTERN;
PROCEDURE IeeeOpen (VAR Handle:TEXT) ; EXTERN;
PROCEDURE IOCTL (VAR Handle:TEXT) ; EXTERN;
```

Once you have successfully opened (or ASSIGNed) the IEEE device for reading and writing, you can then send commands to the driver and receive data either directly from the KM-488-DD driver (for example, via the HELLO command) or from GPIB instruments through the KM-488-DD driver (for example, via the ENTER command).

Writing commands to the KM-488-DD driver can be accomplished either by using the WRITELN command, or by using the IEEEWrite procedure, provided in the *KMPASIO.LIB* library. Direct command writes appear in the following fashion:

```
WRITELN(IEEE, 'HELLO');
Rewrite (IEEE);     (* Rewind after write *)
```

The following is an example of using the supplied IEEEWrite procedure to send a command string to the KM-488-DD driver:

```
IEEEWrite (IEEE, 'HELLO');
```

Notice that a PASCAL REWRITE command is issued following the WRITELN command. This forces your PASCAL program to write the command immediately, rather than buffering your command as PASCAL's file internals would normally do in order to optimize buffered writes to disk files. The procedure IEEEWrite automatically performs the Rewrite operation for you.

If a syntax error or GPIB bus error occurs as a result of writing a command to the KM-488-DD device, the error will appear in the '.ERRS' filed of the TEXT variable directly after the Rewrite command is issued. Here is an example of writing a command checking for an error:

```
WRITELN (IEEE, 'datadatadata');
Rewrite (IEEE);      (* Rewind after write *)
if (IEEE.ERRS ,. 0 ) THEN
WRITELN('ERROR DURING COMMAND : data data data');
```

The following is an example of performing the write using IEEEWrite :

```
IeeeWrite(IEEE, 'SPOLL SCOPE');
IF (IEEE.ERRS <> 0) THEN
WRITELN('ERROR DURING COMMAND : data data data') ;
```

Note that IEEEWrite does not check the '.ERRS' filed of the IEEE TEXT variable after writing your command string to the KM-488-DD driver. Your program should check this record field for a non-zero value just as it should do if you issue a Writeln of the command directly.

2. Note that many of the arguments to the procedures in *KMPASIO.LIB* are all declared as 'VAR XXX' in the case of STRING variables. This type of declaration instructs the PASCAL compiler to pass these arguments by reference, rather than by value. Passing variables by reference passes the address of a variable, rather than the contents of that variable into a procedure. This is always the convention for calling KM-488-DD functions that will return a value via a call.

When performing BUFFERED I/O, as in the ENTER and OUTPUT commands, it is necessary to supply the KM-488-DD driver with a segmented (far) address of the location in which to store or retrieve data to be transferred. PASCAL provides an intrinsic directive for determining the segment and offset of variables. In this example, data is being entered into the character array **sdata** from a device at GPIB address 05. The intrinsic PASCAL 'ADS' directive is used to determine first the segment (**(ADS sdata) .S**) and then the offset (**(DS sdata) .R**) of the character array **sdata**.

```
IEEE.ERRS := 0;
WRITELN(IEEE, 'ENTER 05 #10 BUFFER ', (ADS sda-
ta).S, ':' , (ADS sdata).R');
Rewrite (IEEE);
```

```
if (IEEE.ERRS <> 0) THEN
  WRITELN('Error occurred during ENTER From Device
05');
```

4. The default input and output EOL (End of Line) for PASCAL is the Carriage Return/Line Feed pair. This is the default for the KM-488-DD driver, and should not be changed unless you are running other application programs requiring different EOL characters. For example, a program written in C would possibly have the NULL (0) character as a single character EOL sequence. Should you need to reset the EOL terminator to the default, the IOCTL procedure is the correct mechanism for doing so. The IOCTL command is issued as follows:

   ```
   IOCTL(IEEE) ;
   ```

   The maximum allowable string length for a command string written to any of the KM-488-DD devices is 256 characters, including the two character EOL sequence. Any command requiring more than 256 characters should be broken up into several commands, each of which is issued separately. For example, a lengthy 'SEND' command can be sent as repeated smaller 'SEND' commands:

   ```
   IEEEWrite(IEEE, 'SEND UNT UNL MTA LISTEN 10 DATA
   1, 2, 3, 4');
   ```

   is equivalent to:

   ```
   IEEEWrite(IEEE, 'SEND UNT') ;
   IEEEWrite(IEEE, 'SEND UNL') ;
   IEEEWrite(IEEE, 'SEND MTA') ;
   IEEEWrite(IEEE, 'SEND LISTEN 10') ;
   IEEEWrite(IEEE, 'SEND DATA 1') ;
   IEEEWrite(IEEE, 'SEND DATA 2') ;
   IEEEWrite(IEEE, 'SEND DATA 3') ;
   IEEEWrite(IEEE, 'SEND DATA 4') ;
   ```

   Consult the provided example programs for further details about using the various GPIB control commands accessible via the KM-488-DD driver.

4.4 IN TURBO PASCAL

Driver Configuration

EOL IN - CR LF

EOL OUT - CR LF

FILL - ERROR

Versions Supported

Turbo PASCAL versions 4.0 and higher

Preparing the Environment

To use these supplied functions in the file *IEEEIO.PAS*, you should include the statement `USES IEEEIO ;` in your application program. This statement tells the TURBO PASCAL compiler to include the code in the *IEEEIO.TPU* module in the following form:

```
PROGRAM tdmaf ;
USES crt, dos, IeeeIO ; (* Version 4.0 Users replace
IEEEIO with IEEEIOV4 *)
```

(Note that several of the example programs used the CRT and DOS units supplied with TURBO PASCAL).

Users of version 4.0 of TURBO PASCAL should 'USES' the unit *IEEEIOV4.TPU*, rather than *ieeeio.tpu* which is intended for users of Version 5.0 and above. The *.TPU* file format is different for these two versions of TURBO PASCAL, although the source code used to create the two *.TPU* modules is identical. To create the file *IEEEIO.TPU* from the source file *IEEEIO.PAS*, type the following compiler statement:

```
TPC IEEEIO.PAS [Enter]
```

This statement will work for both versions of TURBO PASCAL, and in each case will create a file called *IEEEIO.TPU*. The version 4.0 *IEEEIO.TPU* was renamed to *IEEEIOV4.TPU* for the sake of clarity.

To create an executable *MYPROG.EXE*, type:

```
TPC MYPROG.PAS [Enter]
```

Opening the Driver

In TURBO PASCAL, File I/O to and from the KM-488-DD driver is accomplished via the intrinsic PASCAL type TEXT. This type of file handle allows files opened for input and output at the same time. Remember that when you installed the KM-488-DD driver (See Chapter 2.), it created three DOS devices for each installed KM-488-DD board. These devices were IEEE, IEEEIN, IEEEOUT. The IEEEIN and IEEEOUT devices should be used for developing application programs in TURBO PASCAL.

The IEEEIN and IEEEOUT devices are used via the PASCAL ASSIGN command. This command assigns each device to a variable of type TEXT. This is accomplished in the *IEEEIO.TPU* unit in the following manner:

```
VAR IEEEIn, IEEEOut  : TEXT;
(* KM-488-DD Device Handle Variables  *)

(*****BEGIN MAIN *****)

BEGIN
     Assign(IeeeOut,'IeeeOut');
     Rewrite(IeeeOut);

     Assign(IeeeIn, 'IeeeIn');
     Reset(IeeeIn);
END.
```

You must ensure that the driver is installed prior to opening the device IEEEOUT for output or TURBO PASCAL will create a disk file called IEEEOUT in the current working directory.

Programming Notes

1. Several commonly used functions are supplied in the TURBO PASCAL UNIT (.TPU) file *IEEEIO.PAS*, which is compiled to produce *IEEEIO.TPU*. The functions available are:

Routine	Description
IOCTL ( )	Perform the IOCTL "BREAK" command on the KM-488-DD driver. (See section 4.1., BASIC.)
IOCTLRead	Performs a DOS int 21H function 44H subfunction 2

RawMode()	Sets characters transfer to Binary.
ieeeComplete	Closes IEEE Input and Output streams.
MAIN	Opens the IEEEIN and IEEEOUT devices for I/O operations, performs an IOCTL "BREAK" EOL reset on the devices, sets the command/data transfer mode to binary, and performs a warm reset of the driver and interface card.

2. Once you have successfully opened (or ASSIGNed) the IEEEIN and IEEEOUT devices for reading and writing, you can then send commands to the driver and receive data directly from either the KM-488-DD driver (for example, by using the HELLO command) or from the GPIB instruments via the KM-488-DD driver (for example, by using the ENTER command).

   Writing commands to the KM-488-DD driver can be accomplished by using the WRITELN command in the following manner:

```
WRITELN(IEEEOut, 'HELLO');
```

   If a syntax error or GPIB bus error occurs as a result of writing a command to the KM-488-DD device, the error will appear in the global TURBO PASCAL variable IOResult directly after the WriteLn command is issued. It is wise to check the IOResult variable directly after the WriteLn command is issued. Here is an example of writing a command and checking for an error:

```
WRITELN(IEEEOut, 'data data data data');
if (IOResult <> 0) THEN
  WRITELN('ERROR DURING COMMAND : data data data
data');
```

2. Note that many of the arguments to the procedures in *IEEEIO.PAS* are all declared as **VAR XXX** or **VARS XXX** in the case of string variables. This type of declaration instructs the PASCAL compiler to pass these arguments by reference, rather than by value. Passing variables by reference passes the address of a variable, rather than the contents of that variable into a procedure. This is always the convention for calling KM-488-DD functions that will return a value via a call.

When performing BUFFERED I/O as in the ENTER and OUTPUT commands, it is necessary to supply the KM-488-DD driver with a segmented (far) address of the location in which to store or retrieve data to be transferred. TURBO PASCAL provides intrinsic directive for determining the segment and offset of variables. In this example, data is being entered into a character array sdata from a device at GPIB address 05. The intrinsic PASCAL **SEG** and **OFS** directives are used to determine first the segment **(SEG(sdata[0]))** and then the offset **(OFS(sdata[0]))** of the character array **sdata**.

```
IOResult := 0;
WRITELN(IEEEOut, "ENTER 05 #100 BUFFER ',SEG(sda-
ta[0]),':',OFS(sdata[0]));
If (IOResult <> 0) then
 Writeln('Error During Enter From Device 05');
```

3. The default input and output EOL (End of Line) terminator for PASCAL is the Carriage Return/Line Feed combination. The is the default for the KM-488-DD driver, and should not be changed unless you are running other application programs requiring different EOL characters. For example, a program written in C would possibly have the NULL (0) character as a single character EOL sequence. Should you need to reset the EOL terminator to the default, the IOCTL procedure is the correct mechanism for doing so. The IOCTL command is issued as follows:

```
IOCTL(IEEE) ;
```

The maximum allowable string length for a command string written to any of the KM-488-DD devices is 256 characters, including the two character EOL sequence. Any command requiring more than 256 characters should be broken up into several commands, each of which is issued separately. For example a lengthy SEND command can be sent as repeated smaller SEND commands:

```
IEEEWrite(IEEE, 'SEND UNT UNL MTA LISTEN 10 DATA
1, 2, 3, 4') ;
```

is equivalent to :

```
IEEEWrite(IEEE, 'SEND UNT') ;
IEEEWrite(IEEE, 'SEND UNL') ;
IEEEWrite(IEEE, 'SEND MTA') ;
IEEEWrite(IEEE, 'SEND LISTEN 10') ;
IEEEWrite(IEEE, 'SEND DATA 1') ;
IEEEWrite(IEEE, 'SEND DATA 2') ;
IEEEWrite(IEEE, 'SEND DATA 3') ;
IEEEWrite(IEEE, 'SEND DATA 4') ;
```

Consult the provided example programs fro further details about using the various GPIB control commands accessible via the KM-488-DD driver.

4.5 IN MICROSOFT C

Driver Configuration

EOL IN - NULL (0)

EOL OUT - LF

FILL - ERROR

Versions Supported

Microsoft C version 4.0 and later

Preparing the Environment

The *IEEEIO.C* source file contains several routines to facilitate interfacing applications to the driver. It is compiled using the following statement :

```
CL /c IEEEIO.C ;
```

This operation produces the file *IEEEIO.OBJ* which you can then link to your application program to create an executable that takes advantage of the various functions supplied in *IEEEIO.C.* For example :

```
cl /c ieeeio.c ;
   cl /c myprog.c ;
   link myprog + ieeeio ;
```

(Note that your application should contain the line:
`#include "ieeeio.h"` instructing the C compiler to include the file *IEEEIO.H* which contains various definitions pertaining to the functions and variables in the *IEEEIO.C* module)

Opening the Driver

File or device style I/O to and from the KM-488-DD driver is accomplished either via a device handle of type FILE as defined in the *STDIO.H* header file or by using the older UNIX-style integer file handle with the lower level (open(), write(), read(), etc.) file internals in the C library.

Each of these file handles allows files opened for input and output at the same time. For each installed card, the KM-488-DD driver provides three devices whose operation is identical. They are :

`'IEEE', 'IEEEIN' and 'IEEEOUT'`

The KM-488-DD driver provides three devices for languages such as BASIC which cannot open devices for read and write operations via a single handle. Use of only the 'IEEE' device should suffice for all Microsoft C application programs, although the others, notably 'IEEEIN' and 'IEEEOUT' are available should you need to use them.

The 'IEEE' device can be opened two ways. The first is to open the file via a UNIX-style open and assigned the returned handle to an integer variable. An example is given here:

```
int ieee ;
if ((ieee = open ("ieee", O_RDWR | O_BINARY)) == -1)
   printf("\n Error Opening IEEE Device ") ;
```

The second method of opening the file involves defining a variable of type FILE * and assigning it the value returned by the fopen() function as shown here:

```
FILE *ieee ;
 if ((ieee = fopen("$KM488DD", "r+")) == NULL)
 {
  printf("\n Error Opening IEEE Device ") ;
  exit(1) ;
 }
```

In the file *IEEEIO.C*, several functions have been supplied to facilitate the handling of the KM-488-DD's devices, including the function **`ieeei-nit()`** which opens the 'IEEE' device via a UNIX style integer handle.

`ieeeinit()` also puts the 'IEEE' device into raw mode (No Carriage Return/Line Feed Translation is performed as characters are passed through DOS en route to your program from the KM-488-DD driver), issues the IOCTL "BREAK" command to reset the default EOL's to Carriage Return/Line Feed, commands a warm reset of the KM-488-DD interface card and device, and sets the EOL (end of line) terminator sequence to the NULL character (0) for inputs and Line Feed (0x0A) for output.

When utilizing the UNIX style file open, you must ensure that the KM-488-DD driver is installed prior to opening the device 'IEEE' for reading and writing, or Microsoft C's internal file routines will create a disk file called IEEE in the current working directory. An alternative to this is to attempt to open the IEEE device for read-only operations. This type of file open will fail if the target file does not currently exist (i.e. the KM-488-DD driver has not been loaded).

IEEEIO.C Functions

Several functions are supplied in the *IEEEIO.C* module to facilitate interfacing an application program with the driver. These are :

ioctl_rd()	Performs DOS int 21H function 44H subfunction 2
ioctl_wt()	Performs DOS int 21H function 44H subfunction 3
segment()	Returns Segment value of a pointer.
offset()	Returns Offset value of a pointer.
ieeerd()	Reads from specified IEEE device.
ieeewt()	Writes to specified IEEE device.
false()	Does nothing; just returns false (0).
no_op()	Does nothing.
cklpint()	Checks if Light Pen interrupt is pending.
rawmode()	Sets character transfer to Binary.
ieeeinit()	Initializes KM-488.
ieeeprtf()	Special printf().
ieeescnf()	Special scanf().

Once you have successfully opened (or "fopened)" the 'IEEE' device for reading and writing, you can then send commands to the driver and receive data from either the KM-488-DD driver (i.e. the HELLO command) or from GPIB instruments via the KM-488-DD driver (i.e. the ENTER command).

Writing commands to the KM-488-DD driver can be accomplished in several fashions -- many of which are listed below. The intrinsic C functions which use the FILE * style device handle include:

fprintf() ;	Intrinsic C formatted file printf

`fwrite() ;`	Intrinsic C buffered file write
`fputs() ;`	Intrinsic C unformatted file puts
`fputc() ;`	Intrinsic C character file write

The Intrinsic C functions which utilize the UNIX style integer file handle include:

`write() ;`	UNIX-style buffered file write
`dos_write() ;`	UNIX-handle buffered file write

The *IEEEIO.C* customized device write functions include:

`ieeewt()`	'IEEE' specific write
`ieeeprtf()`	'IEEE' specific variable arg write

Likewise, there are numerous ways of reading data from the KM-488-DD driver via the 'IEEE', 'IEEEIN' and 'IEEEOUT' devices. They include the Intrinsic C functions (using the FILE * style device handle):

`fread()`	Intrinsic C buffered file read
`fgets()`	Intrinsic C file gets()
`fgetchar()`	Intrinsic C file getchar()
`fgetc()`	Intrinsic C file getc()
`fscanf()`	Intrinsic C file scanf()

The intrinsic C functions using the UNIX style integer file handle include:

`read()`	UNIX-style direct buffered read
`dos_read()`	UNIX-style direct DOS file read

The *IEEEIO.C* customized device read functions include:

`ieeerd()`	'IEEE' Specific UNIX buffered read
`ieeescnf()`	'IEEE' Specific formatted read of up to five values

Because the number of options available for communicating with the KM-488-DD driver via the 'IEEE', 'IEEEIN' and 'IEEEOUT' devices, it is suggested that you consult the provided example programs for working examples of these types of operations.

Programming Notes

1. When performing command writes via a **`FILE *`** style file handle, (**`fprintf`**, **`fwrite`**, etc.) issue a **`rewind()`** or **`fflush()`** directly following the command output. This insures that your command is written to the 'IEEE' device immediately, rather than buffered as is normally the case with disk file writes.

2. It is conventional to set the input EOL terminator to the NULL (0) character, and the output EOL terminator to Line-Feed (0xA) or Carriage Return/Line Feed (0xa, 0xD). When returning data to an application program, the 'IEEE' (or 'IEEEIN') device will append the default EOL terminator to the input data except in the case of buffered inputs such as **`ENTER 01 BUFFER &H9000:0`**. Use the EOL command to reset the default terminators to your needs. Notice that the **`ieeeinit()`** function in the *IEEEIO.C* module performs this task automatically for you by setting the default input terminator to the NULL (0) character and the default output terminator sequence
to Line-Feed (0xA).

3. Understandably, it is often necessary to provide the KM-488-DD driver with a far address of where to store or retrieve data used by an application program. A far address is always needed in a BUFFER operation, as the KM-488-DD has no knowledge of your application program's data areas. Microsoft C provides the **`FP_SEG`** and **`FP_OFF`** macros for deriving the segment and offset of a pointer, and the *IEEEIO.C* module contains two routines -- **`segment();`** and **`offset();`** which can perform the same task using a pointer (near or far) as their argument.

   Although C supports a **`%p`** format identifier in its printf-style string writes, this option cannot be used because it does not adhere to the more common &Hssss:&Hoooo segment:offset address specification. The following is an example of issuing a BUFFER command containing a far address to a data buffer :

```
if (ieeeprtf("ENTER 01 #1000 BUFFER %d:%d \n",
                 segment(datarray), offset(datarray))
== -1)
      {
        printf("Error on writing to driver.\n") ;
        exit(1) ;
       }
```

4. The maximum allowable string length for a command string written to any of the KM-488-DD devices is 256 characters, including the two character EOL sequence. Any command requiring more than 256 characters should be broken up into several commands, each of which is issued separately. For example, a lengthy 'SEND' command can be sent as repeated smaller 'SEND' commands :

   ```
   ieeewt(IEEE, 'SEND UNT UNL MTA LISTEN 10 DATA 1,
   2, 3, 4') ;
   ```

   is equivalent to :

   ```
   ieeewt("SEND UNT") ;
        ieeewt("SEND UNL") ;
        ieeewt("SEND MTA") ;
        ieeewt("SEND LISTEN 10") ;
        ieeewt("SEND DATA 1") ;
        ieeewt("SEND DATA 2") ;
        ieeewt("SEND DATA 3") ;
        ieeewt("SEND DATA 4") ;
   ```

   Consult the provided example programs for further details about using the various GPIB control commands accessible via the KM-488-DD driver.

4.6 IN TURBO C

Driver Configuration

EOL IN - NULL (0)

EOL OUT - LF

FILL - ERROR

Versions Supported

TURBO C version 4.0 and later

Preparing the Environment

The *IEEEIO.C* source file contains several routines to facilitate interfacing application programs to the driver. It is compiled using the following statement:

```
tcc - c IEEEIO.C
```

and the resulting object file is linked to the object file created by compiling your program.

If your program is to make use of these external procedures, it should **'#include'** the header file *IEEEIO.H* which is supplied with the KM-488-DD driver. This header file is needed so that the Turbo C compiler perform a syntax and argument type-check during the compile process.

Opening the Driver

File or device style I/O to and from the KM-488-DD driver is accomplished either via a device handle of type FILE as defined in the *STDIO.H* header file or by using the older UNIX-style integer file handle with the lower level (**open()**, **write()**, **read()**, etc.) file internals in the C library.

Each of these file handles allows files opened for input and output at the same time. For each installed card, the KM-488-DD driver provides three devices whose operation is identical. They are 'IEEE', 'IEEEIN' and 'IEEEOUT'.

The KM-488-DD driver provides three devices for languages such as BASIC which cannot open devices for read and write operations via a single handle. Use of only the 'IEEE' device should suffice for all Turbo C application programs, although the others, notably 'IEEEIN' and 'IEEE-OUT' are available should you need to use them.

The 'IEEE' device can be opened two ways. The first is to open the file via a UNIX-style open and assigned the returned handle to an integer variable. An example is given here:

```
int ieee ;

if ((ieee = open ("ieee", O_RDWR | O_BINARY)) == -1)
   printf("\n Error Opening IEEE Device ") ;
```

The second method of opening the file involves defining a variable of type **`FILE *`** and assigning it the value returned by the **`fopen()`** function as shown here:

```
FILE *ieee ;
 if ((ieee = fopen("$KM488DD", "r+")) == NULL)
 {
  printf("\n Error Opening IEEE Device ") ;
  exit(1) ;
 }
```

In the file *IEEEIO.C*, several functions have been supplied to facilitate the handling of the KM-488-DD's devices, including the function **`ieeeinit()`** which opens the 'IEEE' device via a UNIX style integer handle. **`ieeeinit()`** also puts the 'IEEE' device into raw mode (No Carriage Return/ Line Feed Translation is performed as characters are passed through DOS en route to your program from the KM-488-DD driver), issues the IOCTL "BREAK" command to reset the default EOL's to Carriage Return/Line Feed, commands a warm reset of the KM-488-DD interface card and device, and sets the EOL (end of line) terminator sequence to the NULL character (0) for inputs and Line Feed (0x0A) for output.

IEEEIO Functions When utilizing the UNIX style file open, you must ensure that the KM-488-DD driver is installed prior to opening the device 'IEEE' for reading and writing, or Turbo C's internal file routines will create a disk file called IEEE in the current working directory. An alternative to this is to attempt to open the IEEE device for read-only operations. This type of file open will fail if the target file does not currently exist (i.e. the KM-488-DD driver has not been loaded). Several functions are supplied in *IEEEIO.C* to facilitate interfacing an application program with the driver. These are:

`ioctl_rd()`	Performs DOS int 21H function 44H subfunction 2
`ioctl_wt()`	Performs DOS int 21H function 44H subfunction 3
`segment()`	Returns Segment value of a pointer.
`offset()`	Returns Offset value of a pointer.
`ieeerd()`	Reads from specified IEEE device.
`ieeewt()`	Writes to specified IEEE device.
`_false_()`	Does nothing; just returns false (0).
`no_op()`	Does nothing.
`cklpint()`	Checks if Light Pen interrupt is pending.
`rawmode()`	Sets character transfer to Binary.
`ieeeinit()`	Initializes KM-488.
`ieeeprtf()`	Special printf().
`ieeescnf()`	Special scanf().

Once you have successfully opened (or "fopened)" the 'IEEE' device for reading and writing, you can then send commands to the driver and receive data from either the KM-488-DD driver (i.e. the HELLO command) or from GPIB instruments via the KM-488-DD driver (i.e. the ENTER command).

Writing commands to the KM-488-DD driver can be accomplished in several fashions -- many of which are listed below. The intrinsic C functions which use the FILE * style device handle include:

`fprintf() ;`	Intrinsic C formatted file printf
`fwrite() ;`	Intrinsic C buffered file write
`fputs() ;`	Intrinsic C unformatted file puts
`fputc() ;`	Intrinsic C character file write

The Intrinsic C functions which utilize the UNIX style integer file handle include:

`write() ;`	UNIX-style buffered file write
`dos_write() ;`	UNIX-handle buffered file write

The *IEEEIO.C* customized device write functions include:

`ieeewt()`	'IEEE' specific write
`ieeeprtf()`	'IEEE' specific variable arg write

Likewise, there are numerous ways of reading data from the KM-488-DD driver via the 'IEEE', 'IEEEIN' and 'IEEEOUT' devices. They include the Intrinsic C functions (using the FILE * style device handle):

`fread()`	Intrinsic C buffered file read
`fgets()`	Intrinsic C file gets()
`fgetchar()`	Intrinsic C file getchar()
`fgetc()`	Intrinsic C file getc()
`fscanf()`	Intrinsic C file scanf()

The intrinsic C functions using the UNIX style integer file handle include:

`read()`	UNIX-style direct buffered read
`dos_read()`	UNIX-style direct DOS file read

The *IEEEIO.C* customized device read functions include:

`ieeerd()`	'IEEE' Specific UNIX buffered read
`ieeescnf()`	'IEEE' Specific formatted read of up to five values

Because the number of options available for communicating with the KM-488-DD driver via the 'IEEE', 'IEEEIN' and 'IEEEOUT' devices, it is suggested that you consult the provided example programs for working examples of these types of operations.

Programming Notes

1. When performing command writes via a **`FILE *`** style file handle, (**`fprintf`**, **`fwrite`**, etc.) issue a **`rewind()`** or **`fflush()`** directly following the command output. This insures that your command is written to the 'IEEE' device immediately, rather than buffered as is normally the case with disk file writes.

2. It is conventional to set the input EOL terminator to the NULL (0) character, and the output EOL terminator to Line-Feed (0xA) or Carriage Return/Line Feed (0xa, 0xD). When returning data to an application program, the 'IEEE' (or 'IEEEIN') device will append the default EOL terminator to the input data except in the case of buffered inputs such as 'ENTER 01 BUFFER &H9000:0'. Use the EOL command to reset the default terminators to your needs. Notice that the ieeeinit() function in the *IEEEIO.C* module performs this task automatically for you by setting the default input terminator to the NULL (0) character and the default output terminator sequence to Line-Feed (0xA).

3. It is often necessary to provide the KM-488-DD driver with a far address of where to store or retrieve data used by an application program. A far address is always needed in a BUFFER operation, as the KM-488-DD has no knowledge of your application program's data areas. Turbo C provides the FP_SEG and FP_OFF macros for deriving the segment and offset of a pointer, and the *IEEEIO.C* module contains two routines -- segment() ; and offset() ; which can perform the same task using a pointer (near or far) as their argument.

   Although C supports a **`%p`** format identifier in its printf-style string writes, this option cannot be used because it does not adhere to the more common &Hssss:&Hoooo segment:offset address specification. The following is an example of issuing a BUFFER command containing a far address to a data buffer :

```
if (ieeeprtf("ENTER 01 #1000 BUFFER %d:%d \n",
   segment(datarray), offset(datarray)) == -1)
 {
   printf("Error on writing to driver.\n") ;
   exit(1) ;
 }
```

4. The maximum allowable string length for a command string written to any of the KM-488-DD devices is 256 characters, including the two character EOL sequence. Any command requiring more than 256 characters should be broken up into several commands, each of which is issued separately. For example, a lengthy SEND command can be sent as repeated smaller SEND commands :

```
ieeewt(IEEE, 'SEND UNT UNL MTA LISTEN 10 DATA 1,
2, 3, 4') ;
```

is equivalent to :

```
ieeewt("SEND UNT") ;
     ieeewt("SEND UNL") ;
     ieeewt("SEND MTA") ;
     ieeewt("SEND LISTEN 10") ;
     ieeewt("SEND DATA 1") ;
     ieeewt("SEND DATA 2") ;
     ieeewt("SEND DATA 3") ;
     ieeewt("SEND DATA 4") ;
```

Consult the provided example programs for further details about using the various GPIB control commands accessible via the KM-488-DD driver.

CHAPTER 5
THE FILE I/O COMMANDS

This chapter provides detailed information about each of the commands provided with the KM-488-DD driver. The File I/O Commands are listed within this chapter in alphabetical order.

5.1 COMMAND DESCRIPTION FORMAT

Each command description is divided into several sections as described below.

Syntax This section describes the syntax required by each command. The commands are implemented as described in Chapter 4. The following rules are used in this section.

Command names and input parameters which are mandatory appear in **`this typeface`**.

Parameters and delimiters which are optional appear in *`this typeface`*.

Square Brackets ([ ]) enclose a list of optional parameters. You may select any combination of these options as described in the command-specific parameters section.

Parameters This section describes the input parameters. In some instances, multiple parameters may be specified (for example, multiple device addresses).

Returns This part of the command description describes any data which will be returned after execution of the command has been completed. Your application program must read these responses. Errors will occur if it is not.

Programming Notes This section describes any special programming considerations for the File I/O Command.

Example This section provides an example of valid File I/O Command Syntax. This is independent of the programming language. Refer to Chapter 4 for more information regarding how to implement the File I/O Command in each language.

5.2 THE FILE I/O COMMANDS

Table 5-1 lists the available File I/O Commands. It provides a brief description of the command and the corresponding action on the GPIB interface. In the pages following Table 5-1, more detailed descriptions of each command are provided.

Table 5-1. List of File I/O Commands

COMMAND NAME	DESCRIPTION	KM-488-DD MODE	BUS ACTION
ABORT	Halts bus activity.	Sys Cont	Assert IFC, then Unassert IFC
		Active Controller	Assert ATN line, then MTA.
ARM	Enables a lightpen interrupt condition(s).	Any	—
BUFFER	Defines direction of access within memory buffers.	—	–
BUFFERED	Reads the number of bytes last transferred by buffered ENTER or OUTPUT command.	Any	–
CLEAR	Clears designated device(s).	Active Controller	For All Devices Assert ATN, then DCL. For Selected Device(s) Assert ATN, then UNL, MTA, LAG, and SDC.
CODE ADDRESS	Returns a memory segment address.	Any	—
CONFIG	Alters driver parameters.	Any	–
DISARM	Disables a lightpen interrupt condition(s).	Any	—
DMA	Define DMA transfer mode.	Any	—
ENTER	Read data from a bus device.	Active Controller	With Address Assert ATN, UNL, MLA, TAG, Unassert ATN, read data, and Assert ATN. Without Address Unassert ATN, read data, and Assert ATN.
		Peripheral	Read data when addressed to listen and ATN is unasserted.
ENTER BUFFER	Reads data from a bus device into a buffer.	Active Controller	With Address Assert ATN, UNL, MLA, TAG, Unassert ATN, read data, and Assert ATN. Without Address Unassert ATN, read data, and Assert ATN.
		Peripheral	Determined by the Active Controller.
EOL	Sets the EOL terminators.	Any	–

Table 5-1. List of File I/O Commands

COMMAND NAME	DESCRIPTION	KM-488-DD MODE	BUS ACTION
ERROR	Enable/Disable error message display.	Any	--
FILL	Defines response if no data is available.	Any	--
HELLO	Reads KM-488-DD identification string.	Any	--
LOCAL	Allows designated device(s) to be locally programmed.	System Controller	Unasserts REM
		Active Controller	Asserts ATN then UNL, MTA, LAG, and GTL.
LOCAL LOCKOUT	Disables the local button on a GPIB device.	Active Controller	Asserts ATN then LLO.
OUTPUT	Transmits data to the designated device(s).	Active Controller	<u>With Address</u> (If System Controller assert REN). Then assert ATN, MTA, UNL, LAG, Unassert ATN, and write data.
			<u>Without Address</u> (If System Controller assert REN). Unassert ATN, write data.
		Peripheral	If System Controller, assert REN. Then write data when addressed to talk and ATN is unasserted
			Write data when addressed to talk and ATN is unasserted.
OUTPUT BUFFER	Transmits contents of a memory buffer to the designate device(s).	Active Controller	<u>With Address</u> (If System Controller assert REN). Then assert ATN, MTA, UNL, LAG, Unassert ATN, and write data.
			<u>Without Address</u> (If System Controller assert REN). Unassert ATN, write data.
		Peripheral	If System Controller, assert REN. Then as determined by the Active Controller.
			Otherwise, as determined by the Active Controller.
PASS CONTROL	Assign another device as the Active Controller.	Active Controller	Assert ATN, UNL, MLA, TAG, UNL, TCT, then Unassert ATN.
PPOLL	Read the Parallel Poll response from all bus devices.	Active Controller	Assert ATN, then EOI, receive Parallel Poll Response, then Unassert EOI.

Table 5-1. List of File I/O Commands

COMMAND NAME	DESCRIPTION	KM-488-DD MODE	BUS ACTION
PPOLL CONFIG (PPC)	Structure the Parallel Poll response string.	Active Controller	Assert ATN, UNL, MTA, LAG, PPC, PPE.
PPOLL DISABLE (PPD)	Disables Parallel Polling for designated device(s).	Active Controller	Assert ATN, UNL, MTA, LAG, PPC,PPD.
PPOLL UNCONFIG (PPU)	Disables Parallel Polling for <u>all</u> devices.	Active Controller	Assert ATN, then PPU.
REMOTE	Puts device(s) into remote state and optionally addresses them to listen.	System Controller	Assert REN
		System Controller and Active Controller	Assert REN, then ATN and UNL, MTA, with address(es) LAG.
REQUEST	Defines own Serial Poll Response.	Peripheral	<u>If Request for Service bit set</u> Assert SRQ. <u>If Request for Service bit not set</u> Unassert SRQ.
RESET	Resets KM-488-DD to its default parameters.	Any	--
RESUME	Unasserts ATN. Allows Peripheral to Peripheral transfers.	Active Controller	Unassert ATN.
SEND	Sends low-level bus commands and data.	Active Controller *This mode can use any subcommand.*	User defined
		Peripheral *This mode can use only the DATA and EOI subcommands.*	User defined
SPOLL	Reads response(s) to serial poll.	Any	<u>With Adr(Active Controller)</u> Assert ATN, UNL MLA, TAG, SPE, Unassert ATN, receive data. Assert ATN, then SPD, UNT. <u>Without Adr(Active Controller) or Peripheral</u> None.
STATUS	Returns KM-488-DD status string.	Any	--
STOP	Stops DMA CONTINUE transfers.	Any	<u>If Active Controller</u> Asserts ATN. <u>If Peripheral</u> None

Table 5-1. List of File I/O Commands

COMMAND NAME	DESCRIPTION	KM-488-DD MODE	BUS ACTION
TERM	Sets the GPIB bus terminators.	Any	--
TIME OUT	Sets timeout elapse.	Any	--
TRIGGER	Triggers device(s).	Active Controller	With Address Assert ATN, UNL, MTA, LAG, and GET. Without Address Assert ATN, GET.
WAIT	Waits for the current DMA continue transfer to complete.		

ABORT

Purpose The ABORT command allows the KM-488-DD to take control of the GPIB bus by forcing all other devices on the bus to the idle state. The KM-488-DD must be the Active Controller.

Syntax

```
ABORT
```

Parameters None.

Returns None.

Programming Notes If the KM-488-DD is both the Active and System Controller, ABORT asserts the IFC line to retain control of the bus. Otherwise, the KM-488-DD will assert the ATN line, and then untalk and unlisten all devices.

Example This forces the KM-488-DD to regain control of the bus and put all GPIB bus devices into an idle state.

```
ABORT
```

ARM

Purpose Defines the conditions under which the lightpen status becomes true.

Syntax `ARM` *`interrupt, interrupt...`*

Parameters **`interrupt`** specifies the conditions under which an interrupt may be generated. Multiple conditions may be selected. **`interrupt`** can be any combination of the following:

SRQ	The lightpen status will become true when the KM-488-DD's internal Service Request Bit (SRQ) is set.
PERIPHERAL *	The lightpen status will become true when control is passed to the KM-488-DD and it becomes the Active Controller.
CONTROLLER *	The lightpen status will become true when the KM-488-DD which has been acting as a Peripheral becomes the Active Controller.
TRIGGER	The lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.
CLEAR	The lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
TALK*	The lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Talker and can output data to the GPIB.
LISTEN*	The lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Listener and can accept data from the GPIB.
IDLE*	The lightpen status will become true when the KM-488-DD, acting as a Peripheral, is unaddressed as a Talker or a Listener.

ARM

CHANGE	The lightpen status will become true when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller). This encompasses all of the above conditions which are marked with an asterisk (*).

☞ ***NOTE:*** The conditions marked by an asterisk (*) will generate an interrupt only when the KM-488-DD's address status has changed. This is indicated by the state of the Address Change Bit in the Status byte. See the STATUS command description for more information.

Returns None

Programming Notes Lightpen emulation must be enabled at driver installation via the *CONFIG.DAT* file. (Default is no lightpen emulation.)

Example This activates a lightpen interrupt when the KM-488-DD as a peripheral receives a trigger or a clear command.

```
ARM CLEAR, TRIGGER
```

BUFFER

Purpose Selects the direction in which memory is addressed for during buffered transfers.

Syntax **BUFFER** *direction*

Parameters **direction** indicates which direction the memory is to be addressed. Select either **INCREMENT** or **DECREMENT** where:

INCREMENT increases the buffer address after each byte is transferred.

DECREMENT decreases the buffer address after each byte is transferred.

Returns None.

Programming Notes When using **INCREMENT** during a bus I/O operation, specify the lowest address of the designated buffer in the command. Likewise, when using **DECREMENT**, select the highest address.

Example This instructs the KM-488-DD to increment the buffer address during a buffered transfer.

```
BUFFER INCREMENT
```

BUFFERED

Purpose Returns the number of characters transferred in the last BUFFERED transfer or, in the case of DMA transfers, the current transfer.

Syntax **`BUFFERED`**

Parameters None.

Returns An integer from 0 to 1,048,575 (2^{20}-1).

Programming Notes

1. If the data was transferred using DMA with the CONTINUE option, the integer returned represents the number of characters transferred thus far into the operation.
2. It is useful to call this command after an **`ENTER #count BUFFER term`**. This will help you determine if the expected number of characters has been received and/or the transfer has terminated as the result of a detected terminator.
3. In BASIC, do not input the returned value into an integer variable, as **`counts`** greater than 32K will cause an overflow.

Example This will return the number of characters transferred during the last buffer transfer.

```
BUFFERED
```

CLEAR

Purpose Resets device(s) to their power-up parameters. The KM-488-DD must be an Active Controller.

Syntax **CLEAR** *addr*

Parameters **addr** is the GPIB BUS address of the device to be reset. Multiple addresses may be selected. **addr** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

Returns None

Programming Notes

1. When no address is specified, the universal GPIB DCL (Device Clear) command will be issued and all bus devices will be cleared.

2. When an address(es) is specified, the corresponding device(s) is listened and sent a GPIB SDC (Select Device Clear) command.

Example This will issue a device clear to device 15.

```
CLEAR 15
```

CONFIG

Purpose Resets the KM-488-DD to its default operating parameters or to those specified.

Syntax `CONFIG` *`options`*

Parameters **`options`** is a user-defined string. This string can be comprised of any of the following options:

`/Bnn`	**`/Bnn`** sets the primary GPIB bus address of the KM-488-DD. **`nn`** is the primary GPIB address and can be any value between 00 and 30. A secondary address is not allowed.
`/T` *`[term term EOI]`*	Sets the input and output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **`/T`**, the GPIB bus terminators will be reset to no terminators.

Where:

`term` is an optional parameter which represents a GPIB bus terminator byte to be used. This terminator byte can be represented as any of the following:

`$char`	where **`char`** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **`char`** must be preceded by a dollar sign ($). Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.
`CR`	This represents the *Carriage Return* character (13 decimal, 0D hex).

CONFIG (cont'd)

LF	This represents the *Line Feed* character (10 decimal, 0A hex).
'X	where **X** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **'B** represents the character *B*.

EOI (End or Identify) is an optional GPIB BUS terminator. If included, the KM-488-DD will detect/assert the GPIB bus EOI line. (This would indicate that the last character has been sent.) Data will continue to be read until this terminator, a valid terminator sequence, or both are detected. Use of the EOI terminator is further described in Chapter 4.

/TI *[term term EOI]*	Sets the input GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TI**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/TO *[term term EOI]*	Sets the output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TO**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/E *[term term]*	Sets the input and output EOL terminators. **term** is described above.
/EI *[term term]*	Sets the input EOL terminators.**term** is described above.
/EO *[term term]*	Sets the output EOL terminators. **term** is described above.

CONFIG (cont'd)

/N name addr *[sec_addr]*	Configures a named device. Sets its address to the given value and its GPIB bus terminators to the current GPIB bus terminator settings. **name** is the name of the device. **addr** is the primary GPIB address (00 to 30). **sec_addr** is the secondary GPIB address (00 to 31).

If the **options** string is not given, then the configuration will revert to the installation configuration.

Returns None

Programming Notes String length cannot exceed 255 characters.

Example This would reset the KM-488-DD to its initial configuration.

```
CONFIG
```

DISARM

Purpose Resets previously set conditions which cause lightpen interrupts.

Syntax `DISARM` *`interrupt, interrupt...`*

Parameters **`interrupt`** specifies which conditions should be reset. Multiple conditions may be selected. **`interrupt`** can be any of the following:

`SRQ`	The lightpen status will not change when the KM-488-DD's internal Service Request Bit (SRQ) is set. See the SPOLL command for more information.
`PERIPHERAL`	The lightpen status will not change when the KM-488-DD is acting as a peripheral.
`CONTROLLER`	The lightpen status will not change when the KM-488-DD is acting as an Active Controller.
`TRIGGER`	The lightpen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.
`CLEAR`	The lightpen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
`TALK`	The lightpen status will not change when the KM-488-DD is addressed as a Talker and can output data to the GPIB.
`LISTEN`	The lightpen status will not change when the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
`IDLE`	The lightpen status will not change when the KM-488-DD is not addressed as a Talker or a Listener.
`CHANGE`	The lightpen status will not change when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).

DISARM (cont'd)

Returns None.

Programming Notes If **`interrupt`** is not provided, then all conditions which will cause a light-pen interrupt to occur will be disabled.

Example This will disable the KM-488-DD from causing a lightpen interrupt condition when the KM-488-DD becomes addressed as a talker, or when an SRQ occurs.

```
DISARM TALK,SRQ
```

DMA

Purpose Sets the DMA transfer mode.

Syntax **DMA** *mode autoinitialize*

Parameters **mode** is an optional parameter which determines the DMA mode to be used in DMA transfers. This mode affects the way in which control is returned to the PC bus during DMA transfers. **mode** choices include **SINGLE** and **DEMAND.** If no **mode** parameter is given, **SINGLE** mode will be used. **SINGLE** and **DEMAND** are defined as follows:

SINGLE - In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus and transfers a single byte of data. Control of the bus is then returned to the microprocessor.

DEMAND - In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus. The DMA controller retains control of the bus until the DMA request signal is unasserted. Once this signal has been unasserted for more than one microprocessor clock cycle, control of bus is returned to the microprocessor. This mode allows the DMA controller chip to pass data at a slightly faster rate and the microprocessor to access the bus when it is not needed.

AUTOINITIALIZE is an optional parameter which enables the DMA AUTOINITIALIZE mode. Under normal circumstances, the DMA controller transfers the specified number of bytes to/from the PC memory from the given starting address and terminates when completed. When the AUTOINITIALIZE mode is enabled, the DMA controller will reset the original byte count, reset the initial address, and repeat the transfer again. The AUTOINITIALIZE option is only in effect until the ENTER BUFFER DMA CONTINUE or OUTPUT BUFFER DMA CONTINUE routine has been completed. If the ENTER BUFFER DMA CONTINUE option is specified (See the ENTER BUFFER command), the AUTOINITIALIZE option will only be in effect for that transfer.

Returns None.

DMA (cont'd)

Programming Notes

1. All DMA AUTOINITIALIZE transfers must occur entirely within a single DMA 64 KByte page. If a DMA buffer operation involves a transfer of more than 64K or occurs across a DMA page boundary, the AUTOINITIALIZE option affects only those bytes in the last transfer occurring within one DMA page.

   Also, note that DMA AUTOINITIALIZE transfers specified for buffers that are located in more than a single DMA page are unpredictable.

2. It is impossible for a program to halt a DMA AUTOINITIALIZE operation unless the DMA CONTINUE option is selected. (See ENTER BUFFER, OUTPUT BUFFER.) This is because the driver does not return control to a program using non-CONTINUE operations until the transfer completes.

Example

This will program DMA transfers to occur in DEMAND, AUTOINITIALIZE mode.

```
DMA DEMAND AUTOINITIALIZE
```

ENTER

Purpose Allows the KM-488-DD to receive data from a GPIB bus device.

Syntax **ENTER** *addr; [#count term1 term2 EOI]*

Parameters **addr** is an optional parameter which specifies the GPIB BUS address of the device which is to send the data. If this parameter is included, the KM-488-DD is also addressed to listen. If this parameter is not included, some Active Controller must have already addressed a device to talk and the KM-488-DD to listen. **addr** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

; is a delimiter. This delimiter must be used if only **addr** and a **term** are specified. Otherwise, it is optional.

[#count term1 term2 EOI] are the conditions which can be used to terminate ENTER. You may specify a count OR a one or two character GPIB bus terminator, with or without EOI OR just EOI alone. The parameters are:

#count is an optional parameter. It is a long integer representing the number of characters to be read. It must always be preceded by a pound sign (#). Valid counts are within the range 1 to 4294967295 (2^{32}-1) decimal, or from 1 to FFFFFFFF hex. All hex values must be preceded with an &H, i.e. #&H300 would specify 768 bytes.

term1 is an optional parameter which represents the first GPIB bus input terminator byte to be used. This terminator byte can be represented as any of the following:

$char	where **char** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **char** must be preceded by a dollar sign ($). Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.
CR	This represents the *Carriage Return* character (13 decimal, 0D hex).
LF	This represents the *Line Feed* character (10 decimal, 0A hex).

ENTER (cont'd)

'X	where **X** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **' B** represents the character *B*.

term2 is an optional parameter which represents the second GPIB bus input terminator byte (in a two-byte terminator) to be received. This parameter is represented in the same manner as the **term1** parameter.

EOI (End or Identify) is an optional GPIB BUS terminator. If included, the KM-488-DD will check for the assertion of the GPIB bus EOI line. (This would indicate that the last character has been sent.) Data will continue to be read until this terminator, a valid terminator sequence, or both are detected. Use of the EOI terminator is further described in Chapter 4.

Returns The returned data is device-dependent.

Programming Notes

1. If the KM-488-DD is a Peripheral, the **addr** parameter must not be included and an Active Controller must address a device to talk and the KM-488-DD to listen before the ENTER command is called, unless that bus has already been addressed.

2. If the **term1**, **term2**, or **EOI** parameters are included, these settings will override the default terminator and EOI selections made while configuring your system. (See Chapter 2.) If not, the defaults are used.

3. If both **term** characters and **EOI** are specified then ENTER will terminate on the condition which occurs first, i.e., the terminators or EOI.

4. If a **count** is given, then only that number of characters will be returned. No EOL terminator will be appended. If no **count** is given, data will not be accepted after the KM-488-DD detects the GPIB BUS input terminator(s) or EOI. In this case, an EOL sequence will be appended to the accepted data.

5. If **count** is given, the sending device must have at least that number of bytes to send. Otherwise, a time out will occur (if time out is enabled) or the process will "hang".

Example This will read data from device 25 until a CR LF pair or EOI is detected.

```
ENTER; 25 CR LF EOI
```

ENTER BUFFER

Purpose Reads data from a bus device into an area of memory.

Syntax `ENTER addr ; #count BUFFER buf_addr DMA CONTINUE term EOI`

Parameters **addr** is an optional parameter which specifies the GPIB BUS address of the device which is to send the data. If this parameter is included, the KM-488-DD is also addressed to listen. If this parameter is not included, the Active Controller must have already addressed a device to talk and the KM-488-DD to listen. **addr** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

; is a delimiter. This delimiter must be used if only **addr** and a **term** are specified. Otherwise, it is optional.

#count is an integer representing the number of characters to be read. It must always be preceded by a pound sign (#). Valid counts are within the range 1 to 4294967295 (2^{32}-1) decimal, or from &H1 to &HFFFFFFFF hex. All hex values must be preceded with an &H, i.e. #&H300 represents 768 bytes.

buf-addr represents the memory buffer address. **buf-addr** may be given either as **segment:offset** (the colon is required), or as an **absolute** memory address, where:

> **segment:offset** - The segment and offset values must be separated by a colon. The **segment** portion of the address represents a particular 64-KByte area of PC memory. This must in the range 0 to 65535 (&H0 to &HFFFF hex). If this value is greater than 32767 and is printed out, in languages without an unsigned integer type (such as BASICA), it will be printed as a negative number. Nevertheless, the KM-488-DD will properly interpret this value.
>
> **offset** represents the actual address of memory location with regard to the **segment** value. That is, the number of bytes from the **segment:0th** location. The **offset** value must be an integer in the range 0 to 65535 (&H0 to &HFFFF hex). As with **segment**, negative **offset** value will be interpreted as an unsigned 16-bit integer.
>
> **absolute** - This is a value which identifies a "real" address. It is a 20-bit long integer in the range 0 to 1048575 (&H0 to &HFFFFF). Consult your language reference manual for manipulating address pointer values in this fashion.

ENTER BUFFER (cont'd)

Addresses of the format SEGMENT:OFFSET are converted to ABSOLUTE format by multiplying the **segment** value by 16 and adding the **offset** value.

DMA is an optional parameter which is used to enable DMA. Using DMA will transfer data at a higher speed.

CONTINUE is an optional parameter which is used in conjunction with the DMA parameter. When this parameter is specified, the KM-488-DD returns control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a STOP command is invoked or an GPIB BUS input terminator is detected.

term is an optional parameter which represents the GPIB bus input terminator byte to be used. This terminator byte can be represented as any of the following:

$char	where **char** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **char** must be preceded by a dollar sign ($). Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.
CR	This represents the *Carriage Return* character (13 decimal, 0D hex).
LF	This represents the *Line Feed* character (10 decimal, 0A hex).
'X	where **X** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **' B** represents the character *B*.

ENTER BUFFER (cont'd)

EOI (End or Identify) is an optional GPIB BUS terminator. If included, the KM-488-DD will check for the assertion of the GPIB bus EOI line. (This would indicate that the last character has been sent.) Data will continue to be read until this terminator is detected. Use of the EOI terminator is further described in Chapter 4.

Returns Data from the GPIB bus is placed directly in the program memory location.

Programming Notes

1. If the KM-488-DD is a Peripheral, the **addr** parameter must not be included and an Active Controller must address a device to talk and the KM-488-DD to listen before the ENTER command is called, unless that bus has already been addressed.

2. During a DMA CONTINUE transmission, the KM-488-DD will only check for a timeout after the first byte is transferred. It is important that your application program check for completion of DMA CONTINUE transfers, especially before accessing the memory where DMA is taking place.

   The purpose of DMA CONTINUE is to allow the application program to continue while the GPIB transfer takes place in the background through DMA. At some point, when the ENTER data to be used, the application program should issue a WAIT command. If the DMA transfer is still in progress, the program should wait for completion; otherwise, it can proceed immediately. The WAIT command also starts the timeout check (if enabled), so that a timeout error will be found if the transfer has "hung". Be sure to specify a timeout value that allows for the entire DMA CONTINUE operation to complete.

3. If the designated memory buffer region crosses a 64 kbyte page boundary, DMA CONTINUE will not return control to the application program until a significant portion of the transfer is complete. This is because the PC's DMA controller is unable to continue a background transfer that crosses a 64 KByte DMA page boundary. Thus, the transfer must be divided into several intra-page transfers. When the last transfer is begun, the KM-488-DD will return control to the application program. The DMA AUTOINITIALIZE (See DMA command description.) option cannot be used if the buffer crosses a 64K page boundary, without unpredictable results.

4. If **term** or **EOI** is not specified, then the SEND device must send the number of bytes specified by **count**. Otherwise, a timeout will occur (if enabled) or the process will "hang".

ENTER BUFFER (cont'd)

Example This will read 95 characters from device 19 into memory at &H40000.

```
ENTER 19#95 BUFFER &H4000:0
```

EOL

Purpose This modifies the End-Of-Line (EOL) terminators selected at initial start-up or selected by the CONFIG command. The EOL terminators are used in transfers between the driver and the application program.

Syntax `EOL` *`dir term`*

Parameters **`dir`** is an optional parameter which represents the EOL terminator to be programmed, where:

`IN` = the INPUT EOL terminator.
`OUT` = the OUTPUT EOL terminator

If this parameter is omitted, both the input and output EOL terminators will be set as specified. See Chapter 3 for more information regarding the use of EOL terminators.

`term` is an optional parameter which specifies either no terminators are to be used or one or two ASCII character(s) to be used as the EOL. This terminator character can be represented in any of the following manners:

`$char`	where **`char`** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.
`CR`	This represents the *Carriage Return* character (13 decimal, 0D hex).
`LF`	This represents the *Line Feed* character (10 decimal, 0A hex).
`'X`	where **`X`** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **`'B`** represents the character *B*.
`NONE`	This specifies that no EOL terminator is to be used. This is not recommended for operations that transfer binary data equal to EOL's, buffered transfers should be used.

Returns None.

EOL (cont'd)

Programming Notes

1. If you elect to not use EOL Terminators (e.g., EOL NONE), be sure to consult Chapter 4.

2. When the application program sends a string to the driver, it will normally append the EOL output terminator to the string. When the driver returns a string, it will append the EOL input terminator. It the programmer's responsibility to know what EOL's are appended by a programming language.

3. EOL terminators are not used in BUFFERED transfers.

Example

This will set the input and output EOL terminators to Line Feed.

```
EOL LF
```

This will specify that no output EOL terminator is to be used.

```
EOL OUT NONE
```

This will set that input EOL terminator to Carriage Return and Line Feed.

```
EOL IN CR LF
```

ERROR

Purpose Enables/disables display of error messages.

Syntax **ERROR** *x*

Parameters **x** is an optional parameter which determines whether the error message display function will be enabled or disabled. If **ON** is specified, the error message display is enabled. If **OFF** is specified, the error message display is disabled.

Returns None.

Example This will disable display of error messages.

```
ERROR OFF
```

FILL

Purpose	Defines the KM-488-DD driver's response to a request for data when none is available.
Syntax	**FILL** *x*

Parameters **x** represents the fill condition. This can be any one of the following:

OFF. This type of response will not return any data characters or a DOS error message.

ERROR. This type of a response will not return any data characters, but will generate an Error Message. The error message number can then be trapped by calling the STATUS command. (Refer to the STATUS command for more information.)

ASCII indicates that a specific ASCII character be returned. This character is designated according to the following conventions:

$char	where **char** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **char** must be preceded by a dollar sign ($). Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.
CR	This represents the *Carriage Return* character (13 decimal, 0D hex).
LF	This represents the *Line Feed* character (10 decimal, 0A hex).
'X	where **X** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **' B** represents the character *B*.

If **x** is not specified, a NULL character will be returned when there is no data available.

Returns None.

FILL

Example This will generate an error message and will not return any characters, if no data is available when requested.

```
FILL ERROR
```

HELLO

Purpose Returns an identification string from the KM-488-DD driver.

Syntax `HELLO`

Parameters None.

Returns A string similar to the following:

```
copyright (c) 1991 Keithley MetraByte Corp
```

Programming Notes None.

Example This example would return the KM-488-DD's identification string.

```
HELLO
```

LOCAL

Purpose Forces the specified bus device(s) to return to being programmed locally from their front panels. <u>This command can only be used if the KM-488-DD is acting as a System or Active Controller.</u>

Syntax **LOCAL** *addr, addr*

Parameters **addr** specifies the GPIB BUS device address(es) of the device(s) to be sent to LOCAL. If this parameter is not included and the KM-488-DD is the System Controller, all devices on the GPIB bus will be sent to LOCAL. If the KM-488-DD is an Active Controller and this parameter is not included, only those devices which have been previously addressed to listen will be sent to LOCAL. Up to 15 device addresses can be given. **addr** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

Returns None

Programming Notes

1. If the KM-488-DD is the System Controller and no address is given, then the GPIB REN (Remote Enable) line is unasserted and all devices are returned to Local. In order to return them to remote mode, it will be necessary to issue a KLOL call, if Local Lockout is required.

2. As an Active Controller, the KM-488-DD can issue the GPIB GTL (Go To Local) message to those devices specified. In this case, the GPIB REN (Remote Enable) line remains asserted and devices will return to remote when addressed to listen. If a KLOL (Local Lockout) call has been issued previously, it should still be in effect when a device is returned to Remote.

Example This would send device 25 to local.

```
LOCAL 25
```

LOCAL LOCKOUT

Purpose This command can only be used if the KM-488-DD is an Active or System Controller. It will disable the GPIB bus devices from being returned to local control by means of the Local Remote button on the device.

☞ ***NOTE:*** This command issues an IEEE-488 bus signal, LOL. This signal is not supported by all IEEE-488 bus devices.

Syntax `LOCAL LOCKOUT`

Alternate Syntax `LOL`

Parameters None.

Returns None.

Programming Notes

1. The "LOCAL" button is disabled so that a device cannot be returned to local state for manual programming or control. The Active Controller can return specific devices to "local with lockout state", whereby an operator can then use the front panel controls. When the device is addressed to listen again, it returns to "remote with lockout state". Thus, the effect of the LOL call will remain until the REN line is unasserted (LOCAL) by the System Controller.

2. It is good practice to issue a LOL so that devices are under strict program control. LOL can be issued before a device is put in remote and will take effect when the device's LOCAL button is set to remote.

Example This would send the Local Lockout command to all listening devices.

```
LOL
```

OUTPUT

Purpose Transmits data from the KM-488-DD to the GPIB bus.

Syntax `OUTPUT` *`addr #count;`*`data`

Parameters **`addr`** specifies the GPIB BUS device address(es) of the device(s) to which the data is to be sent. Multiple device addresses can be given, separated by commas. If this parameter is not included, the Active Controller must have already addressed the KM-488-DD to talk and the GPIB Bus device(s) to listen before the OUTPUT is issued. **`addr`** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

; is a delimiter.

`#count` is an optional parameter. It is an integer representing the number of characters to output. It must always be preceded by a pound sign (#). Valid counts are within the range 1 to 4294967295 (2^{32}-1) decimal, or from &H1 to &HFFFFFFFF hex. All hex values must be preceded with an &H, i.e. #&H300 would specify 768 bytes.

`data` is a string of characters to be sent to the GPIB bus. This string will be terminated by the GPIB default output terminator, unless **`count`** is specified.

Returns None.

Programming Notes

1. The total number of character which can be contained in the **`data`** string is unlimited. The length of the string is controlled only by **`count#`** or the EOL character in the data.

2. If the KM-488-DD is the System Controller, it will automatically assert REN.

3. **`addr`** can only be specified if the KM-488-DD is the Active Controller.

4. If no character count is specified, the EOL output terminator will signal the end of data. This EOL terminator will be replaced with the GPIB Bus Output Terminator before the data is sent to the GPIB bus device. If, however, a character count is specified, exactly that number of characters will be accepted (regardless of whether it includes the EOL output terminator) and sent to the GPIB bus device.

OUTPUT

Example This example will send the string CAT to devices 9 and 25.

```
OUTPUT 09,25;CAT
```

OUTPUT BUFFER

Purpose Transmits data from an area of PC memory to the GPIB bus.

Syntax `OUTPUT` *`addr`* **`#count`** **`BUFFER`** **`buf-addr`** *`DMA CONTINUE EOI`*

Parameters **`addr`** specifies the GPIB BUS device address(es) of the device(s) to which the data is to be sent. Multiple device addresses can be given. If this parameter is not included, the Active Controller must have already addressed the KM-488-DD to talk and the GPIB Bus device(s) to listen before the OUTPUT is issued. **`addr`** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

`#count` is an optional parameter. It is an integer representing the number of characters to transmit. It must always be preceded by a pound sign (#). Valid counts are within the range 1 to 4294967295 (2^{32}-1) decimal, or from &H1 to &HFFFFFFFF hex. All hex values must be preceded with an &H, i.e. #&H300 would specify 768 bytes.

`buf-addr` is the memory buffer address. This address can be represented in **segment:offset** or **absolute** format.

segment:offset - The segment and offset values <u>must</u> be separated by a colon. The **segment** portion of the address represents a particular 64-KByte area of PC memory. This must in the range 0 to 65535 (&H0 to &HFFFF hex). In languages without an unsigned integer type (such as BASICA), if this value is greater than 32767, it will be printed as a negative number. The KM-488-DD will automatically interpret this negative number as its corresponding positive value.

offset represents the actual address of memory location with regard to the **segment** value. That is, the number of bytes from the **segment:0**Th location. The **offset** value must be an integer in the range 0 to 65535 (&H0 to &HFFFF hex). As with **segment**, a negative **offset** value will be interpreted as an unsigned 16-bit integer.

`absolute` - This is a value which identifies a "real" address. It is a 20-bit long integer in the range 0 to 1048575 (&H0 to &HFFFFF). Consult your language reference manual for manipulating address pointer values in this fashion.

Addresses of the format **segment:offset** are converted to **absolute** format by multiplying the **segment** value by 16 and adding the **offset** value.

OUTPUT BUFFER (cont'd)

EOI is an optional parameter which specifies that the EOI signal is to be asserted when the last data byte is transferred.

DMA is an optional parameter, which enables DMA. Using DMA will transfer data at a higher speed.

CONTINUE is an optional parameter which is used in conjunction with the DMA parameter. When this parameter is specified, the KM-488-DD returns control to the application program immediately without waiting for completion of the transfer. The KM-488-DD will wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

If the DMA AUTOINITIALIZE (See the DMA command description.) option is used, the DMA transfer will continue until a STOP command is issued.

Returns None

Programming Notes

1. All DMA AUTOINITIALIZE transfers must occur entirely within a single DMA 64 KByte page. See the notes regarding DMA pages provided in Chapter 3.
2. If the KM-488-DD is the System Controller, it will automatically assert REN.
3. **addr** can only be specified if the KM-488-DD is the Active Controller.

Example This syntax would send 512 characters (&H200) from memory location &H10000 to device 21 using DMA.

```
OUTPUT 21 #&H200 BUFFER &H1000:0 DMA
```

PASS CONTROL

Purpose Designates another controller to be the Active Controller. The KM-488-DD must be the Active Controller.

Syntax `PASS CONTROL` *addr*

Parameters **addr** specifies the GPIB BUS device address. **addr** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

Returns None

Programming Notes If the KM-488-DD which is relinquishing its position as the Active Controller is also a System Controller, it retains its status as System Controller.

Example This example will pass control to device 19.

```
PASS CONTROL 19
```

PPOLL

Purpose	Initiates a Parallel Poll. The KM-488-DD must be an Active Controller.
☞	***NOTE:*** Many GPIB devices do not support parallel polling. Check your device's documentation.
Syntax	`PPOLL`
Parameters	None.
Returns	An integer in the range 0 to 255 decimal that indicates the Data Lines which have been asserted (DIO1-DIO8).
Programming Notes	None.
Example	This would initiate a parallel poll.

```
PPOLL
```

PPOLL CONFIG

Purpose Configures the Parallel Poll response of a GPIB bus device. The KM-488-DD must be an Active Controller.

☞ *NOTE:* Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax `PPOLL CONFIG addr;response`

Alternate Syntax `PPC addr;response`

Parameters **`addr`** specifies the GPIB BUS device address of the device whose parallel poll response byte is to be configured. **`addr`** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

; is an optional delimeter.

`response` is an integer which represents the desired Parallel Poll Response Byte. This decimal integer is of the following format:

Parallel Poll Response Byte

Bit	7	6	5	4	3	2	1	0
	0	1	1	0	S	P2	P1	P0

Where:

S is the parallel poll response value (0 or 1) that the device uses to respond to the parallel poll when service is required. This bit is generally set to 1.

☞ *NOTE:* This value must correspond to the setting of the GPIB bus device's *ist* bit. Refer to the Device's documentation for more information.

P2 P1 P0 is a 3-bit value which tells the device being configured which data bit (DIO1 through DIO8) it should use as its parallel poll response.

PPOLL CONFIG (cont'd)

Returns None.

Programming Notes None.

Example This would configure device 12 to assert (= 1) DIO4 when it is parallel polled and needs service.

```
PPOLL CONFIG 12; &H0B
```

PPOLL DISABLE

Purpose Disables the Parallel Poll response capability of the specified GPIB bus device(s). The KM-488-DD must be an Active Controller.

Syntax **PPOLL DISABLE** *addr*

Alternate Syntax **PPD** *addr*

Parameters **addr** specifies the GPIB BUS address of the device(s) which is to be parallel poll disabled. Multiple **addr** parameters can be included. **addr** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

Returns None

Programming Notes None

Example This will disable the ability of devices 2 and 14 to respond to a parallel poll.

```
PPOLL DISABLE 2,14
```

PPOLL UNCONFIG

Purpose Disables the Parallel Poll Response of all GPIB bus device. The KM-488-DD must be an Active Controller.

Syntax `PPOLL UNCONFIG`

Alternate Syntax `PPU`

Parameters None.

Returns None.

Programming Notes None.

Example This will disable the ability of all devices to respond to a parallel poll response.

```
PPU
```

REMOTE

Purpose Forces the GPIB bus device(s) to the REMOTE mode (ignore the bus).

☞ ***NOTE:*** The KM-488-DD must be a System Controller to execute this command.

Syntax **REMOTE** *addr*

Parameters **addr** is an optional parameter which specifies the GPIB BUS device address to be sent to REMOTE. Multiple devices can be specified. **addr** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

Returns None.

Programming Notes

1. If **addr** is included, those devices will be addressed to listened and the GPIB REN line will be asserted. Once this has happened, they will go to REMOTE. If no **addr** is specified, the GPIB REN line will be asserted. The next time a device(s) is addressed to listen by the Active controller, it will go to REMOTE.

2. It is good programming practice to issue a Local Lockout to prevent the devices from being returned to local mode.

Example This will force devices 5 and 6 to the remote state and address them to listen.

```
REMOTE 05,06
```

REQUEST

Purpose Sets the Serial Poll Response of a KM-488-DD which is a Peripheral.

Syntax `REQUEST; response`

Parameters ; is an optional delimeter.

response is an INTEGER in the range 0 to 255 which represents the serial poll response and the state of the SRQ (Service Request) bit. This serial poll response byte is of the following format:

Serial Poll Response Byte

Bit	7	6	5	4	3	2	1	0
	DIO8	RSV	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
RSV	If this bit is 1, the SRQ (Service Request) line will be asserted to request servicing. Otherwise, SRQ will not be asserted.

Returns None.

Programming Notes

1. When the KM-488-DD is serial polled by the Active Controller, the SRQ bit will be automatically cleared and SRQ unasserted (assuming they were set).
2. Use the STATUS or SPOLL command to check if the Peripheral has been serial polled (check the status of the SRQ bit).

Example This will generate an SRQ and set DIO1 and DIO2 in the Serial Poll Response Byte.

```
REQUEST : 64 + 1 + 2
```

RESET

Purpose Performs a "warm" reset of the KM-488-DD and the GPIB bus.

Syntax `RESET`

Parameters None.

Returns None.

Programming Notes This command is the equivalent of issuing the following commands:

• STOP	• ERROR ON
• DISARM	• FILL NULL
• CONFIG	• LOCAL
• ABORT	• REQUEST 0 (If Peripheral)
• BUFFER INCREMENT	• Clear CHANGE, TRIGGER, and CLEAR STATUS
• DMA SINGLE	• TIMEOUT 10

It also clears all error conditions.

2. If the KM-488-DD is the System Controller, it will assert the GPIB IFC (Interface Clear) line for at least 200 µsecs.

Example This will clear all error conditions.

```
RESET
```

RESUME

Purpose Initiates data transfers between two non-Active Controller GPIB devices, by unasserting the ATN line. The KM-488-DD must be an Active Controller in order to use this function.

Syntax **RESUME**

Parameters None.

Returns None.

Programming Notes This command is normally called after a SEND command has addressed a talker and a listener.

Example This will unassert the ATN line.

```
RESUME
```

SEND

Purpose Sends GPIB commands and data from a string.

Syntax `SEND; subcommand`

Parameters **subcommand** can be any one of the following: CMD, UNT, MTA, MLA, TALK, LISTEN, DATA, and EOI. Multiple subcommands may be specified; however, the length of the entire SEND command string cannot exceed 255 characters. The subcommands are as follows:

CMD - Sends information with the ATN line asserted. This indicates to the bus devices that the characters are to be interpreted as GPIB bus commands. The command may be sent as either a quoted string (e.g., **'COMMAND'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **&H0D** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **SEND CMD 67,79,77,77,65,78,68**).

An EOI cannot be sent with this subcommand, because an EOI with ATN asserted would initiate a Parallel Poll.

UNT - Untalks all devices by sending a GPIB UNTALK command with ATN asserted.

UNL - Unlistens all devices by sending the GPIB UNLISTEN command with ATN asserted.

MTA - Designates the KM-488-DD to address itself to talk by sending the GPIB MTA (My Talk Address) command with the ATN line asserted.

MLA - Designates the KM-488-DD to address itself to listen by sending the GPIB MLA (My Listen Address) command with the ATN line asserted.

SEND (cont'd)

`TALK addr` - Addresses another GPIB device or KM-488-DD to talk by sending a GPIB TAG (Talk Address Group) command with the ATN line asserted. **`addr`** is an integer representing the GPIB BUS device address of the device to talk. This integer ranges from 00 to 30 decimal.

`LISTEN addr` - Addresses another GPIB device(s) or KM-488-DD('s) to listen by sending a GPIB LAG (Listen Address Group) command with ATN asserted. **`addr`** is an integer representing the GPIB BUS device address of the device(s) to talk. This integer ranges from 00 to 30 decimal. Multiple listeners can be specified. If **`addr`** is not specified, all other devices on the GPIB BUS will be designated listeners.

`DATA` - Sends information with the ATN line unasserted. This indicates to the bus devices that the characters are to be interpreted as data. This is the technique which is to be used to send device-dependent commands. (See the IEEE-4888 Tutorial in Appendix C for more information.) The data may be sent as either a quoted string (i.e., **`'DATA'`**) or as a numeric equivalent of an individual ASCII character (i.e., **`13`** decimal or **`&H0D`** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **`SEND DATA 68,65,84,65`**).

This sub-command is useful when you are sending commands which are unique to your interface.

`EOI` - Sends information with the ATN line unasserted. EOI will be asserted when the last character is transmitted. This information is interpreted as GPIB bus data and may be sent as either a quoted string (e.g., **`'xyz'`**) or as a numeric equivalent of an individual ASCII character (e.g., **`13`** decimal or **`&H0D`** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **`SEND EOI 120,121,122`**).

Returns None.

SEND (cont'd)

Programming Notes

1. The maximum length of the SEND command, including any subcommands, is 255 characters. To SEND large amounts of data, use multiple SEND commands.

2. SEND should only be used when a non-conforming device requires a special command sequence or a non-standard GPIB command. Do not use the SEND command unless you are extremely familiar with GPIB.

3. SEND would typically be used to address a device to talk and a second device to listen (such as a scope sending its display to a plotter) so that data passes between the devices without passing through the KM-488-DD . In this case, a RESUME command must follow.

4. If a DATA subcommand is not included in the SEND string, be sure to call RESUME immediately after SEND. This is necessary because the ATN line must be dropped so that the transfer will proceed.

5. The KM-488-DD must be the Active Controller to send commands. Any KM-488-DD can send data.

Example

This will address the KM-488-DD to talk, unlisten all devices, and send the bytes representing HI to device 19.

```
SEND MTA UNL LISTEN 19 DATA 'HI'
```

This will cause a device at address 5 to send data to a device at address 10. It must be followed by a RESUME command.

```
SEND UNL TALK 05 LISTEN 10
RESUME
```

SPOLL

Purpose Initiates an Serial Poll.

Syntax **SPOLL** *addr*

Parameters **addr** is the GPIB bus address of the device to be Serial Polled. This parameter can only be included if the KM-488-DD is the Active Controller. **addr** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

Returns If the KM-488-DD is an Active Controller and the parameter **addr** is specified, an integer in the range 0 to 255 will be returned to **response**. The definition of this integer varies from device to device; however, Bit 6 is always used to indicate whether the device is in need of service. Consult the manufacturer's operator's manual for more information.

If the KM-488-DD is an Active Controller and the parameter **addr** is not specified, the KM-488-DD will read its internal SRQ status bit. This means that **response** will contain an integer either of value 0 or 64 decimal. If **response** is 64, a device is asserting SRQ.

If the KM-488-DD is a Peripheral, it will return an integer in the range 0 to 255. This integer represents the contents of the KM-488-DD's serial poll mode register and can be interpreted according to the format of the serial poll response byte, as shown below. If bit DIO7 (decimal value 64) is asserted, it signifies that the KM-488-DD has not been serial polled since issuing the last REQUEST. This is the way a peripheral can know when the controller has recognized its request for service.

Serial Poll Response Byte

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	DIO8	PEND	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
PEND	If this bit is 1, the SRQ (Service Request) line will be asserted to request servicing. Otherwise, SRQ will not be asserted.

SPOLL (cont'd)

Programming Notes

1. The internal SRQ bit status will not be reset if SRQ is unasserted for any reason other than an ABORT, RESET, or SPOLL.

2. If an Active Controller conducts a serial poll (with or without an address), the internal serial poll bit as indicated by Bit 6 of the Serial Poll byte or S1 from the STATUS command will be cleared. So that a subsequent SPOLL or STATUS will not reveal an SRQ until a new SRQ is received.

3. If two devices are simultaneously asserting SRQ, serial polling one of the devices will cause that device to relinquish SRQ, but the presence of the second device's SRQ will cause a new SRQ to be detected.

Example This serial polls device 22.

```
SPOLL 22
```

STATUS

Purpose Returns a character string describing the current operating state of the KM-488-DD.

Syntax `STATUS`

Parameters None.

Returns A character string describing the current operating state. The string is of the format shown in Table 5-2.

Table 5-2. Status String

Starting Col	# of Cols.	Name and Description

1	1	Operating Mode - Tells if the KM-488-DD is acting as an *Active Controller* or *Peripheral*. Can be C or P where:
		
		C = The KM-488-DD is an *Active Controller*
		P = The KM-488-DD is a *Peripheral*

3	2	Bus Address - Gives the IEEE-488 Bus Address of the KM-488-DD. This is a two-digit decimal integer ranging from 00 to 30.
6	2	Addressed State Change - Indicates if the device has changed addressed state, i.e., if it has cycled between the *Talker*, *Listener*, or *Active Controller* states. This is reset whenever a STATUS is read. This can be one of the following: G0 = There has not been a change in the addressed status. G1 = There has been a change in the addressed status.

9	1	Current Addressed State - Indicates if the KM-488-DD is currently acting as a *Talker* or *Listener*, or is *Idle*. Can be T, L, or I, where:
		
		T = The KM-488-DD is a Talker
		L = The KM-488-DD is a Listener
		I = The KM-488-DD is Idle

11	2	Service Request - Represents the current internal SRQ status where:
		
		S0 = SRQ not asserted
		S1 = SRQ asserted
		
		If the KM-488-DD is in the *Active Controller* mode, the internal SRQ state initially reflects the state of the GPIB bus SRQ. Conducting an SPOLL can clear this bit even though the SRQ line is still asserted.

STATUS (cont'd)

Table 5-2. Status String

Starting Col	# of Cols.	Name and Description
		If the KM-488-DD is acting as a *Peripheral*, the internal SRQ state is set by using the REQUEST command. It is cleared by a serial poll from the Active Controller.
14	3	Error Code - Indicates whether or not an error has occurred. The three characters consist of the letter 'E' followed by a two digit error code. (A list of these error codes is provided in Appendix G.) An error code of 00 indicates no error has occurred.
18	2	Triggered State - Indicates if a Peripheral KM-488-DD has received a GPIB trigger command. This is not updated during DMA CONTINUE transfers. Triggered State values are: T0 = No trigger command has been received. T1 = A trigger command has been received.
21	2	Cleared - Indicates if a Peripheral KM-488-DD has received a GPIB Clear command. This is not updated during DMA CONTINUE transfers. Cleared values are: C0 = No Clear command has been received. C1 = A Clear command has been received.

24	2	Transfer Status - Indicates if a data transfer is in progress. Values which may appear are:
		
		P0 = No transfer
		P1 = DMA CONTINUE Transfer occurring
		P2 = DMA AUTOINITIALIZE Transfer occurring

27	45	Error Message Text - Contains the Error Message Text associated with the given error code (EXX). These error messages are listed in Appendix G.

Programming Notes If the KM-488-DD is the Active Controller, the Service Request line may be cleared (i.e., S0) as a result of a SPOLL, even when the SRQ line is still asserted. See the discussion in SPOLL.

Example This retrieves the status string.

```
STATUS
```

STOP

Purpose	Stops a DMA CONTINUE transfer.
Syntax	`STOP`
Parameters	None.
Returns	None.
Programming Notes	1. If the DMA CONTINUE transfer has been completed, a STOP command will have no effect. 2. Use the BUFFERED command to determine the actual number of characters which were transferred before the STOP command took effect.
Example	This will stop a DMA CONTINUE transfer. `STOP`

TERM

Purpose	Defines the GPIB bus terminator(s) to be used in ENTER and OUTPUT calls.
Syntax	**`TERM`** *`[dir term1 term2 EOI] [NONE]`*

Parameters

`dir` is an optional parameter which represents the GPIB BUS terminator to be programmed, where:

`IN` = the INPUT GPIB BUS terminator sequence.
`OUT` = the OUTPUT GPIB BUS terminator sequence.

If this parameter is omitted, both the input and output GPIB BUS terminators will be set as specified. See Chapter 3 for more information regarding the use of GPIB BUS terminators.

`term1` is an optional parameter which represents the first GPIB bus terminator byte to be used. This terminator byte can be represented as any of the following:

`$char`	where **`char`** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **`char`** must be preceded by a dollar sign (**$**). Hexadecimal values must also be preceded by **&H**. For example, $84 represents the letter "T" as does $&H54.
CR	This represents the *Carriage Return* character (13 decimal, 0D hex).
LF	This represents the *Line Feed* character (10 decimal, 0A hex).
`'X`	where **`X`** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **`'B`** represents the character *B*.

`term2` is an optional parameter which represents the second GPIB bus input terminator byte (in a two-byte terminator) to be sent. This parameter is represented in the same manner as the **`term1`** parameter.

TERM (cont'd)

EOI (End or Identify) is an optional GPIB BUS terminator which indicates that the EOI line will be asserted when the last character is transferred (OUTPUT). Data will continue to be read/transmitted until this terminator is detected (ENTER). Use of the EOI terminator is further described in Chapter 4. See also Programming Note 3 below.

Returns None

Programming Notes

1. The terminators set by the TERM command will change the default settings for the GPIB terminators made during software installation.
2. The termination characters can also be modified by CONFIG.
3. EOI is used differently in input operations than output. During an ENTER, EOI indicates that input will be terminated upon detection of the EOI bus signal; whereas during an OUTPUT it indicates that EOI bus signal will be asserted during transfer of the last byte.
4. The terminator character(s), if specified, is appended to a message in KOUTPUT and checked for in KENTER.

Example This sets the input bus terminator to Horizontal Tab with no EOI detection.

```
TERM IN $09
```

TIME OUT

Purpose Defines a timeout period.

Syntax `TIME OUT` *n.nn*

Parameters **n.nn** is the number of seconds to allow before a time-out. This number must fall within the range 0.00 to 65535.99 seconds. Timeout intervals must be specified two decimal places. Any timeout below 1.00 must include a 0, e.g. 0.61.

☞ ***NOTE:*** Even though you can specify the timeout period to the nearest 1/100 second, the actual interval is always a multiple of 55 milliseconds due to computer limitations. Thus, there is an uncertainty of 55 msec in the actual time out interval. Time out intervals from 0.01 to 0.11 seconds are rounded to 0.11 seconds. Larger intervals are rounded to the nearest multiple of 55 msec.

Returns None

Programming Notes

1. If no timeout period is specified, the default of 10 seconds will be used.
2. To suppress Timeout checking, set **n.nn** to 0.
3. If a DMA CONTINUE transfer is in progress, the KM-488-DD will check for timeouts only for the first byte that is transmitted/received. During other types of transfers, the KM-488-DD will check for a time-out between transmission of bytes.

   You must be certain to check that a DMA CONTINUE transfer has been completed using the BUFFERED command.

Example This will disable timeout checking.

```
TIME OUT 0
```

This will set the timeout to 5 seconds.

```
TIME OUT 5
```

TRIGGER

Purpose Triggers the specified device(s). The KM-488-DD must be an Active Controller.

Syntax `TRIGGER` *addr*

Parameters **addr** is an optional parameter which represents the GPIB bus address of the device to be triggered. Multiple devices may be specified. If **addr** is not specified, then the Active Controller must have previously listened the device(s) to be triggered. **addr** is a primary address (00 - 30) with an optional secondary address (00 - 31), each of which must be specified as two digits, i.e. 05, 10, 0901, etc.

Returns None.

Programming Notes None.

Example This will issue a Group Execute Trigger (GET) to device 4.

```
TRIGGER 04
```

WAIT

Purpose Forces the KM-488-DD driver to wait until a DMA CONTINUE transfer has been completed before returning control to the application program.

Syntax `WAIT`

Parameters None.

Returns None.

Programming Notes

1. The WAIT command is particularly useful in situations where an ENTER BUFFER DMA CONTINUE or OUTPUT BUFFER DMA CONTINUE specifies a terminator. (Remember that the DMA CONTINUE function allows your application program to do some processing while DMA is finishing in the background.)

2. Time out checking, if enabled, is performed while WAITing.

3. This is an explicit wait. The following commands perform an implicit wait:

- ABORT
- CLEAR
- ENTER
- LOCAL
- LOCAL LOCKOUT
- OUTPUT
- PASS CONTROL
- PPOLL
- PPOLL CONFIG
- PPOLL DISABLE
- PPOLL UNCONFIG
- REMOTE
- REQUEST
- RESUME
- SEND
- SPOLL
- TRIGGER

Example This will force the KM-488-DD to wait for a DMA CONTINUE transfer to complete before it returns control to the application program.

```
WAIT
```

This page intentionally left blank.

CHAPTER 6
INTRODUCTION TO PROGRAMMING VIA THE CALL INTERFACE

Chapters 7, 8, 9, 10, and 11 contain language-specific information and a detailed description of all of the KM-488-DD calls. The calls are listed in alphabetical order. Each of the above listed chapters describes the use of the call within one of the supported languages: BASICA, QuickBASIC, Microsoft PASCAL, or TURBO PASCAL. This chapter provides an overview of the call interface and an introduction to the more common commands.

6.1 ACCESSING THE CALL INTERFACE

Once the KM-488-DD driver is installed, CALLs can be issued from a program to access the driver capabilities. (A list of all of the available calls can be found in the rear of this manual.) With the exception of KDIAGOFF, all of the calls have two or more arguments. The details of passing arguments are language-dependent and are discussed in the following chapters.

All calls contain an input parameter **board_nr** and a returned value **err_code**. **board_nr** is an integer 0-3 which selects the particular KM-488-DD that the command affects and **err_code** is an integer which returns an error code after the call. You should perform an error check after each call. No error corresponds to 0. Refer to appendix G for a listing of error numbers and their corresponding codes.

All data is transferred directly between the program data area and the driver; thus the address of the program data structure where the data is to be found or sent is passed as a parameter.

All calls such as KSTATUS, KENTER, or KOUTPUT which require a GPIB bus device(s) to be addressed contain an address argument **mygads**. Address structures can contain from 1 to 15 devices or can specify no address where appropriate.

The calls KENTER and KOUTPUT contain an argument **term_list** which specifies the GPIB terminators to be used.

In addition, there are some miscellaneous arguments (usually integers) which specify the number of bytes to be transferred, a mode of operation, or a switch.

KRESET

When starting a program, it is a good idea to reset the driver by calling KRESET. This returns the driver to its default conditions. If the KM-488-DD is also the System Controller, the GPIB IFC line will be asserted for 500 μsecs.

6.2 GPIB TERMINATORS

Data is sent from the application programs to GPIB bus devices and data is returned to application program from GPIB bus devices. The GPIB BUS Terminators are used to delimit data transmitted between the KM-488-DD driver and the GPIB Bus. There are two types of GPIB BUS Terminators: **Input** and **Output**.

The **GPIB BUS Output Terminator** sequence delimits the data transferred from the KM-488-DD driver to the GPIB bus. These terminator sequences can be comprised of one or two ASCII characters and may include an asserted GPIB EOI signal. Any combination of ASCII characters and EOI may be selected, including EOI alone. If only EOI is used to signal the end of an output data transmission, the EOI signal will be asserted when the last character in the sequence is transmitted. Otherwise, the terminator sequence character(s) is appended to the data characters, and EOI, if used, is asserted on the last character sent.

The **GPIB BUS Input Terminator** sequence marks the end of data transferred from the GPIB bus to the KM-488-DD driver. This terminator sequence can be comprised of one or two ASCII characters and may include detection of an EOI signal. Any combination of ASCII characters and EOI may be selected, including EOI alone.

If only EOI is chosen to signal the end of input data, then all the characters received from the bus device, including the one on which EOI was asserted will be returned to the application program.

In KENTER calls, if only ASCII terminator characters are used, then all the characters including the terminator characters will be returned. However, if ASCII characters are selected in combination with EOI and EOI is received and the complete ASCII terminator sequence has not been received, all of the received characters are returned to the program.

Because it is sometimes convenient to send data in two "pieces" without terminators between the pieces and because different GPIB devices require different GPIB terminators, there will be cases where it will be necessary to change the GPIB terminators within a program.

The KOUTPUT call allows the same data to be sent to multiple GPIB devices. Because KOUTPUT can only specify one set of terminators, care must be taken that all of the listening devices will respond to the terminators used. For other routines, you can modify the GPIB terminator defaults using the KTERM call.

KTERM

This call is used to change the default terminators when transferring data between the KM-488-DD driver and a GPIB bus device.

6.3 KCONFIG

This routine re-configures the operating parameters of the KM-488-DD. These include:

- GPIB Bus Terminators
- Named Devices
- GPIB Bus Address
- EOL Terminators

6.4 CLEAR DEVICES

Devices can be returned to their power-up initial state (ranges and functions) by calling KCLEAR. If the KCLEAR contains an address structure with addresses, then only those devices are cleared (via the GPIB Selected Device Clear command). If the no address structure is selected, then all GPIB bus devices are cleared via the universal Device Clear.

6.5 TRANSMITTING DATA

Data is sent from the program to the GPIB bus by using the KOUTPUT (or in BASICA/Quick BASIC KOUTPUTSTR and KOUTPUTBUFF). An address structure with no addresses can be used if the appropriate GPIB devices are addressed; otherwise an address structure containing the address(es) of the device(s) to receive the data is used. In the latter case, the receiving devices are addressed to listen and the KM-488-DD is addressed to talk.

A terminator structure is also a required argument and can either specify the existing driver GPIB output terminators be used or override those terminators for the one call.

6.6 RECEIVING DATA

Data is received by the program from the GPIB bus by using the KENTER (or KENTERBUFF or KENTERSTR in BASICA or QuickBASIC). An address structure with no addresses can be used, if the appropriate GPIB devices are addressed. Otherwise, an address structure containing the address of the device to send the data is used. In the latter case, the sending device is addressed to talk and the KM-488-DD is addressed to listen.

A terminator structure is also a required argument and can either specify the existing driver GPIB input terminators be used or override those terminators for the one call.

6.7 KTO

A timeout check is available to insure that the calls KENTER, KOUTPUT, KSEND, and KSPOLL, which perform handshake data transfers on the GPIB bus, complete their data transfer within a limited amount of time. This prevents the application program and/or GPIB bus from hanging because a device was turned off or failed to function. The timeout can be set in *CONFIG.DAT* or modified by the KTO call.

The timeout can be disabled by sending 0 for the timeout value; but this is not recommended. The timeout should be long enough so that a data transfer can complete. This will depend on the speed of the GPIB device and the amount of data to be transferred. In normal buffered or DMA without CONTINUE transfers, this includes all the characters transferred. In a DMA with CONTINUE transfer, the driver only monitors the first character transferred so the timeout applies to only that character. After the first character, the transfer occurs automatically while the application program performs other tasks. At some point, the application program will want to verify that the transfer is complete by calling KWAITC. When KWAITC is issued, the timeout function will start. Alternatively, the KBUFD call can be used for verifying the completion of a transfer.

6.8 KBUFD

The KBUFD call can be used to determine the number of characters transferred in the last buffered transfer. This is particularly useful where a DMA CONTINUE transfer has been initiated and the application program has worked on other tasks. By calling KBUFD, the application program can determine the number of byte transferred thus far. If the number of transferred characters equals the expected number to be transferred, the application program knows the transfer is complete. Otherwise, the application program can check back later or issue KWAITC and wait for the completion of a timeout.

6.9 KTRIGGER

GPIB devices can be triggered by a GPIB command to perform some function such as to make a measurement. KTRIGGER is used to trigger the GPIB devices specified by address argument.

6.10 KSTATUS AND KSPOLL

Many GPIB bus devices have a status register which can be read by the controller by means of a serial poll mechanism. To eliminate the need for the controller to serial poll a device when no change of status has occurred, many devices can assert an SRQ (Service Request) line on a status change. In this way, the controller can be performing other tasks while devices do not need attention and furthermore, some instruments do not take kindly to being queried while they are in the middle of something. Because all bus devices share the same SRQ line, the controller may have to poll several devices before it arrives at the one requesting service.

The Keithley 196, for example, can be triggered to make a measurement and assert SRQ when the measurement is complete. An application program can determine when an SRQ is present by looking at the driver status by means of the KSTATUS call. The status string returned by KSTATUS is described in the KSTATUS call description. When the status reveals the presence of an SRQ, a serial poll can be conducted by the KSPOLL call which returns a status byte from the device polled. If the spoll byte indicates that DIO7 of the GPIB bus is asserted, then the device is asserting SRQ. The act of serial polling a device asserting SRQ causes the SRQ of the device to be cleared.

6.11 LIGHTPEN INTERRUPTS

The KM-488-DD board can be configured to generate hardware interrupts on receipt of an SRQ. The KM-488-DD driver does not support hardware interrupts; but does implement the lightpen "pseudo" interrupt.

To employ the lightpen interrupt, the driver must be configured to accept interrupt at load time via the *CONFIG.DAT* file. Then, the driver will periodically check for the SRQ and set a flag that is normally set when a lightpen is activated if the presence of an SRQ is detected. The application program must then check for the presence of the lightpen interrupt.

In BASIC and QuickBASIC, if the PEN ON command is issued the lightpen interrupt will be checked at the completion of each BASIC command. (A FOR loop is considered a single command so detection does not take place within a FOR loop.) If an interrupt is detected, the program can jump to a routine which services the interrupt and then return to the main routine. In other languages, it is necessary to decide your own strategy for checking the lightpen interrupt. (See *IEEEIO.C* for an example of testing the lightpen from C.)

6.12 KPPOLL and KPPC

A few GPIB devices support a parallel poll mode of operation. A device which supports parallel polling can be configured to assert one particular GPIB data line, when the controller conducts the parallel poll. Many times devices can be configured remotely over the GPIB bus by using the KPPC call. If different devices assert different data lines, the controller can rapidly determine which device needs service by issuing a parallel poll and then interpreting the result.

Many times the device's parallel response is asserted when the device asserts SRQ. In most cases, the controller will generally perform a serial poll of the device(s) which a parallel poll has revealed needed service. This is to obtain more complete status information and to cause the device to unassert its SRQ and parallel responses.

A parallel poll is conducted by the KPPOLL call.

6.13 KREQUEST

The KM-488-DD can function in the non-controller or peripheral mode. In this mode, the KM-488-DD can set its serial poll response which can be read by an Active Controller. The seventh bit, when set, is reserved to generate an SRQ when desired. The significance of the other bits is user-defined. The KREQUEST call can be used to write the status byte. A KSPOLL call with an address argument specifying no address can be used to determine if the SRQ has been serviced. If the value returned is 64, then the SR is still pending, if 0 then SRQ has been serviced by the controller and the KM-488-DD has automatically stopped asserting the SRQ.

CHAPTER 7 USING THE CALL INTERFACE IN BASICA

BASICA uses the **CALL** statement to access GPIB language extension for BASICA from within a user program. Before any **CALL** statement can be executed, these things must occur:

- The location of the routines (**offset addresses**) must be specified (see *HEADER.BAS*)
- The **parameters** used by the routine must be declared
- The interface module (*KM488BAS.BIN*) must be loaded

The **offset address** of each KM-488-DD Routine must be identified so that BASIC knows how many bytes from the start of the (**BLOAD**ed) *KM488BAS.BIN* file to jump to execute the called routine. Each KM-488-DD interface routine must be assigned a variable which is set to the offset for that routine. The offsets of the routines in *KM488BAS.BIN* are not user-selectable. They have been defined for you in the file *HEADER.BAS*. For example, the offset for the **KABORT** routine is always zero; therefore you must include the line **KABORT = 0** before calling KABORT.

Note that any name can be used for these routines, as long as the alternate name matches the offset of the desired function. For example, if we define **KABT = 0** and **KABORT = 0** within a program, the statements **CALL KABT** and **CALL KABORT** will physically execute the same function.

Each KM-488-DD Interface Routine requires certain **parameters** to be provided. These parameters are always integer or string variables. They must be defined prior to executing the **CALL** statement. The variable names must be enclosed within parentheses and follow the function name within the CALL statement, for example:

```
BOARDNR% = 0 : ERRORFLAG% = 0
CALL KABORT(BOARDNR%,ERRORFLAG%)
```

These call parameters are used to pass values into and out of the KM-488-DD driver. When passing values into a call routine, you must equate a named variable of the appropriate type with the desired value, and subsequently pass that variable name into the call.

The example below shows the proper way to initiate a CALL statement sequence. In this example, the variable names **BOARD%** and **ERRFLAG%** have been used to pass the values 0,0 into the KABORT routine. Note that any legal BASICA name can be assigned to these variables. However, they must be the correct data type and value, and must be passed into a callable routines in the same order as shown in the routine descriptions.

```
xx  BLOAD "KM488DAS.BIN", 0
xx  KABORT=0:BOARD%=0:ERRFLAG%=0

'Gives offset of KABORT routine & variable definitions
xx  CALL KABORT(BOARD%,ERRFLAG%)     'uses call statement
```

Software Configuration

A number of KM-488-DD configuration parameters are set via the *CONFIG* program. (See Chapter 2.) Some of these can be modified within a program (i.e., at runtime). The defaults for these are listed in Table 7-1. There are other defaults you may have to re-program if you are using File I/O Commands in the same program as the call interface routines.

Table 7-1. Default KM-488-DD Operating Parameters

Parameter	Default
Device Timeout	10.0 seconds
GPIB Bus Input Terminator	CR LF with no EOI detection
GPIB Bus Output Terminator	CR LF with no EOI assertion
EOL Terminators	CR LF

The KCONFIG call can be used to reset the GPIB input and output bus terminators.

The KTERM call can be used to change the default GPIB bus terminators settings.

The KTO call can be used to change the default device timeout value.

Programming Notes

1. In BASICA, only variables may be passed into and out of functions. Values are not allowed as CALL arguments.

2. You must also be sure to include all of the parameters for the Call Interface Routine. The parameters must be the same data type and appear in the same order as those given. You can, however, change their names. BASICA has no means for checking that the exact number of parameters are given or that the parameters of the appropriate type. If an incorrect number or type of parameters are specified, your program may crash.

3. Strings are limited to the BASICA maximum of 256 characters.

4. All integers are treated by the KM-488-DD routines as unsigned values (0 to 65535). However, BASICA treats them as signed magnitudes (-32768 to +32767). When you need to express a value which is greater than or equal to 32768, you will need to express it in one of two ways:

   • Convert it to a hexadecimal value. Be sure to prefix these values with **&H** when equating them to a variable name. Legal hexadecimal values range from 0 to &HFFFF and can be used to represent values from 0 to 65535.

   • Use unsigned values from 0 to 32767 as is, but for values of 32768 to 65535 subtract 65536.

5. The file *HEADER.BAS* has been provided to assist you with defining CALL routine offsets. This is a BASICA source file which predefines the offsets. It can be modified to suit your needs.

6. Do not name your variables with the same name as any of the KM-488-DD routines.

7.1 DESCRIPTION FORMAT

The routines are described according to the following criteria.

Purpose This section provides a "one-line" description of the routine. A more detailed general description of each routine is provided in Chapter 3.

Offset This section gives the BASICA offset for each routine.

Syntax This section gives the syntax for each routine. The provided syntax assumes that the input parameters are passed in as variables.

Parameters This section describes the input parameters. In some instances, an array of multiple parameters may be specified (for example, multiple device addresses).

MYGADS(N*2) is used in many calls. This is an array of **n** (1 to 15) pairs of GPIB addresses which are required by the call. A GPIB address pair consists of two integers. The first is the primary address of a device. The second is the secondary address. If the first is set to -1, the device is assumed to not exist. If no address is provided, it is assumed that the bus is already addressed. When the KM-488-DD driver encounters a primary address = -1, it assumes that there are no more addresses in **mygads**.

An example of a structure with two devices is:

```
DIM MYGADS%(5)   ' 2*3 -1
MYGADS%(0) = 3   ' first device - pri addr
MYGADS%(1) = -1  ' first device - no sec addr
MYGADS%(2) = 15  ' second device - pri addr 15
MYGADS%(3) = 2   ' second device - sec addr 2
MYGADS%(4) = -1  ' third device doesn't exist
```

If the KM-488-DD is a Peripheral (in which case the call will not require a GPIB bus address), you will want to use the following **MYGADS** array:

```
MYGADS%(0) = -1 : MYGADS%(1) = -1 : MYGADS%(2) = -1
```

Returns This section describes any values which are returned by the routine.

ERRCODE% - All calls except KUNPACK, KPACK, KSWAP, KDIAGON and KDIAGOFF return an integer error code which will be 0 if no error was detected. It is good practice to check **ERRCODE%** after each call and provide a routine to handle errors if they arise.

Programming Notes This section lists any special programming considerations for the routine.

Example This section gives a programming example using the routine. Note that all of the examples given, assume that the offsets have been set. (The *HEADER.BAS* file which defines the BASICA call routine offsets definitively has been included in the first section of the program.)

KABORT

Purpose This routine allows the KM-488-DD to take control of the GPIB bus by forcing all other devices on the bus to the idle state. The KM-488-DD must be the Active Controller.

Offset 0

Syntax

```
XX   BOARDNR% =
XX   ERRCODE% =
XX   CALL KABORT(BOARDNR%, ERRCODE%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the KM-488-DD is both the Active and System Controller, KABORT asserts the IFC line to retain control of the bus. Otherwise, the KM-488-DD will assert the ATN line, and then untalk and unlisten all devices.

Example

```
770    BRDNUM% = 0 ' First GPIB Board
1572   '*** Seize Control of GPIB Bus ***
1573   '
1575   CALL KABORT(BRDNUM%, ERRNUM%)
1580   IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KARM

Purpose Defines the conditions under which the lightpen status becomes true.

Offset 3

Syntax

```
XX  BOARDNR% =
XX  ARMCODE% =
XX  ERRCODE% =
XX  CALL KARM(BOARDNR%, ERRCODE%, ARMCODE%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

ARMCODE% is a 16-bit integer describing on which conditions a light pen interrupt may occur. The integer is of the format:

Bit	15-9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC	**Address Status Change.** If this bit is set to 1, the lightpen status will become true when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller). This encompasses all the conditions marked with an asterisk (*) below.
CIC*	**Controller in Charge.** If this bit is set to 1, the lightpen status will become true when the control is passed to the KM-488-DD and it becomes the Active Controller.
PER*	**Peripheral.** If this bit is set to 1, the lightpen status will become true when the KM-488-DD passes control to another device and becomes a Non-Active Controller.
DET	**Device Triggered.** If this bit is set to 1, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.

KARM (cont'd)

SRQ	**Internal SRQ.** If this bit is set to one, the lightpen status will become true when the KM-488-DD's internal Service Request Bit (SRQ) is set.
DEC	**Device Cleared.** If this bit is set to one, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA*	**Listen Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Listener and can accept data from the GPIB.
TA*	**Talk Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Talker and can output data to the GPIB.
IDLE*	**Idle.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is unaddressed as a Talker or a Listener.

☞ ***NOTE:*** The conditions marked by an asterisk (*) will force the lightpen status to true only when the KM-488-DD's address status has changed. This is indicated by the state of the Address Change Bit in the Status byte. See the KSTATUS or KQUIKSTAT routine descriptions for more information.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The driver must be configured for lightpen emulation at load time via the *CONFIG.DAT* "pipe" file, as the default for lightpen emulation is off.

KARM (cont'd)

Example

```
770   BRDNUM% = 0 ' First GPIB Board
1150  ' SET UP LIGHT PEN INTERRUPT
1160  '
1170  'Enable light pen interrupt detection on SRQ
1180  '
1190  ON PEN GOSUB 2000
1195  'BASICA connection of subroutine to the light pen
1200  PEN ON
1210  'BASICA command to turn on light pen interrupt
1270  ARMCODE%=&H10
1271  'KM-488-DD will signal light pen status hit on SRQ
1275  ERRNUM% = 0
1280  CALL KARM(BRDNUM%, ERRNUM%, ARMCODE%)
1285  IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
1290  '
1300  'other program operations
.
.
.
2000  'lightpen handler here
.
.
.
3000  RESUME NEXT   'continue program
```

KBUF

Purpose Selects in which direction the memory is to be addressed for both DMA and non-DMA buffered transfers.

Offset 6

Syntax

```
XX  DIR% =
XX  BOARDNR% =
XX  ERRCODE% =
XX  CALL KBUF(BOARDNR%, ERRCODE%, DIR%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

DIR% selects the direction in which the buffer is to be read. If **DIR%** is equal to 1, the buffer address will be incremented. If **DIR%** is equal to 0, the buffer address will be decremented.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the buffer's address is to be incremented during a bus I/O operation, be sure to use the lowest address in the designated buffer. Likewise, if it is to be decremented, select the highest address.

Example

```
770 BRDNUM% = 0  ' First GPIB Board
780 DECRM% = 0   '  Decrement Memory Address On DMA
790 ERRNUM% = 0
1000 'Set Direction Of Memory Address Change With DMA
1010 CALL KBUF(BRDNUM%, ERRNUM%, DECRM%)
1020 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KBUFD

Purpose Returns the number of characters transferred in the last buffered transfer or, in the case of DMA transfers, the current transfer.

Offset 9

Syntax

```
XX   BOARDNR% =
XX   ERRCODE% =
XX   BCOUNT% =
XX   CALL KBUFD(BOARDNR%, ERRCODE%, BCOUNT%)
```

Parameters BOARDNR% is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns ERRCODE% is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

BCOUNT% represents the number of characters which were transferred during the last buffered transaction. This will be an integer (16-bits) in the range 0 to 65,535.

Programming Notes

1. If the data was transferred in the DMA CONTINUE mode, KBUFD returns the number of characters transferred thus far.

2. It is useful to call this routine after a buffered KENTER has been performed. This will help you to determine if the expected number of characters has been received and/or the transfer has terminated as the result of a detected terminator.

Example

```
770   BRDNUM% = 0 ' First GPIB Board
780   ERRNUM% = 0
1000   '  *** Return Number Of Bytes In Last GPIB BUS
1001   '  Data Transfer ***
1010 CALL KBUFD(BRDNUM%, ERRNUM%, XFER%)
1020 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KCLEAR

Purpose The KM-488-DD must be an Active Controller. Resets device(s) to their power-up parameters.

Offset 12

Syntax

```
XX   BOARDNR% =
XX   ERRCODE% =
XX   DIM MYGADS%() 'for N address pairs
XX   CALL KCLEAR(BOARDNR%, ERRCODE%, MYGADS%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

MYGADS (N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If **MYGADS** contains no addresses, then the universal GPIB DCL (Device Clear) command is issued and all devices will be cleared.

2. If **MYGADS** contains addresses, then those devices specified will be listened and sent a GPIB SDC (Select Device Clear) Command.

Example

```
770  BRDNUM% = 0 ' First GPIB Board
780  ERRNUM% = 0
600  DIM NOADR%(3)
620  NOADR%(0) = -1 ' first primary address = NONE
1000 '
1610 ' With No Addresses Issue The Universal Device Clear
1620 CALL KCLEAR(BRDNUM%, ERRNUM%, NOADR%(0))
1630 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KCONFIG

Purpose Resets the KM-488-DD to its default operating parameters or to those specified.

Offset 15

Syntax

```
XX   BOARDNR% =
XX   ERRCODE% =
XX   CFGSTRING$ =
XX   STRINGLEN% =
XX   CALL KCONFIG(BOARDNR%, ERRCODE%, CFGSTRING$, STRING-
LEN%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

CFGSTRING$ is a user-defined string. This string can be comprised of any of the following options:

/T *[term term EOI]* Sets the input and output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/T**, the GPIB bus terminators will be set to none.

term is an optional parameter which represents a terminator byte to be used. This terminator byte can be represented as any of the following:

$char where **char** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **char** must be preceded by a dollar sign ($).

Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.

CR This represents the *Carriage Return* character (13 decimal, 0D hex).

LF	This represents the *Line Feed* character (10 decimal, 0A hex).
'X	where **X** represents a printable ASCII character (32 to 254 decimal). The character must be preceded by an apostrophe ('), for example: **'B** represents the character *B*.

EOI (End or Identify) is an optional GPIB BUS terminator. If included, the KM-488-DD will detect/assert the GPIB bus EOI line. (This would indicate that the last character has been sent.) Data will continue to be read until this terminator, a valid terminator sequence, or both are detected.

/TI *[term term EOI]*	Sets the input GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TI**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/TO *[term term EOI]*	Sets the output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TO**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/E *[term term]*	Sets the input and output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **term** is described above.

KCONFIG (cont'd)

/EI *[term term]*	Sets the input EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **term** is described above.
/EO *[term term]*	Sets the output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.)
/N name addr *[secaddr]*	Configures a named device. Sets its address to the given value and its GPIB bus terminators to the current GPIB bus terminator settings. (Note that this should only be used if the application program also includes File I/O Commands utilizing the named device configurations.) **name** is the name of the device. **addr** is the primary GPIB address (00 to 30). **secaddr** is the secondary GPIB address (00 to 31).

☞ ***NOTE:*** Do not create a named device with the same name as an existing directory on the current working disk.

If **CFGSTRING** is empty, then the configuration will revert to the load-time configuration.

STRINGLEN% is an integer from 0 to 255 which gives the length of the **CFGSTRING** string.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KCONFIG (cont'd)

Programming Notes

1. The **/E**, **/EI**, **/EO**, and **/N** options will only take effect if File I/O Commands are used within the same application program as the callables.

2. If you are converting an application program previously written using the File I/O commands to callables, be sure to note that the CONFIG string for the KCONFIG call does not include the "CONFIG" keyword (i.e., as it would if you were using the CONFIG File I/O command).

3. Primary and Secondary Addresses must each be two characters long, e.g. 01 or 0209.

Example

```
760 CONF$ = STRING$(20,32)    'initialize string
770 BRDNUM% = 0               ' First GPIB Board
780 ERRNUM% = 0
1000 ' *** Configure Terminators for Carriage Return,
1005 ' Line Feed With EOI ***
1010 CONF$ = "/T CR LF EOI"
1020 STNLEN% = LEN(CONF$)                'Length of CONF$
1030 CALL KCONFIG(BRDNUM%, ERRNUM%, CONF$, STNLEN%)
1040 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KDIAGOFF

Purpose	Disables file diagnostics.
Offset	18
Syntax	`XX CALL KDIAGOFF()`
Parameters	None.
Returns	None.
Programming Notes	None.
Example	`1000 CALL KDIAGOFF()`

KDIAGON

Purpose Turns file diagnostics on. (Default is off)

Offset 21

Syntax

```
XX   FILENAME$ =  "MYDIAG.DAT"
XX   STRLEN% =
XX   CALL KDIAGON(FILENAME$, STRLEN%)
```

Parameters **FILENAME$** is a string containing the name of the file to which the diagnostic information is to be written.

STRLEN% is the length of the string representing the name of the file.

Returns None.

Programming Notes This will echo the value of call parameters into a file in readable form.

Example

```
760  DIAGON$ = STRING$(10,32) 'initialize string
1010 DIAGON$ = "DEBUG.DAT"   'Target Diagnostic File Name
1020 STNLEN% = LEN(DIAGON$)  'Length of DIAGON$
1030 ' Turn Driver Diagnostics On
1040 CALL KDIAGON(DIAGON$,STNLEN%)   'File Name & Length
```

KDISARM

Purpose Resets previously set conditions which cause lightpen interrupts.

Offset 24

Syntax

```
XX   BOARDNR% =
XX   ARMCODE% =
XX   ERRCODE% =
XX   CALL KDISARM(BOARDNR%, ERRCODE%, ARMCODE%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

ARMCODE% is a 16-bit integer describing which conditions should be reset. The integer is of the format:

Bit	15 - 9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC	**Address Status Change.** If this bit is set to 1, light pen status will not change when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).
CIC	**Controller in Charge.** If this bit is set to 1, light pen status will not change when the control is passed to the KM-488-DD and it becomes the Active Controller.
PER	**Peripheral.** If this bit is set to 1, light pen status will not change when the KM-488-DD passes control to another device and becomes a Non-Active Controller (Peripheral).
DET	**Device Triggered.** If this bit is set to 1, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.

KDISARM (cont'd)

SRQ	**Internal SRQ.** If this bit is set to one, light pen status will not change when the KM-488-DD's internal Service Request Bit (SRQ) is set.
DEC	**Device Cleared.** If this bit is set to one, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA	**Listen Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Talker and can output data to the GPIB.
IDLE	**Idle.** If this bit is set to one, light pen status will not change when the KM-488-DD is unaddressed as a Talker or a Listener.

Returns

`ERRCODE%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

None.

Example

```
770  BRDNUM% = 0 ' First GPIB Board
780  ERRNUM% = 0
1150 '****  DISABLE LIGHT PEN INTERRUPT  ***
1160 '
1170 'Disable light pen interrupt detection on SRQ
1180 '
1200 PEN OFF   'BASICA command to turn off light pen
1270 DISARM%=&H10
1275 'KM-488-DD will clear light pen status bit of SRQ
1280 CALL KDISARM(BRDNUM%, ERRNUM%, DISARM%)
1285 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
1290 '
1920 CALL KDISARM(BRDNUM%, ERRNUM%, DISARM%)
```

KDMA

Purpose Sets DMA transfer mode.

Offset 27

Syntax

```
XX   BOARDNR% =
XX   DMAMODE% =
XX   ERRCODE% =
XX   CALL KDMA (BOARDNR%, ERRCODE%, DMAMODE%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

DMAMODE% is a 16-bit integer which represents the DMA mode to be used. If no **DMAMODE%** is given, then single mode without autoinitialization will be assumed. The **DMAMODE%** integer is interpreted as follows:

Bit	15 - 3	2	1	0
	0	SIN	DEM	AUTO

Where:

SIN SINGLE. When this bit is set to one, the SINGLE mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus and transfers a single byte of data. Control of the bus is then returned to the microprocessor.

DEM DEMAND. When this bit is set to one, the DEMAND mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus. The DMA controller retains control of the bus until the DMA request signal is unasserted. Once this signal has been unasserted for more than one microprocessor clock cycle, control of bus is returned to the microprocessor. This mode allows the DMA controller chip to pass data at a slightly faster rate and the microprocessor to access the bus when it is not needed. Note that SINGLE and DEMAND mode cannot be used simultaneously.

KDMA (cont'd)

AUTO AUTOINITIALIZE. When this bit is set to one, the AUTOINITIALIZE mode is selected. Under normal circumstances, the DMA controller transfers the specified number of bytes to/from the PC memory from the given starting address and terminates when completed. When the AUTOINITIALIZE mode is enabled, the DMA controller will reset the original byte count, reset the initial address, and repeat the transfer again. The AUTOINITIALIZE option is only in effect until the next KENTER or KOUTPUT buffered transfer is completed. Additionally, if a KENTER or KOUTPUT transfer in DMA CONTINUE mode is selected, the AUTOINITIALIZE option will only be in effect for that transfer.

Returns None.

Programming Notes

1. All DMA AUTOINITIALIZE transfers should occur entirely within a single DMA 64 KByte page. So, if a DMA buffer operation involves a transfer of more than 64K or occur across a DMA page boundary, the AUTOINITIALIZE option affects only those bytes in the last transfer occurring within one DMA page.

   Also note that DMA AUTOINITIALIZE transfers specified for buffers that are located in more than a single DMA page are unpredictable.

2. It is impossible for a program to halt a DMA AUTOINITIALIZE operation unless the DMA CONTINUE option is selected. This is because the driver does not return control to a program using non-CONTINUE operations until the transfer completes.

Example

```
770 BRDNUM% = 0     ' First GPIB Board
800 ERRNUM% = 0
830 AUTOSING%= 4 + 1  'Single Mode With Autoinitialize
1000 '*** Set Up DMA Mode To Single With Autoinitialize
***
1010 CALL KDMA(BRDNUM%, ERRNUM%, AUTOSING%)
1020 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KENTERBUFF

Purpose Allows the KM-488-DD to receive data from a GPIB bus device into an integer BASIC array.

Offset 30

Syntax

```
XX   BOARDNR% =
XX   ERRCODE% =
XX   DIM DATA%()
XX   COUNT% =
XX   MODE% =
XX   DIM MYGADS%()
XX   DIM TERMLIST%()
XX   CALL KENTERBUFF(BOARDNR%, ERRCODE%, DATA%(0),
COUNT%, MODE%, MYGADS%(0), TERMLIST%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

DATA% is an integer array large enough to accept the returned data. If a non-zero **COUNT%** is specified, **DATA% (COUNT%/2)** will suffice for any returned data. If **COUNT%** is 0 and terminator characters are used, then **DATA% ()** must be large enough to accommodate both data and terminators returned by the device.

COUNT% is an integer representing the number of characters to be read. Valid counts are within the range 0 to 65535 decimal, or from 0 to &HFFFF hex. When 0 is used, the KENTER will stop when the termination specified by **TERMLIST%** is satisfied.

MODE% is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

KENTERBUFF (cont'd)

Where:

CONT	**CONTINUE.** This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer. If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked or a pre-specified GPIB BUS input terminator is detected.
DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer. (See also the KDMA call description.)

MYGADS (N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

TERMLIST% (3) is an integer array which defines the GPIB bus terminators to be used during the KENTERBUFF call. These terminators are used in conjunction with **COUNT** to terminate the input process. Terminator choices include whether or not EOI will be detected to signal the end of input and which character sequence, if any, is to be detected as an end of message code. The default terminator scheme is initially set up by the *CONFIG.DAT* file and can be modified by calling KTERM and KCONFIG. **TERMLIST** can either specify the use of the default terminators or temporary overrides.

TERMLIST% (3) is comprised of the following elements:

TERMLIST%1 (0): **EOI detection.** Possible values are:

-1(&HFFFF)	Use the default EOI setting.
0	Don't Detect EOI.
1	Detect EOI.

KENTERBUFF (cont'd)

`TERMLIST%(1)`	**1st GPIB bus terminator.** Possible values are:		
:		-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.
		0 to 255	Detect this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
`TERMLIST%(2)`	**2nd GPIB bus terminator.** Possible values are:		
:		-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
		0 to 255	Detect this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following **TERMLIST** can also be used:

```
DIM TERMLIST%(2)
TERMLIST(0) = -1
TERMLIST(1) = -1
TERMLIST(2) = -1
```

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KENTERBUFF (cont'd)

Programming Notes

1. If the KM-488-DD is a Peripheral, an active controller must address KM-488-DD as a listener before KENTERBUFF is called. In this instance, you should define **MYGADS** to include no addresses. This is described in section 7.1.

2. If the KM-488-DD is the Active Controller, it will automatically address itself as a listener and another device defined in **MYGADS** as the talker. Since only one device can be a talker, the address group **MYGADS** can contain only one address. If a device has been previously addressed to talk and the KM-488-DD previously addressed to listen, then **MYGADS** can specify no addresses (See section 7.1) and the bus will not be re-addressed.

3. If **COUNT%** is specified as 0, then KENTER will terminate when the terminator(s) specified by **TERMLIST%** is encountered.

4. If it is desirable to terminate on **COUNT%** only and not use the default terminators, then you must select no terminator overrides in **TERMLIST%**, in addition to giving the count. In this case, a byte by byte synchronous transfer is executed.

5. If both counts and termination are specified, the KENTER will terminate on whichever condition is encountered first.

6. If the condition(s) for termination is not encountered and the device has no more data, KTERM will timeout unless the timeout has been disabled.

7. To manipulate the data, which is returned in the form of two data bytes per integer location, you may need to use KUNPACK.

KENTERBUFF (cont'd)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
400 '
470 MODE% = 1 ' DMA mode flag -- 0: no DMA
480           '                  1: DMA Without CONTINUE
490           '                  3: DMA With CONTINUE
500 ERRNUM% = 0
890 DIM GADS%(3) ' Address Array for one device
900 GADS%(0) = 5 ' first primary address = 5
910 GADS%(1) = -1 ' first secondary address = NONE
920 GADS%(2) = -1 ' second primary address = NONE
1000 '
1090 DIM TERM%(3) ' Default terminator array
1100 TERM%(0) = -1  'Use default EOI
1110 TERM%(1) = -1 ' Use default first terminator
1120 TERM%(2) = -1 ' Use default second terminator
1130 '
1240 NUMD% = 1028      'Number of bytes to transfer
1280 DIM DMAIN%(NUMD%/2) 'Array to hold DMA input data
1300 '
1800 'Enter 1028 bytes from the device at GPIB address 5
2460 CALL KENTERBUFF(BRDNUM%, ERRNUM%, DMAIN%(0), NUMD%,
MODE%, GADS%(0), TERM%(0))
```

KENTERSTR

Purpose Allows the KM-488-DD to receive data from a GPIB bus device into a BASIC data string.

Offset 33

Syntax

```
XX  BOARDNR% =
XX  ERRCODE% =
XX  DATA$ =
XX  COUNT% =
XX  MODE% =
XX  DIM MYGADS%()
XX  DIM TERMLIST%()
XX  CALL KENTERSTR(BOARDNR%, ERRCODE%, DATA$, COUNT%,
MODE%, MYGADS%(0), TERMLIST%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

DATA$ is a string large enough to accept the returned data. If a non-zero **COUNT%** is specified, the string should be of lengthCOUNT%. If **COUNT%** is 0 and terminator characters are used, the string must be large enough to accommodate both data and terminator characters returned by the device.

COUNT% is an integer representing the number of characters to be read. Valid counts are within the range 0 to 256 decimal, or from 0 to &HFF. When **COUNT%** is 0, the KENTERSTR will stop when the terminators specified by **TERMLIST%** is satisfied.

MODE% is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

KENTERSTR (cont'd)

Where:

CONT	**CONTINUE.** This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer. If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked or a pre-specified GPIB BUS input terminator is detected.
DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer.

MYGADS(N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

TERMLIST%(3) is an integer array which defines the GPIB bus terminators to be used during the KENTERSTR call. These terminators are used in conjunction with **COUNT** to terminate the input process. Terminator choices include whether or not EOI will be detected to signal the end of input and which character sequence, if any, is to be detected as an end of message code. The default terminator scheme is initially set up by the *CONFIG.DAT* file and can be modified by calling KTERM and KCONFIG. **TERMLIST** can either specify the use of the default terminators or temporary overrides.

TERMLIST%(3) is comprised of the following elements:

TERMLIST%(0) : **EOI detection.** Possible values are:

-1(&HFFFF)	Use the default EOI setting.
0	Don't Detect EOI.
1	Detect EOI.

KENTERSTR (cont'd)

`TERMLIST%(1)` :	**1st GPIB bus terminator.** Possible values are:	
	-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.
	0 to 255	Detect this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
`TERMLIST%(2)` :	**2nd GPIB bus terminator.** Possible values are:	
	-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
	0 to 255	Detect this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default **`TERMLIST%`** can also be used:

```
DIM TERMLIST%(3)
TERMLIST%(0) = -1
TERMLIST%(1) = -1
TERMLIST%(2) = -1
```

Returns **`ERRCODE%`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KENTERSTR (cont'd)

Programming Notes

1. If the KM-488-DD is a Peripheral, an active controller must address the talker as a listener and some other device as the talker, before KENTERSTR is called. In this instance, you should specify no addresses in **MYGADS%**. This is described in section 7.1.

2. If the KM-488-DD is the Active Controller, it will automatically address itself as a listener and another device defined in **MYGADS%** as the talker. Since only one device can be a talker, the address group in **MYGADS%** can contain only one address. If a device has been previously addressed to talk and the KM-488-DD previously addressed to listen, then **MYGADS%** can contain no addresses (See section 7.1) and the bus will not be re-addressed.

3. If **COUNT%** is specified as 0, then KENTERSTR will terminate when the terminator(s) specified by **TERMLIST%(3)** is encountered.

4. If it is desirable to terminate on **COUNT%** only, and not use the default terminators, then you must select no terminator overrides in the **TERMLIST%(3)** array. In this case, a byte by byte synchronous transfer is executed.

5. If both counts and termination are specified, the KENTERSTR will terminate on whichever condition is encountered first.

6. If the condition(s) for termination is not encountered, KTERM will timeout unless the timeout has been disabled.

Example

```
370 BRDNUM% = 0 ' First GPIB Board
420 INDAT$ = STRING$(18,32)  'String for Data from DMM
470 MODE% = 0 ' DMA mode flag -- 0: no DMA
480           '                 1: DMA Without CONTINUE
490           '                 3: DMA With CONTINUE
590 DIM EADS%(3) ' addressing array for ENTER command
600 EADS%(0) = 12 ' first primary address = 12
610 EADS%(1) = -1 ' first secondary address = NONE
620 EADS%(2) = -1 ' second primary address = NONE and
NONE to follow
630 ' ONLY ONE address is allowed for ENTER, EADS%(2)
must = -1
640 '
820 ' Terminator Arrays
830 '
840 DIM ETERM%(3) ' ENTER terminator array
850 ETERM%(0) = -1 ' NO EOI OVERRIDE
860 ETERM%(1) = &HD ' first terminator = CR
870 ETERM%(2) = &HA ' second terminator = LF
1800 'Enter the character string from the device at
1801 'GPIB address 12
1805 COUNT%=0
1806 ERRNUM% = 0
1810 CALL KENTERSTR(BRDNUM%, ERRNUM%, INDAT$, COUNT%,
MODE%, EADS%(0), ETERM%(0))
1920 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KERROR

Purpose Enables/Disables display of Error Messages.

Offset 39

Syntax

```
XX   BOARDNR% =
XX   ERRCODE% =
XX   ESW% =
XX   CALL KERROR(BOARDNR%, ERRCODE%, ESW%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

ESW% is an integer which determines if error message display function will be enabled or disabled. If **ESW%=1**, the error message display is enabled. (This is the default.) If **ESW%=0** is specified, the error message display is disabled.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If KERROR has been used to enable error messages, the message will only reveal that <u>an error has occurred</u> for the KM-488-DD calls. It does not identify what type of error occurred. The KDIAGON routine for should be used to help debug applications utilizing the KM-488-DD calls, and the KSTATUS routine used for identifying the error number.

2. Programs can also check **ERRCODE%** after each call to identify the exact nature of call errors.

Example

```
770 BRDNUM% = 0 ' First GPIB Board
780 ERRON% =  1 ' Enable Error Display
800 ERRNUM% = 0
1000 '   *** Enable Error Display ***
1010 CALL KERROR(BRDNUM%, ERRNUM%, ERRON%)
1020 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KFILL

Purpose Defines the KM-488-DD driver's response to a request for data when none is available.

☞ *NOTE:* KFILL is provided for users who mix File I/O Commands and Calls in the same program. It only affects inputs performed using the File I/O Commands.

Offset 42

Syntax

```
XX   BOARDNR% =
XX   ERRCODE% =
XX   FILLSW% =
XX   CALL KFILL(BOARDNR%, ERRCODE%, FILLSW%)
```

Parameters `BOARDNR%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`FILLSW%` is an integer which represents the fill condition. (The default condition returns a NULL character.) Valid integers are any one of the following:

-1	OFF. The KM-488-DD driver will not return any data characters or a DOS error message, if no input data is available.
-2	ERROR. The KM-488-DD driver will not return any data characters, but will generate an Error Message. The error message number can then be trapped by calling the KSTATUS routine. (Refer to the KSTATUS routine for more information.)
0 - 255	ASCII. The KM-488-DD driver will return the designated ASCII character. (An ASCII Equivalence Chart is provided in Appendix A.) Specify the decimal equivalent of the character to be used.

Returns `ERRCODE%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KFILL (cont'd)

Example

```
770 BRDNUM% =   0  ' First GPIB Board
780 FILLCHAR% = 88 ' Decimal equivalent of "X"
880 ERRNUM% = 0
1000 '  *** Set Fill Character To 'X' ***
1010 CALL KFILL(BRDNUM%, ERRNUM%, FILLCHAR%)
1020 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KHELLO

Purpose Returns an identification string from the KM-488-DD driver.

Offset 45

Syntax

```
XX   BOARDNR% =
XX   ERRCODE% =
XX   IDRESPONSE$ =
XX   CALL KHELLO(BOARDNR%, ERRCODE%, IDRESPONSE$)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **IDRESPONSE$** will contain a string similar to: `copyright (c) 1991 Keithley MetraByte Corp.`

ERRCODE% is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The string into which the HELLO message is returned must be dimensioned prior to calling KHELLO. Otherwise, BASICA assumes that the string of length 0 and calling KHELLO will result in destroying other programmed data. (See the BASIC STRING$ function.)

Example

```
770 BRDNUM% = 0           'First GPIB Board
800 IDSTR$=STRING$(50,0) 'String to hold ID from KHELLO
810 ERRNUM% = 0
1410 'Return ID from driver to verify its presence
1420 CALL KHELLO(BRDNUM%, ERRNUM%, IDSTR$)
1430 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KLOCAL

Purpose Forces the specified bus device(s) to return to being programmed locally from their front panels. This routine can only be used if the KM-488-DD is acting as a System or Active Controller.

Offset 48

Syntax

```
XX   BOARDNR% =
XX   ERRCODE% =
XX   DIM MYGADS%()
XX   CALL KLOCAL(BOARDNR%, ERRCODE%, MYGADS)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

MYGADS (N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is the System Controller and if **MYGADS** does not specify an address, then the GPIB REN (Remote Enable) line is unasserted and all devices are returned to Local. In order to return them to remote mode, it will be necessary to issue a KREMOTE call. Likewise, it will be necessary to issue a KLOL call, if Local Lockout is required.

2. As an Active Controller, the KM-488-DD can issue the GPIB GTL (Go To Local) message to those devices specified by **MYGADS**. In this case, the GPIB REN (Remote Enable) line remains asserted and devices will return to remote when addressed to listen. If a KLOL (Local Lockout) call has been issued previously, it should still be in effect when a device is returned to Remote.

KLOCAL (cont'd)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
500 ERRNUM% = 0
590 DIM GADS%(3) ' addressing array for KLOCAL command
600 GADS%(0) = 12 ' first primary address = 12
610 GADS%(1) = -1 ' first secondary address = NONE
620 GADS%(2) = -1 ' no second device
2000 ' Send Device With GPIB Address Back To Local
2160 CALL KLOCAL(BRDNUM%, ERRNUM%, GADS%(0))
2170 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KLOL

Purpose

This command can only be used if the KM-488-DD is an Active or System Controller. It will disable the GPIB bus devices from being returned to Local Control by means of the Local/Remote button on the device.

Offset

51

☞ ***NOTE:*** This routine issues an IEEE-488 bus signal, LOL. This signal is not supported by all IEEE-488 bus devices.

Syntax

```
XX  BOARDNR% =
XX  ERRCODE% =
XX  CALL KLOL(BOARDNR%, ERRCODE%)
```

Parameters

BOARDNR% is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns

ERRCODE% is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The "LOCAL" button is disabled so that a device cannot be returned to local state for manual programming or control. The Active Controller can return specific devices to "local with lockout state", whereby an operator can then use the front panel controls. When the device is addressed to listen again, it returns to "remote with lockout state". Thus, the effect of the LOL call will remain until the REN line is unasserted (LOCAL) by the System Controller.

2. It is good practice to issue a KLOL so that devices are under strict program control. KLOL can be issued before a device is put in remote and will take effect when the device is set to remote.

Example

```
760 ERRNUM% = 0
770 BRDNUM% =   0  ' First GPIB Board
800'
1000 'Prevent Devices From Being Returned To Local Con-
trol
1010 CALL KLOL(BRDNUM%, ERRNUM%)
1020 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KOUTPUTBUFF

Purpose Transmits data from the KM-488-DD to the GPIB bus from a BASIC array.

Offset 54

Syntax

```
XX BOARDNR% =
XX MODE% =
XX ERRCODE% =
XX DIM DATA%() =
XX COUNT% =
XX DIM MYGADS%()
XX DIM TERMLIST%()
XX CALL KOUTPUTBUFF(BOARDNR%, ERRCODE%, DATA%(0), COUNT%,
MODE%, MYGADS%(0), TERMLIST%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

DATA% is an array of byte data to output.

COUNT% is a long integer representing the number of characters to be output. Valid counts are within the range 1 to 65535 decimal, or from 1 to &HFFFF hex.

MODE% is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

CONT CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

KOUTPUTBUFF (cont'd)

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked.

DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer. (See also the KDMA call description.)

MYGADS(N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

TERMLIST%(3) is an integer array which defines the GPIB bus terminators to be used during the KOUTPUTBUFF call. These terminators are used in conjunction with **COUNT%** to terminate the input process. Terminator choices include whether or not EOI will be asserted to signal the end of input and which character sequence, if any, is to be asserted as an end of message code. The default terminator scheme is initially set up by the *CONFIG.DAT* file and can be modified by calling KTERM and KCONFIG. **TERMLIST** can either specify the use of the default terminators or temporary overrides.

TERMLIST%(3) is comprised of the following elements:

TERMLIST%(0) : **EOI assertion.** Possible values are:

-1(&HFFFF)	Use the default EOI setting.
0	Don't Assert EOI.
1	Assert EOI.

TERMLIST%(1) : **1st GPIB bus terminator.** Possible values are:

-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.

KOUTPUTBUFF (cont'd)

	0 to 255	Send this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
TERMLIST%(2) :	**2nd GPIB bus terminator.** Possible values are:	
	-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
	0 to 255	Send this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default **TERMLIST** can also be used:

```
DIM TERMLIST%(3)
TERMLIST%(0) = -1
TERMLIST%(1) = -1
TERMLIST%(2) = -1
```

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is a peripheral, then an Active Controller must address the KM-488-DD as a talker. In this case, do not specify any addresses using in **MYGADS**. (See section 7.1.).

2. If the KM-488-DD is both the System and Active Controller, and **MYGADS** contains the device(s) to be addressed, the KOUTPUTBUFF will automatically assert the GPIB REN (Remote Enable) line.

KOUTPUTBUFF (cont'd)

3. If the KM-488-DD is the Active Controller and KOUTPUTBUFF is called, it will automatically address itself as the talker and another device(s) contained in **MYGADS** as the listener(s). If the devices have been previously addressed, then **MYGADS** does not have to specify addresses and the bus will not be re-addressed.

4. A non-zero **COUNT%** must always be specified.

5. If you do not wish to append terminator characters, then be sure to set the terminator defaults via *CONFIG.DAT* file or by calling KTERM or KCONFIG and select no terminator overrides in **TERMLIST%**.

6. If a listener does not accept a character, a timeout will occur unless the timeout has been disabled .

7. To manipulate outgoing data, which must be in contiguous bytes within a BASIC integer array, use KSWAP and KPACK.

KOUTPUTBUFF (cont'd)

Example

```
370 BRDNUM% = 0     ' First GPIB Board
470 MODE% = 1       ' DMA mode flag -- 0: no DMA
480                 ' 1: DMA Without CONTINUE
490                 ' 3: DMA With CONTINUE
500 '
590 Address Structure For No Address Can Be Integer
600 NOADR% = -1 ' No first primary address
630 ' Multiple addresses are allowed with KOUTPUTBUFF
640 '
820 ' Terminator Array
830 ERRNUM% = 0
840 DIM TERM%(3) ' KOUTPUTBUFF terminator array
850 TERM%(0) = -1 ' NO EOI OVERRIDE
860 TERM%(1) = -1 ' use default first terminator
870 TERM%(2) = -1 ' use default second terminator
880 '
1240 NUMD% = 1028   'Number of bytes to transfer
1280 DIM DMAOUT%(NUMD%)
1281 'Array to hold DMA output waveform for scope
1300 '
1310 '*** Calculate DMAOUT% ***
1320 '        ...
1330 '
1650 '*** Send DMAOUT% To Previously Addressed Device ***
1680 CALL KOUTPUTBUFF(BRDNUM%, ERRNUM%, DMAOUT%(0),
NUMD%, MODE%, NOADR%, TERM%(0))
1690 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KOUTPUTSTR

Purpose Transmits data from the KM-488-DD to the GPIB bus, from a BASICA string.

Offset 57

Syntax

```
XX  BOARDNR% =
XX  MODE% =
XX  ERRCODE% =
XX  DIM TERMLIST%()
XX  DATA$ =
XX  COUNT% =
XX  DIM MYGADS%()
XX  CALL KOUTPUTSTR(BOARDNR%, ERRCODE%, DATA$, COUNT%,
MODE%, MYGADS%(0), TERMLIST%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

DATA$ is a string of data to output.

COUNT% is an integer representing the number of characters to be output. Valid counts are within the range 1 to 65535 decimal, or from 1 to &HFFFF hex. Note, however, that the limit for a BASICA string is 256 bytes.

MODE% is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

CONT CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

KOUTPUTSTR (cont'd)

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked.

DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer. (See KDMA for more information.)

MYGADS (N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

TERMLIST% (3) is an integer array which defines the GPIB bus terminators to be used during the KOUTPUTBUFF call. These terminators are used in conjunction with **COUNT%** to terminate the input process. Terminator choices include whether or not EOI will be asserted to signal the end of input and which character sequence, if any, is to be asserted as an end of message code. The default terminator scheme is initially set up by the *CONFIG.DAT* file and can be modified by calling KTERM and KCONFIG. **TERMLIST** can either specify the use of the default terminators or temporary overrides.

TERMLIST% (3) is comprised of the following elements:

TERMLIST% (0) : **EOI detection.** Possible values are:

-1(&HFFFF)	Use the default EOI setting.
0	Don't Assert EOI.
1	Assert EOI.

TERMLIST (1) : **1st GPIB bus terminator.** Possible values are:

-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.

KOUTPUTSTR (cont'd)

	0 to 255	Send this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
TERMLIST(2):	**2nd GPIB bus terminator.** Possible values are:	
	-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
	0 to 255	Send this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default **TERMLIST** can also be used:

```
DIM TERMLIST%(3)
TERMLIST%(0) = -1
TERMLIST%(1) = -1
TERMLIST%(2) = -1
```

Returns

ERRCODE% is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is a peripheral, then an Active Controller must address the KM-488-DD as a talker and some other device(s) as the listener(s). In this case, do not specify addresses in **MYGADS**. (See section 7.1.).

2. If the KM-488-DD is both the System and Active Controller, and **MYGADS** contains the device(s) to be addressed, the KOUTPUTSTR will automatically assert the GPIB REN (Remote Enable) line.

KOUTPUTSTR (cont'd)

3. If the KM-488-DD is the Active Controller and KOUTPUTSTR is called, it will automatically address itself as the talker and another device(s) contained in **MYGADS** as the listener(s). If the devices have been previously addressed, then **MYGADS** can specify no addresses and the bus will not be re-addressed.

4. A non-zero **COUNT%** must always be specified.

5. If you do not wish to append terminator characters, then be sure to set the terminator defaults via *CONFIG.DAT* or by calling KTERM or KCONFIG and select no termiantor overrides in the **TERMLIST%** array.

6. If a listener does not accept a character, a timeout will occur unless the timeout has been disabled.

KOUTPUTSTR (cont'd)

Example

```
300 ERRNUM% = 0
370 BRDNUM% = 0 ' First GPIB Board
470 MODE% = 0 ' DMA mode flag -- 0: no DMA
480           '                  1: DMA Without CONTINUE
490           '                  3: DMA With CONTINUE
590 DIM GADS%(3) ' addressing array for ENTER command
600 GADS%(0) = 05 ' first primary address = 5
610 GADS%(1) = -1 ' first secondary address = NONE
620 GADS%(2) = -1 ' no more devices
640 'Multiple addresses are allowed with KOUTPUTSTR
760 S$ = STRING$(100,32)
761 'must initialize string before call
770 '
820 ' Terminator Arrays
830 '
840 DIM TERM%(3) ' KOUTPUTSTR terminator array
850 TERM%(0) = -1 ' NO EOI OVERRIDE
860 TERM%(1) = &HD ' first terminator = CR
870 TERM%(2) = &HA ' second terminator = LF
880 '
1650 '*** Send Message To Device At GPIB Address 5
1660 S$="MESS 10:"+CHR$(34)+"  INITIALIZE SCOPE"+CHR$(34)
1670 COUNT%=LEN(S$)
1680 CALL KOUTPUTSTR(BRDNUM%, ERRNUM%, S$, COUNT%, MODE%,
GADS%(0), TERM%(0))
1690 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KPACK

Purpose This packs discrete 8-bit data values (stored in a 16-bit integer array) into sequential bytes. This routine is useful when an instrument can only accept an 8-bit data value, because the KM-488-DD driver can only transfer contiguous bytes.

Syntax

```
XX  DIM SOURCE%()
XX  DIM DEST%()
XX  COUNT% =
XX  CALL KPACK(SOURCE%(0),DEST%(0),COUNT%)
```

Offset 60

Parameters **SOURCE%** is the name of the integer array which contains the data to be converted.

DEST% is the name of the integer array in which the converted data is to be stored.

COUNT% is an integer which represents the number of elements comprising the **SOURCE%** array. (The number of elements needed in the **DEST%** array is **COUNT%/2**.)

Returns Result is stored in **DEST%**.

Programming Notes

1. **COUNT%** cannot be 0.

KPACK (cont'd)

Example

```
1240 NUMD% = 1024    'Number of bytes to KPACK
1250 '
1260 'Use arrays of integers (2 bytes) in BASICA
1270 '
1280 DIM OUTARRAY%(NUMD%) ' Data To Send To Device
1290 DIM WORKB%(NUMD%/2)  ' Temporary Work Buffer
1300 '
1310 ' ....  Code To Fill OUTARRAY%() Here ....
1320 '
1330 ' Pack OUTARRAY%() Into Contiguous Bytes In
1340 ' WORKB%() And KOUTPUTBUFF WORKB%() Later
1350 '
1370 CALL KPACK(OUTARRAY%(0), WORKB%(0), NUMD%)
1380 '
1390 ' ....  Code To KOUTPUTBUFF WORKB%() Here ....
```

KPASCTL

Purpose Designates another controller to be the Active Controller. The KM-488-DD must be an Active Controller.

Offset 63

Syntax

```
XX  BOARDNR% =
XX  ERRCODE% =
XX  DIM MYGADS%()
XX  CALL KPASCTL(BOARDNR%, ERRCODE%, MYGADS%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

MYGADS (N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (n can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the KM-488-DD which is relinquishing its position as the Active Controller is also a System Controller, it retains its status as System Controller.

KPASCTL (cont'd)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
380 ERRNUM% = 0
960 DIM PERIADS%(3) ' addressing array for PERIPHERAL
970 PERIADS%(0) = 2 ' first primary address = 2
980 PERIADS%(1) = -1 ' first secondary address = NONE
990 PERIADS%(2) = -1 ' no more devices
2400 ' Pass Control To Device At GPIB Address 2
2410 CALL KPASCTL(BRDNUM%, ERRNUM%, PERIADS%(0))
2420 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KPPOLL

Purpose Initiates a Parallel Poll. The KM-488-DD must be the Active Controller.

Offset 66

☞ **NOTE:** Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax

```
XX  BOARDNR% =
XX  ERRCODE% =
XX  PPRTN% =
XX  CALL KPPOLL(BOARDNR%, ERRCODE%, PPRTN%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

PPRTN% is an integer variable which will contain an integer in the range 0 to 255 decimal. This integer indicates which Data Lines which have been asserted (DIO1-DIO8).

Programming Notes None.

Example

```
370 BRDNUM% = 0 ' First GPIB Board
1500 ERRNUM% = 0
1510 ' Conduct Parallel Poll
1570 CALL KPPOLL(BRDNUM%, ERRNUM%, PPRTN%)
1580 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
1590 '
1600 PRINT
1610 PRINT "The Parallel Poll Returned = ";PPRTN%
```

KPPC

Purpose Configures the Parallel Poll response of a GPIB bus device. The KM-488-DD must be the Active Controller.

Offset 69

☞ *NOTE:* Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax

```
XX  BOARDNR% =
XX  ERRCODE% =
XX  PPCCFG% =
XX  DIM MYGADS%()
XX  CALL KPPOLL(BOARDNR%, ERRCODE%, MYGADS%(0), PPCCFG%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

MYGADS (N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

PPCCFG% is an integer which represents the Parallel Poll Response of the device to be programmed. This integer is of the format:

Bit	7	6	5	4	3	2	1	0
	0	1	1	0	S	P2	P1	P0

KPPC (cont'd)

Where:

S	is the parallel poll response value (0 or 1) that the device uses to respond to the parallel poll when service is required. This bit is generally set to 1.

☞ ***NOTE:*** This value must correspond to the setting of the GPIB bus device's *ist* (individual status) bit. Refer to the Device's documentation for more information.

P2 P1 P0	is a 3-bit value which tells the device being configured which data bit (DIO1 through DIO8) it should use as its parallel poll response.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPC (cont'd)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
380 ERRNUM% = 0
500 DIM GADS%(3) ' addressing array
510 GADS%(0) = 1 ' first primary address = 1
520 GADS%(1) = -1 ' first secondary address = NONE
530 GADS%(2) = -1 ' no more devices
1240 'Configure Device At GPIB Address 1 To Respond
1241 'On DIO3 Line
1250 PPCONFIG%=&HA
1260 CALL KPPC(BRDNUM%, ERRNUM%, GADS%(0), PPCONFIG%)
1270 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KPPD

Purpose Disables the Parallel Poll response capability of the specified GPIB bus device(s). The KM-488-DD must be the Active Controller.

Offset 72

Syntax

```
xx  BOARDNR% =
xx  ERRCODE% =
xx  DIM MYGADS%()
xx  CALL KPPD (BOARDNR%, ERRCODE%, MYGADS%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

MYGADS(N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPD (cont'd)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
380 ERRNUM% = 0
500 DIM GADS%(3) ' addressing array
510 GADS%(0) = 1 ' first primary address = 1
520 GADS%(1) = -1 ' first secondary address = NONE
530 GADS%(2) = -1 ' no more devices
1240 ' Disable Device At GPIB Address 1
1250 'From Responding To KPPOLL
1260 CALL KPPD(BRDNUM%, ERRNUM%, GADS%(0))
```

KPPU

Purpose Disables the Parallel Poll Response of all GPIB bus device(s). The KM-488-DD must be the Active Controller.

Offset 75

Syntax

```
xx BOARDNR% =
xx ERRCODE% =
xx CALL KPPU(BOARDNR%, ERRCODE%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
370 BRDNUM% = 0 ' First GPIB Board
1500 ERRNUM% = 0
1510 'Disable Parallel Poll Response Of All Devices
1570 CALL KPPU(BRDNUM%, ERRNUM%)
1580 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KQUIKSTAT

Purpose Returns the KM-488-DD's status byte.

Offset 78

Syntax

```
xx  BOARDNR% =
xx  ERRCODE% =
xx  QSTAT% =
xx  CALL KQUIKSTAT(BOARDNR%, ERRCODE%, QSTAT%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **QSTAT%** is a pointer to the returned status integer. The status integer is of the following format:

Bit	15-9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PEND	DET	SRQ	DEC	LA	TA	DMA

Where:

ADSC	**Address Status Change.** If this bit is set to 1, a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).
CIC	**Controller in Charge.** If this bit is set to 1, the KM-488-DD is an Active Controller.
PEND	**SRQ Pending.** If this bit is set to 1, the KM-488-DD has an SRQ request pending.
DET	**Device Triggered.** If this bit is set to 1, a GPIB Trigger command has been received.
SRQ	**Internal SRQ.** If the KM-488-DD is an Active Controller and this bit is set to one, a device is requesting service. If the KM-488-DD is a Peripheral and this bit is set to one, then its SRQ (issued by KREQUEST) has not been serviced.
DEC	**Device Cleared.** If this bit is set to one, the KM-488-DD has received a GPIB Clear command.

KQUIKSTAT(cont'd)

LA	**Listen Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Talker and can output data to the GPIB.
DMA	**DMA.** A DMA transfer is currently in progress.

ERRCODE% is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
370 BRDNUM% = 0 ' First GPIB Board
1500 ERRNUM% = 0
1510 QSTATUS% = 0 ' *** Return Status Word ***
1570 CALL KQUIKSTAT(BRDNUM%, ERRNUM%, QSTATUS%)
1580 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
1590 '
1600 RESULT% = QSTATUS% AND &H10
1610 IF RESULT% = 0 THEN PRINT "NO SRQ DETECTED" ELSE
PRINT "SRQ"
```

KREMOTE

Purpose Forces the GPIB bus device(s) to the remote mode.

☞ ***NOTE:*** The KM-488-DD must be a System Controller to execute this command.

Offset 81

Syntax

```
xx  BOARDNR% =
xx  ERRCODE% =
xx  DIM MYGADS%()
xx  CALL KREMOTE(BOARDNR%, ERRCODE%, MYGADS%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

MYGADS (N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KREMOTE (cont')

Programming Notes

1. If **MYGADS** does not specify an address, then the GPIB REN (Remote Enable) line is asserted. Devices will not be in the remote mode until addressed to listen. If **MYGADS** contains address then those devices will also be addressed to listen, so they will be put in remote.

2. It is good programming practice to issue a Local Lockout to prevent the device(s) from being returned to Local mode.

Example

```
370 BRDNUM% = 0 ' First GPIB Board
500 ERRNUM% = 0
590 DIM GADS%(3) ' addressing array for KREMOTE command
600 GADS%(0) = 12 ' first primary address = 12
610 GADS%(1) = -1 ' first secondary address = NONE
620 GADS%(2) = -1 ' no more device
2000 ' Assert REN and Address Device At GPIB Address 12
To Listen
2160 CALL KREMOTE(BRDNUM%, ERRNUM%, GADS%(0))
2170 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KREQUEST

Purpose Sets the Serial Poll Response of a KM-488-DD which is a Peripheral.

Offset 84

Syntax

```
XX  BOARDNR% =
XX  ERRCODE% =
XX  SP% =
XX  CALL KREQUEST(BOARDNR%, ERRCODE%, SP%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

SP% is an integer in the range 0 to 255 which represents the serial poll response of the device. This integer is of the following format:

Bit	7	6	5	4	3	2	1	0
	DIO8	rsv	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
rsv	If this bit is 1, the KM-488-DD will generate a Service Request (assert SRQ).

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The Active Controller can read the value of **SP%** by serial polling the KM-488-DD. This will clear bit 6, if it was set.
2. Use KQUIKSTAT or KSPOLL to check if the Peripheral has been serial polled (checks the status of the SRQ bit).

KREQUEST (cont'd)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
510 ERRNUM% = 0
3100 'Set RSV. Don't Set Any Other Bits In Status Byte
3110 RSV%=64
3120 CALL KREQUEST(BRDNUM%, ERRNUM%, RSV%)
3230 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KRESET

Purpose Performs a "warm" reset of the KM-488-DD and the GPIB bus.

Offset 87

Syntax

```
xx  BOARDNR%=
xx  ERRCODE%=
xx  CALL KRESET(BOARDNR%, ERRCODE%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The KM-488-DD is reset to the following conditions:

   - STOP
   - DISARM
   - CONFIG
   - ABORT
   - BUFFER INCREMENT
   - DMA SINGLE
   - ERROR ON
   - FILL NULL
   - LOCAL
   - REQUEST 0 (If Peripheral)
   - Clear CHANGE, TRIGGER, and CLEAR STATUS
   - TIMEOUT 10

   It also clears all error conditions.

2. If the KM-488-DD is the System Controller, it will assert the GPIB IFC (Interface Clear) line for at least 500 µsecs.

Example

```
370 BRDNUM% = 0 ' First GPIB Board
510 ERRNUM% = 0
3100 ' *** Reset Driver ***
3120 CALL KRESET(BRDNUM%, ERRNUM%)
3230 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KRESUME

Purpose Initiates data transfers between two non-Active Controller GPIB devices, by unasserting the ATN line. <u>The KM-488-DD must be an Active Controller in order to use this function.</u>

Offset 90

Syntax

```
xx BOARDNR% =
xx ERRCODE% =
xx CALL KRESUME(BOARDNR%, ERRCODE%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes This routine is normally called after a KSEND has addressed a talker and a listener. (See KSEND description.)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
510 ERRNUM% = 0
3100 ' *** Drop ATN Line After A Send Command ***
3120 CALL KRESUME(BRDNUM%, ERRNUM%)
3230 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KSEND

Purpose Sends GPIB commands and data from a string.

Offset 93

Syntax

```
XX BOARDNR% =
XX ERRCODE% =
XX STRINGLEN% =
XX SENDSTR$ =
XX CALL KSEND(BOARDNR%, ERRCODE%, SENDSTR$, STRINGLEN%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

SENDSTR$ is a string which includes any of the following subcommands: CMD, UNT, MTA, MLA, TALK, LISTEN, DATA, and EOI. Multiple subcommands may be specified; however, the length of the entire SEND command string cannot exceed 255 characters. The subcommands are as follows:

CMD - Sends information with the ATN line asserted. This indicates to the bus devices that the characters are to be interpreted as GPIB bus commands. (See Appendix B.) The command may be sent as either a quoted string (e.g., **'COMMAND'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **&H0D** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **CMD 67,79,77,77,65,78,68**).

An EOI cannot be sent with this subcommand, because an EOI with ATN asserted would initiate a Parallel Poll. (Use EOI directly in **SENDSTR$**.)

UNT - Untalks all devices by sending a GPIB UNTALK command with ATN asserted.

UNL - Unlistens all devices by sending the GPIB UNLISTEN command with ATN asserted.

KSEND (cont'd)

MTA - Designates the KM-488-DD to address itself to talk by sending the GPIB MTA (My Talk Address) command with the ATN line asserted.

MLA - Designates the KM-488-DD to address itself to listen by sending the GPIB MLA (My Listen Address) command with the ATN line asserted.

TALK addr - Addresses another GPIB device or KM-488-DD to talk by sending a GPIB TAG (Talk Address Group) command with the ATN line asserted. **Addr** is an integer representing the GPIB BUS device address of the device to talk. This integer ranges from 00 to 30 decimal (addresses less than 10 must have a leading 0). A secondary address may be appended to the primary address.

LISTEN addr - Addresses another GPIB device(s) or KM-488-DD('s) to listen by sending a GPIB LAG (Listen Address Group) command with ATN asserted. **Addr** is an integer representing the GPIB BUS device address of the device(s) to talk. This integer ranges from 00 to 30 decimal. (addresses less than 10 must have a leading 0). A secondary address may be appended to the primary address. Multiple listeners can be specified.

DATA - Sends information with the ATN line unasserted. This indicates to the bus devices that the characters are to be interpreted as data. This is the technique which is to be used to send device-dependent commands. (See the IEEE-488 Tutorial in Appendix C for more information.) The data may be sent as either a quoted string (i.e., **'DATA'**) or as a numeric equivalent of an individual ASCII character

KSEND (cont'd)

(i.e., **13** decimal or **&H0D** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **DATA 68,65,84,65**).

This sub-command is useful when you are sending commands which are unique to your interface.

EOI - Sends information with the ATN line unasserted. EOI will be asserted when the last character is transmitted. This information is interpreted as GPIB bus data and may be sent as either a quoted string (e.g., **'xyz'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **&H0D** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **EOI 120,121,122**).

STRINGLEN% is an integer between 0 and 255 which represents the length of the SEND string.

Returns

ERRCODE% is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Typically, KSEND is used to send commands to initiate communication directly between peripheral devices. For example, the KM-488-DD driver might talk a scope and send a trace to a listened plotter.

2. The maximum length of the KSEND command, including any subcommands, is 255 characters. To KSEND large amounts of data, use multiple KSEND commands.

3. KSEND should only be used when a non-conforming device requires a special command sequence or a non-standard GPIB command. Do not use the KSEND command unless you are extremely familiar with GPIB.

4. The KM-488-DD must be the Active Controller to KSEND commands. Any KM-488-DD can KSEND data.

KSEND (cont'd)

5. If a DATA subcommand is not included in the KSEND string, be sure to call KRESUME immediately after the KSEND. This is necessary because the ATN line must be dropped so that the transfer will proceed.

6. Do not include the word SEND within the KSEND string as you might do in the FILE I/O SEND command.

Example

```
370 BRDNUM% = 0 ' First GPIB Board
510 ERRNUM% = 0
2250 'Address The Device At Address 2 To Listen
2251 'And The Device At 12 to talk
2260 S$= "UNT UNL LISTEN 02 TALK 12"
2270 COUNT%=LEN(S$)
2280 CALL KSEND(BRDNUM%, ERRNUM%, S$, COUNT%)
2290 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
2300 '
2310 ' *** Drop ATN Line After A Send Command ***
2320 CALL KRESUME(BRDNUM%, ERRNUM%)
2330 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KSPOLL

Purpose If the KM-488-DD is an Active Controller, KSPOLL will check for an SRQ or conduct a serial poll. If the KM-488-DD is a Peripheral, KSPOLL will check if the KM-488-DD's SRQ has been serviced.

Offset 96

Syntax

```
XX BOARDNR% =
XX ERRCODE% =
XX DIM SPR%()
XX DIM MYGADS(N*2)
XX CALL KSPOLL(BOARDNR%, ERRCODE%, SPR%, MYGADS%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

MYGADS(N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

SPR% is an integer or an array of unsigned integers with an entry for each address in **MYGADS**. The value of the integer is 0 to 255 and corresponds to a byte with the following bitmap:

Bit	7	6	5	4	3	2	1	0
	DIO8	SRQ	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

KSPOLL (cont'd)

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
SRQ	If this bit is 1, the SRQ (Service Request) line has been asserted to request servicing. Otherwise, SRQ has not been asserted by this device.

Bit 6 has the special significance of indicating an SRQ pending. The significance of the other bits will be device-dependent.

Programming Notes

1. The most common use of KSPOLL is for the KM-488-DD, as an active controller, to issue SPOLL with **MYGADS%** containing the address of a single device. This addresses and serial polls the device and, upon return, **SPR%** contains the serial poll response of the polled device. If the SRQ bit in **SPR%** is set, the device had issued an SRQ. The other bits in **SPR%** are device-dependent. Serial polling a device which is currently asserting SRQ will cause the device to unassert its SRQ.

   The string returned by KSTATUS or the integer status word set by KQUIKSTAT can be checked to determine the presence of an SRQ before a serial poll is conducted. If only one device is asserting SRQ, the effect of issuing KSPOLL will be clear to the internal SRQ pending "bit", even if the polled device is not issuing the SRQ. It is also the case when multiple devices are asserting SRQ and device not currently asserting SRQ is polled. In this case, a subsequent use of KSTATUS or KQUIKSTAT will not reveal a pending SRQ.

   To aid in identifying which bus device(s) is currently requesting service, a KM-488-DD which is the Active Controller, can serial poll as many as 14 devices by issuing a KSPOLL call with **MYGADS%** containing the address of more than one device. In this case, **SPR%** should be an array of unsigned integers with an element for each address. In this case, KSPOLL will serial poll each addressed device and return the serial poll bytes in the **SPR%** array. This is a faster way for discerning the source of an SRQ among several devices.

KSPOLL (cont'd)

2. If the KM-488-DD is a peripheral and KSPOLL is called with **MYGADS%** having no addresses, **SPR%** will be the serial poll response (**SP%**) set by KREQUEST - with the possible exception of the SRQ bit, which may have disappeared as a result of an Active Controller having serial polled the KM-488-DD since the last KREQUEST call. In the instance of an Active Controller serial polling a peripheral KM-488-DD, the peripheral KM-488-DD unasserts the SRQ line. For an example, see KREQUEST. The state of the SRQ bit can also be determined from the string returned by KSTATUS or the integer status word returned by KQUIKSTAT.

3. If the KM-488-DD is the Active Controller and issues KSPOLL with **MYGADS%** containing no addresses, **SPR%** will be 64 if a device is asserting SRQ and 0 if not. This same result could have been determined from KQUIKSTAT or KSTATUS.

   Note that if this call is issued when a SRQ is pending, it internally "clears" the SRQ "pending" bit, even though the requesting device has not been polled and is still issuing an SRQ. Thus, a subsequent KSPOLL, KQUIKSTAT, or KSTATUS call would not reveal a pending SRQ.

Example

```
350 DIM SBYT%(2) ' Array To Receive SPOLL Bytes
360 DIM GADS%(5) ' Address Array for two devices
370 BRDNUM% = 0 ' First GPIB Board
380 ERRNUM% = 0
900 GADS%(0) = 5 ' first primary address = 5
910 GADS%(1) = -1 ' first secondary address = NONE
920 GADS%(2) = 7 ' second primary address = 7
930 GADS%(3) = -1 ' second secondary address = NONE
940 GADS%(2) = -1 ' no third device
950 '
960 ' KSPOLL Both Devices In GADS%()
970 '
2950 CALL KSPOLL(BRDNUM%, ERRNUM%, GADS%(0), SBYT%(0))
2970 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
2980 '
2990 ' Test Device Dependent SBYT%() Results Here ...
```

KSTATUS

Purpose Returns a character string describing the current operating state of the KM-488-DD.

Offset 99

Syntax

```
XX  BOARDNR% =
XX  ERRCODE% =
XX  STAT$ =
XX  CALL KSTATUS(BOARDNR%, ERRCODE%, STAT$)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

STAT$** is a character string which describes the current operating state of the KM-488-DD. The string is of the format shown in Table 7-2. **STAT$ should be 75 bytes in order to contain the longest message that might be returned. If it is too short, data may be written in other regions of memory. You can create **STAT$** prior to a call via the BASICA STRING$ function.

Table 7-2. Status String

Starting Col	# of Cols.	Name and Description

1	1	Operating Mode - Tells if the KM-488-DD is acting as an *Active Controller* or *Peripheral*. Can be C or P where:
		
		C = The KM-488-DD is an *Active Controller*
		P = The KM-488-DD is a *Peripheral*

3	2	Bus Address - Gives the IEEE-488 Bus Address of the KM-488-DD. This is a two-digit decimal integer ranging from 00 to 30.
6	2	Addressed Status Change - Indicates if the device has changed addressed state, i.e., if it has cycled between the *Talker, Listener,* or *Active Controller* states. This is reset whenever a STATUS is read. This can be one of the following: G0 = There has not been a change in the addressed status. G1 = There has been a change in the addressed status.

9	1	Current Addressed State - Indicates if the KM-488-DD is currently acting as a *Talker* or *Listener*, or is *Idle*. Can be T, L, or I, where:
		
		T = The KM-488-DD is a Talker
		L = The KM-488-DD is a Listener
		I = The KM-488-DD is Idle

11	2	Service Request - Represents the current internal SRQ status where: S0 = SRQ not asserted S1 = SRQ asserted If the KM-488-DD is in the *Active Controller* mode, the internal SRQ state indicates if a device is asserting SRQ. If the KM-488-DD is acting as a *Peripheral*, the internal SRQ state is set by using the KREQUEST call. It is cleared by a serial poll from the Active Controller and indicates it is asserting SRQ.

KSTATUS (cont'd)

Table 7-2. Status String

Starting Col	# of Cols.	Name and Description
14	3	Error Code - Indicates whether or not an error has occurred. The three characters consist of the letter 'E' followed by a two digit error code. (A list of these error codes is provided in Appendix G.) An error code of 00 indicates no error has occurred.
18	2	Triggered State - Indicates if a IEEE-488 trigger command has been received or not. This is not updated during DMA CONTINUE transfers. Triggered State values are: T0 = No trigger command has been received. T1 = A trigger command has been received.
21	2	Cleared - Indicates if the IEEE Clear command has been received or not. This is not updated during DMA CONTINUE transfers. Cleared values are: C0 = No Clear command has been received. C1 = A Clear command has been received.

24	2	Transfer Status - Indicates if a data transfer is in progress. Values which may appear are:
		
		P0 = No transfer
		P1 = DMA CONTINUE Transfer occurring
		P2 = DMA AUTOINITIALIZE Transfer occurring

27	up to 45	Error Message Text - Contains the Error Message Text associated with the given error code (EXX). These error messages are listed in Appendix G.

Programming Notes If the KM-488-DD is the Active Controller, the Service Request Status may be cleared (i.e, S0) as a result of a KSPOLL, even when the SRQ line is still asserted. See the discussion in KSPOLL.

KSTATUS (cont'd)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
380 STAT$=STRING$(75,32)
390 ERRNUM% = 0
3950 '*** Loop and check status for SRQ ***
3960 '
3970 CALL KSTATUS(BRDNUM%, ERRNUM%, STAT$)
3971 ' Request status
3980 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
3990 IF (MID$(STAT$,12,1) <> "1") GOTO 3970
```

KSTOP

Purpose Stops a DMA CONTINUE transfer.

Syntax

```
xx  BOARDNR% =
xx  ERRCODE% =
xx  CALL KSTOP  (BOARDNR%, ERRCODE%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the DMA CONTINUE transfer has been completed, a KSTOP will have no effect.
2. Use the KBUFD call to determine the actual number of characters which were transferred before the KSTOP took effect.

Example

```
370 BRDNUM% = 0 ' First GPIB Board
510 '
3100 ' *** Stop DMA ***
3120 CALL KSTOP(BRDNUM%, ERRNUM%)
3230 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KSWAP

Purpose Exchanges the bytes contained in the elements of an integer array.

Offset 105

Syntax

```
XX  DIM ARRAY%()
XX  COUNT% =
XX  CALL KSWAP(ARRAY%(0),COUNT%)
```

Parameters **ARRAY%** is the name of the integer array whose bytes are to be "swapped".

COUNT% is an integer representing the number of elements contained in the integer array (**ARRAY%**).

Returns **ARRAY%** will contain the newly formatted integer array.

Programming Notes

1. This routine exchanges the bytes within individual elements of an integer array. Byte swaps are necessary when communicating with devices that transfer binary data (real or integer numbers) in high byte-low byte order.

   When such a device transfers data into a BASICA integer array, your program must swap the bytes in each array location, as BASICA will expect the bytes to be in low byte-high byte order in memory. Likewise, you must KSWAP data in an integer array that is being sent to a device expecting high byte-low byte data. The KM-488-DD driver is unable to perform the swap during a transfer, as it treats PC memory as a linear sequence of bytes.

KSWAP (cont'd)

Example

```
1240 DIM WORKB%(100)
1250 COUNT% = 101    ' WORKB%(100) HAS 101 ELEMENTS
1260 '
1270 ' FILL WORKB%() WITH A SINE WAVE
1280 '
1290 FOR X = 0 TO 100
1300 WORKB%(X) = 10 * SIN(X)
1310 NEXT X
1320 '
1330 ' KSWAP DATA FOR A DEVICE EXPECTING HB/LB
1340 '
1350 CALL KSWAP(WORKB%(0), COUNT%)
1360 '
1370 ' ... OUTPUT DATA TO DEVICE HERE ...
```

KTERM

Purpose Changes the default GPIB bus terminator(s) to be used in ENTER and/or OUTPUT calls.

Offset 108

Syntax

```
xx  BOARDNR% =
xx  ERRCODE% =
xx  DIM TERMLIST%()
xx  CALL KTERM(BOARDNR%, ERRCODE%, TERMLIST%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

TERMLIST%() is an integer array which defines the GPIB bus terminators. See Chapter 6 for more information regarding the use of GPIB bus terminators. **TERMLIST%()** is comprised of the following elements:

TERMLIST%(0) : **Input/Output GPIB Terminators.** Possible values are:

-1	Change the Input Terminators.
0	Change both Input and Output Terminators.
1	Change the Output Terminators.

TERMLIST%(1) : **EOI detection.** Possible values are:

0	No EOI setting.
1	Assert EOI.

KTERM (cont'd)

TERMLIST%(2) : **1st GPIB bus terminator.** Possible values are:

-1	Don't use first or second GPIB bus terminator.
0 to 255	Use this ASCII character as the first GPIB bus terminator instead of the default first GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

TERMLIST%(3) : **2nd GPIB bus terminator.** Possible values are:

-1	No default second GPIB bus terminator.
0 to 255	Use this ASCII character as the second GPIB bus terminator instead of the default second GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. KTERM will change the default bus terminators set during software installation, as directed by **TERMLIST%()**.

KTERM (cont'd)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
510 ERRNUM% = 0
1010 ' Terminator Arrays
1020 '
1030 DIM DRVTERM%(4)
1031 'used by KTERM to initialize driver terminations
1040 DRVTERM%(0)=0
1041 ' set both input & output default terminators
1050 DRVTERM%(1)= 1 ' Assert/Detect EOI
1060 DRVTERM%(2)=-1 ' no first terminator character
1070 DRVTERM%(3)=-1 ' no second terminator character
1080 '
1540 ' *** Initialize Driver Terminators ***
1550 '
1560 CALL KTERM(BRDNUM%, ERRNUM%, DRVTERM%(0))
1570 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KTO

Purpose Changes the timeout period.

Offset 111

Syntax

```
XX BOARDNR% =
XX ERRCODE% =
XX TVAL% =
XX CALL KTO(BOARDNR%, ERRCODE%, TVAL%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

TVAL% is the number of 55 ms. timer ticks to allow before a time-out.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. To suppress Timeout checking, set **TVAL%** to 0.

2. If a DMA CONTINUE transfer is in progress, the KM-488-DD will check for timeouts only for the first byte that is transmitted/received. During other types of transfers, the KM-488-DD will check for a timeout between transmission of bytes.

   You must be certain to check that a DMA CONTINUE transfer has been completed.

3. Because BASIC has no "long" integer type, the maximum timeout available is approximately (65535/18.3) seconds.

Example

```
370 BRDNUM% = 0 ' First GPIB Board
510 ERRNUM% = 0
1480 'Set time out to 15 seconds
1490 '
1500 TIMEO%=15000\55   'Need Number Of Ticks With
55ms/tick
1510 CALL KTO(BRDNUM%, ERRNUM%, TIMEO%)
1520 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KTRIGGER

Purpose Triggers the specified device(s). The KM-488-DD must be an Active Controller.

Offset 114

Syntax

```
xx  BOARDNR% =
xx  ERRCODE% =
xx  DIM MYGADS%()
xx  CALL KTRIGGER(BOARDNR%, ERRCODE%, MYGADS%(0))
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

MYGADS (N*2) represents the GPIB bus address of the device to be accessed. **MYGADS** is a pointer to an array of **n** (**n** can range from 1 to 15) pairs of GPIB addresses. You must tailor **MYGADS** to your application. The **MYGADS** structure is described in section 7.1.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The devices listed in **MYGADS** will be triggered.

KTRIGGER (cont'd)

Example

```
370 BRDNUM% = 0 ' First GPIB Board
500 ERRNUM% = 0
590 DIM GADS%(3) ' addressing array for KTRIGGER command
600 GADS%(0) = 12 ' first primary address = 12
610 GADS%(1) = -1 ' first secondary address = NONE
620 GADS%(2) = -1 'no more devices
2000 '*** Trigger Device With GPIB Address 12 ***
2160 CALL KTRIGGER(BRDNUM%, ERRNUM%, GADS%(0))
2170 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

KUNPACK

Purpose Unpacks consecutive byte values into a BASIC integer array.

Offset 117

Syntax

```
XX  DIM SOURCE%(N)
XX  DIM DEST%(2*N)
XX  COUNT% =
XX  CALL KUNPACK (SOURCE%(0),DEST%(0),COUNT%)
```

Parameters **SOURCE%** is the name of the integer array which contains the data to be converted.

DEST% is the name of the integer array in which the converted data is to be stored. (It must be twice the size of **SOURCE%**.)

COUNT% is an integer which represents the number of bytes to unpack from the **SOURCE%** array. (The number of elements needed in the **DEST%** array in **COUNT%/2**.)

Returns **dest%** will contain the newly converted data.

Programming Notes None.

Example

```
1240 NUMD% = 1028 'Number of bytes to transfer
1250 '
1260 'We can only use arrays of integers (2 bytes)
1270 'in BASICA
1280 DIM DMAOUT%(NUMD%/2)
1281 'Array to hold DMA output waveform from scope
1300 '
1390 DIM WORKB%(NUMD%)
1391 'Working buffer for manipulating curve data
1400 '
3450 'Convert the byte array to word (integer)
3460 '
3470 CALL KUNPACK(DMAOUT%(0), WORKB%(0), NUMD%)
```

KWAITC

Purpose Forces the KM-488-DD driver to wait until a DMA CONTINUE transfer has been completed before returning control to the application program.

Offset 120

Syntax

```
xx BOARDNR% =
xx ERRCODE% =
xx CALL KWAITC(BOARDNR%, ERRCODE%)
```

Parameters **BOARDNR%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **ERRCODE%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Time out checking, if enabled, is performed while WAITing.

2. This is an explicit wait. The following calls perform an implicit wait:

   - KABORT
   - KCLEAR
   - KENTER
   - KLOCAL
   - KLOCAL LOCKOUT
   - KOUTPUT
   - KPASS CONTROL
   - KPPOLL
   - KPPOLL CONFIG
   - KPPOLL DISABLE
   - KPPOLL UNCONFIG
   - KREMOTE
   - KREQUEST
   - KRESUME
   - KSEND
   - KSPOLL
   - KTRIGGER

Example

```
370 BRDNUM% = 0 ' First GPIB Board
500 ERRNUM% = 0
900 ... START UP DMA CONTINUE HERE...
2100 ' *** Wait For DMA CONTINUE To Finish ***
2160 CALL KWAITC(BRDNUM%, ERRNUM%)
2170 IF ERRNUM% THEN PRINT "IEEE Error ";ERRNUM%:STOP
```

This page intentionally left blank.

CHAPTER 8
USING THE CALL INTERFACE IN QUICKBASIC

Versions Supported

QuickBASIC 4.0 and higher

File Header

Be sure to include the following line within your program:

```
'$INCLUDE:'HEADER.BI'
```

The inclusion of this file allows QuickBASIC to check that the correct number and type of parameters are specified for each routine called.

Compiling The Program

Once your QuickBASIC application program has been written, you may want to compile the program. Be sure to include full path names to the various library files where needed.

From within the QuickBASIC Environment

Be sure that the appropriate *.QLB* file (*KM488QB4.QLB* or *KM488QB7.QLB*) is located where QuickBASIC can find it. Then, invoke QuickBASIC by typing:

For QuickBASIC 4.x	For QuickBASIC 7.x(QBX)
qb /Lkm488qb4 *yourprog*	**qb /Lkm488qb7 *yourprog***

where ***yourprog*** is the name of your program.

Software Configuration

A number of KM-488-DD configuration parameters are set via the *CONFIG.DAT* file. (See Chapter 2 and Appendix F.) These govern the default settings of the GPIB input and output bus terminators, lightpen interrupt enable, device timeout period, and the KM-488-DD's Base Address. The defaults for these are listed in Table 8-1. There are other defaults you may have to re-program if you are using File I/O Commands in the same program as the library interface routines (calls).

Table 8-1. Default KM-488-DD Operating Parameters

Parameter	Default
Device Timeout	10.0 seconds
GPIB Bus Input Terminator	CR LF with no EOI
GPIB Bus Output Terminator	CR LF with no EOI
EOL Terminators	CR LF

The KCONFIG call can be used to reset the GPIB address, and the GPIB input and output bus terminators.

The KTERM call can be used to change the default GPIB bus terminator settings.

The KTO call can be used to change the default device timeout value.

To create a Standalone Program

This process compiles the QuickBASIC source code and links it to the QuickBASIC and KM-488-DD library files. This process is slightly different depending on the version of QuickBASIC used. (See your manual for specifics.) The following example shows you how to link the files in Version 4.x:

```
bc /o /d yourprog.bas;
link yourprog,,,bcom4x.lib+km488qb4;
```

where:

yourprog is the name of your program.

bcom4x.lib is the QuickBASIC Runtime library name.

km488qb4.lib is the linkable QuickBASIC library file.

Programming Notes

1. Any parameters that are not modified by a call may be passed as constants.

2. Parameters which are used to return values must be declared as variables.

3. Integer variable names end with a percent sign (or are declared AS INTEGER).

4. All integers are treated by the KM-488-DD routines as unsigned values (0 to 65535). However, QuickBASIC treats them as signed magnitudes (-32768 to +32767). When you need to express a value which is greater than or equal to 32768, you will need to express it in one of two ways:

   - Convert it to a hexadecimal value. Be sure to prefix these values with **&H** when equating them to a variable name. Legal hexadecimal values range from 0 to &HFFFF and can be used to represent values from 0 to 65535.

   - Use unsigned values from 0 to 32767 as is, but for values of 32768 to 65535 subtract 65536.

5. Do not name any of your variables with the same name as those assigned to the KM-488-DD routines.

8.1 DESCRIPTION FORMAT

The routines are described according to the following criteria.

Purpose This section provides a "one-line" description of the routine. A more detailed general description of each routine is provided in Chapter 3.

Syntax This section gives the syntax for each routine. The provided syntax assumes that the input parameters are passed in as variables.

Parameters This section describes the input parameters. In some instances, a structure or array of multiple parameters may be specified (for example, multiple device addresses).

MYGADS(N) is used in many calls. This is an array of **n** (1 to 15) GADS (GPIB addresses) structures which are required by various calls for addressing the bus. A single GADS structure consists of two integers. The first is the primary address of a device. The second is the secondary address. If the first is set to -1, the device is assumed to not exist. If no addresses are provided, it is assumed that the bus is already addressed.

An example of a structure with two devices is:

```
'$INCLUDE : 'HEADER.BI'
DIM MYGADS(15) AS GADS
MYGADS(0).PRIMAD = 3 'first device -- pri. adr. 3
MYGADS(0).SECAD = -1 'first device -- no sec. adr.
MYGADS(1).PRIMAD = 2 'second device -- pri. adr. 2
MYGADS(1).SECAD = 8 ' second device -- sec. adr. 8
MYGADS(2).PRIMAD = -1 ' no third device
```

If the bus is not going to be addressed (i.e., the KM-488-DD is a Peripheral), use the following **mygads** structure:

```
DIM NOADD(1) AS GADS
NOADD(0).PRIMAD = -1
```

Returns This describes any data which will be returned after execution of the call has been completed.

errcode% - All calls except KDIAGON, KUNPACK, KPACK, KSWAP, and KDIAGOFF return an integer error code which will be 0 if no error was detected. It is good practice to check **errcode%** after each call and provide a routine to handle errors if they arise.

Returns This section describes any values which are returned by the routine.

Programming Notes This section lists any special programming considerations for the routine.

Example This section gives a programming example using the routine.

KABORT

Purpose This routine allows the KM-488-DD to take control of the GPIB bus by forcing all other devices on the bus to the idle state. The KM-488-DD must be the Active Controller.

Syntax `CALL KABORT(boardnr%, errcode%)`

Parameters `boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the KM-488-DD is both the Active and System Controller, KABORT asserts the IFC line to retain control of the bus. Otherwise, the KM-488-DD will assert the ATN line, and then untalk and unlisten all devices.

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
brdnum = 0 ' First GPIB Board
'*** Seize Control of GPIB Bus ***
'
errnum = 0
CALL KABORT(brdnum, errnum)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KARM

Purpose Defines the conditions under which the lightpen status becomes true.

Syntax `CALL KARM(boardnr%, errcode%, armcode%)`

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`armcode%` is a 16-bit integer describing on which conditions a light pen interrupt may occur. The integer is of the format:

Bit	15 - 9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC	**Address Status Change.** If this bit is set to 1, the lightpen status will become true when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller). This encompasses all the conditions marked with an asterisk (*) below.
CIC*	**Controller in Charge.** If this bit is set to 1, the lightpen status will become true when the control is passed to the KM-488-DD and it becomes the Active Controller.
PER*	**Peripheral.** If this bit is set to 1, the lightpen status will become true when the KM-488-DD passes control to another device and becomes a Non-Active Controller.
DET	**Device Triggered.** If this bit is set to 1, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.

KARM (cont'd)

SRQ	**Internal SRQ.** If this bit is set to one, the lightpen status will become true when the KM-488-DD's internal Service Request Bit (SRQ) is set.
DEC	**Device Cleared.** If this bit is set to one, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA*	**Listen Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Listener and can accept data from the GPIB.
TA*	**Talk Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Talker and can output data to the GPIB.
IDLE*	**Idle.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is unaddressed as a Talker or a Listener.

☞ ***NOTE:*** The conditions marked by an asterisk (*) will force the lightpen status to true only when the KM-488-DD's address status has changed. This is indicated by the state of the Address Change Bit in the Status byte. See the KSTATUS or KQUIKSTAT routine descriptions for more information.

Returns `errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The driver must be configured for lightpen emulation at load time via the *CONFIG.DAT* "pipe" file, as the default for lightpen emulation is off.

KARM (cont'd)

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM armcode AS INTEGER ' Integer to hold interrupt enable
mask
brdnum = 0 ' First GPIB Board
'**  SET UP LIGHT PEN INTERRUPT  **
'
'Enable light pen interrupt detection on SRQ
'
ON PEN GOSUB 2000
'QB connection of subroutine to the light pen
PEN ON
'QB command to turn on light pen interrupt

armcode=&H10 'KM-488-DD will set light pen status bit on
SRQ
CALL KARM(brdnum, errnum, armcode)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
'
```

KBUF

Purpose Selects in which direction the memory is to be addressed for both DMA and non-DMA buffered transfers.

Syntax `CALL KBUF(boardnr%, errcode%, dir%)`

Parameters `boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`dir%` selects the direction in which the buffer is to be read. If `dir%` is equal to 1, the buffer address will be incremented. If `dir%` is equal to 0, the buffer address will be decremented.

Returns `errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the buffer's address is to be incremented during a bus I/O operation, be sure to use the lowest address in the designated buffer. Likewise, if it is to be decremented, select the highest address.

Example

```
DIM brdnum AS INTEGER     ' Number of board to be
accessed.
' Error code returned if any. 0 indicates no error
DIM errnum AS INTEGER
DIM memdir  AS INTEGER ' Direction to access memory
errnum = 0
brdnum = 0 ' First GPIB Board
memdir = 0 '  Decrement Memory Address On DMA
'  *** Set Direction Of Memory Address Change With DMA
***
CALL KBUF(brdnum, errnum, memdir)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KBUFD

Purpose Returns the number of characters transferred in the last buffered transfer or, in the case of DMA transfers, the current transfer.

Syntax `CALL KBUFD(boardnr%, errcode%, bcount%)`

Parameters `boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`bcount%` represents the number of characters which were transferred during the last buffered transaction. This will be a long integer (32-bits) in the range 0 to 1,048,575 (0 to 20^{20}-1).

Programming Notes

1. If data is transferred in the DMA CONTINUE mode, KBUFD returns represents the number of characters transferred thus far.

2. It is useful to call this routine after a buffered KENTER has been performed. This will help you to determine if the expected number of characters has been received and/or the transfer has terminated as the result of a detected terminator.

Example

```
DIM brdnum AS INTEGER ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM StrCnt AS LONG
errnum = 0
' Number of characters to be transferred to/from memory
brdnum = 0 ' First GPIB Board
'Return # Of Bytes In Last GPIB BUS Data Transfer
CALL KBUFD(brdnum, errnum, StrCnt)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KCLEAR

Purpose

The KM-488-DD must be an Active Controller. Resets device(s) to their power-up parameters.

Syntax

CALL KCLEAR(boardnr%, errcode%, mygads(0))

Parameters

boardnr% is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads(n) is an array of **n** (**n** can range from 1 to 15) GPIB bus addresses to be accessed. **mygads(n)** is comprised of **n** structures of type **GADS**. You must tailor **mygads** to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

Returns

errcode% is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If **mygads** contains no addresses, then the universal GPIB DCL (Device Clear) command is issued and all devices will be cleared.

2. If **mygads** contains addresses, then those devices specified will be listened and sent a GPIB SDC (Select Device Clear) Command.

Example

```
'$INCLUDE : 'HEADER.BI'
DIM noadd(1) AS GADS
' this structure holds primary and
' secondary addresses of the devices for the group
DIM brdnum AS INTEGER    ' Number of board to be
accessed.
DIM errnum AS INTEGER
errnum = 0
' Error code returned if any. 0 indicates  no error
noadd(0).primad = -1
' No first primary address, hence no addresses
'
'** With No Address Issue The Universal Device Clear--
Clear All Devices**
CALL KCLEAR(brdnum, errnum, noadd(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KCONFIG

Purpose Resets the KM-488-DD to its default operating parameters or to those specified.

Syntax `CALL KCONFIG(boardnr%, errcode%, cfgstring$, stringlen%)`

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`cfgstring$` is a user-defined string. This string can be comprised of any of the following options:

`/T` *`[term term EOI]`* Sets the input and output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **`/T`**, the GPIB bus terminators will be reset to none.

`term` is an optional parameter which represents a terminator byte to be used. This terminator byte can be represented as any of the following:

`$char` where **`char`** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **`char`** must be preceded by a dollar sign ($).

KCONFIG (cont'd)

Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.

CR	This represents the *Carriage Return* character (13 decimal, 0D hex).
LF	This represents the *Line Feed* character (10 decimal, 0A hex).
'X	where **X** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **'B** represents the character *B*.

EOI (End or Identify) is an optional GPIB BUS terminator. If included, the KM-488-DD will detect/assert the GPIB bus EOI line. (This would indicate that the last character has been sent.) Data will continue to be read until this terminator, a valid terminator sequence, or both are detected.

/TI *[term term EOI]*	Sets the input GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TI**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/TO *[term term EOI]*	Sets the output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TO**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/E *[term term]*	Sets the input and output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **term** is described above.

KCONFIG (cont'd)

/EI *[term term]*	Sets the input EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **term** is described above.
/EO *[term term]*	Sets the output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.)
/N name addr *[secaddr]*	Configures a named device. Sets its address to the given value and its GPIB bus terminators to the current GPIB bus terminator settings. (Note that this should only be used if the application program also includes File I/O Commands using named devices.) **name** is the name of the device. **addr** is the primary GPIB address (00 to 30). **secaddr** is the secondary GPIB address (00 to 31).

☞ ***NOTE:*** Do not create a named device with the same name as an existing directory on the current working disk.

If **cfgstring** is empty, then the configuration will revert to the installation configuration.

stringlen is an integer from 0 to 255 which gives the length of the **cfgstring** string.

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KCONFIG (cont'd)

Programming Notes

1. The **/E**, **/EI**, **/EO**, and **/N** options will only take effect if File I/O Commands are used within the same application program as the callables.

2. If you are converting an application program previously written using the File I/O commands to callables, be sure to note that the CONFIG string for the KCONFIG call does not include the "CONFIG" keyword (i.e., as you would if you were using the CONFIG File I/O command).

3. Primary and Secondary Addresses must each be two characters long, e.g. 01 or 0209.

Example

```
DIM Str AS STRING        ' String to hold command
DIM brdnum AS INTEGER    ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
DIM StrCnt AS INTEGER    ' Integer to hold size of Str
errnum = 0
brdnum = 0 ' First GPIB Board
' Configure Terminators as Carriage Return, Line Feed
With EOI
Str = "/T CR LF EOI"
Strcnt = LEN(Str)        'Length of Str
CALL KCONFIG(brdnum, errnum, Str, StrCnt)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KDIAGON

Purpose Turns file diagnostics on. (Default is off.)

Syntax `CALL KDIAGON(filename$, strlen%)`

Parameters **`filename$`** is a string containing the name of the file to which the diagnostic information is to be written.

`strlen%` is the length of the string representing the name of the file.

Returns None.

Programming Notes

1. This will echo the value of the call parameters into a disk file.

Example

```
DIM Str AS STRING        ' String to hold command
DIM StrCnt AS INTEGER    ' Integer to hold size of Str
Str = "DEBUG.DAT"      'Target Diagnostic File Name
StrCnt = LEN(Str)      'Length of Str
'Turn Driver Diagnostics On And Send Info To Disk File
CALL KDIAGON(Str,StrCnt)    'File Name & File Name Length
```

KDIAGOFF

Purpose	Disables file diagnostics.
Syntax	`CALL KDIAGOFF()`
Parameters	None.
Returns	None.
Programming Notes	None.
Example	`CALL KDIAGOFF()`

KDISARM

Purpose Resets previously set conditions which cause lightpen interrupts.

Syntax `CALL KDISARM(boardnr%, errcode%, armcode%)`

Parameters **boardnr%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

armcode% is a 16-bit integer describing which conditions should be reset. The integer is of the format:

Bit	15 - 9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC	**Address Status Change.** If this bit is set to 1, light pen status will not change when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).
CIC	**Controller in Charge.** If this bit is set to 1, light pen status will not change when the control is passed to the KM-488-DD and it becomes the Active Controller.
PER	**Peripheral.** If this bit is set to 1, light pen status will not change when the KM-488-DD passes control to another device and becomes a Non-Active Controller (Peripheral).
DET	**Device Triggered.** If this bit is set to 1, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.
SRQ	**Internal SRQ.** If this bit is set to one, light pen status will not change when the KM-488-DD's internal Service Request Bit (SRQ) is set.

KDISARM (cont'd)

DEC	**Device Cleared.** If this bit is set to one, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA	**Listen Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Talker and can output data to the GPIB.
IDLE	**Idle.** If this bit is set to one, light pen status will not change when the KM-488-DD is unaddressed as a Talker or a Listener.

Returns `errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM disarm AS INTEGER  ' Integer to disable mask
errnum = 0
brdnum = 0 ' First GPIB Board
' DISABLE LIGHT PEN INTERRUPT
'
'Disable light pen interrupt detection on SRQ
'
PEN OFF  'QB command to turn off light pen interrupt
'KM-488-DD will not set light pen status bit on SRQ
disarm=&H10
CALL KDISARM(brdnum, errnum, disarm)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KDMA

Purpose Sets DMA transfer mode.

Syntax **`CALL KDMA (boardnr%, errcode%, dmamode%)`**

Parameters **boardnr%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

dmamode% is a 16-bit integer which represents the DMA mode to be used. If no **dmamode%** is given, then single mode without autoinitialization will be assumed. The **dmamode%** integer is interpreted as follows:

Bit	15 - 3	2	1	0
	0	SIN	DEM	AUTO

Where:

SIN **SINGLE.** When this bit is set to one, the SINGLE mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus and transfers a single byte of data. Control of the bus is then returned to the microprocessor.

Note that SINGLE and DEMAND mode can not be invoked simultaneously.

DEM **DEMAND.** When this bit is set to one, the DEMAND mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus. The DMA controller retains control of the bus until the DMA request signal is unasserted. Once this signal has been unasserted for more than one microprocessor clock cycle, control of bus is returned to the microprocessor. This mode allows the DMA controller chip to pass data at a slightly faster rate and the microprocessor to access the bus when it is not needed.

Note that SINGLE and DEMAND mode can not be invoked simultaneously.

KDMA (cont'd)

AUTO **AUTOINITIALIZE.** When this bit is set to one, the AUTOINITIALIZE mode is selected. Under normal circumstances, the DMA controller transfers the specified number of bytes to/from the PC memory from the given starting address and terminates when completed. When the AUTOINITIALIZE mode is enabled, the DMA controller will reset the original byte count, reset the initial address, and repeat the transfer again. The AUTOINITIALIZE option is only in effect until the next KENTER or KOUTPUT buffered transfer is completed. Additionally, if a KENTER or KOUTPUT transfer in DMA CONTINUE mode is selected, the AUTOINITIALIZE option will only be in effect for that transfer.

Returns None.

Programming Notes

1. All DMA AUTOINITIALIZE transfers should occur entirely within a single DMA 64 KByte page. So, if a DMA buffer operation involves a transfer of more than 64K or occur across a DMA page boundary, the AUTOINITIALIZE option affects only those bytes in the last transfer occurring within one DMA page.

   Also note that DMA AUTOINITIALIZE transfers specified for buffers that are located in more than a single DMA page are unpredictable.

2. It is impossible for a program to halt a DMA AUTOINITIALIZE operation unless the DMA CONTINUE option is selected. This is because the driver does not return control to a program using non-CONTINUE operations until the transfer completes.

KDMA (cont'd)

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM singleauto AS INTEGER  ' Integer to hold DMA command
brdnum = 0 ' First GPIB Board
errnum = 0
singleauto = 4 + 1  'Single/autoinitialize DMA Transfer
'*** Set Up DMA Mode To Single With Autoinitialize ***
CALL DMA(brdnum, errnum, singleauto)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KENTERBUF

Purpose Allows the KM-488-DD to receive data from a GPIB bus device into a QuickBASIC integer array.

Syntax `CALL KENTERBUF(boardnr%, errcode%, data%(0), count, mode%, mygads%(0), termlist%(0))`

Parameters `boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`data%` is an integer array large enough to accept the returned data. If a non-zero `count` is specified, `data%(count/2)` will suffice for any returned data. If `count` is 0 and terminator characters are used, then `data%` must be dimensioned to accommodate both data and terminators returned by the device.

`count` is a long integer representing the number of characters to be read. Valid counts are within the range 0 to 4294967295 (2^{32}-1) decimal, or from 0 to &HFFFFFFFF. When 0 is used, the KENTERBUF will stop when the termination specified by `termlist` is satisfied.

`mode%` is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

CONT CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

KENTERBUF (cont'd)

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked or a pre-specified GPIB BUS input terminator is detected.

DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer.

mygads (n) is an array of **n** (**n** can range from 1 to 15) GPIB bus addresses to be accessed. **mygads (n)** is comprised of **n** structures of type **GADS**. You must tailor **mygads** to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

termlist (3) is an integer array which defines the GPIB bus terminators to be used during the KENTERBUF call. These terminators are used in conjunction with **count** to terminate the input process. Terminator choices include whether or not EOI will be detected to signal end of input and which character sequence, if any, is to be detected as an end of message code. The default terminator scheme is initially set-up by the *CONFIG.DAT* file, and can be modified by calling KTERM or KCONFIG. **termlist** can either specify the use of the default terminators or temporary overrides.

termlist is comprised of the following elements:

termlist (0) : **EOI detection.** Possible values are:

-1(&HFFFF)	Use the default EOI setting.
0	Don't Detect EOI.
1	Detect EOI.

termlist (1) : **1st GPIB bus terminator.** Possible values are:

-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.

KENTERBUF (cont'd)

0 to 255	Detect this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

termlist(2): **2nd GPIB bus terminator.** Possible values are:

-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
0 to 255	Detect this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default `termlist` can also be used:

```
DIM DTRM(3) AS INTEGER
DTRM(0) = -1 ' Use the default EOI choice
DTRM(1) = -1 ' Use the default first terminator
DTRM(2) = -1 ' Use the default second terminator
```

Returns `errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KENTERBUF (cont'd)

Programming Notes

1. If the KM-488-DD is a Peripheral, an active controller must address the KM-488-DD as a listener before KENTERBUF is called. In this instance, you should use the **noadd** structure in lieu of **mygads**. This is described in section 8.1.

2. If the KM-488-DD is the Active Controller, it will automatically address itself as a listener and another device defined in **mygads** as the talker. Since only one device can be a talker, the address group **mygads** can contain only one address. If a device has been previously addressed to talk and the KM-488-DD previously addressed to listen, then **mygads** can be **noadd** (See section 8.1) and the bus will not be re-addressed.

3. If **count** is specified as 0, then KENTER will terminate when the terminator(s) specified by **termlist** is encountered.

4. If it is desirable to terminate on **count** only and not use the default terminators, then you must select no terminator overrides in the **termlist** array, in addition to giving the count. In this case, a byte by byte synchronous transfer is executed.

5. If both **count** and terminators are specified, the KENTERBUF will terminate on whichever condition is encountered first.

6. If the condition(s) for termination is not encountered, KENTERBUF will timeout unless the timeout has been disabled.

7. To manipulate the data, which is returned in the form of two data bytes per integer location, you may need to use KUNPACK.

KENTERBUF (cont'd)

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER  ' Error code returned, 0 = no
error
DIM numd AS INTEGER    ' Number of bytes to transfer
DIM mode AS INTEGER    ' DMA mode
' this structure will hold the primary and
' secondary addresses of the devices
DIM SCOPEadd(2) AS GADS
DIM SCOPEtrm(3) AS INTEGER
' Terminators for enter
DIM dmain(514)   'Array to hold DMA input waveform from
scope
SCOPEadd(0).primad = 5  ' first primary address = 5
SCOPEadd(0).secad = -1  ' first secondary address = NONE
SCOPEadd(1).primad = -1
' second primary address = NONE and NONE to follow
brdnum = 0 ' First GPIB Board
errnum = 0
mode = 1  ' DMA mode flag -- 0: no DMA
          '                 1: DMA Without CONTINUE
          '                 3: DMA With CONTINUE
          '
SCOPEtrm(0) = -1  ' Use default EOI
SCOPEtrm(1) = -1  ' Use default first
SCOPEtrm(2) = -1  ' Use default second

numd = 1028              'Number of bytes to transfer
'
'Enter 1028 bytes form the device at GPIB address 5

CALL KENTERBUF(brdnum, errnum, dmain(0), numd, mode, SCO-
PEadd(0), SCOPEtrm(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KENTERSTR

Purpose Allows the KM-488-DD to receive data from a GPIB bus device.

Syntax **`CALL KENTERSTR(boardnr%, errcode%, data$, count, mode%, mygads%(0), termlist%(0))`**

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`data$` is a string large enough to accept the returned data. If a non-zero **`count`** is specified, the string should be of length **`count`**. If **`count`** is 0 and terminator characters are used, the string must be large enough to accommodate both data and terminator characters returned by the device.

`count` is a long integer representing the number of characters to be read. Valid **`counts`** are within the range 0 to 4294967295 (2^{32}-1) decimal, or from 0 to &HFFFFFFFF. Note however, that QuickBASIC strings have a maximum length of 65535(&HFFFF). When **`count`** is 0, the KENTERSTR will stop when the terminators specified by **`termlist`** have been detected.

`mode%` is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

CONT	CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

KENTERSTR (cont'd)

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked or a pre-specified GPIB BUS input terminator is detected.

DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer.

mygads (n) is an array of **n** (**n** can range from 1 to 15) GPIB bus addresses to be accessed. **mygads (n)** is comprised of **n** structures of type **GADS**. You must tailor **mygads** to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

termlist (3) is an integer array which defines the GPIB bus terminators to be used during the KENTERSTR call. These terminators are used in conjunction with **count** to terminate the input process. Terminator choices include whether or not EOI will be detected to signal end of input and which character sequence, if any, is to be detected as an end of message code. The default terminator scheme is initially set-up by the *CONFIG.DAT* file, and can be modified by calling KTERM or KCONFIG. **termlist** can either specify the use of the default terminators or temporary overrides.

termlist is comprised of the following elements:

termlist (0) : **EOI detection.** Possible values are:

-1(&HFFFF)	Use the default EOI setting.
0	Don't Detect EOI.
1	Detect EOI.

termlist (1) : **1st GPIB bus terminator.** Possible values are:

-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.

KENTERSTR (cont'd)

	0 to 255	Detect this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
termlist(2):	**2nd GPIB bus terminator.** Possible values are:	
	-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
	0 to 255	Detect this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default **termlist** can also be used:

```
DIM DTRM(3) AS INTEGER
DTRM(0) = -1 ' Use the default EOI choice
DTRM(1) = -1 ' Use the default first terminator
DTRM(2) = -1 ' Use the default second terminator
```

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KENTERSTR (cont'd)

Programming Notes

1. If the KM-488-DD is a Peripheral, an active controller must address the KM-488-DD as a listener and some other device as the talker, before KENTERSTR is called. In this instance, you should use the **noadd** structure in lieu of **mygads**. This is described in section 8.1.

2. If the KM-488-DD is the Active Controller, it will automatically address itself as a listener and another device defined in **mygads** as the talker. Since only one device can be a talker, the address group **mygads** can contain only one address. If a device has been previously addressed to talk and the KM-488-DD previously addressed to listen, then **mygads** can be **noadd** (See section 8.1) and the bus will not be re-addressed.

3. If **count** is specified as 0, then KENTERSTR will terminate when the terminator(s) specified by **termlist** is encountered.

4. If it is desirable to terminate on **count** only and not use the default terminators, then you must select no terminator overrides in the **ter-mlist** array , in addition to giving the count. In this case, a byte by byte synchronous transfer is executed.

5. If both **count** and terminators are specified, the KENTERSTR will terminate on whichever condition is encountered first.

6. If the condition(s) for termination is not encountered, KENTERSTR will timeout unless the timeout has been disabled.

7. Use the **LONG** function **SSEGADD&()** to obtain the address of a string. This function is intrinsic to QuickBASIC 7.X only. It is supplied for users of Versions 4.X in the files *KM488QB4.LIB* and *KM488QB4.QLB.*

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
DIM numd AS INTEGER    ' Number of bytes to transfer
DIM mode AS INTEGER    ' DMA mode
DIM StrCnt AS INTEGER  ' Integer to hold size of Str
DIM Str AS STRING      ' String to hold command
```

KENTERSTR (cont'd)

```
DIM DMMadd(2) AS GADS  ' this structure will hold the
primary and
                       ' secondary addresses of the
devices
DIM DMMtrm(3) AS INTEGER  ' Terminators for enter

DIM brdnum AS INTEGER  ' This variable will contain the
board
                       ' number to be accessed.

DMMadd(0).primad = 5  ' first primary address = 5
DMMadd(0).secad = -1  ' first secondary address = NONE
DMMadd(1).primad = -1
' second primary address = NONE and NONE to follow
brdnum = 0 ' First GPIB Board
errnum = 0
mode = 0  ' DMA mode flag -- 0: no DMA
          '                  1: DMA Without CONTINUE
          '                  3: DMA With CONTINUE
          '
DMMtrm(0) = -1  ' Use default EOI
DMMtrm(1) = -1  ' Use default first
DMMtrm(2) = -1  ' Use default second
numd = 0
'Number of bytes to transfer -- 0 means go until termina-
tion
'
Str = STRING$(20, 32)
' Must be long enough to hold returned data including
GPIB
' terminators
CALL KENTERSTR(brdnum, errnum, SSEGADD&(Str), numd, mode,
DMMadd(0), DMMtrm(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KERROR

Purpose Enables/Disables display of Error Messages.

Syntax `CALL KERROR(boardnr%, errcode%, esw%)`

Parameters **boardnr%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

esw% is an integer which determines if error message display function will be enabled or disabled. If **esw%=1**, the error message display is enabled. (This is the default.) If **esw%=0** is specified, the error message display is disabled.

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If KERROR has been used to enable error messages, the message will only reveal that an error has occurred for the KM-488-DD calls. It does not identify what type of error occurred. The KDIAGON routine should be used to help debug applications utilizing the KM-488-DD calls, and the KSTATUS routine used for identifying the error number.

2. Programs can also check **err_code%** after each call to identify the exact nature of call errors.

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
' 0 disables error display 1 enables
DIM errsw AS INTEGER
brdnum =  0 ' First GPIB Board
errnum = 0
errsw =  1 ' Enable Error Display
'  *** Enable Error Display ***
CALL KERROR(brdnum, errnum, errsw)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KFILL

Purpose Defines the KM-488-DD driver's response to a request for data when none is available.

☞ ***NOTE:*** KFILL is provided for users who mix File I/O Commands and Calls in the same program. It only affects inputs performed using the File I/O Commands.

Syntax `CALL KFILL(boardnr%, errcode%, fillsw%)`

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`fillsw%` is an integer which represents the fill condition. (The default condition is for the KM-488-DD driver to return a NULL character.) **`fillsw%`** can be any one of the following:

-1	OFF. The KM-488-DD driver will not return any data characters or a DOS error message, if no input data is available.
-2	ERROR. The KM-488-DD driver will not return any data characters, but will generate an Error Message. The error message number can then be trapped by calling the KSTATUS routine. (Refer to the KSTATUS routine for more information.)
0 - 255	ASCII. The KM-488-DD driver will return the designated ASCII character. (An ASCII Equivalence Chart is provided in Appendix A.) Specify the decimal equivalent of the character to be used.

Returns **`errcode%`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KFILL

Example

```
DIM brdnum AS INTEGER     ' Number of board to be
accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM fillchar AS INTEGER  ' ASCII value of fill character
brdnum =     0  ' First GPIB Board
errnum = 0
fillchar = 88 ' Decimal equivalent of "X"
'  *** Set Fill Character To 'X' ***
CALL KFILL(brdnum, errnum, fillchar)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KHELLO

Purpose Returns an identification string from the KM-488-DD driver.

Syntax `CALL KHELLO(boardnr%, errcode%, idresponse$)`

Parameters `boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `idresponse$` will contain a string similar to: `copyright (c) 1991 Keithley MetraByte Corp`.

`errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The string into which the HELLO message is returned must be dimensioned prior to calling KHELLO. Otherwise, QuickBASIC assumes that the string is of length 0 and calling KHELLO will result in destroying other programmed data. (See the QuickBASIC `STRING$` function.)

Example

```
DIM brdnum AS INTEGER   ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM Str AS STRING       ' String to hold returned message
brdnum =   0            'First GPIB Board
' Must be long enough to hold returned id
Str = STRING$(50, 32)
'*** Return ID from driver to verify its presence ***
CALL KHELLO(brdnum, errnum, Str)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KLOCAL

Purpose	Forces the specified bus device(s) to return to being programmed locally from their front panels. This routine can only be used if the KM-488-DD is acting as a System or Active Controller.
Syntax	`CALL KLOCAL(boardnr%, errcode%, mygads(0))`
Parameters	`boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed. `mygads(n)` is an array of `n` (`n` can range from 1 to 15) GPIB bus addresses to be accessed. `mygads(n)` is comprised of `n` structures of type **GADS**. You must tailor `mygads` to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*
Returns	`errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.
Programming Notes	1. If the KM-488-DD is the System Controller and if `mygads` does not specify an address, then the GPIB REN (Remote Enable) line is unasserted and all devices are returned to Local. In order to return them to remote mode, it will be necessary to issue a KREMOTE call. Likewise, if Local Lockout is required, it will be necessary to issue a KLOL call. 2. As an Active Controller, the KM-488-DD can issue the GPIB GTL (Go To Local) message to those devices specified by `mygads`. In this case, the GPIB REN (Remote Enable) line remains asserted and devices will return to remote when addressed to listen. If a KLOL (Local Lockout) call has been issued previously, it should still be in effect when a device is returned to Remote.

KLOCAL (cont'd)

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER  ' Error code returned if any. 0
indicates  no error
DIM DMMadd(2) AS GADS
' this structure will hold the primary
' and secondary addresses of the devices

DMMadd(0).primad = 5  ' first primary address = 5
DMMadd(0).secad = -1  ' first secondary address = NONE
DMMadd(1).primad = -1
' second primary address = NONE and NONE to follow
brdnum = 0 ' First GPIB Board
errnum = 0
'*** Send Device With GPIB Address Back To Local
CALL KLOCAL(brdnum, errnum, DMMadd(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KLOL

Purpose

This command can only be used if the KM-488-DD is an Active or System Controller. It will disable the GPIB bus devices from being returned to Local Control by means of the Local/Remote button on the device.

☞ ***NOTE:*** This routine issues an IEEE-488 bus signal, LOL. This signal is not supported by all IEEE-488 bus devices.

Syntax

`CALL KLOL(boardnr%, errcode%)`

Parameters

`boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns

`errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The "LOCAL" button is disabled so that a device cannot be returned to local state for manual programming or control. The Active Controller can return specific devices to "local with lockout state", whereby an operator can then use the front panel controls. When the device is addressed to listen again, it returns to "remote with lockout state". Thus, the effect of the LOL call will remain until the REN line is unasserted (LOCAL) by the System Controller.

2. It is good practice to issue a KLOL so that devices are under strict program control. KLOL can be issued before a device is put in remote and will take effect when the device's LOCAL button is set to remote.

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
brdnum =   0  ' First GPIB Board
'Prevent Devices From Being Returned To Local Control
errnum = 0
CALL KLOL(brdnum, errnum)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KOUTPUTBUF

Purpose Transmits data from the KM-488-DD to the GPIB bus.

Syntax **`CALL KOUTPUTBUF(boardnr%, errcode%, data%(0), count, mode%, mygads(0), termlist(0))`**

Parameters `boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`data%` is an array containing the data to be output.

`count` is a long integer representing the number of characters to be output. Valid counts are within the range 1 to 4294967295 (2^{32}-1) decimal, or from 1 to &HFFFFFFFF.

`mode%` is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

CONT CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

KOUTPUTBUF (cont'd)

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked.

DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer.

mygads(n) is an array of **n** (**n** can range from 1 to 15) GPIB bus addresses to be accessed. **mygads(n)** is comprised of **n** structures of type **GADS**. You must tailor **mygads** to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

termlist is an integer array which defines the GPIB bus terminators to be used during the KOUTPUTBUF call. These terminators are used in conjunction with **count** to terminate the output process. Terminator choices include whether or not EOI will be asserted with the last byte and whether or not a certain byte or sequence of two bytes will be sent as an end of message code. The default terminator scheme is initially set-up via the *CONFIG.DAT* file, and can be modified by calling KTERM or KCONFIG. **termlist** can be used either to specify the use of the default terminators or to use temporary override terminators.

termlist is comprised of the following elements:

termlist(0): EOI assertion. Possible values are:

-1(&HFFFF)	Use the default EOI setting.
0	Don't Assert EOI.
1	Assert EOI.

termlist(1): **1st GPIB bus terminator.** Possible values are:

-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.

KOUTPUTBUF (cont'd)

	0 to 255	Send this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
termlist(2):	**2nd GPIB bus terminator.** Possible values are:	
	-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
	0 to 255	Send this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default `termlist` can also be used:

```
DIM DTRM(3) AS INTEGER
DTRM(0) = -1 ' Use the default EOI choice
DTRM(1) = -1 ' Use the default first terminator
DTRM(2) = -1 ' Use the default second terminator
```

Returns

errcode% is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is a peripheral, then an Active Controller must address the KM-488-DD as a talker and some other device(s) as the listener(s). In this case, use **noadd** for **mygads**. (See section 8.1.).

2. If the KM-488-DD is both the System and Active Controller, and **mygads** contains the device(s) to be addressed, the KOUTPUTBUF will automatically assert the GPIB REN (Remote Enable) line.

KOUTPUTBUF (cont'd)

3. If the KM-488-DD is the Active Controller and KOUTPUTBUF is called, it will automatically address itself as the talker and another device(s) contained in **mygads** as the listener(s). If the devices have been previously addressed, then **mygads** can be **noadd** and the bus will not be re-addressed.

4. A non-zero **count** must always be specified.

5. If you do not wish to append terminator characters, then be sure to set the terminator defaults via *CONFIG.DAT* or by calling KTERM or KCONFIG and select no terminator overrides in the **termlist** array.

6. If a listener does not accept a character, a timeout will occur unless the timeout has been disabled.

Example

```
DIM SCOPEtrm(3) AS INTEGER      ' Terminators for output
DIM brdnum AS INTEGER
' This will contain the board number to be accessed.
DIM dmaout(514)
'Array to hold DMA output waveform for scope
noadd(0).primad = -1
' No first primary address, hence no addresses
brdnum = 0 ' First GPIB Board
'
mode = 1  ' DMA mode flag -- 0: no DMA
          '                 1: DMA Without CONTINUE
          '                 3: DMA With CONTINUE
          '
SCOPEtrm(0) = -1  ' Use default EOI
SCOPEtrm(1) = -1  ' Use default first
SCOPEtrm(2) = -1  ' Use default second
numd = 1028                'Number of bytes to transfer
errnum = 0
'
'*** Code to fill dmaout here ***
'       .
'
CALL koutputBuf(brdnum, errnum, dmaout(0), numd, mode,
noadd(0), SCOPEtrm(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KOUTPUTSTR

Purpose Transmits data from the KM-488-DD to the GPIB bus.

Syntax `CALL KOUTPUTSTR (boardnr%, errcode%, data$, count, mode%, mygads(0), termlist(0))`

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`data$` is a string of data to be output.

`count%` is a long integer representing the number of characters to be output. Valid counts are within the range 1 to 4294967295 (2^{32}-1) decimal, or from 1 to &HFFFFFFFF hex.

`mode%` is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

CONT CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

KOUTPUTSTR (cont'd)

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked.

DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer.

mygads (n) is an array of **n** (**n** can range from 1 to 15) GPIB bus addresses to be accessed. **mygads (n)** is comprised of **n** structures of type **GADS**. You must tailor **mygads** to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

termlist is an integer array which defines the GPIB bus terminators to be used during the KOUTPUTSTR call. These terminators are used in conjunction with **count** to terminate the output process. Termination choices include whether or not EOI will be asserted with the last byte and whether or not a certain byte or sequence of two bytes will be sent as an end of message code. The default termination scheme is initially set-up via *CONFIG.DAT* program, and can be modified by calling KTERM or KCONFIG. **termlist** can be used to specify the use of the default terminators or a different terminator.

termlist (3) is comprised of the following elements:

termlist (0) : **EOI detection.** Possible values are:

-1(&HFFFF)	Use the default EOI setting.
0	Don't Assert EOI.
1	Assert EOI.

termlist (1) : **1st GPIB bus terminator.** Possible values are:

-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.

KOUTPUTSTR (cont'd)

	0 to 255	Send this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
termlist(2):	**2nd GPIB bus terminator.** Possible values are:	
	-1(&HFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
	0 to 255	Send this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default **termlist** can also be used:

```
DIM DTRM(3) AS INTEGER
DTRM(0) = -1 ' Use the default EOI choice
DTRM(1) = -1 ' Use the default first terminator
DTRM(2) = -1 ' Use the default second terminator
```

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is a peripheral, then an Active Controller must address the KM-488-DD as a talker and some other device(s) as the listener(s). In this case, use the **noadd** structure in lieu of **mygads**. (See section 8.1.).

2. If the KM-488-DD is both the System and Active Controller, and **mygads** contains the device(s) to be addressed, the KOUTPUT will automatically assert the GPIB REN (Remote Enable) line.

KOUTPUTSTR (cont'd)

3. If the KM-488-DD is the Active Controller and KOUTPUT is called, it will automatically address itself as the talker and another device(s) contained in **mygads** as the listener(s). if the devices have been previously addressed, then **mygads** can be **noadd** and the bus will not be re-addressed.

4. A non-zero **count** must always be specified.

5. If you do not wish to append terminator characters, then be sure to set the terminator defaults accordingly. This may be accomplished via *CONFIG.DAT* or by calling KTERM or KCONFIG and use no overrides in the **termlist**.

6. If a listener does not accept a character, a timeout will occur unless the timeout has been disabled.

7. Use the **LONG** function **SSEGADD& ()** to obtain the address of a string. This function is intrinsic to QuickBASIC 7.X only. It is supplied for users of Versions 4.X in the files *KM488QB4.LIB* and *KM488QB4.QLB*.

KOUTPUTSTR (cont'd)

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
 ' Error code returned if any. 0 indicates no error
DIM numd AS INTEGER    ' Number of bytes to transfer
DIM mode AS INTEGER    ' DMA mode
DIM Str AS STRING      ' String to hold returned message
DIM SCOPEadd(2) AS GADS
' this structure will hold the primary
' and secondary addresses of the devices for output
DIM SCOPEtrm(3) AS INTEGER     ' Terminators for output
DIM brdnum AS INTEGER
' This variable will contain the board number to be
' accessed.

SCOPEadd(0).primad = 5  ' first primary address = 5
SCOPEadd(0).secad = -1  ' first secondary address = NONE
SCOPEadd(1).primad = -1
 ' second primary address = NONE and NONE to follow
brdnum = 0 ' First GPIB Board
errnum = 0
mode = 1  ' DMA mode flag -- 0: no DMA
          '                  1: DMA Without CONTINUE
          '                  3: DMA With CONTINUE
          '
SCOPEtrm(0) = -1  ' Use default EOI
SCOPEtrm(1) = -1  ' Use default first
SCOPEtrm(2) = -1  ' Use default second
Str = STRING$(20, 32)
Str = "HOR ASE:5E-4"
StrCnt = LEN(Str)
CALL KOUTPUTSTR(brdnum, errnum, SSEGADD&(Str), StrCnt,
mode, SCOPEadd(0), SCOPEtrm(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KPACK

Purpose This packs discrete 8-bit data values (stored in a 16-bit integer array) into sequential bytes. This routine is useful when an instrument can only accept an 8-bit data value, because the KM-488-DD driver can only transfer contiguous bytes.

Syntax `KPACK(source%(0),dest%(0),count%)`

Parameters `source%` is the name of the integer array which contains the data to be converted.

`dest%` is the name of the integer array in which the converted data is to be stored.

`count%` is an integer which represents the number of elements comprising the `source%` array. (The number of elements needed in the `dest%` array is `count%/2`.)

Returns Result is stored in `dest%`.

Programming Notes

1. Both `source%` and `dest%` must be declared as COMMON-SHARED. The KM-488-DD driver assumes that each of these arrays resides in QuickBASIC's default data segment.

Example

```
DIM numd AS INTEGER     ' Number of bytes to transfer
'Array to hold DMA output waveform
DIM dmaout(514) AS INTEGER
'Working array
DIM workb(1028) AS INTEGER
COMMON SHARED dmaout(), workb()
numd = 1028            'Number of bytes to transfer
'Convert the word (integer) working array into
' a packed array.
CALL KPACK(VARPTR(workb(0)), VARPTR(dmaout(0)), numd%)
```

KPASCTL

Purpose	Designates another controller to be the Active Controller. The KM-488-DD must be an Active Controller.
Syntax	`CALL KPASCTL(boardnr%, errcode%, mygads(0))`
Parameters	`boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed. `mygads(n)` is an array of **n** (**n** can range from 1 to 15) GPIB bus addresses to be accessed. `mygads(n)` is comprised of **n** structures of type **GADS**. You must tailor `mygads` to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*
Returns	`errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.
Programming Notes	If the KM-488-DD which is relinquishing its position as the Active Controller is also a System Controller, it retains its status as System Controller.

KPASCTL (cont'd)

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
DIM PERIadd(2) AS GADS
' this structure will hold the primary and
' secondary addresses of the device to take control
PERIadd(0).primad = 2  ' first primary address = 2
PERIadd(0).secad = -1  ' first secondary address = NONE
PERIadd(1).primad = -1
' second primary address = NONE and NONE to follow
brdnum = 0 ' First GPIB Board
errnum = 0
 ' *** Pass To Device At GPIB Address 2 ***
CALL KPASCTL(brdnum, errnum, PERIadd(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KPPOLL

Purpose Initiates a Parallel Poll. The KM-488-DD must be the Active Controller.

☞ ***NOTE:*** Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax `CALL KPPOLL(boardnr%, errcode%, pprtn%)`

Parameters **boardnr%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

pprtn% is an integer variable which will contain an integer in the range 0 to 255 decimal. This integer indicates which Data Lines which have been asserted (DIO1-DIO8).

Programming Notes None.

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM ppresp AS INTEGER ' Parallel poll response
brdnum = 0 ' First GPIB Board
'
' *** Conduct Parallel Poll ***
errnum = 0
CALL KPPOLL(brdnum, errnum, ppresp)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
'
PRINT
PRINT "The Parallel Poll Returned = ";ppresp
```

KPPC

Purpose Configures the Parallel Poll response of a GPIB bus device. The KM-488-DD must be the Active Controller.

☞ ***NOTE:*** Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax `CALL KPPC(boardnr%, errcode%, mygads(0), ppccfg%)`

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads (n)` is an array of **n** (**n** can range from 1 to 15) GPIB bus addresses to be accessed. **`mygads (n)`** is comprised of **n** structures of type **GADS**. You must tailor **`mygads`** to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

`ppccfg%` is an integer which represents the Parallel Poll Response of the device to be programmed. This integer is of the format:

Bit	7	6	5	4	3	2	1	0
	0	1	1	0	S	P2	P1	P0

KPPC (cont'd)

Where:

S	is the parallel poll response value (0 or 1) that the device uses to respond to the parallel poll when service is required. This bit is generally set to 1.

☞ ***NOTE:*** This value must correspond to the setting of the GPIB bus device's *ist* bit (individual status bit). Refer to the Device's documentation for more information.

P2 P1 P0	is a 3-bit value which tells the device being configured which data bit (DIO1 through DIO8) it should use as its parallel poll response.

Returns **`errcode%`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPC (cont'd)

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM ppconf AS INTEGER ' Parallel poll configuration
DIM PWRadd(2) AS GADS
 ' this structure will hold the primary and
' secondary addresses of the device to take to be confi-
gured
PWRadd(0).primad = 1  ' first primary address = 1
PWRadd(0).secad = -1  ' first secondary address = NONE
PWRadd(1).primad = -1
' second primary address = NONE and NONE to follow
brdnum = 0 ' First GPIB Board
' Configure Device At GPIB Address 1 To Respond On DIO3
Line ***
ppconf = &HA
errnum = 0
CALL KPPC(brdnum, errnum, PWRadd(0), ppconf%)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KPPD

Purpose Disables the Parallel Poll response capability of the specified GPIB bus device(s). The KM-488-DD must be the Active Controller.

Syntax **`CALL KPPD (boardnr%, errcode%, mygads(0))`**

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads (n)` is an array of **`n`** (**`n`** can range from 1 to 15) GPIB bus addresses to be accessed. **`mygads (n)`** is comprised of **`n`** structures of type **GADS**. You must tailor **`mygads`** to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

Returns **`errcode%`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPD (cont'd)

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
DIM PWRadd(2) AS GADS
' this structure will hold the primary and secondary
' addresses of the device to take to be disabled
PWRadd(0).primad = 1  ' first primary address = 1
PWRadd(0).secad = -1  ' first secondary address = NONE
PWRadd(1).primad = -1
 ' second primary address = NONE and NONE to follow
brdnum = 0 ' First GPIB Board
errnum = 0
'  Disable Device @ GPIB Add #1 From Responding To KPPOLL
CALL KPPD(brdnum, errnum, PWRadd(0))
```

KPPU

Purpose Disables the Parallel Poll Response of all GPIB bus device(s). The KM-488-DD must be the Active Controller.

Syntax `CALL KPPU(boardnr%, errcode%)`

Parameters **boardnr%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
 ' Error code returned if any. 0 indicates no error
brdnum = 0 ' First GPIB Board
errnum = 0
' Disable Parallel Poll Response Of All Devices
CALL KPPU(brdnum, errnum)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KQUIKSTAT

Purpose Returns the KM-488-DD's status byte.

Syntax `CALL KQUIKSTAT(boardnr%, errcode%, qstat%)`

Parameters `boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `qstat%` is a pointer to the returned status integer. The status integer is of the following format:

Bit	15-9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PEND	DET	SRQ	DEC	LA	TA	DMA

Where:

ADSC	**Address Status Change.** If this bit is set to 1, a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).
CIC	**Controller in Charge.** If this bit is set to 1, the KM-488-DD is an Active Controller.
PEND	**SRQ Pending.** If this bit is set to 1, the KM-488-DD has an SRQ request pending.
DET	**Device Triggered.** If this bit is set to 1, a GPIB Trigger command has been received.
SRQ	**Internal SRQ.** If the KM-488-DD is an Active Controller and this bit is set to one, a device is requesting service. If the KM-488-DD is a Peripheral and this bit is set to one, then its SRQ (issued by KREQUEST) has not been serviced.
DEC	**Device Cleared.** If this bit is set to one, the KM-488-DD has received a GPIB Clear command.

KQUIKSTAT(cont'd)

LA	**Listen Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Talker and can output data to the GPIB.
DMA	DMA. A DMA transfer is currently in progress.

errcode% is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes
None.

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM qstatus AS INTEGER
' Number to hold the quick status work
brdnum = 0 ' First GPIB Board
errnum = 0
' *** Return Status Word ***
CALL KQUIKSTAT(brdnum, errnum, qstatus)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
'
if (qstatus AND &H10)
THEN PRINT "NO SRQ DETECTED" ELSE PRINT "SRQ"
```

KREMOTE

Purpose Forces the GPIB bus device(s) to the remote mode (ignore the bus).

☞ *NOTE:* The KM-488-DD must be a System Controller to execute this command.

Syntax `CALL KREMOTE(boardnr%, errcode%, mygads(0))`

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads (n)` is an array of **`n`** (**`n`** can range from 1 to 15) GPIB bus addresses to be accessed. **`mygads (n)`** is comprised of **`n`** structures of type **GADS**. You must tailor **`mygads`** to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

Returns **`errcode%`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KREMOTE (cont')

Programming Notes

1. If **mygads** does not specify an address (i.e., **noadd** is used), then the GPIB REN (Remote Enable) line is asserted. Devices will not be in the remote mode until addressed to listen. If **mygads** contains address then those devices will also be addressed to listen, so they will be put in remote.

2. It is good programming practice to issue a Local Lockout to prevent the device(s) from being returned to Local mode.

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM DMMadd(2) AS GADS
' this structure will hold the primary and
' secondary addresses of the devices
DMMadd(0).primad = 12  ' first primary address = 12
DMMadd(0).secad = -1  ' first secondary address = NONE
DMMadd(1).primad = -1
' second primary address = NONE and NONE to follow
brdnum = 0 ' First GPIB Board
errnum = 0
'Assert REN + Address Device At Address 12 To Listen
CALL KREMOTE(brdnum, errnum, DMMadd(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KREQUEST

Purpose Sets the Serial Poll Response of a KM-488-DD which is a Peripheral.

Syntax **`CALL KREQUEST(boardnr%, errcode%, sp%)`**

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`sp%` is an integer in the range 0 to 255 which represents the serial poll response of the device. This integer is of the following format:

Bit	7	6	5	4	3	2	1	0
	DIO8	rsv	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
rsv	If this bit is 1, the KM-488-DD will generate a Service Request (assert SRQ).

Returns **`errcode%`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The Active Controller can read the value of **`sp`** by serial polling the KM-488-DD. This will clear bit 6, if it was set.

2. Use KQUIKSTAT or KSPOLL to check if the Peripheral has been serial polled (checks the status of the SRQ bit).

KREQUEST (cont'd)

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM srqreq AS INTEGER
' Integer to hold byte to request service
brdnum = 0 ' First GPIB Board
'
' Set RSV But Don't Set Any Other Bits In Status Byte
srqreq = 64
errnum = 0
CALL KREQUEST(brdnum, errnum, srqreq)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KRESET

Purpose Performs a "warm" reset of the KM-488-DD and the GPIB bus.

Syntax **`CALL KRESET(boardnr%, errcode%)`**

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **`errcode%`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The KM-488-DD is reset to the following conditions:

   - STOP
   - DISARM
   - CONFIG
   - ABORT
   - BUFFER INCREMENT
   - DMA SINGLE
   - ERROR ON
   - FILL NULL
   - LOCAL
   - REQUEST 0 (If Peripheral)
   - Clear CHANGE, TRIGGER, and CLEAR STATUS
   - TIMEOUT 10

   It also clears all error conditions.

2. If the KM-488-DD is the System Controller, it will assert the GPIB IFC (Interface Clear) line for at least 500 µsecs.

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
brdnum = 0 ' First GPIB Board
'
' *** Reset Driver ***
errnum = 0
CALL KRESET(brdnum, errnum)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KRESUME

Purpose
Initiates data transfers between two non-Active Controller GPIB devices, by unasserting the ATN line. <u>The KM-488-DD must be an Active Controller in order to use this function.</u>

Syntax
`CALL KRESUME(boardnr%, errcode%)`

Parameters
`boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns
`errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes
This routine is normally called after a KSEND has addressed a talker and a listener. (See KSEND description.)

Example
```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
brdnum = 0 ' First GPIB Board
'
' *** Drop ATN Line After A Send Command ***
errnum = 0
CALL KRESUME(brdnum, errnum)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KSEND

Purpose Sends GPIB commands and data from a string.

Syntax `CALL KSEND(boardnr%, errcode%, sendstr$, stringlen%)`

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`sendstr$` is a string which includes any of the following subcommands: CMD, UNT, MTA, MLA, TALK, LISTEN, DATA, and EOI. Multiple subcommands may be specified; however, the length of the entire SEND command string cannot exceed 255 characters. The subcommands are as follows:

CMD - Sends information with the ATN line asserted. This indicates to the bus devices that the characters are to be interpreted as GPIB bus commands. The command may be sent as either a quoted string (e.g., **`'COMMAND'`**) or as a numeric equivalent of an individual ASCII character (e.g., `13` decimal or `&H0D` hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **CMD** `67,79,77,77,65,78,68`).

An EOI cannot be sent with this subcommand, because an EOI with ATN asserted would initiate a Parallel Poll.

UNT - Untalks all devices by sending a GPIB UNTALK command with ATN asserted.

UNL - Unlistens all devices by sending the GPIB UNLISTEN command with ATN asserted.

MTA - Designates the KM-488-DD to address itself to talk by sending the GPIB MTA (My Talk Address) command with the ATN line asserted.

KSEND (cont'd)

MLA - Designates the KM-488-DD to address itself to listen by sending the GPIB MLA (My Listen Address) command with the ATN line asserted.

TALK addr - Addresses another GPIB device or KM-488-DD to talk by sending a GPIB TAG (Talk Address Group) command with the ATN line asserted. **addr** is an integer representing the GPIB BUS device address of the device to talk. This integer ranges from 00 to 30 decimal.

LISTEN addr - Addresses another GPIB device(s) or KM-488-DD('s) to listen by sending a GPIB LAG (Listen Address Group) command with ATN asserted. **addr** is an integer representing the GPIB BUS device address of the device(s) to talk. This integer ranges from 00 to 30 decimal. Multiple listeners can be specified.

DATA - Sends information with the ATN line unasserted. This indicates to the bus devices that the characters are to be interpreted as data. This is the technique which is to be used to send device-dependent commands. (See the IEEE-488 Tutorial in Appendix C for more information.) The data may be sent as either a quoted string (i.e., **'DATA'**) or as a numeric equivalent of an individual ASCII character (i.e., **13** decimal or **&H0D** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **DATA 68,65,84,65**).

This sub-command is useful when you are sending commands which are unique to your interface.

KSEND (cont'd)

EOI - Sends information with the ATN line unasserted. EOI will be asserted when the last character is transmitted. This information is interpreted as GPIB bus data and may be sent as either a quoted string (e.g., **'xyz'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **&H0D** hex for a Carriage Return). Hex values must be preceded by **&H**. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **EOI 120,121,122**).

stringlen% is an integer between 0 and 255 which represents the length of the SEND string.

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Typically, KSEND is used to send commands to initiate communication directly between peripheral devices. For example, the KM-488-DD driver might talk a scope and send a trace to a listened plotter.
2. The maximum length of the SEND command, including any subcommands, is 255 characters. To SEND large amounts of data, use multiple SEND commands.
3. SEND should only be used when a non-conforming device requires a special command sequence or a non-standard GPIB command. Do not use the SEND command unless you are extremely familiar with GPIB.
4. The KM-488-DD must be the Active Controller to send commands. Any KM-488-DD can send data.
5. If a DATA subcommand is not included in the KSEND string, be sure to call KRESUME immediately after the KSEND. This is necessary because the ATN line must be dropped so that the transfer will proceed.
6. Do not include the word SEND within the SEND string as you might do in the File I/O SEND Command.

KSEND (cont'd)

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM StrCnt AS INTEGER  ' Length of Str
DIM Str AS STRING      ' String to hold returned message
brdnum = 0 ' First GPIB Board
'
'Address Device @ 2 To Listen + Device @ 12 to talk
Str = "UNT UNL LISTEN 02 TALK 12"
StrCnt = LEN(Str)
errnum = 0
CALL KSEND(brdnum, errnum, Str, StrCnt)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
'
'Drop ATN Line After A Send Command
CALL RESUME(brdnum, errnum)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KSPOLL

Purpose If the KM-488-DD is an Active Controller, KSPOLL will check for an SRQ or conduct a serial poll. If the KM-488-DD is a Peripheral, KSPOLL will check if the KM-488-DD's SRQ has been serviced.

Syntax `CALL KSPOLL(boardnr%, errcode%, spr%, mygads(0))`

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads (n)` is an array of **`n`** (**`n`** can range from 1 to 15) GPIB bus addresses to be accessed. **`mygads (n)`** is comprised of **`n`** structures of type **`GADS`**. You must tailor **`mygads`** to your application. The **`GADS`** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

Returns **`errcode%`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`spr%` is an unsigned integer or an array of unsigned integers with an entry for each address in **`mygads`**. The value of the integer is 0 to 255 and corresponds to a byte with the following bitmap:

Bit	7	6	5	4	3	2	1	0
	DIO8	SRQ	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

KSPOLL(cont'd)

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
SRQ	If this bit is 1, the SRQ (Service Request) line will be asserted to request servicing. Otherwise, SRQ will not be asserted.

Bit 6 has the special significance of indicating an SRQ pending. The significance of the other bits will be application dependent.

Programming Notes

1. The most common use of KSPOLL is for the KM-488-DD, as an active controller, to issue SPOLL with **mygads** containing the address of a single device. This addresses and serial polls the device and, upon return, **spr** contains the serial poll response of the polled device. If the SRQ bit in **spr** is set, the device had issued an SRQ. The other bits in **spr**are device-dependent. Serial polling a device which is currently asserting SRQ will cause the device to unassert SRQ.

   The string returned by KSTATUS or the integer status word set by KQUIKSTAT can be checked to determine the presence of an SRQ before a serial poll is conducted. If only one device is asserting SRQ, the effect of issuing KSPOLL will be to clear the internal SRQ pending "bit", even if the polled device is not issuing the SRQ. It is also the case when multiple devices are asserting SRQ and a device not currently asserting SRQ is polled. In this case, a subsequent use of KSTATUS or KQUIKSTAT will not reveal a pending SRQ.

   To aid in identifying which bus device(s) is currently requesting service, a KM-488-DD which is the Active Controller can serial poll as many as 14 devices by issuing a KSPOLL call with **mygads** containing the address of more than one device. In this case, **spr** should be an array of unsigned integers with one element for each address in **mygads**. In this case, KSPOLL will serial poll each addressed device and return the serial poll bytes in the **spr** array. This is a faster way for discerning the source of an SRQ among several devices.

KSPOLL (cont'd)

2. If the KM-488-DD is a peripheral and KSPOLL is called with **mygads** equal to the no address structure (**noadd**), **spr** will be the serial poll response (**sp**) set by KREQUEST - with the possible exception of the SRQ bit, which may have disappeared as a result of an Active Controller having serial polled the KM-488-DD since the last KREQUEST call. In the instance of an Active Controller serial polling a peripheral KM-488-DD, the peripheral KM-488-DD unasserts the SRQ line. For an example, see KREQUEST. The state of the SRQ bit can also be determined from the string returned by KSTATUS or the integer status word set by a KQUIKSTAT call.

3. If the KM-488-DD is the Active Controller and issues KSPOLL with **mygads** equal to **noadd**, **spr** will be 64 if a device is asserting SRQ and 0 if not. This same result could have been determined from KQUIKSTAT or KSTATUS.

   Note that if this call is issued when a SRQ is pending, it internally "clears" the SRQ "pending" bit, even though the requesting device has not been polled and is still issuing an SRQ. Thus, a subsequent KSPOLL, KQUIKSTAT, or KSTATUS call will not reveal a pending SRQ.

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM sbyt(3)   AS INTEGER  ' Array to hold result of spoll
DIM devadd(3) AS GADS
' this structure will hold the primary and
' secondary addresses of the devices
devadd(0).primad = 5  ' first primary address = 5
devadd(0).secad = -1  ' first secondary address = NONE
devadd(1).primad = 7  ' second primary address = 7
devadd(1).secad =  -1 ' second secondary address = NONE
devadd(2).primad = -1 'no third device
brdnum = 0              ' First GPIB Board
errnum = 0
' serial poll devices in devadd(). results go in sbyt()
CALL KSPOLL(brdnum, errnum, devadd(0), sbyt(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KSTATUS

Purpose Returns a character string describing the current operating state of the KM-488-DD.

Syntax `CALL KSTATUS(boardnr%, errcode%, stat$)`

Parameters **`boardnr%`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **`errcode%`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`stat$` is a character string which describes the current operating state of the KM-488-DD. The string is of the format shown in Table 8-2. **`stat$`** should be 75 bytes long in order to contain the longest message that might be returned. If it is too short, data will be written over other regions of memory .

Table 8-2. Status String

Starting Col	# of Cols.	Name and Description

1	1	Operating Mode - Tells if the KM-488-DD is acting as an *Active Controller* or *Peripheral*. Can be C or P where:
		
		C = The KM-488-DD is an *Active Controller*
		P = The KM-488-DD is a *Peripheral*

3	2	Bus Address - Gives the IEEE-488 Bus Address of the KM-488-DD. This is a two-digit decimal integer ranging from 00 to 30.
6	2	Addressed Status Change - Indicates if the device has changed addressed state, i.e., if it has cycled between the *Talker*, *Listener*, or *Active Controller* states. This is reset whenever a STATUS is read. This can be one of the following: G0 = There has not been a change in the addressed status. G1 = There has been a change in the addressed status.

9	1	Current Addressed State - Indicates if the KM-488-DD is currently acting as a *Talker* or *Listener*, or is *Idle*. Can be T, L, or I, where:
		
		T = The KM-488-DD is a Talker
		L = The KM-488-DD is a Listener
		I = The KM-488-DD is Idle

11	2	Service Request - Represents the current internal SRQ status where: S0 = SRQ not asserted S1 = SRQ asserted If the KM-488-DD is in the *Active Controller* mode, the internal SRQ state indicates if a device is asserting SRQ. If the KM-488-DD is acting as a *Peripheral*, the internal SRQ state is set by using the KREQUEST call. It is cleared by a serial poll from the Active Controller and indicates it is asserting SRQ.

KSTATUS(cont'd)

Table 8-2. Status String

Starting Col	# of Cols.	Name and Description
14	3	Error Code - Indicates whether or not an error has occurred. The three characters consist of the letter 'E' followed by a two digit error code. (A list of these error codes is provided in Appendix G.) An error code of 00 indicates no error has occurred.
18	2	Triggered State - Indicates if a IEEE-488 trigger command has been received or not. This is not updated during DMA CONTINUE transfers. Triggered State values are: T0 = No trigger command has been received. T1 = A trigger command has been received.
21	2	Cleared - Indicates if the IEEE Clear command has been received or not. This is not updated during DMA CONTINUE transfers. Cleared values are: C0 = No Clear command has been received. C1 = A Clear command has been received.

24	2	Transfer Status - Indicates if a data transfer is in progress. Values which may appear are:
		
		P0 = No transfer
		P1 = DMA CONTINUE Transfer occurring
		P2 = DMA AUTOINITIALIZE Transfer occurring

27	up to 45	Error Message Text - Contains the Error Message Text associated with the given error code (EXX). These error messages are listed in Appendix G.

Programming Notes If the KM-488-DD is the Active Controller, the Service Request Status may be cleared (i.e, S0) as a result of a KSPOLL, even when the SRQ line is still asserted. See the discussion in KSPOLL.

KSTATUS (cont'd)

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
DIM Str AS STRING      ' String to hold returned message
brdnum = 0 ' First GPIB Board
errnum = 0
Str = STRING$(75, 32)
'*** Loop and check status for SRQ ***
'
chkstatus:
CALL KSTATUS(brdnum, errnum, Str) ' Request status
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
IF (MID$(Str,12,1) <> "1") GOTO chkstatus
```

KSTOP

Purpose Stops a DMA CONTINUE transfer.

Syntax `CALL KSTOP (boardnr%, errcode%)`

Parameters **boardnr%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the DMA CONTINUE transfer has been completed, a KSTOP will have no effect.

2. Use the KBUFD call to determine the actual number of characters which were transferred before the KSTOP took effect.

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
brdnum = 0 ' First GPIB Board
'
' *** Stop DMA ***
errnum = 0
CALL KSTOP(brdnum, errnum)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KSWAP

Purpose Exchanges the bytes contained in the elements of an integer array.

Syntax `CALL KSWAP(array%(0),count%)`

Parameters **array%** is the name of the integer array whose bytes are to be "swapped".

count% is an integer representing the number of elements contained in the integer array (**array%**).

Returns **array%** will contain the newly formatted integer array.

Programming Notes

1. This routine exchanges the bytes within individual elements of an integer array. Byte swaps are necessary when communicating with devices that transfer binary data (real or integer numbers) in high byte-low byte order.

   When such a device transfers data into a QuickBASIC integer array, your program must swap the bytes in each array location, as QuickBASIC will expect integers to be in low byte-high byte order in memory. Likewise, you must KSWAP data within an integer array that is being sent to a device expecting high byte-low byte data. The KM-488-DD driver is unable to perform the swap during a transfer, as it treats PC memory as a linear sequence of bytes.

Example

```
DIM  WORKBUFF%(100)
DIM  COUNT AS INTEGER

COUNT = 101            ' WORKBUFF%(100) HAS 101 ELEMENTS

FOR X = 0 TO 100       ' CALCULATE A SINE WAVE
WORKBUFF%(X) = 10 * SIN(X)
NEXT X

CALL KSWAP(WORKBUFF%(0), COUNT) ' SWAP THE INTEGERS

' ... OUTPUT DATA TO DEVICE NEXT ...
```

KTERM

Purpose Changes the default GPIB bus terminator(s) to be used in ENTER and/or OUTPUT calls.

Syntax `CALL KTERM(boardnr%, errcode%, termlist%(0))`

Parameters `boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`termlist%(0)` is an integer array which contains the new default GPIB bus terminators for the KM-488-DD device specified by `boardnr%`. See Chapter 6 for more information regarding the use of GPIB bus terminators. `termlist(4)` is comprised of the following elements:

`termlist(0)`: Input/Output GPIB Terminators. Possible values are:

-1	Change the Input Terminators.
0	Change both Input and Output Terminators.
1	Change the Output Terminators.

`termlist(1)`: EOI detection/assertion. Possible values are:

0	No EOI assert/detect.
1	Assert/Detect EOI.

KTERM (cont'd)

termlist(2): **1st GPIB bus terminator.** Possible values are:

-1(&HFFFF)	Don't use first or second GPIB bus terminator.
0 to 255	Use this ASCII character as the first GPIB bus terminator instead of the default first GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

termlist(3): **2nd GPIB bus terminator.** Possible values are:

-1(&HFFFF)	Don't use second GPIB bus terminator.
0 to 255	Use this ASCII character as the second GPIB bus terminator instead of the default second GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. KTERM will change the default bus terminators set during software installation as directed by **termlist(0)**.

KTERM (cont'd)

Example

```
DIM brdnum AS INTEGER
' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
DIM modterm(4) AS INTEGER
' used to modify the default terminators
brdnum = 0 ' First GPIB Board
'
' Terminator Arrays
'
modterm(0) = 0' reset both input & output terminators
modterm(1) = 1 ' Enable EOI assert/detect
modterm(2) = -1' no terminator characters
modterm(3) = -1' no terminator characters
' *** Initialize Driver Terminators ***
'
errnum = 0
CALL KTERM(brdnum, errnum, modterm(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KTO

Purpose Changes the timeout period.

Syntax `CALL KTO(boardnr%, errcode%, tval)`

Parameters `boardnr%` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`tval` is a long integer which contains the number of 55 ms. timer ticks to allow before a time-out.

Returns `errcode%` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If no timeout period is specified, the default of 10 seconds will be used.

2. To suppress Timeout checking, set `tval` to 0.

3. If a DMA CONTINUE transfer is in progress, the KM-488-DD will check for timeouts only for the first byte that is transmitted/received. During other types of transfers, the KM-488-DD will check for a time-out between transmission of bytes.

   You must be certain to check that a DMA CONTINUE transfer has been completed.

Example

```
DIM brdnum AS INTEGER
' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates no error
DIM ticks AS LONG       ' Number of ticks to set time
brdnum = 0 ' First GPIB Board
errnum = 0
'Set time out to 15 seconds
ticks = 15000 \ 55      'Where a tick is 55 msecs.
CALL KTO(brdnum, errnum, ticks)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KTRIGGER

Purpose Triggers the specified device(s). The KM-488-DD must be an Active Controller.

Syntax `CALL KTRIGGER(boardnr%, errcode%, mygads(0))`

Parameters **boardnr%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads(n) is an array of n (**n** can range from 1 to 15) GPIB bus addresses to be accessed. **mygads(n)** is comprised of **n** structures of type **GADS**. You must tailor **mygads** to your application. The **GADS** structure is described in section 8.1 and defined in the QuickBASIC include file *HEADER.BI.*

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The devices listed in `mygads` will be triggered.

KTRIGGER (cont'd)

Example

```
'$INCLUDE : 'HEADER.BI'
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
' this structure holds the primary
' and secondary addresses of the devices
DIM DMMadd(2) AS GADS
DMMadd(0).primad = 12  ' first primary address = 12
DMMadd(0).secad = -1  ' first secondary address = NONE
DMMadd(1).primad = -1
' second primary address = NONE and NONE to follow
brdnum = 0 ' First GPIB Board
errnum = 0
'Trigger Device With GPIB Address 12
CALL KTRIGGER(brdnum, errnum, DMMadd(0))
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

KUNPACK

Purpose Converts an integer array to consecutive 16-bit integers.

Syntax **`CALL KUNPACK (source%(0),dest%(0),count%)`**

Parameters **`source%`** is the name of the integer array which contains the data to be converted.

`dest%` is the name of the integer array in which the converted data is to be stored.

`count%` is an integer which represents the number of elements comprising the **`source%`** array. (The number of elements needed in the **`dest%`** array is **`count%/2`**.)

Returns **`dest%`** will contain the newly converted data.

Programming Notes

1. Both **`source%`** and **`dest%`** must be declared as COMMON-SHARED. The KM-488-DD driver assumes that each of these arrays resides in QuickBASIC's default data segment.

Example

```
DIM numd AS INTEGER     ' Number of bytes to transfer
DIM dmain%(514)          'Array to hold DMA input waveform
DIM workb%(1028)         'Working array
COMMON SHARED dmain%(), workb%()
numd = 1028             'Number of bytes to transfer
'Convert the packed byte array into a word
' (integer) working array
CALL KUNPACK(VARPTR(dmain%(0)), VARPTR(workb%(0)), numd)
```

KWAITC

Purpose Forces the KM-488-DD driver to wait until a DMA CONTINUE transfer has been completed before returning control to the application program.

Syntax `CALL KWAITC(boardnr%, errcode%)`

Parameters **boardnr%** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **errcode%** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Time out checking, if enabled, is performed while KWAITCing.

2. This is an explicit wait. The following calls perform an implicit wait:

- KABORT
- KCLEAR
- KENTER
- KLOCAL
- KLOCAL LOCKOUT
- KOUTPUT
- KPASS CONTROL
- KPPOLL
- KPPOLL CONFIG
- KPPOLL DISABLE
- KPPOLL UNCONFIG
- KREMOTE
- KREQUEST
- KRESUME
- KSEND
- KSPOLL
- KTRIGGER

Example

```
DIM brdnum AS INTEGER  ' Number of board to be accessed.
DIM errnum AS INTEGER
' Error code returned if any. 0 indicates  no error
brdnum = 0 ' First GPIB Board
'
'Code which issued a DMA with CONTINUE followed by
' additional code
' Wait For DMA To Finish
errnum = 0
CALL KWAITC(brdnum, errnum)
IF errnum THEN PRINT "IEEE Error ";errnum:STOP
```

CHAPTER 9 USING THE CALL INTERFACE IN MICROSOFT PASCAL

Versions Supported

Microsoft PASCAL versions 3.0 and higher

File Header

Be sure to include the following lines within your program:

```
{$include: 'mp488dd.inc'}
```

mp488dd.inc is an "include" file with function/procedure prototypes, structure definitions, and constants pertinent to the MICROSOFT PASCAL call interface.

km488ifc.obj is a linkable module containing the functions and procedures defined in *mp488dd.inc.* This module dispatches KM-488-DD calls into the KM driver.

If either of these files resides in a directory other than the one in which you are compiling your program, be sure to specify a complete path to each file.

Compiling The Program

Typically, the following statements will suffice for compiling your program provided you have adhered to the instructions in the file header section:

```
PAS1 MYPROG;
PAS2
LINK MYPROG + KM488IFC;
```

Software Configuration

A number of KM-488-DD configuration parameters are set via the *CONFIG.DAT* file. (See Chapter 2 and Appendix F.) These govern the default settings of the GPIB input and output bus terminators, lightpen interrupt enable, device timeout period, and the KM-488-DD's Base Address. The defaults for these are listed in Table 9-1. There are other defaults you may have to re-program if you are using File I/O Commands in the same program as the library interface routines (calls).

Table 9-1. Default KM-488-DD Operating Parameters

Parameter	Default
Device Timeout	10.0 seconds
GPIB Bus Input Terminator	CR LF with no EOI
GPIB Bus Output Terminator	CR LF with no EOI
EOL Terminators	CR LF

The KCONFIG call can be used to reset the GPIB address, and the GPIB input and output bus terminators.

The KTERM call can be used to change the default GPIB bus terminator settings.

The KTO call can be used to change the default device timeout value.

Programming Notes

1. Any values passed into a call in a variable may also be passed as a constant or as the returned value of a function.

2. Parameters which are also used to return values must be declared as variables.

3. Any of the KM-488-DD routines which are used to receive data require that a string or array be declared to store the received data. The length of the string or size of the array should be sufficient to store the number of bytes that are expected. In addition, when a KM-488-DD routine requires a parameter specifying the maximum number of data bytes to be returned, it is extremely important that the amount of storage space allocated is at least as great as this maximum length parameter. Otherwise, data may be stored into memory which has been allocated for use by other parts of your program, or for use by DOS. This could lead to erroneous operation and possibly a system crash.

4. In Microsoft PASCAL, LSTRINGs are actually a special type of character array. The first byte of the array is used to store the number of bytes contained within the string. Hence, strings may range from 0 to 255 bytes in length and the KM-488-DD routines which pass data to or from strings are limited to 255 bytes maximum.

When values are returned to the calling program via a LSTRING variable, the string length must be calculated and placed in location 0 of the string, where Microsoft PASCAL normally stores it. The string length must be calculated in this manner as the KM-488-DD driver is unaware of which language is issuing the current call.

5. Do not name the variables in your application program with the same name as any of the KM-488-DD routines.

6. Do not assign a program name which is the same name as any of the KM-488-DD routines.

9.1 CALL DESCRIPTION FORMAT

Each call description is divided into several sections as described below.

Syntax This section describes the syntax required by each call. The following rules are used in this section.

Call names and input parameters appear in **`this typeface`**.

Parameters This section describes the input parameters. In some instances, a structure or array of multiple parameters may be specified (for example, multiple device addresses).

`mygads[n]` is used in many calls. This is an array of **`n`** (1 to 15) structures of type **`gads`** which contain the GPIB address information required by the call. This GPIB address structure consists of:

```
TYPE
     gads = RECORD
     primad : INTEGER;          (*primary address 0-30*)
     secad : INTEGER;           (*secondary address 0-31*)
     END;
```

When **`secad`** is set equal to -1, it indicates that no secondary address is applicable. When **`primad`** is set equal to -1, it terminates the array.

You must tailor **`mygads`** to your application. The **`mygads`** structure is defined in the include file *mp488dd.inc*. An example of a structure with two devices is:

```
EXAM : ARRAY [0..2] OF GADS;

exam.primad[0] := 3 (* first device - pri addr 3*)
exam.secad[0] := -1 (* and no sec addr      *)
exam.primad[1] := 15 (* second device - pri addr 15 *)
exam.secad[1] := 2  (* and sec addr 2 *)
exam.primad[2] := -1 (* there are only two devices *)
```

If the KM-488-DD is a Peripheral (in which case the call will not require a GPIB bus address), you will want to use the following **mygads** structure:

```
VAR
    no_add : ARRAY [0..2] OF GADS;
. . .
BEGIN
   no_add.primad[0] := -1  (* there are no devices in group *)
```

Returns This part of the callable description describes any data which will be returned after execution of the callable has been completed.

err_code - All calls except KDIAGON and KDIAGOFF return an integer error code which will be 0 if no error was detected. It is good practice to check **err_code** after each call and provide a routine to handle errors if they arise.

Programming Notes This section lists any special programming considerations for the routine.

Example This section gives programming examples.

KABORT

Purpose This routine allows the KM-488-DD to take control of the GPIB bus by forcing all other devices on the bus to the idle state. The KM-488-DD must be the Active Controller.

Syntax

```
VAR

     board_nr : INTEGER;
     err_code : WORD;
...
BEGIN
...
kabort(board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the KM-488-DD is both the Active and System Controller, KABORT asserts the IFC line to retain control of the bus. Otherwise, the KM-488-DD will assert the ATN line, and then untalk and unlisten all devices.

Example

```
CONST
     CARDNUM    = 0 ;   (* Card Number Parameter For Calls *)
VAR
     ErrorFlag : WORD; (* Word To Hold Error Code *)
BEGIN
kabort(CARDNUM, ErrorFlag);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KARM

Purpose Defines the conditions under which the lightpen status becomes true.

Syntax

```
VAR
     board_nr  : INTEGER;
     arm_code  : INTEGER;
     err_code  : WORD;
...
BEGIN
karm(board_nr, err_code, arm_code);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

arm_code is a 16-bit integer describing on which conditions a light pen interrupt may occur. The integer is of the format:

Bit	15-9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC	**Address Status Change.** If this bit is set to 1, the lightpen status will become true when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller). This encompasses all the conditions marked with an asterisk (*) below.
CIC*	**Controller in Charge.** If this bit is set to 1, the lightpen status will become true when the control is passed to the KM-488-DD and it becomes the Active Controller.
PER*	**Peripheral.** If this bit is set to 1, the lightpen status will become true when the KM-488-DD passes control to another device and becomes a Non-Active Controller.
DET	**Device Triggered.** If this bit is set to 1, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.

KARM (cont'd)

SRQ	**Internal SRQ.** If this bit is set to one, the lightpen status will become true when the KM-488-DD's internal Service Request Bit (SRQ) is set.
DEC	**Device Cleared.** If this bit is set to one, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA*	**Listen Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Listener and can accept data from the GPIB.
TA*	**Talk Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Talker and can output data to the GPIB.
IDLE*	**Idle.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is unaddressed as a Talker or a Listener.

☞ *NOTE:* The conditions marked by an asterisk (*) will force the lightpen status to true only when the KM-488-DD's address status has changed. This is indicated by the state of the Address Change Bit in the Status byte. See the kstatus or kquikstat routine descriptions for more information.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The driver must be configured for lightpen emulation at load time via the *CONFIG.DAT* "pipe" file, as the default for lightpen emulation is off.

KARM (cont'd)

Example

```
CONST
   CARDNUM    = 0 ;     (* Card Number Parameter For Calls *)
   SRQENAB    = 16 ;    (* BIT 4 -- SRQ Detection *)
VAR
     ErrorFlag : WORD;         (* Word To Hold Error Code *)
BEGIN
(*
  *** Enable Light Pen Interrupt Detection On SRQ ***
*)
karm(CARDNUM, ErrorFlag, SRQENAB);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KBUF

Purpose Selects in which direction the memory is to be addressed for both DMA and non-DMA buffered transfers.

Syntax

```
VAR
     dir : INTEGER;
     board_nr : INTEGER;
     err_code : WORD;
...
BEGIN
kbuf(board_nr, err_code, dir);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`dir` selects the direction in which the buffer is to be read. If `dir` is equal to 1, the buffer address will be incremented. If `dir` is equal to 0, the buffer address will be decremented.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the buffer's address is to be incremented during a bus I/O operation, be sure to use the lowest address in the designated buffer. Likewise, if it is to be decremented, select the highest address.

KBUF

Example

```
CONST
  CARDNUM    = 0 ;    (* Card Number Parameter For Calls *)
  DECREMENT  = 0 ;    (* Decrement Memory Address on DMA *)
VAR
  ErrorFlag : WORD;   (* Word To Hold Error Code *)
BEGIN
(*
  *** Set Direction Of Memory Address Change With DMA ***
*)
kbuf(CARDNUM, ErrorFlag, DECREMENT);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KBUFD

Purpose Returns the number of characters transferred in the last buffered transfer or, in the case of DMA transfers, the current transfer.

Syntax

```
VAR
     board_nr : INTEGER;
     err_code : WORD;
     bcount : INTEGER4;
BEGIN
...
     kbufd(board_nr, err_code, bcount);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`bcount` represents the number of characters which were transferred during the last buffered transaction. This will be a long integer (32- bits) in the range 0 to 1,048,575 (0 to 20^{20}-1).

Programming Notes

1. If the data was transferred in the DMA CONTINUE mode, the value returns the number of characters transferred thus far.

2. It is useful to call this routine after a buffered KENTER has been performed. This will help you to determine if the expected number of characters has been received and/or the transfer has terminated as the result of a detected terminator.

KBUFD

Example

```
CONST
   CARDNUM    = 0 ;      (* Card Number Parameter For Calls *)
VAR
    ErrorFlag : WORD; (* Word To Hold Error Code *)
   XferCount : INTEGER4;
  (* Number Of Bytes Moved In Last Transfer *)
BEGIN
(*
  Return Number Of Bytes In Last GPIB BUS Data Transfer
*)
kbufd(CARDNUM, ErrorFlag, XferCount);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KCLEAR

Purpose The KM-488-DD must be an Active Controller. Resets device(s) to their power-up parameters.

Syntax

```
VAR
     board_nr  : INTEGER;
     err_code  : WORD;
     mygads  : ARRAY[0..n] OF GADS;
BEGIN
...
kclear(board_nr, err_code, mygads);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] represents the GPIB bus address of the device to be cleared. **mygads** is an array of **n** (**n** can range from 1 to 15) structures of type gads. See section 10.1 for more information.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If **mygads** contains no addresses, then the universal GPIB DCL (Device Clear) command is issued and all devices will be cleared.

2. If **mygads** contains addresses, then those devices specified will be listened and sent a GPIB SDC (Select Device Clear) Command.

KCLEAR

Example

```
CONST
  CARDNUM     = 0 ;     (* Card Number Parameter For Calls *)
VAR
  add_12     : ARRAY[0..1] OF gads;
(* Address Group (One Device At 12) *)
   ErrorFlag : WORD;
(* Word For Code Returned By Calls  *)
(*
 Address Structure For Meter At GPIB Address 12
*)
add_12[0].primad:=12;
    (* 1st Instrument Has GPIB Address 12*)
add_12[0].secad:=-1;
     (* -1 Indicates 1st Instrument Has No Secondary Add *)
add_12[1].primad:=-1;
    (* -1 Indicates No 2nd Instrument In Group *)
BEGIN
(* Return The Device To Its Power Up Condition
*)
kclear(CARDNUM, ErrorFlag,add_12[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KCONFIG

Purpose Resets the KM-488-DD to its default operating parameters or to those specified.

Syntax

```
VAR
     board_nr : INTEGER;
     err_code : WORD;
     cfg_string : LSTRING(255);
     stringlen : INTEGER;
...
BEGIN
     kconfig(board_nr, err_code, cfg_string, stringlen);
...
```

Parameters **`board_nr`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`cfg_string` is a user-defined string. This string can be comprised of any of the following options:

`/T` *`[term term EOI]`* Sets the input and output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **`/T`**, the GPIB bus terminators will be reset to their defaults.

`term` is an optional parameter which represents a terminator byte to be used. This terminator byte can be represented as any of the following:

`$char` where **`char`** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **`char`** must be preceded by a dollar sign ($).

KCONFIG (cont'd)

Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.

CR	This represents the *Carriage Return* character (13 decimal, 0D hex).
LF	This represents the *Line Feed* character (10 decimal, 0A hex).
'X	where **X** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **'B** represents the character *B*.

EOI (End or Identify) is an optional GPIB BUS terminator. If included, the KM-488-DD will detect/assert the GPIB bus EOI line. (This would indicate that the last character has been sent.) Data will continue to be read until this terminator, a valid terminator sequence, or both are detected.

/TI *[term term EOI]*	Sets the input GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TI**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/TO *[term term EOI]*	Sets the output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TO**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/E *[term term]*	Sets the input and output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **term** is described above.

KCONFIG (cont'd)

`/EI` *[term term]*	Sets the input EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **`term`** is described above.
`/EO` *[term term]*	Sets the output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.)
`/N` **`name addr`** *[sec_addr]*	Configures a named device. Sets its address to the given value and its GPIB bus terminators to the current GPIB bus terminator settings. (Note that this should only be used if the application program also includes File I/O Commands utilizing the named device configurations.) **`name`** is the name of the device. **`addr`** is the primary GPIB address (00 to 30). **`sec_addr`** is the secondary GPIB address (00 to 31).

☞ ***NOTE:*** Do not create a named device with the same name as an existing directory on the current working disk.

If **`cfg_string`** is empty, then the configuration will revert to the installation configuration.

`stringlen` is an integer from 0 to 255 which gives the length of the **`cfg_string`** string.

Returns **`err_code`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KCONFIG (cont'd)

Programming Notes

1. The **/E**, **/EI**, **/EO**, and **/N** options will only take effect if File I/O Commands are used within the same application program as the callables.

2. If you are converting an application program previously written using the File I/O commands to callables, be sure to note that the CONFIG string for the KCONFIG call does not include the "CONFIG" keyword (i.e., as you would if you were using the CONFIG File I/O command).

3. Primary and Secondary Addresses must be two characters long, e.g. 01 or 0209.

Example

```
CONST
  CARDNUM    = 0 ;
 (* Card Number Parameter For Calls *)
VAR
  ConfStr     : LSTRING(70); (* For Command Used In KCONFIG *)
  ErrorFlag : WORD;        (* Word For Code Returned By Calls
*)
BEGIN
(*
  Configure Terminations for Carriage Return, Line Feed With
EOI *)
ConfStr := '/T CR LF EOI';
kconfig(CARDNUM, ErrorFlag,confstr[1],12);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
```

KDIAGON

Purpose Turns file diagnostics on. (Default is off.)

Syntax

```
VAR
     filename : LSTRING(255);
...
BEGIN
     kdiagon(filename, strlen(filename));
...
```

Parameters `filename` is a string containing the name of the file to which the diagnostic information is to be written.

`strlen` is the length of the string representing the name of the file.

Returns None.

Programming Notes This will echo the value of the call parameters into a disk file.

Example

```
CONST
   CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
VAR
     tdstr     : LSTRING(255);
(* String To Hold Target Diagnostic File Name *)
BEGIN
tdstr:='DEBUG.DAT';        (* Target Diagnostic File Name *)
(*
 Turn Driver Diagnostics On And Send Info To Disk File
*)
kdiagon(tdstr[1], 9);      (* File Name And File Name Length *)
END.
```

KDIAGOFF

Purpose Temporarily disables file diagnostics.

Syntax `kdiagoff()`

Parameters None.

Returns None.

Programming Notes None.

Example

```
BEGIN
(*
  *** Disable File Diagnostics
*)
kdiagoff();
END.
```

KDISARM

Purpose Resets previously set conditions which cause lightpen interrupts.

Syntax

```
VAR
     board_nr : INTEGER;
     arm_code : INTEGER;
     err_code : WORD;
...
BEGIN
     kdisarm(board_nr, err_code, arm_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`arm_code` is a 16-bit integer describing which conditions should be reset. The integer is of the format:

Bit	15 - 9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC **Address Status Change.** If this bit is set to 1, light pen status will not change when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).

CIC **Controller in Charge.** If this bit is set to 1, light pen status will not change when the control is passed to the KM-488-DD and it becomes the Active Controller.

PER **Peripheral.** If this bit is set to 1, light pen status will not change when the KM-488-DD passes control to another device and becomes a Non-Active Controller (Peripheral).

DET **Device Triggered.** If this bit is set to 1, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.

KDISARM (cont'd)

SRQ	**Internal SRQ.** If this bit is set to one, light pen status will not change when the KM-488-DD's internal Service Request Bit (SRQ) is set.
DEC	**Device Cleared.** If this bit is set to one, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA	**Listen Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Talker and can output data to the GPIB.
IDLE	**Idle.** If this bit is set to one, light pen status will not change when the KM-488-DD is unaddressed as a Talker or a Listener.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
CONST
    CARDNUM = 0 ;   (* Card Number Parameter For Calls *)
    SRQENAB = 16 ;  (* BIT 4 -- SRQ Detection          *)
VAR
    ErrorFlag : WORD;        (* Word To Hold Error Code *)
BEGIN
(*
  *** Disable Light Pen Interrupt Detection On SRQ ***
*)
kdisarm(CARDNUM, ErrorFlag, SRQENAB);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KDMA

Purpose Sets DMA transfer mode.

Syntax

```
VAR
     board_nr : INTEGER;
     dma_mode : INTEGER;
     err_code : WORD;
...
BEGIN
     kdma (board_nr, &err_code, dma_mode);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

dma_mode is a 16-bit integer which represents the DMA mode to be used. If no **dma_mode** is given, then single mode without autoinitialization will be assumed. The **dma_mode** integer is interpreted as follows:

Bit	15 - 3	2	1	0
	0	SIN	DEM	AUTO

Where:

SIN SINGLE. When this bit is set to one, the SINGLE mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus and transfers a single byte of data. Control of the bus is then returned to the microprocessor.

DEM **DEMAND.** When this bit is set to one, the DEMAND mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus. The DMA controller retains control of the bus until the DMA request signal is unasserted. Once this signal has been unasserted for more than one microprocessor clock cycle, control of bus is returned to the microprocessor.

KDMA (cont'd)

This mode allows the DMA controller chip to pass data at a slightly faster rate and the microprocessor to access the bus when it is not needed. Note that SINGLE and DEMAND mode can not be invoked simultaneously.

AUTO **AUTOINITIALIZE.** When this bit is set to one, the AUTOINITIALIZE mode is selected. Under normal circumstances, the DMA controller transfers the specified number of bytes to/from the PC memory from the given starting address and terminates when completed. When the AUTOINITIALIZE mode is enabled, the DMA controller will reset the original byte count, reset the initial address, and repeat the transfer again. The AUTOINITIALIZE option is only in effect until the next KENTER or KOUTPUT buffered transfer is completed. Additionally, if a KENTER or KOUTPUT transfer in DMA CONTINUE mode is selected, the AUTOINITIALIZE option will only be in effect for that transfer.

Returns None.

Programming Notes

1. All DMA AUTOINITIALIZE transfers should occur entirely within a single DMA 64 KByte page. So, if a DMA buffer operation involves a transfer of more than 64K or occur across a DMA page boundary, the AUTOINITIALIZE option affects only those bytes in the last transfer occurring within one DMA page.

   Also note that DMA AUTOINITIALIZE transfers specified for buffers that are located in more than a single DMA page are unpredictable.

2. It is impossible for a program to halt a DMA AUTOINITIALIZE operation unless the DMA CONTINUE option is selected. This is because the driver does not return control to a program using non-CONTINUE operations until the transfer completes.

KDMA (cont'd)

Example

```
CONST
    CARDNUM  = 0;      (* Card Number Parameter For Calls *)
    AUTO     = 1;      (* Auto Mode of DMA Transfer *)
    DEMAND   = 2;      (* Demand Mode of DMA Transfer *)
    SINGLE   = 4;      (* Single Mode of DMA Transfer *)

VAR
     ErrorFlag : WORD;        (* Word To Hold Error Code *)
BEGIN
(*
  *** Set Up DMA Mode To Single With Autoinitialize ***
*)
kdma(CARDNUM, ErrorFlag, SINGLE+AUTO);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KENTER

Purpose Allows the KM-488-DD to receive data from a GPIB bus device.

Syntax

```
VAR
  board_nr : INTEGER;
  error_code : WORD ;
  mode : INTEGER;
  count : INTEGER4;
  mygads : ARRAY[0..1] OF gads;
  darray : ARRAY[0..N-1] OF CHAR ;
  term_list: ARRAY[0..2] OF INTEGER;
...
BEGIN
  kenter(board_nr, err_code, darray, count, mode, mygads,
term_list)
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

darray is a character array large enough to accept the returned data. If a non-zero count is specified, the array should be of size **count.** If **count** is 0 and terminator characters are used, then **darray** must be large enough to accommodate both data and terminators returned by the device.

count is a long integer representing the number of characters to be read. Valid counts are within the range 0 to 4294967295 (2^{32}-1) decimal, or from 0 to #FFFFFFFF hex. When 0 is used, the KENTER will stop when the termination specified by **term_list** is satisfied.

mode is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

KENTER (cont'd)

Where:

CONT	CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer. If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked or a pre-specified GPIB BUS input terminator is detected.
DMA	DMA. If this bit is set to one, then DMA will be used in the data transfer.

mygads[n] represents the GPIB bus address of the device to be reset. **mygads** is an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. The **mygads** structure is defined in the include file *mp488dd.inc* and is described in section 10.1.

term_list[3] is an integer array which defines the GPIB bus terminators to be used during the KENTER call. These terminators are used in conjunction with **count** to terminate the input process. Termination choices include whether or not EOI will be detected with the last byte and whether or not a certain byte or sequence of two bytes will be tested for as an end of message code. The default termination scheme is initially set-up by the *CONFIG.DAT* file, and can be modified by calling KTERM or KCONFIG. **termlist[3]** can either specify the use of the default terminators or temporary overrides.

termlist[3] is defined in the *mp488dd.inc* header file. **termlist[3]** is comprised of the following elements:

KENTER (cont'd)

`termlist[0]`: **EOI detection.** Possible values are:

-1(#FFFF)	Use the default EOI setting.
0	Don't Detect EOI.
1	Detect EOI.

`termlist[1]`: **1st GPIB bus terminator.** Possible values are:

-1(#FFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.
0 to 255	Detect this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

`termlist[2]`: **2nd GPIB bus terminator.** Possible values are:

-1(#FFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
0 to 255	Detect this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

KENTER (cont'd)

The following default **termlist** can also be used:

```
VAR
     dtrm : ARRAY[0..2] OF INTEGER;
...
BEGIN
dtrm[0] := -1; (* Use the default EOI choice *)
dtrm[1] := -1; (* Use the default first terminator *)
dtrm[2] := -1; (* Use the default second terminator *)
```

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is a Peripheral, an active controller must address the KM-488-DD as a listener and some other device as the talker, before KENTER is called. In this case, you should use the **no_addr** structure in lieu of **mygads**. This is described in section 10.1.

2. If the KM-488-DD is the Active Controller, it will automatically address itself as a listener and another device defined in **mygads** as the talker. Since only one device can be a talker, the address group **mygads** can contain only one address. If a device has been previously addressed to talk and the KM-488-DD previously addressed to listen, then **mygads** can be **no_add** (See section 10.1) and the bus will not be re-addressed.

3. If **count** is specified as 0, then KENTER will terminate when the terminator(s) specified by **term_list[3]** is encountered.
4. If it is desirable to terminate on **count** only and not use the default terminators, then you must select no terminator overrides in the **termlist** array, in addition to giving the count. In this case, a byte by byte synchronous transfer is executed.

5. If both counts and termination are specified, the KENTER will terminate on whichever condition is encountered first.

6. If the condition(s) for termination is not encountered, KTERM will timeout unless the timeout has been disabled.

Example

```
CONST
     CARDNUM  = 0 ;      (* Card Number Parameter For Calls *)
     DMAXFER  = 1;
      (* Transfer mode: non DMA = 0, DMA without
       continue = 1, DMA with continue = 3 *)
     NUMD     = 1028;    (* Number Of Bytes To Transfer *)
VAR
  ErrorFlag : WORD ;     (* Word For Returned Error Code  *)
  add_05    : ARRAY[0..1] OF gads;
      (* Address Group For Device @ 05 *)
  sdata     : ARRAY[0..NUMD-1] OF CHAR ;
      (* Array For Data From Device    *)
  dtrm      : ARRAY[0..2] OF INTEGER;
      (* Array Of GPIB Terminators For "KENTER" And "KOUTPUT"
*)
BEGIN
(*
 *** Address Structure For Meter At GPIB Address 05
*)
(* Instrument Has GPIB Address 5 *)
add_05[0].primad := 5 ;
(* -1 Indicates No Secondary Address *)
add_05[0].secad  := -1 ;
(* -1 Indicates No Other Instrument  *)
add_05[1].primad := -1 ;
(*
  Structure To Signal:
                        Use Default Terminators
                        Detect EOI On ENTER/Assert EOI On
Output
*)
dtrm[0] := 1 ; (* 1  : Assert EOI (KOUTPUT)/Detect EOI
(KENTER) *)
dtrm[1] := -1 ;       (* -1 : Use Default 1st Terminator    *)
dtrm[2] := -1 ;       (* -1 : Use Default 2nd Terminator *)
WRITELN('ENTER DATA FROM METER');
kenter(0, ErrorFlag, sdata[0], NUMD, DMAXFER, add_05[0],
dtrm[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KERROR

Purpose Enables/Disables display of Error Messages.

Syntax

```
VAR
   board_nr : INTEGER;
   err_code : WORD;
   e_sw : INTEGER;
...
BEGIN
   kerror(board_nr, err_code, e_sw)
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

e_sw is an integer which determines if error message display function will be enabled or disabled. If **e_sw=1**, the error message display is enabled. (This is the default.) If **e_sw=0** is specified, the error message display is disabled.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If KERROR has been used to enable error messages, the message will only reveal that an error has occurred for the KM-488-DD calls. It does not identify what type of error occurred. The KDIAGON routine should be used to help debug applications utilizing the KM-488-DD calls.

2. Programs can also check **err_code** after each call to identify the exact nature of call errors.

KERROR (cont'd)

Example

```
CONST
    CARDNUM     = 0 ;  (* Card Number Parameter For Calls *)
    ERR_ON      = 1 ;  (* Enable Error Display*)
VAR
     ErrorFlag : WORD;         (* Word To Hold Error Code *)
BEGIN
(*
  *** Enable Error Display ***
*)
kerror(CARDNUM, ErrorFlag, ERR_ON);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KFILL

Purpose Defines the KM-488-DD driver's response to a request for data when none is available.

☞ ***NOTE:*** KFILL is provided for users who mix File I/O Commands and Calls in the same program. It only affects inputs performed using the File I/O Commands.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  fill_sw : INTEGER;
...
BEGIN
  kfill(board_nr, err_code, fill_sw)
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`fill_sw` is an integer which represents the fill condition. (The default condition returns a NULL character.) Valid integers are any one of the following:

-1	OFF. This type of response will not return any data characters or a DOS error message, if no input data is available.
-2	ERROR. This type of a response will not return any data characters, but will generate an Error Message. The error message number can then be trapped by calling the KSTATUS routine. (Refer to the KSTATUS routine for more information.)
0 - 255	ASCII. This type of response will return the designated ASCII character. (An ASCII Equivalence Chart is provided in Appendix A.) Specify the decimal equivalent of the character to be used.

KFILL

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Example

```
CONST
    CARDNUM    = 0 ;   (* Card Number Parameter For Calls *)
    FILL_SW   = 88;    (* Use 'X' As Fill Character *)
VAR
     ErrorFlag : WORD; (* Word To Hold Error Code *)
BEGIN
(*
  *** Set Fill Character To 'X' ***
*)
kfill(CARDNUM, ErrorFlag, FILL_SW);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KHELLO

Purpose Returns an identification string from the KM-488-DD driver.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  id_response :LSTRING(255);
...
BEGIN
   khello(board_nr, err_code, id_response)
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `id_response` will contain a string similar to: `copyright (c) 1991 Keithley MetraByte Corp`.

`err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The string into which the KHELLO message is returned must be dimensioned prior to calling KHELLO. Otherwise, Microsoft PASCAL assumes that the string is of length 0 and calling KHELLO will result in destroying other programmed data.

KHELLO

Example

```
CONST
    CARDNUM    = 0 ;     (* Card Number Parameter For Calls *)
VAR
    indat      : LSTRING(255);
    (* Array To Hold Copyright Message *)
    ErrorFlag : WORD;        (* Word To Hold Error Code *)
    i          : INTEGER;
(*Procedure To Determine String Length And Stuff It
 In Location 0 *)
PROCEDURE char_count (VAR ch_strng:LSTRING);
BEGIN
  (* Check For NULL or 255 Characters *)
  i := 1 ;
  WHILE ((i<>255) AND (ch_strng[i] <> CHR(0))) DO i := i + 1 ;
  (* Assign Calculated Count To String Location 0 *)
  ch_strng[0] := CHR(i - 1) ;
END;

BEGIN
(*
  *** Return Driver Copyright Message ***
*)
khello(CARDNUM,  ErrorFlag, indat[1]);
IF (ErrorFlag > 0) THEN ABORT('Error : Driver Not Instal-
led',1,0) ;
char_count(indat);
(* Update String Count In PASCAL's Descriptor *)
END.
```

KLOCAL

Purpose Forces the specified bus device(s) to return to being programmed locally from their front panels. This routine can only be used if the KM-488-DD is acting as a System or Active Controller.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  mygads : ARRAY[0..1] OF gads;
BEGIN
klocal(board_nr, err_code, mygads);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads represents the GPIB bus address of the device to be sent to local. **mygads** is an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. The **mygads** structure is defined in the include file *mp488dd.inc* and is described in section 10.1.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is the System Controller and if **mygads** does not specify an address, then the GPIB REN (Remote Enable) line is unasserted and all devices are returned to Local. In order to return them to remote mode, it will be necessary to issue a KREMOTE call. Likewise, if Local Lockout is required, it will be necessary to issue a KLOL call.

2. As an Active Controller, the KM-488-DD can issue the GPIB GTL (Go To Local) message to those devices specified by **mygads**. In this case, the GPIB REN (Remote Enable) line remains asserted and devices will return to remote when addressed to listen. If a KLOL (Local Lockout) call has been issued previously, it should still be in effect when a device is returned to Remote.

KLOCAL (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;        (* Card Number Parameter For Calls
*)
VAR
     add_12    : ARRAY[0..1] OF gads;
     (* Address Group (One Device At 12) *)
     ErrorFlag : WORD;
     (* Word For Code Returned By Calls  *)
(*
  *** Address Structure For Meter At GPIB Address 12 ***
*)
     add_12[0].primad:=12; (* 1st Device Pri. Address 12 *)
     add_12[0].secad:=-1;  (* 1st Device No Sec. Address *)
     add_12[1].primad:=-1; (* -1 = No 2nd Device In Group *)
BEGIN
(*
  *** Send The Addressed Device To Local Control ***
*)
klocal(CARDNUM, ErrorFlag,add_12[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KLOL

Purpose This command can only be used if the KM-488-DD is an Active or System Controller. It will disable the GPIB bus devices from being returned to Local Control by means of the Local/Remote button on the device.

☞ ***NOTE:*** This routine issues an IEEE-488 bus signal, LOL. This signal is not supported by all IEEE-488 bus devices.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
...
BEGIN
   klol(board_nr, err_code)
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The "LOCAL" button is disabled so that a device cannot be returned to local state for manual programming or control. The Active Controller can return specific devices to "local with lockout state", whereby an operator can then use the front panel controls. When the device is addressed to listen again, it returns to "remote with lockout state". Thus, the effect of the LOL call will remain until the REN line is unasserted (LOCAL) by the System Controller.

2. It is good practice to issue a KLOL so that devices are under strict program control. KLOL can be issued before a device is put in remote and will take effect when the device's LOCAL button is set to remote.

KLOL

Example

```
CONST
  CARDNUM    = 0 ;     (* Card Number Parameter For Calls *)
VAR
  ErrorFlag : WORD ;   (* For Returned Error Code 0 = OK *)
BEGIN
klol(CARDNUM, ErrorFlag);
 (* Lock Out Local Control Of Devices *)
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KOUTPUT

Purpose Transmits data from the KM-488-DD to the GPIB bus.

Syntax

```
VAR
  board_nr : INTEGER;
  mode : INTEGER;
  err_code : WORD;
  darray : ARRAY[0..N-1] OF CHAR;
  count : INTEGER4;
  mygads : ARRAY[0..1] OF gads;
  term_list : ARRAY[0..2] OF INTEGER;
...
BEGIN
koutput(board_nr, err_code, darray, count, mode, mygads,
term_list)
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`darray` is an array of data to output.

`count` is a long integer representing the number of characters to be output. Valid counts are within the range 1 to 4294967295 (2^{32}-1) decimal, or from 1 to #FFFFFFFF hex.

`mode` is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

KOUTPUT (cont'd)

CONT	**CONTINUE.** This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer. If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked.
DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer.

`mygads[n]` represents the GPIB bus address of the device to receive the data. **`mygads`** is a pointer to an array of **`n`** (**`n`** can range from 1 to 15) structures of type gads. You must tailor **`mygads`** to your application. **`mygads`** structure is defined in the include file *mp488dd.inc* and is described in section 10.1.

`term_list[3]` is an integer array which defines the GPIB bus terminators to be used during the KOUTPUT call. These terminators are used in conjunction with **`count`** to terminate the output process. Termination choices include whether or not EOI will be detected with the last byte and whether or not a certain byte or sequence of two bytes will be tested for as an end of message code. The default termination scheme is initially set-up by the *CONFIG.DAT* file, and can be modified by calling KTERM or KCONFIG. **`termlist[3]`** can either specify the use of the default terminators or temporary overrides.

`termlist[3]` is defined in the *mp488dd.inc* header file. **`termlist[3]`** is comprised of the following elements:

`termlist[0]`: **EOI detection.** Possible values are:

-1(#FFFF)	Use the default EOI setting.
0	Don't Assert EOI.
1	Assert EOI.

KOUTPUT (cont'd)

`termlist[1]`: **1st GPIB bus terminator.** Possible values are:

-1(#FFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.
0 to 255	Send this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

`termlist[2]`: **2nd GPIB bus terminator.** Possible values are:

-1(#FFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
0 to 255	Send this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default `termlist` can also be used:

```
VAR
     dtrm : ARRAY[0..2] OF INTEGER;
...
BEGIN
dtrm[0] := -1; (* Use the default EOI choice *)
dtrm[1] := -1; (* Use the default first terminator *)
dtrm[2] := -1; (* Use the default second terminator *)
```

KOUTPUT (cont'd)

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is a peripheral, then an Active Controller must address the KM-488-DD as a talker and some other device(s) as the listener(s). In this case, use **no_add** for **mygads**. (See section 10.1.).

2. If the KM-488-DD is both the System and Active Controller, and **my_gads** contains the device(s) to be addressed, the KOUTPUT will automatically assert the GPIB REN (Remote Enable) line.

3. If the KM-488-DD is the Active Controller and KOUTPUT is called, it will automatically address itself as the talker and another device(s) contained in **my_gads** as the listener(s). If the devices have been previously addressed, then **my_gads** can be **no_add** and the bus will not be re-addressed.

4. A non-zero **count** must always be specified.

5. If you do not wish to append terminator characters, then be sure to set the terminator defaults via *CONFIG.DAT* or by calling KTERM or CONFIG and select no terminator overrides in the **termlist** array.

6. If a listener does not accept a character, a timeout will occur unless the timeout has been disabled.

KOUTPUT (cont'd)

Example

```
CONST
  DMAXFER  = 1;
   (* Transfer mode: non DMA = 0, DMA without
   continue = 1, DMA with continue = 3 *)
  XFER = 0;     (* Normal Non-DMA Style Transfer *)
  NUMD = 1028; (* Number Of Points To OUTPUT     *)
VAR
  ErrorFlag : WORD ;
    (* Word To Hold Error Code Returned By
     Each Call (0 = false = no errors) *)

  no_add     : ARRAY[0..1] OF gads;
  (* Addr Group Signaling No Addressing *)
  dtrm       : ARRAY[0..2] OF INTEGER;
  (* Array Of GPIB Terminators For "KENTER" And "KOUTPUT" *)
  sdata      : ARRAY[0..NUMD-1] OF CHAR;
   (* Array Of Data To OUTPUT *)
BEGIN

(**** Structure For No Default Terminator/EOI Overrides ****)
dtrm[0] := 1 ;
(* 1: Assert EOI (koutput)/ Detect EOI (kenter) *)
dtrm[1] := -1 ;
(* -1: Use Default 1st Terminator *)
dtrm[2] := -1 ; (* -1 : Use Default 2nd Terminator
*)
(* Address Structure For Not Readdressing The GPIB Bus **)
no_add[0].primad := -1 ;
(* -1 Indicates No New Address *)
no_add[0].secad  := -1 ;
(* -1 Indicates No Secondary Address*)
no_add[1].primad := -1 ;
(* -1 Indicates No Other Instruments *)
(* .
   .
   sdata Array Initialized Here
   .
   . *)
koutput(CARDNUM, ErrorFlag, sdata[0], NUMD, DMAXFER,
no_add[0], dtrm[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KPASCTL

Purpose

Designates another controller to be the Active Controller. The KM-488-DD must be an Active Controller.

Syntax

```
VAR
  board_nr: INTEGER;
  err_code : WORD;
  mygads    : ARRAY[0..14] OF gads;
...
BEGIN
  kpasctl(board_nr, err_code, mygads);
...
```

Parameters

board_nr is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads[n] contains the GPIB bus address of the device to become the new Active Controller. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. **mygads** structure is defined in the include file *mp488dd.inc* and is described in section 10.1.

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

If the KM-488-DD which is relinquishing its position as the Active Controller is also a System Controller, it retains its status as System Controller.

KPASCTL (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;        (* Card Number Parameter For Calls
*)
VAR
  ErrorFlag : WORD;          (* Word For Code Returned By Calls
*)
  add_02    : ARRAY[0..14] OF gads;
  (* Address Group (One Device At 02) *)
BEGIN
(* Address Structure For Peripheral At GPIB Address 02 *)
add_02[0].primad:= 2;
(* 2 : 1st Instrument Has GPIB Address 02*)
add_02[0].secad:= -1;
(* -1 : 1st Instrument Has No Secondary Address *)
add_02[1].primad:=-1;
(* -1 : No 2nd Instrument In Address Group  *)

(* PASS CONTROL TO SECOND KM-488-DD *)
WRITELN('Pass Control To Second KM-488-DD');
kpasctl(CARDNUM, ErrorFlag, add_02[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KPPOLL

Purpose Initiates a Parallel Poll. The KM-488-DD must be the Active Controller.

☞ *NOTE:* Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  pprtn : INTEGER;
...
BEGIN
kppoll(board_nr, err_code, pprtn);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`pprtn` is an integer variable which will contain an integer in the range 0 to 255 decimal. This integer indicates which Data Lines which have been asserted (DIO1-DIO8).

Programming Notes None.

KPPOLL

Example

```
CONST
     CARDNUM  = 0 ; (* Card Number Parameter For Calls *)
VAR
     ErrorFlag : WORD;
   (* Integer to Hold Error Code Returned By
      Each CALL (0 = false = no error) *)

     pp_resp  : INTEGER; (* Integer For Parallel Response *)
(*
  *** Conduct Parallel Poll And Print Result ***
*)
kppoll(CARDNUM, ErrorFlag, pp_resp);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
WRITELN('pp_resp= ',pp_resp);
END.
```

KPPC

Purpose Configures the Parallel Poll response of a GPIB bus device. The KM-488-DD must be the Active Controller.

☞ ***NOTE:*** Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax

```
VAR
  board_nr  : INTEGER;
  err_code  : WORD;
  ppc_cfg  : INTEGER;
  mygads     : ARRAY[0..14] OF gads;
...
BEGIN
  kppoll(board_nr, err_code, mygads, ppc_cfg);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] contains the GPIB bus address(es) of the device(s) to be configured. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. **mygads** structure is defined in the include file *mp488dd.inc* and is described in section 10.1.

ppc_cfg is an integer which represents the Parallel Poll Response of the device to be programmed. This integer is of the format:

Bit	7	6	5	4	3	2	1	0
	0	1	1	0	S	P2	P1	P0

KPPC (cont'd)

Where:

S	is the parallel poll response value (0 or 1) that the device uses to respond to the parallel poll when service is required. This bit is generally set to 1.

☞ ***NOTE:*** This value must correspond to the setting of the GPIB bus device's *ist* (individual status) bit. Refer to the Device's documentation for more information.

P2 P1 P0	is a 3-bit value which tells the device being configured which data bit (DIO1 through DIO8) it should use as its parallel poll response.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPC (cont'd)

Example

```
CONST
     CARDNUM = 0 ;       (* Card Number Parameter For Calls *)
VAR
     pp_enab   : INTEGER;
  (* Integer For Parallel Poll Enabling*)
     add_1     : ARRAY[0..1] OF gads;
  (* Address Group (will have one device @ 1) *)
BEGIN
(*
  Device Supporting PPOLL Is At Address 01
*)
add_1[0].primad := 1 ;
(* 1st Instrument Has GPIB Address 01  *)
add_1[0].secad  := -1 ;
(* -1 : 1st Instrument Has No Secondary Addr  *)
add_1[1].primad := -1 ;
(* -1 : No 2nd Instrument In Address Group    *)
(*
  Configure Device To Assert DIO3 When SRQ Is Asserted
*)
pp_enab := 106;
kppc(CARDNUM,  ErrorFlag, add_1[0], pp_enab) ;
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KPPD

Purpose Disables the Parallel Poll response capability of the specified GPIB bus device(s). The KM-488-DD must be the Active Controller.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  mygads    : ARRAY[0..14] OF gads;
...
BEGIN
  kppd(board_nr, err_code, mygads);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads[n]` contains the GPIB bus address(es) of the device(es) to be disabled. `mygads` is a pointer to an array of `n` (`n` can range from 1 to 15) structures of type gads. You must tailor `mygads` to your application. `mygads` structure is defined in the include file *mp488dd.inc* and is described in section 10.1.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPD (cont'd)

Example

```
CONST
    CARDNUM    = 0 ;    (* Card Number Parameter For Calls *)
VAR
    ErrorFlag : WORD;   (* Holds Returned Error Code       *)
    add_1     : ARRAY[0..1] OF gads;
  (* Address Group (will have one device @ 1) *)

(*
  Address Structure For Device At GPIB Address 01
*)
add_1[0].primad := 1 ;
  (* 1st Instrument Has GPIB Address 01         *)
add_1[0].secad  := -1 ;
  (* -1 : 1st Instrument Has No Secondary Addr  *)
add_1[1].primad := -1 ;
  (* -1 : No 2nd Instrument In Address Group    *)
(*
  *** Disable Parallel Poll Of Addressed Devices ***
*)
kppd(CARDNUM,  ErrorFlag, add_1[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KPPU

Purpose Disables the Parallel Poll Response of all GPIB bus device(s). The KM-488-DD must be the Active Controller.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
...
BEGIN
  kppu(board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
CONST
     CARDNUM  = 0 ;    (* Card Number Parameter For Calls *)
VAR
     ErrorFlag : WORD;
       (* Integer to Hold Error Code Returned By Each CALL (0
= false = no error) *)
BEGIN
(*
  *** Disable Parallel Poll Response Of All Bus Devices ***
*)
kppu(CARDNUM,  ErrorFlag);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KQUIKSTAT

Purpose Returns the KM-488-DD's status byte.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  qstat : INTEGER;
...
BEGIN
  kquikstat(board_nr, err_code, qstat);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `qstat` is a pointer to the returned status integer. The status integer is of the following format:

Bit	15 - 9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PEND	DET	SRQ	DEC	LA	TA	DMA

Where:

ADSC	**Address Status Change.** If this bit is set to 1, a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).
CIC	**Controller in Charge.** If this bit is set to 1, the KM-488-DD is an Active Controller.
PEND	**SRQ Pending.** If this bit is set to 1, the KM-488-DD has an SRQ request pending.
DET	**Device Triggered.** If this bit is set to 1, a GPIB Trigger command has been received.
SRQ	**Internal SRQ.** If the KM-488-DD is an Active Controller and this bit is set to one, a device is requesting service. If the KM-488-DD is a Peripheral and this bit is set to one, then its SRQ (issued by KREQUEST) has not been serviced.

KQUIKSTAT(cont'd)

DEC	**Device Cleared.** If this bit is set to one, the KM-488-DD has received a GPIB Clear command.
LA	**Listen Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Talker and can output data to the GPIB.
DMA	**DMA.** A DMA transfer is currently in progress.

`err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
CONST

    ADSC = #100 ;

VAR
  board_nr : INTEGER;
  err_code : WORD;
  qstat : INTEGER;
...
BEGIN
  QSTAT := 0 ;    (* Initialize QSTAT To 0 *)
  kquikstat(board_nr, err_code, qstat);

  IF (QSTAT AND ADSC) THEN
     WRITELN('ADDRESS STATUS CHANGED') ;
  ...
```

KREMOTE

Purpose Forces the GPIB bus device(s) to the remote mode (ignore the bus).

☞ ***NOTE:*** The KM-488-DD must be a System Controller to execute this command.

Syntax

```
VAR
  board_nr  : INTEGER;
  err_code  : WORD;
  mygads     : ARRAY[0..14] OF gads;
...
BEGIN
  kremote(board_nr, err_code, mygads);
...
```

Parameters **`board_nr`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads[n]` contains the GPIB bus address(es) of the device(es) to be put in the remote mode. **`mygads`** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **`mygads`** to your application. **`mygads`** structure is defined in the include file *mp488dd.inc* and is described in section 10.1.

Returns **`err_code`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KREMOTE (cont')

Programming Notes

1. If **mygads** does not specify an address (i.e., **no_addr** is used), then the GPIB REN (Remote Enable) line is asserted. Devices will not be in the remote mode until addressed to listen. If **mygads** contains address then those devices will also be addressed to listen, so they will be put in remote.

2. It is good programming practice to issue a Local Lockout to prevent the device(s) from being returned to Local mode.

Example

```
CONST
     CARDNUM     = 0 ;   (* Card Number Parameter For Calls *)
VAR
     ErrorFlag : WORD;
(* Integer to Hold Error Code Returned By Each CALL (0 = false
= no error) *)
     add_05      : ARRAY[0..1] OF gads;
  (* Address Group (will have one device @ 05) *)
BEGIN
(*
  *** Address Structure For Device At GPIB Address 5 ***
*)
add_05[0].primad := 5 ;
(* 1st Instrument Has GPIB Address 05 *)
add_05[0].secad  := -1 ;
(* -1 : 1st Instrument Has No Secondary Addr  *)
add_05[1].primad := -1 ;
(* -1 : No 2nd Instrument In Address Group    *)
(*
  Assert REN And Listen Specified Devices
*)
kremote(CARDNUM,  ErrorFlag, add_05);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KREQUEST

Purpose Sets the Serial Poll Response of a KM-488-DD which is a Peripheral.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
 sp : INTEGER;
...
BEGIN
  krequest(board_nr, err_code, sp);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

sp is an integer in the range 0 to 255 which represents the serial poll response of the device. This integer is of the following format:

Bit	7	6	5	4	3	2	1	0
	DIO8	rsv	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
rsv	If this bit is 1, the KM-488-DD will generate a Service Request (assert SRQ).

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The Active Controller can read the value of **sp** by serial polling the KM-488-DD. This will clear bit 6, if it was set.

2. Use KQUIKSTAT or KSPOLL to check if the Peripheral has been serial polled (checks the status of the SRQ bit).

KREQUEST (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
     ASSERTSRQ  = 64;
(* Set Reserved SRQ Bit In Status Byte *)
     ErrorFlag : WORD;
(* Word For Code Returned By Calls  *)
BEGIN
(*
  *** Generate SRQ ***
*)
krequest(CARDNUM, ErrorFlag, ASSERTSRQ);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KRESET

Purpose Performs a "warm" reset of the KM-488-DD and the GPIB bus.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
...
BEGIN
 kreset(board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The KM-488-DD is reset to the following conditions:

   - STOP
   - DISARM
   - CONFIG
   - ABORT
   - BUFFER INCREMENT
   - DMA SINGLE
   - ERROR ON
   - FILL NULL
   - LOCAL
   - REQUEST 0 (If Peripheral)
   - Clear CHANGE, TRIGGER, and CLEAR STATUS
   - TIMEOUT 10

   It also clears all error conditions.

2. If the KM-488-DD is the System Controller, it will assert the GPIB IFC (Interface Clear) line for at least 500 μsecs.

KRESET (cont'd)

Example

```
CONST
  CARDNUM    = 0 ;  (* Card Number Parameter For Calls *)
VAR
  ErrorFlag : WORD; (* Word For Code Returned By Calls  *)
BEGIN
(*
  *** Reset Driver ***
*)
kreset(CARDNUM,  ErrorFlag) ;
IF (ErrorFlag > 0) THEN
  ABORT('Error In KRESET Call', 1, 0) ;
END.
```

KRESUME

Purpose Initiates data transfers between two non-Active Controller GPIB devices, by unasserting the ATN line. The KM-488-DD must be an Active Controller in order to use this function.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
...
BEGIN
kresume(board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes This routine is normally called after a KSEND has addressed a talker and a listener. (See KSEND description.)

Example

```
CONST
  CARDNUM    = 0 ;    (* Card Number Parameter For Calls *)
VAR
  ErrorFlag : WORD;   (* Word For Code Returned By Calls  *)
BEGIN
(*
  Drop ATN Line To Allow Inter-Peripheral Communication
*)
kresume(CARDNUM, ErrorFlag);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KSEND

Purpose Sends GPIB commands and data from a string.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  stringlen : INTEGER;
  sendstr : LSTRING(255);
...
BEGIN
  ksend(board_nr, err_code, sendstr, stringlen);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

sendstr is a string which includes any of the following subcommands: CMD, UNT, MTA, MLA, TALK, LISTEN, DATA, and EOI. Multiple subcommands may be specified; however, the length of the entire SEND command string cannot exceed 255 characters. The subcommands are as follows:

CMD - Sends information with the ATN line asserted. This indicates to the bus devices that the characters are to be interpreted as GPIB bus commands. The command may be sent as either a quoted string (e.g., **'COMMAND'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **#0D** hex for a Carriage Return). Hex values must be preceded by #. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **CMD 67,79,77,77,65,78,68**).

An EOI cannot be sent with this subcommand, because an EOI with ATN asserted would initiate a Parallel Poll.

UNT - Untalks all devices by sending a GPIB UNTALK command with ATN asserted.

KSEND (cont'd)

UNL - Unlistens all devices by sending the GPIB UNLISTEN command with ATN asserted.

MTA - Designates the KM-488-DD to address itself to talk by sending the GPIB MTA (My Talk Address) command with the ATN line asserted.

MLA - Designates the KM-488-DD to address itself to listen by sending the GPIB MLA (My Listen Address) command with the ATN line asserted.

`TALK addr` - Addresses another GPIB device or KM-488-DD to talk by sending a GPIB TAG (Talk Address Group) command with the ATN line asserted. **`addr`** is an integer representing the GPIB BUS device address of the device to talk. This integer ranges from 00 to 30 decimal.

`LISTEN addr` - Addresses another GPIB device(s) or KM-488-DD('s) to listen by sending a GPIB LAG (Listen Address Group) command with ATN asserted. **`addr`** is an integer representing the GPIB BUS device address of the device(s) to talk. This integer ranges from 00 to 30 decimal. Multiple listeners can be specified. If **`addr`** is not specified, all other devices on the GPIB BUS will be designated listeners.

KSEND (cont'd)

DATA - Sends information with the ATN line unasserted. This indicates to the bus devices that the characters are to be interpreted as data. This is the technique which is to be used to send device-dependent commands. (See the IEEE-488 Tutorial in Appendix C for more information.) The data may be sent as either a quoted string (i.e., **'DATA'**) or as a numeric equivalent of an individual ASCII character (i.e., **13** decimal or **#0D** hex for a Carriage Return). Hex values must be preceded by #. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **DATA 68,65,84,65**).

This sub-command is useful when you are sending commands which are unique to your interface.

EOI - Sends information with the ATN line unasserted. EOI will be asserted when the last character is transmitted. This information is interpreted as GPIB bus data and may be sent as either a quoted string (e.g., **'xyz'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **#0D** hex for a Carriage Return). Hex values must be preceded by #. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **SEND EOI 120,121,122**).

stringlen is an integer between 0 and 255 which represents the length of the SEND string.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Typically, KSEND is used to have to device transfer data without it passing through the KM-488-DD. For example, a KM-488-DD might talk a scope and send a trace to a listened plotter.

2. The maximum length of the KSEND command, including any subcommands, is 255 characters. To KSEND large amounts of data, use multiple KSEND commands.

KSEND (cont'd)

3. KSEND should only be used when a non-conforming device requires a special command sequence or a non-standard GPIB command. Do not use the KSEND command unless you are extremely familiar with GPIB.

4. The KM-488-DD must be the Active Controller to KSEND commands. Any KM-488-DD can KSEND data.

5. If a DATA subcommand is not included in the KSEND string, be sure to call KRESUME immediately after the KSEND. This is necessary because the ATN line must be dropped so that the transfer will proceed.

6. Do not include the word SEND within the KSEND string as you might do in the File I/O SEND command.

Example

```
CONST
  CARDNUM   = 0 ;    (* Card Number Parameter For Calls *)
VAR
  snstr      : LSTRING(255);
(* For Command Used In KSEND      *)
  ErrorFlag : WORD;
(* Word For Code Returned By Calls   *)
BEGIN
(*
  HAVE 196 DMM SEND READING TO A PERIPHERAL KM-488-DD
  DMM Is At Address 12
  Peripheral KM-488-DD Is At Address 02
*)
snstr := 'UNT UNL LISTEN 02 TALK 12';
ksend(CARDNUM, ErrorFlag, snstr[1], 25);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
```

KSPOLL

Purpose

If the KM-488-DD is an Active Controller, KSPOLL will check for an SRQ or conduct a serial poll. If the KM-488-DD is a Peripheral, KSPOLL will check if the KM-488-DD's SRQ has been serviced.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  spr : INTEGER;
  mygads     : ARRAY[0..14] OF gads;
...
BEGIN
  kspoll(board_nr, err_code, spr, mygads);
...
```

Parameters

board_nr is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] contains the GPIB bus address(es) of the device(es) to be disabled. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. **mygads** structure is defined in the include file *mp488dd.inc* and is described in section 10.1.

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

spr is an unsigned integer or an array of unsigned integers with an entry for each address in **mygads**. The value of the integer is 0 to 255 and corresponds to a byte with the following bitmap:

Bit	7	6	5	4	3	2	1	0
	DIO8	SRQ	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

KSPOLL(cont'd)

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
SRQ	If this bit is 1, the SRQ (Service Request) line will be asserted to request servicing. Otherwise, SRQ will not be asserted.

Bit 6 has the special significance of indicating an SRQ pending. The significance of the other bits will be application dependent.

Programming Notes

1. The most common use of KSPOLL is for the KM-488-DD, as an active controller, to issue KSPOLL with **mygads** containing the address of a single device. This addresses and serial polls the device and, upon return, **spr** contains the serial poll response of the polled device. If the SRQ bit in **spr** is set, the device had issued an SRQ. The other bits in **spr**are device-dependent. Serial polling a device which is currently asserting SRQ will cause the device to unassert SRQ.

   The string returned by KSTATUS or the integer word by KQUIKSTAT can be checked to determine the presence of an SRQ before a serial poll is conducted. If only one device is asserting SRQ, the effect of issuing KSPOLL will be to clear the internal SRQ pending "bit", even if the polled device is not issuing the SRQ. It is also the case when multiple devices are asserting SRQ and a device not currently asserting SRQ is polled. In this case, a subsequent use of KSTATUS or KQUIKSTAT will not reveal a pending SRQ.

   To aid in identifying which bus device(s) is currently requesting service, a KM-488-DD which is the Active Controller can serial poll as many as 14 devices by issuing a KSPOLL call with **mygads** containing the address of more than one device. In this case, **spr** should be an array of unsigned integers with an element for each address. In this case, KSPOLL will serial poll each addressed device and return the serial poll bytes in the **spr** array. This is a faster way for discerning the source of an SRQ among several devices.

KSPOLL (cont'd)

2. If the KM-488-DD is a peripheral and KSPOLL is called with **mygads** equal to the no address structure (**no_addr**), **spr** will be the serial poll response (**sp**) set by KREQUEST - with the possible exception of the SRQ bit. (This may have disappeared as a result of an Active Controller having serial polled the KM-488-DD since the last KREQUEST call.) In the instance of an Active Controller serial polling a peripheral KM-488-DD, the peripheral KM-488-DD unasserts the SRQ line. For an example, see KREQUEST. The state of the SRQ bit can also be determined from the string returned by KSTATUS or the integer status word set by a KQUIKSTAT call.

3. If the KM-488-DD is the Active Controller and issues KSPOLL with **mygads** equal to **no_addr**, **spr** will be 64 if a device is asserting SRQ and 0 if not. This same result could have been determined from KQUIKSTAT or KSTATUS.

   Note that if this call is issued when a SRQ is pending, it internally "clears" the SRQ "pending" bit, even though the requesting device has not been polled and is still issuing an SRQ. That is, a subsequent KSPOLL, KQUIKSTAT, or KSTATUS call will not reveal a pending SRQ.

KSPOLL (cont'd)

Example

```
CONST
   CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
VAR
   add_12    : ARRAY[0..1] OF gads;
  (* Address Group (One Device At 12) *)
   ErrorFlag : WORD;     (* Word For Code Returned By Calls  *)
BEGIN
(*
  *** Address Structure For Meter At GPIB Address 12 ***
*)
add_12[0].primad:=12;
     (* 1st Instrument Has GPIB Address 12  *)
add_12[0].secad:=-1;
     (* -1 Indicates 1st Instrument Has No Secondary Add *)
add_12[1].primad:=-1;
     (* -1 Indicates No 2nd Instrument In Group *)
(*
  Acknowledge/Remove SRQ Of The Device @ Address 12
*)
kspoll(CARDNUM,  ErrorFlag, add_12[0], sbyt) ;
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag) ;
END.
```

KSTATUS

Purpose Returns a character string describing the current operating state of the KM-488-DD.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  stat : LSTRING(255);
...
BEGIN
  kstatus(board_nr, err_code, stat);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`stat` is a character string which describes the current operating state of the KM-488-DD. The string is of the format shown in Table 9-2. `stat` should be 75 bytes in order to contain the longest message that might be returned. If it is too short, data will be written over other portions of the program causing problems.

Table 9-2. Status String

Starting Col	# of Cols.	Name and Description

1	1	Operating Mode - Tells if the KM-488-DD is acting as an *Active Controller* or *Peripheral*. Can be C or P where:
		
		C = The KM-488-DD is an *Active Controller*
		P = The KM-488-DD is a *Peripheral*

3	2	Bus Address - Gives the IEEE-488 Bus Address of the KM-488-DD. This is a two-digit decimal integer ranging from 00 to 30.
6	2	Addressed Status Change - Indicates if the device has changed addressed state, i.e., if it has cycled between the *Talker, Listener,* or *Active Controller* states. This is reset whenever a STATUS is read. This can be one of the following: G0 = There has not been a change in the addressed status. G1 = There has been a change in the addressed status.

9	1	Current Addressed State - Indicates if the KM-488-DD is currently acting as a *Talker* or *Listener*, or is *Idle*. Can be T, L, or I, where:
		
		T = The KM-488-DD is a Talker
		L = The KM-488-DD is a Listener
		I = The KM-488-DD is Idle

11	2	Service Request - Represents the current internal SRQ status where: S0 = SRQ not asserted S1 = SRQ asserted If the KM-488-DD is in the *Active Controller* mode, the internal SRQ state indicates if a device is asserting SRQ. If the KM-488-DD is acting as a *Peripheral*, the internal SRQ state is set by using the KREQUEST call. It is cleared by a serial poll from the Active Controller and indicates it is asserting SRQ.

KSTATUS(cont'd)

Table 9-2. Status String

Starting Col	# of Cols.	Name and Description
14	3	Error Code - Indicates whether or not an error has occurred. The three characters consist of the letter 'E' followed by a two digit error code. (A list of these error codes is provided in Appendix G.) An error code of 00 indicates no error has occurred.
18	2	Triggered State - Indicates if a IEEE-488 trigger command has been received or not. This is not updated during DMA CONTINUE transfers. Triggered State values are: T0 = No trigger command has been received. T1 = A trigger command has been received.
21	2	Cleared - Indicates if the IEEE Clear command has been received or not. This is not updated during DMA CONTINUE transfers. Cleared values are: C0 = No Clear command has been received. C1 = A Clear command has been received.

24	2	Transfer Status - Indicates if a data transfer is in progress. Values which may appear are:
		
		P0 = No transfer
		P1 = DMA CONTINUE Transfer occurring
		P2 = DMA AUTOINITIALIZE Transfer occurring

27	up to 45	Error Message Text - Contains the Error Message Text associated with the given error code (EXX). These error messages are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is the Active Controller, the Service Request Status may be cleared (i.e, S0) as a result of a KSPOLL, even when the SRQ line is still asserted. See the discussion in KSPOLL.

2. If the KSTATUS string is read into a variable of type LSTRING (as opposed to a character array), the string length must be calculated upon return from the KSTATUS call and inserted in location 0 of the LSTRING variable. The string length can be calculated by searching through the LSTRING (starting with location STRING[1]) for a NULL character (0).

KSTATUS (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
VAR
     indat      : LSTRING(255);(* Array To Hold Status Message
*)
     ErrorFlag : WORD;        (* Word To Hold Error Code *)
     i          : INTEGER;
(*
  Proc. To Determine String Length And Put It In Loc.0
*)
PROCEDURE char_count (VAR ch_strng:LSTRING);
BEGIN
(*
  Check For NULL or 255 Characters
*)
  i := 1 ;
  WHILE ((i<>255) AND (ch_strng[i] <> CHR(0))) DO
     i := i + 1 ;
(*
  Assign Calculated Count To String Location 0
*)

  ch_strng[0] := CHR(i - 1) ;
END;
BEGIN
(*
  *** Return Status Message ***
*)
kstatus(CARDNUM, ErrorFlag, st[1]) ;  (* Request Status *)
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
char_count(st);
WRITELN('STATUS :', st);
END.
```

KSTOP

Purpose Stops a DMA CONTINUE transfer.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
...
BEGIN
  kstop (board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the DMA CONTINUE transfer has been completed, a KSTOP will have no effect.
2. Use the KBUFD call to determine the actual number of characters which were transferred before the KSTOP took effect.

Example

```
CONST
  CARDNUM    = 0 ;      (* Card Number Parameter For Calls *)
VAR
  ErrorFlag : WORD ;
   (* Word To Hold Error Code Returned By Each Call (0 = false
= no errors) *)
BEGIN
(*
  *** Stop DMA ***
*)
kstop(CARDNUM,  ErrorFlag);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KTERM

Purpose Changes the default GPIB bus terminator(s) to be used in ENTER and/or OUTPUT calls.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
 trm_list[4]; ARRAY [0...3] OF INTEGER
...
BEGIN
  kterm(board_nr, err_code, trm_list);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`term_list` is an integer array which defines the GPIB bus terminators. See Chapter 6 for more information regarding the use of GPIB bus terminators. (`termlist[4]` is defined in the *mp488dd.inc* header file.) `termlist[4]` is comprised of the following elements:

`termlist[0]`: **Input/Output GPIB Terminators.** Possible values are:

-1	Change the Input Terminators.
0	Change both Input and Output Terminators.
1	Change the Output Terminators.

`termlist[1]`: **EOI detection/Assertion.** Possible values are:

0	No assert/dectect EOI
1	Assert/Detect EOI.

KTERM (cont'd)

`termlist[2]`: **1st GPIB bus terminator.** Possible values are:

-1(#FFFF)	Don't use first or second GPIB bus terminator.
0 to 255	Use this ASCII character as the first GPIB bus terminator instead of the default first GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

`termlist[3]`: **2nd GPIB bus terminator.** Possible values are:

-1(#FFFF)	Don't use second GPIB bus terminator.
0 to 255	Use this ASCII character as the second GPIB bus terminator instead of the default second GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. KTERM will change the default bus terminators set during software installation, as directed by `termlist[]`.

KTERM (cont'd)

Example

```
CONST
  CARDNUM    = 0 ;        (* Card Number Parameter For Calls *)
VAR
  ErrorFlag : WORD ;
(* Word To Hold Error Code Returned By Each
Call (0 = false = no errors) *)
  trm_lst    : ARRAY[0..3] OF INTEGER;
   (* Array Of GPIB Terminators For "KTERM" *)
BEGIN
(*
  *** Configure Terminators For BUS Operations
*)
(*
 *** Structure To Reset Configuration Terminators ***
*)
trm_lst[0] := 0 ;
     (* 0 Indicates Change Input AND Output Terminators *)
trm_lst[1] := 1 ;
     (* 1 = Assert EOI On Output, Detect EOI On Enter   *)
trm_lst[2] := -1 ;
    (* -1 Indicates No 1st Terminator                  *)
trm_lst[3] := -1 ;
    (* -1 Indicates No 2nd Terminator                  *)
kterm(CARDNUM,  ErrorFlag, trm_lst[0]) ;
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag) ;
END.
```

KTO

Purpose Changes the timeout period.

Syntax

```
VAR
 board_nr : INTEGER
 err_code : WORD;
 tval : INTEGER4;
...
BEGIN
 kto(board_nr, err_code,tval);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`tval` is the number of 55 ms. timer ticks to allow before a time-out.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. To suppress Timeout checking, set `tval` to 0.

2. If a DMA CONTINUE transfer is in progress, the KM-488-DD will check for timeouts only for the first byte that is transmitted/received. During other types of transfers, the KM-488-DD will check for a timeout between transmission of bytes.

You must be certain to check that a DMA CONTINUE transfer has been completed.

KTO (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;     (* Card Number Parameter For Calls *)
     FIFTEEN  = 273 ;     (* 15 seconds at 18.2 ticks per
second *)
VAR
  ErrorFlag : WORD ;     (* Word To Hold Error Code Returned By
                            Each Call (0 = false = no errors)
*)
BEGIN
(*
  Set Time Out To 15 Seconds (Approx 18.2 Ticks/Sec)
*)
kto(CARDNUM,  ErrorFlag, FIFTEEN);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KTRIGGER

Purpose Triggers the specified device(s). The KM-488-DD must be an Active Controller.

Syntax

```
VAR
  board_nr  : INTEGER
  err_code  : WORD;
  mygads    : ARRAY[0..14] OF gads;
...
BEGIN
  ktrigger(board_nr, err_code, mygads);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] contains the GPIB bus address(es) of the device(es) to be triggered. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. **mygads** structure is defined in the include file *mp488dd.inc* and is described in section 10.1.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The devices listed in **mygads** will be triggered.

KTRIGGER (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;        (* Card Number Parameter For Calls
*)
VAR
     add_12     : ARRAY[0..1] OF gads;
  (* Address Group (One Device At 12) *)
     ErrorFlag : WORD;
  (* Word For Code Returned By Calls  *)
(*
  *** Address Structure For Meter At GPIB Address 12 ***
*)
add_12[0].primad:=12;
    (* 1st Instrument Has GPIB Address *)
add_12[0].secad:=-1;
     (* -1 Indicates 1st Instrument Has No Secondary Add *)
add_12[1].primad:=-1;
    (* -1 Indicates No 2nd Instrument In Group *)
BEGIN
(*
  *** Trigger The Device ***
*)
ktrigger(CARDNUM, ErrorFlag,add_12[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KWAITC

Purpose Forces the KM-488-DD driver to wait until a DMA CONTINUE transfer has been completed before returning control to the application program.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
...
BEGIN
 kwaitc(board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Time out checking, if enabled, is performed while KWAITCing.

2. This is an explicit wait. The following calls perform an implicit wait:

   - KABORT
   - KCLEAR
   - KENTER
   - KLOCAL
   - KLOCAL LOCKOUT
   - KOUTPUT
   - KPASS CONTROL
   - KPPOLL
   - KPPOLL CONFIG
   - KPPOLL DISABLE
   - KPPOLL UNCONFIG
   - KREMOTE
   - KREQUEST
   - KRESUME
   - KSEND
   - KSPOLL
   - KTRIGGER

KWAITC (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;  (* Card Number Parameter For Calls *)
VAR
     ErrorFlag : WORD; (* Word For Code Returned By Calls  *)
BEGIN
(*
  *** Wait For DMA To Finish ***
*)
(*
        .
DMA CONTINUE started here.
        .
*)
kwaitc(CARDNUM,  ErrorFlag);
IF (ErrorFlag > 0) THEN
     WRITELN('Time Out Waiting For DMA To Finish') ;
END.
```

This page intentionally left blank.

CHAPTER 10
USING THE CALL INTERFACE IN TURBO PASCAL

Versions Supported

Turbo PASCAL versions 4.0, 5.0 and higher

File Header

Be sure to include the following lines within your program:

```
{$I tp488dd.inc}
{$L km488tp.obj}
```

tp488dd.inc is an "include" file with function/procedure prototypes, structure definitions, and constants pertinent to the TURBO PASCAL call interface.

km488tp.obj is a linkable module containing the functions and procedures defined in *tp488dd.inc*. This module dispatches KM-488-DD calls into the KM driver.

If either of these files resides in a directory other than the one in which you are compiling your program, be sure to specify a complete path to each file.

The **{$V-}** is used throughout the example programs to relax string checking during procedure and function calls. You may want to use this metacommand in instances where a value is returned in a string and the ultimate length of the string is unknown.

Compiling The Program

Typically, the following simple statement will suffice for compiling your program provided you have adhered to the instructions in the file header section:

```
tpc myprog.pas
```

Software Configuration

A number of KM-488-DD configuration parameters are set via the *CONFIG.DAT* file. (See Chapter 2 and Appendix F.) These govern the default settings of the GPIB input and output bus terminators, lightpen interrupt enable, device timeout period, and the KM-488-DD's Base Address. The defaults for these are listed in Table 10-1. There are other defaults you may have to re-program if you are using File I/O Commands in the same program as the library interface routines (calls).

Table 10-1. Default KM-488-DD Operating Parameters

Parameter	Default
Device Timeout	10.0 seconds
GPIB Bus Input Terminator	CR LF with no EOI
GPIB Bus Output Terminator	CR LF with no EOI
EOL Terminators	CR LF

The KCONFIG call can be used to reset the GPIB address, and the GPIB input and output bus terminators.

The KTERM call can be used to change the default GPIB bus terminator settings.

The KTO call can be used to change the default device timeout value.

Programming Notes

1. Any values passed into a call in a variable may also be passed as a constant or as the returned value of a function.

2. Parameters which are also used to return values must be declared as variables.

3. Any of the KM-488-DD routines which are used to receive data require that a string or array be declared to store the received data. The length of the string or size of the array should be sufficient to store the number of bytes that are expected. In addition, when a KM-488-DD routine requires a parameter specifying the maximum number of data bytes to be returned, it is extremely important that the amount of storage space allocated is at least as great as this maximum length parameter. Otherwise, data may be stored into memory which has been allocated for use by other parts of your program, or for use by DOS. This could lead to erroneous operation and possibly a system crash.

4. In Turbo PASCAL, strings are actually a special type of character array. The first byte of the array is used to store the number of bytes contained within the string. Hence, strings may range from 0 to 255 bytes in length and the KM-488-DD routines which pass data to or from strings are limited to 255 bytes maximum.

   When values are returned to the calling program via a string variable, the string length must be calculated and placed in location 0 of the string, where Turbo PASCAL normally stores it. The string length must be calculated in this manner as the KM-488-DD driver is unaware of which language is issuing the current call.

5. Do not name the variables in your application program with the same name as any of the KM-488-DD routines.

6. Do not assign a program name which is the same name as any of the KM-488-DD routines.

10.1 CALL DESCRIPTION FORMAT

Each call description is divided into several sections as described below.

Syntax This section describes the syntax required by each call. The following rules are used in this section.

Call names and input parameters appear in **`this typeface`**.

Parameters This section describes the input parameters. In some instances, a structure or array of multiple parameters may be specified (for example, multiple device addresses).

`mygads[n]` is used in many calls. This is an array of **`n`** (1 to 15) structures of type **`gads`** which contain the GPIB address information required by the call. This GPIB address structure consists of:

```
TYPE
     gads = RECORD
     primad : INTEGER;          (*primary address 0-30*)
     secad  : INTEGER;          (*secondary address 0-31*)
     END;
```

When **`secad`** is set equal to -1, it indicates that no secondary address is applicable. When **`primad`** is set equal to -1, it terminates the array.

You must tailor **`mygads`** to your application. The **`mygads`** structure is defined in the include file *tp488dd.inc*. An example of a structure with two devices is:

```
EXAM : ARRAY [0..2] OF GADS;

exam.primad[0] := 3 (* first device - pri addr 3*)
exam.secad[0] := -1 (* and no sec addr      *)
exam.primad[1] := 15 (* second device - pri addr 15 *)
exam.secad[1] := 2  (* and sec addr 2 *)
exam.primad[2] := -1 (* there are only two devices *)
```

If the KM-488-DD is a Peripheral (in which case the call will not require a GPIB bus address), you will want to use the following **mygads** structure:

```
VAR
    no_add : ARRAY [0..2] OF GADS;
...
BEGIN
   no_add.primad[0] := -1
(* there are no devices in group *)
```

Returns This part of the callable description describes any data which will be returned after execution of the callable has been completed.

err_code - All calls except KDIAGON and KDIAGOFF return an integer error code which will be 0 if no error was detected. It is good practice to check **err_code** after each call and provide a routine to handle errors if they arise.

Programming Notes This section lists any special programming considerations for the routine.

Example This section gives programming examples.

KABORT

Purpose This routine allows the KM-488-DD to take control of the GPIB bus by forcing all other devices on the bus to the idle state. The KM-488-DD must be the Active Controller.

Syntax

```
VAR

     board_nr : INTEGER;
     err_code : WORD;
...
BEGIN
...
kabort(board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the KM-488-DD is both the Active and System Controller, KABORT asserts the IFC line to retain control of the bus. Otherwise, the KM-488-DD will assert the ATN line, and then untalk and unlisten all devices.

Example

```
CONST
     CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
VAR
     ErrorFlag : WORD;      (* Word To Hold Error Code *)
BEGIN
kabort(CARDNUM, ErrorFlag);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KARM

Purpose Defines the conditions under which the lightpen status becomes true.

Syntax

```
VAR
    board_nr : INTEGER;
    arm_code : INTEGER;
    err_code : WORD;
...
BEGIN
karm(board_nr, err_code, arm_code);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

arm_code is a 16-bit integer describing on which conditions a light pen interrupt may occur. The integer is of the format:

Bit	15 - 9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC **Address Status Change.** If this bit is set to 1, the lightpen status will become true when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller). This encompasses all the conditions marked with an asterisk (*) below.

CIC* **Controller in Charge.** If this bit is set to 1, the lightpen status will become true when the control is passed to the KM-488-DD and it becomes the Active Controller.

PER* **Peripheral.** If this bit is set to 1, the lightpen status will become true when the KM-488-DD passes control to another device and becomes a Non-Active Controller.

DET **Device Triggered.** If this bit is set to 1, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.

KARM (cont'd)

SRQ	**Internal SRQ.** If this bit is set to one, the lightpen status will become true when the KM-488-DD's internal Service Request Bit (SRQ) is set.
DEC	**Device Cleared.** If this bit is set to one, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA*	**Listen Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Listener and can accept data from the GPIB.
TA*	**Talk Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Talker and can output data to the GPIB.
IDLE*	**Idle.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is unaddressed as a Talker or a Listener.

☞ ***NOTE:*** The conditions marked by an asterisk (*) will force the lightpen status to true only when the KM-488-DD's address status has changed. This is indicated by the state of the Address Change Bit in the Status byte. See the kstatus or kquikstat routine descriptions for more information.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The driver must be configured for lightpen emulation at load time via the *CONFIG.DAT* "pipe" file, as the default for lightpen emulation is off.

KARM (cont'd)

Example

```
CONST
   CARDNUM     = 0 ;
(* Card Number Parameter For Calls *)
   SRQENAB     = 16 ;
(* BIT 4 -- SRQ Detection          *)
VAR
     ErrorFlag : WORD;  (* Word To Hold Error Code *)
BEGIN
(*
  *** Enable Light Pen Interrupt Detection On SRQ ***
*)
karm(CARDNUM, ErrorFlag, SRQENAB);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KBUF

Purpose Selects in which direction the memory is to be addressed for both DMA and non-DMA buffered transfers.

Syntax

```
VAR
     dir : INTEGER;
     board_nr : INTEGER;
     err_code : WORD;
...
BEGIN
kbuf(board_nr, err_code, dir);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

dir selects the direction in which the buffer is to be read. If **dir** is equal to 1, the buffer address will be incremented. If **dir** is equal to 0, the buffer address will be decremented.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the buffer's address is to be incremented during a bus I/O operation, be sure to use the lowest address in the designated buffer. Likewise, if it is to be decremented, select the highest address.

KBUF (cont'd)

Example

```
CONST
  CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
  DECREMENT  = 0 ;
(* Decrement Memory Address on DMA *)
VAR
  ErrorFlag : WORD;         (* Word To Hold Error Code *)
BEGIN
(*
  *** Set Direction Of Memory Address Change With DMA ***
*)
kbuf(CARDNUM, ErrorFlag, DECREMENT);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KBUFD

Purpose Returns the number of characters transferred in the last buffered transfer or, in the case of DMA transfers, the current transfer.

Syntax

```
VAR
    board_nr : INTEGER;
    err_code : WORD;
    bcount : INTEGER4;
BEGIN
...
    kbufd(board_nr, err_code, bcount);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`bcount` represents the number of characters which were transferred during the last buffered transaction. This will be a long integer (32- bits) in the range 0 to 1,048,575 (0 to 20^{20}-1).

Programming Notes

1. If the data was transferred in the DMA CONTINUE mode, the value returns the number of characters transferred thus far.

2. It is useful to call this routine after a buffered KENTER has been performed. This will help you to determine if the expected number of characters has been received and/or the transfer has terminated as the result of a detected terminator.

KBUFD (cont'd)

Example

```
CONST
   CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
VAR
    ErrorFlag : WORD; (* Word To Hold Error Code *)
   XferCount : INTEGER4;
  (* Number Of Bytes Moved In Last Transfer *)
BEGIN
(*
 Return Number Of Bytes In Last GPIB BUS Data Transfer
*)
kbufd(CARDNUM, ErrorFlag, XferCount);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KCLEAR

Purpose

The KM-488-DD must be an Active Controller. Resets device(s) to their power-up parameters.

Syntax

```
VAR
    board_nr : INTEGER;
    err_code : WORD;
    mygads : ARRAY[0..n] OF GADS;
BEGIN
...
kclear(board_nr, err_code, mygads);
```

Parameters

`board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads[n]` represents the GPIB bus address of the device to be accessed. `mygads` is an array of `n` (`n` can range from 1 to 15) structures of type gads. See section 10.1 for more information.

Returns

`err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If `mygads` contains no addresses, then the universal GPIB DCL (Device Clear) command is issued and all devices will be cleared.

2. If `mygads` contains addresses, then those devices specified will be listened and sent a GPIB SDC (Select Device Clear) Command.

KCLEAR (cont'd)

Example

```
CONST
  CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
VAR
  add_12     : ARRAY[0..1] OF gads;
(* Address Group (One Device At 12) *)
   ErrorFlag : WORD;
(* Word For Code Returned By Calls  *)
(*
  *** Address Structure For Meter At GPIB Address 12 ***
*)
add_12[0].primad:=12;
    (* 1st Instrument Has GPIB Address 12*)
add_12[0].secad:=-1;
     (* -1 Indicates 1st Instrument Has No Secondary Add
*)
add_12[1].primad:=-1;
    (* -1 Indicates No 2nd Instrument In Group *)
BEGIN
(*
  *** Return The Device To Its Power Up Condition ***
*)
kclear(CARDNUM, ErrorFlag,add_12[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KCONFIG

Purpose Resets the KM-488-DD to its default operating parameters or to those specified.

Syntax

```
VAR
    board_nr : INTEGER;
    err_code : WORD;
    cfg_string : STRING;
    stringlen : INTEGER;
...
BEGIN
    kconfig(board_nr, err_code, cfg_string, stringlen);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

cfg_string is a user-defined string. This string can be comprised of any of the following options:

/T *[term term EOI]* Sets the input and output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/T**, the GPIB bus terminators will be reset to their defaults.

term is an optional parameter which represents a terminator byte to be used. This terminator byte can be represented as any of the following:

$char where **char** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **char** must be preceded by a dollar sign ($).

KCONFIG (cont'd)

Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.

CR	This represents the *Carriage Return* character (13 decimal, 0D hex).
LF	This represents the *Line Feed* character (10 decimal, 0A hex).
'X	where **X** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **'B** represents the character *B*.

EOI (End or Identify) is an optional GPIB BUS terminator. If included, the KM-488-DD will detect/assert the GPIB bus EOI line. (This would indicate that the last character has been sent.) Data will continue to be read until this terminator, a valid terminator sequence, or both are detected.

/TI *[term term EOI]*	Sets the input GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TI**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/TO *[term term EOI]*	Sets the output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TO**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/E *[term term]*	Sets the input and output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **term** is described above.

KCONFIG (cont'd)

/EI *[term term]*	Sets the input EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **term** is described above.
/EO *[term term]*	Sets the output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.)
/N name addr *[sec_addr]*	Configures a named device. Sets its address to the given value and its GPIB bus terminators to the current GPIB bus terminator settings. (Note that this should only be used if the application program also includes File I/O Commands utilizing the named device configurations.) **name** is the name of the device. **addr** is the primary GPIB address (00 to 30). **sec_addr** is the secondary GPIB address (00 to 31).

☞ ***NOTE:*** Do not create a named device with the same name as an existing directory on the current working disk.

If **cfg_string** is empty, then the configuration will revert to the installation configuration.

stringlen is an integer from 0 to 255 which gives the length of the **cfg_string** string.

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KCONFIG (cont'd)

Programming Notes

1. The **/E**, **/EI**, **/EO**, and **/N** options will only take effect if File I/O Commands are used within the same application program as the callables.

2. If you are converting an application program previously written using the File I/O commands to callables, be sure to note that the CONFIG string for the KCONFIG call does not include the "CONFIG" keyword (i.e., as you would if you were using the CONFIG File I/O command).

3. Primary and Secondary Addresses must be two characters long, e.g. 01 or 0209.

Example

```
CONST
  CARDNUM     = 0 ;   (* Card Number Parameter For Calls *)
VAR
  ConfStr      : STRING(70);
(* For Command Used In KCONFIG *)
  ErrorFlag : WORD;
(* Word For Code Returned By Calls   *)
BEGIN
(*
 Configure Terminations for Carriage Return, Line Feed
With EOI
*)
ConfStr := '/T CR LF EOI';
kconfig(CARDNUM, ErrorFlag,confstr[1],12);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
```

KDIAGON

Purpose Turns file diagnostics on. (Default is off.)

Syntax

```
VAR
     filename : STRING;
...
BEGIN
    kdiagon(filename, strlen(filename));
...
```

Parameters `filename` is a string containing the name of the file to which the diagnostic information is to be written.

`strlen` is the length of the string representing the name of the file.

Returns None.

Programming Notes This will echo the value of the call parameters into a disk file.

Example

```
CONST
   CARDNUM    = 0 ;    (* Card Number Parameter For Calls
*)
VAR
     tdstr     : STRING;
    (* String To Hold Target Diagnostic File Name *)
BEGIN
tdstr:='DEBUG.DAT';    (* Target Diagnostic File Name *)
(*
  *** Turn Driver Diagnostics On And Send Info To Disk
File ***
*)
kdiagon(tdstr[1], 9);
(* File Name And File Name Length *)
END.
```

KDIAGOFF

Purpose	Temporarily disables file diagnostics.
Syntax	`kdiagoff()`
Parameters	None.
Returns	None.
Programming Notes	None.

Example

```
BEGIN
(*
  *** Disable File Diagnostics
*)
kdiagoff();
END.
```

KDISARM

Purpose Resets previously set conditions which cause lightpen interrupts.

Syntax

```
VAR
    board_nr : INTEGER;
    arm_code : INTEGER;
    err_code : WORD;
...
BEGIN
    kdisarm(board_nr, err_code, arm_code);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

arm_code is a 16-bit integer describing which conditions should be reset. The integer is of the format:

Bit	15-9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC	**Address Status Change.** If this bit is set to 1, light pen status will not change when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).
CIC	**Controller in Charge.** If this bit is set to 1, light pen status will not change when the control is passed to the KM-488-DD and it becomes the Active Controller.
PER	**Peripheral.** If this bit is set to 1, light pen status will not change when the KM-488-DD passes control to another device and becomes a Non-Active Controller (Peripheral).
DET	**Device Triggered.** If this bit is set to 1, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.

KDISARM (cont'd)

SRQ	**Internal SRQ.** If this bit is set to one, light pen status will not change when the KM-488-DD's internal Service Request Bit (SRQ) is set.
DEC	**Device Cleared.** If this bit is set to one, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA	**Listen Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Talker and can output data to the GPIB.
IDLE	**Idle.** If this bit is set to one, light pen status will not change when the KM-488-DD is unaddressed as a Talker or a Listener.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KDISARM (cont'd)

Example

```
CONST
    CARDNUM     = 0 ;
(* Card Number Parameter For Calls *)
    SRQENAB     = 16 ;
(* BIT 4 -- SRQ Detection           *)
VAR
    ErrorFlag : WORD;
(* Word To Hold Error Code *)
BEGIN
(*
  *** Disable Light Pen Interrupt Detection On SRQ ***
*)
kdisarm(CARDNUM, ErrorFlag, SRQENAB);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KDMA

Purpose Sets DMA transfer mode.

Syntax

```
VAR
    board_nr  : INTEGER;
    dma_mode  : INTEGER;
    err_code  : WORD;
...
BEGIN
    kdma (board_nr, &err_code, dma_mode);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

dma_mode is a 16-bit integer which represents the DMA mode to be used. If no **dma_mode** is given, then single mode without autoinitialization will be assumed. The **dma_mode** integer is interpreted as follows:

Bit	15 - 3	2	1	0
	0	SIN	DEM	AUTO

Where:

SIN SINGLE. When this bit is set to one, the SINGLE mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus and transfers a single byte of data. Control of the bus is then returned to the microprocessor.

DEM **DEMAND.** When this bit is set to one, the DEMAND mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus. The DMA controller retains control of the bus until the DMA request signal is unasserted. Once this signal has been unasserted for more than one microprocessor clock cycle, control of bus is returned to the microprocessor. This mode allows the DMA controller chip to pass data at a slightly faster rate and the microprocessor to access the bus when it is not needed.

KDMA (cont'd)

Note that SINGLE and DEMAND mode can not be invoked simultaneously.

AUTO AUTOINITIALIZE. When this bit is set to one, the AUTOINITIALIZE mode is selected. Under normal circumstances, the DMA controller transfers the specified number of bytes to/from the PC memory from the given starting address and terminates when completed. When the AUTOINITIALIZE mode is enabled, the DMA controller will reset the original byte count, reset the initial address, and repeat the transfer again. The AUTOINITIALIZE option is only in effect until the next KENTER or KOUTPUT buffered transfer is completed. Additionally, if a KENTER or KOUTPUT transfer in DMA CONTINUE mode is selected, the AUTOINITIALIZE option will only be in effect for that transfer.

Returns None.

Programming Notes

1. All DMA AUTOINITIALIZE transfers should occur entirely within a single DMA 64 KByte page. So, if a DMA buffer operation involves a transfer of more than 64K or occur across a DMA page boundary, the AUTOINITIALIZE option affects only those bytes in the last transfer occurring within one DMA page.

   Also note that DMA AUTOINITIALIZE transfers specified for buffers that are located in more than a single DMA page are unpredictable.

2. It is impossible for a program to halt a DMA AUTOINITIALIZE operation unless the DMA CONTINUE option is selected. This is because the driver does not return control to a program using non-CONTINUE operations until the transfer completes.

KDMA (cont'd)

Example

```
CONST
    CARDNUM = 0; (* Card Number Parameter For Calls *)
    AUTO    = 1; (* Auto Mode of DMA Transfer *)
    DEMAND  = 2; (* Demand Mode of DMA Transfer *)
    SINGLE  = 4; (* Single Mode of DMA Transfer *)

VAR
     ErrorFlag : WORD;  (* Word To Hold Error Code *)
BEGIN
(*
  *** Set Up DMA Mode To Single With Autoinitialize ***
*)
kdma(CARDNUM, ErrorFlag, SINGLE+AUTO);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KENTER

Purpose Allows the KM-488-DD to receive data from a GPIB bus device.

Syntax

```
VAR
  board_nr : INTEGER;
  error_code : WORD ;
  mode : INTEGER;
  count : INTEGER4;
  mygads : ARRAY[0..1] OF gads;
  darray : ARRAY[0..N-1] OF CHAR ;
  term_list: ARRAY[0..2] OF INTEGER;
...
BEGIN
  kenter(board_nr, err_code, darray, count, mode, mygads,
term_list)
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`darray` is a character array large enough to accept the returned data. If a non-zero count is specified, the array should be of size `count`. If `count` is 0 and terminator characters are used, then `darray` must be large enough to accommodate both data and terminators returned by the device.

`count` is a long integer representing the number of characters to be read. Valid counts are within the range 0 to 4294967295 (2^{32}-1) decimal, or from 0 to $FFFFFFFF hex. When 0 is used, the KENTER will stop when the termination specified by `term_list` is satisfied.

`mode` is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

KENTER (cont'd)

Where:

CONT	**CONTINUE.** This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer. If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked or a pre-specified GPIB BUS input terminator is detected.
DMA	**DMA.** If this bit is set to one, then DMA will be used in the data transfer.

mygads[n] represents the GPIB bus address of the device to be accessed. **mygads** is an array of **n** (**n** can range from 1 to 15) structures of type gads. See section 10.1 for more information.

term_list[3] is an integer array which defines the GPIB bus terminators to be used during the KENTER call. These terminators are used in conjunction with **count** to terminate the input process. Termination choices include whether or not EOI will be detected with the last byte and whether or not a certain byte or sequence of two bytes will be tested for as an end of message code. The default termination scheme is initially set-up by the *CONFIG.DAT* file, and can be modified by calling KTERM or KCONFIG. **termlist[3]** can either specify the use of the default terminators or temporary overrides.

termlist[3] is defined in the *tp488dd.inc* header file. **termlist[3]** is comprised of the following elements:

KENTER (cont'd)

`termlist[0]`: **EOI detection.** Possible values are:

-1($FFFF)	Use the default EOI setting.
0	Don't Detect EOI.
1	Detect EOI.

`termlist[1]`: **1st GPIB bus terminator.** Possible values are:

-1($FFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.
0 to 255	Detect this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

`termlist[2]`: **2nd GPIB bus terminator.** Possible values are:

-1($FFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
0 to 255	Detect this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

KENTER (cont'd)

The following default **termlist** can also be used:

```
VAR
      dtrm : ARRAY[0..2] OF INTEGER;
...
BEGIN
dtrm[0] := -1; (* Use the default EOI choice *)
dtrm[1] := -1; (* Use the default first terminator *)
dtrm[2] := -1; (* Use the default second terminator *)
```

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is a Peripheral, an active controller must address the KM-488-DD as a listener and some other device as the talker, before KENTER is called. In this case, you should use the **no_addr** structure in lieu of **mygads**. This is described in section 10.1.

2. If the KM-488-DD is the Active Controller, it will automatically address itself as a listener and another device defined in **mygads** as the talker. Since only one device can be a talker, the address group **mygads** can contain only one address. If a device has been previously addressed to talk and the KM-488-DD previously addressed to listen, then **mygads** can be **no_add** (See section 10.1) and the bus will not be re-addressed.

3. If **count** is specified as 0, then KENTER will terminate when the terminator(s) specified by **term_list[3]** is encountered.
4. If it is desirable to terminate on **count** only and not use the default terminators, then you must select no terminator overrides in the **termlist**, in addition to giving the count. In this case, a byte by byte synchronous transfer is executed.

5. If both counts and termination are specified, the KENTER will terminate on whichever condition is encountered first.

6. If the condition(s) for termination is not encountered, KTERM will timeout unless the timeout has been disabled.

Example

```
CONST
     CARDNUM  = 0 ;      (* Card Number Parameter For
Calls *)
     DMAXFER  = 1;       (* Transfer mode: non DMA = 0,
DMA without
                             continue = 1, DMA with
continue = 3 *)
     NUMD     = 1028;    (* Number Of Bytes To Transfer *)
VAR
  ErrorFlag : WORD ;     (* Word For Returned Error Code
*)
  add_05     : ARRAY[0..1] OF gads;
      (* Address Group For Device @ 05 *)
  sdata      : ARRAY[0..NUMD-1] OF CHAR ;
      (* Array For Data From Device    *)
  dtrm       : ARRAY[0..2] OF INTEGER;
      (* Array Of GPIB Terminators For "KENTER" And
"KOUTPUT" *)
BEGIN
(*
 *** Address Structure For Meter At GPIB Address 05
*)
add_05[0].primad := 5 ;
(* Instrument Has GPIB Address 5 *)
add_05[0].secad  := -1 ;
(* -1 Indicates No Secondary Address *)
add_05[1].primad := -1 ;
(* -1 Indicates No Other Instrument  *)
(*
  Structure To Signal:
     Use Default Terminators
     Detect EOI On ENTER/Assert EOI On Output
*)
dtrm[0]  := 1 ;
(* 1  : Assert EOI (KOUTPUT)/Detect EOI (KENTER) *)
dtrm[1]  := -1 ; (* -1 : Use Default 1st Terminator   *)
dtrm[2]  := -1 ; (* -1 : Use Default 2nd Terminator *)
WRITELN('ENTER DATA FROM METER');
kenter(0, ErrorFlag, sdata[0], NUMD, DMAXFER, add_05[0],
dtrm[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KERROR

Purpose Enables/Disables display of Error Messages.

Syntax

```
VAR
   board_nr : INTEGER;
   err_code : WORD;
   e_sw : INTEGER;
. . .
BEGIN
   kerror(board_nr, err_code, e_sw)
. . .
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

e_sw is an integer which determines if error message display function will be enabled or disabled. If **e_sw=1**, the error message display is enabled. (This is the default.) If **e_sw=0** is specified, the error message display is disabled.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If KERROR has been used to enable error messages, the message will only reveal that an error has occurred for the KM-488-DD calls. It does not identify what type of error occurred. The KDIAGON routine for should be used to help debug applications utilizing the KM-488-DD calls.

2. Programs can also check **err_code** after each call to identify the exact nature of call errors.

KERROR

Example

```
CONST
    CARDNUM     = 0 ;
(* Card Number Parameter For Calls *)
    ERR_ON      = 1 ;
(* Enable Error Display *)
VAR
     ErrorFlag : WORD;
 (* Word To Hold Error Code *)
BEGIN
(*
  *** Enable Error Display ***
*)
kerror(CARDNUM, ErrorFlag, ERR_ON);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KFILL

Purpose Defines the KM-488-DD driver's response to a request for data when none is available.

☞ ***NOTE:*** KFILL is provided for users who mix File I/O Commands and Calls in the same program. It only affects inputs performed using the File I/O Commands.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  fill_sw : INTEGER;
...
BEGIN
  kfill(board_nr, err_code, fill_sw)
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`fill_sw` is an integer which represents the fill condition. (The default condition returns a NULL character.) Valid integers are any one of the following:

-1	OFF. This type of response will not return any data characters or a DOS error message, if no input data is available.
-2	ERROR. This type of a response will not return any data characters, but will generate an Error Message. The error message number can then be trapped by calling the KSTATUS routine. (Refer to the KSTATUS routine for more information.)
0 - 255	ASCII. This type of response will return the designated ASCII character. (An ASCII Equivalence Chart is provided in Appendix A.) Specify the decimal equivalent of the character to be used.

KFILL (cont'd)

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Example

```
CONST
    CARDNUM    = 0 ;          (* Card Number Parameter For
Calls *)
    FILL_SW   = 88;           (* Use 'X' As Fill Character
*)
VAR
     ErrorFlag : WORD;         (* Word To Hold Error Code
*)
BEGIN
(*
  *** Set Fill Character To 'X' ***
*)
kfill(CARDNUM, ErrorFlag, FILL_SW);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KHELLO

Purpose Returns an identification string from the KM-488-DD driver.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  id_response :STRING;
...
BEGIN
   khello(board_nr, err_code, id_response)
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `id_response` will contain a string similar to: `copyright (c) 1991 Keithley MetraByte Corp`.

`err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The string into which the KHELLO message is returned must be dimensioned prior to calling KHELLO. Otherwise, Turbo PASCAL assumes that the string is of length 0 and calling KHELLO will result in destroying other programmed data.

KHELLO (cont'd)

Example

```
CONST
    CARDNUM    = 0 ; (* Card Number Parameter For Calls
*)
VAR
    indat     : STRING; (* Array To Hold Copyright
Message *)
    ErrorFlag : WORD;  (* Word To Hold Error Code *)
    i         : INTEGER;
(* Procedure To Determine String Length And Stuff It In
Location 0 *)
PROCEDURE char_count (VAR ch_strng:STRING);
BEGIN
  (**** Check For NULL or 255 Characters ****)
  i := 1 ;
  WHILE ((i<>255) AND (ch_strng[i] <> CHR(0))) DO i := i
+ 1 ;
  (* Assign Calculated Count To String Location 0 *)
  ch_strng[0] := CHR(i - 1) ;
END;

BEGIN
(*
  *** Return Driver Copyright Message ***
*)
khello(CARDNUM,  ErrorFlag, indat[1]);
IF (ErrorFlag > 0) THEN HALT(ErrorFLAG) ;
char_count(indat);    (* Update String Count In PASCAL's
Descriptor *)
END.
```

KLOCAL

Purpose Forces the specified bus device(s) to return to being programmed locally from their front panels. This routine can only be used if the KM-488-DD is acting as a System or Active Controller.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  mygads : ARRAY[0..1] OF gads;
BEGIN
klocal(board_nr, err_code, mygads);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] represents the GPIB bus address of the device to be accessed. **mygads** is an array of **n** (**n** can range from 1 to 15) structures of type gads. See section 10.1 for more information.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is the System Controller and if **mygads** does not specify an address, then the GPIB REN (Remote Enable) line is unasserted and all devices are returned to Local. In order to return them to remote mode, it will be necessary to issue a KREMOTE call. Likewise, if Local Lockout is required, it will be necessary to issue a KLOL call.

2. As an Active Controller, the KM-488-DD can issue the GPIB GTL (Go To Local) message to those devices specified by **mygads**. In this case, the GPIB REN (Remote Enable) line remains asserted and devices will return to remote when addressed to listen. If a KLOL (Local Lockout) call has been issued previously, it should still be in effect when a device is returned to Remote.

KLOCAL (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;
  (* Card Number Parameter For Calls *)
VAR
     add_12     : ARRAY[0..1] OF gads;
     (* Address Group (One Device At 12) *)
     ErrorFlag : WORD;
     (* Word For Code Returned By Calls  *)
(*
  *** Address Structure For Meter At GPIB Address 12 ***
*)
     add_12[0].primad:=12;
  (* 1st Device Pri. Address 12 *)
     add_12[0].secad:=-1;
  (* 1st Device No Sec. Address *)
     add_12[1].primad:=-1;
  (* -1 = No 2nd Device In Group *)
BEGIN
(*
  *** Send The Addressed Device To Local Control ***
*)
klocal(CARDNUM, ErrorFlag,add_12[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KLOL

Purpose

<u>This command can only be used if the KM-488-DD is an Active or System Controller.</u> It will disable the GPIB bus devices from being returned to Local Control by means of the Local/Remote button on the device.

☞ ***NOTE:*** This routine issues an IEEE-488 bus signal, LOL. This signal is not supported by all IEEE-488 bus devices.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
...
BEGIN
   klol(board_nr, err_code)
...
```

Parameters

`board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns

`err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The "LOCAL" button is disabled so that a device cannot be returned to local state for manual programming or control. The Active Controller can return specific devices to "local with lockout state", whereby an operator can then use the front panel controls. When the device is addressed to listen again, it returns to "remote with lockout state". Thus, the effect of the LOL call will remain until the REN line is unasserted (LOCAL) by the System Controller.

2. It is good practice to issue a KLOL so that devices are under strict program control. KLOL can be issued before a device is put in remote and will take effect when the device's LOCAL button is set to remote.

KLOL (cont'd)

Example

```
CONST
  CARDNUM    = 0 ;     (* Card Number Parameter For Calls
*)
VAR
  ErrorFlag : WORD ;   (* For Returned Error Code 0 = OK
*)
BEGIN
klol(CARDNUM, ErrorFlag);
(* Lock Out Local Control Of Devices *)
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KOUTPUT

Purpose Transmits data from the KM-488-DD to the GPIB bus.

Syntax

```
VAR
  board_nr : INTEGER;
  mode : INTEGER;
  err_code : WORD;
  darray : ARRAY[0..N-1] OF CHAR;
  count : INTEGER4;
  mygads : ARRAY[0..1] OF gads;
  term_list : ARRAY[0..2] OF INTEGER;
...
BEGIN
koutput(board_nr, err_code, darray, count, mode, mygads,
term_list)
...
```

Parameters

board_nr is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

darray is an array of data to output.

count is a long integer representing the number of characters to be output. Valid counts are within the range 1 to 4294967295 (2^{32}-1) decimal, or from 1 to $FFFFFFFF hex.

mode is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

CONT CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at

KOUTPUT (cont'd)

least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked.

DMA DMA. If this bit is set to one, then DMA will be used in the data transfer.

mygads[n] represents the GPIB bus address of the device to be accessed. **mygads** is an array of **n** (**n** can range from 1 to 15) structures of type gads. See section 10.1 for more information.

term_list[3] is an integer array which defines the GPIB bus terminators to be used during the KOUTPUT call. These terminators are used in conjunction with **count** to terminate the output process. Termination choices include whether or not EOI will be detected with the last byte and whether or not a certain byte or sequence of two bytes will be tested for as an end of message code. The default termination scheme is initially set-up by the *CONFIG.DAT* file, and can be modified by calling KTERM or KCONFIG. **termlist[3]** can either specify the use of the default terminators or temporary overrides.

termlist[3] is defined in the *tp488dd.inc* header file. **termlist[3]** is comprised of the following elements:

termlist[0]: **EOI detection.** Possible values are:

-1($FFFF)	Use the default EOI setting.
0	Don't Assert EOI.
1	Assert EOI.

termlist[1]: **1st GPIB bus terminator.** Possible values are:

-1($FFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.

KOUTPUT (cont'd)

	0 to 255	Send this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
termlist[2]:	**2nd GPIB bus terminator.** Possible values are:	
	-1($FFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
	0 to 255	Send this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default **termlist** can also be used:

```
VAR
     dtrm : ARRAY[0..2] OF INTEGER;
...
BEGIN
dtrm[0] := -1; (* Use the default EOI choice *)
dtrm[1] := -1; (* Use the default first terminator *)
dtrm[2] := -1; (* Use the default second terminator *)
```

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is a peripheral, then an Active Controller must address the KM-488-DD as a talker and some other device(s) as the listener(s). In this case, use **no_add** for **my_gads**. (See section 10.1.).

KOUTPUT (cont'd)

2. If the KM-488-DD is both the System and Active Controller, and **my_gads** contains the device(s) to be addressed, the KOUTPUT will automatically assert the GPIB REN (Remote Enable) line.

3. If the KM-488-DD is the Active Controller and KOUTPUT is called, it will automatically address itself as the talker and another device(s) contained in **my_gads** as the listener(s). If the devices have been previously addressed, then **my_gads** can be **no_add** and the bus will not be re-addressed.

4. A non-zero **count** must always be specified.

5. If you do not wish to append terminator characters, then be sure to set the terminator via *CONFIG.DAT* or by calling KTERM or KCONFIG and select no terminator overrides in the **termlist** array.

6. If a listener does not accept a character, a timeout will occur unless the timeout has been disabled.

KOUTPUT (cont'd)

Example

```
CONST
  DMAXFER  = 1;
   (* Transfer mode: non DMA = 0, DMA without
      continue = 1, DMA with continue = 3 *)
  XFER        = 0;
   (* Normal Non-DMA Style Transfer *)
  NUMD        = 1028;
   (* Number Of Points To OUTPUT    *)
VAR
  ErrorFlag : WORD ;
 (* Word To Hold Error Code Returned By
    Each Call (0 = false = no errors) *)

  no_add     : ARRAY[0..1] OF gads;
   (* Addr Group Signaling No Addressing *)
  dtrm       : ARRAY[0..2] OF INTEGER;
   (* Array Of GPIB Terminators For "KENTER" And "KOUTPUT"
*)
  sdata      : ARRAY[0..NUMD-1] OF CHAR;
    (* Array Of Data To OUTPUT *)
BEGIN

(* Structure For No Default Terminator/EOI Overrides *)
dtrm[0] := 1 ;   (* 1: Assert EOI (koutput)/ Detect EOI
(kenter) *)
dtrm[1] := -1 ; (* -1: Use Default 1st Terminator
*)
dtrm[2] := -1 ; (* -1 : Use Default 2nd Terminator
*)
```

KOUTPUT (cont'd)

```
 (* Address Structure For Not Readdressing The GPIB Bus *)
no_add[0].primad := -1 ;    (* -1 Indicates No New Address
*)
no_add[0].secad  := -1 ;    (* -1 Indicates No Secondary
Address*)
no_add[1].primad := -1 ;    (* -1 Indicates No Other
Instruments *)
(* .
   .
   sdata Array Initialized Here
   .
   . *)
koutput(CARDNUM, ErrorFlag, sdata[0], NUMD, DMAXFER,
no_add[0], dtrm[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KPASCTL

Purpose Designates another controller to be the Active Controller. The KM-488-DD must be an Active Controller.

Syntax

```
VAR
  board_nr: INTEGER;
  err_code : WORD;
  mygads    : ARRAY[0..14] OF gads;
...
BEGIN
  kpasctl(board_nr, err_code, mygads);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads[n] contains the GPIB bus address of the device to become the new Active Controller. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. **mygads** structure is defined in the include file *tp488dd.inc* and is described in section 10.1.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the KM-488-DD which is relinquishing its position as the Active Controller is also a System Controller, it retains its status as System Controller.

KPASCTL (cont'd)

Example

```
CONST
     CARDNUM     = 0 ;        (* Card Number Parameter For
Calls *)
VAR
  ErrorFlag : WORD;           (* Word For Code Returned By
Calls  *)
  add_02     : ARRAY[0..14] OF gads;
   (* Address Group (One Device At 02) *)
BEGIN
(* Address Structure For Peripheral At GPIB Address 02*)
add_02[0].primad:= 2;
(* 2 : 1st Instrument Has GPIB Address 02*)
add_02[0].secad:= -1;
(* -1 : 1st Instrument Has No Secondary Address *)
add_02[1].primad:=-1;
(* -1 : No 2nd Instrument In Address Group  *)

(**** PASS CONTROL TO SECOND KM-488-DD ****)
WRITELN('Pass Control To Second KM-488-DD');
kpasctl(CARDNUM, ErrorFlag, add_02[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KPPOLL

Purpose Initiates a Parallel Poll. The KM-488-DD must be the Active Controller.

☞ ***NOTE:*** Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  pprtn : INTEGER;
...
BEGIN
kppoll(board_nr, err_code, pprtn);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

pprtn is an integer variable which will contain an integer in the range 0 to 255 decimal. This integer indicates which Data Lines which have been asserted (DIO1-DIO8).

Programming Notes None.

KPPOLL

Example

```
CONST
     CARDNUM  = 0 ;(* Card Number Parameter For Calls *)
VAR
     ErrorFlag : WORD;
   (* Integer to Hold Error Code Returned By
     Each CALL (0 = false = no error) *)

     pp_resp  : INTEGER; (* Integer For Parallel Response
*)
(*
  *** Conduct Parallel Poll And Print Result ***
*)
kppoll(CARDNUM, ErrorFlag, pp_resp);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
WRITELN('pp_resp= ',pp_resp);
END.
```

KPPC

Purpose Configures the Parallel Poll response of a GPIB bus device. The KM-488-DD must be the Active Controller.

☞ ***NOTE:*** Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  ppc_cfg  : INTEGER;
  mygads     : ARRAY[0..14] OF gads;
...
BEGIN
  kppoll(board_nr, err_code, mygads, ppc_cfg);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads[n] represents the GPIB bus address of the device to be accessed. **mygads** is an array of **n** (**n** can range from 1 to 15) structures of type gads. See section 10.1 for more information.

ppc_cfg is an integer which represents the Parallel Poll Response of the device to be programmed. This integer is of the format:

Bit	7	6	5	4	3	2	1	0
	0	1	1	0	S	P2	P1	P0

KPPC (cont'd)

Where:

S	is the parallel poll response value (0 or 1) that the device uses to respond to the parallel poll when service is required. This bit is generally set to 1.

☞ ***NOTE:*** This value must correspond to the setting of the GPIB bus device's *ist* (individual status) bit. Refer to the Device's documentation for more information.

P2 P1 P0	is a 3-bit value which tells the device being configured which data bit (DIO1 through DIO8) it should use as its parallel poll response.

Returns **`err_code`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPC (cont'd)

Example

```
CONST
     CARDNUM     = 0 ;
(* Card Number Parameter For Calls *)
VAR
     pp_enab    : INTEGER;
(* Integer For Parallel Poll Enabling*)
     add_1      : ARRAY[0..1] OF gads;
    (* Address Group (will have one device @ 1) *)
BEGIN
(*
  Device Supporting PPOLL Is At Address 01
*)
add_1[0].primad := 1 ;
(* 1st Instrument Has GPIB Address 01  *)
add_1[0].secad  := -1 ;
(* -1 : 1st Instrument Has No Secondary Addr  *)
add_1[1].primad := -1 ;
(* -1 : No 2nd Instrument In Address Group    *)
(*
  Configure Device To Assert DIO3 When SRQ Is Asserted
*)
pp_enab := 106;
kppc(CARDNUM,  ErrorFlag, add_1[0], pp_enab) ;
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KPPD

Purpose Disables the Parallel Poll response capability of the specified GPIB bus device(s). The KM-488-DD must be the Active Controller.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  mygads    : ARRAY[0..14] OF gads;
...
BEGIN
  kppd(board_nr, err_code, mygads);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads[n] represents the GPIB bus address of the device to be accessed. **mygads** is an array of **n** (**n** can range from 1 to 15) structures of type gads. See section 10.1 for more information.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPD (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;    (* Card Number Parameter For
Calls *)
VAR
     ErrorFlag : WORD;   (* Holds Returned Error Code*)
     add_1     : ARRAY[0..1] OF gads;
  (* Address Group (will have one device @ 1) *)

(*
  Address Structure For Device At GPIB Address 01
*)
add_1[0].primad := 1 ;
  (* 1st Instrument Has GPIB Address 01         *)
add_1[0].secad  := -1 ;
  (* -1 : 1st Instrument Has No Secondary Addr  *)
add_1[1].primad := -1 ;
  (* -1 : No 2nd Instrument In Address Group    *)
(*
  *** Disable Parallel Poll Of Addressed Devices ***
*)
kppd(CARDNUM,  ErrorFlag, add_1[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KPPU

Purpose Disables the Parallel Poll Response of all GPIB bus device(s). The KM-488-DD must be the Active Controller.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
...
BEGIN
  kppu(board_nr, err_code);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
CONST
     CARDNUM  = 0 ;         (* Card Number Parameter For
Calls *)
VAR
     ErrorFlag : WORD;
       (* Integer to Hold Error Code Returned By Each
CALL (0 = false = no error) *)
BEGIN
(*
  *** Disable Parallel Poll Response Of All Bus Devices
***
*)
kppu(CARDNUM,  ErrorFlag);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KQUIKSTAT

Purpose Returns the KM-488-DD's status byte.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  qstat : INTEGER;
...
BEGIN
  kquikstat(board_nr, err_code, qstat);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **qstat** is a pointer to the returned status integer. The status integer is of the following format:

Bit	15-9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PEND	DET	SRQ	DEC	LA	TA	DMA

Where:

ADSC	**Address Status Change.** If this bit is set to 1, a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).
CIC	**Controller in Charge.** If this bit is set to 1, the KM-488-DD is an Active Controller.
PEND	**SRQ Pending.** If this bit is set to 1, the KM-488-DD has an SRQ request pending.
DET	**Device Triggered.** If this bit is set to 1, a GPIB Trigger command has been received.
SRQ	**Internal SRQ.** If the KM-488-DD is an Active Controller and this bit is set to one, a device is requesting service. If the KM-488-DD is a Peripheral and this bit is set to one, then its SRQ (issued by KREQUEST) has not been serviced.

KQUIKSTAT(cont'd)

DEC	**Device Cleared.** If this bit is set to one, the KM-488-DD has received a GPIB Clear command.
LA	**Listen Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Talker and can output data to the GPIB.
DMA	**DMA.** A DMA transfer is currently in progress.

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KQUIKSTAT(cont'd)

Example

```
CONST

    ADSC = #100 ;

VAR
  board_nr : INTEGER;
  err_code : WORD;
  qstat : INTEGER;
...
BEGIN
  QSTAT := 0 ;    (* Initialize QSTAT To 0 *)
  kquikstat(board_nr, err_code, qstat);

  IF (QSTAT AND ADSC) THEN
     WRITELN('ADDRESS STATUS CHANGED') ;
  ...
```

KREMOTE

Purpose Forces the GPIB bus device(s) to the remote mode (ignore the bus).

☞ ***NOTE:*** The KM-488-DD must be a System Controller to execute this command.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  mygads    : ARRAY[0..14] OF gads;
...
BEGIN
  kremote(board_nr, err_code, mygads);
...
```

Parameters **`board_nr`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads[n]` represents the GPIB bus address of the device to be accessed. **`mygads`** is an array of **`n`** (**`n`** can range from 1 to 15) structures of type gads. See section 10.1 for more information.

Returns **`err_code`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KREMOTE (cont')

Programming Notes

1. If **mygads** does not specify an address (i.e., **no_addr** is used), then the GPIB REN (Remote Enable) line is asserted. Devices will not be in the remote mode until addressed to listen. If **mygads** contains address then those devices will also be addressed to listen, so they will be put in remote.

2. It is good programming practice to issue a Local Lockout to prevent the device(s) from being returned to Local mode.

Example

```
CONST
     CARDNUM    = 0 ;
 (* Card Number Parameter For Calls *)
VAR
     ErrorFlag : WORD;
(* Integer to Hold Error Code Returned By Each CALL (0 =
false = no error) *)
     add_05     : ARRAY[0..1] OF gads;
  (* Address Group (will have one device @ 05) *)
BEGIN
(*
  *** Address Structure For Device At GPIB Address 5 ***
*)
add_05[0].primad := 5 ;
(* 1st Instrument Has GPIB Address 05 *)
add_05[0].secad  := -1 ;
(* -1 : 1st Instrument Has No Secondary Addr  *)
add_05[1].primad := -1 ;
(* -1 : No 2nd Instrument In Address Group    *)
(*
  Assert REN And Listen Specified Devices
*)
kremote(CARDNUM,  ErrorFlag, add_05);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KREQUEST

Purpose Sets the Serial Poll Response of a KM-488-DD which is a Peripheral.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
 sp : INTEGER;
...
BEGIN
  krequest(board_nr, err_code, sp);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

sp is an integer in the range 0 to 255 which represents the serial poll response of the device. This integer is of the following format:

Bit	7	6	5	4	3	2	1	0
	DIO8	rsv	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
rsv	If this bit is 1, the KM-488-DD will generate a Service Request (assert SRQ).

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The Active Controller can read the value of **sp** by serial polling the KM-488-DD. This will clear bit 6, if it was set.

2. Use KQUIKSTAT or KSPOLL to check if the Peripheral has been serial polled (checks the status of the SRQ bit).

KREQUEST (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
     ASSERTSRQ  = 64;
(* Set Reserved SRQ Bit In Status Byte *)
     ErrorFlag : WORD;
(* Word For Code Returned By Calls  *)
BEGIN
(*
  *** Generate SRQ ***
*)
krequest(CARDNUM, ErrorFlag, ASSERTSRQ);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KRESET

Purpose Performs a "warm" reset of the KM-488-DD and the GPIB bus.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
...
BEGIN
 kreset(board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The KM-488-DD is reset to the following conditions:

   - STOP
   - DISARM
   - CONFIG
   - ABORT
   - BUFFER INCREMENT
   - DMA SINGLE
   - ERROR ON
   - FILL NULL
   - LOCAL
   - REQUEST 0 (If Peripheral)
   - Clear CHANGE, TRIGGER, and CLEAR STATUS
   - TIMEOUT 10

   It also clears all error conditions.

2. If the KM-488-DD is the System Controller, it will assert the GPIB IFC (Interface Clear) line for at least 500 µsecs.

KRESET

Example

```
CONST
  CARDNUM    = 0 ;   (* Card Number Parameter For Calls *)
VAR
  ErrorFlag : WORD;
(* Word For Code Returned By Calls  *)
BEGIN
(*
  *** Reset Driver ***
*)
kreset(CARDNUM,  ErrorFlag) ;
IF (ErrorFlag > 0) THEN
 HALT(ErrorFlag) ;
END.
```

KRESUME

Purpose

Initiates data transfers between two non-Active Controller GPIB devices, by unasserting the ATN line. The KM-488-DD must be an Active Controller in order to use this function.

Syntax

```
VAR
  board_nr  : INTEGER;
  err_code  : WORD;
. . .
BEGIN
kresume(board_nr, err_code);
. . .
```

Parameters

`board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns

`err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

This routine is normally called after a KSEND has addressed a talker and a listener. (See KSEND description.)

Example

```
CONST
  CARDNUM    = 0 ;
   (* Card Number Parameter For Calls *)
VAR
  ErrorFlag : WORD;
 (* Word For Code Returned By Calls  *)
BEGIN
(*
  Drop ATN Line To Allow Inter-Peripheral Communication
*)
kresume(CARDNUM, ErrorFlag);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KSEND

Purpose Sends GPIB commands and data from a string.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  stringlen : INTEGER;
  sendstr : STRING;
...
BEGIN
  ksend(board_nr, err_code, sendstr, stringlen);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`sendstr` is a string which includes any of the following subcommands: CMD, UNT, MTA, MLA, TALK, LISTEN, DATA, and EOI. Multiple subcommands may be specified; however, the length of the entire SEND command string cannot exceed 255 characters. The subcommands are as follows:

CMD - Sends information with the ATN line asserted. This indicates to the bus devices that the characters are to be interpreted as GPIB bus commands. The command may be sent as either a quoted string (e.g., **'COMMAND'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **#0D** hex for a Carriage Return). Hex values must be preceded by #. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **CMD 67,79,77,77,65,78,68**).

An EOI cannot be sent with this subcommand, because an EOI with ATN asserted would initiate a Parallel Poll.

UNT - Untalks all devices by sending a GPIB UNTALK command with ATN asserted.

KSEND (cont'd)

UNL - Unlistens all devices by sending the GPIB UNLISTEN command with ATN asserted.

MTA - Designates the KM-488-DD to address itself to talk by sending the GPIB MTA (My Talk Address) command with the ATN line asserted.

MLA - Designates the KM-488-DD to address itself to listen by sending the GPIB MLA (My Listen Address) command with the ATN line asserted.

`TALK addr` - Addresses another GPIB device or KM-488-DD to talk by sending a GPIB TAG (Talk Address Group) command with the ATN line asserted. **`addr`** is an integer representing the GPIB BUS device address of the device to talk. This integer ranges from 00 to 30 decimal.

`LISTEN addr` - Addresses another GPIB device(s) or KM-488-DD('s) to listen by sending a GPIB LAG (Listen Address Group) command with ATN asserted. **`addr`** is an integer representing the GPIB BUS device address of the device(s) to talk. This integer ranges from 00 to 30 decimal. Multiple listeners can be specified. If **`addr`** is not specified, all other devices on the GPIB BUS will be designated listeners.

KSEND (cont'd)

DATA - Sends information with the ATN line unasserted. This indicates to the bus devices that the characters are to be interpreted as data. This is the technique which is to be used to send device-dependent commands. (See the IEEE-488 Tutorial in Appendix C for more information.) The data may be sent as either a quoted string (i.e., **'DATA'**) or as a numeric equivalent of an individual ASCII character (i.e., **13** decimal or **#0D** hex for a Carriage Return). Hex values must be preceded by #. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **DATA 68,65,84,65**).

This sub-command is useful when you are sending commands which are unique to your interface.

EOI - Sends information with the ATN line unasserted. EOI will be asserted when the last character is transmitted. This information is interpreted as GPIB bus data and may be sent as either a quoted string (e.g., **'xyz'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **#0D** hex for a Carriage Return). Hex values must be preceded by #. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **SEND EOI 120,121,122**).

stringlen is an integer between 0 and 255 which represents the length of the SEND string.

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Typically, KSEND is used to have to device transfer data without it passing through the KM-488-DD. For example, a KM-488-DD might talk a scope and send a trace to a listened plotter.

2. The maximum length of the KSEND command, including any subcommands, is 255 characters. To KSEND large amounts of data, use multiple KSEND commands.

KSEND (cont'd)

3. KSEND should only be used when a non-conforming device requires a special command sequence or a non-standard GPIB command. Do not use the KSEND command unless you are extremely familiar with GPIB.

4. The KM-488-DD must be the Active Controller to KSEND commands. Any KM-488-DD can KSEND data.

5. If a DATA subcommand is not included in the KSEND string, be sure to call KRESUME immediately after the KSEND. This is necessary because the ATN line must be dropped so that the transfer will proceed.

6. Do not include the word SEND within the KSEND string as you might do in the File I/O SEND command.

Example

```
CONST
  CARDNUM   = 0 ;
   (* Card Number Parameter For Calls *)
VAR
  snstr     : STRING(70);
(* For Command Used In KSEND      *)
  ErrorFlag : WORD;
(* Word For Code Returned By Calls  *)
BEGIN
(*
  HAVE 196 DMM SEND READING TO A PERIPHERAL KM-488-DD
  DMM Is At Address 12
  Peripheral KM-488-DD Is At Address 02
*)
snstr := 'UNT UNL LISTEN 02 TALK 12';
ksend(CARDNUM, ErrorFlag, snstr[1], 25);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
```

KSPOLL

Purpose If the KM-488-DD is an Active Controller, KSPOLL will check for an SRQ or conduct a serial poll. If the KM-488-DD is a Peripheral, KSPOLL will check if the KM-488-DD's SRQ has been serviced.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  spr : INTEGER;
  mygads     : ARRAY[0..14] OF gads;
...
BEGIN
  kspoll(board_nr, err_code, spr, mygads);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] represents the GPIB bus address of the device to be accessed. **mygads** is an array of n (n can range from 1 to 15) structures of type gads. See section 10.1 for more information.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

spr is an unsigned integer or an array of unsigned integers with an entry for each address in **mygads**. The value of the integer is 0 to 255 and corresponds to a byte with the following bitmap:

Bit	7	6	5	4	3	2	1	0
	DIO8	SRQ	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

KSPOLL(cont'd)

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
SRQ	If this bit is 1, the SRQ (Service Request) line will be asserted to request servicing. Otherwise, SRQ will not be asserted.

Bit 6 has the special significance of indicating an SRQ pending. The significance of the other bits will be application dependent.

Programming Notes

1. The most common use of KSPOLL is for the KM-488-DD, as an active controller, to issue KSPOLL with **`mygads`** containing the address of a single device. This addresses and serial polls the device and, upon return, **`spr`** contains the serial poll response of the polled device. If the SRQ bit in **`spr`** is set, the device had issued an SRQ. The other bits in **`spr`**are device-dependent. Serial polling a device which is currently asserting SRQ will cause the device to unassert SRQ.

   The string returned by KSTATUS or the integer word by KQUIKSTAT can be checked to determine the presence of an SRQ before a serial poll is conducted. If only one device is asserting SRQ, the effect of issuing KSPOLL will be to clear the internal SRQ pending "bit", even if the polled device is not issuing the SRQ. It is also the case when multiple devices are asserting SRQ and a device not currently asserting SRQ is polled. In this case, a subsequent use of KSTATUS or KQUIKSTAT will not reveal a pending SRQ.

   To aid in identifying which bus device(s) is currently requesting service, a KM-488-DD which is the Active Controller can serial poll as many as 14 devices by issuing a KSPOLL call with **`mygads`** containing the address of more than one device. In this case, **`spr`** should be an array of unsigned integers with an element for each address. In this case, KSPOLL will serial poll each addressed device and return the serial poll bytes in the **`spr`** array. This is a faster way for discerning the source of an SRQ among several devices.

KSPOLL (cont'd)

2. If the KM-488-DD is a peripheral and KSPOLL is called with **mygads** equal to the no address structure (**no_addr**), **spr** will be the serial poll response (**sp**) set by KREQUEST - with the possible exception of the SRQ bit. (This may have disappeared as a result of an Active Controller having serial polled the KM-488-DD since the last KREQUEST call.) In the instance of an Active Controller serial polling a peripheral KM-488-DD, the peripheral KM-488-DD unasserts the SRQ line. For an example, see KREQUEST. The state of the SRQ bit can also be determined from the string returned by KSTATUS or the integer status word set by a KQUIKSTAT call.

3. If the KM-488-DD is the Active Controller and issues KSPOLL with **mygads** equal to **no_addr**, **spr** will be 64 if a device is asserting SRQ and 0 if not. This same result could have been determined from KQUIKSTAT or KSTATUS.

   Note that if this call is issued when a SRQ is pending, it internally "clears" the SRQ "pending" bit, even though the requesting device has not been polled and is still issuing an SRQ. That is, a subsequent KSPOLL, KQUIKSTAT, or KSTATUS call will not reveal a pending SRQ.

KSPOLL(cont'd)

Example

```
CONST
   CARDNUM     = 0 ;
(* Card Number Parameter For Calls *)
VAR
   add_12     : ARRAY[0..1] OF gads;
  (* Address Group (One Device At 12) *)
   ErrorFlag : WORD;
 (* Word For Code Returned By Calls  *)
BEGIN
(*
  *** Address Structure For Meter At GPIB Address 12 ***
*)
add_12[0].primad:=12;
     (* 1st Instrument Has GPIB Address 12  *)
add_12[0].secad:=-1;
     (* -1 Indicates 1st Instrument Has No Secondary Add
*)
add_12[1].primad:=-1;
     (* -1 Indicates No 2nd Instrument In Group *)
(*
  Acknowledge/Remove SRQ Of The Device @ Address 12
*)
kspoll(CARDNUM,  ErrorFlag, add_12[0], sbyt) ;
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag) ;
END.
```

KSTATUS

Purpose Returns a character string describing the current operating state of the KM-488-DD.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
  stat : STRING;
...
BEGIN
  kstatus(board_nr, err_code, stat);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`stat` is a character string which describes the current operating state of the KM-488-DD. The string is of the format shown in Table 10-2. `stat` should be 75 bytes in order to contain the longest message that might be returned. If it is too short, data will be written over other portions of the program causing problems.

Table 10-2. Status String

Starting Col	# of Cols.	Name and Description

1	1	Operating Mode - Tells if the KM-488-DD is acting as an *Active Controller* or *Peripheral*. Can be **C** or **P** where:
		
		C = The KM-488-DD is an *Active Controller*
		P = The KM-488-DD is a *Peripheral*

3	2	Bus Address - Gives the IEEE-488 Bus Address of the KM-488-DD. This is a two-digit decimal integer ranging from 00 to 30.
6	2	Addressed Status Change - Indicates if the device has changed addressed state, i.e., if it has cycled between the *Talker, Listener,* or *Active Controller* states. This is reset whenever a STATUS is read. This can be one of the following: G0 = There has not been a change in the addressed status. G1 = There has been a change in the addressed status.

9	1	Current Addressed State - Indicates if the KM-488-DD is currently acting as a *Talker* or *Listener*, or is *Idle*. Can be T, L, or I, where:
		
		T = The KM-488-DD is a Talker
		L = The KM-488-DD is a Listener
		I = The KM-488-DD is Idle

11	2	Service Request - Represents the current internal SRQ status where: S0 = SRQ not asserted S1 = SRQ asserted If the KM-488-DD is in the *Active Controller* mode, the internal SRQ state indicates if a device is asserting SRQ. If the KM-488-DD is acting as a *Peripheral*, the internal SRQ state is set by using the KREQUEST call. It is cleared by a serial poll from the Active Controller and indicates it is asserting SRQ.

KSTATUS(cont'd)

Table 10-2. Status String

Starting Col	# of Cols.	Name and Description
14	3	Error Code - Indicates whether or not an error has occurred. The three characters consist of the letter 'E' followed by a two digit error code. (A list of these error codes is provided in Appendix G.) An error code of 00 indicates no error has occurred.
18	2	Triggered State - Indicates if a IEEE-488 trigger command has been received or not. This is not updated during DMA CONTINUE transfers. Triggered State values are: T0 = No trigger command has been received. T1 = A trigger command has been received.
21	2	Cleared - Indicates if the IEEE Clear command has been received or not. This is not updated during DMA CONTINUE transfers. Cleared values are: C0 = No Clear command has been received. C1 = A Clear command has been received.

24	2	Transfer Status - Indicates if a data transfer is in progress. Values which may appear are:
		
		P0 = No transfer
		P1 = DMA CONTINUE Transfer occurring
		P2 = DMA AUTOINITIALIZE Transfer occurring

27	up to 45	Error Message Text - Contains the Error Message Text associated with the given error code (EXX). These error messages are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is the Active Controller, the Service Request Status may be cleared (i.e, S0) as a result of a KSPOLL, even when the SRQ line is still asserted. See the discussion in KSPOLL.

2. If the KSTATUS string is read into a variable of type STRING (as opposed to a character array), the string length must be calculated upon return from the KSTATUS call and inserted in location 0 of the STRING variable. The string length can be calculated by searching through the STRING (starting with location STRING[1]) for a NULL character (0).

KSTATUS (cont'd)

Example

```
CONST
      CARDNUM    = 0 ;
  (* Card Number Parameter For Calls *)
VAR
      indat      : STRING(255);
 (* Array To Hold Status Message *)
      ErrorFlag : WORD;
  (* Word To Hold Error Code *)
      i          : INTEGER;
 (*
   Proc. To Determine String Length And Put It In Loc.0
 *)
PROCEDURE char_count (VAR ch_strng:STRING);
BEGIN
 (*
   Check For NULL or 255 Characters
 *)
   i := 1 ;
   WHILE ((i<>255) AND (ch_strng[i] <> CHR(0))) DO
       i := i + 1 ;
 (*
   Assign Calculated Count To String Location 0
 *)

   ch_strng[0] := CHR(i - 1) ;
END;
BEGIN
 (*
   *** Return Status Message ***
 *)
kstatus(CARDNUM, ErrorFlag, st[1]) ;   (* Request Status
*)
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
char_count(st);
WRITELN('STATUS :', st);
END.
```

KSTOP

Purpose Stops a DMA CONTINUE transfer.

Syntax

```
VAR
  board_nr : INTEGER;
  err_code : WORD;
...
BEGIN
  kstop (board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the DMA CONTINUE transfer has been completed, a KSTOP will have no effect.

2. Use the KBUFD call to determine the actual number of characters which were transferred before the KSTOP took effect.

Example

```
CONST
  CARDNUM     = 0 ;
 (* Card Number Parameter For Calls *)
VAR
  ErrorFlag : WORD ;
   (* Word To Hold Error Code Returned By Each
      Call (0 = false = no errors) *)
BEGIN
(*
  *** Stop DMA ***
*)
kstop(CARDNUM,  ErrorFlag);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KTERM

Purpose Changes the default GPIB bus terminator(s) to be used in ENTER and/or OUTPUT calls.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
 trm_list[4]; ARRAY [0...3] OF INTEGER
...
BEGIN
  kterm(board_nr, err_code, trm_list);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`term_list` is an integer array which defines the GPIB bus terminators. See Chapter 6 for more information regarding the use of GPIB bus terminators. (`termlist[4]` is defined in the *tp488dd.inc* header file.) `termlist[4]` is comprised of the following elements:

`termlist[0]`: **Input/Output GPIB Terminators.** Possible values are:

-1	Change the Input Terminators.
0	Change both Input and Output Terminators.
1	Change the Output Terminators.

`termlist[1]`: **EOI detection/Assertion.** Possible values are:

0	No assert/detect EOI
1	Assert/Detect EOI.

KTERM (cont'd)

`termlist[2]`: **1st GPIB bus terminator.** Possible values are:

-1($FFFF)	Don't use first or second GPIB bus terminator.
0 to 255	Use this ASCII character as the first GPIB bus terminator instead of the default first GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

`termlist[3]`: **2nd GPIB bus terminator.** Possible values are:

-1($FFFF)	Don't use second GPIB bus terminator.
0 to 255	Use this ASCII character as the second GPIB bus terminator instead of the default second GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

Returns **`err_code`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. KTERM will change the default bus terminators set during software installation, as directed by **`termlist[]`**.

KTERM (cont'd)

Example

```
CONST
  CARDNUM    = 0 ;
(* Card Number Parameter For Calls *)
VAR
  ErrorFlag : WORD ;
  (* Word To Hold Error Code Returned By
     Each  Call (0 = false = no errors) *)
  trm_lst    : ARRAY[0..3] OF INTEGER;
  (* Array Of GPIB Terminators For "KTERM" *)
BEGIN
(*
  *** Configure Terminators For BUS Operations
*)
(*
 *** Structure To Reset Configuration Terminators ***
*)
trm_lst[0] := 0 ;
(* 0 Indicates Change Input AND Output Terminators *)
trm_lst[1] := 1 ;
(* 1 = Assert EOI On Output, Detect EOI On Enter   *)
trm_lst[2] := -1 ;
(* -1 Indicates No 1st Terminator *)
trm_lst[3] := -1 ;
(* -1 Indicates No 2nd Terminator *)
kterm(CARDNUM,  ErrorFlag, trm_lst[0]) ;
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag) ;
END.
```

KTO

Purpose Changes the timeout period.

Syntax

```
VAR
 board_nr : INTEGER
 err_code : WORD;
 tval : INTEGER4;
...
BEGIN
 kto(board_nr, err_code, tval);
...
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

tval is the number of 55 ms. timer ticks to allow before a time-out.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. To suppress Timeout checking, set **tval** to 0.

2. If a DMA CONTINUE transfer is in progress, the KM-488-DD will check for timeouts only for the first byte that is transmitted/received. During other types of transfers, the KM-488-DD will check for a timeout between transmission of bytes.

You must be certain to check that a DMA CONTINUE transfer has been completed.

KTO

Example

```
CONST
     CARDNUM    = 0 ;
    (* Card Number Parameter For Calls *)
     FIFTEEN  = 273 ;
    (* 15 seconds at 18.2 ticks per second *)
VAR
  ErrorFlag : WORD ;
    (* Word To Hold Error Code Returned By
       Each Call (0 = false = no errors) *)
BEGIN
(*
  Set Time Out To 15 Seconds (Approx 18.2 Ticks/Sec)
*)
kto(CARDNUM,  ErrorFlag, FIFTEEN);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KTRIGGER

Purpose Triggers the specified device(s). The KM-488-DD must be an Active Controller.

Syntax

```
VAR
  board_nr : INTEGER
  err_code : WORD;
  mygads     : ARRAY[0..14] OF gads;
...
BEGIN
  ktrigger(board_nr, err_code, mygads);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads[n]` represents the GPIB bus address of the device to be accessed. `mygads` is an array of `n` (`n` can range from 1 to 15) structures of type gads. See section 10.1 for more information.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The devices listed in `mygads` will be triggered.

KTRIGGER (cont'd)

Example

```
CONST
     CARDNUM    = 0 ;        (* Card Number Parameter For
Calls *)
VAR
     add_12    : ARRAY[0..1] OF gads;
   (* Address Group (One Device At 12) *)
     ErrorFlag : WORD;
   (* Word For Code Returned By Calls  *)
(*
  *** Address Structure For Meter At GPIB Address 12 ***
*)
add_12[0].primad:=12;
    (* 1st Instrument Has GPIB Address *)
add_12[0].secad:=-1;
     (* -1 Indicates 1st Instrument Has No Secondary Add
*)
add_12[1].primad:=-1;
    (* -1 Indicates No 2nd Instrument In Group *)
BEGIN
(*
  *** Trigger The Device ***
*)
ktrigger(CARDNUM, ErrorFlag,add_12[0]);
IF (ErrorFlag > 0) THEN err_handler(ErrorFlag);
END.
```

KWAITC

Purpose Forces the KM-488-DD driver to wait until a DMA CONTINUE transfer has been completed before returning control to the application program.

Syntax

```
VAR
 board_nr : INTEGER;
 err_code : WORD;
...
BEGIN
 kwaitc(board_nr, err_code);
...
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Time out checking, if enabled, is performed while KWAITCing.

2. This is an <u>explicit</u> wait. The following calls perform an <u>implicit</u> wait:

   - KABORT
   - KCLEAR
   - KENTER
   - KLOCAL
   - KLOCAL LOCKOUT
   - KOUTPUT
   - KPASS CONTROL
   - KPPOLL
   - KPPOLL CONFIG
   - KPPOLL DISABLE
   - KPPOLL UNCONFIG
   - KREMOTE
   - KREQUEST
   - KRESUME
   - KSEND
   - KSPOLL
   - KTRIGGER

KWAITC

Example

```
CONST
     CARDNUM    = 0 ;
 (* Card Number Parameter For Calls *)
VAR
     ErrorFlag : WORD;
 (* Word For Code Returned By Calls  *)
BEGIN
(*
  *** Wait For DMA To Finish ***
*)
(*
      .
DMA CONTINUE started here.
      .
*)
kwaitc(CARDNUM,  ErrorFlag);
IF (ErrorFlag > 0) THEN
     WRITELN('Time Out Waiting For DMA To Finish') ;
END.
```

CHAPTER 11 USING THE CALL INTERFACE IN C

Versions Supported

Microsoft C version 4.0 and later
TURBO C version 1.0 and later

Preparing the Environment

The C support files and example programs are located on the C Release Disk. Be sure that all of these files are copied to your working directory.

File Header

When you write your program, make sure to include the line:

```
#include "km488dd.h"
```

This header file contains all the necessary function prototypes, structures, etc.

Compiling The Program

Compile your program in the normal manner, being sure to link it with the library *KM488IFC.OBJ*. For example, when working in Microsoft C, at the DOS prompt, type either:

```
cl yourprog.c /link km488ifc.obj
```

or

```
cl /c yourprog.c;
link yourprog + km488ifc;
```

Software Configuration

A number of KM-488-DD configuration parameters are set via the *CONFIG* program. (See Chapter 2.) These govern the default settings of the GPIB input and output bus terminators, lightpen interrupt enable, device timeout periods, and the KM-488-DD's Base Address. The defaults for these are listed in Table 11-1. There are other defaults you may have to re-program if you are using File I/O Commands in the same program as the library interface routines (calls).

Table 11-1. Default KM-488-DD Operating Parameters

Parameter	Default
Device Timeout	10.0 seconds
GPIB Bus Input Terminator	CR LF with no EOI
GPIB Bus Output Terminator	CR LF with no EOI
EOL Terminators	CR LF

The KCONFIG call can be used to reset the GPIB input and output bus terminators.

The KTERM call can be used to change the default GPIB bus terminators settings.

The KTO call can be used to change the default device timeout value.

Programming Notes

1. Any arguments which are passed as values may also be passed as constants.

2. "Strings" in C are actually character arrays. Thus, any KM-488-DD routines which require a string for input or output will need a character array. The far address of this character array is passed into the KM-488-DD Routine.

3. Any KM-488-DD routine which returns a value into a string requires an additional parameter. This defines the total number of bytes available as string space for storage of received data.

4. For calls which return strings rather than data bytes, (e.g. KSTATUS, KHELLO), it is very important that the number of bytes allocated for storage within a character array is at least one greater than the maximum byte count of the expected string. This extra byte is necessary so that a NULL can mark the end of the received data. If a routine attempts to receive more bytes than have been allocated for storage into that variable, other internal program variables may be overwritten, producing unexpected results or a program crash.

5. Note that function and parameter names in C are case-sensitive. The KM-488-DD routine names must appear in lower-case.

6. Do not name any of your variables with the same name as any of the KM-488-DD routines. This will cause a linker error.

11.1 CALL DESCRIPTION FORMAT

Each call description is divided into several sections as described below.

Syntax This section describes the syntax required by each call. The following rules are used in this section.

Call names and input parameters appear in `this typeface`.

Parameters This section describes the input parameters. In some instances, a structure or array of multiple parameters may be specified (for example, multiple device addresses).

`mygads[n]` is used in many calls. This is an array of `n` (1 to 15) structures of type `gads` which contain the GPIB address information required by the call. This GPIB address structure consists of:

```
struct    gads{
     int primad ;    /*primary address 0-30*/
     int secad ;     /*secondary address 0-31*/
     };
```

When `secad` is set equal to -1, it indicates that no secondary address is applicable. When `primad` is set equal to -1, it terminates the array.

You must tailor `mygads` to your application. The `mygads` structure is defined in the include file *km488dd.h*. An example of a structure with two devices is:

```
struct gads exam[3]
exam.primad[0] = 3 /* first device - pri addr 3*/
exam.secad[0] = -1 /* and no sec addr      */
exam.primad[1] = 15 /* second device - pri addr 15 */
exam.secad[1] = 2  /* and sec addr 2 */
exam.primad[2] = -1 /* there are only two devices */
```

If the KM-488-DD is a Peripheral (in which case the call will not require a GPIB bus address), you will want to use the following **mygads** structure:

```
struct gads no_add[1]
no_add.primad[0] = -1  /* there are no devices in group
*/
```

Returns This part of the callable description describes any data which will be returned after execution of the callable has been completed.

err_code - All calls except KDIAGON and KDIAGOFF return an integer error code which will be 0 if no error was detected. It is good practice to check **err_code** after each call and provide a routine to handle errors if they arise.

Programming Notes This section lists any special programming considerations for the routine.

Example This section gives a programming example using the routine.

KABORT

Purpose This routine allows the KM-488-DD to take control of the GPIB bus by forcing all other devices on the bus to the idle state. The KM-488-DD must be the Active Controller.

Syntax

```
int board_nr;
int err_code;
kabort (board_nr, &err_code);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the KM-488-DD is both the Active and System Controller, KABORT asserts the IFC line to retain control of the bus. Otherwise, the KM-488-DD will assert the ATN line, and then untalk and unlisten all devices.

Example

```
board_nr = 0;
kabort (board_nr, &err_code);
if (err_code) myerrorhandler (err_code);
```

KARM

Purpose Defines the conditions under which the lightpen status becomes true.

Syntax

```
int board_nr, arm_code;
int err_code;
karm(board_nr, &err_code, arm_code);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

arm_code is a 16-bit integer describing on which conditions a light pen interrupt may occur. The integer is of the format:

Bit	15-9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC	**Address Status Change.** If this bit is set to 1, the lightpen status will become true when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller). This encompasses all the conditions marked with an asterisk (*) below.
CIC*	**Controller in Charge.** If this bit is set to 1, the lightpen status will become true when the control is passed to the KM-488-DD and it becomes the Active Controller.
PER*	**Peripheral.** If this bit is set to 1, the lightpen status will become true when the KM-488-DD passes control to another device and becomes a Non-Active Controller.
DET	**Device Triggered.** If this bit is set to 1, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.

KARM (cont'd)

SRQ	**Internal SRQ.** If this bit is set to one, the lightpen status will become true when the KM-488-DD's internal Service Request Bit (SRQ) is set.
DEC	**Device Cleared.** If this bit is set to one, the lightpen status will become true when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA*	**Listen Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Listener and can accept data from the GPIB.
TA*	**Talk Addressed.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is addressed as a Talker and can output data to the GPIB.
IDLE*	**Idle.** If this bit is set to one, the lightpen status will become true when the KM-488-DD, acting as a Peripheral, is unaddressed as a Talker or a Listener.

☞ ***NOTE:*** The conditions marked by an asterisk (*) will force the lightpen status to true only when the KM-488-DD's address status has changed. This is indicated by the state of the Address Change Bit in the Status byte. See the kstatus or kquikstat routine descriptions for more information.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The driver must configured for lightpen emulation at load time via the *CONFIG.DAT* "pipe" file, as the default for lightpen emulation..

KARM (cont'd)

Example

```
#define LA 0x4
#define PERI 0x40
.
.
.
.
int err_code = 0;
int board_nr = 0;
karm(board_nr, &err_code, PERI | LA);
if (err_code) my errorhandler (err_code);
```

KBUF

Purpose Selects in which direction the memory is to be addressed for both DMA and non-DMA buffered transfers.

Syntax

```
int dir, board_nr;
int err_code;
kbuf(board_nr, &err_code, dir);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`dir` selects the direction in which the buffer is to be read. If `dir` is equal to 1, the buffer address will be incremented. If `dir` is equal to 0, the buffer address will be decremented.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the buffer's address is to be incremented during a bus I/O operation, be sure to use the lowest address in the designated buffer. Likewise, if it is to be decremented, select the highest address.

Example

```
#define inc 1
#define dec 0
.
.
.
int board_nr;
int err_code = 0;
kbuf(board_nr, &err_code, dec);
if (err_code) myerrorhandler (err_code);
```

KBUFD

Purpose　Returns the number of characters transferred in the last buffered transfer or, in the case of DMA transfers, the current transfer.

Syntax

```
int board_nr;
int err_code;
long bcount;
kbufd(board_nr, &err_code, &bcount);
```

Parameters　`board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns　`err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`bcount` represents the number of characters which were transferred during the last buffered transaction. This will be a long integer (32-bits) in the range 0 to 1,048,575 (0 to 20^{20}-1).

Programming Notes

1. If the data was transferred in the DMA CONTINUE mode, KBUFD returns the number of characters transferred thus far.

2. It is useful to call this routine after a buffered KENTER has been performed. This will help you to determine if the expected number of characters has been received and/or the transfer has terminated as the result of a detected terminator.

Example

```
long xfercount;
int board_nr = 0, err_code = 0;
kbufd(board_nr, &err_code, &xfercount);
if (err_code) myerrorhandler (err_code);
else printf("%lu Samples Transferred",xfercount);
```

KCLEAR

Purpose

The KM-488-DD must be an Active Controller. Resets device(s) to their power-up parameters.

Syntax

```
int board_nr;
int err_code;
struct gads mygads[n];
kclear(board_nr, &err_code, &mygads);
```

Parameters

board_nr is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads[n] represents the GPIB bus address of the device to be cleared. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. The **mygads** structure is defined in the include file *km488dd.h* and is described in section 11.1.

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If **mygads** contains no addresses, then the universal GPIB DCL (Device Clear) command is issued and all devices will be cleared.

2. If **mygads** contains addresses, then those devices specified will be listened and sent a GPIB SDC (Select Device Clear) Command.

Example

```
struct gads mygads[2] = {{2, -1},
                         {-1, -1}};
int board_nr = 0, err_code = 0 ;
kclear(board_nr, &err_code, mygads);
if (err_code) myerrorhandler (err_code);
```

KCONFIG

Purpose Resets the KM-488-DD to its default operating parameters or to those specified.

Syntax

```
int board_nr, err_code;
char cfg_string[50];
int stringlen;
kconfig(board_nr, &err_code, sym_name, stringlen);
```

Parameters **`board_nr`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`cfg_string` is a far character pointer to a user-defined string. This string can be comprised of any of the following options:

`/T` *`[term term EOI]`* Sets the input and output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **`/T`**, the GPIB bus terminators will be reset to their defaults.

`term` is an optional parameter which represents a terminator byte to be used. This terminator byte can be represented as any of the following:

`$char` where **`char`** is an integer representing the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix B for ASCII Equivalents.) **`char`** must be preceded by a dollar sign ($).

KCONFIG (cont'd)

Hexadecimal values must also be preceded by &H. For example, $84 represents the letter "T" as does $&H54.

CR	This represents the *Carriage Return* character (13 decimal, 0D hex).
LF	This represents the *Line Feed* character (10 decimal, 0A hex).
'X	where **X** represents a printable ASCII character. The character must be preceded by an apostrophe ('), for example: **'B** represents the character *B*.

EOI (End or Identify) is an optional GPIB BUS terminator. If included, the KM-488-DD will detect/assert the GPIB bus EOI line. (This would indicate that the last character has been sent.) Data will continue to be read until this terminator, a valid terminator sequence, or both are detected.

/TI *[term term EOI]*	Sets the input GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TI**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/TO *[term term EOI]*	Sets the output GPIB bus terminators. You can select one or two GPIB bus terminators followed by EOI or just EOI. If you just specify **/TO**, the input GPIB bus terminator will be reset to none. **term** and **EOI** are described above.
/E *[term term]*	Sets the input and output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **term** is described above.

KCONFIG (cont'd)

/EI *[term term]*	Sets the input EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.) **term** is described above.
/EO *[term term]*	Sets the output EOL terminators. (Note that this should only be used if the application program also includes File I/O Commands.)
/N name addr *[sec_addr]*	Configures a named device. Sets its address to the given value and its GPIB bus terminators to the current GPIB bus terminator settings. (Note that this should only be used if the application program also includes File I/O Commands utilizing the named device configurations.) **name** is the name of the device. **addr** is the primary GPIB address (00 to 30). **sec_addr** is the secondary GPIB address (00 to 31).

☞ ***NOTE:*** Do not create a named device with the same name as an existing directory on the current working disk.

If **cfg_string** is empty, then the configuration will revert to the installation configuration.

stringlen is an integer from 0 to 255 which gives the length of the **cfg_string** string.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KCONFIG (cont'd)

Programming Notes

1. The **/E**, **/EI**, **/EO**, and **/N** options will only take effect if File I/O Commands are used within the same application program as the callables.

2. If you are converting an application program previously written using the File I/O commands to callables, be sure to note that the CONFIG string for the KCONFIG call does not include the "CONFIG" keyword (i.e., as you would if you were using the CONFIG File I/O command).

3. Primary and Secondary Addresses must be two characters long, e.g. 01 or 0209.

Example

```
char far *cfg = "/T CR LF EOI";
int board_nr=0, err_code=0;
.
.
.
kconfig(board_nr, &err_code, cfg, strlen(cfg));
if (err_code) myerrorhandler (err_code);
```

KDIAGON

Purpose Turns file diagnostics on. (Default is off.)

Syntax

```
char filename[]= "MYDIAG.DAT"
kdiagon(filename, strlen(filename));
```

Parameters **`filename`** is string containing the name of the file to which the diagnostic information is to be written.

`strlen` is the length of the string representing the name of the file.

Returns This will echo the value of the call parameters into a disk file.

Programming Notes None.

Example

```
char FileName[] = "MYDIAG.DAT"
kdiagon(FileName, strlen(FileName)) ;
```

KDIAGOFF

Purpose Temporarily disables file diagnostics.

Syntax `kdiagoff()`

Parameters None.

Returns None.

Programming Notes None.

Example

```
char FileName[] = "MYDIAG.DAT"
kdiagon(FileName, strlen(FileName)) ;

/* Disable File Diagnostics */

kdiagoff() ;
```

KDISARM

Purpose Resets previously set conditions which cause lightpen interrupts.

Syntax

```
int board_nr, arm_code;
int err_code;
kdisarm(board_nr, &err_code, arm_code);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

arm_code is a 16-bit integer describing which conditions should be reset. The integer is of the format:

Bit	15-9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PER	DET	SRQ	DEC	LA	TA	IDLE

Where:

ADSC	**Address Status Change.** If this bit is set to 1, light pen status will not change when a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).
CIC	**Controller in Charge.** If this bit is set to 1, light pen status will not change when the control is passed to the KM-488-DD and it becomes the Active Controller.
PER	**Peripheral.** If this bit is set to 1, light pen status will not change when the KM-488-DD passes control to another device and becomes a Non-Active Controller (Peripheral).
DET	**Device Triggered.** If this bit is set to 1, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Trigger command.
SRQ	**Internal SRQ.** If this bit is set to one, light pen status will not change when the KM-488-DD's internal Service Request Bit (SRQ) is set.

KDISARM (cont'd)

DEC	**Device Cleared.** If this bit is set to one, light pen status will not change when a KM-488-DD, acting as a Peripheral, receives a GPIB Clear command.
LA	**Listen Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, light pen status will not change when the KM-488-DD is addressed as a Talker and can output data to the GPIB.
IDLE	**Idle.** If this bit is set to one, light pen status will not change when the KM-488-DD is unaddressed as a Talker or a Listener.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
#define ASC 0x100;
#define DT 0x20;
.
.
.
/*Turn off lightpen active on ASC and DT*/
int board_nr = 0, err_code=0;
kdisarm(board_nr, &err_code, ASC | DT);
if (err_code) myerrorhandler (err_code);
```

KDMA

Purpose Sets DMA transfer mode.

Syntax

```
int board_nr, dma_mode;
int err_code;
kdma (board_nr, &err_code, dma_mode);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

dma_mode is a 16-bit integer which represents the DMA mode to be used. If no **dma_mode** is given, then single mode without autoinitialization will be assumed. The **dma_mode** integer is interpreted as follows:

Bit	15 - 3	2	2	0
	0	SIN	DEM	AUTO

Where:

SIN SINGLE. When this bit is set to one, the SINGLE mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus and transfers a single byte of data. Control of the bus is then returned to the microprocessor.

Note that SINGLE and DEMAND mode can not be invoked simultaneously.

DEM DEMAND. When this bit is set to one, the DEMAND mode is selected. In this mode, when the DMA Request line is asserted the DMA controller assumes control of the bus. The DMA controller retains control of the bus until the DMA request signal is unasserted. Once this signal has been unasserted for more than one microprocessor clock cycle, control of bus is returned to the microprocessor. This mode allows the DMA controller chip to pass data at a slightly faster rate and the microprocessor to access the bus when it is not needed.

Note that SINGLE and DEMAND mode can not be invoked simultaneously.

KDMA (cont'd)

AUTO AUTOINITIALIZE. When this bit is set to one, the AUTOINITIALIZE mode is selected. Under normal circumstances, the DMA controller transfers the specified number of bytes to/from the PC memory from the given starting address and terminates when completed. When the AUTOINITIALIZE mode is enabled, the DMA controller will reset the original byte count, reset the initial address, and repeat the transfer again. The AUTOINITIALIZE option is only in effect until the next KENTER or KOUTPUT buffered transfer is completed. Additionally, if a KENTER or KOUTPUT transfer in DMA CONTINUE mode is selected, the AUTOINITIALIZE option will only be in effect for that transfer.

Returns None.

Programming Notes

1. All DMA AUTOINITIALIZE transfers should occur entirely within a single DMA 64 KByte page. So, if a DMA buffer operation involves a transfer of more than 64K or occur across a DMA page boundary, the AUTOINITIALIZE option affects only those bytes in the last transfer occurring within one DMA page.

   Also note that DMA AUTOINITIALIZE transfers specified for buffers that are located in more than a single DMA page are unpredictable.

2. It is impossible for a program to halt a DMA AUTOINITIALIZE operation unless the DMA CONTINUE option is selected. This is because the driver does not return control to a program using non-CONTINUE operations until the transfer completes.

Example

```
#define SINGLE 0x4
#define AUTO 0x1
int err_code=0, board_nr=0;

/* next (and only next) dma will be single and auto */
kdma(board_nr, &err_code, SINGLE+AUTO);
if (err_code) myerrorhandler (err_code);
```

KENTER

Purpose Allows the KM-488-DD to receive data from a GPIB bus device.

Syntax

```
int board_nr, mode, err_code, term_list[3];
char huge *data_ptr;
unsigned long count;
struct gads mygads[n];
kenter(board_nr, &err_code, data_ptr, count, mode, &my-
gads, &term_list);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`data_ptr` is a pointer to the starting buffer position. Be sure to allocate an area of memory large enough to accept the returned data. If a non-zero count is specified, the data area can be equal to `count`. If `count` is 0 and termination characters are used, the data area must be large enough to accommodate both the data returned by the device and the termination characters.

`count` is a long integer representing the number of characters to be read. Valid counts are within the range 0 to 4294967295 (2^{32}-1) decimal, or from 0 to 0xFFFFFFFF hex. When 0 is used, the KENTER will stop when the termination specified by `term_list` is satisfied.

`mode` is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

CONT CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

KENTER (cont'd)

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked or a pre-specified GPIB BUS input terminator is detected.

DMA	DMA. If this bit is set to one, then DMA will be used in the data transfer.

`mygads [n]` represents the GPIB bus address of the device to be reset. **`mygads`** is a pointer to an array of **`n`** (**`n`** can range from 1 to 15) structures of type gads. You must tailor **`mygads`** to your application. The **`mygads`** structure is defined in the include file *km488dd.h* and is described in section 11.1.

`term_list [3]` is an integer array which defines the GPIB bus terminators to be used during the KENTER call. These terminators are used in conjunction with **`count`** to terminate the input process. Terminator choices include whether or not EOI will be detected to signal end of input and which character sequence, if any, is to be detected as an end of message code. The default terminators scheme is initially set-up by the *CONFIG.DAT* file, and can be modified by calling KTERM or KCONFIG. **`termlist`** can either specify the used of the default terminators or temporary overrides. **`termlist`** is defined in the *km488dd.h* header file. **`termlist [3]`** is comprised of the following elements:

`termlist [0]` : **EOI detection.** Possible values are:

-1(0xFFFF)	Use the default EOI setting.
0	Don't Detect EOI.
1	Detect EOI.

`termlist [1]` : **1st GPIB bus terminator.** Possible values are:

-1(0xFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.

KENTER (cont'd)

	0 to 255	Detect this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
termlist[2]:	**2nd GPIB bus terminator.** Possible values are:	
	-1(0xFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
	0 to 255	Detect this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default **termlist** can also be used:

```
int dtrm[3];
dtrm[0] = -1; /* Use the default EOI choice */
dtrm[1] = -1; /* Use the default first terminator */
dtrm[2] = -1; /* Use the default second terminator
                 (redundant since dtrm[1] = -1) */
```

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KENTER (cont'd)

Programming Notes

1. If the KM-488-DD is a Peripheral, an active controller must address KM-488-DD as a listener and some other device as the talker, before KENTER is called. In this instance, you should use the **no_addr** structure in lieu of **mygads**. This is described in section 11.1.

2. If the KM-488-DD is the Active Controller, it will automatically address itself as a listener and another device defined in **mygads** as the talker. Since only one device can be a talker, the address group **mygads** can contain only one address. If a device has been previously addressed to talk and the KM-488-DD previously addressed to listen, then **mygads** can be **no_add** (See section 11.1) and the bus will not be re-addressed.

3. If **count** is specified as 0, then KENTER will terminate when the terminator(s) specified by **term_list** is encountered.

4. If it is desirable to terminate on **count** only and not use the default terminators, then you must select no terminator overrides in **term_list**, in addition to giving the count. In this case, a byte by byte synchronous transfer is executed.

5. If both count and terminators are specified, the KENTER will terminate on whichever condition is encountered first.

6. If the condition(s) for termination is not encountered, KTERM will timeout unless the timeout has been disabled.

Example

```
/*Enter data with CR LF EOI from addr.12*/

int board_nr = 0, err_code = 0, trmlist[3] = {1,13,10};
struct gads mygads[2]= {{12,-1}, {-1,-1}};
unsigned long xfercount = 1000;
int mode = 0; /*don't use DMA*/
char datarray[1000] = {0}; /*target array*/
kenter(board_nr,&err_code,datarray,xfercount,mode,mygads,
termlist);
if (err_code) myerrorhandler (err_code);
```

KERROR

Purpose Enables/Disables display of Error Messages.

Syntax

```
int board_nr, err_code, e_sw;
kerror(board_nr, &err_code, e_sw);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

e_sw is an integer which determines if error message display function will be enabled or disabled. If **e_sw=1**, the error message display is enabled. (This is the default.) If **e_sw=0** is specified, the error message display is disabled.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If KERROR has been used to enable error messages, the message will only reveal that <u>an error has occurred</u> for the KM-488-DD calls. It does not identify what type of error occurred. The KDIAGON routine for should be used to help debug applications utilizing the KM-488-DD calls.

2. Programs can also check **err_code** after each call to identify the exact nature of the call errors.

Example

```
#define err_on 1
int board_nr = 0, err_code = 0, err_switch = err_on;
kerror(board_nr, &err_code, err_switch);
if (err_code) myerrorhandler (err_code);
```

KFILL

Purpose Defines the KM-488-DD driver's response to a request for data when none is available.

☞ ***NOTE:*** KFILL is provided for users who mix File I/O Commands and Calls in the same program. It only affects inputs performed using the File I/O Commands.

Syntax

```
int board_nr, err_code, fill_sw;
kfill(board_nr, &err_code, fill_sw);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`fill_sw` is an integer which represents the fill condition. (The default condition returns a NULL character.) Valid integers are any one of the following:

-1	OFF. This type of response will not return any data characters or a DOS error message, if no input data is available.
-2	ERROR. This type of a response will not return any data characters, but will generate an Error Message. The error message number can then be trapped by calling the KSTATUS routine. (Refer to the KSTATUS routine for more information.)
0 - 255	ASCII. This type of response will return the designated ASCII character. (An ASCII Equivalence Chart is provided in Appendix A.) Specify the decimal equivalent of the character to be used.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Example

```
/* pad DOS inputs with 'X' character */
int err_code = 0, board_nr = 0, fill_switch = (int) 'X';
kfill(board_nr, &err_code, fill_switch);
if (err_code) myerrorhandler (err_code);
```

KHELLO

Purpose Returns an identification string from the KM-488-DD driver.

Syntax

```
int board_nr, err_code;
char id_response [50];
khello(board_nr, &err_code, id_response);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `id_response` will contain a string similar to: `copyright (c) 1991 Keithley MetraByte Corp`.

`err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The string into which the HELLO message is returned must be dimensioned prior to calling KHELLO. Otherwise, C assumes the string is of length 0 and calling KHELLO will result in destroying other programmed data.

Example

```
/* Array for ID String Returned by "KHELLO" command*/

  char idstr[50];
  int err_code = 0;

/* Return Driver Copyright Message*/

  khello(0,&ErrorFlag, idstr);
  if (ErrorFlag)
{
  printf("\n\nError : Driver Not Installed / Exit and
Install KM.EXE");
  exit(1);
}
```

KLOCAL

Purpose

Forces the specified bus device(s) to return to being programmed locally from their front panels. This routine can only be used if the KM-488-DD is acting as a System or Active Controller.

Syntax

```
int board_nr, err_code;
struct gads mgads[n];
klocal(board_nr, &err_code, mygads);
```

Parameters

board_nr is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] represents the GPIB bus address of the device to be sent to local. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. The **mygads** structure is defined in the include file *km488dd.h* and is described in section 11.1.

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is the System Controller and if **mygads** does not specify an address, then the GPIB REN (Remote Enable) line is unasserted and all devices are returned to Local. In order to return them to remote mode, it will be necessary to issue a KREMOTE call. Likewise, if Local Lockout is required, it will be necessary to issue a KLOL call.

2. As an Active Controller, the KM-488-DD can issue the GPIB GTL (Go To Local) message to those devices specified by **mygads**. In this case, the GPIB REN (Remote Enable) line remains asserted and devices will return to remote when addressed to listen. If a KLOL (Local Lockout) call has been issued previously, it should still be in effect when a device is returned to Remote.

KLOCAL (cont'd)

Example

```
/* Address Group (will have one device @ 05) */

   struct gads add_05[15] ;
   int ErrorFlag = 0;

/* Address Structure For Device With GPIB Address 05 */

/* 1st instrument has GPIB primary address 05, no second-
ary address */

   add_05[0].primad = 05 ;
   add_05[0].secad = -1 ;
   add_05[1].primad = -1 ;  /* no 2nd instrument */

/*  Put Device In Local Mode*/

    klocal(0, &ErrorFlag, add_05) ;
    if (ErrorFlag) err_handler(ErrorFlag) ;
```

KLOL

Purpose This command can only be used if the KM-488-DD is an Active or System Controller. It will disable the GPIB bus devices from being returned to Local Control by means of the Local/Remote button on the device.

☞ ***NOTE:*** This routine issues an IEEE-488 bus signal, LOL. This signal is not supported by all IEEE-488 bus devices.

Syntax

```
int board_nr, err_code;
klol(board_nr, &err_code);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The "LOCAL" button is disabled so that a device cannot be returned to local state for manual programming or control. The Active Controller can return specific devices to "local with lockout state", whereby an operator can then use the front panel controls. When the device is addressed to listen again, it returns to "remote with lockout state". Thus, the effect of the LOL call will remain until the REN line is unasserted (LOCAL) by the System Controller.

2. It is good practice to issue a KLOL so that devices are under strict program control. KLOL can be issued before a device is put in remote and will take effect when the device's LOCAL button is set to remote.

Example

```
/* Integer To Hold Error Code Returned By Calls*/

    int ErrorFlag;

/* Lock Out Local Control Of Devices */

    klol(0, &ErrorFlag);
    if (ErrorFlag)
    err_handler(ErrorFlag);
```

KOUTPUT

Purpose Transmits data from the KM-488-DD to the GPIB bus.

Syntax

```
int board_nr, mode, err_code, term_ptr;
char huge *data_ptr;
unsigned long count;
struct gads mgads[n];
koutput(board_nr, &err_code, data_ptr, count, mode, &my-
gads, &term_list);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`data_ptr` is a pointer to the starting buffer position.

`count` is a long integer representing the number of characters to be output. Valid counts are within the range 1 to 4294967295 (2^{32}-1) decimal, or from 1 to 0xFFFFFFFF hex.

`mode` is an integer which represents whether or not DMA is to be used and if the CONTINUE mode is to be used. This integer is interpreted as follows:

Bit	15 - 2	1	0
	0	CONT	DMA

Where:

CONT CONTINUE. This an optional mode which is used in conjunction with DMA. When this bit is set to one, the CONTINUE mode will be used. If CONTINUE is specified, the KM-488-DD will return control to the application program as soon as it can without waiting for completion of the transfer. The KM-488-DD will at least wait for the first byte to check for time-out (Unless TIME OUT = 0) before continuing. DMA CONTINUE execution concludes when the KM-488-DD completes its transfer.

KOUTPUT (cont'd)

If CONTINUE is used and the DMA AUTOINITIALIZE has been enabled, then the DMA transfer will continue until a KSTOP routine is invoked.

DMA	DMA. If this bit is set to one, then DMA will be used in the data transfer.

`term_ptr` is a pointer to **`termlist[3]`** integer array. (This is defined in the *km488dd.h* header file.) This array defines the GPIB bus terminators to be used in the KOUTPUT. **`termlist[3]`** is comprised of the following elements:

`mygads[n]` represents the GPIB bus address of the device to receive the data. **`mygads`** is a pointer to an array of **`n`** (**`n`** can range from 1 to 15) structures of type gads. You must tailor **`mygads`** to your application. **`mygads`** structure is defined in the include file *km488dd.h* and is described in section 11.1.

`term_list[3]` is an integer array which defines the GPIB bus terminators to be used during the KOUTPUT call. These terminators are used in conjunction with **`count`** to terminate the output process. Termination choices include whether or not EOI will be asserted with the last byte and whether or not a certain byte or sequence of two bytes will be sent as an end of message code. The default terminator scheme is initially set-up via *CONFIG.DAT*, and can be modified by calling KTERM or KCONFIG. **`termlist`** can be used to specify the use of the default terminators or a different terminator.

`termlist[3]` is defined in the *km488dd.h* header file. **`termlist[3]`** is comprised of the following elements:

`termlist[0]`: **EOI detection.** Possible values are:

-1(0xFFFF)	Use the default EOI setting.
0	Don't Assert EOI.
1	Assert EOI.

`termlist[1]`: **1st GPIB bus terminator.** Possible values are:

-1(0xFFFF)	Use the default terminator. Note that if the default first terminator is used, the default second terminator will also be used.

KOUTPUT (cont'd)

	0 to 255	Send this ASCII character as the first GPIB bus terminator instead of the default first GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)
`termlist[2]`:	2nd GPIB bus terminator. Possible values are:	
	-1(0xFFFF)	Use the default terminator. Note that if the default first terminator is used, this will automatically be selected.
	0 to 255	Send this ASCII character as the second GPIB bus terminator instead of the default second GPIB Input Terminator. The 0 - 255 value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

The following default `termlist` can also be used:

```
int dtrm[3];
dtrm[0] = -1; /* Use the default EOI choice */
dtrm[1] = -1; /* Use the default first terminator */
dtrm[2] = -1; /* Use the default second terminator
                 (redundant since dtrm[1] = -1) */
```

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the KM-488-DD is a peripheral, then an Active Controller must address the KM-488-DD as a talker and some other device(s) as the listener(s). In this case, use `no_add` for `my_gads`. (See section 11.1.).

2. If the KM-488-DD is both the System and Active Controller, and `my_gads` contains the device(s) to be addressed, the KOUTPUT will automatically assert the GPIB REN (Remote Enable) line.

KOUTPUT (cont'd)

3. If the KM-488-DD is the Active Controller and KOUTPUT is called, it will automatically address itself as the talker and another device(s) contained in **my_gads** as the listener(s). If the devices have been previously addressed, then **my_gads** can be **no_add** and the bus will not be re-addressed.

4. A non-zero **count** must always be specified.

5. If you do not wish to append terminator characters, then be sure to set the terminator defaults via *CONFIG.DAT* or by calling KTERM or KCONFIG and select no terminator overrides in **term_list**.

6. If a listener does not accept a character, a timeout will occur unless the timeout has been disabled.

Example

```
/*  Message To Display On Digital Scope Screen */
   char init_mess[] = { "MESS 10:\"  INITIALIZE SCOPE\""
};
   int dtrm[3] ;

/*  Address Group (will have one device @ 05) */
   struct gads add_05[15] ;

/* Address Structure For Scope With GPIB Address 05 */
  add_05[0].primad = 05 ;
  add_05[0].secad = -1 ;
  add_05[1].primad = -1 ;   /* no 2nd instrument */

/* Array With Default Terminators*/
   dtrm[0] = 1 ;     /* Assert EOI On Last Output Byte*/
   dtrm[1] = -1 ;    /* Do Not Override 1st Terminator */
   dtrm[2] = -1 ;    /* Do Not Override 2nd Terminator */

/* Write Initialization Message on Scope Screen */
  koutput(0, &ErrorFlag, init_mess, strlen(init_mess), 0,
add_05, dtrm) ;
  if (ErrorFlag) err_handler(ErrorFlag);
```

KPASCTL

Purpose Designates another controller to be the Active Controller. The KM-488-DD must be an Active Controller.

Syntax

```
int board_nr, err_code;
struct gads mgads[n];
kpasctl(board_nr, &err_code, &mygads);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] contains the GPIB bus address of the device to become the new Active Controller. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. **mygads** structure is defined in the include file *km488dd.h* and is described in section 11.1.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes If the KM-488-DD which is relinquishing its position as the Active Controller is also a System Controller, it retains its status as System Controller.

KPASCTL (cont'd)

Example

```
/* Address Group (will have one device @ 02) */
  int ErrorFlag = 0;
  struct gads add_02[2];

/* 1st instrument has GPIB primary address 02 and no
secondary address */

  add_02[0].primad = 2 ;
  add_02[0].secad = -1 ;
  add_02[1].primad = -1 ;  /*  no 2nd instrument  */

/* Pass Control to a Peripheral KM-488-DD At Bus Address
02*/

   kpasctl(0,&ErrorFlag,add_02);
   if (ErrorFlag) err_handler(ErrorFlag);
```

KPPOLL

Purpose Initiates a Parallel Poll. The KM-488-DD must be the Active Controller.

☞ *NOTE:* Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax

```
int board_nr, err_code, pprtn;
kppoll(board_nr, &err_code, &pprtn);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

pprtn is an integer variable which will contain an integer in the range 0 to 255 decimal. This integer indicates which Data Lines which have been asserted (DIO1-DIO8).

Programming Notes None.

Example

```
/* Integer For Parallel Poll Response */

     int pp_resp, ErrorFlag;

/*   Conduct Parallel Poll And Print Result */

     kppoll(0, &ErrorFlag, &pp_resp);
     if (ErrorFlag) err_handler(ErrorFlag);
     printf("\nParallel Poll Response = %u\n",pp_resp);
```

KPPC

Purpose Configures the Parallel Poll response of a GPIB bus device. The KM-488-DD must be the Active Controller.

☞ ***NOTE:*** Many GPIB devices do not support parallel polling. Check your device's documentation.

Syntax

```
int board_nr, err_code, ppc_cfg;
struct gads mgads[n];
kppoll(board_nr, &err_code, &mygads, ppc_cfg);
```

Parameters **`board_nr`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads[n]` contains the GPIB bus address(es) of the device(es) to be configured. **`mygads`** is a pointer to an array of **`n`** (**`n`** can range from 1 to 15) structures of type gads. You must tailor **`mygads`** to your application. **`mygads`** structure is defined in the include file *km488dd.h* and is described in section 11.1.

`ppc_cfg` is an integer which represents the Parallel Poll Response of the device to be programmed. This integer is of the format:

Bit	7	6	5	4	3	2	1	0
	0	1	1	0	S	P2	P1	P0

KPPC (cont'd)

Where:

S	is the parallel poll response value (0 or 1) that the device uses to respond to the parallel poll when service is required. This bit is generally set to 1.

☞ ***NOTE:*** This value must correspond to the setting of the GPIB bus device's *ist* (individual status) bit. Refer to the Device's documentation for more information.

P2 P1 P0	is a 3-bit value which tells the device being configured which data bit (DIO1 through DIO8) it should use as its parallel poll response.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPC (cont'd)

Example

```
/* Address Group (will have one device @ 1) */

struct gads add_1[15] ;
int ErrorFlag;
int pp_enab;             /* integer for parallel poll
enabling */

/* Set-up 1st instrument with primary address 1 (hex 1),
no secondary address */

add_1[0].primad=0x1;
add_1[0].secad=-1;
add_1[1].primad=-1;  /* no 2nd instrument */

/*  Configure Instrument At Address 01 To Assert DIO3
when SRQ is Asserted */

  pp_enab = 106;
  kppc(0, &ErrorFlag, add_1, pp_enab);
  if (ErrorFlag) err_handler(ErrorFlag);
```

KPPD

Purpose Disables the Parallel Poll response capability of the specified GPIB bus device(s). The KM-488-DD must be the Active Controller.

Syntax

```
int board_nr, err_code;
struct gads mgads[n];
kppd(board_nr, &err_code, &mygads);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`mygads[n]` contains the GPIB bus address(es) of the device(es) to be disabled. `mygads` is a pointer to an array of `n` (`n` can range from 1 to 15) structures of type gads. You must tailor `mygads` to your application. `mygads` structure is defined in the include file *km488dd.h* and is described in section 11.1.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

KPPD (cont'd)

Example

```
/*  Address Group (will have one device @ 1) */
int ErrorFlag;
struct gads add_1[15];

/*  Set-up 1st instrument with GPIB primary address 1
    (HEX 1) and  no secondary address */

add_1[0].primad=0x1;
add_1[0].secad=-1;
add_1[1].primad=-1;  /*  no 2nd instrument   */

/*   Disable Parallel Poll */

  kppd(0,&ErrorFlag,add_1);
  if (ErrorFlag) err_handler(ErrorFlag);
```

KPPU

Purpose Disables the Parallel Poll Response of all GPIB bus device(s). The KM-488-DD must be the Active Controller.

Syntax

```
int board_nr, err_code;
kppu(board_nr, &err_code);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
int board_nr = 0, err_code = 0 ;
kppu(board_nr, &err_code) ;
if (err_code) err_handler(err_code) ;
```

KQUIKSTAT

Purpose Returns the KM-488-DD's status byte.

Syntax

```
int board_nr, err_code, qstat;
kquikstat(board_nr, &err_code, &qstat);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `qstat` is a pointer to the returned status integer. The status integer is of the following format:

Bit	15-9	8	7	6	5	4	3	2	1	0
	0	ADSC	CIC	PEND	DET	SRQ	DEC	LA	TA	DMA

Where:

ADSC	**Address Status Change.** If this bit is set to 1, a change in address status has occurred (i.e., a Talker becomes a Listener, or a Peripheral becomes an Active Controller).
CIC	**Controller in Charge.** If this bit is set to 1, the KM-488-DD is an Active Controller.
PEND	**SRQ Pending.** If this bit is set to 1, the KM-488-DD has an SRQ request pending.
DET	**Device Triggered.** If this bit is set to 1, a GPIB Trigger command has been received.
SRQ	**Internal SRQ.** If the KM-488-DD is an Active Controller and this bit is set to one, a device is requesting service. If the KM-488-DD is a Peripheral and this bit is set to one, then its SRQ (issued by KREQUEST) has not been serviced.
DEC	**Device Cleared.** If this bit is set to one, the KM-488-DD has received a GPIB Clear command.

KQUIKSTAT(cont'd)

LA	**Listen Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Listener and can accept data from the GPIB.
TA	**Talk Addressed.** If this bit is set to one, the KM-488-DD is addressed as a Talker and can output data to the GPIB.
DMA	DMA. A DMA transfer is currently in progress.

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes None.

Example

```
int board_nr = 0, err_code = 0, stat = 0;
kquikstat(board_nr, &err_code, &qstat) ;
if (err_code) myerrorhandler(err_code) ;
else
    printf("Quick Status Returned (Hex) : %x", stat) ;
```

KREMOTE

Purpose Forces the GPIB bus device(s) to the remote mode (ignore the bus).

☞ ***NOTE:*** The KM-488-DD must be a System Controller to execute this command.

Syntax

```
int board_nr, err_code;
struct gads mgads[n];
kremote(board_nr, &err_code, &mygads);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] contains the GPIB bus address(es) of the device(es) to be put in the remote mode. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. **mygads** structure is defined in the include file *km488dd.h* and is described in section 11.1.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

KREMOTE (cont')

Programming Notes

1. If **mygads** does not specify an address (i.e., **no_addr** is used), then the GPIB REN (Remote Enable) line is asserted. Devices will not be in the remote mode until addressed to listen. If **mygads** contains address then those devices will also be addressed to listen, so they will be put in remote.

2. It is good programming practice to issue a Local Lockout to prevent the device(s) from being returned to Local mode.

Example

```
/*  Address Group (will have one device @ 05) */

struct gads add_05[15];

/*  Address Structure For Device With GPIB Address 05 */

add_05[0].primad = 5 ;
add_05[0].secad = -1 ;
add_05[1].primad = -1 ; /* no 2nd instrument */

/*   Put Device In Remote State */

  kremote(0, &ErrorFlag, add_05);
  if (ErrorFlag) err_handler(ErrorFlag);
```

KREQUEST

Purpose Sets the Serial Poll Response of a KM-488-DD which is a Peripheral.

Syntax

```
int board_nr, err_code, sp;
krequest(board_nr, &err_code, sp);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

sp is an integer in the range 0 to 255 which represents the serial poll response of the device. This integer is of the following format:

Bit	7	6	5	4	3	2	1	0
	DIO8	rsv	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
rsv	If this bit is 1, the KM-488-DD will generate a Service Request (assert SRQ).

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The Active Controller can read the value of **sp** by serial polling the KM-488-DD. This will clear bit 6, if it was set.
2. Use KQUIKSTAT or SPOLL to check if the Peripheral has been serial polled (checks the status of the SRQ bit).

KREQUEST (cont'd)

Example

```
int ErrorFlag = 0, pend = 0 ;

/* Generate SRQ and Wait for Controller to Respond */

  krequest(0,&ErrorFlag,64);
  printf("Wait For Controller To Read SRQ\n\n");
  do
  {
    kspoll(0, &ErrorFlag, no_add, &pend);
    if (ErrorFlag) err_handler(ErrorFlag);
    printf("%i\n ", pend) ;
  } while (pend == 64);
```

KRESET

Purpose Performs a "warm" reset of the KM-488-DD and the GPIB bus.

Syntax

```
int board_nr, err_code;
kreset(board_nr, &err_code);
```

Parameters **`board_nr`** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **`err_code`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. The KM-488-DD is reset to the following conditions:

   - STOP
   - DISARM
   - CONFIG
   - ABORT
   - BUFFER INCREMENT
   - DMA SINGLE
   - ERROR ON
   - FILL NULL
   - LOCAL
   - REQUEST 0 (If Peripheral)
   - Clear CHANGE, TRIGGER, and CLEAR STATUS
   - TIMEOUT 10

   It also clears all error conditions.

2. If the KM-488-DD is the System Controller, it will assert the GPIB IFC (Interface Clear) line for at least 500 µsecs.

Example

```
int board_nr = 0, err_code = 0 ;
kreset(board_nr, &err_code) ;
if (err_code) myerrorhandler(err_code) ;
```

KRESUME

Purpose Initiates data transfers between two non-Active Controller GPIB devices, by unasserting the ATN line. The KM-488-DD must be an Active Controller in order to use this function.

Syntax

```
int board_nr, err_code;
kresume(board_nr, &err_code);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes This routine is normally called after a KSEND has addressed a talker and a listener. (See KSEND description.)

Example

```
    /*  Drop Attention Line If Active Controller  To
Start Data Transfer */

  int ErrorFlag = 0 ;
  printf("Drop Attention.\n") ;
  kresume(0,&ErrorFlag) ;
  if (ErrorFlag) err_handler(ErrorFlag);
```

KSEND

Purpose Sends GPIB commands and data from a string.

Syntax

```
int board_nr, err_code, stringlen;
char far sendstr;
ksend(board_nr, &err_code, &sendstr, stringlen);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

sendstr is a string which includes any of the following subcommands: CMD, UNT, MTA, MLA, TALK, LISTEN, DATA, and EOI. Multiple subcommands may be specified; however, the length of the entire SEND command string cannot exceed 255 characters. The subcommands are as follows:

CMD - Sends information with the ATN line asserted. This indicates to the bus devices that the characters are to be interpreted as GPIB bus commands. The command may be sent as either a quoted string (e.g., **'COMMAND'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **&HOD** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **CMD 67,79,77,77,65,78,68**).

An EOI cannot be sent with this subcommand, because an EOI with ATN asserted would initiate a Parallel Poll.

UNT - Untalks all devices by sending a GPIB UNTALK command with ATN asserted.

UNL - Unlistens all devices by sending the GPIB UNLISTEN command with ATN asserted.

MTA - Designates the KM-488-DD to address itself to talk by sending the GPIB MTA (My Talk Address) command with the ATN line asserted.

KSEND (cont'd)

MLA - Designates the KM-488-DD to address itself to listen by sending the GPIB MLA (My Listen Address) command with the ATN line asserted.

TALK addr - Addresses another GPIB device or KM-488-DD to talk by sending a GPIB TAG (Talk Address Group) command with the ATN line asserted. **addr** is an integer representing the GPIB BUS device address of the device to talk. This integer ranges from 00 to 30 decimal.

LISTEN addr - Addresses another GPIB device(s) or KM-488-DD('s) to listen by sending a GPIB LAG (Listen Address Group) command with ATN asserted. **addr** is an integer representing the GPIB BUS device address of the device(s) to talk. This integer ranges from 00 to 30 decimal. Multiple listeners can be specified. If **addr** is not specified, all other devices on the GPIB BUS will be designated listeners.

DATA - Sends information with the ATN line unasserted. This indicates to the bus devices that the characters are to be interpreted as data. This is the technique which is to be used to send device-dependent commands. (See the IEEE-488 Tutorial in Appendix C for more information.) The data may be sent as either a quoted string (i.e., **'DATA'**) or as a numeric equivalent of an individual ASCII character (i.e., **30** decimal or **&H0D** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **DATA 68,65,84,65**).

This sub-command is useful when you are sending commands which are unique to your interface.

KSEND (cont'd)

EOI - Sends information with the ATN line unasserted. EOI will be asserted when the last character is transmitted. This information is interpreted as GPIB bus data and may be sent as either a quoted string (e.g., **'xyz'**) or as a numeric equivalent of an individual ASCII character (e.g., **13** decimal or **&H0D** hex for a Carriage Return). Hex values must be preceded by &H. Multiple quoted strings or ASCII values bytes may be sent if they are separated by commas (e.g., **SEND EOI 120,121,122**).

stringlen is an integer between 0 and 255 which represents the length of the SEND string.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Typically, KSEND is used to send commands to initiate communication directly between peripheral devices. For example, the KM-488-DD driver might talk a scope and send a trace to a listened plotter.

2. The maximum length of the KSEND call, including any subcommands, is 255 characters. To KSEND large amounts of data, use multiple KSEND calls.

3. KSEND should only be used when a non-conforming device requires a special command sequence or a non-standard GPIB command. Do not use the KSEND command unless you are extremely familiar with GPIB.

4. The KM-488-DD must be the Active Controller to send commands. Any KM-488-DD can send data.

5. If a DATA subcommand is not included in the KSEND string, be sure to call KRESUME immediately after the KSEND. This is necessary because the ATN line must be dropped so that the transfer will proceed.

6. Do not include the word SEND within the KSEND string, as you might do in the File I/O SEND command.

KSEND (cont'd)

Example

```
 int ErrorFlag;
/* String To Listen Peripheral Device 02 And Talk*/
/* Peripheral Device 12 */
  char talk_listen[] = { "UNT UNL LISTEN 02 TALK 12" } ;
  ksend(0, &ErrorFlag, talk_listen, strlen(talk_listen));
  if (ErrorFlag) err_handler(ErrorFlag);

  kresume(0,&ErrorFlag);
  if (ErrorFlag) err_handler(ErrorFlag);
```

KSPOLL

Purpose

If the KM-488-DD is an Active Controller, KSPOLL will check for an SRQ or conduct a serial poll. If the KM-488-DD is a Peripheral, KSPOLL will check if the KM-488-DD's SRQ has been serviced.

Syntax

```
int board_nr, err_code;
unsigned int spr[n];
struct gads mgads[n];
kspoll(board_nr, &err_code, &spr, &mygads);
```

Parameters

board_nr is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads[n] contains the GPIB bus address(es) of the device(es) to be disabled. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. **mygads** structure is defined in the include file *km488dd.h* and is described in section 11.1.

Returns

err_code is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

spr is an unsigned integer or an array of unsigned integers with an entry for each address in **mygads**. The value of the integer is 0 to 255 and corresponds to a byte with the following bitmap:

Bit	7	6	5	4	3	2	1	0
	DIO8	SRQ	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

KSPOLL(cont'd)

Where:

DIO1-8	Bits 1 through 8 of this device's Serial Poll Response Byte (correspond to data lines DIO1-DIO8).
SRQ	If this bit is 1, the SRQ (Service Request) line will be asserted to request servicing. Otherwise, SRQ will not be asserted.

Bit 6 has the special significance of indicating an SRQ pending. The significance of the other bits will be application dependent.

Programming Notes

1. The most common used of KSPOLL is for the KM-488-DD, as an active controller, to issue KSPOLL with **mygads** containing the address of a single device. This addresses and serial polls the device and, upon return, **spr** will contain the serial poll response of the polled device. If the SRQ bit in **spr** is set, the device had issued an SRQ. The other bits in **spr** will be device-dependent. Serial polling a device which is currently asserting SRQ will cause the device to unassert SRQ.

   The string returned by KSTATUS or the integer status word set by KQUIKSTAT can be checked to determine the presence of an SRQ before a serial poll is conducted. If only one device is asserting SRQ, the effect of issuing KSPOLL will be to clear the internal SRQ pending "bit". This is reflected in the returned by KSTATUS or KQUIKSTAT. This is true even if the polled device is not issuing the SRQ. It is also the case with multiple devices asserting SRQ when a device not issuing an SRQ is polled. In this case, a subsequent use of KSTATUS or KQUIKSTAT would not reveal a pending SRQ.

   To aid in identifying which bus device(s) is currently requesting service, a KM-488-DD which is the Active Controller can serial poll as many as 14 devices by issuing a KSPOLL call with **mygads** containing the address of more than one device. In this case, **spr** should be an array of unsigned integers with an element for each address. In this case, KSPOLL will serial poll each addressed device and return the serial poll bytes in the **spr** array. This is a faster way for discerning the source of an SRQ among several devices.

KSPOLL (cont'd)

2. If the KM-488-DD is a peripheral and KSPOLL is called with **mygads** equal to the no address structure (**no_addr**), **spr** will be the serial poll response (**sp**) set by KREQUEST - with the possible exception of the SRQ bit, which may have disappeared as a result of an Active Controller having serial polled the KM-4888-DD since the last KREQUEST call. In the instance of an Active Controller serial polling a peripheral KM-488-DD, the peripheral KM-488-DD unasserts the SRQ line. For an example, see KREQUEST. The state of the SRQ bit can also be determined from the string returned by KSTATUS or the integer status word set by a KQUIKSTAT call.

3. If the KM-488-DD is the Active Controller and issues KSPOLL with **mygads** equal to **no_addr**, **spr** will be 64 if a device is asserting SRQ and 0 if not. This same result could have been determined from KQUIKSTAT or KSTATUS.

   Note that if this call is issued when a SRQ is pending, it internally "clears" the SRQ "pending" bit, even though the requesting device has not been polled and is still issuing an SRQ. Thus, a second KSPOLL or subsequent KQUIKSTAT or KSTATUS calls would not reveal an SRQ pending.

Example

```
/* Address Group (will have one device @ 05) */

  struct gads add_05[15];
  int DevReady = 0x10 ;      /* Device Specific Value*/
  int sbyt ;       /* integer to hold byte returned by
"spoll" */

/*   Address Structure For Device With GPIB Address 05 */
  add_05[0].primad = 05 ;
  add_05[0].secad = -1 ;
  add_05[1].primad = -1 ;  /* no 2nd instrument */
  do
  {
   kspoll(0,&ErrorFlag,add_05,&sbyt);
   if (ErrorFlag) err_handler(ErrorFlag);
  } while (!(sbyt & DevReady)) ;
```

KSTATUS

Purpose Returns a character string describing the current operating state of the KM-488-DD.

Syntax

```
int board_nr, err_code;
char stat[70];
kstatus(board_nr, &err_code, stat);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

`stat` is a character string which describes the current operating state of the KM-488-DD. The string is of the format shown in Table 11-2. `stat` should be 75 bytes in order to contain the longest message that might be returned. If it is too short, data will be written over other portions of the program causing problems.

Table 11-2. Status String

Starting Col	# of Cols.	Name and Description

1	1	Operating Mode - Tells if the KM-488-DD is acting as an *Active Controller* or *Peripheral*. Can be **C** or **P** where:
		
		C = The KM-488-DD is an *Active Controller*
		P = The KM-488-DD is a *Peripheral*

3	2	Bus Address - Gives the IEEE-488 Bus Address of the KM-488-DD. This is a two-digit decimal integer ranging from 00 to 30.
6	2	Addressed Status Change - Indicates if the device has changed addressed state, i.e., if it has cycled between the *Talker*, *Listener*, or *Active Controller* states. This is reset whenever a STATUS is read. This can be one of the following: G0 = There has not been a change in the addressed status. G1 = There has been a change in the addressed status.

9	1	Current Addressed State - Indicates if the KM-488-DD is currently acting as a *Talker* or *Listener*, or is *Idle*. Can be **T**, **L**, or **I**, where:
		
		T = The KM-488-DD is a Talker
		L = The KM-488-DD is a Listener
		I = The KM-488-DD is Idle

11	2	Service Request - Represents the current internal SRQ status where: S0 = SRQ not asserted S1 = SRQ asserted If the KM-488-DD is in the *Active Controller* mode, the internal SRQ state indicates if a device is asserting SRQ. If the KM-488-DD is acting as a *Peripheral*, the internal SRQ state is set by using the KREQUEST call. It is cleared by a serial poll from the Active Controller and indicates it is asserting SRQ.

KSTATUS(cont'd)

Table 11-2. Status String

Starting Col	# of Cols.	Name and Description
14	3	Error Code - Indicates whether or not an error has occurred. The three characters consist of the letter 'E' followed by a two digit error code. (A list of these error codes is provided in Appendix G.) An error code of 00 indicates no error has occurred.
18	2	Triggered State - Indicates if a IEEE-488 trigger command has been received or not. This is not updated during DMA CONTINUE transfers. Triggered State values are: T0 = No trigger command has been received. T1 = A trigger command has been received.
21	2	Cleared - Indicates if the IEEE Clear command has been received or not. This is not updated during DMA CONTINUE transfers. Cleared values are: C0 = No Clear command has been received. C1 = A Clear command has been received.

24	2	Transfer Status - Indicates if a data transfer is in progress. Values which may appear are:
		
		P0 = No transfer
		P1 = DMA CONTINUE Transfer occurring
		P2 = DMA AUTOINITIALIZE Transfer occurring

27	up to 45	Error Message Text - Contains the Error Message Text associated with the given error code (EXX). These error messages are listed in Appendix G.

Programming Notes If the KM-488-DD is the Active Controller, the Service Request Status may be cleared (i.e, S0) as a result of a KSPOLL, even when the SRQ line is still asserted. See the discussion in KSPOLL.

KSTATUS (cont'd)

Example

```
void err_handler(int err_num)
{

/* For Status String Returned By "STATUS" Command */
char st[75];

 /* integer to hold error code returned by each
CALL (0 = false = no error) */

int ErrorFlag;

 printf("\nError #%d -- Consult List In
KM488DD.H",err_num);
 kstatus(0, &ErrorFlag, st);
 printf("\n%s\n",st + 26);  /* Print Error Message */
 exit(err_num) ;
}
```

KSTOP

Purpose Stops a DMA CONTINUE transfer.

Syntax

```
int board_nr, err_code;
kstop (board_nr, &err_code);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. If the DMA CONTINUE transfer has been completed, a KSTOP will have no effect.
2. Use the KBUFD call to determine the actual number of characters which were transferred before the KSTOP took effect.

Example

```
int board_nr = 0, err_code = 0 ;
kstop(board_nr, &err_code) ;
if (err_code) myerrorhandler(err_code) ;
else printf("DMA Halted") ;
```

KTERM

Purpose Changes the default GPIB bus terminator(s) to be used in ENTER and/or OUTPUT calls.

Syntax

```
int board_nr, err_code;
unsigned int trm_list[4];
kterm(board_nr, &err_code, &trm_list);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`term_list` is an integer array which defines the GPIB bus terminators. See Chapter 6 for more information regarding the use of GPIB bus terminators. (`termlist[4]` is defined in the *km488dd.h* header file.) If you wish to modify the `termlist[4]` array, it is comprised of the following elements:

`termlist[0]`: **Input/Output GPIB Terminators.** Possible values are:

-1	Change the Input Terminators.
0	Change both Input and Output Terminators.
1	Change the Output Terminators.

`termlist[1]`: **EOI detection/assertion.** Possible values are:

0	No EOI assert/detect.
1	Assert/Detect EOI.

KTERM (cont'd)

`termlist[2]`: **1st GPIB bus terminator.** Possible values are:

-1(0xFFFF)	Don't use first or second GPIB bus terminator.
0 to 255	Use this ASCII character as the first GPIB bus terminator instead of the default first GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

`termlist[3]`: **2nd GPIB bus terminator.** Possible values are:

-1(0xFFFF)	Don't use second GPIB bus terminator.
0 to 255	Use this ASCII character as the second GPIB bus terminator instead of the default second GPIB Terminator. The 0 - 255 decimal value represents the ASCII value of the terminator character to be used. (An ASCII Equivalence Chart is provided in Appendix A.)

Returns **`err_code`** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. KTERM will change the default bus terminators set during software installation, as directed by **`termlist`**.

KTERM (cont'd)

Example

```
int ErrorFlag, trm_lst[4];

    /* Change Configuration Termination to EOI Without
    Termination Characters */

    trm_lst[0]=0;      /* change input and output termi-
    nators */
    trm_lst[1]=1;      /* 1 indicates assert EOI */
    trm_lst[2]=-1;     /* -1 indicates no 1st terminator
    */
    trm_lst[3]=-1;     /* -1 indicates no 2nd terminator
    */

    /* Change Input and Output GPIB Message Termination
    to EOI Only */
      kterm(0,&ErrorFlag,trm_lst);
      if (ErrorFlag) err_handler(ErrorFlag);
```

KTO

Purpose Changes the timeout period.

Syntax

```
int board_nr, err_code;
int long tval;
kto(board_nr, &err_code,tval);
```

Parameters `board_nr` is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

`tval` is the number of 53 ms. timer ticks to allow before a time-out.

Returns `err_code` is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. To suppress Timeout checking, set `tval` to 0.

2. If a DMA CONTINUE transfer is in progress, the KM-488-DD will check for timeouts only for the first byte that is transmitted/received. During other types of transfers, the KM-488-DD will check for a timeout between transmission of bytes.

You must be certain to check that a DMA CONTINUE transfer has been completed.

Example

```
unsigned int ErrorFlag = 0 ;

/*  Set Time Out To 15 Seconds (Approx 18.2 Tick Counts
*/
/*  Per Second) */

  kto(0, &ErrorFlag, (unsigned long) 0x00000111);
  if (ErrorFlag) err_handler(ErrorFlag);
```

KTRIGGER

Purpose Triggers the specified device(s). The KM-488-DD must be an Active Controller.

Syntax

```
int board_nr, err_code;
struct gads mgads[n];
ktrigger(board_nr, &err_code, &mygads);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

mygads [n] contains the GPIB bus address(es) of the device(es) to be triggered. **mygads** is a pointer to an array of **n** (**n** can range from 1 to 15) structures of type gads. You must tailor **mygads** to your application. **mygads** structure is defined in the include file *km488dd.h* and is described in section 11.1.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes The devices listed in **mygads** will be triggered.

KTRIGGER (cont'd)

Example

```
/*  Address Group (will have one device @ 12) */

struct gads add_12[15];

/*  Address Structure For Meter With GPIB Address 12 */

add_12[0].primad = 12 ;
add_12[0].secad = -1 ;
add_12[1].primad = -1 ;     /* no 2nd instrument */

/* Trigger DMM To Perform A Measurement */
  ktrigger(0, &ErrorFlag, add_12);
  if (ErrorFlag) err_handler(ErrorFlag);
```

KWAITC

Purpose Forces the KM-488-DD driver to wait until a DMA CONTINUE transfer has been completed before returning control to the application program.

Syntax
```
int board_nr, err_code;
kwaitc(board_nr, &err_code);
```

Parameters **board_nr** is an integer which represents the board identification number. This is an integer in the range 0 to 3 and represents the board to be programmed.

Returns **err_code** is a variable through which error codes are returned. If this variable is set to 0, then no error occurred. If it is set to a value other than 0, then an error did occur. The error codes are listed in Appendix G.

Programming Notes

1. Time out checking, if enabled, is performed while KWAITCing.

2. This is an explicit wait. The following commands perform an implicit wait:

   - KABORT
   - KCLEAR
   - KENTER
   - KLOCAL
   - KLOCAL LOCKOUT
   - KOUTPUT
   - KPASS CONTROL
   - KPPOLL
   - KPPOLL CONFIG
   - KPPOLL DISABLE
   - KPPOLL UNCONFIG
   - KREMOTE
   - KREQUEST
   - KRESUME
   - KSEND
   - KSPOLL
   - KTRIGGER

Example
```
int board_nr = 0, err_code = 0 ;

kwaitc(board_nr, &err_code) ;
if (err_code) myerrorhandler(err_code) ;
else printf("DMA Continue Completed") ;
```

This page intentionally left blank.

CHAPTER 12
WARRANTY INFORMATION

12.1 WARRANTY

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

CAUTION

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

12.2 RETURN TO FACTORY INFORMATION

Before returning any equipment to the factory for repair, you must first call the Technical Support Department at (508) 880-3000. They will try to diagnose and solve your problem over the phone. If they ascertain that the unit has to be returned to the factory for repair, they will issue a Return Material Authorization (RMA) number. Note that if the board is to be repaired under warranty, the Technical Support Department will need your invoice number and the date the board was purchased. Please reference the RMA number on any correspondence regarding the board.

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

1. A brief description of the problem.
2. Your name, address, and telephone number.
3. The invoice number and the date when the board was purchased.
4. Repackage the board in its original anti-static wrapping (Handle it with ground protection.) and ship it back to:

Repair Department
Keithley MetraByte Corporation
440 Myles Standish Boulevard
Taunton, Massachusetts 02780

Telephone:(508)880-3000
Telex:503989
FAX:(508)880-0179

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

ASCII CODE EQUIVALENCE CHART

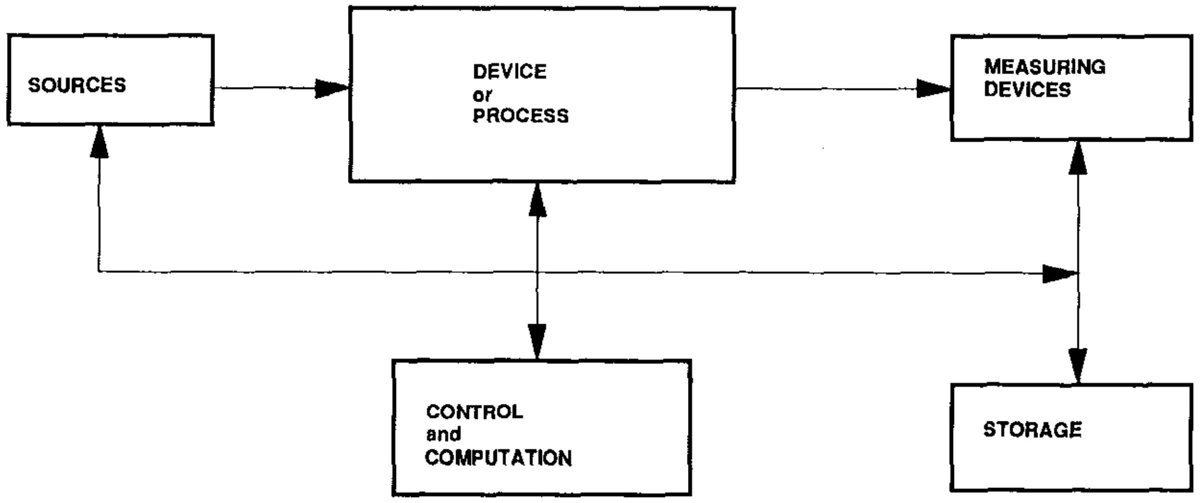
ASCII CHARACTER	HEX	DEC	ASCII CHARACTER	HEX	DEC
NUL	00	0	! (Exclamation Point)	21	33
SOH (Start of Heading)	01	1	" (Quote Mark)	22	34
STX (Start of Transmission)	02	2	# (Pound Sign)	23	35
ETX (End of Transmission)	03	3	$ (Dollar Sign)	24	36
EOT (End of Text)	04	4	% (Per Cent Sign)	25	37
ENQ (Enquiry)	05	5	& (Ampersand)	26	38
ACK (Acknowledge)	06	6	' (Apostrophe)	27	39
BEL (Bell)	07	7	((Left Parenthesis)	28	40
BACKSPACE	08	8	) (Right Parenthesis)	29	41
HT (Horizontal Tab)	09	9	* (Asterisk)	2A	42
LF (Line Feed)	0A	10	+ (Plus Sign)	2B	43
VT (Vertical Tab)	0B	11	, (Comma)	2C	44
FF (Form Feed)	0C	12	- (Minus Sign)	2D	45
CR (Carriage Return)	0D	13	. (Period)	2E	46
SO (Shift Out)	0E	14	/ (Slash)	2F	47
SI (Shift In)	0F	15	0	30	48
DLE (Data Link Escape)	10	16	1	31	49
DC1 (Data Control 1)	11	17	2	32	50
DC2 (Data Control 2)	12	18	3	33	51
DC3 (Data Control 3)	13	19	4	34	52
DC4 (Data Control 4)	14	20	5	35	53
NAK (Not Acknowledge)	15	21	6	36	54
SYN (Synchronous Idle)	16	22	7	37	55
ETB (End of Trans. Blank)	17	23	8	38	56
CAN (Cancel)	18	24	9	39	57
EM (End of Medium)	19	25	: (Colon)	3A	58
SUB (Substitute)	1A	26	; (Semi-Colon)	3B	59
ESC (Escape)	1B	27	< (Less than)	3C	60
FS (File Separator)	1C	28	= (Equal)	3D	61
GS (Group Separator)	1D	29	> (Greater than)	3E	62
RS (Record Separator)	1E	30	? (Question Mark)	3F	63
US (Unit Separator)	1F	31	@ (At, per sign)	40	64
SP (Space)	20	32	A	41	65

ASCII CHARACTER	HEX	DEC	ASCII CHARACTER	HEX	DEC
B	42	66	c	63	99
C	43	67	d	64	100
D	44	68	e	65	101
E	45	69	f	66	102
F	46	70	g	67	103
G	47	71	h	68	104
H	48	72	i	69	105
I	49	73	j	6A	106
J	4A	74	k	6B	107
K	4B	75	l	6C	108
L	4C	76	m	6D	109
M	4D	77	n	6E	110
N	4E	78	o	6F	111
O	4F	79	p	70	112
P	50	80	q	71	113
Q	51	81	r	72	114
R	52	82	s	73	115
S	53	83	t	74	116
T	54	84	u	75	117
U	55	85	v	76	118
V	56	86	w	77	119
W	57	87	x	78	120
X	58	88	y	79	121
Y	59	89	z	7A	122
Z	5A	90	{ (Left Brace)	7B	123
[(Left Bracket)	5B	91	| (Vertical Slash)	7C	124
\ (Backslash)	5C	92	} (Right Brace)	7D	125
] (Right Bracket)	5D	93	~ (Tilde)	7E	126
^ (Caret)	5E	94	DEL (Delete)	7F	127
_ (Underline)	5F	95			
' (Accent, Grave)	60	96			
a	61	97			
b	62	98			

IEEE-488 TUTORIAL

The evolution of electronics over the past few decades has lead to concepts and implementations of test/measurement and control systems of continually increasing complexity and sophistication. For example, measurement started out as "go no go" tests equivalent to plugging a lamp into an electrical outlet to determine if the outlet is "hot". Next, meters appeared which yielded a single number characterizing a quantity and then oscilloscopes which displayed how signals varied with time. Today, logic and spectrum analyzers allow us to further manipulate and display the data in a variety of specialized ways.

At the same time, our expectations on collecting, saving and manipulating the results of measurements has escalated from writing down meter readings and hand calculations to automated storage of and complicated computations on large numbers of measurements. Many instruments have these capabilities "built-in"; thus freeing the system controller from having to handle complex calculations. A modern test/measurement or control system can be represented as:



A typical test would be to measure the "frequency response" of a device. The source would be capable of supplying a sine wave of varying frequency to the input of the device and the measuring device would measure the magnitude and phase of the output. In an automated system, the CONTROL box would step the source through a range of frequencies. At each frequency the control would request the measuring device(s) to return a value and the results could be stored and used to calculate the "transient" response of the device, for example.

Traditional test instruments have provided the basic measurement functions for years. For example, there are oscillators which generate sine waves of various frequencies and meters to measure responses. The essence of today's system is that the different functional units of the system can communicate with each other as required and be run automatically by a controller. To accomplish this goal, a bus has been defined which allows instruments to be interconnected and to communicate with each other through a standard hardware arrangement. This bus is often referred to as the GENERAL PURPOSE INTERFACE BUS (GPIB). It is also identified as the IEEE-488 bus because it has been standardized in specifications from the Institute for Electrical and Electronic Engineers.

B.1 TOPOLOGY

An IEEE-488 system allows different manufacturers' devices to be connected. Systems can be connected following a star or linear-type topology or using a combination of both. The system should adhere to the following constraints:

- No more than 15 devices can be connected by a single bus.
- The total transmission length cannot exceed 20 meters or 2 meters times the number of devices (which ever is less).
- The data rate through any signal line must be less than or equal to one megabyte per second.

B.2 THE SYSTEM

The simplest IEEE-488 system consists of a single device sending data to another, such as a meter outputting data to a printer. A more typical IEEE-488 bus system (See Figure B-1.) is comprised of up to 15 devices, each of which acts as one or more of the following: Controller, Listener, and Talker.

There are a variety of interface functions which GPIB devices can support at various levels. The IEEE standard recommends that a label listing the device codes be placed on the instrument near the IEEE connector. Codes consisting of 1 or 2 letters indicating the function type followed by a number indicating the level of support are used to characterize the device. If the number is 0, it means that the function is not supported. Each device's applicable device codes should be listed within its manual or specification. Appendix D lists the device codes.

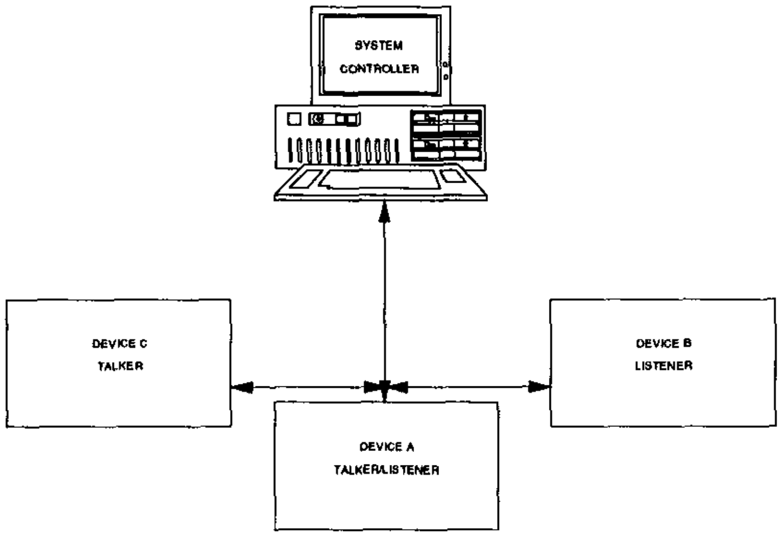


Figure B-1. Typical System

Listeners

A Listener is any device which is able to receive data when properly addressed. There can be up to 14 *active* listeners on the bus concurrently. Some devices can also be a talker or controller; however, only one of these functions can be performed at a time.

Talkers

A Talker is any device which can transmit data over the bus when properly addressed. Only one device can transmit at a time. Some devices can also be a listener or controller; however, a device can perform only one of these functions at a time.

Controllers

Most IEEE-488 systems contain at least one IEEE-488 Controller (e.g., the KM-488 board). There may be more than one Controller per system, but only one can be *active* at any given time. This function is very important because the Active Controller has the ability to mediate all communications which occur over the bus. In other words, the Active Controller designates (addresses) which device is to talk and which devices are to listen. The Active Controller is also capable of relinquishing its position as Active Controller and designating another Controller to become the Active Controller.

There is always one *System Controller* in an IEEE-488 system. The *System Controller* is defined at system initialization either through the use of hardware switches or by some type of configuration software, and usually would not be changed. This *System Controller* can be the same controller as the one which is the current *Active Controller* or an entirely different one. If the controller is both a System Controller and an Active Controller and it passes control to another controller, the system controller capability is not passed along with it.

The *System Controller* has the unique ability to retrieve active control of the bus or to enable devices to be remotely programmed. It takes control of the bus by issuing an IFC (Interface Clear) message. The System Controller issues this message by asserting the IFC Control line (See section B.3.) for a period of at least 200 µsecs.

Likewise, devices cannot be put into the remote state (can be programmed from the GPIB bus rather than from the normal controls) unless the *System Controller* is asserting the REN (Remote Enable) line. (See section B.3.) With REN asserted, a device will go into the remote state the first time it is addressed to listen by any *Active Controller.* All the devices will return to local control if the *System Controller* unasserts REN.

If an IEEE-488 device is not a System Controller or an Active Controller, then it will be referred to as a *device.* In this capacity, it can be idle, act as a talker and/or listener, when it has been addressed or unaddressed by the *Active Controller.*

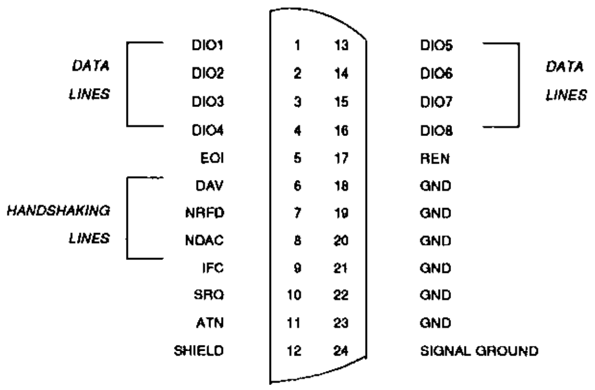


Figure B-2. IEEE-488 Bus Connector

B.3 BUS LINES

The IEEE-488 bus is a parallel bus containing 24 lines, 16 of which are signal lines. (See Figure B-2.) These 16 lines consist of eight data lines, five control lines, and three handshake lines. The manner in which the bus lines are used is described in the section B.5 .

Data Lines

The Data Lines (DIO1 through DIO8) are used to convey messages such as: device addresses, Parallel Poll Responses, IEEE-488 *Interface Commands,* or *Data/Device Dependent Commands.* They are discussed extensively in section B.4.

Control Lines

The control Lines perform a variety of control, request, and coordinating functions which assure the orderly flow of information on the bus. The IEEE standard refers to any bus activity as being a "message". Messages used to control bus functions, as opposed to sending data between devices, are called interface messages. Asserting a control line is said to send a uniline interface message because a specific effect usually occurs as the result of the assertion. Table B-1 briefly describes the control lines and lists their name, associated acronyms and functions. Their functions will be elaborated in subsequent sections.

Table B-1. Control Lines

Acronym	Line Name	Function
ATN	Attention	This line can only be asserted/unasserted by the *Active Controller.* It designates whether the current data on the data lines is data or a command. When this line is set low(true), it indicates that the information to follow represents commands and/or addresses. When this line is set high (false), the active talker is transmitting device-dependent data to all active listeners. This line is also used with EOI to conduct a parallel poll.
EOI	End or Identify	Signals that the last data byte of a multibyte sequence is being transferred. This line is also used in conjunction with the ATN line to initiate parallel polling.
IFC	Interface Clear	When this line is asserted (set low), the bus is cleared and all talkers/listeners are placed in an idle state. This is a pulse of 200 µ or more. This line can only be asserted by the *System Controller.*
REN	Remote Enable	If this line is asserted, bus devices can be programmed via IEEE bus commands issued from an *active talker.* This line can only be asserted by the *System Controller.*
SRQ	Service Request	This line when asserted indicates that service is required from the *Active Controller.* SRQ can be asserted by any bus device which supports the function.

Handshake Lines

There are three Handshake Lines which are used to coordinate data transfers between talkers and listeners on the bus. Table B-2 briefly describes the Handshake lines. It lists their names, associated acronyms, and functions.

Table B-2. Handshake Lines

Acronym	Line Name	Function
DAV	Data Valid	This signal is used to inform the system that valid data is ready for transmission.
NDAC	Not Data Accepted	Indicates if all devices accepted the data or not. As each listener receives data, it will set its NDAC line high. Once all intended listeners have accepted the data, the NDAC line to the talker will be set high.
NRFD	Not Ready For Data	Indicates whether or not the listeners are ready to receive data. When each listener is ready, it sets its NRFD line high.

Section B.4 describes the use of the handshaking lines. Figures B-3 and B-4 illustrate the Handshaking Sequence.

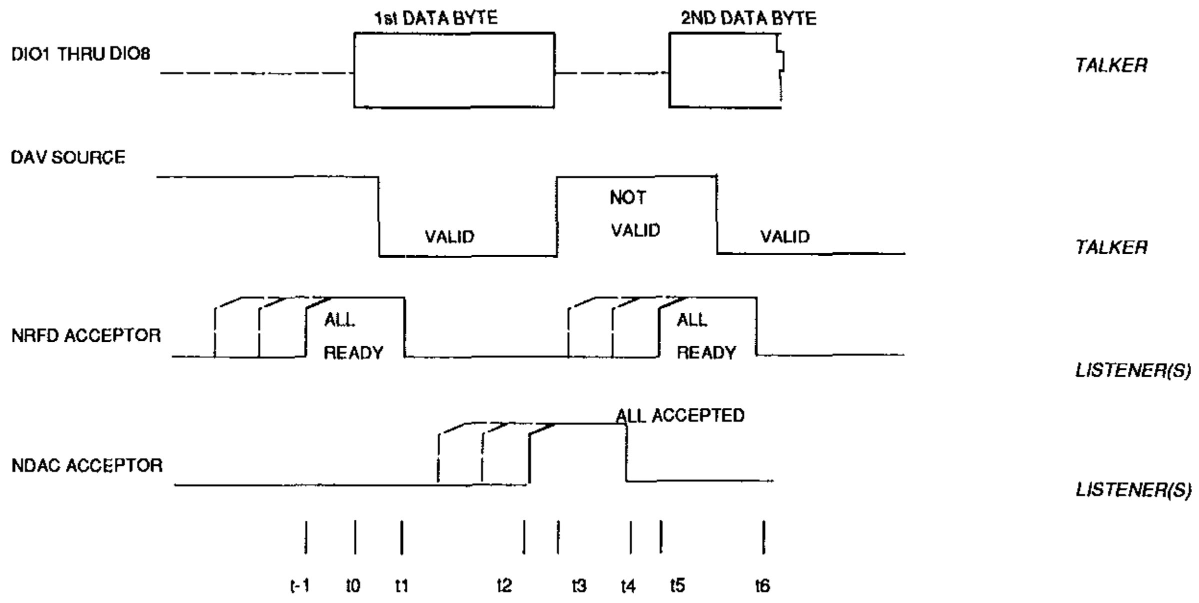


Figure B-3. Handshake Timing

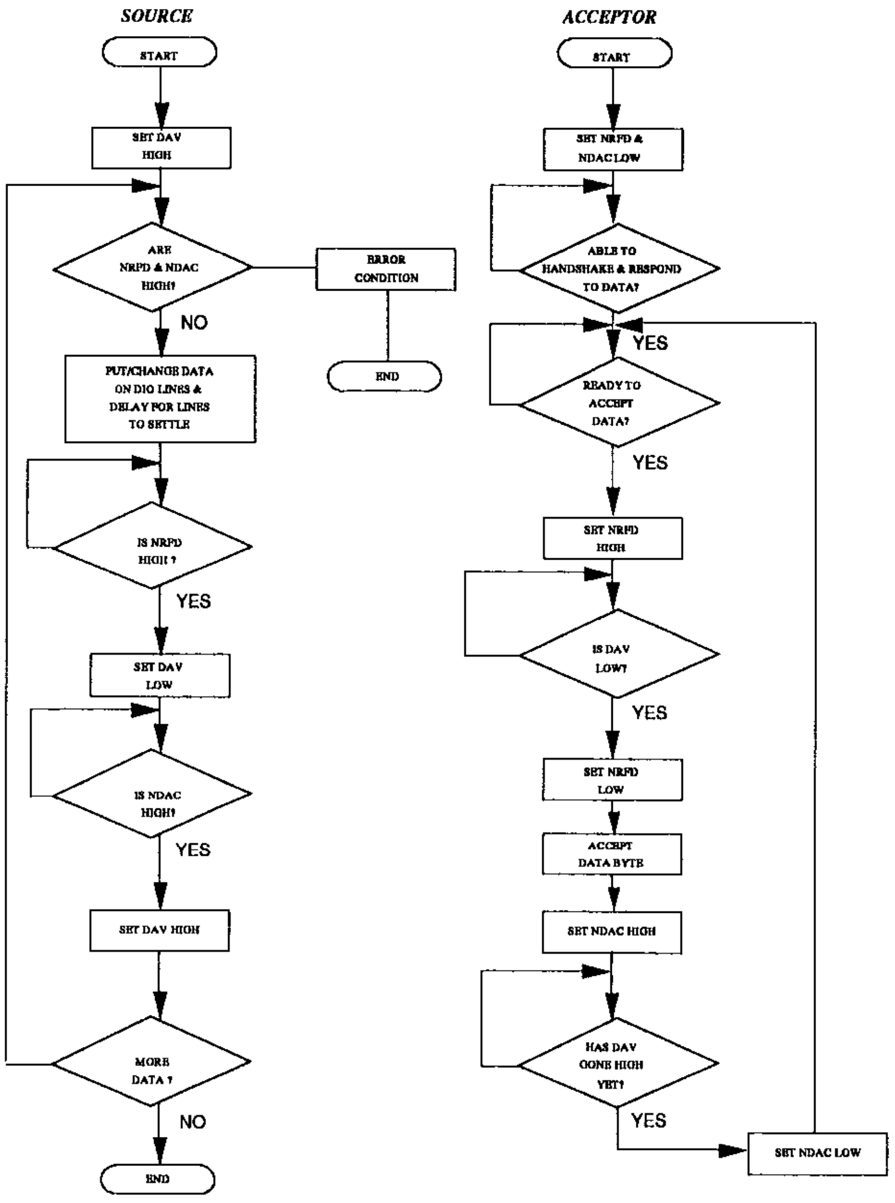


Figure B-4. Transmission of Data from Talker to Listener

Preliminary: Source checks for listeners and places data byte on data lines.

t_{-1}: All acceptors become ready for byte. NRFD goes high with slowest one.

t_0: Source validates data (DAV low)

t_1: First acceptor sets NRFD low to indicate it is no longer ready for a new byte.

t_2: NDAC goes high with slowest acceptor to indicate all have accepted the data.

t_3: DAV goes high to indicate this data byte is no longer valid.

t_4: First acceptor sets NDAC low in preparation for next cycle.

t_5: Back to t_{-1} again.

t_6: Back to t_0 again.

All devices that are to be "sources" i.e., be talkers and send data on the GPIB must be able to perform the "source" handshake functions of responding to the NRFD and NDAC lines and controlling the DIO lines and DAV as described above. In terms of the codes of Appendix D, they must have SH1 capability. Devices listed as SH0 cannot act as sources.

Likewise, all devices which are to be "acceptors", i.e. be listeners and receive data on the GPIB must be able to perform the "acceptor" handshake of responding to the DIO lines and DAV and controlling NDAC and NRFD as described above. They must have the AH1 capability as defined in Appendix D. Devices listed as AH0 cannot act as acceptors.

B.4 BUS FUNCTIONS

The purpose of the IEEE-488 Bus is to provide a mechanism for the orderly flow of information between bus devices. To accomplish this, the IEEE-488.1 specification refers to two types of messages as occurring on the bus. This first is interface messages which manage the interface itself and the second are device dependent messages which are used to transfer information between bus devices.

Interface messages are summarized in Appendix C and can be placed in two groups. The first group consists of the so-called "Uniline Interface Messages" introduced in B.3 which are sent by the controller asserting the special control lines. The second group, the so-called "multiline interface messages", which are treated separately in section B.5. The *Active Controller* sends multiline interface messages by asserting the ATN line and placing data on the DIO lines. The multiline interface messages are broken up into 5 groups: Addressed, Universal, Listen Address Group, Talk Address Group, and Secondary Address Group.

The second type of message is the device-dependent message and is sent by the *Active Talker* by placing data on the DIO lines (the ATN line will not be asserted). Device-Dependent messages are not discussed in this section.

The major functions performed by these messages are: System Initialization and Control, Device Addressing, Sending and Receiving Data/device Commands, Requesting Service, Polling and Triggering. These functions are described within this section.

System Initialization

When a typical IEEE-488 system is initialized, there will be one device which will be the *System Controller*. The *System Controller* will usually assert the Interface Clear line (See section B.3.) for at least 200 µsecs. to make sure it has control of the IEEE-488 bus and that no device is addressed to be an active talker or active listener. The *System Controller* will then unassert IFC.

Typically the system controller will assert the Remote Enable line (REN , See Section B.3) so that bus devices will go to remote when they are addressed to listen. When a device is in remote it can receive instructions remotely over the GPIB bus which will program its functions and ranges rather than locally from panel controls on the device. The controller might also issue a Local Lockout message (LLO, see Section B.5) which prevents an operator from returning a device to local control. In this way, the devices are completely under bus control.

All Devices can be put back into local by the *System Controller* unasserting REN or by any active controller issuing a Go To Local (GTL See section B.5) message to specific devices. In the latter case, devices will go back to remote the next time they are addressed to listen. The remote/local capability of a device is specified by the RL code of Appendix D.

The *Active Controller* can also issue device clear commands which will return the device(s) to its initial power-up programming state, for example, its original range and function. In some cases this means returning to factory-set default values while in others it means returning to previously saved operator-chosen settings. The functionality of a device is specified by its DC functionality of Appendix D.

Passing Control

Control can be passed to another controller by addressing a prospective controller to listen and then issuing a Take Control (TCT, See Section B.5.) message. Care must be taken that the prospective controller is capable of accepting control because generally no error will be detected if it is not. Having issued the message the previous controller becomes an inactive controller and a normal bus device. A system controller can always seize control by asserting IFC.

The function codes of Appendix D which describe controller function start with C. Multiple numbers are used. C0 indicates no controller capability, whereas C1-C5 would indicate complete capability.

Addressing a Device

Devices are addressed by the *Active Controller* issuing multiline interface messages from either the talk address group (TAG) or listen address group (LAG) as described in section B.5. Normally, up to 15 IEEE bus devices can be configured within one IEEE-488 system. In order to avoid data conflicts, each device is assigned a unique primary address in the range 0 to 30. Some devices can support more than one address although usually the device will present only one electrical load to the bus.

Because there can only be one talker at a time, a talker will be unaddressed automatically when another device is addressed to talk. However, there will be times when the controller will want to untalk a device without addressing another. It will always be necessary to unaddress listeners that no longer should be listening because it is possible to have any number of devices listening at the same time. Within each of the LAG and TAG groups is either an unlisten or an untalk command. The talk and listen function codes of talkers and listeners as listed in Appendix D begin with T and L respectively.

Secondary addresses are used to extend the total number of addresses on the bus. (Secondary addresses also must fall within the range 0 to 31.) Devices which employ a secondary address(es) in addition to their primary address and are said to be extended talkers and/or extended listeners. The function codes describing these functions are TE and LE and listed in Appendix D.

Frequently secondary addressing is used to access additional operating modes on a single device or a specific device within a rack of devices where the rack is assigned the primary address. In either case, the electrical load to the IEEE-488 bus should only be the equivalent of 1 device. To access such a device, a command from the LAG or TAG group would be issued for the primary address and followed immediately by a command from the secondary command group as described in Section B.5.

☞ ***NOTE:*** Most IEEE instruments are assigned a device address by setting hardware DIP switches, front panel controls, or by running some type of setup software.

Sending and Receiving Data/Device Commands

Data/Device Commands is a message which is sent over the bus with ATN unasserted. For example, a multimeter might send the results of several readings to a printer or display. Data can be sent by any device on the bus which is a talker.

The *Device Commands* control what tasks the IEEE-488 instrument performs. For example, a sequence of these commands might set a meter to a particular measuring range. These commands are *device-specific*. That is, the command required to set the voltage range of one manufacturer's multimeter cannot necessarily be used to set the voltage range on a multimeter produced by another company. The device(s) which is addressed to listen can distinguish *Device Commands* from data because certain character or command sequences are included.

Newer devices which conform to the IEEE-488.2 and or SCPI (Standard Commands for Programmable Instruments) specifications may have more standardized command sets. Consult the documentation accompanying the device for its command set. *Device Commands* can be issued by any device on the bus which is a talker.

Message Terminators
A Message Termination scheme is required if messages of unknown length are to be sent in order for the receiving device to know when the data transmission has ended. One way of terminating a message is to employ the End or Identify (EOI) line. (See Section B.5) The device transmitting the data will assert the EOI when it puts the last data byte on the DIO lines. The receiving device then recognizes that the byte it receives with the EOI will be the last.

As second termination scheme is for the transmitting device to append one or two characters (which would normally not appear in the message) to the end of the message. The characters causing a carriage return and line feed are frequently used where the message is a string of text. If the message consists of values between 0 and 255 then termination characters cannot be used because they might be mistaken for data (Carriage return = 13, line feed = 10). In this case, an EOI would have to be used or frequently the number of data bytes to be sent is known so that the receiver could accept that amount of data.

Usually devices provide some flexibility in the terminators they support. By means of switches or programming one can choose whether or not termination will be used and if so, whether termination characters and/or EOI will be used.

TRIGGERING

The *Active Controller* can issue the addressed multiline message of Group Enable Trigger (GET) which will cause devices to start executing some function such as to make a measurement. This allows the active controller to synchronize various activities. Whether a device support trigger functions is defined by its DT capability code of Appendix D. See Section B.5 for further information on GET.

REQUESTING SERVICE

The service request line (SRQ) introduced in Section B.5 provides a means for bus devices to request service from the *Active Controller*. When a device requires service, as for example, when it has completed a task, the device will assert the SRQ line. All bus devices share the SRQ line so it will be necessary for the controller to use the polling techniques of the next section to determine which device is responsible for the SRQ. It is also because a device will not unassert the SRQ line until it has been serially polled.

The service request capability of the device is defined by the SR code of Appendix D and the controller must have C4 capability in order to respond to the SRQ.

Polling

Polling is used on the IEEE-488 bus to ascertain if a device needs service. For example, if it needs to pass data to the *Active Controller.* There are two types of polling which are used on the IEEE-488 interface: **serial** and **parallel.** Often, they are used in combination. For example, sometimes parallel poll is followed by a serial poll. This enables the *Active Controller* to determine the type of service needed by a device.

Serial Polling

Serial polling permits the *Active Controller* to find out whether any device(s) needs service. The *Active Controller* serial polls one device at a time by first issuing the serial poll enable (SPE) multiline message of Section B.5. Now when a device is addressed to talk the device will return a special status byte. If the bit returned on DIO-7 is 1, the device requires service. The other bits indicate user-defined status and can indicate why the SRQ was asserted. The controller can conduct a serial poll even when an SRQ is not generated in order to determine the status information. If a device has asserted SRQ, it must be polled before it will release SRQ.

At the end of a serial poll, the controller will issue the serial poll disable (SPD) message of Section B.5 and the next time the device is addressed to talk, it will return to its normal data.

Devices must have the talker (T) or extend talker (ET) capability as listen in Appendix D in order to return a status byte.

Parallel Polling

Parallel Polling allows the *Active Controller* to check the status of up to 8 devices (or groups of services) at the same time to determine which device(s) may require service. When the *Active Controller* asserts both the ATN and EOI lines, devices which support parallel polling will return a status bit via one of the DIO lines. If the parallel poll indicates a device needs attention, the *Active Controller* may have to conduct a serial poll of the device to determine the reason the device requires service.

There must also be some mechanism to clear the bit the device returns for a parallel poll. Frequently this bit is tied to the SRQ request. In this case, a device generates a SRQ at the same time it sets the bit that will be returned by the parallel poll. The *Active controller* conducts a parallel poll to rapidly determine the device requiring service and then a serial poll to gain more information about the cause of the SRQ and to clear the SRQ and the bit that will be returned by parallel polling.

Depending on the device the DIO line assignment will be allocated by the controller or by switches or jumpers on the device. If the device can be assigned a line by the controller, the controller will do so by issuing a parallel poll configure (PPC) interface message followed by a parallel poll enable (PPE) interface message.

A relative few number of devices support parallel poll. Their capability including the manner of DIO assignment is specified by the PP code Appendix D. Only certain controller C codes support parallel poll.

B.5 BUS INTERFACE COMMANDS

Bus commands are issued by the *Active Controller*. There are five types of bus commands:

- Universal
- Listen Address Group (LAG)
- Talk Address Group (TAG)
- Addressed Commands
- Secondary Commands

These are described within this section. Also refer to Appendix C for an ASCII table containing a complete interface message summary.

Universal Commands

Devices on the bus respond to these commands whether they have been addressed or not. However, the commands may affect different devices in different manners. Note too that all commands are not necessarily supported by all devices. The interface capability codes of Appendix D are used to specify the functionality of a device. In order to issue one of these commands, the *Active Controller* must go through the following sequence:

- Assert the ATN line.
- Place the desired command byte on the data bus.

Descriptions of the *Universal Commands* are shown in Table B-3.

Table B-3. Universal Commands

Acronym	Command Name	Description
DCL	Device Clear	This command re-initializes the device. This is device-dependent.
LLO	Local Lockout	This command disables the device's front panel LOCAL button.
SPE	Serial Poll Enable	This command enables serial poll mode. When addressed to talk, the device will return a single status byte.
SPD	Serial Poll Disable	This command disables serial polling. Upon being addressed, a device will return to its normal state and begin outputting device-dependent data.
PPU	Parallel Poll Unconfi-gure	This command resets all parallel poll devices to the idle state (They will not respond to a parallel poll.).

Talk Address Group (TAG)

The Talk Address Group (TAG) message defines the specified device to be an *active talker.* Only one device can be an *active talker* at a time. The message contains the primary address (0 to 30) of the device which is to talk. This address consists of a primary address in the range 0 to 30. (Address 31 can be used to UNTALK all devices.) This may be accompanied by a secondary address in the range 0 to 31.

Generally, when an *Active Controller* issues a TAG command, it:

- Asserts the ATN line.
- Untalks all devices.
- Sends a TAG.
- Unasserts the ATN line.
- The talker then sends its data.

Listen Address Group (LAG)

The Listen Address Group (LAG) command defines the specified device(s) to be an *Active Listener.* A command from this group contains the bus address of the device to be listened. This address consists of a primary address in the range 0 to 30. This may be accompanied by a secondary address in the range 0 to 31. Note that sending a primary address of 31 will unlisten all devices. Generally, when an *Active Controller* issues a LAG command, it:

- Asserts the ATN line.
- Unlistens all devices.
- Sends a LAG with the address(es) of the device(s) to listen.
- Unasserts the ATN line.
- Sends data.

Addressed Commands

These commands are issued by the *Active Controller* and affect only those devices which have been properly addressed. Not all devices support these commands.

In order to issue an *Addressed Command,* the *Active Controller* must go through the following sequence:

- Assert the ATN line.
- Address the device(s) to listen.
- Place the command byte on the data bus.

The addressed commands are shown in Table B-4.

Table B-4. Primary Addressed Commands

Acronym	Command Name	Description
GET	Group Execute Trigger	This command allows you to trigger a group of devices concurrently.
SDC	Selected Device Clear	This initializes the addressed device to its reset state. This is device-dependent.
GTL	Go to Local	This command allows the device to be programmed locally, i.e.. through the switches on the front panel. Once the device is addressed to listen again, it will exit the local mode.
PPC	Parallel Poll Configure	When combined with the use of the secondary commands PPE and PPD, this command enables/disables the addressed device to be remotely parallel polled by the controller.
TCT	Take Control	This allows the active controller to pass control to another controller on the system. The second controller then becomes the active controller.

Secondary Commands

Secondary commands are sent immediately following a PPC (Parallel Poll Configure), TAG (Talk Address Group), or LAG (Listen Address Group). Secondary commands following a member of the TAG or LAG cause the device identified by the primary and secondary address to be an active talker or listener. The sequence would be:

- Assert the ATN line.
- Place a member of the TAG or LAG group containing the primary address on the data bus.
- Place a secondary command containing the secondary address on the data bus.
- Unassert the ATN line.

Secondary commands following PPC are divided into the Parallel Poll Enable group and the Parallel Poll Disable group. Recall that PPC requires devices to be addressed as listeners. The sequence in this case will be:

- Assert the ATN line.
- Address the appropriate device(s) to listen (including a secondary address if required).

- Place PPC on the data lines.
- Place a command from the PPC group (to enable) or from the PPD group (to disable) on the data lines.
- Unassert the ATN line.

Any member of the PPD group will disable the addressed device(s) from responding to a parallel poll. To enable a device(s) to respond to a parallel poll, the 3 lowest bits of the PPE command form a code of 0 to 7 which tells the device to control the data line 1 to 8 when a parallel poll is conducted. Setting the 4th lowest bit of the PPE command tells the device to assert its assigned line when service is required while setting the 4th lowest bit low will cause the device to assert its line when service is not required.

B.6 REFERENCE DOCUMENTS

If you require more detailed information than this tutorial provides, refer to the following documents:

- *ANSI/IEEE 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation*
- *ANSI/IEEE 488.2-1987, Codes, Formats, Protocols and Common Commands for Use with IEEE 488.1-1987*

The above two documents are available from:

IEEE Service Center
445 Hoes Lane
Piscataway, N.J. 08855
(800)678-IEEE

- *Standard Commands for Programmable Instruments Manual*

This document is available from:

SCPI Consortium
8380 Hercules Drive, Suite P3
La Mesa, California 92042
(619)697-5955

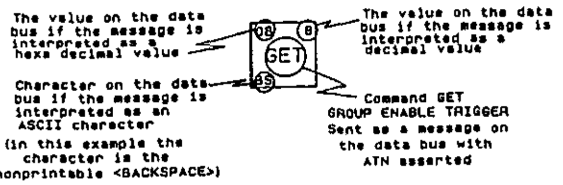
IEEE MULTILINE COMMANDS

		20 32 00 SP	30 48 16 0	40 64 00 @	50 80 16 P	60 96 00 `	70 112 16 p
01 1 GTL SOH	11 17 LLO DC1	21 33 01 !	31 49 17 1	41 65 01 A	51 81 17 Q	61 97 01 a	71 113 17 q
		22 34 02 "	32 50 18 2	42 66 02 B	52 82 18 R	62 98 02 b	72 114 18 r
		23 35 03 #	33 51 19 3	43 67 03 C	53 83 19 S	63 99 03 c	73 115 19 s
04 4 SDC EOT	14 20 DCL DC4	24 36 04 $	34 52 20 4	44 68 04 D	54 84 20 T	64 100 04 d	74 116 20 t
05 5 PPC ENQ	15 21 PPU NAK	25 37 05 %	35 53 21 5	45 69 05 E	55 85 21 U	65 101 05 e	75 117 21 u
		26 38 06 &	36 54 22 6	46 70 06 F	56 86 22 V	66 102 06 f	76 118 22 v
		27 39 07 '	37 55 23 7	47 71 07 G	57 87 23 W	67 103 07 g	77 119 23 w
08 8 GET BS	18 24 SPE CAN	28 40 08 (	38 56 24 8	48 72 08 H	58 88 24 X	68 104 08 h	78 120 24 x
09 9 TCT HT	19 25 SPD EM	29 41 09)	39 57 25 9	49 73 09 I	59 89 25 Y	69 105 09 i	79 121 25 y
		2A 42 10 *	3A 58 26 :	4A 74 10 J	5A 90 26 Z	6A 106 10 j	7A 122 26 z
		2B 43 11 +	3B 59 27 ;	4B 75 11 K	5B 91 27 [	6B 107 11 k	7B 123 27 {
		2C 44 12 ,	3C 60 28 <	4C 76 12 L	5C 92 28 \	6C 108 12 l	7C 124 28 |
		2D 45 13 -	3D 61 29 =	4D 77 13 M	5D 93 29]	6D 109 13 m	7D 125 29 }
		2E 46 14 .	3E 62 30 >	4E 78 14 N	5E 94 30 ^	6E 110 14 n	7E 126 30 ~
		2F 47 15 /	3F 63 UNL ?	4F 79 15 O	5F 95 UNT _	6F 111 15 o	7F 127 31 DEL
ACG	UCG	LAG		TAG		PPE SCG	PPD

MULTILINE MESSAGES

IEEE MULTILINE COMMANDS

Multiline Commands consist of Multiline Messages sent over the data bus with the ATN control line asserted. The messages are given in the chart at the left. Each message is enclosed in a box. the content of a block is interpreted as:


The value on the data bus if the message is interpreted as a hexa decimal value
08
8
GET
BS
The value on the data bus if the message is interpreted as a decimal value
Character on the data bus if the message is interpreted as an ASCII character
(in this example the character is the nonprintable <BACKSPACE>)
Command GET GROUP ENABLE TRIGGER Sent as a message on the data bus with ATN asserted

The commands are grouped by columns as:

ACG (Addressed Command Group): These commands affect devices which have been addressed.
- GTL (Go To Local): Addressed devices accept panel controls
- SDC (Selected Device Clear): Return addressed *devices to power up programmed state*
- PPC (Parallel Poll Configure): Configure an addressed device's Parallel Poll response (requires secondary command)
- GET (Group Execute Trigger): Addressed devices execute some preassigned function
- TCT (Take Control): Active controller passes control to an addressed device

UCG (Universal Command Group); All devices respond
- LOL (Local Lockout): Disable panel REMOTE/LOCAL buttons
- DCL (Device Clear): Return all devices to power up programmed state
- PPU (Parallel Poll Unconfigure): Disables all devices from responding to Parallel Poll Command
- SPE (Serial Poll Enable): Device will return serial byte when addressed to talk
- SPD (Serial Poll Disable): Retracts SPE so device returns data when addressed to talk

LAG (Listen Address Group): Addresses/Unaddresses device(s) to listen
- 0..30 (GPIB Address): Addresses device(s) to listen (can be used more than once)
- UNL (Unlisten): Unlistens all devices

TAG (Talk Address Group): Address/Unaddress one device to talk
- 0..30 (GPIB Address): Addresses ONE device to talk
- UNT (Untalk): Untalks the talker

SCG (Secondary Command Group): Used for Secondary Addressing and *Parallel Poll Configuring*
- 0..31 (GPIB Secondary Address): When following a command from the LAG or TAG group a secondrary as well as primary address is assigned to the device
- 0..15 (Parallel Poll Enable PPE): When following a PPC command the 4 bit pattern 0 1 1 0 S P2 P1 P0 configures the parallel response of the addressed device. P2 P1 P0 define a number 0 to 7 which determines the data line DIO1 to DIO8 the device will control when the Active Controller simultaneously asserts ATN and EOI. The device asserts its assigned line true or false depending on whether its internal status bit (ist) is or is not the same sense as S.
- 16..31 (Parallel Poll Disable PPD): When following a *PPC command the 4 bit pattern* 0 1 1 1 x x x x disables the parallel response of the addressed device

IEEE UNILINE COMMANDS

Uniline commands use one (one exception of two) special command lines as follows:
- ATN (Attention): Used by the Active Controller with data bus to send the multiline commands above or with EOI for conducting a parallel poll
- IFC (Interface Clear): Used by the System Controller to initialize the GPIB bus. i.e., all devices unlistened untalk and System Controller regains active control
- EOI (End or Identify): Used by the Active Talker to *indicate an end of transmission* or by the Active Controller with ATN to command devices to send their parallel poll bit
- REN (Remote Enable): Turned on or off by the System Controller. When asserted devices can be programmed remotely by messages sent over the GPIB bus rather than by the device's knobs and buttons
- SRQ (Service Request): Asserted by bus addresses when they require attention from the Active controller

DEVICE CAPABILITY CODES

AH Function Allowable Subsets

Identification	Description	Other Function Subsets Required
AH0	No capability	None
AH1	complete	None

SH Function Allowable Subsets

Identification	Description	Other Function Subsets Required
SH0	No capability	None
SH1	Complete capability	T1-T8,TE1-TE8, or C5-C28

T Function Allowable Subsets

Identification	Description				Other Function Subsets Required
	Basic Talker	Serial Poll	Talk Only Mode	Unaddress if MLA	
T0	N	N	N	N	None
T1	Y	Y	Y	N	SH1 and AH1
T2	Y	Y	N	N	SH1 and AH1
T3	Y	N	Y	N	SH1 and AH1
T4	Y	N	N	N	SH1 and AH1
T5	Y	Y	Y	Y	SH1 and L1-L4 or LE1-LE4
T6	Y	Y	N	Y	SH1 and L1-L4 or LE1-LE4
T7	Y	N	Y	Y	SH1 and L1-L4 or LE1-LE4
T8	Y	N	N	Y	SH1 and L1-L4 or LE1-LE4

T Function (With Address Extension) Allowable Subsets

Identification	Description				Other Function Subsets Required
	Basic Talker	Serial Poll	Talk Only Mode	Unaddress if MSA ^LPAS	
TE0	N	N	N	N	None
TE1	Y	Y	Y	N	SH1 and AH1
TE2	Y	Y	N	N	SH1 and AH1
TE3	Y	N	Y	N	SH1 and AH1
TE4	Y	N	N	N	SH1 and AH1
TE5	Y	Y	Y	Y	SH1 and L1-L4 or LE1-LE4
TE6	Y	Y	N	Y	SH1 and L1-L4 or LE1-LE4
TE7	Y	N	Y	Y	SH1 and L1-L4 or LE1-LE4
TE8	Y	N	N	Y	SH1 and L1-L4 or LE1-LE4

RL Function Allowable Subsets

Identification	Description	Other Function Subsets Required
RL0	No capability	None
RL1	Complete capability	L1-L4, or LE1-LE4
RL2	No Local Lockout	L1-L4, or LE1-LE4

PP Function Allowable Subsets

Identification	Description	Other Function Subsets Required
PP0	No capability	None
PP1	Remote capability	L1-L4, or LE1-LE4
PP2	Local Configuration	None

DC Function Allowable Subsets

Identification	Description	Other Function Subsets Required
DC0	No capability	None
DC1	Complete capability	L1-L4, or LE1-LE4
DC2	Omit Selective Device Clear	AH1

DT Function Allowable Subsets

Identification	Description	Other Function Subsets Required
DT0	No capability	None
DT1	Complete capability	L1-L4, or LE1-LE4

L Function Allowable Subsets

Identification	Description Basic Listener	Listen Only Mode	Unaddress if MTA	Other Function Subsets Required
L0	N	N	N	None
L1	Y	Y	N	AH1
L2	Y	N	N	AH1
L3	Y	Y	Y	AH1 and T1-T8 or TE1-TE8
L4	Y	N	Y	AH1 and T1-T8 or TE1-TE8

L Function (with Address Extension) Allowable Subsets

Identification	Description Basic Listener	Listen Only Mode	Unaddress if MSA ^ $\boxed{\text{TPAS}}$*	Other Function Subsets Required
LE0	N	N	N	None
LE1	Y	Y	N	AH1
LE2	Y	N	N	AH1
LE3	Y	Y	Y	AH1 and T1-T8 or TE1-TE8
LE4	Y	N	Y	AH1 and T1-T8 or TE1-TE8

* Replaced by MTA when used together with the T function

SR Function Allowable Subsets

Identification	Description	Other Function Subsets Required
SR0	No capability	None
SR1	Complete Capability	T1,T2,T5,T6,TE1,TE2,TE5,or TE6

C Function Allowable Subsets

Identification *	Capabilities										Notes
	System Controller	Send IFC or TCT	Send REN	Respond to SRQ	Send I.F. Messages	Receive Control	Pass Control	Pass Control to Self	Parallel Poll	Take Control Synchronously	
C0	N	N	N	N	N	N	N	N	N	N	
C1	Y										1
C2		Y									1,6
C3			Y								1
C4				Y							1
C5					Y	Y	Y	Y	Y	Y	2,3
C6					Y	Y	Y	Y	Y	N	2,3
C7					Y	Y	Y	Y	N	Y	2,3
C8					Y	Y	Y	Y	N	N	2,3
C9					Y	Y	Y	N	Y	Y	2,3
C10					Y	Y	Y	N	Y	N	2,3
C11					Y	Y	Y	N	N	Y	2,3
C12					Y	Y	Y	N	N	N	2,3
C13					Y	Y	N	N	Y	Y	2
C14					Y	Y	N	N	Y	N	2
C15					Y	Y	N	N	N	Y	2
C16					Y	Y	N	N	N	N	2
C17					Y	N	Y	Y	Y	Y	2,3,4
C18					Y	N	Y	Y	Y	N	2,3,4
C19					Y	N	Y	Y	N	Y	2,3,4
C20					Y	N	Y	Y	N	N	2,3,4
C21					Y	N	Y	N	Y	Y	2,3,4
C22					Y	N	Y	N	Y	N	2,3,4
C23					Y	N	Y	N	N	Y	2,3,4
C24					Y	N	Y	N	N	N	2,3,4
C25					Y	N	N	N	Y	Y	2,5
C26					Y	N	N	N	Y	N	2,5
C27					Y	N	N	N	N	Y	2,5
C28					Y	N	N	N	N	N	2,5

* Typical notation to describe a controller consists of the letter C followed by one or more of the numbers indicating the subsets selected. For example: C1,2,3,4,8.

NOTES:

1. One or more of subsets C1 through C4 may be chosen in any combination with any one of C5 through C28.
2. Only one subset may be chosen from C5 through C28.
3. The CTRS state must be included in devices which are to be operated in multicontroller systems.
4. These subsets are not allowed unless C2 is included.
5. These subsets are intended to be used in devices and systems where no control passage is possible.
6. When a system controller asserts IFC during the time another physical device is operating as controller-in-charge, the system controller should refrain from active assertion of the source handshake and ATN until the rmoval of the IFC message to preclude multiple controller contention.

PRINTER AND SERIAL PORT RE-DIRECTION

The *KMLPT* and *KMCOM* utilities automatically redirect communications destined for printer or serial ports to specified IEEE-488 bus devices. This is useful in that it allows application programs which are unaware of the IEEE-488 bus to control bus devices as if they were printer (*KMLPT*) and serial (*KMCOM*) devices.

Before you use these programs, you must understand the difference between logical and physical printer port devices. A *physical device* is the actual port which is installed in the computer. For example, you might have two parallel printer ports and one serial communications port installed in your computer. These are the *physical devices.* Physical devices are depicted by using the port name. For example, the first printer port identified by the computer is referred to as LPT1, the second LPT2, etc.

A *logical device* is a device which is currently configured to receive the data to be printed. *Logical devices* are represented using a colon, for example LPT1:. (This would indicate the device which is currently configured to receive the data to be printed.)

The computer maintains two tables, each of which has four entries. These tables are used to assign a physical device to a logical device. For example, if two printer ports and one serial port were installed, these tables would initially appear as:

	Printer Assignments		**Serial Port Assignments**
LPT1:	LPT1	COM1:	COM1
LPT2:	LPT2	COM2:	None
LPT3:	None	COM3:	None
LPT4:	None	COM4:	None

PARALLEL PORT RE-DIRECTION

Parallel Port re-direction is accomplished by using the *KMLPT* utility. This is a unidirectional re-director which intercepts a character from the DOS BIOS and writes it to the GPIB via an LPT: port. This accomplished by assigning the logical LPT: port to a GPIB device address.

The next sections describe how to load/unload the *KMLPT* re-director from the DOS command line. If you need help loading *KMLPT*, from the DOS command line, type:

KMLPT /HELP

The *KMLPT* utility is invoked as follows:

1. Change to the directory where your KM-488-DD software is located.

2. At the DOS prompt, type:

   KMLPT *n1 /A&Hioaddr /Baddr /t*

   Where:

   n1 . . . n4 are up to 4 optional device parameters. Each is of the format **IEEEppss** or **LPTn** where:

   > **IEEEppss** identifies the IEEE-488 device. **ppss** is the address of the IEEE-488 device. **pp** is the address of the IEEE-488 device. This is a primary address, with a secondary address (**ss**) if needed. For example, you might specify the device **IEEE2022**.
   >
   > **LPTn** identifies a physical printer port where **n** is the printer port number, i.e. **LPT1**.

   /A&Hioaddr is a requried parameter which follows the **n1** parameter. It specifies the I/O Base Address (in hex) of the KM-488-DD.

   /Baddr is an optional parameter which follows the **n1** parameter. It specifies the IEEE-488 Bus address (0 to 30 decimal) of the IEEE-488 interface board and must be included if the IEEE-488 interface board is not located at the default address of 0 decimal.

   /t is an optional parameter which specifies the timeout period. This can be any value between 1 to 30 seconds. The default value is 1 second. The timeout period should be set long enough to allow for the slowest plotter function.

Notes

- If *KMLPT* is executed with no arguments, then it just displays the current logical printer port assignments.
- If one or more arguments are provided, then the first logical printer port (LPT1:) is re-directed to the physical device by the first argument, the next logical port (LPT2:) is re-directed to the next specified physical, and so on.
- If less than four devices are specified, then the remaining logical printers are re-directed to any unused physical parallel printer ports.

Examples

These examples assume that your PC has two functioning LPT ports.

`KMLPT IEEE05 /A&H2E1`

Configures LPT1: for output to IEEE device 05 on an interface card located at 2E1 (hex).

`KMLPT LPT1 IEEE05 /A&H2E1`

<u>Resulting Printer Port Table</u>

LPT1:	LPT1
LPT2:	IEEE05
LPT3:	LPT2
LPT4:	None

`KMLPT IEEE05 IEEE1201 /A&H2E1`

<u>Resulting Printer Port Table</u>

LPT1:	IEEE05
LPT2:	IEEE1201
LPT3:	LPT1
LPT4:	LPT2

Resulting GPIB Bus Activity

When the *KMLPT* changes from one GPIB bus address to another, the GPIB activity will be as follows: REN is asserted followed by the ATN line, then the following bus "commands" are sent UNT, UNL, MTA, LA. ATN is unasserted and the data is sent.

If the GPIB bus address used by the *KMLPT* re-direct driver remains the same, the data is simply sent over the bus.

UNLOADING KMLPT FROM DOS

To unload the *KMLPT* utility from the DOS command line:

1. Change to the directory where your KM-488-DD software is located.

2. At the DOS prompt, type:

```
KMLPT /U
```

Notes

- If the driver is already resident and re-direction is requested, the printer assignments are altered and reported.
- Both of the KMCOM and KMLPT drivers may be loaded at the same time and name the same IEEE addresses. The drivers must be unloaded in reverse order of loading.
- If any other TSR is loaded after the re-director, it will not be possible to unload the re-director until subsequent drivers are unloaded.
- The IFC message is sent when the driver loads.

SERIAL PORT RE-DIRECTION

Serial Port re-direction is accomplished in the same manner as Parallel Port re-direction. The only difference is that you use the *KMCOM* utility. This is a bi-directional redirector which intercepts a character request from DOS BIOS and reads/writes the data from/to the GPIB. If data is read from the GPIB, the driver will execute synchronous inputs. This insures that data will not be lost if a different GPIB bus address from the previous one is used. Note, however, that some devices may "flush" their output buffer when they are "un-addressed".

The next sections describe how to load and unload the *KMCOM* re-director from the DOS command line. If you need help loading *KMCOM*, from the DOS command line, type:

```
KMCOM /HELP
```

LOADING OR CHANGING KMCOM FROM DOS

To load the *KMCOM* utility from the DOS command line:

1. Change to the directory where your KM-488-DD software is located.

2. At the DOS prompt, type:

   KMCOM *n1 /Ioaddr /Baddr /t*

   Where:

   ☞ ***NOTE:*** Parameters must appear in all UPPER CASE or all lower case. UPPER CASE and lower case cannot be mixed.

   n1 . . . n4 designates a GPIB or COM port device. Up to a total of 4 devices may be specified.

   GPIB bus devices are denoted as **IEEEppss**, where:
   IEEEppss identifies the IEEE-488 device. **pp** is the address of the IEEE-488 device. This is a primary address, with a secondary address (**ss**)if needed. For example, you might specify the device **IEEE2022**.

   COM port devices are denoted as **COMn**, where:
   COMn identifies a physical printer port where **n** is the printer port number (1,2,3,or 4), i.e. **COM1**.

   /A&Hioaddr is a required parameter which follows the **n1** parameter. It specifies the I/O Base Address (in hex) of the KM-488-DD.

   /Baddr is an optional parameter which specifies the IEEE-488 Bus address (0 to 30 decimal) of the KM-488-DD. It must be included if the IEEE-488 interface board is not located at the default address of 00 decimal.

   /t is an optional parameter which specifies the timeout period. This can be any value between 1 to 30 seconds. The default value is 1 second. The timeout period should be set long enough to allow for the slowest plotter function.

Notes

- If *KMCOM* is executed with no arguments, then it just displays the current logical printer port assignments.
- If one or more arguments are provided, then the first logical COM port (COM1:) is re-directed to the physical device by the first argument, the next logical port (COM2:) is re-directed to the next specified physical, etc.
- If less than four devices are specified, then the remaining logical COM ports are re-directed to any unused physical COM ports.
- For the serial or parallel port to be re-directed effectively, the application program should be configured to send its output to a disk file rather than directly to the printer or plotter. If, for example, a file such as **`com1.dat`** is specified, the program will act as if it were writing the data to a genuine file. However, the output will really be sent to the IEEE bus device to which COM1 was re-directed. The program may even issue a warning message that the specified file exists and will be overwritten. If it does, instruct it to delete or overwrite the file.

☞ ***NOTE:*** When using COM port re-direction, it may be necessary to use the DOS **MODE** command to set the serial printer's parameters (baud rate, etc.). If the re-direction takes place before the printer is initialized, the **MODE** command should be invoked on the *logical device* (i.e., COM2:) to which the physical device has been re-assigned.

☞ ***NOTE:*** The DOS BIOS system always monitors the communications lines coming from the serial printer; therefore, the DSR, CD, RTS, etc. signals must be correctly terminated in order to communicate with the RS-232C printer.

Examples

These examples assume that your PC has two functioning COM ports.

`KMCOM IEEE05 /A&H2E1` Configures COM1: for output to IEEE device 05 on an interface card located at 2E1 (hex).

`KMCOM COM1 IEEE05 /A&H2E1`

Resulting Printer Port Table

COM1:	COM1
COM2:	IEEE05
COM3:	COM2
COM4:	None

`KMCOM IEEE05 IEEE1201 /A&H2E1`

Resulting Printer Port Table

COM1:	IEEE05
COM2:	IEEE1201
COM3:	COM1
COM4:	COM2

Resulting GPIB Bus Activity

When the *KMCOM* changes from one GPIB bus address to another, the GPIB activity will occur as follows:

On a Write

REN is asserted followed by the ATN line, then the following bus "commands" are sent UNT, UNL, MTA, LA. ATN is unasserted and the data is sent.

On a Read

REN is asserted followed by the ATN line, then the following bus "commands" are sent UNT, UNL, MLA, TA. ATN is unasserted and the data is received.

If the GPIB bus address used by the *KMCOM* re-direct driver remains the same, the data is simply sent or received over the bus.

UNLOADING KMCOM FROM DOS

To unload the *KMCOM* utility from the DOS command line:

1. Change to the directory where your KM-488-DD software is located.

2. At the DOS prompt, type:

```
KMCOM /U
```

Notes

- If the driver is already resident and re-direction is requested, the COM port assignments are altered and reported.

- Both of the KMCOM and KMLPT drivers may be loaded at the same time and name the same IEEE addresses. The drivers must be unloaded in reverse order of loading.

- If any other TSR is loaded after the re-director, it will not be possible to unload the re-director until subsequent drivers are unloaded.

- The IFC message is sent when the driver loads.

APPLICATION NOTES

You may encounter several problems which attempting to send plotter files to your GPIB plotter. For example, Direct Output to I/O ports can be a problem because many applications will use their own I/O driver routines rather than the DOS BIOS routines that the redirector intercepts. These routines will directly route the data to a hardware I/O card. This is particularly true with COM ports or input devices which are installed on COM ports.

Another problem which may occur is that communications are successfully established with the requested port; however a plotter error occurs. This is usually caused by the fact that the application thinks that it is talking to an RS-232C plotter and has interspersed software handshaking commands, with the plotter graphics commands, that the GPIB plotter does not understand. To avoid this problem, determine if your application will allow you to turn off this hardware handshaking. If you can, strip out the RS-232 handshaking commands and send a pure plot file to a port (i.e., use indirect output).

If you are Indirectly Outputting your plot files, try to name your file something which includes an I/O port name (e.g., COM3.X). However, this may result in the program searching the DOS device driver list and finding a matching device name. If this happens, the application may refuse to create a file with the same name as a device. If all else fails, create a plot file, exit the application, and send the plot file to the re-directed device.

EXAMPLE PROGRAM

An example program in BASICA, *COMTEST.BAS*, is provide on the KM-488-DD Disks. This example program illustrates how to use the *KMCOM* re-director feature.

A plot file, *HPEXAMPL.PLT*, is also provided on the KM-488-DD Disks. This file can be printed to an HP plotter using the KMLPT Re-Direct Driver.

(This page intentionally left blank.)

THE CONFIG.DAT FILE

The *CONFIG.DAT* file is generated by the SETUP or CONFIG programs. It must be "piped" into the KM-488-DD driver file when the driver is loaded. (This process is explained further in Chapter 2.) To pipe in the file, at the DOS prompt, type:

KM < CONFIG.DAT

The *CONFIG.DAT* file is of the following format;

```
COMMENT
COMMENT
COMMENT ********************************************************************
COMMENT *                                                                  *
COMMENT * CONFIG.DAT : Optional KM-488-DD Driver Configuration File        *
COMMENT *                                                                  *
COMMENT * PURPOSE : To Configure The KM-488-DD Driver When It Is Loaded    *
COMMENT *                                                                  *
COMMENT * OPTIONS : See The Configuration Options Discussed Below          *
COMMENT *                                                                  *
COMMENT * TO USE -- Type :                                                 *
COMMENT *                                                                  *
COMMENT *                    KM < CONFIG.DAT  (Enter)                      *
COMMENT *                                                                  *
COMMENT *          At The DOS Prompt                                       *
COMMENT *                                                                  *
COMMENT ********************************************************************
COMMENT
COMMENT
COMMENT This file accomplishes the following commands :
COMMENT
COMMENT CONFIG /COUNT 1
COMMENT CONFIG /CARD 1 /A&H2E1 /SC /B00 /I /F8 /D3
COMMENT CONFIG /E CR LF /E $13 $10 /T CR LF EOI /T $13 $10 EOI
COMMENT TIMEOUT 15.0
COMMENT
COMMENT For clarity, the commands have been broken down into individual
COMMENT parameters and explained separately.
COMMENT
COMMENT
```

```
        CONFIG  /COUNT 1
COMMENT
COMMENT
COMMENT *******************************************************************
COMMENT *                                                                 *
COMMENT * "/COUNT n" : Configure The Number Of KM-488-DD Boards Installed *
COMMENT *                                                                 *
COMMENT * PURPOSE : Tells The KM Driver How Many Boards Are Installed     *
COMMENT *                                                                 *
COMMENT * OPTIONS : Possible Values For n Are 1, 2, 3 or 4                *
COMMENT *                                                                 *
COMMENT * DEFAULT : If Not Specified, KM.EXE Assumes 1 Card Is Installed  *
COMMENT *                                                                 *
COMMENT * SPECIAL : If Specified, It Must Be The First Line In CONFIG.DAT *
COMMENT *                                                                 *
COMMENT *******************************************************************
COMMENT
COMMENT
        CONFIG /CARD 1
COMMENT
COMMENT
COMMENT *******************************************************************
COMMENT *                                                                 *
COMMENT * "/CARD n" : Specify Card To Which Hardware Parameters Are Applied*
COMMENT *                                                                 *
COMMENT * PURPOSE : Select Which Card To Configure                        *
COMMENT *                                                                 *
COMMENT * OPTIONS : Possible Values For n Are 1 - 4 For n Installed Cards *
COMMENT *                                                                 *
COMMENT * DEFAULT : No Default -- Must Be Specified For :                 *
COMMENT *                                                                 *
COMMENT *           "/B", "/SC", "/F", "/I" & "/D"                        *
COMMENT *                                                                 *
COMMENT * SPECIAL : Must Appear As First Parameter On A Line              *
COMMENT *                                                                 *
COMMENT *******************************************************************
COMMENT
        CONFIG /CARD 1 /A &H2E1
COMMENT
COMMENT *******************************************************************
COMMENT *                                                                 *
COMMENT * "/A &Hnnnn" : Specify Base Address For KM-488-DD Interface Card *
COMMENT *                                                                 *
COMMENT * PURPOSE : Select Base Address Of KM-488-DD Card In PC I/O Space *
COMMENT *                                                                 *
COMMENT * OPTIONS : &H2E1, &H22E1, &H42E1, &H62E1                         *
COMMENT *                                                                 *
COMMENT * DEFAULT : Unless Specified, KM.EXE Assumes Each Card Is At &H2E1 *
COMMENT *                                                                 *
COMMENT * SPECIAL : The "&H" Means Number Is Hexidecimal And Must Be Used *
COMMENT *                                                                 *
COMMENT *******************************************************************
COMMENT
```

```
        CONFIG  /CARD 1 /SC
COMMENT
COMMENT *********************************************************************
COMMENT *                                                                   *
COMMENT * "/SC" or "/SC0" : Specify Controller Status Of KM-488-DD Card     *
COMMENT *                                                                   *
COMMENT * PURPOSE : Configure KM-488-DD As System Controller Or Peripheral *
COMMENT *                                                                   *
COMMENT * OPTIONS : "/SC" = System Controller "/SC0" = Peripheral          *
COMMENT *                                                                   *
COMMENT * DEFAULT : "/SC" Or System Controller                              *
COMMENT *                                                                   *
COMMENT * SPECIAL : none                                                    *
COMMENT *                                                                   *
COMMENT *********************************************************************
COMMENT
        CONFIG /CARD 1 /B00
COMMENT
COMMENT *********************************************************************
COMMENT *                                                                   *
COMMENT * "/Bnn" : Specify GPIB Address nn Of KM-488-DD Interface Card     *
COMMENT *                                                                   *
COMMENT * PURPOSE : Configure Driver To Address KM-488-DD Card At Addr. nn *
COMMENT *                                                                   *
COMMENT * OPTIONS : nn Is A Decimal Number From 00 To 31                   *
COMMENT *                                                                   *
COMMENT * DEFAULT : GPIB Address 00                                         *
COMMENT *                                                                   *
COMMENT * SPECIAL : Leading Zero Is Required On Addresses Less Than 10     *
COMMENT *                                                                   *
COMMENT *********************************************************************
COMMENT
        CONFIG /CARD 1 /I
COMMENT
COMMENT *********************************************************************
COMMENT *                                                                   *
COMMENT * "/I" or "/I0" : Light Pen Emulation                               *
COMMENT *                                                                   *
COMMENT * PURPOSE : Enable/Disable Light Pen Emulation In KM-488-DD Driver *
COMMENT *                                                                   *
COMMENT * OPTIONS : "/I" = Enable Light Pen "/I0" = Disable Light Pen      *
COMMENT *                                                                   *
COMMENT * DEFAULT : Light Pen Emulation Off                                 *
COMMENT *                                                                   *
COMMENT * SPECIAL : Emulation Must Be Enabled For KARM And ARM Commands    *
COMMENT *                                                                   *
COMMENT *********************************************************************
COMMENT
```

```
        CONFIG /CARD 1 /F8
COMMENT
COMMENT *********************************************************************
COMMENT *                                                                   *
COMMENT * "/Fn" : Clock Frequency                                           *
COMMENT *                                                                   *
COMMENT * PURPOSE : Match 7210 Timing To Input Clock Frequency              *
COMMENT *                                                                   *
COMMENT * OPTIONS : n Is A Decimal Number From 1-8 Signifying MegaHertz     *
COMMENT *                                                                   *
COMMENT * DEFAULT : "/F8" Or 8 Megahertz                                    *
COMMENT *                                                                   *
COMMENT * SPECIAL : When Using On Board KM-488-DD Crystal, Specify "/F8"    *
COMMENT *                                                                   *
COMMENT *********************************************************************
COMMENT
        CONFIG /CARD 1 /D1
COMMENT
COMMENT *********************************************************************
COMMENT *                                                                   *
COMMENT * "/Dn" : DMA Level                                                 *
COMMENT *                                                                   *
COMMENT * PURPOSE : Specify The DMA Level Used By The KM-488-DD Card        *
COMMENT *                                                                   *
COMMENT * OPTIONS : "/D1" = Level 1, "/D3" = Level 3, "/D" = No DMA Used    *
COMMENT *                                                                   *
COMMENT * DEFAULT : DMA Level 1                                             *
COMMENT *                                                                   *
COMMENT * SPECIAL : DMA Levels Are Not Shareable Between Cards              *
COMMENT *                                                                   *
COMMENT *********************************************************************
COMMENT
COMMENT
COMMENT The Following CONFIG Statements Each Set The EOL Sequence To
COMMENT Carriage Return (0D Hex) / Line Feed (0A Hex)
COMMENT
COMMENT
        CONFIG /E CR LF
COMMENT
COMMENT
```

```
        CONFIG /E $13 $10
COMMENT
COMMENT
COMMENT *******************************************************************
COMMENT *                                                                 *
COMMENT * "/E eol1 eol2" : Configure Device End Of Line (EOL) Terminators  *
COMMENT *                                                                 *
COMMENT * PURPOSE : Specify Character(s) That Signify EOL To KM-488-DD     *
COMMENT *                                                                 *
COMMENT * OPTIONS :                                                       *
COMMENT *                                                                 *
COMMENT *   "/E"           = No EOL Detection Occurs In KM-488-DD Driver  *
COMMENT *   "/E eol1"      = Character eol1 Signifies End Of Line         *
COMMENT *   "/E eol1 eol2" = Characters eol1 and eol2 together mean EOL   *
COMMENT *                                                                 *
COMMENT *   where eol1 and eol2 are one of the following :                *
COMMENT *                                                                 *
COMMENT *   $nnn : ASCII Character nnn (0 <= nnn <= 255)                  *
COMMENT *   'X   : Printable ASCII Character X                            *
COMMENT *   CR   : Carriage Return (13 Decimal)                           *
COMMENT *   LF   : Line Feed       (10 Decimal)                           *
COMMENT *                                                                 *
COMMENT * SPECIAL : Using "/EI" Instead Of "/E" Sets Input EOL's Only      *
COMMENT *           Using "/EO" Instead Of "/E" Sets Output EOL's Only     *
COMMENT *                                                                 *
COMMENT *******************************************************************
COMMENT
COMMENT
COMMENT The Following CONFIG Statements Each Set The Terminator Sequence To
COMMENT Carriage Return (0D Hex) / Line Feed (0A Hex) w/EOI Asserted
COMMENT
COMMENT
        CONFIG /T CR LF EOI
COMMENT
COMMENT
        CONFIG /T $13 $10 EOI
```

```
COMMENT
COMMENT
COMMENT ********************************************************************
COMMENT *                                                                  *
COMMENT * "/T trml trm2" : Configure Device Bus Terminators                *
COMMENT *                                                                  *
COMMENT * PURPOSE : Specify Terminators To Send (OUTPUT) / Detect (ENTER)  *
COMMENT *                                                                  *
COMMENT * OPTIONS :                                                        *
COMMENT *                                                                  *
COMMENT *   "/T"                = Send/Detect No Terminators               *
COMMENT *   "/T trml"           = Send/Detect trml                         *
COMMENT *   "/T trml trm2"      = Send/Detect trml & trm2                  *
COMMENT *   "/T EOI"            = Send/Detect Last Data Byte w/EOI Asserted *
COMMENT *   "/T trml"           = Send/Detect trml with EOI Asserted       *
COMMENT *   "/T trml trm2 EOI" = Send/Detect trml & trm2 w/EOI Asserted    *
COMMENT *                                                                  *
COMMENT *   where trml and trm2 are one of the following :                 *
COMMENT *                                                                  *
COMMENT *   $nnn : ASCII Character nnn (0 <= nnn <= 255)                   *
COMMENT *   'X   : Printable ASCII Character X                             *
COMMENT *   CR   : Carriage Return (13 Decimal)                            *
COMMENT *   LF   : Line Feed       (10 Decimal)                            *
COMMENT *                                                                  *
COMMENT * SPECIAL : Using "/TI" Instead Of "/T" Sets Input trm's Only      *
COMMENT *           Using "/TO" Instead Of "/T" Sets Output trm's Only     *
COMMENT *                                                                  *
COMMENT ********************************************************************
COMMENT
COMMENT
COMMENT Set The Timeout To Fifteen Seconds
COMMENT
COMMENT
       TIMEOUT 15.0
COMMENT
COMMENT ********************************************************************
COMMENT *                                                                  *
COMMENT * "TIMEOUT nnnnn.n" : Set The Device Timeout Value                 *
COMMENT *                                                                  *
COMMENT * PURPOSE : Specify How Long To Wait For A Device Response         *
COMMENT *                                                                  *
COMMENT * OPTIONS : nnnnn.n Is A Number Between 0.1 And 65535.9            *
COMMENT *                                                                  *
COMMENT * SPECIAL : "TIMEOUT 0" Turns Timeout Checking Off.                *
COMMENT *           Timeout Values Less Than 0.1 Are Rounded Up To .1      *
COMMENT *                                                                  *
COMMENT ********************************************************************
COMMENT
COMMENT
COMMENT Create Two Named Devices -- One, An Oscilloscope At GPIB Address 01
COMMENT And The Other A DMM At Primary Address 02 And Secondary Address 14
COMMENT
COMMENT
        CONFIG /N SCOPE 01 /N DMM 0214
COMMENT
```

```
COMMENT **********************************************************************
COMMENT *                                                                    *
COMMENT * "CONFIG /N devname addr" : Create A Symbolic Or "Named" Device     *
COMMENT *                                                                    *
COMMENT * PURPOSE : Provide A Symbolic Alias For A GPIB Device               *
COMMENT *                                                                    *
COMMENT * OPTIONS : devname is a name consisting of up to eight characters.*
COMMENT *             The first character must not be a digit.               *
COMMENT *                                                                    *
COMMENT *           addr = the device's GPIB address in the form xxyy        *
COMMENT *                  where :                                           *
COMMENT *                                                                    *
COMMENT *             xx is in the range 00 - 30                             *
COMMENT *             yy is in the range 00 - 31 and is optional             *
COMMENT *                                                                    *
COMMENT *                                                                    *
COMMENT * SPECIAL : Named Devices Are Assigned The Current Default           *
COMMENT *           Terminators As They Exist When CONFIG /N Is Encountered*
COMMENT *                                                                    *
COMMENT **********************************************************************
COMMENT
```

INDEX OF ERROR MESSAGES

ERROR NO.	ERROR TEXT	DESCRIPTION
00	OK	No error has occurred.
01	TIME OUT - NOT ADDRESSED TO LISTEN	The KM-488-DD, acting as a Peripheral, did not receive data within the TIME OUT period.
02	AUTOINITIALIZE MODE NOT ALLOWED	DMA with AUTOINITIALIZE is not allowed if the buffer crosses a 64 KBYTE page boundary.
03	NOT USED	
04	TIME OUT ERROR ON DATA READ	Expected bus data was not received within the TIME OUT period.
05	NOT USED	
06	INVALID CHANNEL FOR DMA	DMA transfer requested but the KM-488-DD was not configured for DMA.
07	TIME OUT ON DMA TRANSFER	Expected bus transfer did not occur during TIME OUT period.
08	TIME OUT - NOT ADDRESSED TO TALK	The KM-488-DD, acting as a Peripheral, could not output data within the TIME OUT period.
09	TIME OUT OR BUS ERROR ON WRITE	Error occurred transferring a data byte to a bus device.
10	SEQUENCE - NO DATA AVAILABLE	Application program requested data from the KM-488-DD; but no data was available.

ERROR NO.	ERROR TEXT	DESCRIPTION
11	SEQUENCE - DATA HAS NOT BEEN READ	Application program attempted to write data or commands to the KM-488-DD without reading back a response to the previous command.
12	SYSTEM ERROR - ON PEN INTS ALREADY ON	Internal System Error. Report to Factory. See Section 12.2.
13	SYSTEM ERROR - INVALID ON PEN INIT	Internal System Error. Report to Factory.See Section 12.2.
14	SYSTEM ERROR - LIKELY MEMORY CORRUPTION	Internal System Error. Report to Factory. See Section 12.2.
15	SYSTEM ERROR - ON PEN INTS ALREADY OFF	Internal System Error. Report to Factory. See Section 12.2.
16	BOARD DOES NOT RESPOND AT SPECIFIED ADDRESS	The KM-488-DD can't communicate with the IEEE interface board. Make sure that the board address configuration and that the software installation parameters are correct.
17	TIME OUT ON COMMAND (MTA)	My Talk Address could not be sent within the TIME OUT period.
18	TIME OUT ON COMMAND (MLA)	My Listen Address could not be sent within the TIME OUT period.
19	TIME OUT ON COMMAND (LAG)	Listen Address could not be sent within the TIME OUT period.
20	TIME OUT ON COMMAND (TAG)	Talk Address could not be sent within the TIME OUT period.

ERROR NO.	ERROR TEXT	DESCRIPTION
21	`TIME OUT ON COMMAND (UNL)`	Unlisten could not be sent within the TIME OUT period.
22	`TIME OUT ON COMMAND (UNT)`	Untalk could not be sent within the TIME OUT period.
23	`ONLY AVAILABLE TO SYSTEM CON-TROLLER`	The KM-488-DD must be a System Controller to execute this command.
24	`RESPONSE MUST BE 0 THROUGH 15`	The response parameter has a range of 0 to 15.
25	`NOT A PERIPHERAL`	The KM-488-DD must be a peripheral.
26	`SYSTEM ERROR - TIME INTS ALREADY ON`	Internal system error. Report to factory. See Section 12.2.
27	`SYSTEM ERROR - INVALID TIMER INIT`	Internal system error. Report to factory. See Section 12.2.
28	`SYSTEM ERROR - TIMER INTS ALREADY OFF`	Internal system error. Report to factory. See Section 12.2.
29	`ADDRESS REQUIRED`	Command/Call requires an address.
30	`TIME OUT VALUE MUST BE FROM 0 TO 65535`	The TIME OUT period must be within the specified range.
31	`MUST BE ADDRESSED TO TALK`	The KM-488-DD must first be addressed to talk (via MTA).

ERROR NO.	ERROR TEXT	DESCRIPTION
32	VALUE MUST BE BETWEEN 0 AND 255	Data bytes must be 8-bit integers.
33	INVALID BASE ADDRESS	I/O Port Base Addresses must be hexadecimal values which end in 0, 1, or 8.
34	INVALID BUS ADDRESS	GPIB Bus Addresses must be in the range 0 to 30.
35	BAD DMA CHAN NO. OR DMA NOT ENABLED	DMA requested, but an invalid DMA channel is assigned.
36	NOT AVAILABLE TO A PERIPHERAL	A command which cannot be used in the Peripheral mode has been sent.
37	INVALID PRIMARY ADDRESS	GPIB primary bus addresses must be in the range of 0 to 30.
38	INVALID SECONDARY ADDRESS	GPIB secondary bus addresses must be in the range of 0 to 31.
39	INVALID - TRANSFER OF ZERO BYTES	A count of zero bytes is not valid.
40	NOT ADDRESSED TO LISTEN	The KM-488-DD has not been addressed to listen properly.
41	COMMAND SYNTAX ERROR	The command syntax given is incorrect.
42	UNABLE TO CHANGE MODE AFTER BOOTUP	The KM-488-DD can only be configured as a System Controller or non-System Controller at initial start-up. This must be done during the installation process.

ERROR NO.	ERROR TEXT	DESCRIPTION
43	TIME OUT WAITING FOR ATTENTION	When in the peripheral mode, the KM-488-DD did not sense that ATN was unasserted within the given timeout period.
47	VALUE MUST BE BETWEEN 1 AND 8	The KM-488-DD clock source frequency must be between 1 and 8.
48	TIME OUT - CONTROL NOT ACCEPTED	No device took control of the GPIB bus after a PASS CONTROL.
49	UNABLE TO ADDRESS SELF TO TALK OR LISTEN	A TALK or LISTEN subcommand is a SEND command specified the controller's own address. Use MTA or MLA instead.
50	TIMEOUT ON COMMAND	A Time out error occurred during a SEND CMD.
51	SPECIFIED BOARD HAS NOT BEEN INITIALIZED	Check Base Address settings.
52	SEQUENCE REQUIRES ACTIVE CONTROLLER	KM-488-DD needs to be an Active Controller.
53	DATA SEQUENCE ERROR	Application program attempted to write data or commands to the KM-488-DD without reading back a response to the previous command.

CALL SUMMARY

ROUTINE †	DESCRIPTION
kabort(board_nr, err_code)	Forces the KM-488-DD to take control of the GPIB bus.
karm(board_nr, err_code, arm_code)	Enables a lightpen interrupt condition(s).
kbuf(board_nr, err_code, dir)	Defines direction of access within memory buffers.
kbufd(board_nr, err_code, bcount)	Reads the number of bytes last transferred by buffered ENTER or OUTPUT command.
kclear(board_nr, err_code, mygads)	Clears designated device(s).
kconfig(board_nr, err_code, sym_name, stringlen)	Resets KM-488-DD default parameters.
kdisarm(board_nr, err_code, arm_code)	Disables a lightpen interrupt condition(s).
kdiagoff()	Disables kdiagon.
kdiagon(filename)	Writes diagnostic information to a disk file
kdma (board_nr, err_code, dma_mode)	Define DMA transfer mode.
kenter(board_nr, err_code, data_ptr, count, mode, mygads, term_list)	Read data from a bus device.
kerror(board_nr, err_code, e_sw)	Enable/Disable error message display.
kfill(board_nr, err_code, fill_sw)	Defines response if no data is available.
khello(board_nr, err_code, id_response)	Reads KM-488-DD identification string.
klocal(board_nr, err_code, mygads)	Allows designated device(s) to be locally programmed.
klol(board_nr, err_code)	Disables the local button on a GPIB device.
koutput(board_nr, err_code, data_ptr, count, mode, mygads, term_ptr)	Transmits data to the designated device(s).
kpasctl(board_nr, err_code, mygads)	Assign another device as the Active Controller.
kppoll(board_nr, err_code, pprtn)	Read the Parallel Poll response from all bus devices.
kppc(board_nr, err_code, mygads, ppc_cfg)	Structure the Parallel Poll response.
kppd(board_nr, err_code, mygads)	Disables Parallal Polling for designated device(s).
kppu(board_nr, err_code)	Disables Parallel Polling for <u>all</u> devices.
kquikstat(board_nr, err_code, qstat)	Returns the KM-488-DD's status word.
kremote(board_nr, err_code, mygads)	Puts device(s) into remote state and optionally addresses them to listen.

†There are other routines available which can only be called from BASICA or QuickBASIC programs. See Chapters 7 and 8 for more information.

ROUTINE	DESCRIPTION
krequest(board_nr, err_code, sp)	Defines own Serial Poll Reponse.
kreset(board_nr, err_code)	Resets KM-488-DD to its default parameters.
kresume(board_nr, err_code)	Unasserts ATN. Allows Peripheral to Peripheral transfers.
ksend(board_nr, err_code, sendstr, stringlen)	Sends low-level bus commands and data.
kspoll(board_nr, err_code, spr, mygads)	Reads response(s) to serial poll.
kstatus(board_nr, err_code, stat)	Returns KM-488-DD status string.
kstop(board_nr, err_code)	Stops DMA transfers.
kterm(board_nr, err_code, trm_list)	Sets the GPIB bus terminators.
kto(board_nr, err_code, tval)	Sets timeout elapse.
ktrigger(board_nr, err_code, mygads)	Triggers device(s).
kwaitc(board_nr, err_code)	Waits for the current DMA continue transfer to complete.

FILE I/O COMMAND SUMMARY

COMMAND	DESCRIPTION
ABORT	Forces the KM-488-DD to take control of the GPIB bus.
ARM *interrupt*	Enables a lightpen interrupt condition(s).
BUFFER *direction*	Defines direction of access within memory buffers.
BUFFERED	Reads the number of bytes last transferred by buffered ENTER or OUTPUT command.
CLEAR *addr*	Clears designated device(s).
CONFIG *options*	Resets the KM-488-DD to its defaults.
DISARM *interrupt*	Disables a lightpen interrupt condition(s).
DMA *mode autoinitialize*	Define DMA transfer mode.
ENTER *addr; [#count term1 term2 EOI EOI]*	Read data from a bus device.
ENTER *addr* **; #count BUFFER buf_addr** *DMA CONTINUE term EOI*	Reads data from a bus device into a buffer.
EOL *dir term*	Sets the language EOL terminators.
ERROR *x*	Enable/Disable error message display.
FILL *x*	Defines response if no data is available.
HELLO	Reads KM-488-DD identification string.
LOCAL *addr*	Allows designated device(s) to be locally programmed.
LOCAL LOCKOUT	Disables the local button on a GPIB device.
OUTPUT *addr #count;***data**	Transmits data to the designated device(s).
OUTPUT *addr* **#count BUFFER buf-addr** *[DMA [CONTINUE]] EOI*	Transmits contents of a memory buffer to the designated device(s).
PASS CONTROL *addr*	Assign another device as the Active Controller.
PPOLL	Read the Parallel Poll response from all bus devices.
PPOLL CONFIG addr;response	Structure the Parallel Poll response string.
PPOLL DISABLE *addr*	Disables Parallal Polling for designated device(s).
PPOLL UNCONFIG	Disables Parallel Polling for <u>all</u> devices.
REMOTE *addr*	Puts device(s) into remote state and optionally addresses them to listen.
REQUEST *;* **response**	Defines own Serial Poll Reponse.
RESET	Resets KM-488-DD to its default parameters.
RESUME	Unasserts ATN. Allows Peripheral to Peripheral transfers.
SEND; subcommand	Sends low-level bus commands.

COMMAND	DESCRIPTION
SPOLL *addr*	Reads response(s) to serial poll.
STATUS	Returns KM-488-DD status string.
STOP	Stops DMA transfers.
TERM; *dir term1 term2 EOI*	Sets the GPIB bus terminators.
TIME OUT *n.nn*	Sets timeout elapse.
TRIGGER *addr*	Triggers device(s).
WAIT	Waits for the current DMA continue transfer to complete.

GPIB SYSTEM CONFIGURATION WORKSHEET

KM-488-DDs

KM-488-DD #	Base Address	GPIB Primary Address	Lightpen Interrupt Enabled?	Device Timeout	DMA Level	Mode	CPU Clock
DEFAULTS	02E1 hex	00 dec	Disabled	10.0 sec	3	Controller	8
0							
1							
2							
3							

TERMINATOR DEFAULTS

Terminator	EOL		GPIB		
	1ST	2ND	1ST	2ND	EOI
DEFAULTS (Input and Output)	CR	LF	CR	LF	ON
Input					
Output					

DEVICES

Device Name	GPIB Primary Address	GPIB Second-ary Address	GPIB Input Terminator			GPIB Output Terminator		
			1ST	2ND	EOI	1ST	2ND	EOI