KM-488-ROM

FCC Class B Compliance

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

User Guide

for the

KM-488-ROM

IEEE-488 Interface

Board

Revision A - March 1991

Part Number: 24408

KEITHLEY DATA ACQUISITION - Keithley MetraByte/Asyst

440 Myles Standish Blvd., Taunton, MA 02780
TEL. 508/880-3000, FAX 508/880-0179

Warranty Information

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

Warning

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

Disclaimer

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

Note:

Keithley MetraByte™ is a trademark of Keithley Instruments.

Basic™ is a trademark of Dartmouth College.

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

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

Microsoft® is a registered trademark of Microsoft Corporation.

Turbo C® is a registered trademark of Borland International.

Contents

Contents

■ ■ ■

Chapter 1

INTRODUCTION

1.1 OVERVIEW

The KM-488-ROM is an IEEE-488 interface board that allows programs written on IBM PC/XT/ATs, IBM PS2 25/30s, or compatibles to communicate with an IEEE-488 bus. This Board complies with the 1978 IEEE-488 standard and is thus compatible with other IEEE-488 products. Up to fourteen other devices may be connected to the IEEE-488 bus, including instruments, printers, and other computers. The KM-488-ROM comprises a board, software, and documentation.

Figure 1-1 is a block diagram of the KM-488-ROM board.

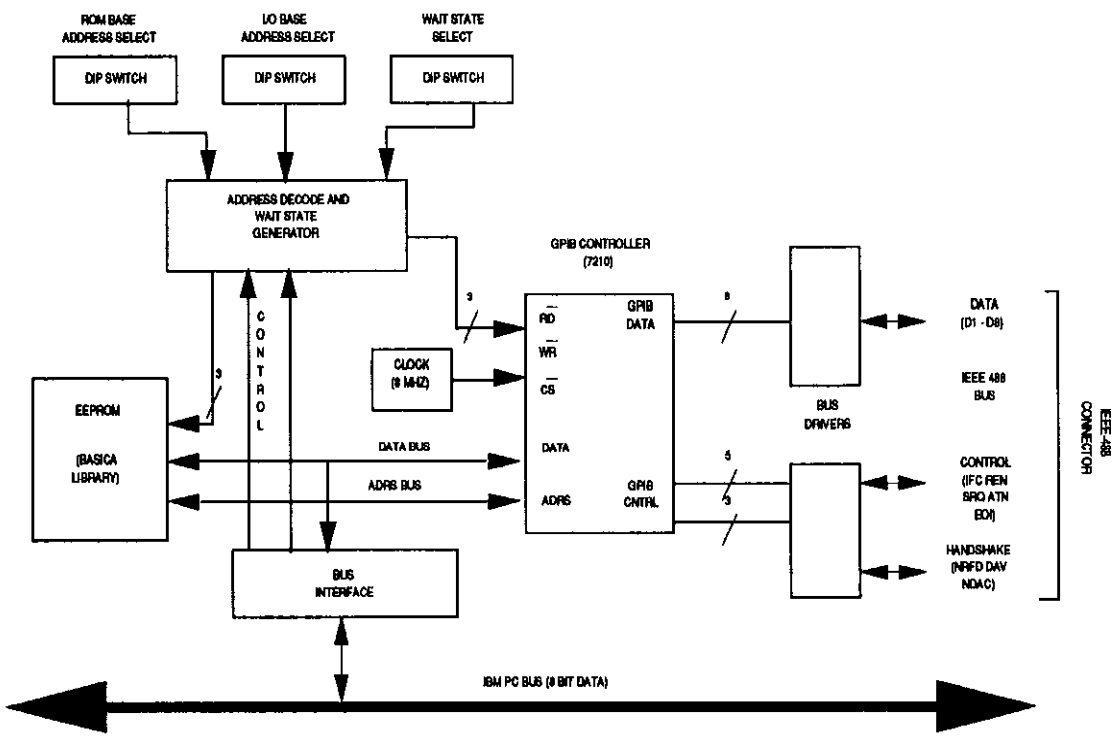


Figure 1-1. KM-488-ROM Block Diagram

The KM-488-ROM design includes a Wait State Generator to adjust bus timing, allowing performance within operating specifications of the GPIB controller chip on the fastest PCs. This Board can also generate programmed interrupts on any of six interrupt request lines and DMA transfers on Channels 1, 2, and 3. Selection of message terminators and timeouts is modifiable to allow communication with GPIB devices using non-standard characters and timeouts.

The KM-488-ROM also features an 8-KB EEPROM (Electrically Erasable Programmable Read Only Memory) containing firmware routines callable from a BASICA program. These routines perform the IEEE-488 transfer functions. KM-488-ROM software libraries allow access to routines from programs in QuickBASIC, Microsoft C, and TURBO PASCAL. Examples for each language are included.

1.2 SPECIFICATIONS

Dimensions:	One Short PC Slot size.
DMA Level:	Channels 1, 2,3, or None (Jumper Selectable).
Interrupt (IRQ) Capability:	Levels 2 through 7 or None (Jumper Selectable).
Data Transfer Rate (Governed by the slowest device):	> 300 Kb per second.
IEEE Controller Chip:	NEC7210.
Power Consumption:	< 500 mAmps.
Operating Temperature:	0 to 50 °C.
Storage Temperature:	-4 to 158 °F (-20 to +70 °C).
Humidity:	0 to 90% noncondensing.
Wait States:	1, 2, 3, or 4 (Switch Selectable).
Net Weight:	.31 lb (.14 kg).
ROM Base Address:	Switch Selectable.
I/O Base Address:	Switch Selectable.
Device Interface Capabilities Supported:	SH1, AH1, T5, TE5, L3, LE3, SR1, RL1, PP1, PP2, DC1, DT1, C1-5, E1/2. (See Appendix D for clarification.)

1.3 ORDERING INFORMATION

PART NUMBER	DESCRIPTION
KM-488-ROM	Includes the KM-488-ROM IEEE-488 Interface Board, Software (on 5.25" disks), and appropriate documentation.
KM-488-ROM/3.5	Includes the KM-488-ROM IEEE-488 Interface Board, Software (on 3.5" disks), and appropriate documentation.
CGPIB-1	1 meter IEEE-488 cable.
CGPIB-2	2 meter IEEE-488 cable.
CGPIB-4	4 meter IEEE-488 cable.

1.4 HOW TO USE THIS MANUAL

This manual provides the information necessary to install and program the KM-488-ROM. The manual assumes you are familiar with the language in which you are developing your application program; it also assumes you are familiar with the IEEE-488 protocol.

Chapter 2, *Installation*, details how to unpack, inspect, configure, and install the KM-488-ROM and how to copy the accompanying software. Additionally, Chapter 2 describes how to install the KM-488-ROM software and to configure the EEPROM and reload EEPROM software. There are also notes on using multiple boards in one system.

Chapter 3, *Introduction to the Callable Routines*, provides a brief functional description of each KM-488-ROM Interface Routine.

Chapter 4, *Programming the KM-488-ROM*, provides a detailed description of each KM-488-ROM Interface Routine and how it is called from each of the supported languages: BASICA, QuickBASIC, C, and TURBO PASCAL.

Chapter 5, *Factory Returns*, gives instructions for returning the board to the factory.

The appendices contain additional useful information. Appendix A contains an ASCII Equivalence Chart. This gives hex and decimal equivalents for the ASCII 128 Character set. Appendix B is an IEEE-488 tutorial. Appendix C provides an explanation of the Device Capability Identification codes. Appendix D provides a cross-reference chart of IEEE Multiline Commands. Appendix E describes how to use the KM-488-DD Printer Port Re-director.

■ ■ ■

Chapter 2

INSTALLATION

2.1 GENERAL

Installation begins with procedures for unpacking and inspection followed by recommendations and instructions for software. Next is a section on switch and jumper settings. Board installation is the next step, followed by EEPROM configuration.

2.2 UNPACKING & INSPECTING

After removing the wrapped Board from its outer shipping carton, proceed as follows:

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

2.3 SOFTWARE INSTALLATION

Backing Up The Distribution Software

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

Installing The Distribution Software

Install the KM-488-ROM Distribution Software on your computer's hard drive using the DOS *COPY* command.

NOTE: If you are using BASICA and the factory default settings, you may run the KM-488-ROM board without installing any software. Instead, proceed to Section 2.4.

To install the software:

1. Turn on your PC and its display. You should see the standard DOS-level prompt.

NOTE: If you install example programs written in multiple languages, you may want to create a directory for each language. (This is the way the Distribution Software is organized.)

2. The following instructions create a directory named *KM488R*. Type **md \KM488R**
3. Change to the KM488R directory by typing **cd \KM488R**
4. Place a KM-488-ROM Diskette into the floppy drive (assume this is Drive a:) and type **copy a:*.***

Repeat this step for each disk and/or subdirectory, until copying is complete.

Distribution Software Contents

Your Distribution Software contains the file *FILES.DOC*, an ASCII text file readable with any text editor or with the DOS *TYPE* command. FILES.DOC lists and briefly describes all files in the Distribution Software.

The README.DOC File

To learn of last-minute changes, be sure to read the ASCII file *README.DOC*.

2.4 SWITCHES & JUMPERS

Factory Settings

The KM-488-ROM contains three DIP switches and two jumper banks (see Figure 2-1). These switches and jumpers are factory-configured to work with most PC configurations. Table 2-1 lists the factory selections.

Table 2-1. Factory Switch & Jumper Settings

SWITCH/JUMPER	FACTORY SETTING
I/O Base Address:	2b8h.
ROM Base Address:	CC00h ROM Enabled.
I/O Wait State:	1 Wait State; System Controller Enabled; EEPROM Write Disabled.
Interrupt (IRQ) Level:	Disabled.
DMA Level:	Disabled.

For assistance with setting the switches or the jumpers, run the INSTALL program. This program illustrates the correct switch settings for your selections. To run the INSTALL program, make sure you are in the appropriate directory and type **INSTALL** at the DOS prompt. Then, follow program directions.

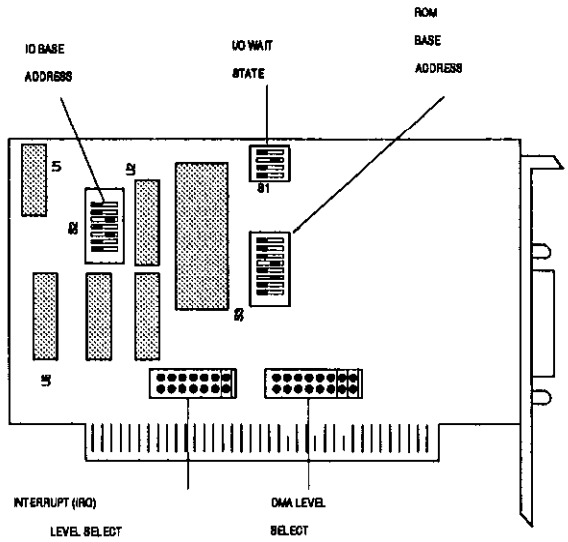


Figure 2-1. Switch and Jumper Locations

Switches

There are three DIP switch blocks on the KM-488-ROM board, as follows: Wait State (S1), I/O Base Address (S2), and ROM Base Address (S3). The switches are factory-set to work with most PC configurations (see Table 2-1 for settings).

NOTE: If you are using BASICA and change the I/O Base Address DIP switch settings, be sure to run the configuration program, CONFIG. See Section 2.7.

I/O Base Address Switch

Setting an I/O Base Address enables the KM-488-ROM to communicate with the PC. You set an I/O Base Address for the Board by setting the seven positions of Switch S2 for the assigned address. Setting a switch position to ON puts the corresponding address line at a logic 0 (low).

The KM-488-ROM requires a series of 8 I/O port addresses that begin with the I/O Base Address. Therefore, be sure to select an I/O Base Address on an 8-byte boundary that does not conflict with other devices in your computer (refer to your PC manual for the I/O address list to determine available spaces).

Figure 2-2 shows examples of I/O Base Address settings. Note that the factory-set Base Address is 2B8 hex; the I/O ports occupy the address range 2B8 - 2Bf Hex.

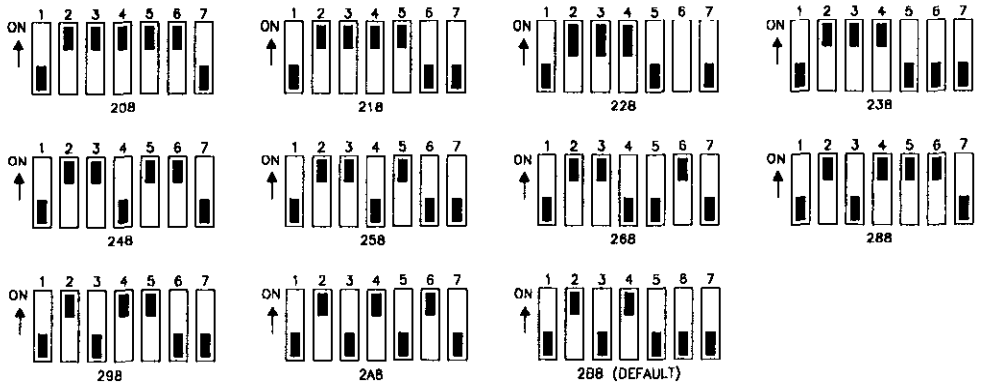


Figure 2-2. Examples of I/O Base Address Settings

ROM Base Address Switch

This switch determines whether the ROM memory is to be enabled and, if so, where within the first 1 MB of PC memory it is to be located. Enable the ROM if you are programming in BASICA. The ROM Base Address Switch (S3) is an 8-position DIP switch.

Seven of the S3 positions (1 - 7) to select the ROM Base Address. Position 8 enables/disables the ROM. Setting a position at ON puts the corresponding address line to a logic 0.

To enable or disable the ROM, set S3 Position 8 as shown in Figure 2-3. This position should be ON only if the KM-488-ROM is used with BASICA software.

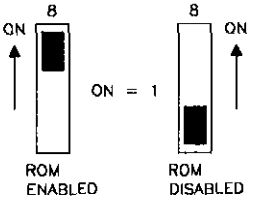


Figure 2-3. Enabling the ROM

Some alternative ROM Base Address switch settings are shown in Figure 2-4. The default Base Address is CC00 hex. Be sure to select an 8 KB address space that is within the first 1 MB of PC memory and not occupied.

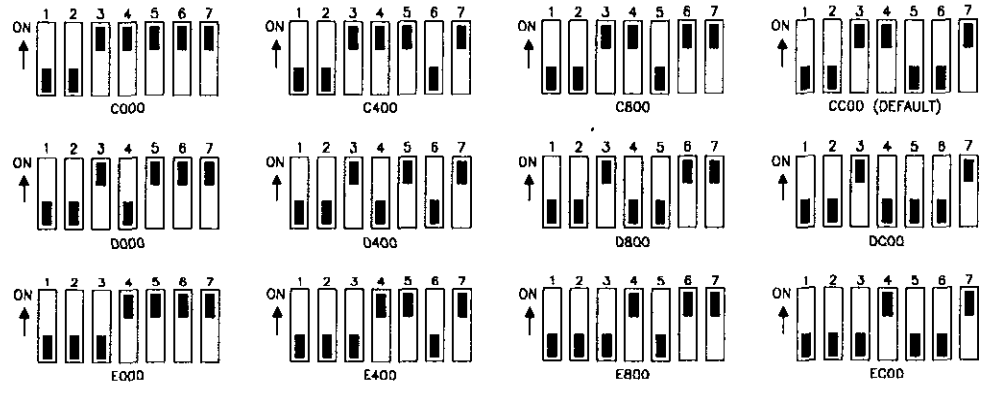


Figure 2-4. ROM Base Address Selection

If you are unsure which address to assign to the EEPROM, use the MEMMAP program provided with the KM-488-ROM. This program scans your computer's memory and determines what memory areas are available. To invoke the MEMMAP program, switch to the appropriate directory and type **MEMMAP** . Choose an unoccupied address space.

Wait State Switch

Switch 1 (S1) configures Wait States and the System Controller Mode, and it enables Memory Write Protection. S1 is a 4-position DIP switch (see Figure 2-5). Setting a position to ON puts the corresponding address line at signal low (logical 0). Two positions (1 and 2) select the wait states.

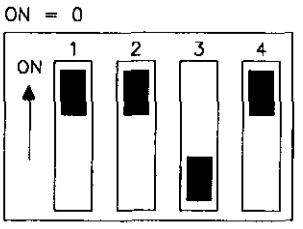


Figure 2-5. Wait State Switch.

Configure the System Controller function using Position 3 and the EEPROM protection using Position 4.

I/O Wait States

The KM-488-ROM design includes a switch-selectable wait-state generator. A selectable Wait State insures optimum performance and reliable operation at the differing bus clocks found in PCs. The default number of Wait States (1) should be correct for most PCs. However, if your data is garbled or your program crashes, you may need to adjust the number of Wait States. Some general guidelines are presented in Table 2-2. Select the number of Wait States by setting Positions 1 and 2 (marked Wait State) on the DIP switch. You may program the KM-488-ROM to generate one, two, three, or four Wait States during I/O. Note that the number of memory Wait States is automatically set to a value which is one less than the I/O Wait States. To select a number other than the default, set the switches to one of the positions shown in Figure 2-6.

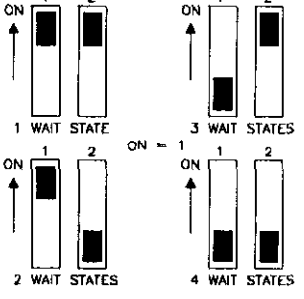


Figure 2-6. I/O Wait State Selections

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.

System Controller

This switch determines whether or not the KM-488-ROM will act as a System Controller. If the KM-488-ROM is a System Controller, it has the ability to assert the IFC or REN lines.

Position 3 on the Wait State DIP Switch determines whether the KM-488-ROM is acting as a Device/Controller or a System Controller. Valid selections are shown in Figure 2-7.

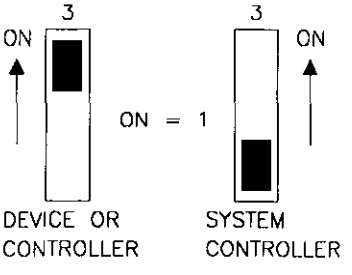


Figure 2-7. Device Mode Selection

Memory Write Enable

Positione 4 on the Wait State DIP Switch enables or disables writes to the EEPROM on the KM-488-ROM. Valid selections are shown in Figure 2-8.

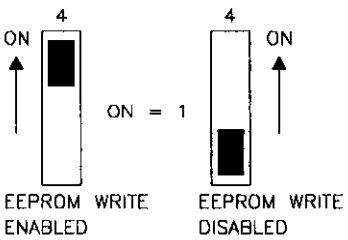


Figure 2-8. EEPROM Enable Selection

This switch should normally be at DISABLE. It should be at ENABLED only when initializing or configuring the EEPROM BASICA support software.

Jumpers

The KM-488-ROM contains two jumper blocks. These blocks select the Interrupt Level and DMA Level.

Selecting an Interrupt Level

The KM-488-ROM is capable of interrupting the PC. The Interrupt Level (IRQ) Jumper (J1) defines the Interrupt Level. Valid Interrupt Level selections (2 through 7 and none) and the jumper positions are shown in Figure 2-9.

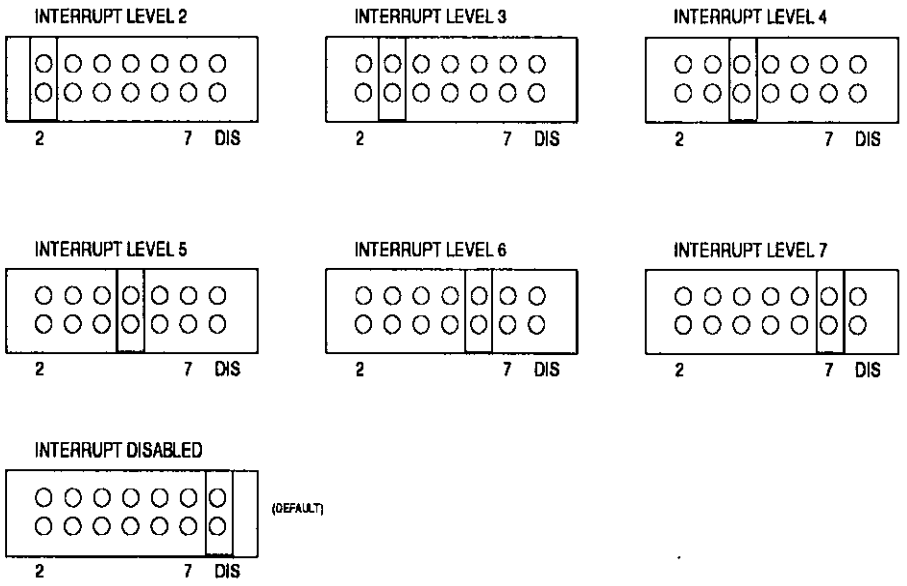


Figure 2-9. Interrupt Level (IRQ) Jumpers

Selecting a DMA Level

DMA (Direct Memory Access) is a PC facility for speeding up data transfer from a peripheral to the computer. Select an appropriate DMA level using the DMA Level Jumpers. Refer to

Figure 2-10 for jumper positions.

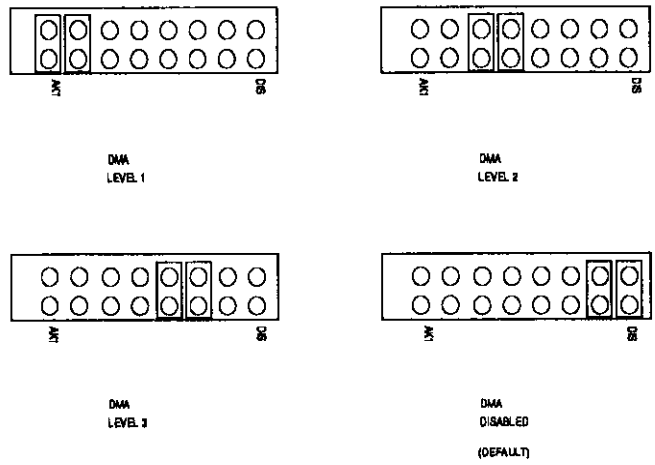


Figure 2-10. DMA Level Jumpers

2.5 BOARD INSTALLATION

To install the KM-488-ROM in a PC, proceed as follows:

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

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

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

If you are developing KM-488-ROM application programs in C, TURBO PASCAL or QuickBASIC, the installation process is now complete. However, if you are developing programs in BASICA and have changed the factory default settings, you must to run the EEPROM configuration program *CONFIG*.

2.6 CONFIGURATION OF THE EEPROM

When KM-488-ROM application programs use BASICA, the programs read interface functions directly from the on-board EEPROM. Thus, the EEPROM must be properly configured, which may be accomplished using the CONFIG program. This program allows you to change such parameters of the EEPROM configuration as I/O Base Address, I/O Timeout, DMA Timeout, and Transmit/Receive Terminators.

Before changing the EEPROM configuration, you may want to read the descriptions of the DMA, RCV, and XMIT routines in Chapter 3. Also make sure that the ROM Base Address switch has the ROM Write function enabled. (See Section 2.4.)

Invoking The CONFIG Program

Invoke the CONFIG program as follows:

1. Install the Distribution Software (see Section 2.3) and the KM-488-ROM board (see Section 2.5).
2. Switch to the appropriate directory. At the DOS prompt, type **`CONFIG`**

The PC monitor will show a screen labelled *KM-488-ROM CONFIGURATION*. The settings will reflect any changes which were made by running the INSTALL program.

The following PC function keys are now active:

[F1] **HELP**. Invoke Help at any time by pressing **[F1]**.

[F2] **SHOW NEXT**. In multiple board systems, pressing **[F1]** shows the configuration of the next KM-488-ROM.

[F3] **LOAD**. Pressing this key loads the file KM488ROM.BIN to the EEPROM. This function is useful when you want to load the factory defaults back into the KM-488-ROM's EEPROM.

[Shift][F3] **LOAD NEW MEMORY**. Pressing this key combination allows you to load the contents of the KM-488-ROM'S EEPROM to a new segment of DOS memory. The value you enter here must agree with the address selected by the ROM Base Address Switch. If you have trouble identifying an unoccupied space, run the MEMMAP program (see Section 2.4).

[Alt][F3] **EDIT I/O ADDRESS**. This key combination permits you to edit the I/O Address field only. This is the address for access to the KM-488-ROM. It is important that you select an I/O Base Address on an 8-byte boundary that does not conflict with other devices in your computer. The I/O Base Address must fall within the range 200h to 3F8h.

[F8] **EDIT**. This key allows editing of the configuration parameters (see the next section for parameter descriptions). When editing is complete, press **[F10]**. When the prompt *Save changes to KM-488-ROM memory? Y/N* appears, enter the appropriate response.

[**F10**] **EXIT** . Pressing this key exits the editing process. Otherwise, pressing [**F10**] exits to the DOS prompt.

Once you have completed writing to the EEPROM, be sure to disable the EEPROM Write function (see Section 2.4).

NOTE: Be sure to reset the EEPROM Write Switch when you complete writing to the EEPROM. Many software programs are designed to search for free address space within the computer and may interpret the EEPROM as such.

Editing The Configuration Parameter Fields

Once you have invoked the EDIT function, you will be able to edit the configuration parameters. To exit from the EDIT function at any time, press [**F10**] . To move between fields, use [↑] and [↓] . Once you make your selection for a given parameter, press [**Enter**] . These parameters include the following:

I/O Timeout

If the time elapsed between the transfer of individual bytes exceeds the specified I/O Timeout period, an I/O Timeout Error will occur. This parameter sets the maximum amount of time (in milliseconds) which is to elapse. Enter a value between 0 and 65535 milliseconds for the I/O timeout. The default value is 10010 ms.

DMA Timeout

A DMA Timeout Error is generated when the time to transfer (via DMA) an entire message exceeds the set DMA Timeout value. Valid entries for the DMA Timeout parameter are between 0 and 65535 milliseconds. The default value is 10010 ms.

Transmit Terminators

Transmit Terminators (also referred to as Output Terminators) are appended to data sent from the KM-488-ROM to another IEEE-488 device. The terminators signal the end of the data transfer. The Transmit Terminator sequence consists of one or two ASCII characters with EOI optionally asserted, when the last terminator character is sent. Up to four different Transmit Terminator sequences may be selected.

To select a terminator sequence,

1. Referring to the ASCII Equivalence Chart in Appendix A, enter the HEX VALUE (00h - FFh) of the first terminator byte. Press [**Enter**] .

2. Repeat Step 1 for the second terminator byte. If a second terminator byte is not required, enter spaces. Press [**Enter**] .

3. Press [**Space Bar**] to enable EOI(EOI) or disable EOI (NOEOI). Press [**Enter**] .

Repeat these three steps for each of the remaining Transmit Terminator Sequences.

The default Transmit Terminator Sequences are as follows:

Terminator 0	LF EOI
Terminator 1	CR LF EOI
Terminator 2	CR EOI
Terminator 3	LF CR EOI

Receive Terminators

The KM-488-ROM uses these items (also referred to as Input Terminators.) to detect the end of a data transfer received from another device. The Receive Terminator sequence consists of one ASCII character with EOI optionally asserted. If the chosen terminator character is detected in the incoming data, reception will terminate. Note that any data byte received with EOI asserted will always terminate reception, regardless of the selected terminator.

Up to four different Receive Terminator sequences are available for selection, as follows:

Terminator 0	LF EOI
Terminator 1	CR EOI
Terminator 2	, (comma) EOI
Terminator 3	; (semi-colon) EOI

To change the terminator character, use the procedure previously outlined for Transmit Terminators.

2.7 RELOADING THE EEPROM

Under some conditions (for example, if the EEPROM contents have been destroyed), you will have to reload the EEPROM with the contents of the KM488ROM.BIN file. To perform this requirement, run the CONFIG program, as described in the previous section.

Before you reload the EEPROM, be sure its Write/Enable switch is enabled (see Section 2.4). The proceed as follows:

1. Invoke the CONFIG program. Switch to the appropriate directory and at the DOS prompt, type **CONFIG**.
2. Press [**F3**].

When you completed the EEPROM reload, be sure to disable the EEPROM Write Enable switch (see Section 2.4).

2.8 MULTIPLE BOARD INSTALLATION NOTES

The KM-488-ROM software allows installation of up to four boards in a given system. Typically, situations with excessive cable lengths or more than 14 instruments require multiple boards.

When using multiple KM-488-ROMs, set the I/O Port Base Address to a different value on each of the boards. Routines within the software library allow you to determine which board to use by specifying the Base Address of the I/O port on that board.

When using BASICA, each board requires its own copy of software. This means that you must select a different EEPROM memory address and I/O Base Address for each board. These Base Address ranges CANNOT overlap other address ranges within the system.

■ ■ ■

Chapter 3

INTRODUCTION TO CALLABLE ROUTINES

To use the KM-488-ROM within a custom data acquisition or control environment, you have to write software that will access the GPIB. The KM-488-ROM includes a number of "callable" routines allowing this access from high-level languages such as BASIC, Quick BASIC, C, and TURBO PASCAL.

This chapter describes the callable-interface routines from a functional approach. Chapter 4 provides the exact syntax for calling the routine from BASIC, Quick BASIC, C, and TURBO PASCAL. Table 3-1 provides an alphabetical listing of the available routines. The remainder of the chapter tracks the order of a routine's usage and is organized as follows:

- Initializing the KM-488-ROM.
- Selecting the Receive and Transmit Message Terminators.
- Transmitting Commands and Data.
- Reading Data.
- Transmitting/Receiving Data via DMA.
- Checking the Status of a Device.
- Low-level Routines.
- Configuring the Board.

NOTE: Explanations within this chapter assume you are familiar with IEEE-488 communications. If you are new to IEEE-488 or do not recognize some of the terminology used, refer to the IEEE-488 Tutorial in Appendix B.

Table 3-1. The Callable Routines

ROUTINE NAME	DESCRIPTION	GPIB OPERATIONS
DMA	Used to transmit/receive array data via DMA. (BASICA only)	
DMATIMEOUT [1]	Sets maximum length of time for a DMA transfer.	None.
ENTER	Addresses a device to talk and receives the talker's data into a string.	Asserts REN. Sends UNL, UNT, TALK adrs, MLA, data, UNL, UNT.
INIT	Initializes the KM-488-ROM.	If KM-488-ROM is Sys. Contr., asserts IFC.
INTERM [1]	Redefines input terminator settings.	None.
IOTIMEOUT [1]	Sets the maximum length of time for an I/O transfer.	None.
OUTTERM [1]	Redefines output terminator settings.	None.
PPOLL	Performs a parallel poll.	Asserts ATN and EOI and reads data byte.
RCV	Receives data into a string.	Receives data.
RCVA	Receives data into an array.	Receives data.
SEND	Addresses a specific device to listen and allows the current talker to send the data from a string.	Asserts REN. Issues UNL, UNT, Listen Adrs, MTA, and sends data followed by a message terminator.
SETBOARD [2]	Identifies, in a multiple board system, the board to be programmed.	None.
SETDMA [2]	Allows use of DMA in conjunction with XMITA and RCVA routines.	None.
SETINT	Allows the KM-488-ROM interrupt enable bits to be set.	None.
SETPORT [2]	Selects a non-default Base Address.	None.
SETSPOLL	Sets Serial Poll Response of the KM-488-ROM.	If RSV bit is set, will assert SRQ.
SPOLL	Conducts a serial poll on a specified device.	Asserts REN. Issues UNL UNT, Talk adrs, SPE. Receives Serial Poll Response. Issues SPD.
SRQ [2]	Detects the state of the SRQ signal on the bus.	None.
STATUS	Returns values of the various setup parameters.	None.
XMIT	Sends GPIB commands and data.	Sends GPIB commands and data as specified in string.
XMITA	Transmits data from an array.	Sends data, optionally terminates by EOI and/or terminator characters

[1] This routine is not supported in BASICA. To modify this parameter, use the CONFIG program.

[2] This call is not supported in BASICA. Its function, however, can be achieved through different means.

3.1 INITIALIZING THE KM-488-ROM

The first step in any KM-488-ROM application program is to initialize the KM-488-ROM board(s), using the INIT routine.

INIT

This routine configures the KM-488-ROM as a device or a controller. INIT also defines the GPIB address and determines whether Bus Handshaking is to be High or Low Speed. If INIT designates the KM-488-ROM as a System Controller, the Interface Clear (IFC) line on the GPIB is asserted momentarily when INIT is called.

Either High or Low Speed Handshaking is available. In High Speed mode, the KM-488-ROM asserts the GPIB bus signal DAV approximately 500 ns after data is placed onto the bus. In the low speed mode, DAV is asserted about 2 microseconds after the data. In most cases, you will see no apparent differences in data throughput with Low Speed Handshaking. To maximize data throughput when using DMA, select High Speed Handshaking.

NOTE: Use the High Speed mode only in smaller installations, because High Speed Handshake mode allows less time for data to settle. Thus, as cable lengths increase, the probability of transmission errors from cable reflections will increase.

NOTE: INIT must be the first KM-488-ROM routine called within the program.

IOTIMEOUT

This routine is not usable in BASICA. IOTIMEOUT allows you to reset the length of time that is to elapse before a Timeout Error occurs. A timeout Error occurs when the time between transmission and reception of adjacent bytes exceeds the set time. (I/O Timeout Error reports occur when using SEND, ENTER, XMITA, XMIT, and RCVA calls without DMA.) The default value of the timeout period is 10 seconds.

NOTE: The I/O Timeout may be changed at any point in the program.

3.2 SELECTING THE RECEIVE & TRANSMIT TERMINATORS

When data is transmitted to or from the KM-488-ROM, it may contain message terminator characters. These terminator characters are used to signal the end of data transmission.

Every KM-488-ROM routine that transmits or receives data contains a parameter allowing you to define which of the default terminator sequences is to be used. If your application program is in C, QuickBASIC, or Turbo PASCAL, you may change the default terminator sequences by calling the INTERM and OUTTERM routines.

If you are programming in BASICA, you may change the default Transmit/Receive Terminator sequences and the I/O Timeout period only by running the CONFIG program (see Sections 2.6 and 2.7).

INTERM

This routine does not work in BASICA. INTERM allows you to change the values of each of the four input message terminators. These terminators can be detected by the ENTER, RCV, and RCVA routines.

Each terminator sequence consists of one ASCII character (7-bit value). The default value for each terminator is shown below.

TERM #	ASCII CHARACTER	DECIMAL EQUIVALENT	HEX EQUIVALENT
0	LF (Line Feed)	10	0A
1	LF (Line Feed)	13	0D
2	, (comma)	44	2C
3	; (semi-colon)	59	3B

Note that if EOI is asserted with any data byte, data reception will be unconditionally terminated.

Instrument manufacturers frequently specify message terminators using ASCII representations. You may pass either the decimal or hexadecimal equivalents of the desired ASCII character into the INTERM routine. If using the hexadecimal value, be sure to use the correct prefix. This prefix is language-dependent. Check the language manual for more information.

OUTTERM

This routine does not work with BASICA. OUTTERM allows changes of values for each of the four output message terminator sequences. You may append these terminators to the data sent by the SEND, XMIT, and XMITA routines to signal the end of message.

Each terminator sequence consists of one or two ASCII characters (7-bit values) and may or may not assert EOI when the last terminator character is sent. The default values for each terminator appear in the following table.

	ASCII CHARACTER		DEC EQUIV		HEX EQUIV		
TERM #	1ST	2ND	1ST	2ND	1ST	2ND	EOI
0	LF		10		0A		YES
1	CR	LF	13	10	0D	0A	YES
2	CR		13		0D		YES
3	LF	CR	10	13	0A	0D	YES

Instrument manufacturers frequently specify message terminators using ASCII representations. You may pass either the decimal or hexadecimal equivalents of the desired ASCII character into the INTERM routine. For example, specify a Line Feed as 0Ah. If using the hexadecimal value, be certain to use the correct prefix; this prefix is language-dependent. Check the language manual for more information.

Terminators specified with this routine must be at least one character long. If you have an instrument or application requiring no terminator bytes (requiring assertion of EOI), use the XMIT or XMITA routine to transmit the data.

3.3 TRANSMITTING COMMANDS AND DATA

Once the GPIB system is initialized, the next step is usually to send commands and/or data to a device. Use any of the following routines to send:

- SEND
- XMIT
- XMITA
- IOTIMEOUT

SEND

Use this routine only when the KM-488-ROM is an Active Controller. SEND transfers string data from the KM-488-ROM to the device specified by first addressing the KM-488-ROM as a talker and the indicated device as a listener, and then asserting the REN line. Next, the command sends the string, followed by the selected message terminator, to the listener. The routine returns a status variable indicating whether or not the transfer is properly completed.

XMIT

The XMIT Routine allows the greatest amount of flexibility for sending GPIB commands (see Section 3.4.) and data. Data and commands to be sent over the GPIB are expressed in string form and then passed into the XMIT routine. All commands within the string may be UPPER or lower case; but they must be separated by one or more spaces.

If the KM-488-ROM is acting as a Controller, the XMIT routine sends both commands and data. If executing the XMIT routine, the KM-488-ROM must

- Untalk and Unlisten all Devices.
- Assign a Listener.
- Address itself as a Talker.

If, however, the KM-488-ROM is acting as a Device, the XMIT routine can only send data. In this instance, the KM-488-ROM must be a talker before the XMIT routine can execute.

The XMIT routine will then parse the string and extract and send the commands over the bus in the specified sequence. The commands to carry out this sequence can all be within a single string and handled by a single call to the XMIT routine.

The XMIT routine returns a single status variable to indicate the state of the data transfer. XMIT will report cases of invalid syntax, invalid address, undefined commands, timeout errors, and attempts to send bus commands while not the active controller.

THE XMIT COMMANDS

Send these commands in the XMIT command's info string; they consist of rudimentary GPIB and other commands and separate by function into three categories, as follows:

1. Data Transmission.
2. Polling.
3. Miscellaneous.

DATA Use this command after the KM-488-ROM has been addressed to talk. (If the KM-488-ROM is controller, issue an MTA. Otherwise, the Controller must address the KM-488-ROM. See the STATUS routine description for more information.) DATA sends the message that trails it to all previously addressed listeners.

Data may be in two forms. In one form, data is a string of ASCII characters that trails the DATA command. The ASCII string will be in single quotes (for example, 'BYE').

In the other form, data may be a string of numeric values, each of which ranges from 0 to 255. Each numeric value is the decimal equivalent of an ASCII character (see Appendix A for ASCII Equivalents). One or more spaces must separate each numeric entry. This form of entry is useful where transmission of nonprintable characters is required. Note that you may switch freely between the ASCII and Decimal representations after the DATA command, as long as ASCII characters are in a string enclosed by single quotes.

Example

```
DATA 'Hello' 13 10
DATA 'Line 1' 13 10 'Line 2' 13 10
```

END If END follows the DATA command string, Message Terminator 0 signals the End of Transmission. Section 3.2 describes the default values of the transmit terminators and how to change them. Set the terminators to one or two bytes, and send them with or without EOI asserted on the last byte.

Example

```
DATA 'Hello' END
```

EOI If EOI (END OR IDENTIFY) follows the DATA command string, it indicates that the character following EOI mnemonic will be sent with the EOI line asserted.

Example

```
DATA  'Hello' 13 EOI 10
```

GTL The GTL command forces bus devices addressed to listen to the *Go To Local* (front panel controllable) state, as opposed to controlled via GPIB. This command also unasserts the REN signal on the GPIB. Only the System Controller may use GTL. Note that this command DOES NOT allow you to selectively force only one device to Go To Local.

Note that it is more practical to use GTLA and LOC commands than GTL.

Example

```
GTL
```

GTLA Only a KM-488-ROM acting as a System Controller may issue this command. Use this command is used to send a Go To Local (GTL) GPIB command to those devices previously addressed to listen. This command does not affect the state of the GPIB REN line.

Example

```
GTLA
```

LISTEN The KM-488-ROM must be the Active Controller to execute this command. This command addresses a given device(s) as a listener(s). LISTEN is trailed by the decimal GPIB address (0 to 30) of the device(s) to be addressed. When assigning multiple listeners, separate the addresses by one or more spaces.

Note that it is good practice to untalk and unlisten all devices prior to sending a LISTEN command. (See the UNT and UNL descriptions.)

Example

```
LISTEN 2

LISTEN 5 9 30
```

LOC Use this command only if the KM-488-ROM is acting as the System Controller. When the LOC command is executed, it unasserts the GPIB REN (Remote Enable) line. This action forces all devices on the GPIB to the local state.

Example

```
LOC
```

MLA The KM-488-ROM must be the Active Controller to execute MLA (My Listen Address). MLA forces the KM-488-ROM to become a listener; it sends a listen address command containing the GPIB address of the KM-488-ROM over the GPIB.

Example

```
MLA
```

MTA The KM-488-ROM must be the Active Controller to execute MTA (My Talk Address). MTA makes the KM-488-ROM the present talker (and unaddresses any other talker); it sends a talk address command containing the address of the KM-488-ROM over the GPIB.

Example

```
MTA
```

REN This command can function only if the KM-488-ROM is the System Controller. The REN command asserts the REN (Remote Enable) Control line on the IEEE-488 bus. Many devices require REN to be asserted before they will accept commands or data.

Example

```
REN
```

SEC Use this command in conjunction with TALK and LISTEN to specify a secondary address. SEC must appear immediately after the primary address in a TALK or LISTEN command. The KM-488-ROM must be an Active Controller to use SEC.

Example

```
TALK 3 SEC 5
LISTEN 4 SEC 8
```

T0 If this command follows the DATA command, a Transmit Message Terminator 0 will signal the end of data transmission. Section 3.2 describes the default values of the transmit terminators and how to change them. Set the terminators to one or two bytes, and send with or without EOI asserted on the last byte.

Example

```
DATA 'Hello' T0
```

T1 If this command follows the DATA command, Transmit Message Terminator 1 will signal the end of data transmission. Section 3.2 describes the default values of the transmit terminators and how to change them. Set the terminators to one or two bytes, and send with or without EOI asserted on the last byte.

Example

```
DATA 'Hello' T1
```

T2 If this command follows the DATA command, Transmit Message Terminator 2 will signal the end of data transmission. Section 3.2 describes the default values of the transmit terminators and how to change them. Set the terminators to one or two bytes, and send with or without EOI asserted on the last byte.

Example

```
DATA 'Hello' T2
```

T3 If this command follows the DATA command, Transmit Message Terminator 3 will signal the end of data transmission. Section 3.2 describes the default values of the transmit terminators and how to change them. Set the terminators to one or two bytes, and send with or without EOI asserted on the last byte.

Example

```
DATA 'Hello' T3
```

TALK The KM-488-ROM must be the Active Controller to execute this command. TALK designates the specified device as a Talker and is followed by the decimal GPIB address (0 to 30) of the device. Remember that only one device can talk at a given time; thus, if multiple TALK commands are in a command string, only the last one takes effect. Note that it is good practice to untalk and unlisten all devices prior to sending a TALK command (see the UNT and UNL descriptions).

Example

```
TALK 1
TALK 22
```

UNL The KM-488-ROM must be the Active Controller to execute this command. UNLISTEN unaddresses the present listeners, if any.

Example

```
UNL
```

UNT The KM-488-ROM must be the Active Controller to execute this command. UNTALK is used to unaddress the present talker, if any.

Example

```
UNT
```

POLLING COMMANDS

PPC The Parallel Poll Configure (PPC) command signals a previously addressed listener that a Parallel Poll Enable (PPE) byte or Parallel Poll Disable (PPD) command is to follow. Note that not all devices support parallel polling.

PPC is rudimentary GPIB command byte and is thus sent using the CMD command (see Miscellaneous Commands). The CMD command immediately follows the PPC command; for example,

```
PPC CMD nnn
```

Where **nnn** is the decimal value of the Parallel Poll Enable byte. This byte has the following format:

```
0110SPPP
```

Where **S** is 0 or 1. The addressed device will set the designated GPIB data line (determined by PPP) to the given value if service is required. **PPP** is a 3-bit value which represents a GPIB data line (0 - 7). The configured device will use this data line to respond to a parallel poll.

Example

```
UNL LISTEN 6 MTA PPC CMD 101
```

PPD The PPD (Parallel Poll Disable) command disables parallel poll response of any previously addressed listeners. PPD must always immediately follow a PPC.

Example

```
UNL LISTEN 12 MTA PPC PPD
```

PPU The PPU (Parallel Poll Unconfigure) command disables the parallel poll response of all devices on the bus.

Example

```
PPU
```

SPD The Serial Poll Disable (SPD) command returns the currently addressed talker from the serial poll state to the "normal" talker state.

Example

```
SPD
```

SPE The Serial Poll Enable (SPE) command forces a device, previously addressed to talk, to send its serial poll response instead of its normal data.

Example

```
UNL UNT MLA TALK 20 SPE
```

MISCELLANEOUS COMMANDS

CMD CMD indicates the next byte is to be sent as a GPIB command. A GPIB command is any data byte sent in conjunction with the ATN control line asserted on the bus. The byte is must be specified in decimal format (range 0 to 255) and must follow the CMD mnemonic within the XMIT command string.

Example

```
PPC CMD 96
```

DCL The Device Clear command forces all devices attached to the GPIB (addressed or not) to a predefined state. The actual response of a device to this command is device-dependent.

Example

```
DCL
```

GET The GET (Group Execute Trigger) command synchronizes the start of a device-dependent operation in all previously addressed listeners. In many devices, GET allows the KM-488-ROM to trigger a measurement. This function is not supported by all devices.

Example

```
LISTEN 12 GET
```

IFC This command can only be issued by a KM-488-ROM which is the System Controller. The IFC (Interface clear) command resets the interface state of all devices which are tied to the GPIB. It unaddresses all devices and forces the System Controller to become the Active Controller (if control had been passed to another device).

Example

```
IFC
```

LLO The LLO (Local Lockout) command allows you to disable the front panel control of all devices that support this command. In many cases, this command works in conjunction with the GPIB REN signal. Local control may be restored with the GTLA or LOC commands.

Example

```
LLO
```

SDC This command forces those devices attached to the GPIB and addressed to listen to a predefined state. The actual response of a device to this command is device-dependent.

Example

```
SDC
```

TCT The (TCT) Take Control command allows the KM-488-ROM to pass control to another device (with controller capabilities) on the bus, and is able to receive control. The device to receive control must first be addressed to talk.

Example

```
TALK 5 TCT
```

XMITA

The XMITA routine programs the KM-488-ROM to send array data when the KM-488-ROM is a device or the Active Controller. XMITA also sends binary data; for example, data containing embedded line feeds or other control characters. Optionally, terminator characters may be used to mark the end of data or the data byte may be sent with EOI specified. XMITA allows the KM-488-ROM to send up to 64 KBytes of data from an array, and is especially useful in situations where KM-488-ROM must send large amounts of data (up to 64K).

The XMITA routine transmits data stored in adjacent bytes within the computer's memory, starting from a specified location. The data is transferred from the lowest specified memory address first, then from increasingly higher addresses until the end of the data is reached. In other words, the least significant byte of the first element of the array is the first character sent. The array may be of any data type, provided the language you are using has stored array elements of increasing index in increasing memory addresses, and the least significant byte of each location is the lowest address. The actual number of data bytes per location varies according to the type of data elements contained within the array and the language being used. Refer to a language reference manual which describes the language that you are using for exact details.

Before you call the XMITA routine, be sure to designate the KM-488-ROM as a Talker. Hint: If the KM-488-ROM is the Active Controller, call the XMIT routine with a My Talk Address (MTA) command. If the KM-488-ROM is a device, call the STATUS routine and check the state of the TA bit in the Address Status Register.

3.4 READING DATA

Once an instrument has taken a measurement, its data must be read into the computer, using any of the following routines:

- ENTER
- RCV
- RCVA

ENTER

Use this routine only if the KM-488-ROM is an Active Controller. The ENTER routine transfers data from the specified device through the KM-488-ROM to the application program. Calling ENTER addresses the KM-488-ROM as a listener, the device at the specified GPIB address as a talker, and asserts the GPIB REN line. The received data is then placed into a string specified within the ENTER call. This data string is returned with a status byte and a count variable containing the actual number of bytes received by the routine.

The ENTER routine returns to the calling program when any of the following occur:

- Calling ENTER when the KM-488-ROM is not the active controller.
- Receiving a byte (other than the specified terminator) with the EOI signal asserted.
- Receiving the specified message terminator (any one of the four default terminators may be selected).
- Filling the receive data string.
- Expiration of the timeout period.

NOTE: If programming in BASICA, you may modify the default receive terminator sequences by running the CONFIG program. Otherwise, call the INTERM routine. See Section 3.2 for defaults.

When data reception is complete, all devices are at UNTALK and UNLISTEN. Therefore, to receive strings in "pieces," avoid using ENTER.

All Carriage Returns and the receive message terminator character are stripped from the received data and are not stored within the string or included in the byte count. If the ENTER routine terminates due to reception of a data character with EOI asserted (other than the chosen receive terminator character), that character will be stored and included in the byte count.

Before you call the ENTER routine, be sure to set up a string to store the received data. Regardless of the language, you must allocate a string length greater than or equal to the number of bytes you expect to receive. Otherwise, data may be stored in areas allocated from DOS or other parts of your program, and the program will crash.

RCV

Use this routine to program the KM-488-ROM to receive data when the KM-488-ROM is a non-System Controller. RCV is useful in situations where KM-488-ROM must receive data concurrently with other listeners on the bus. The RCV routine is also useful when data must be received immediately after sending a string of commands with the XMIT command.

Received data is placed in the string named within the call. The data string is returned along with a status byte, and a variable containing the actual number of received bytes. The RCV routine stores data in a manner similar to ENTER (carriage returns and the message terminator are stripped from the received data).

The RCV routine will return to the calling program when one of the following events occurs:

- Calling RCV when the KM-488-ROM is not a listener.
- Receiving the selected terminator character.

- Receiving a data byte with EOI asserted.
- Receiving the maximum number of bytes that will fit into the receive string.
- Expiration of the timeout period.

Before you call the RCV routine, be sure to designate the KM-488-ROM as a listener. Hint: If the KM-488-ROM is the Active Controller, call the XMIT routine with a My Listen Address (MLA) or LISTEN nn command. If the KM-488-ROM is a device, wait until the KM-488-ROM is addressed to listen by the Active Controller by calling the STATUS routine and checking the state of the LA bit in the Address Status Register.

Set up a string to store the received data. Regardless of the language, you must allocate a string length greater than or equal to the number of bytes you expect to receive. Otherwise, data may get stored in areas allocated from DOS or other parts of your program, and the program will crash.

RCVA

The RCVA routine is similar to the RCV Routine in that it programs the KM-488-ROM to receive data when the KM-488-ROM is a device (not the Active Controller). The principal differences are that the RCVA routine stores the received data in a specified array and all received bytes will be stored. RCVA can also receive binary data; for example, data containing embedded line feeds or other control characters.

The received data is placed into the array named within the call. The number of bytes available for storage must also be specified. A status byte and a variable containing the actual number of received bytes are also returned. The RCVA routine stores every received character, including carriage returns and message terminator characters. These characters will also be included within the byte count.

The RCVA routine will return to the calling program when one of the following events occurs:

- RCVA is called when the KM-488-ROM is not a listener.
- The selected terminator character was received (if terminators were enabled).
- A data byte was received with EOI asserted.
- The number of bytes specified in the COUNT parameter has been received.
- The timeout period has expired.

Before you call the RCVA routine, be sure to designate the KM-488-ROM as a listener. Hint: If the KM-488-ROM is the Active Controller, call the XMIT routine with a My Listen Address (MLA) command. If the KM-488-ROM is a device, call the STATUS routine and check the state of the LA bit in the Address Status Register. You must wait for LA before calling RCVA.

Set up an array to store the received data. The number of bytes per array location will vary according to the type of array. When the array contains more than one byte per location, storage of the received data will begin at the least significant byte of the specified array location and progress in accordance with the manner most languages store data in arrays.

Regardless of array size or the language, you must allocate data storage greater than or equal to the number of bytes specified in the count variable. Otherwise, data may be stored in areas allocated from DOS or other parts of your program, and the program will crash.

Refer to the XMITA routine for a discussion of the relationship between number of array locations vs. number of data bytes.

3.5 TRANSMITTING/RECEIVING DATA VIA DMA

When using DMA, the computer transfers data directly between its memory and the KM-488-ROM, resulting in the high speed transmission or reception of up to 64 KB of data to or from an array. In contrast, when transferring data while not using DMA, the computer transfers data between its memory and the device's controller chip through registers in the microprocessor. Because the microprocessor must also execute other instructions, the rate at which it passes data far slower than when DMA is used.

The implementation of a DMA transfer is language-dependent. If you are programming in BASICA, you must call the DMA routine. Other languages initiate DMA by calling the RCVA and XMITA routines in conjunction with the SETDMA routine.

DMA

This routine works only in BASICA. The DMA (Direct Memory Access) routine permits high speed transmission or reception of up to 64K bytes of data. This data is received into/transmitted from an array. In order to use the DMA routine, the KM-488-ROM must be assigned to a DMA "channel." Each DMA channel consists of an address pointer and a pair of hardware signals. The KM-488-ROM signals its need to transfer data via the DMA request signal (DMAREQ). Other logic in the system arbitrates control of the address and data busses between the microprocessor and the DMA controller. When the busses are available, the DMA controller places the contents of the address pointer register for that channel onto the address bus and notifies the KM-488-ROM that it is ready to perform the transfer via the DMA Acknowledge signal (DMA ACK). The DMA controller then generates all the other signals required to perform the transfer, with data passing directly between the KM-488-ROM and memory.

DMATIMEOUT

This routine does not work in BASICA. DMATIMEOUT allows you to reset the length of time to elapse before a DMA Timeout Error occurs. (DMA Timeout Errors are reported when XMITA and RCVA calls are used with DMA.) The default value of the timeout period is 10 seconds.

A DMA Timeout Error occurs when the time to transmit or receive an entire message exceeds the set time. This is different from the I/O timeout, which occurs when the time between adjacent bytes exceeds the timeout value. Note that it may be better to set the I/O timeout period to a shorter length than the DMA timeout period.

NOTE: In BASICA, the DMA Timeout period is changed using the CONFIG program. See Chapter 2.

SETDMA

This routine does not work in BASICA. The SETDMA routine, in conjunction with the XMITA and RCVA routines, initiates a DMA data transfer.

To perform a DMA transfer, the KM-488-ROM must be assigned to a DMA "channel." Each DMA channel consists of an address pointer and a pair of hardware signals. The KM-488-ROM signals its need to transfer data via the DMA request signal (DMAREQ). Other logic in the system arbitrates control of the address and data busses between the microprocessor and the DMA controller. When the busses are available, the DMA controller places the contents of the address pointer register for that channel onto the address bus and notifies the KM-488-ROM that it is ready to perform the transfer via the DMA Acknowledge signal (DMA ACK). The DMA controller then generates all of the other signals required to perform the transfer, with data passing directly between the KM-488-ROM and memory.

The SETDMA routine designates a DMA channel for data transfers. The channel you assign must agree with the setting of the DMA Level Jumpers on the KM-488-ROM board (see Section 2.4). To initiate a DMA transfer,

- Call SETDMA with the appropriate channel number to enable DMA transfer.
- Call XMITA/RCVA.
- Call SETDMA with a channel number other than 1, 2, and 3 to disable DMA transfers.

3.6 CHECKING DEVICE STATUS

Generally, GPIB devices indicate whether or not they need servicing by means of serial polling and/or parallel polling. Often, serial polling and parallel polling are used together to determine the type of service needed by a device. This section describes those routines associated with serial and parallel polling. They include

- SRQ
- SPOLL
- PPOLL
- SETSPOLL

NOTE: The SRQ routine does not work in BASICA. When programming in BASICA, use the STATUS routine to check the state of the SRQ signal.

SRQ

This routine does not work in BASICA. SRQ detects the state of the SRQ signal on the GPIB bus. When this routine returns a 1, it indicates that the SRQ line has been asserted. When the routine returns a 0, it indicates that the SRQ line remains unasserted.

SRQ response can be fed into a conditional statement within your program. For example, normally you would want to conduct a serial poll only when the SRQ line has been asserted. In this case, you could call the SRQ function and then feed its result into a conditional which would call an SPOLL if SRQ had been asserted.

NOTE: Once you have obtained a TRUE response from the SRQ function, the SRQ response will reset to FALSE -- even if the SRQ line is still active. In order to reset the SRQ response to TRUE, you must serial poll at least one device requesting service. This action will reset the device's SRQ line. At this time, if other devices were asserting SRQ, the output of the SRQ function would again reset to TRUE. Otherwise, the SRQ function would become TRUE on the next assertion of the SRQ line.

SPOLL

The SPOLL routine allows the Active Controller to check the state of the devices tied to the bus. Devices may be polled "at will" or in response to the Service Request line (SRQ) being asserted on the GPIB. Calling SPOLL will return the serial poll response byte from the addressed device.

The SPOLL routine does the following:

- Addresses the specified device to talk.
- Enables the specified device to send its serial poll response byte.
- Receive the device's serial poll response byte.
- Disables the serial poll.
- Untalks the specified device.

PPOLL

Use this routine only when the KM-488-ROM is an Active Controller. Calling this routine initiates a GPIB parallel poll. The parallel poll, like the serial poll, is a mechanism allowing the active controller to determine which device(s) need service. The parallel poll allows you to quickly check the state of up to eight (groups of) devices simultaneously.

Before a parallel poll can be issued, each device to be polled must be assigned to a GPIB Bus Data Line (D0 - D7). This is the device's response mechanism. If the device requires service when the Parallel Poll command is issued, it will assert its designated bit within the data bus. The assigned bit and its asserted value (0 or 1) must be preconfigured. This is accomplished via a set of GPIB commands sent to the device over the bus.

To configure a device for Parallel Polling,

- Address the device to listen.
- Issue a GPIB Parallel Poll Configure (PPC) command accompanied by a command byte to the device. (Hint: Use the KM-488-ROM's XMIT command.)

Once configured, the device will retain its parallel poll configuration until it is powered down (or reset by other hardware means), or until unconfigured by a GPIB Parallel Poll Unconfigure (PPU) or Parallel Poll Disable (PPD) commands.

A parallel poll is limiting in that it can determine only that a device(s) requires service. It cannot identify the specific conditions requiring service. In order to identify the condition(s), the KM-488-ROM must then perform a serial poll of each device requiring service (use the KM-488-ROM SPOLL command). The serial poll allows you to distinguish which device(s) need service and what type of service is required.

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

SETSPOLL

This routine allows you to program the serial poll response byte of the KM-488-ROM when it is acting as a device (non-Controller). The actual usage and meaning of each bit is user-defined. Optionally, it allows you to drive the SRQ line to request service from the Active Controller.

For example, consider an application where the KM-488-ROM transfers files from a computer containing a KM-488-ROM acting as a device to a second computer containing a KM-488-ROM that is the system controller. You could define a simple protocol in which the device (KM-488-ROM) is addressed to listen, and the controller passes a string containing a filename and a command byte. The command byte might signify a file read, write, create or append operation. If the command specified a read of a filename that could not be found, the device would notify the Controller of this error condition using the SETPOLL routine. You would define one of the Serial Poll Response bits to mean "File Not Found." Then, you would call SETPOLL, with the appropriate bits set. This would immediately notify the controller of the error condition.

3.7 LOW-LEVEL ROUTINES

It is sometimes useful to be able to check the bits of the various GPIB Controller chip registers. Two routines enable you to do this, as follows:

- SETINT
- STATUS

SETINT

This routine sets the Interrupt mask bits within the GPIB controller chip. The most common reason for this is to allow the generation of interrupts upon receiving a Service Request (SRQ). Other possible reasons include using the interrupts to enable detection of other bus related events.

If the KM-488-ROM is acting as a device, SETINT can check its address status. For example, using the ADSC (Address Status Change) interrupt would alleviate constant monitoring of the state of the TA (talk addressed) and LA (listen addressed) bits in the Address Status Register.

It is important to note that when using interrupts, you must set up an interrupt service routine to handle the interrupting condition. The method for setting up such a routine is language-dependent. You must also assign the KM-488-ROM to an Interrupt Level not used by other devices within the computer. The KM-488-ROM contains an Interrupt Level selection jumper That must be set accordingly. Refer to the INSTALL program and Chapter 2 for assistance in setting the jumpers.

STATUS

The STATUS routine checks the status bits within the GPIB Interface Chip and also the state of DMA transfers. It is especially useful when the KM-488-ROM is acting as a device, rather than a controller.

This routine can also

- Examine the state of various setup parameters within the firmware. The STATUS routine obtains the value of the I/O Port Base address of the GPIB Controller Chip, the GPIB address of the Controller Chip, each of the four transmit/receive message terminators, and the timeout values used in conjunction with normal and DMA transfers. This function is particularly useful in a multiple board environment, or while developing and debugging software.
- Read the state of the Interrupt Status registers within the GPIB Interface Chip. These registers provide information for using the KM-488-ROM as either a device or the active controller. This feature may be useful in a "polling" environment (one in which software checks for certain conditions). When acting as the Controller, the STATUS routine may check the state of SRQ or, if interrupts are set up and enabled, the STATUS routine may check which conditions caused the interrupt.

When the KM-488-ROM is acting as a device, STATUS can check for reception of a Group Execute Trigger (GET) or Device Clear (DCL) command by reading Interrupt Status Register 1. Interrupt status register 2 can be checked to see if the device has been set to local lockout or remote states. When the board is the active controller, STATUS can check the SRQI bit to see if the SRQ line has been asserted.

Whenever the state of the Interrupt Status Registers is read, all "interrupt" bits within the register are reset. It is important to note this when reading Interrupt Status Register 1. The XMIT and RCV routines check the D1 and D0 bits to determine when to read or write the next data character. If you read the Interrupt Status Register 1 and the first byte of data has been received, the D1 bit will be cleared. If the RCV routine is then called, it will "hang up" waiting for the D1 bit to set.

Read the state of the Terminal Count bits for each one of three possible DMA channels, by setting the reg parameter to 3. This information is useful when using the BASICA DMA routine in the "background" mode.

3.8 BOARD CONFIGURATION ROUTINES

This section describes those routines to use for a nonstandard interface setting. For example, if you are developing application programs in a language other than BASICA and have changed the factory-default setting of the I/O Base Address switch, you must call the SETPORT routine. (In BASICA, you will have to run the CONFIG program as described in Section 2.6.)

If an application requires the installation of more than one KM-488-ROM board in a single computer, you will use the SETBOARD routine (except in BASICA). In BASICA, each board has its own software EEPROM which must be assigned to its own base address. Boards are selected by using a DEF SEG statement to point to the desired board prior to the call.

SETPORT

You will use this routine only if you have changed the default Base Address (and are not programming in BASICA). If using multiple boards within a single computer, use SETPORT to assign a "board number" to a given I/O port address.

SETBOARD

You will use this routine only if your system has multiple KM-488-ROMs. This routine identifies the board to be programmed and thus is called prior to executing a series of routines. Only the board identified with the SETBOARD routine will be affected, until another SETBOARD routine identifying another board is called. The "board numbers" are associated with the I/O Port Base Address of a given board.

3.9 MULTIPLE BOARD PROGRAMMING NOTES

In a multiple-board environment, set each board to either CONTROLLER or DEVICE mode, and assign each board any legal GPIB address (including the same GPIB address as other boards within the same computer). It is possible to assign multiple controllers within the same computer. Note, however, that you will NOT be able to communicate between two KM-488-ROM boards within the SAME computer, even if one is configured as a device and the other as a controller.

In a multiple-board environment, the message terminator settings and timeout values are GLOBAL parameters. In other words, all the KM-488-ROM boards within a computer share the values of these parameters. The IOTIMEOUT, DMATIMEOUT, INTERM, and OUTTERM routines are callable at any time, regardless of the board most recently selected, and the values that are set will affect all of the boards.

When DMA is used, it will behave in a similar manner (DMA is enabled independently of the board was selected at that time. A call to the XMITA and RCVA routines will use DMA on every board once DMA has been selected at the time DMA was enabled.

■ ■ ■

Chapter 4

PROGRAMMING IN BASICA OR GWBASIC

While Chapter 3 gives a brief overview of the routines available for programming the KM-488-ROM, this chapter gives instructions for calling the routines from BASICA and GWBASIC. The routines appear in alphabetical order and include a sample program for each.

4.1 GENERAL

The KM-488-ROM uses an EEPROM (Electrically Erasable Read Only Memory) that contains GPIB language extension for BASIC. BASIC uses the CALL statement to access those language extensions within a user program. Before any CALL statement can function, it must contain three definitions, as follows:

- The memory segment address of the KM-488-ROM library code.
- The location of the routine (offset address).
- The parameters used by the routine.

Definition of the memory segment address of the interface should appear at the start of a user program in a DEF SEG statement. This statement is followed by the memory address to which the EEPROM is mapped. The memory address is a hexadecimal value; thus it should have a **&H** prefix. The memory address must match the setting of the KM-488-ROM Memory Address Switches. Refer to Section 2.4 for more information.

When multiple KM-488-ROM boards are present in the same system, each must have its own unique segment address. You may then select which of the boards is to be accessed by executing a DEF SEG to its segment address.

BASIC requires identification of the offset address of each KM-488-ROM routine to know where to call the routine from within the ROM. The offset address represents the number of bytes the routine is offset from the DEF SEG address. Each KM-488-ROM interface routine must have a variable set to the offset for that routine. For example, the offset for the INIT routine is zero; therefore, you must include the line INIT = 0 before calling the routine.

Note that you may use any name for these routines, so long as the alternate name matches the offset of the desired function. For example, if we define INIT = 0 and INITIALIZE = 0 within a program, the statements CALL INIT and CALL INITIALIZE will execute the same function.

NOTE: You must define one segment address in every program and an offset address for each KM-488-ROM routine. The most recent DEF SEG statement must reflect the starting address of the EEPROM on the board being used.

Each KM-488-ROM Interface Routine requires certain parameters for execution. These parameters are always integer or string variables that must be defined prior to executing the CALL statement. The variable names must be enclosed in parentheses and follow the function name within the CALL statement. For example,

```
CALL INIT(ADRS%,MODE%)
```

These variables will pass values into and out of each of the call routines. 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. It assumes that the EEPROM is mapped to segment CCOO hex and the INIT routine has an offset value of 0. In this example, the variable names ADRS% and MODE% pass the values 0,0 into the INIT routine. Note that you may assign any legal BASICA to these variables. However, the variables 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 DEF SEG = &HCC00              'Assigns memory segment address
xx INIT=0 : ADRS%=0 : MODE%=0    'Gives offset of INIT routine & variable
                                   'definitions
xx CALL INIT(ADRS%,MODE%)        'uses call statement
```

Software Configuration

KM-488-ROM firmware contains a number of configuration parameters that govern the default settings of the input and output message terminator settings, message timeout periods, and I/O port addresses. If these default values are unsatisfactory, they may be changed by running the CONFIG program (see Chapter 2).

The default DMA and I/O Timeouts are 10 seconds.

The default terminators are as shown in the following table.

TERM #	OUTPUT TERMINATOR	INPUT TERMINATOR
0	LF EOI	LF
1	CR LF EOI	CR
2	CR EOI	, (comma)
3	LF CR EOI	; (semi-colon)

Programming Notes

1. In BASICA, only variable names may be passed into and out of functions.
2. Be sure to include all the parameters for the Interface Routine. The parameters must be the same data type and appear in the same order as those given. You may, 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 you specify an incorrect number or type of parameters, your program may crash.
3. Strings are limited to the BASICA maximum of 256 characters.

4. All integers are treated by the KM-488-ROM routines as unsigned values (0 to 65535). However, BASICA treats them as signed magnitudes (-32768 to +32767). When you want to express a value which is greater than or equal to 32768, you will have to express it in one of two ways, as follows:

   • 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 is available to assist you with defining CALL routine offsets. This is a BASICA source file that predefines the offsets. It can be modified to suit your needs.
6. Do not give your variables the same name as any of the KM-488-ROM routines.

4.2 DESCRIPTION FORMAT FOR ROUTINES

The format for each descriptions is as follows:

purpose ... a brief description of the routine. See Chapter 3 for more detailed descriptions.

offset ... gives the BASICA offset for each routine.

usage ... gives an example of usage for each routine and assumes the input parameters are passed in as variables. These parameters can also be passed in directly. See the General Programming Notes for more information.

alternate usage ... lists alternate usage for the routine, if any. Unless otherwise noted, the alternate usage performs exactly the same function as the usage.

parameters ... describes each of the input parameters.

returns ... describes any values returned by the routine.

notes ... lists any special programming considerations.

example ... gives a programming example using the routine.

4.3 ROUTINES

DMA

NOTE: DMA allows data transfer rates in excess of 100 kilobytes per second. However, the actual data rates are limited by the rates at which other bus-connected devices can send or receive data. These rates are governed automatically by the GPIB handshaking signals.

purpose Initiates a DMA transfer.

offset 206

DMA (cont.)

usage

```
...
xx DMA = 206
xx count% =
xx mode% =
xx stat% = 0
xx DIM DATA%(100)      'Assigns storage space for
                       'received data
xx seg% =
xx ofs%= VARPTR(DATA%(0))
xx CALL DMA(seg%,ofs%,count%,mode%,stat%)...
```

parameters

seg% is an INTEGER representing the segment portion of the memory address of the data. seg% is set to -1 to indicate the BASICA data segment.

ofs% is an INTEGER representing the offset portion of the memory address of the data. This is usually obtained using the VARPTR function. The VARPTR function must be called immediately prior to the DMA function call, and all variables used within the program must be declared prior to the VARPTR function. The reason for this is that BASICA can dynamically allocate storage space and if variables are declared after the VARPTR call, the array may be relocated and the data placed in the wrong location. This could cause your program to "crash".

count% is an INTEGER containing the maximum number of data bytes to be transmitted or received. If you wish to send or receive more than 32767 bytes, you must express count% differently. See Programming Note 4 in the beginning of this section.

The DMA routine also performs "byte packing"; that is, two bytes of data are stored in each of the integer array locations. The first byte received is placed into the least significant byte of the first array location.

mode% is an INTEGER that defines the type of DMA transfer to be made and the operating characteristics of the DMA controller. The most common settings for mode% are &H2005 for DMA input and &H2009 for DMA output.

The mode byte format is

Mode - High Byte

BIT	15	14	13	12	11	10	9	8
	X	X	WAIT	X	X	X	X	X

Mode - High Byte

BIT	7	6	5	4	3	2	1	0
	MOD1	MOD0	ADEC	INIT	OUT	INP	CS1	CS0

Where

x May be any value.

DMA (cont.)

WAIT This bit enables the DMA wait option. When this bit is 0, the DMA routine waits for the DMA transfer to be completed or a timeout to occur before returning to the called program.

When this bit is 1, the DMA controller is setup for the transfer and control returns to the user program without waiting for the end of the transfer.

CS0, 1 Select the channel for DMA transfer. Possible selections are as follows:

CS1	CS0	
0	1	Select DMA Channel 1
1	0	Select DMA Channel 2
1	1	Select DMA Channel 3

The selected DMA channel must agree with the setting of the DMA Level Jumpers. See Section 2.4 for more information.

NOTE: Some DMA channels may be assigned to other hardware within the PC. Check your PC system documentation to determine which channels are available.

INP When set to 1, this bit indicates the received data is written to PC memory via DMA. Both this bit and the OUT bit cannot be set to 1 at the same time.

OUT When set to 1, indicates the transmitted data is read from PC memory via DMA. Both this bit and the INP bit cannot be set to 1 at the same time.

INIT Enables DMA autoinitialize mode, when it is set to 1.

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 byte count, reset the initial address, and repeat the transfer again. This continues indefinitely until the DMA routine is called with INIT=0.

ADEC Controls the direction in which the DMA controller generates its addresses and obtains data. If ADEC = 0, the DMA controller is set to address increment mode. This means that the data is accessed from successive locations with ascending addresses within the PC memory. This mode is most often selected because it duplicates the manner in which array locations are accessed from the calling program.

If ADEC = 1, the DMA controller is set for address decrement mode. This means that the data is accessed from subsequent locations with descending addresses.

MOD0, 1 The DMA controller within the PC is capable of operating in three distinct modes. These two bits set the DMA controller mode. Available selections are

MOD1	MOD0	MODE
0	0	Demand Mode
0	1	Single Mode
1	0	Block Mode

Descriptions of these three modes follow.

Demand Mode - 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 processor 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.

Single Mode - 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.

Block Mode - In this mode, the DMA controller gains control of the bus and remains in control until the specified number of bytes has been transferred, regardless of the state of the DMA request line. Block Mode allows the fastest data transfer rate possible.

NOTE: BLOCK MODE IS NOT RECOMMENDED FOR MOST APPLICATIONS. This is because when block mode is selected, all other DMA channels are locked out and the microprocessor cannot execute any bus cycles. This can be dangerous in some circumstances. For example, in many PCs (particularly the older XT type machines), one of the DMA channels was used to refresh the dynamic RAM chips. (These store user programs and data.) If memory refresh were to be halted for an excessive period of time (hundreds of microseconds), all data within the RAMs would be lost.

returns **stat%** is an INTEGER describing the state of the transfer returned after the call. This stat% word differs from the one used in other routines because it indicates when "warning" conditions, as well as error conditions, occur. A warning differs from an error in that some (or all) of the transfer may have completed.

If the most significant bit of the stat% word is set, a warning has occurred. This results in a negative stat% value.

The stat% value is interpreted according to the following format:

DMA (cont.)

Stat (Output) - High Byte

BIT	15	14	13	12	11	10	9	8
	WARN	0	0	0	0	0	0	0

Stat (Output) - High Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	IOER	MDER	CSER

Where

NOTE: The meaning of the stat%'s low bits is dependent on the setting of the WARN bit.

WARN Warning. If this bit is set to 1, the other bits in stat% indicate that a Warning has occurred. If it is set to 0, the other bits indicate that an Error has occurred.

The following are Warning indications (WARN=1):

CSER Address Wraparound Error. If CSER=1 and WARN=1, an "address wraparound" has occurred. This condition arises as a result of hardware limitations within the PC. The DMA controller within most PCs generates 16 bits of address; however, a 20-bit address is required by the PC. Most PCs generate the four additional address bits with a "page register," which is wired to the most significant address lines. Address wraparound occurs whenever the DMA controller counts past its maximum count (FFFF rolls over to 0000), because there is no mechanism to "carry" the most significant bit into the page register.

For example, if the DMA routine were called with the SEG parameter set to 2000, and the OFS parameters set to FFFF, the DMA controller would be loaded with a count value of FFFF, and the page register with a 2. The first location accessed would be absolute address 2FFFF. The DMA controller would then increment its address (to 0), however the page register would not change. Thus, the next location accessed would be 20000, rather than 30000.

MDER Mode Error. If MDER=1 and WARN=1, it signifies that an invalid Mode Selection was made (see the MODE parameter description). The DMA routine substitutes Demand Mode (00) and continues.

TMO Timeout Error. If this bit = 1 and WARN=1, it signifies a Timeout Error. A Timeout Error indicates that the transfer was not completed during the designated DMA Timeout Period. It is possible for the timeout period to expire during a transfer of a large number of bytes to or from a slow device, even if the transfer occurs correctly.

The following are Errors (WARN=0):

DMA (cont.)

IOER Input/Output Error. If this bit = 1 and WARN=0, it indicates than the DMA routine has been called with an invalid selection of the INP and OUT bits in the mode parameter. Either INPUT or OUTPUT must be selected; but not both.

CSER DMA Channel Select Error. If CSER=1 and WARN=0, it indicates that an invalid DMA channel (channel 0) was selected for the transfer.

notes When calling DMA, you must declare an INTEGER array to store received data. Since each integer in BASICA uses 2 Bytes of memory, the total number of array locations allocated must be equal to or greater than one half the total number of bytes to be received.

example This example shows how to avoid an address wraparound error. The program transfers data into the computer's memory at an even segment boundary. This boundary is above the area used by DOS and your program (for example, &H7000). The data can then be moved into an array using the BASIC PEEK instruction. Note that this example stores one byte per array location.

```
100  DMA = 206          'DMA call offset
110  COUNT% = 1028      'Transfer 1028 points
120  DMAOFFS% =0        'Start with first array element
130  MODE% = &H2005     'Input, DMA Chan 1
140  STAT% = 0          'Initialize variable
150  DEF SEG =0         'BASIC's segment
160  DIM WAVE%(514)
170  DMASEG% = &H2000
180  DEF SEG = &HCC00 'KM-488-ROM memory segment
190  CALL DMA(DMASEG%,DMAOFFS%,COUNT%,MODE%, STAT%)
200  IF (STATUS%<> 0 ) THEN PRINT "FAILED", STAT% : STOP
210  DEF SEG = DMASEG% 'Get data from memory to array
220  FOR I% = 0 TO COUNT%-1
230  WAVE%(I%) = PEEK(I%)
240  NEXT I%
```

ENTER

purpose Addresses a specified device to talk, the KM-488-ROM to listen, and receives data into a string from the addressed device.

offset 21

usage

```
...
xx   ENTER = 21
xx   info$ = SPACE$(max.chars%) '(max.chars% < 256)
xx   leng% = 0
xx   adrs% =
xx   stat% = 0
xx   CALL ENTER (info$,leng%,adrs%,stat%)
...
xx   RCV$ = LEFT$(info$,leng%)
...
```

ENTER (cont.)

parameters **info$** is a STRING (up to 256 characters) which is to hold the received data. The string must be long enough to receive the expected number of characters. Carriage returns and the message terminator character in the incoming data are ignored and not stored with the received data. You should use the BASIC SPACE$ function to declare a string which is long enough to store the expected maximum number of characters to be received. After the data has been received, it should be copied into a new string which has been "trimmed" to length with the BASIC LEFT$ function. Or else, you may trim the first string to length, provided that you resize it with the BASIC SPACE$ function prior to calling the ENTER function again.

adrs% is an INTEGER containing the IEEE bus address of the device that will sent the data and the terminator to be used. This byte is of the following format:

Adrs (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	TRM1	TRM0	0	ADR4	ADR3	ADR2	ADR1	ADR0

Where

TRM1-0 Terminator Select. These two bits select the Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

These terminators are defined upon system configuration and are stored (along with the BASICA library code) in the EEPROM. They can be changed by running the CONFIG program as described in Chapter 2.

The easiest way to specify an alternate terminator is to add a factor to the GPIB address of the desired instrument which is specified within the ENTER call. The factors added for each terminator are as follows:

GPIB Address + 0 = Terminator 0
GPIB Address + 64 = Terminator 1
GPIB Address + 128 = Terminator 2
GPIB Address + 192 = Terminator 3

For example, if you wanted to receive a message using terminator 2 from a device at GPIB address 10, the value of adrs% supplied to ENTER would be 138 decimal (10 + 128).

ADR4-0 GPIB Address. These five bits are used to represent the GPIB address of the device to which the data is to be sent. GPIB addresses can range from 0 to 30.

ENTER (cont.)

returns **info$** is a STRING variable, up to 256 characters, which will contain the received data. The length of the string must be long enough to receive the expected number of characters. Enter will terminate reception of data when: 1) the number of characters received exceeds the length of the string, 2) the specified terminator is received, or 3) any character is received with the EOI signal asserted. Carriage returns and the terminator character in the incoming data are ignored and not stored with the received data. However, bytes other than the terminator which are received with EOI asserted will be stored.

leng% is an INTEGER, less than or equal to 256, which indicates the actual number of bytes which were stored. This number does not include message terminator characters or carriage returns.

stat% is an INTEGER which describes the state of the transfer returned after the call. If a stat value of 0 is returned, the transfer completed normally. Otherwise, the returned stat values (or combination of) are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	OVF	NC	ADRS

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

OVF Overflow Error. If this bit is a 1, then the info string was filled, before a terminator character or EOI was detected.

NC KM-488-ROM not an Active Controller . If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as an Active Controller.

ADRS Invalid GPIB address. If this bit is set to 1, then an invalid GPIB address was given.

example In the following example, data is sent from two different instruments to a KM-488-ROM. The KM-488-ROM is acting as the System Controller and is assigned to GPIB address 0. One of the two instruments is a voltmeter, requiring a Carriage Return-Line Feed terminator combination, assigned to GPIB address 7. The second instrument, located at GPIB address 10, requires a line feed as its terminator.

The voltmeter is first sent a string of data which represents its instrument setup command. Then, when addressed to talk, it sends its most current reading to the KM-488-ROM. The second instrument is instructed to send its status, when addressed to talk. It is assumed that the string sent by both instruments is 25 characters or less. The string is printed out on the computer screen.

ENTER (cont.)

```
10    DEF SEG=&HCC00
20    INIT = 0 : SEND=9 : ENTER=21
30    ADRS%=0 : MODE%=4
31    '
32    'Set up KM-488-ROM as System Controller with High Speed
34    'Bus Handshake at GPIB address 0
36    '
40    CALL INIT (ADRS%,MODE)
45    '
50    INST1%=7
60    INST1.TERM%=INST1%+64  'Use Terminator 1 for Instrument 1
70    INST2%=10  'Use Terminator 0 for Instrument 2
80    SETUP$="FOROTOMOX"     'String to setup Instrument 1
90    STAT$ = "SEND STATUS" 'String to obtain status from Inst 2
100   INSTRING$=SPACE$(25)  'Allocate space for received data
110   RLEN%=0      'Allocate variable for rcv length
120   CALL SEND (INST1%.TERM,SETUP$,STAT%)   'Setup Instrument 1
130   '
140   '  Check status returned by SEND call
150   '
160   IF STAT%<>0 THEN PRINT "Error sending to Inst 1 Status
        =";STAT%
170   '
172   'Read the data returned
180   CALL ENTER(INSTRING$,RLEN%,INST1%,STAT%)
190   IF STAT%<>0 THEN PRINT "Error receiving from Instrument 1"
200   DSP$=LEFT$(INSTRING$,RLEN%)
210   PRINT "INSTRUMENT 1 DATA ="; DSP$
220   CALL SEND(INST2%,STAT$,STAT%)
230   IF STATUS<>0 THEN PRINT "Error sending to Inst 2 - Status
        =";STAT%
240   CALL ENTER (INSTRING$,RLEN%,INST2%,STAT%)
250   IF STAT%<>0 THEN PRINT "Error receiving from Instrument 2"
260   DPS$=LEFT$(INSTRING$,RLEN%)
270   PRINT "Instrument 2 data ="; DSP$
```

INIT

purpose Initializes a KM-488-ROM by assigning it a GPIB address and establishing it as a System Controller or Device.

offset 0

usage

```
...
xx    INIT = 0
xx    adrs%= : mode%=
xx    CALL INIT(adrs%,mode%)
...
```

alternate usage `CALL INITIALIZE(adrs%,mode%)`

parameters adrs% is an INTEGER representing the IEEE bus address of the KM-488-ROM. This is an integer from 0 to 30.

INIT (cont.)

mode% is an INTEGER representing the operating mode of the KM-488-ROM. These can be any of the following values:

Mode - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	FAST	DEV	X

Where

X May be any value.

FAST Handshake Speed. If this bit is set to 1, High Speed GPIB bus handshaking will be used(500ns.). If it is set to 0, Low Speed GPIB bus handshaking (2 s.) will be used. See Chapter 3 for more information regarding the handshake speed.

DEV Device. If this bit is set to 1, then the KM-488-ROM is acting as a Device. Otherwise, when this bit is set to 0, the KM-488-ROM is acting as a System Controller. When System Controller is selected, the GPIB IFC line is momentarily asserted.

returns None.

example This example initializes the KM-488-ROM as a System Controller with a IEEE address of 0 with a High Speed Handshake.

```
10   DEF SEG=&HCC00
20   INIT=0
30   ADRS%=0 : MODE% =4
40   CALL INIT(ADRS%,MODE%)
```

PPOLL

purpose Initiates a Parallel Poll and returns a parallel poll response byte.

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

offset 15

usage

```
...
xx   PPOLL = 15
xx   resp% = 0
xx   PPOLL(resp%)
..
```

parameters None.

returns **resp%** is an INTEGER which will contain the parallel poll response.

PPOL (cont.)

notes Before you call the PPOLL routine, you must configure the Parallel Poll response of the device. To do this,

- Address it to listen.

- Send it a GPIB Parallel Poll Configure (PPC) command, using the XMIT command.

- Send a Parallel Poll Enable byte using the KM-488-ROM XMIT command. (Use the mnemonic CMD followed by nnn where nnn is the decimal value of the Parallel Poll Enable byte.

The Parallel Poll Enable Byte is of the format 0110SPPP, 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.

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

example This example assumes that the KM-488-ROM is connected to a Sorenson HPD30-10 Power Supply. This device is located at GPIB address 1. It is also assumed that this device drives bit 3 of the Parallel Poll Response byte to a logic "1" when service is required. To program the device to respond properly, send the Parallel Poll enable byte 01101011 (107) via the XMIT command.

```
10    DEF SEG=&HCC00
20    INIT=0 : XMIT=3 : PPOLL=15 : CTLADRS%=0 : MODE%=0
30    CALL INIT(CTLADRS%,MODE%)
40    COMMAND$="REN UNL UNT LISTEN 1 PPC CMD 107"
50    CALL XMIT(COMMAND$,STAT%)
60    IF (STAT%<>0) THEN PRINT "Error sending PPC cmd Status="; STAT%
70    CALL PPOLL(RESP%)
80    IF (RESP% AND 8) <> 0 THEN PRINT "HPD30-10 Requesting Service..."
```

RCV

purpose Receives data into a string.

offset 6

usage

```
...
xx    RCV = 6
xx    info$= SPACE$(max.chars%)  '(max.chars% < 256)
xx    leng%=
xx    stat% =
xx    CALL RCV (info$,leng%,stat%)
...
```

alternate usage

```
CALL RECEIVE(info$,leng%,stat%)
```

NOTE: The alternate usage assumes the use of Input Message Terminator 0.

RCV (cont.)

parameters **info$** is a STRING (256 characters max.) which will hold the received data. The string must be long enough to receive the expected number of characters. Carriage returns and the message terminator character in the incoming data are ignored and not placed in received data. However, bytes other than the terminator received with EOI will be stored. You should use the BASIC SPACE$ function to declare a string which is long enough to store the expected maximum number of characters to be received. After the data has been received, it should be copied into a new string and "trimmed" to length with the BASIC LEFT$ function.

NOTE: Before calling RCV, stat% must be initialized for terminator selection.

stat% is an INTEGER that selects the input terminator to be used. The terminator (stat%) values follow:

Stat (Input) - Low Byte

BIT	7	6	5	4	3	2	1	0
	TRM1	TRM0	X	X	X	X	X	X

Where

X May be any value.

TRM1-0 Terminator Select. These two bits select the Input Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

These terminators are defined upon system configuration and are stored (along with the BASICA library code) in the EEPROM. To change their values, run the CONFIG program as described in Chapter 2.

returns **info$** is a STRING variable (256 characters max.) containing the received data. The string must be long enough to receive the expected number of characters. Enter will terminate reception of data when 1) the number of characters received exceeds the length of the string, 2) a message terminator is received, or 3) any character is received with the EOI signal asserted. Carriage returns and the message terminator in the incoming data are ignored and not placed in received data. However, bytes other than the message terminator received with EOI are stored.

leng% is an INTEGER, less than or equal to 256, which indicates the actual number of bytes which were received and stored.

stat% is an INTEGER which describes the state of the transfer returned after the call. The returned stat% values (or combination of) are interpreted as follows:

RCV (cont.)

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	OVF	NL	0

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

OVF Overflow Error. If this bit is a 1, then the RCV routine received more characters than could fit into the info string.

NL KM-488-ROM not a Listener. If this bit is set to a 1, it indicates the RCV was called before the KM-488-ROM was designated as a Listener.

notes The KM-488-ROM must be addressed to listen and a device must be addressed to talk prior to calling this routine.

example This example shows how the RCV routine might be used together with the XMIT routine to receive data. It uses the XMIT routine to command a Keithley 196 voltmeter to take a reading. The meter reading is received using the RCV routine. It is assumed that the meter reading returned will fit into a 25-character string.

This example assumes that the KM-488-ROM has been configured such that transmit message terminator 1 is Carriage Return-Line Feed combination and this combination is also used by the Keithley 196.

```
10 DEF SEG=&HCC00
20 INIT = 0 : XMIT = 3 : RCV=6 : ADRS%=0 : MODE%=0
30 CALL INIT(ADRS%,MODE%)
40 SETUP$="REN UNL UNT LISTEN 7 MTA DATA 'F0R3S1T3X' T1 GET
    UNL UNT TALK 7 MLA"
50 CALL XMIT (SETUP$,STAT%)
60 IF (STAT%<>0) THEN PRINT "Error sending cmd string
    Status="; STAT%
70 RCVDAT$=SPACE$(25)       'Allocate space for receive data
80 STAT%=0                  'use Input terminator 0
81 RCVLEN% = 0
90 CALL RCV(RCVDAT$,RCVLEN%,STAT%)
100 IF (STAT%<>) THEN PRINT "RCV Status Error Status=";STAT%
105 DAT$ = LEFT$(RCVDAT$,RCVLEN%)
110 PRINT "Received data=";DAT$;"Length=";RCVLEN%
```

RCVA

purpose Receives data into an array.

offset 203

RCVA (cont.)

usage

```
...
xx   RCVA = 203
xx   DIM INDAT%(N%)
xx   seg% =
xx   ofs% =
xx   maxlen% = 2*N%
xx   rcvlen% =
xx   stat% =
xx   ofs%=VARPTR(indat%(0))
xx   CALL RCVA(seg%,ofs%,maxlen%,rcvlen%,stat%)
```

alternate usage

```
RARRAY(seg%,ofs%,maxlen%,rcvlen%,stat%)
```

NOTE: The alterante usage assumes the use of Input Message Terminator 0.

parameters

seg% is an integer representing the segment portion of the memory address of the data. seg% is usually set to -1 to indicate the BASICA data segment.

ofs% is the offset portion of the memory address of the data. This is usually obtained using the VARPTR function. The VARPTR function must be called immediately prior to the RCVA function call and all variables used within the program must be declared prior to the VARPTR function. This is because BASICA dynamically allocates storage space and if variables are declared after the VARPTR call, the array may be relocated and the data will be placed into the wrong location. This may result in a program crash.

maxlen% is an INTEGER containing the maximum number of data bytes to be received. See Programming Note 4, found at the beginning of this section, if you want to send more than 32767 bytes.

stat% is an INTEGER which selects the type of terminator to be used. This parameter must be initialized every time you call the RCVA routine. This integer is interpreted according to the following format:

Stat (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	STRM	TRM1	TRM0	X	X	X	X	X

Where

STRM Enable/Disable String Message Terminators. If this bit is 1, a Message Terminator Character will be used to detect the end of reception. If this bit is 0, a Message Terminator Character will not be used.

TRM1-0 Terminator Select. These two bits select the Input Message Terminator to be used to signal the end of a transmission. (The STRM bit must be set to 1.) Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

The values for these terminators can be changed by running the CONFIG program as described in Chapter 2.

returns **rcvlen%** is an INTEGER containing the actual number of data bytes which were received.

stat% is an INTEGER describing the state of the transfer returned after the call.

The RCVA routine returns three status bits within the STAT variable. The TMO bit is used to signal a timeout error. The REOI bit signals that the routine returned because the terminator was detected (if enabled), or EOI was received. The NL bit is set if the RCVA routine was called and the board was not addressed to listen. Unlike other KM-488-ROM routines, it is possible to return a non-zero status when the call was completed successfully.

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	REOI	0	TMO	0	NL	0

Where

REOI Reason for RCVA Termination. If this bit is a 1, then RCVA routine ceased because an EOI or terminator character was received. If this bit is a 0, then the RCVA was terminated because an error occurred or the maximum byte count was reached.

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NL KM-488-ROM not a Listener. If this bit is set to a 1, it indicates the RCVA was called before the KM-488-ROM was designated as a Listener.

notes 1. The KM-488-ROM must be addressed to listen before calling this routine.

2. When calling RCVA, you must declare an integer array to store received data. Since each integer in BASICA uses 2 bytes of memory, the total number of array locations allocated must be equal to or greater than one half the total number of bytes to be received.

3. The maxlen% parameter must not exceed twice the number of array locations or else the data will be stored into an area of memory which has been allocated to different parts of the system.

RCVA (cont.)

example This example illustrates a typical way to use RCVA.

```
100   DEF SEG = &HCC00              'KM-488-ROM memory segment
110   RCVA = 203 : XMIT = 3         'RCVA and XMIT call offsets
120   MAXLEN% = 200 : RCVLEN% = 0   'Initialize variables
130   DIM X%(99) : T$ = 'MLA TALK 5";
135   CALL XMIT(T$,stat%0
140   BSEG% = -1
150   STAT% = 0                     'No message terminator
160   OFS% = VARPTR(X%(0))
170   CALL RCVA(BSEG%,OFS%,MAXLEN%,RCVLEN%,STAT%)
180   IF STAT%<>0 THEN PRINT "ERROR", STAT% : STOP
```

SEND

purpose Addresses a specified device to listen, the KM-488-ROM to talk, and sends data from a string.

offset 9

usage

```
xx SEND = 9
xx adrs%=  : stat% = : info$ = "..."
xx CALL SEND (adrs%,info$,stat%)
...
```

parameters adrs% is an INTEGER containing the IEEE bus address of the device that the data is to be sent to and the terminator to be used. This byte is of the following format:

Adrs (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	TRM1	TRM0	0	ADR4	ADR3	ADR2	ADR1	ADR0

Where

TRM1-0 Terminator Select. These two bits select the Input Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

The values for these terminators can be changed by running the CONFIG program as described in Chapter 2.

The easiest way to specify an alternate terminator is to add a factor to the GPIB address of the desired instrument which is specified within the SEND call. The factors added for each terminator are as follows:

GPIB Address + 0	= Terminator 0
GPIB Address + 64	= Terminator 1
GPIB Address + 128	= Terminator 2
GPIB Address + 192	= Terminator 3

For example, if you wanted to send a message using message terminator 2 to a device at GPIB address 10, the value of adrs% supplied to SEND would be 138 decimal (10 + 128).

ADR4-0 GPIB Address. These five bits are used to represent the GPIB address of the device to which the data is to be sent. GPIB addresses can range from 0 to 30.

info$ is a STRING (256 chars max.) containing the data to be sent.

returns **stat%** is an INTEGER describing the state of the transfer returned after the call. The returned stat values (or combination of) are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NC	ADRS

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NC Not Active Controller. If this bit is a 1, then the SEND routine was called when the KM-488-ROM was not an Active Controller.

ADRS Invalid Address. If this bit is set to a 1, an invalid IEEE-488 device address was given.

example This example shows how to send data from a KM-488-ROM to a device. The KM-488-ROM is initialized as a System Controller located at GPIB address 10. The KM-488-ROM uses high-speed handshaking. The data (a device setup string) is sent to a device located at GPIB address 2 using terminator 0.

```
10   DEF SEG=&HCC00          'Assumes EEPROM is at &HCC00
20                           'Change to suit your setup
25   INIT = 0                'Offset of INIT routine
30   SEND=9                  'Offset of SEND routine
40   ADRS%=10 : MODE%=4      'Setup as System Controller at
                               'GPIB adrs 10 with High Speed
                                'Handshake
60   CALL INIT(ADRS%,MODE%)
70                           'Declare setup string and address
80                           ' of instrument
90   SETUP$="F0R0T0M0X" : ADRS%=2 : STAT% =0
100  CALL SEND (ADRS%,SETUP$,STAT%)
110  IF STAT <> 0 THEN PRINT "Error sending Status=";STAT%
```

SETINT

purpose Sets the KM-488-ROM's interrupt enable bits.

offset 212

usage

```
...
xx   SETINT = 212
xx   intval% =
xx   CALL SETINT(intval%)
..
```

parameters **intval%** is an INTEGER containing the address and value of the Interrupt Mask Register. This is interpreted as follows:

INTVAL (Input) - High Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	X	ADRS

Where

X May be any value.

ADRS If this bit is set to 0, bits 0 through 7 will be written to Interrupt Mask 1. If this bit is set to 1, bits 0 through 7 will be written to Interrupt Mask 2.

INTERRUPT MASK 1

INTVAL (Input) - Low Byte (ADRS = 0)

BIT	7	6	5	4	3	2	1	0
	0	0	GET	0	DEC	0	0	0

Where

GET When this bit is set to 1, an interrupt will be generated when a KM-488-ROM acting as a device received a GPIB GET (Group Execute Trigger) command while addressed to listen.

DEC When this bit is set to 1, an interrupt is generated when a Device Clear is received.

INTERRUPT MASK 2

INTVAL (Input) - Low Byte (ADRS = 1)

BIT	7	6	5	4	3	2	1	0
	0	SRQI	0	0	0	LOKC	REMC	ADSC

Where

SRQI When this bit is set to 1, an interrupt is generated when SRQ is received.

LOKC When this bit is set to 1, an interrupt is generated when the state of the Local Lockout bit changes.

SETINT (cont.)

REMC When this bit is set to 1, an interrupt is generated when the state of the Local/Remote bit changes.

ADSC When this bit is set to 1, an interrupt is generated when the state of the LA, TA, or CIC bits within the address status register changes.

returns None.

notes You must have an interrupt handling routine set-up in order to use the interrupts. In BASICA, the most common way to handle interrupts is through a routine which maps the interrupt into BASICA's lightpen interrupt, allowing you to execute a BASICA ON PEN statement to execute the interrupt service routine.

example This example shows you how to use the SETINT routine to enable the SRQ interrupt.

```
10   DEF SEG = &HCC00
20   SETINT = 212
30   INTVAL% = &H140
40   CALL SETINT (INTVAL%)   'Enable SRQ interrupt
```

SETSPOLL

purpose Defines the Serial Poll Response of a KM-488-ROM acting as a device (non-Controller).

offset 215

usage

```
xx SETSPOLL = 215
xx resp% =
xx CALL SETSPOLL (resp%)
```

parameters **resp%** is an INTEGER describing the serial poll response and the state of the SRQ bit. This byte is of the following format:

Resp% (Input) - Low Byte

BIT	7	6	5	4	3	2	1	0
	SPR8	RSV	SPR6	SPR5	SPR4	SPR3	SPR2	SPR1

Where

SPR1-8 Bits 1 through 8 of this device's Serial Poll Response Byte.

RSV If this bit is 1, SRQ will be asserted to request servicing. Otherwise, SRQ will not be asserted.

returns None.

SETSPOLL (cont.)

example This example illustrates a common use of SETSPOLL.

```
100   DEF SEG=&HCC00     'KM-488-ROM memory seg
110   SETSPOLL = 215     'Call offset
120   RESP% = 0
130   IF (ERROR1 = TRUE) THEN RESP%=RESP%+1
     'Check local errors
140   IF (ERROR2 = TRUE) THEN RESP%=RESP%+2
     'and set bits
150   IF (ERROR3 = TRUE) THEN RESP%=RESP%+4
160   IF (ERROR4 = TRUE) THEN RESP%=RESP%+8
170   IF (ERROR5 = TRUE) THEN RESP%=RESP%+16
180   IF (ERROR6 = TRUE) THEN RESP%=RESP%+32
190   IF (ERROR7 = TRUE) THEN RESP%=RESP%+128
200   IF (RESP% <> 0) THEN RESP%=RESP%+&H40
     'Indicate SRQ asserted
210   CALL SETSPOLL(RESP%)
```

SPOLL

purpose Performs a serial poll of the specified device.

offset 12

usage

```
...
xx SPOLL = 12
xx adrs% =
xx resp% =
xx stat% =
xx CALL SPOLL(adrs%,resp%,stat%)
```

parameters **adrs%** is an INTEGER containing the IEEE bus address of the device that is to be serial polled. This can range from 0 to 30.

returns **resp%** is an INTEGER containing the serial poll response received. 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.

stat% is an INTEGER describing the state of the transfer returned after the call. The stat value is interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NC	ADR

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NC KM-488-ROM not a Controller. If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as an Active Controller.

SPOLL (cont.)

ADR Invalid GPIB Address. If this bit is set to 1, an invalid GPIB address was provided.

example This example illustrates a simple serial poll of a device located at GPIB address 10.

```
10    DEF SEG=&HCC00
20    INIT=0 : SPOLL=12 : CTLADRS%=0 : MODE%=0 : ADRS%=10
25    RESP%=0 : STAT%=0
30    CALL INIT(CTLADRS%,MODE%)
40    CALL SPOLL(ADRS%,RESP%,STAT%)
50    IF (STAT%<>0) PRINT "SPOLL Status Error Status=",STAT% : STOP
60    PRINT "Serial Poll Response=";RESP%
70    IF ((RESP% AND &H40)<>0) PRINT "Device Requesting
         Service...."
```

STATUS

purpose Returns the current setting of the requested status parameter.

usage

```
xx STATUS = 209
xx reg% =
xx stat% =
XX CALL STATUS(reg%,stat%)
```

parameters **reg%** is an INTEGER containing the address of the register or configuration parameter to be queried. This value corresponds to a 4-bit field specifying the status register or configuration parameter to be read. The format of the reg% byte is as follows:

Reg (input) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	ADR3	ADR2	ADR1	ADR0

Where

X May be any value.

ADR3-0 REGISTER/PARAMETER SELECT. This is a 4-bit field which specifies the status register or configuration parameter to be read. Registers and parameters are selected as follows:

ADR3	ADR2	ADR1	ADR0	REGISTER/PARAMETER
0	0	0	0	Address Status Reg
0	0	0	1	Interrupt Status 1 Reg
0	0	1	0	Interrupt Status 2 Reg
0	0	1	1	DMA Status Reg
0	1	0	0	Output Terminator 0
0	1	0	1	Output Terminator 1
0	1	1	0	Output Terminator 2
0	1	1	1	Output Terminator 3
1	0	0	0	Input Terminator 0
1	0	1	1	Input Terminator 1
1	1	1	0	Input Terminator 2
1	1	1	1	Input Terminator 3
1	0	0	0	I/O Timeout Parameter
1	0	0	1	DMA Timeout Parameter
1	1	1	0	I/O Port Address
1	1	1	1	GPIB Address of KM-488-ROM

returns **reg%** - When STATUS obtains the value of one of the four transmit message terminators, this variable will contain two flag bits which determine the length of the terminator and whether or not EOI is asserted with the last byte. When obtaining other parameters, reg% will retain its input value.

Reg (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	LEN	EOI

Where

LEN Terminator Length. If this bit is set to 0, then the terminator is one byte long. If this bit is set to 1, then the terminator is two bytes long.

EOI If this bit is set to 1, EOI is asserted when the last terminator byte is sent. Otherwise, EOI is not asserted.

stat% is an INTEGER describing the status bits for the register or the configuration parameter which was specified by the reg% parameter. Unless otherwise noted, the high byte of stat% is returned as 0.

Address Status Register

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	CIC	X	X	X	X	LA	TA	X

Where

X This bit may be any value.

CIC Active Controller. If this bit is set to 1, then the KM-488-ROM is a System Controller.

STATUS (cont.)

LA Listener. If this bit is set to 1, then the KM-488-ROM is a Listener.

TA Talker. If this bit is set to 1, then the KM-488-ROM is a Talker.

Interrupt Status Register 1

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	GET	X	DEC	X	X	X

Where

X This bit may be any value.

GET Group Execute Trigger. If this bit is set to 1, then a Group Execute Trigger command was received while the KM-488-ROM was a device.

DEC When this bit is set to 1, a Device Clear was received.

Interrupt Status Register 2

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	SRQ1	LOK	REM	X	X	X	ADSC

Where

X *This bit may be any value.*

SRQI When this bit is set to 1, it indicates SRQ was active. (Active Controller mode only.)

LOK When this bit is set to 1, the device was set to Local Lockout. (Device mode only.)

REMC *When this bit is set to 1, the device was configured for* remote operation. (Device mode only.)

ADSC When this bit is set to 1, a change of the address status occurred (i.e., untalk to talk, device to active controller, etc.).

DMA Status Register

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	TC3	TC2	TC1	X

NOTE: *DMA Status Register: it is useful to check the status* of this register when running DMA operations in background mode.

Where

X This bit may be any value.

STATUS (cont.)

TC1 When this bit is set to 1, it indicates that DMA channel 1 has reached terminal count.

TC2 When this bit is set to 1, it indicates that DMA channel 2 has reached terminal count.

TC3 When this bit is set to 1, it indicates that DMA channel 3 has reached terminal count.

Message Terminator #0-3: Contains First and Last bytes of the message terminator. Input terminators are only one byte long and are contained in the Least Significant Byte. In the case of a two character Output Terminator, the Most Significant Byte of this parameter is the first character sent.

DMA Timeout and I/O Timeout Parameters: Contains the value of the desired parameter as an unsigned value in the low and high bytes of stat%. The timeout value is expressed in milliseconds (0 to 65535).

notes The bits contained in the Interrupt Status 1 and 2 registers are extremely volatile. When you read these registers, any bits which were set are automatically cleared by the READ operation. This is extremely important to note when reading Interrupt Status Register 1, as some of the bits (not shown above) are used by various KM-488-ROM routines. It may be possible to cause various KM-488-ROM routines to report a timeout error if this register is read while the KM-488-ROM is addressed to talk or listen.

example This example illustrates how to use the STATUS routine.

```
100   DEF SEG=&HCC00                'KM-488-ROM memory segment
110   STATUS=209                    'STATUS call offset
120   REG%=0 : STAT% = 0
130   CALL STATUS(REG%,STAT%)
140   IF (STAT% AND &H80) <>0 THEN PRINT "KM-488-ROM = Sys Contr"
150   IF (STAT% AND 2) <>0 THEN PRINT "Addressed to talk..."
160   IF (STAT% AND 4) <>0 THEN PRINT "Addressed to Listen..."
170   REG%=15
180   CALL STATUS(REG%,STAT%)
190   PRINT "IEEE-488 Address = "; STAT%
```

XMIT

purpose Sends GPIB commands and data from a string.

offset 3

usage

```
...
xx XMIT = 3
xx info$ = "..."
xx stat% =
xx CALL XMIT (info$,stat%)
...
```

alternate usage

```
CALL TRANSMIT(info$,stat%)
```

XMIT (cont.)

parameters **info$** is a STRING variable containing a series of GPIB commands and data. Each item must be separated by one or more spaces. Commands can be in UPPER or lower case. The Transmit commands are described in Chapter 3. These commands include

CMD	GTL	MTA	SDC	T0
DATA	GTLA	MLA	SEC	T1
DCL	IFC	PPC	SPE	T2
END	LISTEN	PPD	SPD	T3
EOI	LLO	PPU	TALK	UNL
GET	LOC	REN	TCT	UNT

returns **stat%** is an INTEGER which describes the state of the transfer returned after the call. The returned stat value can be interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	x	ADRS	NCTL	UNDF	TMO	STR	NT	STX

Where

X May be any value.

ADRS Invalid GPIB address. If this bit is set to 1, then an invalid GPIB address was given.

NCTL Not a System Controller. If this bit is set to 1, it indicates that the KM-488-ROM tried to send GPIB Bus Commands when it was not an Active Controller.

UNDF Undefined Command. If this bit is set to 1, the info string contained an undefined command.

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

STR String Error. If this bit is set to one, then a quoted string, END, or terminator was found without a DATA subcommand preceding it.

NT KM-488-ROM not a Talker. If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as a Talker.

STX Syntax Error. If this bit is set to 1, a syntax error was found.

example This example illustrates one way to use the XMIT command with a Keithley 196 Voltmeter. This meter is assigned GPIB address 7 and is configured to a 30 Volt DC range with 4 1/2 digit accuracy. The meter is also configured to take a new reading each time a Group Execute Trigger Bus command (GET) is received. It is assumed that the meter has been set to use a CR, LF, EOI (the default for Message Terminator 1). The program then triggers the instrument to get the first reading, and makes it a talker and the KM-488-ROM a listener in order to get the first reading.

The device to receive the setup command string which must be sent to the meter contains the following device commands:

XMIT (cont.)

F0 Select DC Volts mode
R3 Select 30 Volt range
S1 Select 4 1/2 digit accuracy
T3 Take one reading when GET received
X Execute the prior commands within the string

The device to receive the setup command string must also be programmed to assert the GPIB REN signal (This allows the meter to receive GPIB commands.) and to LISTEN (This allows the device to receive the string.). The programming sequence used consists of the following:

- Setting Remote Enable (REN).
- Setting all devices to UNTalk and UNListen.
- Addressing the 196 to LISTEN.
- Addressing the KM-488-ROM to talk (My Talk Address).
- Sending the Device-Dependent Commands as a string of DATA.
- Sending the appropriate message terminator characters after the data.
- Issuing the Group Execute Trigger bus command.
- Unaddressing all devices.
- Addressing the meter to TALK and the KM-488-ROM to LISTEN (My Listen Address) in preparation for receiving the latest reading.

```
10    DEF SEG=&HCC00
20    INIT=0 : XMIT=3 : ADRS%=0 : MODE%=0 : STAT%=0
30    CALL INIT(ADRS%,MODE%)
40    SETUP$ = "REN UNL UNT LISTEN 7 MTA DATA 'F0R3S1T3X' T1 GET
       UNL UNT TALK 7 MLA"
50    CALL XMIT(SETUP$,STAT%)
60    IF (STAT%<>0) THEN PRINT "Error send cmd string
       Status=";STAT%
```

XMITA

purpose Transmits data from an array.

offset 200

usage

```
...
xx XMITA = 200
xx seg% =
xx ofs% =
xx DIM INFO%(n%)
xx count% = 2*n%
xx term% =
xx stat% = 0
xx ofs% = VARPTR(info%(0))
xx CALL XMITA(seg%,ofs%,count%,term%,stat%)
...
```

alternate usage

```
CALL TARRAY(seg%,ofs%,count%,term%,stat%)
```

XMITA (cont.)

parameters **seg%** is an INTEGER representing the segment portion of the memory address of the data. seg% is set to -1 to the BASICA data segment.

ofs% is an INTEGER representing the offset portion of the memory address of the data. This is usually obtained using the VARPTR function. The VARPTR function must be called immediately prior to the XMITA function call and all variables used within the program must be declared prior to the VARPTR function. The reason for this is that BASICA can dynamically allocated storage space and if variables are declared after the VARPTR call, there is a good possibility that the array will be relocated and the data will be placed into the wrong location.

count% is an INTEGER containing thhe number of data bytes to be transmitted. To send more than 32767 bytes, refer to Programming Note 4 in the beginning of this section.

term% is an INTEGER which selects the terminator to be used. This byte is of the following format:

Term (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	STRM	TRM1	TRM0	X	X	X	X	EOI

Where

X This bit may be any value.

STRM Send Message Terminators. If this bit is set to 1, then the message terminator(s) will be sent at the end of the transmission. Otherwise, they will not.

TRM1-0 Terminator Select. These two bits select the Output Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF EOI
0	1	1	CR LF EOI
1	0	2	CR EOI
1	1	3	LF CR EOI

These terminators can be redefined by running the CONFIG program as described in Chapter 2.

EOI Asserts EOI. If this bit is set to 1, then EOI will be asserted when the last byte is sent. Otherwise, EOI will not be asserted.

returns **stat%** is an INTEGER describing the state of the transfer returned after the call. The stat value is interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NT	0

XMITA (cont.)

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NT KM-488-ROM not a Talker. If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as a Talker.

notes When calling XMITA, you must declare an integer array from which to transmit data. Since each integer in BASICA uses 2 Bytes of memory, the total number of array locations allocated must be equal to or greater than one half the total number of bytes to be received.

example This example illustrates the use of XMITA. It shows you how to properly set-up an array from which to send the data. Note that the data is sent without a terminator or EOI asserted.

```
100   DIM Z%(1023)
110   CHKSUM%=0
120   FOR I%=0 TO 1023
130   Z%(I%)=I%                  '2 bytes packed per element
140   CHKSUM%=(CHKSUM%+I%) AND &H00FF
150   NEXT I%
160   DEF SEG=&HCC00     'KM-488-ROM memory segment
170   XMITA=200      'XMITA call offset
180   COUNT% = 2048 : TERM%=0 : FLAG%=0  ' No EOI or Terminator
190   SEG%=-1 : OFS%=VARPTR(Z%(0))    'BASICA segment
200   CALL XMITA(SEG%,OFS%,COUNT%,TERM%,FLAG%) 'Data sent
210   IF FLAG%<>0 THEN GOTO 2000
220   EOI%=1 : COUNT%=1    'Send checksum with EOI
230   SEG%=-1 : OFS%=VARPTR(CHKSUM%)  'Checksum sent
240   CALL XMITA(SEG%,OFS%,COUNT%,EOI%,FLAG%)
250   IF FLAG%<>0 THEN GOTO 2000
.
.
.
2000  PRINT "ERROR NUMBER:      ";FLAG%;
2010  STOP
```

■ ■ ■

Chapter 5

PROGRAMMING IN QUICKBASIC

While Chapter 3 gives a brief overview of the routines available for programming the KM-488-ROM, this chapter gives instructions for calling the routines from QuickBASIC. The routines appear in alphabetical order and include a sample program for each.

5.1 GENERAL

Supported Versions

QuickBASIC 4.0 and higher

The Environment

Before you begin to develop programs in QuickBASIC, several files must be present in your working directory. Copy the following files from the KM-488-ROM disks to your working directory:

QUICKBASIC 4.0	QUICKBASIC 7.0(QBX)
\QB\KM488QB.BI	\QB\KM488QB.BI
\QB\KM488QB4.QLB	\QB\KM488QB7.QLB
\QB\KM488QB4.LIB	\QB\KM488QB7.LIB

File Header

Be sure to include the following line within your program:

```
'$INCLUDE:'km488qb'
```

Including of this file allows QuickBASIC to check that the correct number and type of parameters are specified for each routine called.

Compiling

Once your QuickBASIC application program has been written, you will 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 4.0	FOR QUICKBASIC 7.0(QBX)
`qb /Lkm488qb4 yourprog`	`qb /Lkm488qb7 yourprog`

where *yourprog* is the name of your program.

To create a Standalone Program

This process compiles the QuickBASIC source code and links it to the QuickBASIC and KM-488-ROM 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 4.0:

```
bc /o /d yourprog.bas;
link yourprog,,,bcom45+km488qb4;
```

where

yourprog is the name of your program.
bcom45 is the QuickBASIC Runtime library name.
km488qb4 is the linkable BASIC library file.

Software

The KM-488-ROM firmware contains a number of configuration parameters that govern the default settings of the input and output message terminator settings, message timeout periods, and I/O port addresses. The default terminators are shown in the following table. If these default values are unsatisfactory, they may be changed by calling either the INTERM or OUTTERM routine.

The default DMA and I/O Timeouts are 10 seconds. These defaults may be altered by calling the DMATIMEOUT or IOTIMEOUT routine.

Default Terminator Settings

TERM #	OUTPUT TERMINATOR	INPUT TERMINATOR
0	LF EOI	LF
1	CR LF EOI	CR
2	CR EOI	, (comma)
3	LF CR EOI	; (semi-colon)

Programming Notes

1. Any parameters which appear as variables may also be passed as constants.

2. Parameters which are also used to return values must be declared as variables.

3. Integer variable names end with a percent sign and integer constants do not contain a decimal point.

4. All integers are treated by the KM-488-ROM 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-ROM routines.

5.2 DESCRIPTION FORMAT FOR ROUTINES

The format for each descriptions is as follows:

purpose ... a brief description of the routine. See Chapter 3 for more detailed descriptions.

usage ... gives an example of usage for each routine and assumes the input parameters are passed in as variables. These parameters can also be passed in directly. See the General Programming Notes for more information.

alternate usage ... lists alternate usage for the routine, if any. Unless otherwise noted, the alternate usage performs exactly the same function as the usage.

parameters ... describes each of the input parameters.

returns ... describes any values returned by the routine.

notes ... lists any special programming considerations.

example ... gives a programming example using the routine.

5.3 ROUTINES

DMATIMEOUT

purpose Sets the maximum length of time for a DMA transfer to complete before a timeout error is reported. (See RCVA and XMITA routine descriptions.)

usage

```
CALL DMATIMEOUT(time%)
```

alternate usage CALL SETTIMEOUT(time%)

NOTE: The alternate usage sets both the DMA and I/O Timeouts to the specified value.

parameters time% is an INTEGER which represents the timeout period to elapse during a DMA transfer. A DMA Timeout Error will be generated when the time to transfer (via DMA) an entire message exceeds the set DMA timeout value (time). time% can range from 0 to 65535 milliseconds and is internally rounded to the closest integer multiple of 55 milliseconds. For values greater than or equal to 32768, time must be represented differently. See Programming Note 4 at the begining of this section.

returns None.

example This example sets the DMA Timeout period to 5 seconds.

```
DMA TIMEOUT (5000)
```

ENTER

purpose Addresses a specified device to talk, the KM-488-ROM to listen, and receives data from the addressed device into a string.

usage

```
...                              'Setup an 80 char string
info$ = SPACE$(80)               ' to receive the data
CALL ENTER (info$,leng%,adrs%,stat%)
...
```

parameters **info$** is a STRING which is to hold the receive data. The string must be long enough to receive the expected number of characters. This may be accomplished using the QuickBASIC SPACE$ function. For example, the line INFO$ = SPACE$(100) allocates a 100 character string for storing data. Carriage returns and the message terminator character in the incoming data are ignored and not placed in received data.

adrs% is an INTEGER containing the IEEE bus address of the device that the data is to be sent to and the terminator to be used. This byte is of the following format:

Adrs (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	TRM1	TRM0	0	ADR4	ADR3	ADR2	ADR1	ADR0

Where

TRM1-0 Terminator Select. These two bits select the Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

These terminators may be changed by the INTERM routine.

The easiest way to specify an alternate terminator is to add a factor to the GPIB address of the desired instrument which is specified within the ENTER call. The factors added for each terminator are as follows:

GPIB Address + 0 = Terminator 0
GPIB Address + 64 = Terminator 1
GPIB Address + 128 = Terminator 2
GPIB Address + 192 = Terminator 3

For example, if you wanted to receive a message using terminator 2 from a device at GPIB address 10, the value of adrs% supplied to ENTER would be 138 decimal (10 + 128).

ADR4-0 GPIB Address. These five bits are used to represent the GPIB address of the device to which the data is to be sent. GPIB addresses can range from 0 to 30.

ENTER (cont.)

returns **info$** is a STRING variable, up to 256 characters, which will contain the received data. The length of the string must be long enough to receive the expected number of characters. Enter will terminate reception of data when: 1) the number of characters received exceeds the length of the string, 2) the specified terminator is received, or 3) any character is received with the EOI signal asserted. Carriage returns and the terminator character in the incoming data are ignored and not stored with the received data. However, bytes other than the terminator which are received with EOI asserted will be stored.

leng% is an INTEGER, less than or equal to 256, which indicates the actual number of bytes which were stored. This number does not include message terminator characters or carriage returns.

stat% is an INTEGER which describes the state of the transfer returned after the call. If a stat value of 0 is returned, the transfer completed normally. Otherwise, the returned stat values (or combination of) are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	OVF	NC	ADRS

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

OVF Overflow Error. If this bit is a 1, then the info string was filled, before a terminator character or EOI was detected.

NC KM-488-ROM not an Active Controller . If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as an Active Controller.

ADRS Invalid GPIB address. If this bit is set to 1, then an invalid GPIB address was given.

example In the following example, data is sent from two different instruments to a KM-488-ROM. The KM-488-ROM is acting as the System Controller and is assigned to GPIB address 0. One of the two instruments is a voltmeter, requiring a Carriage Return-Line Feed terminator combination (Term 1), assigned to GPIB address 7. The second instrument, located at GPIB address 10, requires a line feed (Term 0) as its terminator. The voltmeter is first sent a string of data which represents its instrument setup command. Then, when addressed to talk, it sends its most current reading to the KM-488-ROM. The second instrument is instructed to send its status, when addressed to talk.

It is assumed that the string sent by both instruments is 25 characters or less. The string is printed out on the computer screen.

ENTER (cont.)

```
'$INCLUDE: 'km488qb'              'Use terminator 1 to send to INST1
INST1% = 7 : INST2% = 10 : INST1.TERM%=INST1%+64
INSTRING$=SPACE$(25)              'Allocate space for received data
CALL init(0,4)
CALL send(INST1.TERM%,"F0R0T0M0X",STAT%)
                                  'check status of SEND call...
IF (STAT%<>0) THEN PRINT "Error sending to Instrument 1 status
  =";STAT%                        'Receive the data...
CALL enter(INSTRING$,RLEN%,INST1%,STAT%)
                                  'Check status of ENTER call...
IF (STAT%<>0) THEN PRINT "Error receiving from Instrument 1 status
  =";STAT%                        'Clean up received data...
DSP$=LEFT$(INSTRING$,RLEN%)
PRINT "Instrument 1 data = ";DSP$
                                  'Setup and get data from Instrument
                                  '2...
CALL SEND(INST2%,"SEND STATUS",STAT%)
IF (STAT% <> 0) THEN PRINT "Error sending to Instrument 2 Status =
  ", STAT%
CALL ENTER(INSTRING$,RLEN%,INST2%,STAT%)
IF (STAT%<>0) THEN PRINT "Error receiving from Instrument 2"
DSP$=LEFT$(INSTRING$,RLEN%)
PRINT "Instrument 2 data =";DSP$
```

INIT

purpose Initializes the KM-488-ROM by assigning its GPIB address and establishing it as a System Controller or Device.

usage `CALL INIT (adrs%,mode%)`

alternate usage `CALL INITIALIZE (adrs%,mode%)`

parameters **adrs%** is an INTEGER representing the IEEE bus address of the KM-488-ROM. This is an integer from 0 to 30.

mode% is an INTEGER representing the operating mode of the KM-488-ROM. These can be any of the following values:

Mode - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	FAST	DEV	X

Where

X May be any value.

FAST Handshake Speed. If this bit is set to 1, High Speed GPIB bus handshaking will be used(500ns.). If it is set to 0, Low Speed GPIB bus handshaking (2 s.) will be used. See Chapter 3 for more information regarding the handshake speed.

INIT (cont.)

DEV Device. If this bit is set to 1, then the KM-488-ROM is acting as a Device. Otherwise, when this bit is set to 0, the KM-488-ROM is acting as a System Controller. When System Controller is selected, the GPIB IFC line is momentarily asserted.

returns None.

example This example initializes the KM-488-ROM as a System Controller with a IEEE address of 0 with a High Speed Handshake.

```
CALL INIT (0,4)
```

INTERM

purpose Changes the input message terminator settings.

usage `CALL INTERM(num%,term%)`

alternate usage `CALL SETINPUTEOS(term%)`

NOTE: The Alternate Syntax will only change the value of Input Message Terminator 0.

parameters **num%** is an integer which selects the number of the receive message terminator to be changed. This ranges from 0 to 3, where

num%	TERMINATOR #	DEFAULT
0	0	LF
1	1	CR
2	2	,
3	3	;

term% is an integer representing the terminator byte to be programmed. This integer is the decimal or hex equivalent of the terminator's ASCII representation. Hex equivalents must be preceded by &H. See Appendix A for ASCII Equivalents.

returns None.

example This example sets Input Terminator 3 to Line Feed (Hex A).

```
CALL INTERM(3,&HA)
```

IOTIMEOUT

purpose Changes the length of time to elapse before an I/O Timeout occurs.

usage

```
...
CALL IOTIMEOUT(time%)
...
```

IOTIMEOUT (cont.)

parameters **time%** is the time elapsed before a timeout error is reported. This occurs if the time elapsed between the transfer of individual bytes exceeds the specified I/O Timeout period. time% is between 0-65535 ms, internally rounded to the closest multiple of 55 ms. The default is 10 seconds.

returns None.

example This sets the I/O timeout to 1 second.

```
CALL IOTIMEOUT(1000)
```

OUTTERM

purpose Changes the output message terminator sequences.

usage `CALL OUTTERM(num%,chars%,eoi%,trm1%,trm2%)`

alternate usage `CALL SETOUTPUTEOS(trm1%,trm2%)`

NOTE: The Alternate Syntax will only change the value of Terminator 0, and will always assert EOI upon the transmission of the last character. In addition, a single character terminator is programmed by setting trm2% to 0.

parameters **num%** is an INTEGER which selects the number of the transmit message terminator to be changed. This ranges from 0 to 3, where:

num%	TERMINATOR #	DEFAULT
0	0	LF EOI
1	1	CR LF EOI
2	2	CR EOI
3	3	LF CR EOI

chars% is an INTEGER that selects the length of the transmit terminator. This is 0 if a 1-character terminator is required or 1 for a 2-character terminator.

eoi% is an INTEGER that determines whether EOI is asserted when the last terminator byte is sent. If this bit is 1, EOI will be sent. If this bit is 0, EOI will not be sent.

trm1% is an INTEGER representing the first terminator byte to be sent; it is the decimal or hex equivalent of the terminator's ASCII representation. Be sure to precede all hex values with &H. See Appendix A for ASCII Equivalents.

trm2% is an INTEGER representing the second terminator byte (in a 2-byte terminator); it is the decimal or hex equivalent of the terminator's ASCII representation. Be sure to precede all hex values with &H. If a 1-byte terminator is programmed, trm2% may be any value.

returns None.

example This example sets Output Terminator 1 to Carriage Return, Line Feed, EOI.

```
CALL OUTTERM(1,1,1,&HD,&HA)
```

PPOLL

purpose Initiates a parallel poll.

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

usage
```
CALL PPOLL(resp%)
```

parameters None.

returns **resp%** is an INTEGER which will contain the received parallel poll response.

notes Before you call the PPOLL routine, you must first configure the Parallel Poll response of the device. To do this:

- Address it to listen.
- Send it a GPIB Parallel Poll Configure (PPC) command.
- Send a Parallel Poll Enable byte using the KM-488-ROM XMIT command. (Use the mnemonic CMD followed by nnn where nnn is the decimal value of the Parallel Poll Enable byte.

The Parallel Poll Enable Byte is of the format 0110SPPP, 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.

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

example This example assumes that the KM-488-ROM is connected to a Sorenson HPD30-10 Power Supply. This device is located at GPIB address 1. It is also assumed that this device drives bit 4 of the Parallel Poll Response byte to a logic "1" when service is required. In order to retrieve this response, the device's parallel poll response must be configured to respond in this manner. This is accomplished by using the KM-488-ROM XMIT routine with the CMD command accompanied by the command byte 01101011 (107).

```
CALL init(0,0)
CALL xmit("REN UNL UNT LISTEN 1 PPC CMD 107", stat%)

IF stat%<>0 THEN PRINT "Error sending PPC command STATUS=", stat%

CALL ppoll(resp%)

IF (resp% AND 8)<>0 THEN PRINT "HPD30-10 Requesting Service..."
```

RCV

purpose Receives data into a string.

usage
```
...
info$=SPACE$(80)     'Allocate space for received data
CALL RCV(info$,term%,rcvlen%,stat%)
...
```

alternate usage
```
CALL RECEIVE(info$,rcvlen%,stat%)
```

RCV (cont.)

NOTE: The Alternate Syntax assumes the use of Input Message Terminator 0.

parameters **info$** is a STRING which will hold the received data. Prior to calling RCV, you must initialize a string which is long enough to receive the expected number of characters. This may be accomplished using the QuickBASIC SPACE$ function. For example, the line INFO$ = SPACE$(100) allocates a 100 character string for storing data. Carriage returns and the message terminator character in the incoming data are ignored and are not stored with the received data.

term% is an INTEGER containing the number of the IEEE bus terminator to be used, where:

term%	TERMINATOR #	DEFAULT
0	0	LF
1	1	CR
2	2	,
3	3	;

These terminators can be changed by calling the INTERM routine.

returns **info$** is a STRING variable (up to 64 KBytes) which will contain the received data. The length of the string must be long enough to receive the expected number of characters. RCV will terminate reception of data when: 1) the number of characters received exceeds the length of the string, 2) a terminator is received, or 3) any character is received with the EOI signal asserted. Carriage returns and the message terminator character in the incoming data are ignored and not stored with the received data.

rcvlen% is an INTEGER which indicates the actual number of bytes which were received and stored.

stat% is an INTEGER which describes the state of the transfer returned after the call. The returned stat values are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	OVF	NL	0

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

OVF Overflow Error. If this bit is a 1, then the info string was filled, before a terminator character or EOI was detected.

NL KM-488-ROM not a Listener. If this bit is set to a 1, it indicates the RCV was called before the KM-488-ROM was designated as a Listener.

notes The KM-488-ROM must be addressed to listen and another device addressed to talk before calling RCV.

RCV (cont.)

example The following example shows how the RCV routine might be used together with the XMIT routine to receive data. It demonstrates a method of triggering the Keithley 196 voltmeter to take a reading using the XMIT command and then receiving the meter reading using the RCV command. It is assumed that the meter reading returned will fit into a string of 25 characters. This example also assumes that the KM-488-ROM has been configured to use Transmit Message terminator 1 as a Carriage Return-Line Feed combination.

```
'$INCLUDE:'KM488QB.BI'
CALL INIT(0,0)

CALL XMIT("REN UNL UNT LISTEN 7 MTA DATA 'F0R3S1T3X' T1 GET UNL UNT
  TALK 7 MLA", STAT%)
IF STAT%<>0 THEN
     PRINT "Error sending cmd string status =";'STAT%
RCVDAT$=SPACE$(25)  'Allocate space for receive data
CALL RCV(RCVDAT$,0,RCVLEN%,STAT%)     'Use RCV terminator 0
DAT$=LEFT$(RCVDAT$,RCVLEN%)
IF STAT%<>0 THEN PRINT "RCV Status Error  Status =";STAT%
PRINT "Received data = "; DAT$;"Length=";RCVLEN%
```

RCVA

purpose Receives data into an array. This routine may also be used to receive data via DMA (See SETDMA.)

usage

```
...
DIM indata%(1000)    'allocate 1000 array locations
maxlen% = 2000
CALL RCVA(indata%(0),maxlen%,term%,rcvlen%,stat%)
...
```

alternate usage

```
CALL RARRAY(indata%, maxlen%,rcvlen%,stat%)
```

NOTE: The Alternate Syntax assumes the use of EOI as a terminator.

parameters **term%** is an INTEGER which selects the type of terminator to be used. This integer is interpreted according to the following format:

Term (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	STRM	TRM1	TRM0

Where

X May be any value.

STRM Enable/Disable String Message Terminators. If this bit is 1, a Message Terminator Character will be used to detect the end of reception. If this bit is 0, a Message Terminator Character will not be used.

TRM1-0 Terminator Select. These two bits select the Input Message Terminator to be used to signal the end of a transmission. (The STRM bit must be set to 1.) Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

The values for these terminators can be changed by calling the INTERM routine.

maxlen% is an integer which specifies the maximum number of data bytes which can be received. When you want to receive more than 32767 bytes, use the technique outlined in Programming Note 4 presented at the beginning of this section. maxlen% must be less than or equal to twice the total number of bytes allocated in the indata% array or a program crash may occur.

returns **indata%** is an array which will contain the received data. All characters received are stored.

rcvlen% is an integer which will contain the actual number of data bytes which were received. Note that half this many array locations will contain data. To specify more than 32767 bytes, use the technique outlined in Programming Note 4 presented at the beginning of this section.

stat% is an integer describing the state of the transfer returned after the call. The RCVA routine returns three status bits within the stat% variable. The TM0 bit is used to signal a timeout error. The REOI bit signals that the routine returned because the terminator was detected (if enabled), or EOI was received. The NL bit is set if the RCVA routine was called and the card was not addressed to listen. Unlike other KM-488-ROM routines, it is possible to return a non-zero status when the call was completed successfully.

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	REOI	0	TMO	0	NL	0

Where

REOI Reason for RCVA Termination. If this bit is a 1, then RCVA routine ceased because an EOI or terminator character was received. If this bit is a 0, then the RCVA was terminated because an error occurred or the maximum byte count was reached.

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

RCVA (cont.)

NL KM-488-ROM not a Listener. If this bit is set to a 1, it indicates the RCVA was called before the KM-488-ROM was designated as a Listener.

notes The KM-488-ROM must be addressed to listen before calling this routine.

example Refer to the XMITA example.

SEND

purpose Addresses a specified device to listen, the KM-488-ROM to talk, and sends data from a string.

usage
```
info$ = "data to be transmitted"
CALL SEND (adrs%,info$,status%)
```

parameters adrs% is an INTEGER containing the IEEE bus address of the device that the data is to be sent to and the terminator to be used. This byte is of the following format:

Adrs (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	TRM1	TRM0	0	ADR4	ADR3	ADR2	ADR1	ADR0

Where

TRM1-0 Terminator Select. These two bits select the Input Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

The values for these terminators can be changed by running the CONFIG program as described in Chapter 2.

The easiest way to specify an alternate terminator is to add a factor to the GPIB address of the desired instrument which is specified within the SEND call. The factors added for each terminator are as follows:

GPIB Address + 0	= Terminator 0
GPIB Address + 64	= Terminator 1
GPIB Address + 128	= Terminator 2
GPIB Address + 192	= Terminator 3

For example, if you wanted to send a message using message terminator 2 to a device at GPIB address 10, the value of adrs% supplied to SEND would be 138 decimal (10 + 128).

SEND (cont.)

ADR4-0 GPIB Address. These five bits are used to represent the GPIB address of the device to which the data is to be sent. GPIB addresses can range from 0 to 30.

info$ is a STRING (256 chars max.) containing the data to be sent.

returns **stat%** is an INTEGER describing the state of the transfer returned after the call. The returned stat values (or combination of) are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NC	ADRS

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NC Not Active Controller. If this bit is a 1, then the SEND routine was called when the KM-488-ROM was not an Active Controller.

ADRS Invalid Address. If this bit is set to a 1, an invalid IEEE-488 device address was given.

example This example shows how to send data from a KM-488-ROM to a device. The KM-488-ROM is initialized as a System Controller located at GPIB address 10. The KM-488-ROM uses high-speed handshaking. The data (a device setup string) is sent to a device located at GPIB address 12.

```
'$INCLUDE:'KM488QB'
CALL INIT(10,4)
SETUP$="FOROTOMOX" : ADRS%=12
CALL SEND(ADRS%,SETUP$,STAT%)
IF STAT <> 0 THEN PRINT "Error sending Status=";STAT%
```

As an alternative, the following sequence can be used:

```
'$INCLUDE:'KM488QB'
CALL INIT(10,4)       'Must init as a Sys Contr first
CALL
SEND(12,"FOROTOMOX",STAT%)
IF STAT <> 0 THEN PRINT "Error sending Status=";STAT%
```

SETBOARD

purpose In a multiple board system, identifies the KM-488-ROM to be programmed.

usage `CALL SETBOARD (board%)`

alternate usage `CALL BOARDSELECT(board%)`

SETBOARD (cont.)

parameters **board%** is an INTEGER between 0 and 3 which represents the board to be programmed. Note that up to four boards can be installed in any one system. The board% "number" is associated with the base address of its I/O port.

returns None.

notes You must assign a board "number" for every KM-488-ROM in the system before calling the SETBOARD routine. Board numbers are assigned using the SETPORT routine.

Each board must be must be initialized independently by calling the INIT routine. You must do this the first time a given board is selected, before any other operations are conducted on that board.

Once a board has been selected using SETBOARD, all further I/O operations will be performed on that board until the next SETBOARD is executed.

example This example line selects board "2" for communication.

```
SETBOARD (2)
```

SETDMA

NOTE: DMA allows maximum data transfer rates in excess of 100 kilobytes per second. However, the actual data rates are limited by the rates at which other devices connected to the bus can send or receive data. These rates are governed automatically by the GPIB handshaking signals.

purpose Allows the use of DMA in conjunction with the XMITA and RCVA routines.

usage `CALL SETDMA(channel%)`

alternate usage `CALL DMACHANNEL(channel%)`

parameters **channel%** is an INTEGER which specifies the DMA channel to be used for the data transfer. channel% can be from 1 to 3, where:

1 = Use DMA channel 1.
2 = Use DMA channel 2.
3 = Use DMA channel 3.

To disable DMA. set channel% to a value other than 1, 2 or 3.

returns None.

notes The DMA hardware jumpers must be properly set for the DMA channel selected by SETDMA. Note that the default setting for the jumpers is DMA DISABLED. The jumpers are further described in Chapter 2.

When SETDMA is called to enable the use of DMA, each call to the XMITA and RCVA routines that follows will use DMA to accomplish the transfer until SETDMA is called with a parameter outside the range of 1-3.

SETDMA (cont.)

example This example specifies that DMA transfers are to take place using DMA channel 1 and then DMA is disabled.

```
CALL SETDMA(1) 'enables dma transfers via channel 1

CALL SETDMA(0)  'disables dma transfers
```

SETINT

purpose Sets the KM-488-ROM's interrupt enable bits.

usage `CALL SETINT(intval%)`

parameters intval% is an INTEGER containing the address and value of the Interrupt Mask Register. This is interpreted as follows:

INTVAL (Input) - High Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	X	ADRS

Where

X May be any value.

ADRS If this bit is set to 0, bits 0 through 7 will be written to Interrupt Mask 1. If this bit is set to 1, bits 0 through 7 will be written to Interrupt Mask 2.

INTERRUPT MASK 1

INTVAL (Input) - Low Byte (ADRS = 0)

BIT	7	6	5	4	3	2	1	0
	0	0	GET	0	DEC	0	0	0

Where

GET When this bit is set to 1, an interrupt will be generated when a KM-488-ROM acting as a device received a GPIB GET (Group Execute Trigger) command while addressed to listen.

DEC When this bit is set to 1, an interrupt is generated when a Device Clear is received.

INTERRUPT MASK 2

INTVAL (Input) - Low Byte (ADRS = 1)

BIT	7	6	5	4	3	2	1	0
	0	SRQI	0	0	0	LOKC	REMC	ADSC

SETINT (cont.)

Where

SRQI When this bit is set to 1, an interrupt is generated when SRQ is received.

LOKC When this bit is set to 1, an interrupt is generated when the state of the Local Lockout bit changes.

REMC When this bit is set to 1, an interrupt is generated when the state of the Local/Remote bit changes.

ADSC When this bit is set to 1, an interrupt is generated when the state of the LA, TA, or CIC bits within the address status register changes.

returns None.

notes Be certain to assign the KM-488-ROM to an interrupt level before using this routine. Interrupt Levels are assigned by means of a jumper on the KM-488-ROM board. This jumper is described in detail in Chapter 2.

You must set-up an interrupt handling routine within the QuickBASIC program to deal with the interrupt condition.

example This example enables the KM-488-ROM to generate an interrupt when SRQ is received.

```
CALL SETINT(&H140)
```

SETPORT

purpose This routine is used to alter the range of addresses used by the KM-488-ROM's I/O port. In a multiple board environment, it is also used to associate a given range of I/O addresses with a board number.

usage `CALL SETPORT(board%,ioport%)`

parameters **board%** is an INTEGER between 0 and 3 which represents the board to be programmed. Note that up to four boards can be installed in any one system. The board% "number" is associated with the base address of its I/O ports.

ioport% is an INTEGER representing the I/O Base Address of the KM-488-ROM. The default Base Address is 2B8 Hex. The Base Address selected must match the one selected by the Base Address Switch on the KM-488-ROM. (See Chapter 2 for more information.)

returns None.

notes When multiple boards are used, each board must have its own unique base address. Any Base Address can be assigned to any board number, provided that none of the base addresses overlap.

example This line assigns Board 0 an I/O Base Address of 300h.

```
setport(0,&H300)
```

SETSPOLL

purpose Defines the Serial Poll Response of a KM-488-ROM acting as a device (non-Controller).

usage `CALL SETSPOLL (resp%)`

parameters **resp%** is an INTEGER describing the serial poll response and the state of the SRQ bit. This byte is of the following format:

Resp% (Input) - Low Byte

BIT	7	6	5	4	3	2	1	0
	SPR8	RSV	SPR6	SPR5	SPR4	SPR3	SPR2	SPR1

Where

SPR1-8 Bits 1 through 8 of this device's Serial Poll Response Byte.

RSV If this bit is 1, SRQ will be asserted to request servicing. Otherwise, SRQ will not be asserted.

returns None.

example This example illustrates a common use of SETSPOLL.

```
resp% = 0
    IF (error1 = true) THEN resp%=resp%+1    'check local errors
    IF (error2 = true) THEN resp%=resp%+2    'and set bits
    IF (error3 = true) THEN resp%=resp%+4
    IF (error4 = true) THEN resp%=resp%+8
    IF (error5 = true) THEN resp%=resp%+16
    IF (error6 = true) THEN resp%=resp%+32
    IF (error7 = true) THEN resp%=resp%+128
    IF (resp% <> 0) THEN resp%=resp%+&h40    'indicate srq asserted
CALL SETSPOLL(resp%)
```

SPOLL

purpose Performs a serial poll of the specified device.

usage `CALL SPOLL(adrs%,resp%,stat%)`

parameters **adrs%** is an INTEGER containing the IEEE bus address of the device that is to be serial polled. This can range from 0 to 30.

returns **resp%** is an INTEGER containing the serial poll response received. 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.

stat% is an INTEGER describing the state of the transfer returned after the call. The stat value is interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NC	ADR

SPOLL (cont.)

Where

TMO Indicates whether a Timeout Error occurred during data transfer. If a 1, then a Timeout Error occurred.

NC KM-488-ROM not a Controller. If set to a 1, it indicates the routine was called before the KM-488-ROM was designated as an Active Controller.

ADR Invalid GPIB Address. If this bit is set to 1, an invalid GPIB address was provided.

example This example illustrates a simple serial poll of a device located at GPIB address 10.

```
'$INCLUDE: 'km488qb.bi'
CALL init(0,0)
CALL spoll(10,RESP%, STAT%)
IF STAT%<>0 THEN
    PRINT "SPOLL status error   status="; STAT%
PRINT"Serial Poll Response="; RESP%
IF (RESP% AND &H40) <>0 THEN PRINT "Device Requesting Service"
```

SRQ

purpose Detects the presence of the GPIB SRQ signal.

usage

```
IF (SRQ%) THEN
```

parameters None.

returns The SRQ function returns a 0 or FALSE when not present, or a 1 or TRUE when present.

notes The value returned by SRQ is generally used within a conditional branch in an application program.

Note that after obtaining a TRUE response from SRQ, SRQ response is reset to FALSE even if the SRQ line is still active. In order to reset the SRQ response to TRUE, you must serial poll at least one device with a requesting service. Conducting a serial poll on a device requesting service resets its SRQ line. Then, if other devices were simultaneously asserting SRQ, the output of SRQ will be reset to TRUE. Otherwise, SRQ becomes TRUE on the next SRQ assertion.

example This example assumes that the KM-488-ROM is connected to an instrument located at GPIB address 1 and capable of requesting service via SRQ. When SRQ is detected, the SPOLL function is called and the serial poll response of the device is printed to the computer screen.

```
'$INCLUDE: 'km488qb.bi'
IF(SRQ%) THEN
  CALL spoll(1,pollresp%,stat%)
  IF (stat%=0) THEN
      PRINT "Error calling SPOLL-Status=";stat%
  ELSE
      PRINT"SRQ Received from Device 1-Poll Response=";pollresp%
    END IF
END IF
```

STATUS

purpose Returns the current setting of the requested status parameter.

usage `CALL STATUS(reg%,stat%)`

parameters reg% is an INTEGER containing the address of the register or configuration parameter to be queried. reg% must be passed as a variable. This value corresponds to a 4-bit field which specifies the status register or configuration parameter to be read. The format of the reg% byte is as follows:

Reg (input) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	ADR3	ADR2	ADR1	ADR0

Where

X May be any value.

ADR3-0 REGISTER/PARAMETER SELECT. This is a 4-bit field which specifies the status register or configuration parameter to be read. Registers and parameters are selected as follows:

ADR3	ADR2	ADR1	ADR0	REGISTER/PARAMETER
0	0	0	0	Address Status Reg
0	0	0	1	Interrupt Status 1 Reg
0	0	1	0	Interrupt Status 2 Reg
0	0	1	1	DMA Status Reg
0	1	0	0	Output Terminator 0
0	1	0	1	Output Terminator 1
0	1	1	0	Output Terminator 2
0	1	1	1	Output Terminator 3
1	0	0	0	Input Terminator 0
1	0	1	1	Input Terminator 1
1	1	1	0	Input Terminator 2
1	1	1	1	Input Terminator 3
1	0	0	0	I/O Timeout Parameter
1	0	0	1	DMA Timeout Parameter
1	1	1	0	I/O Port Address
1	1	1	1	GPIB Address of KM-488-ROM

returns reg% - When STATUS obtains the value of one of the four transmit message terminators, this variable will contain two flag bits which determine the length of the terminator and whether or not EOI is asserted with the last byte. When obtaining other parameters, reg% will retain its input value.

Reg (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	LEN	EOI

STATUS (cont.)

Where

LEN Terminator Length. If this bit is set to 0, then the terminator is one byte long. If this bit is set to 1, then the terminator is two bytes long.

EOI If this bit is set to 1, EOI is asserted when the last terminator byte is sent. Otherwise, EOI is not asserted.

stat% is an INTEGER describing the status bits for the register or the configuration parameter which was specified by the reg% parameter. Unless otherwise noted, the high byte of stat% is returned as 0.

Address Status Register

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	CIC	X	X	X	X	LA	TA	X

Where

X This bit may be any value.

CIC Active Controller. If this bit is set to 1, then the KM-488-ROM is a System Controller.

LA Listener. If this bit is set to 1, then the KM-488-ROM is a Listener.

TA Talker. If this bit is set to 1, then the KM-488-ROM is a Talker.

Interrupt Status Register 1

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	GET	X	DEC	X	X	X

Where

X This bit may be any value.

GET Group Execute Trigger. If this bit is set to 1, then a Group Execute Trigger command was received while the KM-488-ROM was a device.

DEC When this bit is set to 1, a Device Clear was received.

Interrupt Status Register 2

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	SRQ1	LOK	REM	X	X	X	ADSC

Where

X This bit may be any value.

STATUS (cont.)

SRQI When this bit is set to 1, it indicates SRQ was active. (Active Controller mode only.)

LOK When this bit is set to 1, the device was set to Local Lockout. (Device mode only.)

REM When this bit is set to 1, the device was configured for remote operation. (Device mode only.)

ADSC When this bit is set to 1, a change of the address status occurred (i.e., untalk to talk, device to active controller, etc.).

Transmit and Receive Message Terminator #1-4. Contains First and Last bytes of the message terminator. Input Terminators and Single Character Output Terminators are contained in the Least Significant Byte. In the case of a two character Output Terminator, the Most Significant Byte of this parameter is the first character sent.

DMA Timeout and I/O Timeout Parameters. Contains the value of the desired parameter as an unsigned value in the low and high bytes of stat%. The timeout value is expressed in milliseconds (0-65535).

notes The bits contained in the Interrupt Status 1 and 2 registers are extremely volatile. When you read these registers, any bits which were set are automatically cleared by the READ operation. This is extremely important to note when reading Interrupt Status Register 1, as some of the bits (not shown above) are used by various KM-488-ROM routines. It may be possible to cause various KM-488-ROM routines to report a timeout error if this register is read at while the KM-488-ROM is addressed to talk or listen.

example This example illustrates a possible use for the STATUS routine.

```
reg%=0 : stat% = 0
CALL STATUS(reg%,stat%)
   IF (stat% and &h80) <>0 THEN PRINT "KM-488-ROM = SYS CONTR"
   IF (stat% and 2) <>0 THEN PRINT PRINT "ADDRESSED TO TALK..."
   IF (stat% and 4) <>0 THEN PRINT  "ADDRESSED TO LISTEN..."
    reg%=15
CALL STATUS(REG%,STAT%)
PRINT "IEEE-488 Address = "; STAT%
```

XMIT

purpose Sends GPIB commands and data from a string.

usage `CALL XMIT(info$,stat%)`

parameters **info$** is a STRING variable containing a series of GPIB commands and data. Each item must separated by one or more spaces. All the available commands are described in Chapter 3. These commands include

XMIT (cont.)

parameters **info$** is a STRING variable containing a series of GPIB commands and data. Each item must be separated by one or more spaces. Commands can be in UPPER or lower case. The Transmit commands are described in Chapter 3. These commands include

CMD	GTL	MTA	SDC	T0
DATA	GTLA	MLA	SEC	T1
DCL	IFC	PPC	SPE	T2
END	LISTEN	PPD	SPD	T3
EOI	LLO	PPU	TALK	UNL
GET	LOC	REN	TCT	UNT

returns **stat%** is an INTEGER which describes the state of the transfer returned after the call. The returned stat value can be interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	ADRS	NCTL	UNDF	TMO	STR	NT	STX

Where

X May be any value.

ADRS Invalid GPIB address. If this bit is set to 1, then an invalid GPIB address was given.

NCTL Not a System Controller. If this bit is set to 1, it indicates that the KM-488-ROM tried to send GPIB Bus Commands when it was not an Active Controller.

UNDF Undefined Command. If this bit is set to 1, the info string contained an undefined command.

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

STR String Error. If this bit is set to one, then a quoted string, END, or terminator was found without a DATA subcommand preceding it.

NT KM-488-ROM not a Talker. If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as a Talker.

STX Syntax Error. If this bit is set to 1, a syntax error was found.

example This example illustrates one way to use the XMIT command with a Keithley 196 Voltmeter. This meter is assigned GPIB address 7 and is configured to a 30 Volt DC range with 4 1/2 digit accuracy. The meter is also configured to take a new reading each time a Group Execute Trigger Bus command (GET) is received. It is assumed that the meter has been set to use a CR, LF, EOI (the default for Message Terminator 1). The program then triggers the instrument to get the first reading, and makes it a talker and the KM-488-ROM a listener in order to get the first reading.

The device to receive the setup command string which must be sent to the meter contains the following device commands:

XMIT (cont.)

F0 Select DC Volts mode
R3 Select 30 Volt range
S1 Select 4 1/2 digit accuracy
T3 Take one reading when GET received
X Execute the prior commands within the string

The device to receive the setup command string must also be programmed to assert the GPIB REN signal (This allows the meter to receive GPIB commands.) and to LISTEN (This allows the device to receive the string.). The programming sequence used consists of the following:

- Setting Remote Enable (REN).
- Setting all devices to UNTalk and UNListen.
- Addressing the 196 to LISTEN.
- Addressing the KM-488-ROM to talk (My Talk Address).
- Sending the Device-Dependent Commands as a string of DATA.
- Sending the appropriate message terminator characters after the data.
- Issuing the Group Execute Trigger bus command.
- Unaddressing all devices.
- Addressing the meter to TALK and the KM-488-ROM to LISTEN (My Listen Address) in preparation for receiving the latest reading.

```
'$INCLUDE:'km488qb.bi'
CALL init(0,0)
CALL xmit("REN UNL UNT LISTEN 7 MTA DATA'F0R3S1T3X' T1 GET UNL UNT
TALK 7 MLA",STAT%)
IF STAT%<>0 THEN
   PRINT "Error sending cmd string status=";STAT%
```

XMITA

purpose Sends data from an array.

usage `CALL XMITA(outdat%(0),count%,term%,stat%)`

alternate usage `TARRAY(count%,term%,stat%)`

parameters **outdat%** is an INTEGER array containing the data to be sent.

count% is an INTEGER containing the number of data bytes to be transmitted. NOTE: In BASIC, when you want to send more than 32767 bytes, you will have to assign the value to count% in hex.

term% is an INTEGER that describes what sort of terminator should be used. This byte is of the following format:

Term (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	STRM	TRM1	TRM0	X	X	X	X	EOI

Where

X This bit may be any value.

XMITA (cont.)

STRM Send Message Terminators. If this bit is set to 1, then the message terminator(s) will be sent at the end of the transmission. Otherwise, they will not.

TRM1-0 Terminator Select. These two bits select the Output Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF EOI
0	1	1	CR LF EOI
1	0	2	CR EOI
1	1	3	LF CR EOI

These terminators can be redefined by running the CONFIG program as described in Chapter 2.

EOI Asserts EOI. If this bit is set to 1, then EOI will be asserted when the last byte is sent. Otherwise, EOI will not be asserted.

returns **stat%** is an INTEGER describing the state of the transfer returned after the call. The stat value is interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NT	0

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NT KM-488-ROM not a Talker. If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as a Talker.

example This example demonstrates use of SEND, XMITA, and RCVA to send and retrieve waveform data from a GPIB oscilloscope. The scope expects data points to be sent with the most significant byte first. Thus, the data bytes to be sent must be byte-swapped prior to sending them.

```
DECLARE SUB SwapBytes (B$)
DECLARE SUB DisplayKMerr (ErrStat%, ErrorStr$)

'$INCLUDE: 'km488qb.bi'

  DIM X%(1024), Y%(1024), Z%(1024), S%(1200)
  CLS : KEY OFF: COLOR 7, 0
  PRINT "This program demonstrates the use of the XMITA and RCVA"
  PRINT "routines using a Tektronix 11301 or 11302 Oscilloscope"
  PRINT "at GPIB address 16."

  CALL INIT(0,0)    'KM-488-ROM is system controller at GPIB adrs 0
  SCOPE% = 16       'Scope at adrs 16

  PRINT "INITIALIZING SCOPE": PRINT
```

XMITA (cont.)

```
CMD$ = "INIT"
CALL SEND(SCOPE%, CMD$, FLAG%) 'Initialize scope
IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

CMD$ = "RQS OFF"
CALL SEND(SCOPE%, CMD$, FLAG%)          'Disables SRQs
IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

'-------  Calculate data, send points to scope, display curve ----

PRINT "CALCULATING SINE WAVE": PRINT
NUMPTS% = 1024                                  'Words in waveform

FOR I = 0 TO NUMPTS% - 1
   ANGLE = I * 6.28319 / 1024: X%(I) = 400 * SIN(ANGLE)
   H$ = HEX$(X%(I))
   CALL SwapBytes(H$)
   Z%(I) = VAL(H$)
NEXT I

NUMBYT% = NUMPTS% * 2 + 1
H$ = HEX$(NUMBYT%)

CALL SwapBytes(H$)  'Scope wants MSB FIRST, KM-488-ROM sends LSB
                    'first so must swap bytes
N% = VAL(H$)

DO
   PRINT "SENDING SINE WAVE TO SCOPE": PRINT
   CMD$ = "DCL"
   CALL XMIT(CMD$, FLAG%)
   IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

   CMD$ = "INPUT STO1"
   CALL SEND(SCOPE%, CMD$, FLAG%)      'Store in location 1
   IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

   CMD$ = "MTA LISTEN " + STR$(SCOPE%) + " DATA 'CURVE %'"
   CALL XMIT(CMD$, FLAG%)               'Setup for data transfer
   IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

   EOI% = 0: COUNT% = 2
   CALL XMITA(N%, COUNT%, EOI%, FLAG%) 'Byte count sent
   IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, "")

  COUNT% = NUMPTS% * 2
   CALL XMITA(Z%(0), COUNT%, EOI%, FLAG%) 'Data sent
   IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

  CHKSUM% = 0                   'Don't bother actual checksum
  EOI% = 1: COUNT% = 1          'Send EOI with checksum
  CALL XMITA(CHKSUM%, COUNT%, EOI%, FLAG%) 'Checksum sent
  IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, "")

   PRINT "CLEARING ALL TRACES": PRINT
```

XMITA (cont.)

```
      CMD$ = "CLEAR ALL"
      CALL SEND(SCOPE%, CMD$, FLAG%)
      IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

      PRINT "DISPLAYING STORED TRACE": PRINT

      CMD$ = "TRACE1
DESCRIPTION:STO1,VPOSITION:0,HPOSITION:0,UNITS:" + CHR$(34) + "V" +
CHR$(34)
      CALL SEND(SCOPE%, CMD$, FLAG%)
      IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

'----------------  Retrieve data and compare --------------------

      PRINT "RETRIEVING DATA FROM SCOPE": PRINT

      CMD$ = "ENCDG WAVFRM:BINARY"
      CALL SEND(SCOPE%, CMD$, FLAG%)
      IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

      CMD$ = "OUTPUT STO1"
      CALL SEND(SCOPE%, CMD$, FLAG%)
      IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

      CMD$ = "CURVE ?"                'Ask for data to be returned

      CALL SEND(SCOPE%, CMD$, FLAG%)
      IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)

      CMD$ = "TALK " + STR$(SCOPE%) + " MLA"
      CALL XMIT(CMD$, FLAG%)          'Setup for scope to send data
      IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)


      COUNT% = 7:   L% = 0
      CALL RCVA(S%(0), COUNT%, 0, L%, FLAG%) 'Header received
      IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)
      IF L% <> 7 THEN GOTO 2080

      COUNT% = 2050:   L% = 0
      CALL RCVA(S%(0), COUNT%, 0, L%, FLAG%) 'Data received
      IF (FLAG% <> 0) THEN CALL DisplayKMerr(FLAG%, CMD$)
      IF L% <> (2 + NUMPTS% * 2) THEN GOTO 2080

      PRINT "COMPARE SENT AND RECEIVED DATA; . = OK, * = BAD
COMPARE": PRINT

      FOR I% = 1 TO 1024
      H$ = HEX$(S%(I%))
      CALL SwapBytes(H$)
      Y%(I% - 1) = VAL(H$)
      IF X%(I% - 1) = Y%(I% - 1) THEN PRINT "."; ELSE PRINT "*";
      NEXT I%

      PRINT : PRINT
      PRINT "COMPLETE"
```

```
      PRINT
      RESP% = 0
      CALL SPOLL(SCOPE%, RESP%, FLAG%)
      PRINT "SPOLL = "; HEX$(RESP% AND 255)

END

2080 PRINT "COUNT ERROR: "; L%
     STOP
2100 PRINT "READ DATA ERROR"
      STOP


SUB DisplayKMerr (ErrStat%, ErrorStr$)

 IF ErrStat% = 8 THEN
    PRINT "KM488 ERROR : "; ErrStat%; " - Timeout"
 ELSE
    PRINT "KM488 ERROR : "; ErrStat%
 END IF

 PRINT "LAST COMMAND - "; ErrorStr$
 END

END SUB


'---------------- Swap byte subroutine--------------------------'
SUB SwapBytes (B$)

   L% = LEN(B$)
   IF L% = 1 THEN B$ = "000" + B$
   IF L% = 2 THEN B$ = "00" + B$
   IF L% = 3 THEN B$ = "0" + B$

   LSB$ = RIGHT$(B$, 2)
   MSB$ = LEFT$(B$, 2)
   B$ = "&H" + LSB$ + MSB$                    'Swap bytes

END SUB



SUB WaitForKey
   WHILE INKEY$ = "": WEND
END SUB
```

■ ■ ■

Chapter 6

PROGRAMMING IN TURBO PASCAL

While Chapter 3 gives a brief overview of the routines available for programming the KM-488-ROM, this chapter gives instructions for calling the routines from TURBO PASCAL. The routines appear in alphabetical order and include a sample program for each.

6.1 GENERAL

TURBO PASCAL direct support is currently offered for versions 4.0 and 5.0. The interface for TURBO PASCAL includes four different files, as follows:

KM488PAS.TPU	"UNIT" file for use with Borland TURBO PASCAL version 5.0.
KM488P4.TPU	"UNIT" file for use with Borland TURBO PASCAL version 4.0.
KM488PAS.PAS	Source file to be used for re-building TURBO Pascal "UNIT" file.
KM488PAS.OBJ	Object code file to be used for re-building TURBO Pascal "UNIT" file.

The files KM488PAS.PAS and KM488PLB.OBJ can be used to create a new "unit" file should you need to.

Supported Versions

Turbo PASCAL versions 4.0, 5.0 and higher.

The Environment

Before you begin to develop programs in TURBO PASCAL, several files must be present in your working directory. Copy the appropriate files from the KM-488-ROM Disks to your working directory:

TURBO PASCAL 4.0	TURBO PASCAL 5.0
\turbopas\km488p4.tpu	\turbopas\km488pas.tpu

NOTE: km488p4.pas must be renamed km488pas.tpu.

File Header

Compiling

Your application program can be compiled in the usual fashion. Be sure to include the following line in your program:

```
USES km488pas;
```

Software

The KM-488-ROM firmware contains a number of configuration parameters which govern the default settings of the input and output message terminator settings, message timeout periods,

and I/O port addresses. The default terminators are shown in Table 4-3. If these default values are unsatisfactory, they may be changed by calling either the INTERM or OUTTERM routine.

The default DMA and I/O Timeouts are 10 seconds. These defaults may be altered by calling the DMATIMEOUT or IOTIMEOUT routine.

Default Terminator Settings

TERM #	OUTPUT TERMINATOR	INPUT TERMINATOR
0	LF EOI	LF
1	CR LF EOI	CR
2	CR EOI	, (comma)
3	LF CR EOI	; (semi-colon)

notes

1. Any arguments which appear as variables may also be passed as constants.

2. Parameters which are also used to return values must be declared as variables.

3. Any of the KM-488-ROM routines which are used to receive data require that a named 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, these routines require a parameter which specifies the maximum number of bytes to be received. 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-ROM routines which pass data to or from strings are limited to 255 bytes maximum.

5. Do not name the variables in your application program with the same name as any of the KM-488-ROM routines.

6. Do not assign a program name which is the same name as any of the KM-488-ROM routines.

6.2 DESCRIPTION FORMAT FOR ROUTINES

The format for each descriptions is as follows:

purpose ... a brief description of the routine. See Chapter 3 for more detailed descriptions.

usage ... gives an example of usage for each routine and assumes the input parameters are passed in as variables. These parameters can also be passed in directly. See the General Programming Notes for more

information.

alternate usage ... lists alternate usage for the routine, if any. Unless otherwise noted, the alternate usage performs exactly the same function as the usage.

parameters ... describes each of the input parameters.

returns ... describes any values returned by the routine.

notes ... lists any special programming considerations.

example ... gives a programming example using the routine.

6.3 ROUTINES

DMATIMEOUT

purpose Sets the maximum length of time for a DMA transfer to complete before a timeout error is reported, when using DMA in conjunction with XMITA and RCVA routines.

usage

```
...
VAR
   time: WORD;
...
BEGIN
...
dmatimeout(time);
...
```

alternate usage

```
...
settimeout (time);
...
```

NOTE: The alternate usage sets both the DMA and I/O Timeouts to the specified value.

parameters time is an INTEGER which represents the timeout period to elapse during a DMA transfer. A DMA Timeout Error will be generated when the time to transfer (via DMA) an entire message exceeds the set DMA timeout value (time). time% can range from 0 to 65535 milliseconds and is internally rounded to the closest integer multiple of 55 milliseconds.

returns None.

example This example sets the DMA Timeout period to 5 seconds.

```
dmatimeout(5000)
```

ENTER

purpose Addresses a specified device to talk, the KM-488-ROM to listen, and receives data from the addressed device into a string.

ENTER (cont.)

usage

```
...
VAR
   info : STRING;
   leng : WORD;
   maxlen : WORD;
   adrs : INTEGER;
   stat : INTEGER;
...
BEGIN
   enter(info,maxlen,leng,adrs,stat);
...
```

parameters **info** is a STRING which is to hold the receive data. The string must be long enough to receive the expected number of characters. Note that when you declare a variable to be a string in TURBO PASCAL, 255 bytes of string space is allocated. Carriage returns and the message terminator character in the incoming data are ignored and not placed in received data.

maxlen is a WORD which should be set to the maximum number of characters you expect to receive. It must never exceed the number of bytes of string space which have been allocated for storage of the received data.

adrs is an INTEGER containing the IEEE bus address of the device that the data is to be sent to and the terminator to be used. This byte is of the following format:

Adrs (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	TRM1	TRM0	0	ADR4	ADR3	ADR2	ADR1	ADR0

Where

TRM1-0 Terminator Select. These two bits select the Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

These terminators may be changed by the INTERM routine.

The easiest way to specify an alternate terminator is to add a factor to the GPIB address of the desired instrument which is specified within the ENTER call. The factors added for each terminator are as follows:

GPIB Address + 0 = Terminator 0
GPIB Address + 64 = Terminator 1
GPIB Address + 128 = Terminator 2
GPIB Address + 192 = Terminator 3

For example, if you wanted to receive a message using terminator 2 from a device at GPIB address 10, the value of adrs% supplied to ENTER would be 138 decimal (10 + 128).

ADR4-0 GPIB Address. These five bits are used to represent the GPIB address of the device to which the data is to be sent. GPIB addresses can range from 0 to 30.

returns **info** is a STRING variable, up to 255 characters, which will contain the received data. The length of the string must be long enough to receive the expected number of characters. Enter will terminate reception of data when: 1) the number of characters received exceeds the length of the string, 2) the specified terminator is received, or 3) any character is received with the EOI signal asserted. Carriage returns and the terminator character in the incoming data are ignored and not stored with the received data.

leng is an INTEGER, less than or equal to 255, which indicates the actual number of bytes which were stored.

stat is an INTEGER which describes the state of the transfer returned after the call. The returned stat values (or combination of) are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	OVF	NC	ADRS

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

OVF Overflow Error. If this bit is a 1, then the info string was filled, before a terminator character or EOI was detected.

NC KM-488-ROM not an Active Controller . If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as an Active Controller.

ADRS Invalid GPIB address. If this bit is set to 1, then an invalid GPIB address was given.

example In the following example, data is sent from two different instruments to a KM-488-ROM. The KM-488-ROM is acting as the System Controller and is assigned to GPIB address 0. One of the two instruments is a voltmeter, requiring a Carriage Return-Line Feed terminator combination (Term 1), assigned to GPIB address 7. The second instrument, located at GPIB address 10, requires a line feed (Term 0) as its terminator. The voltmeter is first sent a string of data which represents its instrument setup command. Then, when addressed to talk, it sends its most current reading to the KM-488-ROM. The second instrument is instructed to send its status, when addressed to talk.

It is assumed that the string sent by both instruments is 25 characters or less. The string is printed out on the computer screen.

ENTER (cont.)

```
PROGRAM senddemo;
USES    KM488PAS;

VAR

    inst1  :INTEGER;
    inst2  :INTEGER;
    inst1term    :INTEGER;
    rlen   :WORD;
    status :INTEGER;
    instring     :STRING;
{Note declaring a STRING gives us up to 255 bytes of data storage
for receiving.  Strings used to receive data should not be
dimensioned....}

BEGIN
    inst1:=7;
    inst2:=10;
    {Note terminator 1 used to send to Instrument 1}
    inst1term:=inst1+64;

    init(0,4);
    send(inst1term,'F0R0T0M0X',status);

    IF (status<>0) THEN
       WriteLn ('Error sending to Instrument 1 Status=' status);
    {Note that we need to specify the expected maximum number of
    bytes to be received within the ENTER call...}

    enter(instring,25,rlen,inst1,status);   {Read instrument 1}

    {Check status of call and print received value if all is ok}

    IF (status<>0) THEN
       WriteLn('Error receiving Instrument 1 Status='status)
    ELSE
       WriteLn('Data received from Instrument 1 -',instring);

    {Get data from second instrument...}

    send(inst2,'SEND STATUS',status);

    IF (status<>0) THEN
       WriteLn('Error sending to Instrument 2 Status='status)
    ELSE
       WriteLn('Data received from Instrument 2 ',instring);
END.
```

INIT

purpose Initializes the KM-488-ROM by assigning its GPIB address and establishing it as a System Controller or Device.

usage

```
...
VAR
   adrs: INTEGER;
   mode: INTEGER;
...
BEGIN
   adrs:= ;
   mode:= ;
   INIT (adrs, mode);
...
```

alternate usage `INITIALIZE(adrs, mode)`

parameters adrs is an INTEGER representing the IEEE bus address of the KM-488-ROM. This is an integer from 0 to 30.

mode is an INTEGER representing the operating mode of the KM-488-ROM. These can be any of the following values:

Mode - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	FAST	DEV	X

Where

X May be any value.

FAST Handshake Speed. If set to 1, High Speed GPIB bus handshaking is used (500ns). If set to 0, Low Speed GPIB bus handshaking (2 us) is used. See Chapter 3 for more information.

DEV Device. If set to 1, the KM-488-ROM is acting as a Device. Otherwise, the KM-488-ROM is acting as a System Controller. When System Controller is selected, the GPIB IFC line is momentarily asserted.

returns None.

example This example initializes the KM-488-ROM as a System Controller with a IEEE address of 0 with a High Speed Handshake.

```
...
PROGRAM  userprog;
USES   KM488PAS;
...
VAR
    adrs: INTEGER;
    mode: INTEGER;
...
BEGIN
    adrs:=0;
    mode:=4;
    init (adrs,mode);
...
```

INTERM

purpose Changes the input message terminator settings.

usage

```
...
VAR  num : INTEGER;
  term : INTEGER;
...
BEGIN
  interm(num, term);
...
```

alternate usage

```
...
VAR  term : BYTE;
...
BEGIN
  setinputeos(term);
...
```

NOTE: The alternate syntax only changes the value of Input Terminator 0.

parameters num is an integer which selects the number of the receive message terminator to be changed. This ranges from 0 to 3, where

num%	TERMINATOR #	DEFAULT
0	0	LF
1	1	CR
2	2	,
3	3	;

term is an integer representing the terminator byte to be programmed. This integer is the decimal or hex equivalent of the terminator's ASCII representation. Hex equivalents must be preceded by &H. See Appendix A for ASCII Equivalents.

returns None.

notes The parameters may be passed directly into the routine.

example This example sets Input Terminator 0 to Line Feed and Input Terminator 3 to Carriage Return.

```
VAR
...
BEGIN
      interm (0,10);
      interm(3,$D);
...
```

IOTIMEOUT

purpose Changes the length of time to elapse before an I/O Timeout occurs.

usage

```
...
VAR
   time: WORD;
...
BEGIN
   iotimeout(time);
...
```

parameters time is the amount of time to elapse before a timeout error is reported. This will occur if the time elapsed between the transfer of individual bytes exceeds the specified I/O Timeout period (time). time is any value between 0 and 65535 milliseconds and will be internally rounded to 55 milliseconds. The default timeout value is 10 seconds.

returns None.

example This line sets the I/O Timeout period to 1 second.

```
...
iotimeout(1000);
```

OUTTERM

purpose Changes the output message terminator sequences.

usage

```
...
VAR
   num : INTEGER;
   chars : INTEGER;
   eoi : INTEGER;
   trm1 : INTEGER;
   trm2 : INTEGER;
...
BEGIN
   outterm(num, chars, eoi, trm1, trm2)
...
```

alternate usage

```
...
VAR
   trm1 : BYTE;
   trm2 : BYTE;
...
BEGIN
   setoutputeos(trm1, trm2)
...
```

NOTE: The Alternate usage will only change the value of Terminator 0 and will always assert EOI upon transmission of the last character. Additionally, a single terminator is programmed by setting trm2 to 0.

parameters num is an INTEGER which selects the number of the transmit message terminator to be changed. This ranges from 0 to 3, where

OUTTERM (cont.)

num%	TERMINATOR #	DEFAULT
0	0	LF EOI
1	1	CR LF EOI
2	2	CR EOI
3	3	LF CR EOI

chars is an INTEGER that selects the length of the transmit terminator. This is 0 if a 1-character terminator is required or 1 for a 2-character terminator.

eoi is an INTEGER that determines whether EOI is asserted when the last terminator byte is sent. If this bit is 1, EOI will be sent. If this bit is 0, EOI will not be sent.

trm1 is an INTEGER representing the first terminator byte to be sent; it is the decimal or hex equivalent of the terminator's ASCII representation. Be sure to precede all hex values with &H. See Appendix A for ASCII Equivalents.

trm2 is an INTEGER representing the second terminator byte (in a 2-byte terminator); it is the decimal or hex equivalent of the terminator's ASCII representation. Be sure to precede all hex values with &H. If a 1-byte terminator is programmed, trm2% may be any value.

returns None.

example These lines illustrate two different uses of OUTTERM.

```
OUTTERM (0,0,1,$D,0)      {Sets output terminator 0 to CR EOI.}

OUTTERM (3,1,0,$D,$A)     {Sets output terminator 3 to CR,LF with no EOI.}
```

PPOLL

purpose Initiates a parallel poll.

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

usage

```
...
VAR
   resp : byte;
...
BEGIN
   ppoll(resp);
...
```

parameters None.

returns **resp** is a BYTE which will contain the parallel poll response.

notes Before you call the PPOLL routine, you must first configure the Parallel Poll response of the device. To do this,

1. Address it to listen.

2. Send it a GPIB Parallel Poll Configure (PPC) command, using the XMIT command.

PPOLL (cont.)

3. Send a Parallel Poll Enable byte using the XMIT command. (Use the mnemonic CMD followed by nnn where nnn is the decimal value of the Parallel Poll Enable byte.

The Parallel Poll Enable Byte is of the format **0110SPPP**, 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.

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

example This example assumes that the KM-488-ROM is connected to a Sorenson HPD30-10 Power Supply. This device is located at GPIB address 1. It is also assumed that this device drives bit 3 of the Parallel Poll Response byte to a logic "1" when service is required. To program the device to respond properly, send the Parallel Poll enable byte 01101011 (107) via the XMIT command.

```
PROGRAM   ppolldemo;
USES  KM488PAS;

VAR

     stat : INTEGER;
     resp : BYTE;

BEGIN
     init(0,0);
     xmit('REN UNL UNT LISTEN 1 PPC CMD 107', stat);
     IF (stat<>0) THEN
        WriteLn('Error sending PPC cmd  Status=', stat);

     ppoll(resp);
     IF ((resp AND 8)<>0) THEN
        WriteLn('HPD30-10 Requesting Service...');
END.
```

RCV

purpose Receives data into a string.

usage

```
VAR
   info : STRING;
   maxlen : WORD;
   term : INTEGER;
   rcvlen : WORD;
   stat : INTEGER;
...
BEGIN
   rcv(info, maxlen, term, rcvlen, stat);
...
```

alternate usage

```
receive(info, maxlen, rcvlen, stat);
```

NOTE:The Alternate usage assumes the use of Input Message Terminator 0.

RCV (cont.)

parameters **info** is a STRING which will hold the received data. The string must be long enough to receive the expected number of characters. Carriage returns and the message terminator character in the incoming data are ignored and are not stored with the received data.

maxlen is a WORD which specifies the maximum number of data bytes which can be received. maxlen must not exceed the actual number of storage locations that have been allocated to store data. Otherwise, data may be stored in locations other than those allocated for your program and your program may crash.

term is an INTEGER containing the number of the IEEE bus terminator to be used, where:

term%	TERMINATOR #	DEFAULT
0	0	LF
1	1	CR
2	2	,
3	3	;

These terminators can be changed by calling the INTERM routine.

returns **info** is a STRING variable (up to 64 KBytes) which will contain the received data. The length of the string must be long enough to receive the expected number of characters. RCV will terminate reception of data when: 1) the number of characters received exceeds maxlen; 2) a

terminator is received; or 3) any character is received with the EOI signal asserted. Carriage returns and the message terminator character in the incoming data are ignored and not stored with the received data.

rcvlen is a WORD that indicates the actual number of bytes which were received and stored.

stat is an INTEGER which describes the state of the transfer returned after the call. The returned stat values are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	OVF	NL	0

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

OVF Overflow Error. If this bit is a 1, then the info string was filled, before a terminator character or EOI was detected.

NL KM-488-ROM not a Listener. If this bit is set to a 1, it indicates the RCV was called before the KM-488-ROM was designated as a Listener.

notes The KM-488-ROM must be addressed to listen and another device addressed to talk before calling RCV.

RCV (cont.)

example This example shows how the RCV routine might be used together with the XMIT routine to receive data. It uses the XMIT routine to command a Keithley 196 voltmeter to take a reading. The meter reading is received using the RCV routine. It is assumed that the meter reading returned will fit into a 25-character array.

This example assumes that the KM-488-ROM has been configured such that transmit message terminator 1 is Carriage Return-Line Feed combination and this combination is also used by the Keithley 196.

Note that the Voltmeter's setup command string is enclosed within double single quotes (").

```
PROGRAM   xmitdemo;
USES      KM488PAS;

VAR
    status:INTEGER;
    rcvdat:STRING;
    rcvlen:WORD;
BEGIN
    init(0,0);
    xmit('ren unl unt listen 7 mta data ''F0R3S1T3X'' T1 get unl
     unt talk 7 mla',status);

if (status<>0) then
    WriteLn('Error sending -status=',status);

rcv(rcvdat,25,0,rcvlen,status);
{Note that we expect to receive no more than 25 characters.}

if (status<>0) then
    WriteLn('Error receiving - status=',status)
else
    WriteLn('Received data=',rcvdat,' Length=',rcvlen);

END.
```

RCVA

purpose Receives data into a specified array. It may also be used to receive data via DMA (See SETDMA).

usage

```
...
VAR
   data : TYPE[LENGTH]
   maxlen : WORD;
   term : INTEGER;
   rcvlen : WORD;
   stat : INTEGER;
...
BEGIN
...
   rcva(data[0],maxlen,term,rcvlen,stat);
...
```

RCVA (cont.)

alternate usage `rarray(data[0],maxlen,rcvlen,stat);`

NOTE: The Alternate usage is limited to terminating on EOI.

parameters **data[ ]** is an ARRAY which is used to store the received data. It may be any data type. The number of data bytes contained in each array location will vary according to the data type specified. The RCVA routine will "byte pack" data into the array, starting with the least significant byte of the specified location. The size of the array should be large enough to store the expected number of data bytes or a program crash could occur.

term is an INTEGER which selects the type of terminator to be used. This integer is interpreted according to the following format:

Term (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	STRM	TRM1	TRM0

Where

X May be any value.

STRM Enable/Disable String Message Terminators. If this bit is 1, a Message Terminator Character will be used to detect the end of reception. If this bit is 0, a Message Terminator Character will not be used.

TRM1-0 Terminator Select. These two bits select the Input Message Terminator to be used to signal the end of a transmission. (The STRM bit must be set to 1.) Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

The values for these terminators can be changed by calling the INTERM routine.

maxlen is an integer which specifies the maximum number of data bytes which can be received. When you want to receive more than 32767 bytes, use the technique outlined in Programming Note 4 presented at the beginning of this section. maxlen% must be less than or equal to twice the total number of bytes allocated in the indata% array or a program crash may occur.

returns **rcvlen** is a WORD that will contain the actual number of data bytes which were received. Note that half this many array locations will contain data. To specify more than 32767 bytes, use the technique outlined in Programming Note 4 presented at the beginning of this section.

RCVA (cont.)

stat is an integer describing the state of the transfer returned after the call. The RCVA routine returns three status bits within the stat% variable. The TM0 bit is used to signal a timeout error. The REOI bit signals that the routine returned because the terminator was detected (if enabled), or EOI was received. The NL bit is set if the RCVA routine was called and the card was not addressed to listen. Unlike other KM-488-ROM routines, it is possible to return a non-zero status when the call was completed successfully.

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	REOI	0	TMO	0	NL	0

Where

REOI Reason for RCVA Termination. If this bit is a 1, then RCVA routine ceased because an EOI or terminator character was received. If this bit is a 0, then the RCVA was terminated because an error occurred or the maximum byte count was reached.

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NL KM-488-ROM not a Listener. If this bit is set to a 1, it indicates the RCVA was called before the KM-488-ROM was designated as a Listener.

notes The KM-488-ROM must be addressed to listen before calling this routine.

example Refer to the XMITA example.

SEND

purpose Addresses a specified device to listen, the KM-488-ROM to talk, and sends data from a string.

usage

```
...
VAR
   adrs: INTEGER;
   info: STRING[25];
   stat: INTEGER;
...
BEGIN
   info := 'Data to be transmitted';
   send (adrs,setup,stat);
...
```

parameters adrs is an INTEGER containing the IEEE bus address of the device that the data is to be sent to and the terminator to be used. This byte is of the following format:

SEND (cont.)

Adrs (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	TRM1	TRM0	0	ADR4	ADR3	ADR2	ADR1	ADR0

Where

TRM1-0 Terminator Select. These two bits select the Input Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF EOI
0	1	1	CR LF EOI
1	0	2	CR EOI
1	1	3	LF CR EOI

Terminator values may be changed by calling the OUTTERM routine.

The easiest way to specify an alternate terminator is to add a factor to the GPIB address of the desired instrument. Factors for each terminator are as follows:

GPIB Address + 0 = Terminator 0
GPIB Address + 64 = Terminator 1
GPIB Address + 128 = Terminator 2
GPIB Address + 192 = Terminator 3

For example, if you wanted to send a message using message terminator 2 to a device at GPIB address 10, the value of adrs% supplied to SEND would be 138 decimal (10 + 128).

ADR4-0 GPIB Address. These five bits are used to represent the GPIB address of the device to which the data is to be sent. GPIB addresses can range from 0 to 30.

info is a STRING containing the data to be sent.

returns **stat** is an INTEGER describing the state of the transfer returned after the call. The returned stat values (or combination of) are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NC	ADRS

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NC Not Active Controller. If this bit is a 1, then the SEND routine was called when the KM-488-ROM was not an Active Controller.

SEND (cont.)

ADRS Invalid Address. If this bit is set to a 1, an invalid IEEE-488 device address was given.

example This example shows how to send data from a KM-488-ROM to a device. The KM-488-ROM is initialized as a System Controller located at GPIB address 10. The KM-488-ROM uses high-speed handshaking. The data (a device setup string) is sent to a device located at GPIB address 2.

```
PROGRAM    senddemo;
USES       KM488PAS;

VAR
   adrs:INTEGER;
   setup:STRING(9);
   stat:INTEGER;

BEGIN
   adrs:=12;
   setup:='F0R0T0M0X';
   init(10,0);

   {Set KM-488-ROM as System Controller at GPIB adrs 10}

   send(adrs,setup,stat);

   {or, SEND can be specified with parameters directly---
      send(12,'F0R0T0M0X',stat); }

   if (stat<>0) then
     WriteLn('Error sending-status=',stat);
END.
```

SETBOARD

purpose In a multiple board system, identifies the KM-488-ROM to be programmed.

usage

```
...
VAR
   board : integer;
...
BEGIN
   setboard(board);
...
```

alternate usage

```
boardselect(board);
```

parameters **board** is an INTEGER between 0 and 3 which represents the board to be programmed. Note that up to four boards can be installed in any one system. The board "number" is associated with the base address of its I/O port.

returns None.

notes You must assign a board "number" for every KM-488-ROM in the system before calling the SETBOARD routine. Board numbers are assigned using the SETPORT routine.

SETBOARD (cont.)

Each board must be must be initialized independently by calling the INIT routine. You must do this the first time a given board is selected before any other operations may be conducted on that board.

Once a board has been selected using SETBOARD, all further I/O operations will be performed on that board until the next SETBOARD is executed.

example This example select Board 0 and then Board 3 for communications.

```
setboard(0)     {Calls which follow transfer to/from board 0.}
...
setboard(3)     {Calls which follow transfer to/from board 3.}
```

SETDMA

NOTE: DMA allows maximum data transfer rates in excess of 100 kilobytes per second. However, the actual data rates are limited by the rates at which other devices connected to the bus can send or receive data. These rates are governed automatically by the GPIB handshaking signals.

purpose Allows the use of DMA in conjunction with XMITA and RCVA.

usage

```
...
VAR
   channel : integer;
...
BEGIN
   setdma(channel)
...
```

alternate usage `dmachannel(channel)`

parameters channel is an INTEGER which specifies the DMA channel to be used for the transfer, where

1 = Select DMA channel 1.
2 = Select DMA channel 2.
3 = Select DMA channel 3.

To disable DMA, set channel to a value other than 1,2, or 3.

returns None.

notes The DMA hardware jumpers must be properly set for the DMA channel selected by SETDMA. Note that the default setting for the jumpers is DMA DISABLED. The jumpers are further described in Chapter 2.

When SETDMA is called to enable the use of DMA, each call to the XMITA and RCVA routines that follows will use DMA to accomplish the transfer until SETDMA is called with a parameter outside the range of 1-3.

SETDMA (cont.)

example This example specifies that DMA transfers are to take place using DMA Channel 1 and then disables DMA.

```
setdma(1)
{Transfers initiated by RCVA & XMITA occur on DMA channel 1.}
...
setdma(0)  {Disables DMA.}
...
```

SETINT

purpose Sets the KM-488-ROM's interrupt enable bits.

usage

```
...
VAR
   intval : INTEGER;
...
BEGIN
   setint(intval);
...
```

parameters intval is an integer containing the address and value of the Interrupt Mask Register which is to be written to. This is interpreted as follows:

INTVAL (Input) - High Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	X	ADRS

Where

X May be any value.

ADRS If this bit is set to 0, bits 0 through 7 will be written to Interrupt Mask 1. If this bit is set to 1, bits 0 through 7 will be written to Interrupt Mask 2.

INTERRUPT MASK 1

INTVAL (Input) - Low Byte (ADRS = 0)

BIT	7	6	5	4	3	2	1	0
	0	0	GET	0	DEC	0	0	0

Where

GET When this bit is set to 1, an interrupt will be generated when a KM-488-ROM acting as a device received a GPIB GET (Group Execute Trigger) command while addressed to listen.

DEC When this bit is set to 1, an interrupt is generated when a Device Clear is received.

SETINT (cont.)

INTERRUPT MASK 2

INTVAL (Input) - Low Byte (ADRS = 1)

BIT	7	6	5	4	3	2	1	0
	0	SRQI	0	0	0	LOKC	REMC	ADSC

Where

SRQI When this bit is set to 1, an interrupt is generated when SRQ is received.

LOKC When this bit is set to 1, an interrupt is generated when the state of the Local Lockout bit changes.

REMC When this bit is set to 1, an interrupt is generated when the state of the Local/Remote bit changes.

ADSC When this bit is set to 1, an interrupt is generated when the state of the LA, TA, or CIC bits within the address status register changes.

returns None.

notes Be certain to assign the KM-488-ROM to an interrupt level before using this routine. Interrupt Levels are assigned by means of a jumper on the KM-488-ROM board. This jumper is described in detail in Chapter 2.

You must set-up an interrupt handling routine within the QuickBASIC program to deal with the interrupt condition.

example This example enables the KM-488-ROM to generate an interrupt when SRQ is received.

```
SETINT (0,140)
```

SETPORT

purpose This routine is used to alter the range of addresses used by the KM-488-ROM's I/O Port. In a multiple board environment, it is also used to associate a given range of I/O addresses with a board number.

usage

```
...
VAR
   board : integer;
   ioport : word;
...
BEGIN
   setport(board,ioport);
...
```

parameters board is an INTEGER between 0 and 3 which represents the board to be programmed. Note that up to four boards can be installed in any one system. The board "number" is associated with the base address of its I/O port.

SETPORT (cont.)

ioport an INTEGER representing the I/O Base Address of the KM-488-ROM. The default Base Address is 2B8 hex. The Base Address selected must match the one selected by the Base Address Switch on the KM-488-ROM. (See Chapter 2 for more information.)

returns None.

notes When multiple boards are used, each board must have its own unique base address. Any base address can be assigned to any board number provided that none of the base addresses overlap.

example This line assigns Board 0 a Base address of 300 hex.

```
setport(0,$300)
```

SETSPOLL

purpose Sets the Serial Poll Response of the KM-488-ROM, when it is acting as a Device (non-Controller).

usage

```
...
VAR
   resp : INTEGER;
...
BEGIN
   setspoll(resp);
...
```

parameters **resp** is an INTEGER describing the serial poll response and the state of the SRQ bit. This byte is of the following format:

Resp% (Input) - Low Byte

BIT	7	6	5	4	3	2	1	0
	SPR8	RSV	SPR6	SPR5	SPR4	SPR3	SPR2	SPR1

Where

SPR1-8 Bits 1 through 8 of this device's Serial Poll Response Byte.

RSV If this bit is 1, SRQ will be asserted to request servicing. Otherwise, SRQ will not be asserted.

returns None.

SETSPOLL (cont.)

example This example illustrates a common use of SETSPOLL.

```
VAR
    resp:INTEGER
    err1:BOOLEAN
    err2:BOOLEAN
    err3:BOOLEAN
    err4:BOOLEAN
    err6:BOOLEAN
    err7:BOOLEAN
BEGIN
    resp:=0;
          IF (err1=TRUE)
             resp=resp+1;
.
.
.
          IF (err7=TRUE)
             resp=resp + 128;
          IF(resp <>0)
             resp = resp + $40; {Set RSV, if any error}
    setspoll(resp);
```

SPOLL

purpose Initiates a serial poll of the specified device.

usage

```
...
VAR
   adrs : INTEGER
   resp : BYTE;
   stat : INTEGER;
...
BEGIN
   spoll(adrs, resp, stat);
...
```

parameters adrs an INTEGER containing the IEEE bus address of device to be serial polled. Can range from 0 to 30.

returns resp a BYTE containing the serial poll response. The definition of resp varies device; however, Bit 6 always indicates whether the device needs service. Consult the manufacturer's operator's manual for more information.

stat is an INTEGER describing the state of the transfer returned after the call, as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NC	ADR

SPOLL (cont.)

Where

TMO Indicates whether a Timeout Error occurred during data transfer. If a 1, then a Timeout Error occurred.

NC KM-488-ROM not a Controller. If set to a 1, it indicates the routine was called before the KM-488-ROM was designated as an Active Controller.

ADR Invalid GPIB Address. If this bit is set to 1, an invalid GPIB address was provided.

example This example illustrates a simple serial poll of a device located at GPIB address 10.

```
PROGRAM spolldemo;
USES KM488PAS;
VAR
   stat:INTEGER;
   resp: BYTE;

BEGIN
   init(0,0);
   spoll(10,resp,stat);
   if (status<>0) then
     WriteLn('SPOLL Status Error  status=', stat);

   WriteLn('Serial Poll Response=', resp);
   if ((resp and $40)<>0) then
     WriteLn('Device Requesting Service');
END.
```

SRQ

purpose Detects the presence of the GPIB SRQ signal.

usage `IF (SRQ) THEN`

parameters None.

returns The SRQ function returns a 0 or FALSE condition when SRQ has not been detected, or a 1 or TRUE condition when SRQ is present.

notes The value retruned by the SRQ function is generally used within a conditional branch in an applicaiton program.

Note that once you have obtained a TRUE response from the SRQ function, the SRQ response will be reset to FALSE even if the SRQ line is still active. In order to reset the SRQ response to TRUE, you must serial poll at least one device which was requesting service. Conducting a serial poll on a device which was requesting service will reset its SRQ line. At this time, if other devices were simultaneously asserting SRQ, the output of the SRQ function would once again be reset to TRUE. Otherwise, the SRQ function would become TRUE on the next assertion of the SRQ line.

SRQ (cont.)

example This example assumes that the KM-488-ROM is connected to an instrument located at GPIB address 1 which is capable of requesting service via the SRQ. When the SRQ is detected, the SPOLL function will be called and the serial poll response of the device will be printed to the computer screen.

```
PROGRAM  srqdemo;
USES  KM488PAS;

VAR
      resp : BYTE;
      stat : INTEGER;

BEGIN
      IF (srq) THEN
           BEGIN
           spoll(1,resp,stat);
           IF (stat<>0) THEN
                WriteLn('Error calling SPOLL - Status=',stat);
           ELSE
                WriteLn('SRQ received from Device 1 - Poll Response
=',resp);
           END
END.
```

STATUS

purpose Returns the value of the specified setup parameter.

usage

```
VAR
   reg : INTEGER;
   stat : INTEGER;
...
BEGIN
   status(reg,stat)
...
```

parameters **reg** is an INTEGER containing the address of the register or configuration parameter to be queried. You must pass this parameter into the routine as a variable. This value corresponds to a 4-bit field which specifies the status register or configuration parameter to be read. The format of the reg byte is as follows:

Reg (input) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	ADR3	ADR2	ADR1	ADR0

Where

X May be any value.

ADR3-0 REGISTER/PARAMETER SELECT. A 4-bit field that specifies the status register or configuration parameter to be read. Registers and parameters are selected as follows:

STATUS (cont.)

ADR3	ADR2	ADR1	ADR0	REGISTER/PARAMETER
0	0	0	0	Address Status Reg
0	0	0	1	Interrupt Status 1 Reg
0	0	1	0	Interrupt Status 2 Reg
0	0	1	1	DMA Status Reg
0	1	0	0	Output Terminator 0
0	1	0	1	Output Terminator 1
0	1	1	0	Output Terminator 2
0	1	1	1	Output Terminator 3
1	0	0	0	Input Terminator 0
1	0	1	1	Input Terminator 1
1	1	1	0	Input Terminator 2
1	1	1	1	Input Terminator 3
1	0	0	0	I/O Timeout Parameter
1	0	0	1	DMA Timeout Parameter
1	1	1	0	I/O Port Address
1	1	1	1	GPIB Address of KM-488-ROM

returns **reg** - When STATUS obtains the value of one of the four transmit message terminators, this variable will contain two flag bits which determine the length of the terminator and whether or not EOI is asserted with the last byte. When obtaining other parameters, reg% will retain its input value.

Reg (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	LEN	EOI

Where

LEN Terminator Length. If set to 0, then the terminator is one byte long. If set to 1, then the terminator is two bytes long.

EOI If this bit is set to 1, EOI is asserted when the last terminator byte is sent. Otherwise, EOI is not asserted.

stat an INTEGER describing the status bits for the register or the configuration parameter specified by the reg% parameter. Unless otherwise noted, the high byte of stat% is returned as 0.

Address Status Register

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	CIC	X	X	X	X	LA	TA	X

Where

X This bit may be any value.

CIC Active Controller. If set to 1, then the KM-488-ROM is a System Controller.

LA Listener. If this bit is set to 1, then the KM-488-ROM is a Listener.

STATUS (cont.)

TA Talker. If this bit is set to 1, then the KM-488-ROM is a Talker.

Interrupt Status Register 1

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	GET	X	DEC	X	X	X

Where

X This bit may be any value.

GET Group Execute Trigger. If this bit is set to 1, then a Group Execute Trigger command was received while the KM-488-ROM was a device.

DEC When this bit is set to 1, a Device Clear was received.

Interrupt Status Register 2

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	SRQ1	LOK	REM	X	X	X	ADSC

Where

X This bit may be any value.

SRQI When this bit is set to 1, it indicates SRQ was active. (Active Controller mode only.)

LOK When this bit is set to 1, the device was set to Local Lockout. (Device mode only.)

REM When this bit is set to 1, the device was configured for remote operation. (Device mode only.)

ADSC When this bit is set to 1, a change of the address status occurred (i.e., untalk to talk, device to active controller, etc.).

Input and Output Message Terminator #0-3. Contains First and Last bytes of the message terminator. Input Terminators and single character Output Terminators are only one byte long and are contained in the Least Significant Byte (MSB=0). In the case of a two character Output Terminator, the Most Significant Byte of this parameter is the first character sent.

DMA Timeout and I/O Timeout Parameters. Contains the value of the desired parameter as an unsigned value in the low bytes of stat. The timeout value is expressed in milliseconds (0 to 65535).

notes The bits contained in the Interrupt Status 1 and 2 registers are extremely volatile. When you read these registers, any bits which were set are automatically cleared by the READ operation. This is extremely important to note when reading Interrupt Status Register 1, as some of the bits (not shown above) are used by various KM-488-ROM routines. It may be possible to cause various KM-488-ROM routines to report a timeout error if this register is read at certain times.

STATUS (cont.)

example This example illustrates how to use the STATUS routine.

```
VAR
      reg:INTEGER;
      stat:INTEGER;

BEGIN
      reg:= 0;
      status(reg,stat);
      WriteLn('Address Status Register =',stat);
      reg:=12;
      status(reg,stat);


      WriteLn('I/O Timeout =',stat);
END.
```

XMIT

purpose Send GPIB commands and data from a string.

usage

```
...
VAR
   info : string[];
   stat : integer;
...
BEGIN
   info : 'data and commands to be sent'
   xmit(info, stat);
...
```

parameters **info** is a STRING variable containing a series of GPIB commands and data. Each item must be separated by one or more spaces. It may also be specified as a quoted string within the XMIT call. All the available commands are described in Chapter 3. These commands include:

CMD	GTL	MTA	SDC	T0
DATA	GTLA	MLA	SEC	T1
DCL	IFC	PPC	SPE	T2
END	LISTEN	PPD	SPD	T3
EOI	LLO	PPU	TALK	UNL
GET	LOC	REN	TCT	UNT

returns **stat** is an INTEGER which describes the state of the transfer returned after the call. The returned stat value can be interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	ADRS	NCTL	UNDF	TMO	STR	NT	STX

Where

ADRS Invalid GPIB address. If this bit is set to 1, then an invalid GPIB address was given.

XMIT (cont.)

NCTL Not a System Controller. If this bit is set to 1, it indicates that the KM-488-ROM tried to send GPIB Bus Commands when it was not an Active Controller.

UNDF Undefined Command. If this bit is set to 1, the info string contained an undefined command.

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

STR String Error. If this bit is set to one, then a quoted string, END, or terminator was found without a DATA subcommand preceding it.

NT KM-488-ROM not a Talker. If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as a Talker.

STX Syntax Error. If this bit is set to 1, a syntax error was found.

notes When using the DATA subcommand, the data to be sent should be enclosed within double single quotes (") as shown in the example.

example This example illustrates one way to use the XMIT command with a Keithley 196 Voltmeter. This meter is assigned GPIB address 7 and is configured to a 30 Volt DC range with 4 1/2 digit accuracy. The meter is also configured to take a new reading each time a Group Execute Trigger Bus command (GET) is received. The program then triggers the instrument to get the first reading, and makes it a talker and the KM-488-ROM a listener in order to get the first reading.

The device to receive the setup command string which must be sent to the meter contains the following device commands:

F0 Select DC Volts mode
R3 Select 30 Volt range
S1 Select 4 1/2 digit accuracy
T3 Take one reading when GET received
X Execute the prior commands within the string

The device to receive the setup command string must also be programmed to assert the GPIB REN signal (This allows the meter to receive GPIB commands.) and to LISTEN (This allows the device to receive the string.). The programming sequence used consists of the following:

- Setting Remote Enable (REN).
- Setting all devices to UNTalk and UNListen.
- Addressing the 196 to LISTEN.
- Addressing the KM-488-ROM to talk (My Talk Address).
- Sending the Device-Dependent Commands as a string of DATA.
- Sending the appropriate message terminator characters after the data.
- Issuing the Group Execute Trigger bus command.
- Unaddressing all devices.
- Addressing the meter to TALK and the KM-488-ROM to LISTEN (My Listen Address) in preparation for receiving the latest reading.

The default value for transmit message terminator 1 is a carriage-return line-feed combination.

XMIT (cont.)

```
...
VAR
     stat:INTEGER;
...
BEGIN
     init(0,0);
     xmit('ren unl unt listen 7 mta data ''F0R3S1T3X'' tl get unl
unt talk 7 mla',stat);
```

XMITA

purpose Sends data from an array. If SETDMA is called prior to this routine, DMA will be used to transfer the data.

usage

```
VAR
   count : WORD;
   data :  type[length];
   term : INTEGER;
   stat : INTEGER;
...
BEGIN
...
   xmita(data[0],count,term,stat)
```

alternate usage `TARRAY(data[0], count,term,stat)`

parameters **data[0]** is an ARRAY which contains the data to be transmitted. The name of the first array location to be sent should be passed into the routine, i.e., data[0]. This array may be of any type, but the number of bytes per location will vary.

count is a WORD containing the number of data bytes to be transmitted. The number of data bytes stored in each location is a function of the data type. A character array, for example, contains one byte per location; whereas, an integer array contains two bytes per location. The XMITA routine sends the least significant byte of the specified array location first, followed by the bytes in increasing significance and increasing array index.

term is an INTEGER which selects the terminator to be used. This byte is of the format:

Term (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	STRM	TRM1	TRM0	X	X	X	X	EOI

Where

X This bit may be any value.

STRM Send Message Terminators. If this bit is set to 1, then the message terminator(s) will be sent at the end of the transmission. Otherwise, they will not.

XMITA (cont.)

TRM1-0 Terminator Select. These two bits select the Output Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF EOI
0	1	1	CR LF EOI
1	0	2	CR EOI
1	1	3	LF CR EOI

These terminators can be redefined by calling the OUTTERM routine.

EOI Asserts EOI. If this bit is set to 1, then EOI will be asserted when the last byte is sent. Otherwise, EOI will not be asserted.

returns stat is an INTEGER describing the state of the transfer returned after the call. The stat value is interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NT	0

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NT KM-488-ROM not a Talker. If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as a Talker.

example This example illustrates the use of the XMIT, XMITA, RCVA, SEND, and SPOLL routines to send and retrieve waveform data from a GPIB compatible oscilloscope.

```
PROGRAM xrcva; USES CRT, KM488PAS;

PROGRAM xrcva;
USES CRT, KM488PAS;

VAR
   sinedata : array[1..1024] of WORD;     { calculated sine wave }
   txdata : array[1..1024] of WORD;       { data going to scope }
   rxdata : array[1..2100] of BYTE;       { data from scope }
   rcv_sine : array[1..1028] of WORD; { reformatted scope data }
   srcptr : ^BYTE;
   hibyte : WORD;
   txstring : STRING;
   scope : INTEGER;
   kmstat :INTEGER;
   numbytes : WORD;
   numpts : WORD;
   count : WORD;
   chksum : INTEGER;
```

XMITA (cont.)

```
   length : WORD;
   angle : REAL;
   i : INTEGER;
   key : CHAR;


PROCEDURE KMErrorHandler(ErrorFlag:INTEGER);
BEGIN

    writeln('ERROR NUMBER: ',ErrorFlag);

   HALT(0);
END;


BEGIN
   scope := 16;
   ClrScr;
   GoToXY( 1,1 );

writeln(
'This program demonstrates the use of the XMITA and RCVA routines
using a');
writeln(
'Tektronix 11301 or 11302 Oscilloscope at GPIB address 16 ');

{-------------------- Initialize the KM-488-ROM ---------------- }
   scope := 16;               { Scope 488 Bus Address }
   init(0,0);
   iotimeout(5000);           { set IO timeout to 5 seconds }
{ -------------   Initialize the Oscilloscope ------------------ }
   GotoXY(1,5);

   xmit('DCL', kmstat);
   if(kmstat <>  0) THEN
      KMErrorHandler(kmstat);

   writeln( 'INITIALIZING SCOPE');
   send(scope, 'INIT', kmstat);          { Initialize scope }
   if(kmstat <>  0) THEN
      KMErrorHandler(kmstat);

   send(scope, 'RQS OFF', kmstat);       {  Disables SRQs  }
   if(kmstat <> 0) THEN
      KMErrorHandler(kmstat);

{ ----  Calculate data, send points to scope, display curve ---- }

   writeln( 'CALCULATING SINE WAVE');
   numpts := 1024;                           { Words in waveform }
   FOR i:= 1 TO  numpts  DO
      BEGIN
         angle := i * 6.28219 /  1024.0;
         sinedata[i] :=  round(400.0 * sin(angle));
         txdata[i] := Swap(sinedata[i]);
      END;
```

XMITA (cont.)

```
   numbytes := Swap( numpts * 2 + 1);


   writeln( 'SENDING SINE WAVE TO SCOPE');
   send(scope, 'INPUT STO1' , kmstat);   { Store in location 1 }
   if(kmstat <> 0) THEN
      KMErrorHandler(kmstat);

   { Setup for data transfer }
   xmit('MTA LISTEN  16  DATA ''CURVE %''', kmstat);
   if(kmstat <> 0) THEN
      KMErrorHandler(kmstat);

   count := 2;
   xmita(numbytes, count, 0, kmstat);      { Byte count sent }
   if(kmstat <> 0) THEN
      KMErrorHandler(kmstat);


   count := numpts * 2;
   xmita(txdata, count, 0, kmstat);       { Data sent }
   if(kmstat <>  0)  THEN
      KMErrorHandler(kmstat);

   chksum := 0;                    { Don't bother actual checksum }
   count := 1;                          { Send EOI with checksum }

   xmita(chksum, count, 0, kmstat);   { Checksum sent }
   if(kmstat <> 0) THEN
      KMErrorHandler(kmstat);

   writeln('CLEARING ALL TRACES');

   send(scope,'CLEAR ALL', kmstat);
   if(kmstat <> 0) THEN
      KMErrorHandler(kmstat);

   writeln('DISPLAYING STORED TRACE');

   send(scope,
   'TRACE1 DESCRIPTION:STO1,VPOSITION:0,HPOSITION:0,UNITS:"V"',km
stat);
   if(kmstat <> 0) THEN
      KMErrorHandler(kmstat);

{ ---------------  Retrieve data and compare -------------------- }

   writeln('RETRIEVING DATA FROM SCOPE');

   send(scope, 'ENCDG WAVFRM:BINARY', kmstat);
   if(kmstat <> 0) THEN
      KMErrorHandler(kmstat);

   send(scope, 'OUTPUT STO1', kmstat);
   if(kmstat <> 0) THEN
      KMErrorHandler(kmstat);
```

XMITA (cont.)

```
   send(scope, 'CURVE?', kmstat);  { Ask for data to be returned }
   if(kmstat <> 0)  THEN
      KMErrorHandler(kmstat);


   xmit('TALK 16 MLA', kmstat);   { Setup for scope to send data }
   if(kmstat <> 0)  THEN
      KMErrorHandler(kmstat);


   count := 2060;                      { number of bytes expected }
   kmstat := 0;

   rcva(rxdata, count,0, length, kmstat);   { Data received }
   IF ((kmstat <> 32) AND (kmstat <> 0)) THEN
       BEGIN
         writeln('Receive Error.');
         HALT(0);
       END;

   writeln('COMPARE SENT AND RECEIVED DATA; . = OK, * = BAD
COMPARE');

{ ..................................................
.............
The received data is in an array of type BYTE and will be extracted
to an array of type WORD.  Tektronix 110xx waveforms are composed
of a header followed by the data and a final byte checksum.  A
waveform data point is 2 bytes and is received in a low byte - high
byte sequence. The low byte of the first data element is
rxdata[10].
.................................................... ...........
}

   FOR i:=1 TO 1024 DO
     BEGIN
        srcptr := @rxdata[8 + 2 * i];      { low byte }
        rcv_sine[i] := srcptr^;
        srcptr := @rxdata[9 + 2 * i];     { high byte }
        hibyte := srcptr^;

        rcv_sine[i] := (rcv_sine[i] SHL 8) OR  hibyte;
     END;

   FOR i:=1 TO 1024 DO      { compare receive data with sent data }
      BEGIN
         IF ( sinedata[i] = rcv_sine[i]) THEN
            write('.')
         ELSE
           write('*');
      END;

   writeln;
   writeln;
END.
```

■ ■ ■

Chapter 7

PROGRAMMING IN C

While Chapter 3 gives a brief overview of the routines available for programming the KM-488-ROM, this chapter gives instructions for calling the routines from C. The routines appear in alphabetical order and include a sample program for each.

7.1 GENERAL

Supported Versions

Microsoft C version 3.0 and later TURBO C to version 2.5

The Environment

The C support files are located in directory \C on the KM-488-ROM Disks. Copy the following files to your working directory:

\C\KM488ROM.H
\C\KM488ROM.LIB

File Header

When you write your program, make sure to include the line:

```
#include<km488rom.h>
```

Compiling

Compile your program in the normal manner, being sure to link it with the library KM488ROM.LIB. For example, when working in Microsoft C, at the DOS prompt, type either:

```
cl yourprog.c /link km488rom
```

or

```
cl /c yourprog.c;
link yourprog,,,km488rom;
```

Software

The KM-488-ROM firmware contains a number of configuration parameters which govern the default settings of the input and output message terminator settings, message timeout periods, and I/O port addresses. The default terminators are shown in Table 4-4. If these default values are unsatisfactory, they may be changed by calling either the INTERM or OUTTERM routine.

The default DMA and I/O Timeouts are 10 seconds. These defaults may be altered by calling the DMATIMEOUT or IOTIMEOUT routine.

Default Terminator Settings

TERM #	OUTPUT TERMINATOR	INPUT TERMINATOR
0	LF EOI	LF
1	CR LF EOI	CR
2	CR EOI	, (comma)
3	LF CR EOI	; (semi-colon)

notes

1. Any arguments which appear as variables may also be passed as constants.

2. "Strings" in C are actually character arrays. Thus, any KM-488-ROM routines which require a string for input or output will need a character array. The name of this character array should be passed into the KM-488-ROM Routine.

3. Any KM-488-ROM 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. 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 passed into the routine. 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. Values which are returned to a C program by the KM-488-ROM routines must be handled in the following manner. In order to return a value to a named variable in C, the address of the named variable must be passed into the routine. Thus, you must pass pointers to the returned variable into KM-488-ROM call. A pointer is denoted by prefixing the variable name with an ampersand (&). However, the case of strings is an exception. In this instance C interprets the name of a character array as a pointer to the first character in the array. An example of this is shown below:

```
int status;
send(7,"F0R00X", &status);
if (status!=0)
  printf("\nStatus Error_Status=%x",status);
```

6. Arguments which are not pointers to integers or unsigneds may be passed as constants rather than variables.

7. Note that function and parameter names in C are case-sensitive. The KM-488-ROM routine names must appear in lower-case.

8. Do not name any of your variables with the same name as any of the KM-488-ROM routines.

7.2 DESCRIPTION FORMAT FOR ROUTINES

The format for each descriptions is as follows:

purpose ... a brief description of the routine. See Chapter 3 for more detailed descriptions.

usage ... gives an example of usage for each routine and assumes the input parameters are passed in as variables. These parameters can also be passed in directly. See the General Programming Notes for more information.

alternate usage ... lists alternate usage for the routine, if any. Unless otherwise noted, the alternate usage performs exactly the same function as the usage.

parameters ... describes each of the input parameters.

returns ... describes any values returned by the routine.

notes ... lists any special programming considerations.

example ... gives a programming example using the routine.

7.3 ROUTINES

DMATIMEOUT

purpose Sets the maximum length of time for a DMA transfer to complete before a timeout error is reported. (See XMITA and RCVA routines.)

usage

```
...
unsigned time;
...
dmatimeout(time);
...
```

alternate usage

```
settimeout(time);
```

NOTE: The alternate usage sets both the DMA and I/O Timeouts to the specified value.

parameters time is a UNSIGNED INTEGER representing the timeout period to elapse during a DMA transaction. A DMA Timeout Error will be generated when the time to transfer (via DMA) an entire message exceeds the set DMA timeout value (time). time can range from 0 to 65535 milliseconds and is internally rounded to the closest integer multiple of 55 milliseconds.

returns None.

example This example sets a timeout of 5 seconds.

```
dmatimeout(5000)
```

ENTER

purpose Addresses a specified device to talk, the KM-488-ROM to listen, and receives data into a character array from the addressed device.

usage

```
...
unsigned leng; maxlen;
char info[maxlen + 1];
int adrs;
int stat;
...
enter (info,maxlen,&leng,adrs,&stat);
...
```

parameters **info** is a CHARACTER ARRAY which is to hold the received data. The character array must be long enough to receive the expected number of characters plus one. The additional character is necessary so that the end of the "string" can be marked with a NULL byte. Carriage returns and the message terminator character in the incoming data are ignored and not stored with the received data.

maxlen is an UNSIGNED INTEGER which should be equal to the number of data bytes you expect to receive. maxlen must relate to info as described above.

adrs is an INTEGER containing the IEEE bus address of the device that the data is to be sent to and the terminator to be used. This byte is of the following format:

Adrs (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	TRM1	TRM0	0	ADR4	ADR3	ADR2	ADR1	ADR0

Where

TRM1-0 Terminator Select. These two bits select the Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

These terminators may be changed by the INTERM routine.

The easiest way to specify an alternate terminator is to add a factor to the GPIB address of the desired instrument which is specified within the ENTER call. The factors added for each terminator are as follows:

GPIB Address + 0 = Terminator 0
GPIB Address + 64 = Terminator 1
GPIB Address + 128 = Terminator 2
GPIB Address + 192 = Terminator 3

ENTER (cont.)

For example, if you wanted to receive a message using terminator 2 from a device at GPIB address 10, the value of adrs% supplied to ENTER would be 138 decimal (10 + 128).

ADR4-0 GPIB Address. These five bits are used to represent the GPIB address of the device to which the data is to be sent. GPIB addresses can range from 0 to 30.

returns **info** is a CHARACTER ARRAY that will contain the received data. The length of the string must be long enough to receive the expected number of characters (see the info parameter description). The maxlen parameter is used to specify this maximum length and must be one less than the number of locations within the array. ENTER will automatically insert a string terminating "NULL" at the end of the received data. ENTER will terminate reception of data when: 1) the number of bytes received exceeds maxlen, 2) the specified terminator is received, or 3) any character is received with the EOI signal. Carriage returns and the message terminator character in the incoming data are ignored and not stored with the received data.

leng is an UNSIGNED INTEGER, less than or equal to 255, which indicates the actual number of bytes which were stored.

stat is an INTEGER which describes the state of the transfer returned after the call. The returned stat values (or combination of) are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	OVF	NC	ADRS

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

OVF Overflow Error. If this bit is a 1, then the info string was filled, before a terminator character or EOI was detected.

NC KM-488-ROM not an Active Controller . If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as an Active Controller.

ADRS Invalid GPIB address. If this bit is set to 1, then an invalid GPIB address was given.

example In the following example, data is sent from two different instruments to a KM-488-ROM. The KM-488-ROM is acting as the System Controller and is assigned to GPIB address 0. One of the two instruments is a voltmeter, requiring a Carriage Return-Line Feed terminator combination (Term 1), assigned to GPIB address 7. The second instrument, located at GPIB address 10, requires a line feed (Term 0) as its terminator. The voltmeter is first sent a string of data which represents its instrument setup command. Then, when addressed to talk, it sends its most current reading to the KM-488-ROM. The second instrument is instructed to send its status, when addressed to talk.

ENTER (cont.)

It is assumed that the string sent by both instruments is 25 characters or less. The string is printed out on the computer screen.

```
#include<stdio.h>
#include<km488rom.h>
main()
{
     int inst1=7,
          inst2=10,
          inst1term=inst1+64,
     status=0;
     unsigned rlen=0;
     char instring[26];

/* Note input string MUST BE one greater than the number of
characters that will be received*/

init(0,4);
send(inst1term,"FOROTOMOX",&status);
if(status!=0)
     printf("\nError sending to Instrument 1 Status=%x", status);

/* Note send and enter require that POINTERS be passed to returned
values*/

enter(instring, 25,&rlen,inst1,&status);
if(status!=0)
     printf("\nError receiving Instrument 1

Status=%x",status);
else
     printf("\nInstrument 1 data - %s", instring);

/* Get data from second instrument...*/

send(inst2,"SEND STATUS", &status);
if(status!=0)
     printf("\nError sending to Instrument 2 Status=%x",status);
enter(instring,25,&rlen,inst2,&status);
if (status!=0)
     printf("\nError receiving Instrument 2 status=%x",status);
else
     printf("\nInstrument 2 data -s%",instring);
}
```

INIT

purpose Initializes a KM-488-ROM by assigning its GPIB address and establishing it as a System Controller or device.

usage

```
...
int adrs,mode;
...
init(adrs,mode)
...
```

INIT (cont.)

alternate usage `initialize(adrs,mode)`

parameters **adrs** is the IEEE bus address of the KM-488-ROM. This is an integer from 0 to 30.

mode sets the operating mode of the KM-488-ROM. These can be any of the following values:

Mode - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	FAST	DEV	X

Where

X May be any value.

FAST Handshake Speed. If set to 1, High Speed GPIB bus handshaking is used (500 ns). If set to 0, Low Speed GPIB bus handshaking (2 us) is used. See Chapter 3 for more information.

DEV Device. If set to 1, the KM-488-ROM is acting as a Device. Otherwise, the KM-488-ROM is acting as a System Controller. When System Controller is selected, the GPIB IFC line is momentarily asserted.

returns None.

notes You may pass the parameters directly into the routine without using variable assignments, i.e. init(0,4).

example This example initializes the KM-488-ROM as a System Controller with a IEEE address of 0 with a High Speed Handshake.

```
main()
...
{
int adrs=0,
    mode=4;
init (adrs,mode);
}
```

INTERM

purpose Changes the input message terminator settings.

usage

```
...
int num, term;

interm(num, term);
...
```

alternate usage

```
...
char term;
...
setinputeos(term);
...
```

INTERM (cont.)

NOTE: The alternate usage will only change the value of Input Message Terminator 0.

parameters num is an INTEGER which selects the number of the receive message terminator to be changed. This ranges from 0 to 3, where:

num%	TERMINATOR #	DEFAULT
0	0	LF
1	1	CR
2	2	,
3	3	;

term is an integer representing the terminator byte to be programmed. This integer is the decimal or hex equivalent of the terminator's ASCII representation. Hex equivalents must be preceded by &H. See Appendix A for ASCII Equivalents.

returns None.

notes The parameters may be passed directly into the routine.

example This example sets Input Terminator 0 to Line Feed and Input Terminator 3 to Carriage Return.

```
interm(0,10)  /*sets input terminator 0 to LF*/
...
interm (3,0xD) /*sets input terminator 3 to Carriage Return*/
```

IOTIMEOUT

purpose Changes the length of time to elapse before an I/O Timeout occurs.

usage

```
...
unsigned time;
...
   iotimeout(time);
...
```

parameters time is the amount of time to elapse before a timeout error is reported. time is any value between 0 and 65535 milliseconds. It will be internally rounded to the closest integer multiple of 55 milliseconds. The default timeout value is 10 seconds.

returns None.

example This example sets the I/O Timeout to 1 second.

```
iotimeout(1000);
```

OUTTERM

purpose Changes the output message terminator sequences.

usage

```
...
int num,chars,eoi,trm1,trm2;
outterm (num, char, eoi, trm1, trm2);
...
```

alternate usage

```
...
char trm1,trm2;
...
setoutputeos(trm1,trm2);
...
```

NOTE: The alternate usage will change only the value of Terminator 0 and will always assert EOI upon the transmission of the last character. Additionally, a single terminator is programmed by setting trm2 to 0.

parameters **num** is an INTEGER which selects the output message terminator to be changed. This ranges from 0 to 3, where:

num%	TERMINATOR #	DEFAULT
0	0	LF EOI
1	1	CR LF EOI
2	2	CR EOI
3	3	LF CR EOI

chars is an INTEGER which selects the length of the output terminator being programmed. This is 0 if a one-character terminator is required or 1 if a two-character terminator is required.

eoi is an INTEGER which determines whether or not EOI is asserted when the last terminator byte is sent. If this bit is 1, EOI will be sent. If this bit is 0, EOI will not be sent.

trm1 is an INTEGER which represents the first terminator byte to be sent. This integer is the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix A for ASCII Equivalents.) Be sure to precede all hex values with a 0x.

trm2 is an INTEGER which represents the second terminator byte (in a two-byte terminator) to be sent. This integer is the hex or decimal equivalent of the terminator's ASCII representation. (See Appendix A for ASCII Equivalents.) Be sure to precede all hex values with a 0x. If a one byte terminator is programmed, trm2may be any value.

returns None.

example This first line of this example sets Output Terminator 0 to Carriage Return with EOI. The second line of this example sets Output Terminator 3 to Carriage Return, Line Feed without EOI.

```
outterm(0,0,1,0xD,0)
...
outterm(3,1,0,0xD,0xA)
```

PPOLL

purpose Initiates a parallel poll.

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

usage

```
...
int   resp;
...
ppoll(&resp);
...
```

parameters None.

returns resp is an INTEGER which will contain the parallel poll response.

notes Before you call the PPOLL routine, you must first configure the Parallel Poll response of the device. To do this:

- Address it to listen.
- Send it a GPIB Parallel Poll Configure (PPC) command, using the XMIT command.
- Send a Parallel Poll Enable byte using the KM-488-ROM XMIT command. (Use the mnemonic CMD followed by nnn where nnn is the decimal value of the Parallel Poll Enable byte.

The Parallel Poll Enable Byte is of the format 0110SPPP, 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.

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

example This example assumes that the KM-488-ROM is connected to a Sorenson HPD30-10 Power Supply. This device is located at GPIB address 1. It is also assumed that this device drives bit 3 of the Parallel Poll Response byte to a logic "1" when service is required. To program the device to respond properly, send the Parallel Poll enable byte 01101011 (107) via the XMIT command.

```
#include<km488rom.h>
main()
{
   int  status=0,
        resp=0;

   init(0,0);
   xmit("ren unl unt listen 1 ppc cmd 107",&status);
   if (status!=0)
      printf("\nError Sending PPC cmd  Status=%x", status);

   ppoll(&resp);
   if ((resp && 8) !=0)
      printf("\nHPD30-10 Requesting Service...");
}
```

RCV

purpose Receives data into a string.

usage

```
...
char info[maxlen+1];
unsigned maxlen, rcvlen;
int stat;
...
rcv(info, maxlen, term, &rcvlen, &stat);
...
```

alternate usage

```
receive(info, maxlen, &rcvlen, &stat);
```

NOTE: The alternate usage assumes the use of Input Message Terminator 0.

parameters **info** is a CHARACTER ARRAY which will hold the received data. The array must be long enough to receive one more than the expected number of characters. Carriage returns and input terminator characters in the incoming data are ignored and not stored with the received data.

maxlen is an UNSIGNED INTEGER which specifies the maximum number of data bytes which can be received. It must be less than or equal to one less than the maximum number of array locations. This allows the terminating NULL to be stored. Otherwise, data may be stored in locations other than those allocated for your program and your program may crash.

term is an INTEGER containing the number of the input message terminator to be used, where:

term%	TERMINATOR #	DEFAULT
0	0	LF
1	1	CR
2	2	,
3	3	;

These terminators can be changed by calling the INTERM routine.

returns **info** is a CHARACTER ARRAY which will contain the received data. The length of the string must be long enough to receive the expected number of characters. RCV will terminate reception of data when: 1) the number of characters received exceeds maxlen, 2) a terminator is received, or 3) any character is received with the EOI signal. Carriage returns in the incoming data are ignored and not stored with received data.

rcvlen is an UNSIGNED INTEGER which indicates the actual number of bytes which were received and stored.

stat is an INTEGER which describes the state of the transfer returned after the call. The returned stat values are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	OVF	NL	0

RCV (cont.)

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

OVF Overflow Error. If this bit is a 1, then the info string was filled, before a terminator character or EOI was detected.

NL KM-488-ROM not a Listener. If this bit is set to a 1, it indicates the RCV was called before the KM-488-ROM was designated as a Listener.

notes The KM-488-ROM must be addressed to listen and another device addressed to talk before calling RCV.

example This example shows how the RCV routine might be used together with the XMIT routine to receive data. It uses the XMIT routine to command a Keithley 196 voltmeter to take a reading. The meter reading is received using the RCV routine. It is assumed that the meter reading returned will fit into a 25-character array.

This example assumes that the KM-488-ROM is configured such that transmit message terminator 1 is Carriage Return-Line Feed combination and this combination is also used by the Keithley 196.

```
#include<km488rom.h>
main()
{
    int stat = 0;
    char rcvdat[26];
    unsigned rcvlen;
    init(0);
    xmit("ren unl unt listen 7 mta data 'F0R3S1T3X' t1 get unl
    unt talk 7 mla", &stat);

    if(stat!=0)
        printf("\nError sending  stat=%x", stat);
    rcv(rcvdat, 25, 0,&rcvlen, &stat);
    if(stat<>0)
        printf("Error receiving  status=%x", stat);
    else
        printf("Received data=%s Length=%u",rcvdat, rcvlen);
}
```

RCVA

purpose Receives data into a specified array. RCVA may also be used to received data with DMA. (See SETDMA.)

usage

```
...
int data[maxlen/2]  /* or use char data[maxlen]*/
unsigned maxlen,rcvlen;
int term;
int stat;
...
rcva(data[0],maxlen,term,&rcvlen,&stat)
```

RCVA (cont.)

alternate usage `rarray(data[0],maxlen,&rcvlen,&stat)`

NOTE: The alternate usage terminates on EOI only.

parameters **data[ ]** is an array which is used to store the received data. It may be any data type. The array must be dimensioned large enough to store the desired number of bytes. The number of data bytes contained in each array location will vary according to the data type specified. The RCVA routine will "byte pack" data into the array, starting with the least significant byte of the specified location. You should pass the name of the first element within the array (i.e., data[0]) into the RCVA routine. (Note that passing the array name without an index has the same effect.)

term is an INTEGER which selects the type of terminator to be used. This integer is interpreted according to the following format:

Term (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	STRM	TRM1	TRM0

Where

X May be any value.

STRM Enable/Disable String Message Terminators. If this bit is 1, a Message Terminator Character will be used to detect the end of reception. If this bit is 0, a Message Terminator Character will not be used.

TRM1-0 Terminator Select. These two bits select the Input Message Terminator to be used to signal the end of a transmission. (The STRM bit must be set to 1.) Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF
0	1	1	CR
1	0	2	,
1	1	3	;

The values for these terminators can be changed by calling the INTERM routine.

maxlen is an UNSIGNED INTEGER which specifies the maximum number of data bytes which can be received. maxlen must be less than or equal to the total number of bytes which have been allocated for storage. The number of data bytes per array location varies according to the type of array. If an integer array is specified, two bytes are contained within each array location; thus maxlen should be set to twice the maximum number of array locations.

If a character array is specified, there is a one for one correspondence between number of array locations and number of bytes: Hence, maxlen = number of array locations.

The first byte received is stored in the least significant byte of the first array location.

RCVA (cont.)

returns rcvlen is an INTEGER which contains the actual number of data bytes which were received.

stat is an INTEGER describing the state of the transfer returned after the call.

The RCVA routine returns three status bits within the stat variable. The TMO bit is used to signal a timeout error. The REOI bit signals that the routine returned because the terminator was detected (if enabled), or EOI was received. The NL bit is set if the RCVA routine was called and the card was not addressed to listen. Unlike other KM-488-ROM routines, it is possible to return a non-zero status when the call was completed successfully.

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	REOI	0	TMO	0	NL	0

Where

REOI Reason for RCVA Termination. If this bit is a 1, then RCVA routine ceased because an EOI or terminator character was received. If this bit is a 0, then the RCVA was terminated because an error occurred or the maximum byte count was reached.

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NL KM-488-ROM not a Listener. If this bit is set to a 1, it indicates the RCVA was called before the KM-488-ROM was designated as a Listener.

notes The KM-488-ROM must be addressed to listen before calling this routine.

example Refer to the XMITA example.

SEND

purpose Addresses a specified device to listen, the KM-488-ROM to talk, and sends data from a string.

usage

```
...
int   adrs,
   stat;
static char info[25] = {"data to be transmitted"};
...
send(adrs,info,&stat);
...
```

parameters adrs is an INTEGER containing the IEEE bus address of the device that the data is to be sent to and the terminator to be used. This byte is of the following format:

SEND (cont.)

Adrs (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	TRM1	TRM0	0	ADR4	ADR3	ADR2	ADR1	ADR0

Where

TRM1-0 Terminator Select. These two bits select the Input Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF EOI
0	1	1	CR LF EOI
1	0	2	CR EOI
1	1	3	LF CR EOI

Terminator values may be changed by calling the OUTTERM routine.

The easiest way to specify an alternate terminator is to add a factor to the GPIB address of the desired instrument. Factors for each terminator are as follows:

GPIB Address + 0	= Terminator 0
GPIB Address + 64	= Terminator 1
GPIB Address + 128	= Terminator 2
GPIB Address + 192	= Terminator 3

For example, if you wanted to send a message using message terminator 2 to a device at GPIB address 10, the value of adrs% supplied to SEND would be 138 decimal (10 + 128).

ADR4-0 GPIB Address. These five bits are used to represent the GPIB address of the device to which the data is to be sent. GPIB addresses can range from 0 to 30.

info is a STRING containing the data to be sent.

returns **stat** is an INTEGER describing the state of the transfer returned after the call. The returned stat values (or combination of) are interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NC	ADRS

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NC Not Active Controller. If this bit is a 1, then the SEND routine was called when the KM-488-ROM was not an Active Controller.

SEND (cont.)

ADRS Invalid Address. If this bit is set to a 1, an invalid IEEE-488 device address was given.

example This example shows how to send data from a KM-488-ROM to a device. The KM-488-ROM is initialized as a System Controller located at GPIB address 10. The KM-488-ROM uses high-speed handshaking. The data (a device setup string) is sent to a device located at GPIB address 2.

```
#include<km488rom.h>
main()
{
     int  adrs=12,
          stat=0;

     static char setup[10] = {"FOROTOMOX");
/* String length must be set to one more than the total number of
characters*/

     init(10,4);
     send(adrs, setup, &stat);
/*A Pointer to the returned variable must be passed into the Send
Routine*/

if (stat!=0)
     printf("\nError send status=%x", stat);
}


As an alternative, the following sequence could be used:

#include<km488rom.h>
main()
{
     int stat=0;
     init(10,4);
     send(12,"FOROTOMOX",&sta t);
     if(stat!=0)
          printf("/n Error Sending status=%x", stat);
}
```

SETBOARD

purpose In multiple board system, identifies the KM-488-ROM to be programmed.

usage

```
...
int board;
...
setboard(board);
...
```

alternate usage

```
boardselect(board);
```

parameters **board** is an INTEGER between 0 and 3 representing the board to be programmed. Note that up to four boards install in any one system. The board "number" is associated with the base address of its I/O port.

SETBOARD (cont.)

returns None.

notes You must assign a board "number" for every KM-488-ROM in the system before calling the SETBOARD routine. Board numbers are assigned using the SETPORT routine.

Each board must be must be initialized independently by calling the INIT routine. You must do this the first time a given board is selected before any other operations are conducted on that board.

Once a board has been selected using SETBOARD, all further I/O operations will be performed on that board until the next SETBOARD is executed.

example This example selects board number 2.

```
setboard(2);
```

SETDMA

NOTE: DMA allows maximum data transfer rates in excess of 100 kilobytes per second. However, the actual data rates are limited by the rates at which other devices connected to the bus can send or receive data. These rates are governed automatically by the GPIB handshaking signals.

purpose Allows the use of DMA in conjunction with XMITA and RCVA.

usage

```
...
int channel;
setdma(channel);
...
```

alternate usage

```
dmachannel(channel);
```

parameters channel is an INTEGER which specifies the DMA channel to be used for the transfer, where:

1 = Select DMA channel 1.
2 = Select DMA channel 2.
3 = Select DMA channel 3.

To disable DMA, assign any value other than 1,2, or 3 to channel.

returns None.

notes The DMA hardware jumpers must be properly set for the DMA channel selected by SETDMA. Note that the default setting for the jumpers is DMA DISABLED. The jumpers are further described in Chapter 2.

When SETDMA is called to enable the use of DMA, each call to the XMITA and RCVA routines that follows will use DMA to accomplish the transfer until SETDMA is called with a parameter outside the range of 1-3.

SETDMA (cont.)

example This example specifies that DMA transfers are to take place using DMA Channel 1 and then disables DMA.

```
setdma(1);
/*Transfers initiated by RCVA and XMITA will occur over DMA channel
1.*/
...
setdma(0);  /*Disables DMA.*/
...
```

SETINT

purpose Sets the KM-488-ROM's interrupt enable bits.

usage

```
...
int intval;
...
setint(intval);
...
```

parameters **intval** is an integer containing the address and value of the Interrupt Mask Register which is to be written to. This is interpreted as follows:

INTVAL (Input) - High Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	X	ADRS

Where

X May be any value.

ADRS If this bit is set to 0, bits 0 through 7 will be written to Interrupt Mask 1. If this bit is set to 1, bits 0 through 7 will be written to Interrupt Mask 2.

INTERRUPT MASK 1

INTVAL (Input) - Low Byte (ADRS = 0)

BIT	7	6	5	4	3	2	1	0
	0	0	GET	0	DEC	0	0	0

Where

GET When this bit is set to 1, an interrupt will be generated when a KM-488-ROM acting as a device received a GPIB GET (Group Execute Trigger) command while addressed to listen.

DEC When this bit is set to 1, an interrupt is generated when a Device Clear is received.

SETINT (cont.)

INTERRUPT MASK 2

INTVAL (Input) - Low Byte (ADRS = 1)

BIT	7	6	5	4	3	2	1	0
	0	SRQI	0	0	0	LOKC	REMC	ADSC

Where

SRQI When this bit is set to 1, an interrupt is generated when SRQ is received.

LOKC When this bit is set to 1, an interrupt is generated when the state of the Local Lockout bit changes.

REMC When this bit is set to 1, an interrupt is generated when the state of the Local/Remote bit changes.

ADSC When this bit is set to 1, an interrupt is generated when the state of the LA, TA, or CIC bits within the address status register changes.

returns None.

notes Be certain to assign the KM-488-ROM to an interrupt level before using this routine. Interrupt Levels are assigned by means of a jumper on the KM-488-ROM board. This jumper is described in detail in Chapter 2.

You must set-up an interrupt handling routine within the QuickBASIC program to deal with the interrupt condition.

example This example enables the KM-488-ROM to generate an interrupt when SRQ is received.

```
SETINT (0x140);
```

SETPORT

purpose This routine is used to alter the range of addresses used by the KM-488-ROM's I/O Port. In a multiple board environment, it is also used to associate a given range of I/O addresses with a board number.

usage

```
...
int board;
unsigned ioport;
...
setport(board,ioport);
...
```

parameters **board** is an INTEGER between 0 and 3 which represents the board to be programmed. Note that up to four board can be installed in any one system. The board number is associated with the base address of its I/O ports.

ioport is an UNSIGNED INTEGER which represents the I/O Base Address of the KM-488-ROM. The default Base Address is 2B8 Hex. The Base Address selected must match the one selected by the Base Address Switch on the KM-488-ROM. (See Chapter 2 for more information.)

SETPORT (cont.)

returns None.

notes When multiple boards are used, each board must have its own unique base address. Any base address can be assigned to any board number provided that none of the base addresses overlap.

example This line assigns Board 0 to a Base address of 300 hex.

```
setport(0, 0x300);
```

SETSPOLL

purpose Sets the Serial Poll Response of the KM-488-ROM, when it is acting as a device (non-Controller).

usage

```
...
int resp;
setspoll(resp);
...
```

parameters **resp** is an INTEGER describing the serial poll response and the state of the SRQ bit. This byte is of the following format:

Resp% (Input) - Low Byte

BIT	7	6	5	4	3	2	1	0
	SPR8	RSV	SPR6	SPR5	SPR4	SPR3	SPR2	SPR1

Where

SPR1-8 Bits 1 through 8 of this device's Serial Poll Response Byte.

RSV If this bit is 1, SRQ will be asserted to request servicing. Otherwise, SRQ will not be asserted.

returns None.

example This example illustrates how SETSPOLL may be used to notify the controller of local error conditions.

```
...
int resp=0;

    if (error1)  /*Set SPR bits based upon local error conditions*/
        resp|=1;
    if (error2)
        resp|=2;
    .
    .
    .
    if (error7)
        resp|=128;
    if (error!=0)
        resp|=0x40;  /*set RSV if error*/
    call setspoll(resp);
...
```

SPOLL

purpose Initiates a serial poll of the specified device.

usage

```
...
int adrs;
char resp;
int stat;
...
spoll(adrs, &resp, &stat);
...
```

parameters adrs is an INTEGER containing the IEEE bus address of the device that is to be serial polled. This can range from 0 to 30.

returns resp is a CHARACTER containing the serial poll response received. The definition of resp 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.

stat is an INTEGER describing the state of the transfer returned after the call. The stat value is interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NC	ADR

Where

TMO Indicates whether a Timeout Error occurred during data transfer. If a 1, then a Timeout Error occurred.

NC KM-488-ROM not a Controller. If set to a 1, it indicates the routine was called before the KM-488-ROM was designated as an Active Controller.

ADR Invalid GPIB Address. If this bit is set to 1, an invalid GPIB address was provided.

notes Pointers to the RESP and STATUS variables are passed.

example This examples illustrates a simple serial poll of a device located at GPIB address 10.

```
int resp,stat;
spoll(10,&resp,&stat);
if(stat!=0)
    printf("\nError during serial poll ; status%x",stat);
printf("\nSerial Poll Response =%x",resp);
if((resp&0x40)!=0)
    printf("\nDevice Requesting Service");
```

SRQ

purpose Detects the presence of the GPIB SRQ signal.

usage

```
if (srq())...
```

parameters None.

SRQ (cont.)

returns The SRQ function returns a 0 or FALSE condition when SRQ has not been detected, or a 1 or TRUE condition when SRQ is present.

notes The value returned by the SRQ function is generally used within a conditional branch in the application program.

Note that once you have obtained a TRUE response from the SRQ function, the SRQ response will be reset to FALSE even if the SRQ line is still active. In order to reset the SRQ response to TRUE, you must serial poll at least one device which was requesting service. Conducting a serial poll on a device which was requesting service will reset its SRQ line. At this time, if other devices were simultaneously asserting SRQ, the output of the SRQ function would once again be reset to TRUE. Otherwise, the SRQ function would become TRUE on the next assertion of the SRQ line.

example This example assumes that the KM-488-ROM is connected to an instrument located at GPIB address 1 which is capable of requesting service via SRQ. When the SRQ is detected, the SPOLL function will be called and the serial poll response of the device will be printed to the computer screen.

```
#include<km488rom.h>

main()
{
int  resp,
          stat;

if (srq())
     {
     spoll(1,&resp,&stat);
          if (status!=0)
               printf("\nError calling SPOLL-Status=%x,stat);
          else
               printf("\nSRQ Received from device 1 -Poll
Response=%x',resp);
     }
}
```

STATUS

purpose Returns the value of the specified setup parameter.

usage

```
...
int reg,stat;

status(&reg,&stat)
...
```

parameters reg is an INTEGER containing the address of the register or configuration parameter to be queried. This value corresponds to a 4-bit field which specifies the status register or configuration parameter to be read. The format of the reg byte is as follows:

STATUS (cont.)

Reg (input) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	X	X	ADR3	ADR2	ADR1	ADR0

Where

X May be any value.

ADR3-0 REGISTER/PARAMETER SELECT. This is a 4-bit field which specifies the status register or configuration parameter to be read. Registers and parameters are selected as follows:

ADR3	ADR2	ADR1	ADR0	REGISTER/PARAMETER
0	0	0	0	Address Status Reg
0	0	0	1	Interrupt Status 1 Reg
0	0	1	0	Interrupt Status 2 Reg
0	0	1	1	DMA Status Reg
0	1	0	0	Output Terminator 0
0	1	0	1	Output Terminator 1
0	1	1	0	Output Terminator 2
0	1	1	1	Output Terminator 3
1	0	0	0	Input Terminator 0
1	0	1	1	Input Terminator 1
1	1	1	0	Input Terminator 2
1	1	1	1	Input Terminator 3
1	0	0	0	I/O Timeout Parameter
1	0	0	1	DMA Timeout Parameter
1	1	1	0	I/O Port Address
1	1	1	1	GPIB Address of KM-488-ROM

returns **reg** - When STATUS obtains the value of one of the four transmit message terminators, this variable will contain two flag bits which determine the length of the terminator and whether or not EOI is asserted with the last byte. When obtaining other parameters, reg% will retain its input value.

Reg (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	LEN	EOI

Where

LEN Terminator Length. If this bit is set to 0, then the terminator is one byte long. If this bit is set to 1, then the terminator is two bytes long.

EOI If this bit is set to 1, EOI is asserted when the last terminator byte is sent. Otherwise, EOI is not asserted.

stat is an INTEGER describing the status bits for the register or the configuration parameter which was specified by the reg% parameter. Unless otherwise noted, the high byte of stat% is returned as 0.

STATUS (cont.)

Address Status Register

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	CIC	X	X	X	X	LA	TA	X

Where

X This bit may be any value.

CIC Active Controller. If this bit is set to 1, then the KM-488-ROM is a System Controller.

LA Listener. If this bit is set to 1, then the KM-488-ROM is a Listener.

TA Talker. If this bit is set to 1, then the KM-488-ROM is a Talker.

Interrupt Status Register 1

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	X	GET	X	DEC	X	X	X

Where

X This bit may be any value.

GET Group Execute Trigger. If this bit is set to 1, then a Group Execute Trigger command was received while the KM-488-ROM was a device.

DEC When this bit is set to 1, a Device Clear was received.

Interrupt Status Register 2

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	X	SRQ1	LOK	REM	X	X	X	ADSC

Where

X This bit may be any value.

SRQI When this bit is set to 1, it indicates SRQ was active. (Active Controller mode only.)

LOK When this bit is set to 1, the device was set to Local Lockout. (Device mode only.)

REM When this bit is set to 1, the device was configured for remote operation. (Device mode only.)

ADSC When this bit is set to 1, a change of the address status occurred (i.e., untalk to talk, device to active controller, etc.).

STATUS (cont.)

Input and Output Message Terminator #0-3. Contains First and Last bytes of the message terminator. Input Terminators and single character Output Terminators are only one byte long and are contained in the Least Significant Byte (MSB=0). In the case of a two character Output Terminator, the Most Significant Byte of this parameter is the first character sent.

DMA Timeout and I/O Timeout Parameters. Contains the value of the desired parameter as an unsigned value in the low bytes of stat. The timeout value is expressed in milliseconds (0 to 65535).

notes The bits contained in the Interrupt Status 1 and 2 registers are extremely volatile. When you read these registers, any bits which were set are automatically cleared by the READ operation. This is extremely important to note when reading Interrupt Status Register 1, as some of the bits (not shown above) are used by various KM-488-ROM routines. It may be possible to cause various KM-488-ROM routines to report a timeout error if this register is read while the KM-488-ROM is addressed to talk or listen.

example This example illustrates how to use the STATUS routine.

```
#include<km488rom.h>

main ()
{
    int  reg,
         stat;

    reg=0;
    status(&reg,&stat);
    printf("\nAddress Status Register=%x Hexadecimal",stat);
    reg=1100B;
    status(&reg,&stat);
    printf('\nI/O Timeout=%u milliseconds",stat);
}
```

XMIT

purpose Sends GPIB commands and data from a string.

usage

```
...
static char info[] = {"Data and Commands to be Sent"};
int stat;
...
xmit(info, &stat);
...
```

parameters **info** is a CHARACTER ARRAY (string) containing a series of GPIB commands and data. Each item must be separated by one or more spaces. All the available commands are described in Chapter 3. These commands include:

XMIT (cont.)

CMD	GTL	MTA	SDC	T0
DATA	GTLA	MLA	SEC	T1
DCL	IFC	PPC	SPE	T2
END	LISTEN	PPD	SPD	T3
EOI	LLO	PPU	TALK	UNL
GET	LOC	REN	TCT	UNT

info can be specified as a quoted "string" within the XMIT call, or as the name of a character array which has been initialized to the desired string.

returns **stat** is an INTEGER which describes the state of the transfer returned after the call. The returned stat value can be interpreted as follows:

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	ADRS	NCTL	UNDF	TMO	STR	NT	STX

Where

ADRS Invalid GPIB address. If this bit is set to 1, then an invalid GPIB address was given.

NCTL Not a System Controller. If this bit is set to 1, it indicates that the KM-488-ROM tried to send GPIB Bus Commands when it was not an Active Controller.

UNDF Undefined Command. If this bit is set to 1, the info string contained an undefined command.

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

STR String Error. If this bit is set to one, then a quoted string, END, or terminator was found without a DATA subcommand preceding it.

NT KM-488-ROM not a Talker. If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as a Talker.

STX Syntax Error. If this bit is set to 1, a syntax error was found.

example This example illustrates one way to use the XMIT command with a Keithley 196 Voltmeter. This meter is assigned GPIB address 7 and is configured to a 30 Volt DC range with 4 1/2 digit accuracy. The meter is also configured to take a new reading each time a Group Execute Trigger Bus command (GET) is received. The program then triggers the instrument to get the first reading, and makes it a talker and the KM-488-ROM a listener in order to get the first reading.

The device to receive the setup command string which must be sent to the meter contains the following device commands:

F0	Select DC Volts mode
R3	Select 30 Volt range
S1	Select 4 1/2 digit accuracy
T3	Take one reading when GET received
X	Execute the prior commands within the string

XMIT (cont.)

The device to receive the setup command string must also be programmed to assert the GPIB REN signal (This allows the meter to receive GPIB commands.) and to LISTEN (This allows the device to receive the string.). The programming sequence used consists of the following:

- Setting Remote Enable (REN).
- Setting all devices to UNTalk and UNListen.
- Addressing the 196 to LISTEN.
- Addressing the KM-488-ROM to talk (My Talk Address).
- Sending the Device-Dependent Commands as a string of DATA.
- Sending the appropriate message terminator characters after the data.
- Issuing the Group Execute Trigger bus command.
- Unaddressing all devices.
- Addressing the meter to TALK and the KM-488-ROM to LISTEN (My Listen Address) in preparation for receiving the latest reading.

The default value for transmit message terminator 1 is a carriage-return line-feed combination.

```
#include<km488rom.h>
main()
{
     int status=0;
          init(0,0);
          xmit("ren unl unt listen 7 mta data 'F0R3S1T3X' t1 get unl
talk 7 mla",&status);

/*Note that a string which was initialized to the command sequence
could also be passed into the XMIT call.  Additionally, the XMIT
subcommands can be specified in upper or lower case.  However, the
characters sent by the DATA subcommand will be sent as specified
(UPPER or lower case), and the Keithley 196 requires its commands
to be specified as UPPER CASE text.*/

if(status!=0)
     printf("\nTransmit Error =%x", status);
}
```

XMITA

purpose Sends data from an array. It may also be used in conjunction with the SETDMA routine to initiate DMA transfers.

usage

```
...
int data[];
unsigned count;
int term;
int stat;
...
xmita(data[0],count,term,&stat);
```

XMITA (cont.)

alternate usage

```
...
unsigned count;
char term;
int stat;
...
tarray(data[0],count,term,&stat)
...
```

parameters data[] is an ARRAY containing data (of any type) to be transmitted. The only difference between arrays of varying types is the number of data bytes in each array location (char = 1 byte per location; int, unsigned = 2 bytes per location, word = 4 bytes per location). When transmitting data, XMITA sends data from the least significant byte of the specified array location, progressing from a least-significant through most-significant-byte order from increasing array locations.

count is an UNSIGNED INTEGER containing the number of data bytes to be transmitted.

term is an INTEGER which selects the terminator to be used. This byte is of the format:

Term (Input Parameter) - Low Byte

BIT	7	6	5	4	3	2	1	0
	STRM	TRM1	TRM0	X	X	X	X	EOI

Where

X This bit may be any value.

STRM Send Message Terminators. If this bit is set to 1, then the message terminator(s) will be sent at the end of the transmission. Otherwise, they will not.

TRM1-0 Terminator Select. These two bits select the Output Message Terminator to be used to signal the end of a transmission. Available terminator selections are

TRM1	TRM0	TERMINATOR #	DEFAULT
0	0	0	LF EOI
0	1	1	CR LF EOI
1	0	2	CR EOI
1	1	3	LF CR EOI

These terminators can be redefined by calling the OUTTERM routine.

EOI Asserts EOI. If this bit is set to 1, then EOI will be asserted when the last byte is sent. Otherwise, EOI will not be asserted.

returns stat is an INTEGER describing the state of the transfer returned after the call. The stat value is interpreted as follows:

XMITA (cont.)

Stat (Return) - Low Byte

BIT	7	6	5	4	3	2	1	0
	0	0	0	0	TMO	0	NT	0

Where

TMO Timeout Error. Indicates whether or not a Timeout Error occurred during data transfer. If this bit is a 1, then a Timeout Error occurred.

NT KM-488-ROM not a Talker. If this bit is set to a 1, it indicates the routine was called before the KM-488-ROM was designated as a Talker.

example This example illustrates the use of the XMIT, XMITA, RCVA, SEND, and SPOLL routines to send and retrieve waveform data from a GPIB compatible oscilloscope.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <graph.h>

#include "km488rom.h"

/* function prototypes */
void KMErrorHandler(int ErrorFlag);
unsigned int SwapBytes(unsigned value);

unsigned sinedata[1024],   /* sine wave data */
         txdata[1028],     /* sinedata converted by SwapBytes() */
         rcv_sine[1028];   /* for converted data */

char    rxdata[2100];      /* for raw data from Oscilloscope  */

int count,length,scope;
unsigned  numbytes;
char rcvstring[81],
     txstring[80];

void main()
{
double angle;
int key,
    i,
    chksum,
    numpts,
    sp_resp,
    kmstat;
char  *s, *d;
   _clearscreen( _GCLEARSCREEN );
   _settextposition( 1,0 );
printf(
"This program demonstrates the use of the XMITA and RCVA routines
using a\n");
```

XMITA (cont.)

```
printf(
"Tektronix 11301 or 11302 Oscilloscope at GPIB address 16");
  _settextposition( 4,0 );

/*------------------ Initialize the KM-488-ROM ---------------- */

   init(0,0);  /* KM-488-ROM is System Controller at GPIB Adrs 0 */
   iotimeout(5000);     /* set IO timeout to 5 seconds */

/*-------------   Initialize the Oscilloscope ------------------*/
   scope = 16;                    /*  <--Scope 488 Bus Address  */
   printf( "\nINITIALIZING SCOPE\n\n");
   xmit("DCL", &kmstat);
   if(kmstat != 0)
       KMErrorHandler(kmstat);

/*----- Calculate data, send points to scope, display
curve ----*/

   printf( "CALCULATING SINE WAVE\n\n");
   numpts = 1024;                         /* Words in waveform */
   for(i=0; i<numpts; i++) {
      angle = i * 6.28219 / (double) 1024;
      sinedata[i] = (unsigned) (400 * sin(angle));
      txdata[i] = SwapBytes(sinedata[i]);
   }
   numbytes = SwapBytes( numpts * 2 + 1);
   do {
      printf( "SENDING SINE WAVE TO SCOPE\n\n");
    spoll(scope, &sp_resp, &kmstat);
    printf("\nSpoll  = %.2x\n", sp_resp & 255);
   send(scope, "INIT", &kmstat);          /* Initialize scope */
   if(kmstat != 0)
        KMErrorHandler(kmstat);
   send(scope, "RQS OFF", &kmstat);       /*  Disables SRQs  */
   if(kmstat != 0)
        KMErrorHandler(kmstat);
   send(scope, "INPUT STO1" , &kmstat);  /* Store in location 1 */
   if(kmstat != 0)

       KMErrorHandler(kmstat);

   sprintf(txstring,"MTA LISTEN %d DATA 'CURVE %%'", scope );
   xmit(txstring, &kmstat);          /* Setup for data transfer */
   if(kmstat != 0)
       KMErrorHandler(kmstat);
   count = 2;
   xmita(&numbytes, count, 0, &kmstat);    /*   Byte count sent */
   if(kmstat != 0)
       KMErrorHandler(kmstat);
   count = numpts * 2;
   xmita(txdata, count, 0, &kmstat);       /* Data sent  */
   if(kmstat != 0)
        KMErrorHandler(kmstat);
   chksum = 0;                   /* Don't bother actual checksum */
   count = 1;                   /* Send EOI with checksum */
   xmita(&chksum, count, 0, &kmstat);  /* Checksum sent  */
```

XMITA (cont.)

```
   if(kmstat != 0)
        KMErrorHandler(kmstat);

   printf("CLEARING ALL TRACES\n\n");
   send(scope,"CLEAR ALL", &kmstat);
   if(kmstat != 0)
      KMErrorHandler(kmstat);
   printf("DISPLAYING STORED TRACE\n\n");
   sprintf(txstring,
     "TRACE1 DESCRIPTION:STO1,VPOSITION:0,HPOSITION:0,UNITS:\"V\"")
;
   send(scope, txstring, &kmstat);
   if(kmstat != 0)
        KMErrorHandler(kmstat);

/*--------------  Retrieve data and compare --------------------*/

   printf("\n\nRETRIEVING DATA FROM SCOPE\n");
   send(scope, "ENCDG WAVFRM:BINARY", &kmstat);
   if(kmstat != 0)
        KMErrorHandler(kmstat);
   send(scope, "OUTPUT STO1", &kmstat);
   if(kmstat != 0)
        KMErrorHandler(kmstat);
   send(scope, "CURVE?", &kmstat); /* Ask for data to be returned
*/
   if(kmstat != 0)
        KMErrorHandler(kmstat);
   sprintf(txstring, "TALK %d MLA", scope);
   xmit(txstring, &kmstat);      /* Setup for scope to send data */
   if(kmstat != 0)
        KMErrorHandler(kmstat);
   count = 2058;                      /* number of bytes expected */
   kmstat = 0;
   rcva(rxdata, count,0, &length, &kmstat);    /* Data received */
   if (kmstat != 32 && kmstat != 0) {
   printf("Recieve error: %d", kmstat);
        exit(3);
   }
   if (length !=  (10 + numpts * 2)) {
        printf("\nCount Error - received %d", length);
        exit(4);

   }
   xmit("UNT UNL", &kmstat);    /* untalk the scope */
   if(kmstat != 0)
        KMErrorHandler(kmstat);
   printf("COMPARE SENT AND RECEIVED DATA; . = OK, * = BAD
COMPARE\n");
/*
    Received data format starts with header and byte count - CURVE
%XX.    Received data starts at offset 9.
*/
   s = rxdata+9;                  /* start of data */
   d = (char *) rcv_sine;

   for(i=0;i<2048;i++)      /* copy data to rcv_sine[] */
```

XMITA (cont.)

```
      *d++ = *s++;
   for(i=0; i<1024; i++) {
       rcv_sine[i] = SwapBytes(rcv_sine[i]);

       if( sinedata[i] == rcv_sine[i])
          putch('.');
       else
          putch('*');
   }
   printf("\nComplete\n");
   key=getch();
   if(key==27) exit(0);
   exit(0);
}
unsigned int SwapBytes(unsigned value)
{
unsigned rvalue=0;
   rvalue = value >> 8;
   rvalue |= value << 8;
return  rvalue;
}

void KMErrorHandler(int ErrorFlag)
{
    printf("\nERROR NUMBER: %d\n",ErrorFlag);
   exit(-1);
}
```

■ ■ ■

Chapter 8

FACTORY RETURNS

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

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

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

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

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

Ship the equipment 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 the RMA number on the outside of the package!

■ ■ ■

Appendix A

ASCII Code 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			

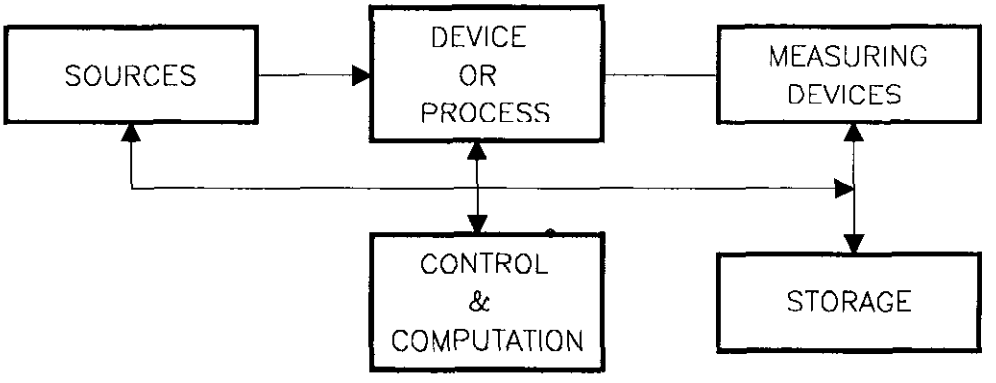
■ ■ ■

Appendix B

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.

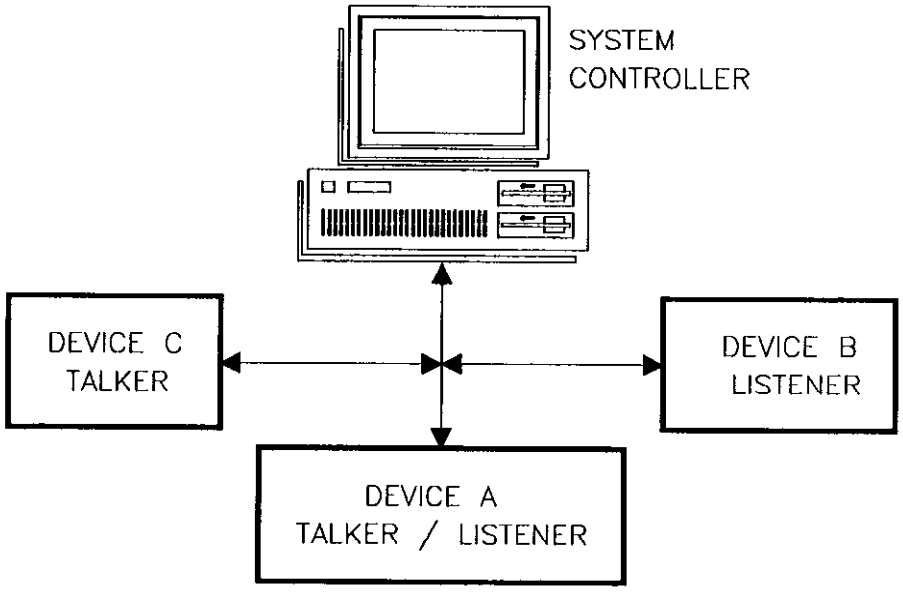


Figure B-2. Typical System

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 usecs.

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.

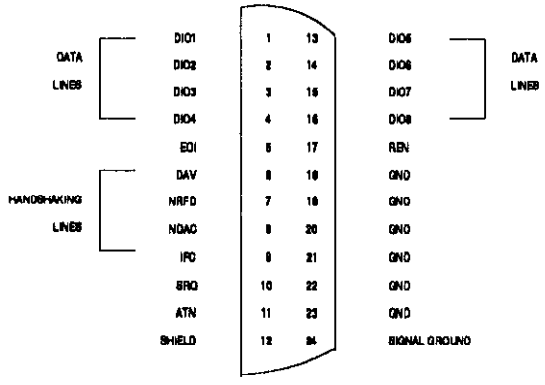


Figure B-3. IEEE-488 Bus Connector

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 u 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.

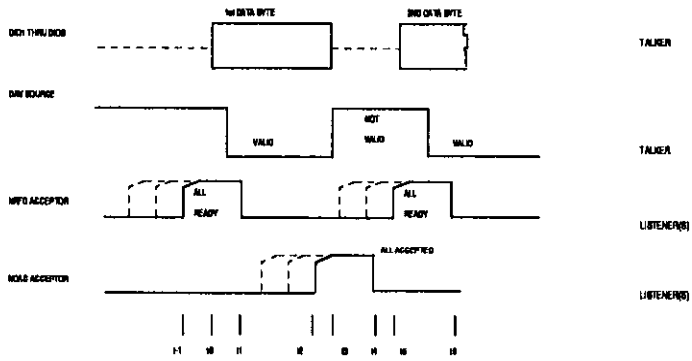


Figure B-4. Handshake Timing

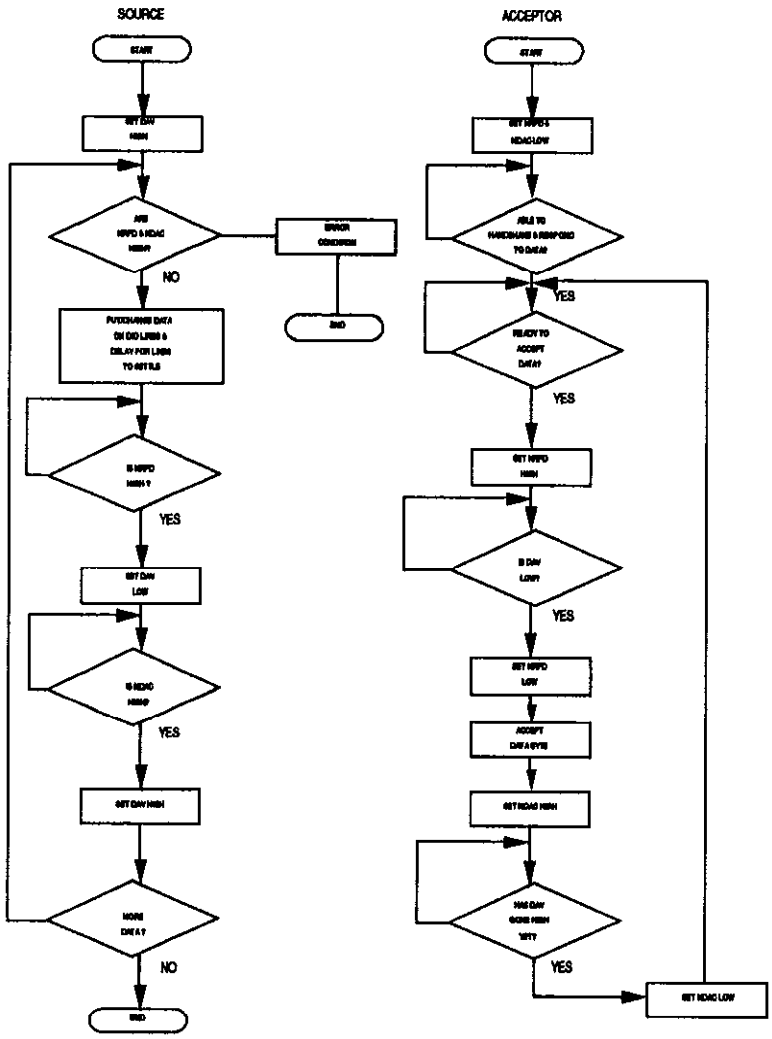


Figure B-5. Transmission of Data from Talker to Listener

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 t0 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 SH1 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 D 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 be 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 secs. 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 determine 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

why the need for 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

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	FUNCTION
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 Unconfigure	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	FUNCTION
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

■ ■ ■

Appendix C

IEEE Multiline Commands

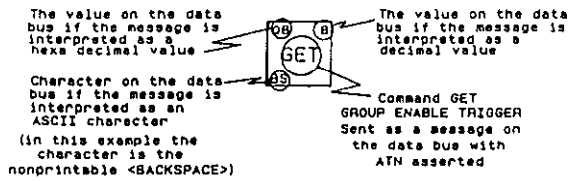
		20 32 SP	00	30 48 0	16	40 64 @	00	50 80 P	16	60 96 `	00	70 112 p	16
01 1	11 17	21 33 !	01	31 49 1	17	41 65 A	01	51 81 Q	17	61 97 a	01	71 113 q	17
GTL SOH	LLO DC1	22 34 "	02	32 50 2	18	42 66 B	02	52 82 R	18	62 98 b	02	72 114 r	18
		23 35 #	03	33 51 3	19	43 67 C	03	53 83 S	19	63 99 c	03	73 115 s	19
04 4	14 20	24 36 $	04	34 52 4	20	44 68 D	04	54 84 T	20	64 100 d	04	74 116 t	20
SDC EOT	DCL DC4	25 37 %	05	35 53 5	21	45 69 E	05	55 85 U	21	65 101 e	05	75 117 u	21
05 5	15 21	26 38 &	06	36 54 6	22	46 70 F	06	56 86 V	22	66 102 f	06	76 118 v	22
PPC ENQ	PPU NAK	27 39 '	07	37 55 7	23	47 71 G	07	57 87 W	23	67 103 g	07	77 119 w	23
08 8	16 24	28 40 (	08	38 56 8	24	48 72 H	08	58 88 X	24	68 104 h	08	78 120 x	24
GET BS	SPE CAN	29 41)	09	39 57 9	25	49 73 I	09	59 89 Y	25	69 105 i	09	79 121 y	25
09 9	19 25	2A 42 *	10	3A 58 :	26	4A 74 J	10	5A 90 Z	26	6A 106 j	10	7A 122 z	26
TCT HT	SPD EM	2B 43 +	11	3B 59 ;	27	4B 75 K	11	5B 91 [	27	6B 107 k	11	7B 123 {	27
		2C 44 ,	12	3C 60 <	28	4C 76 L	12	5C 92 \	28	6C 108 l	12	7C 124 |	28
		2D 45 -	13	3D 61 =	29	4D 77 M	13	5D 93]	29	6D 109 m	13	7D 125 }	29
		2E 46 .	14	3E 62 >	30	4E 78 N	14	5E 94 ^	30	6E 110 n	14	7E 126 ~	30
		2F 47 /	15	3F 63 ?	UNL	4F 79 O	15	5F 95 _	UNT	6F 111 o	15	7F 127 DEL	31

ACG | UCG | LAG | TAG | SCG (PPE, 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 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

■ ■ ■

Appendix D

Device Capability Codes

DEVICE CAPABILITY CODES

AH Function Allowable Subsets

Identification	Description	Other Function Subsets Required
AH0	No capability	None
AH1	No capability	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 ^ (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.

Appendix E

Printer & Serial Port Redirection

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

E.1 PARALLEL PORT REDIRECTION

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**

Invoke the KMLPT utility as follows:

1. Change to the directory where your KM-488-ROM 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-ROM.

/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&H2B8 Configures LPT1: for output to IEEE device 05 on an interface card located at 2B8h.

KMLPT LPT1
IEEE05 /A&H2B8

Resulting Printer Port Table

LPT1:	LPT1
LPT2:	IEEE05
LPT3:	LPT2
LPT4:	None

KMLPT IEEE05
IEEE1201 /A&H2B8

Resulting Printer Port Table

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.

E.2 UNLOADING KMLPT FROM DOS

To unload the KMLPT utility from the DOS command line:

1. Change to the directory where your KM-488-ROM 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.

E.3 SERIAL PORT REDIRECTION

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 "unaddressed."

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`**

E.4 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-ROM software is located.
2. At the DOS prompt, type **`KMCOM n1 /Ioaddr /Baddr /t`**

   Where

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-ROM.

/Baddr is an optional parameter which specifies the IEEE-488 Bus address (0 to 30 decimal) of the KM-488-ROM. 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.

NOTE: Parameters must appear in all UPPER CASE or all lower case. UPPER CASE and lower case cannot be mixed.

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&H2B8	Configures COM1: for output to IEEE device 05 on an interface card located at 2B8 (hex).
KMCOM COM1 IEEE05 /A&H2B8	Resulting Printer Port Table COM1: COM1 COM2: IEEE05 COM3: COM2 COM4: None
KMCOM IEEE05 IEEE1201 /A&H2B8	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.

E.5 UNLOADING KMCOM FROM DOS

To unload the KMCOM utility from the DOS command line:

1. Change to the directory where your KM-488-ROM 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.

E.6 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.

E.7 EXAMPLE PROGRAM

An example program in BASICA, COMTEST.BAS, is provide on the KM-488-ROM Disk. This example program illustrates how to use the KMCOM re-director feature.

A plot file, HPEXAMPLE.PLT, is also provided on the KM-488-ROM Disks. This file can be printed to an HP plotter using the KMLPT Re-Direct Driver.

■ ■ ■