

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

Q22bus ADAPTOR
VMS SUPPORT
SOFTWARE

DISCLAIMER:

Please read and abide by the following paragraphs. Questions and comments should be directed to:

Technical Publications Department
Bit 3 Computer Corporation
8120 Penn Avenue South
Minneapolis, MN 55431-1393
612-881-6955

Bit 3 makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Bit 3 assumes no responsibility for any errors that may appear in this document. The information in this document is subject to change without notice.

Bit 3 Computer Corporation does not authorize the use of its components in life support applications where failure or malfunction of the component may result in injury or death. In accordance with Bit 3's terms and conditions of sale, the user of Bit 3 components in any and all life support applications assumes all risks arising out of such use and further agrees to indemnify and hold Bit 3 harmless against any and all claims of whatsoever kind or nature (including claims of culpable conduct [strict liability, negligence or breach of warranty] on the part of Bit 3) for all costs of defending any such claims.

Bit 3 does not authorize the use of its components in nuclear control and process applications where failure or malfunction of the component may result in radioactive releases, explosions, environmental damage/contamination, personal injury or death. In accordance with Bit 3's terms and conditions of sale, the user of Bit 3 components in any and all nuclear applications assumes all risks arising out of such use and further agrees to indemnify and hold Bit 3 harmless against any and all claims of whatsoever kind or nature (including claims of culpable conduct [strict liability, negligence or breach of warranty] on the part of Bit 3) for all costs of defending any such claims.

Manual copyright (c) 1988, 1989, 1990, 1991 by Bit 3 Computer Corporation.
Software copyright (c) 1988, 1989, 1990, 1991 by Bit 3 Computer Corporation.
All rights reserved.

Revision 2.3 03/91
Pub. No. 82602165

NOTE:

MULTIBUS is a trademark of Intel Corporation.
DEC, Q22-bus, MicroVAX, VAX and VMS are trademarks of Digital Equipment Corporation.
PC/AT, Micro Channel and PS/2 are trademarks of IBM Corporation.
NuBus is a trademark of Texas Instruments.

PREFACE

This manual describes Model 400-901 and Model 400-902 Support Software for Bit 3 Q22-bus Adaptors, its installation, set up and use. Both models are VMS Support Software for Q22-bus adaptors. Model 400-901 is on RX50 diskettes and Model 400-902 is on TK50 compact tapes. In this manual, Model 400-901/902 refers to both Support Software models.

Information about jumper settings, the physical installation of adaptor boards and descriptions of registers available on each board is not included in this manual. This information can be found in the hardware manual included with your Bit 3 adaptor.

To simplify installation and eliminate operation problems, we recommend that you review this manual and the appropriate adaptor hardware manual before beginning to install your new Support Software.

About this manual:

* Chapter One introduces each component of the Support Software. It describes how each component is used, and when it is needed.

* Chapter Two gets you started with information about the software package, important notes, and a listing of additional references.

* Chapter Three contains information needed when installing and configuring the software on a system. Any component of the software package that requires special privileges is described in this chapter. Explanations of the relationships between hardware jumper settings and the Support Software are also included.

* Chapter Four is for applications programmers. It describes the use of the interface library used to access the adaptor memory and registers.

* Chapter Five, also intended for applications programmers, describes how to send and receive interrupts using the interface library and the device driver.

* Chapter Six, also for applications programmers, describes the use of the DMA device driver.

* Chapter Seven describes the example programs included with the Support Software and provides information about: each example program's function; notes and advice on using the programs; warnings; and suggestions for using example programs to test the adaptor and software installation.

* Chapter Eight is for users with multiple adaptors installed in a single system.

Important notes:

* All example program runs in this manual use **BOLD** text to indicate information the user enters. Normal text is used for anything that VMS or an application presents. All example sessions are indented from the normal text.

* This manual uses standard C notation to indicate number radix. A leading '0' indicates an octal number; a leading '0x' indicates a hexadecimal number; otherwise the number is assumed to be decimal.

TABLE OF CONTENTS

Chapter One: Support Software Functions

1.0 Executive Overview.....1

1.1 Components.....2
1.1.1 Privileged Processes.....2
1.1.2 Interface Library.....2
1.1.3 Device Driver.....3
1.1.4 Example Programs.....3

1.2 Multiple Adaptor Unit Support.....4

Chapter Two: Getting Started

2.0 Software Package Contents.....5

2.1 Help!.....5

Chapter Three: Installation

3.0 Introduction.....6

3.1 Installing the Software.....6

3.2 Hardware Jumper Settings.....7
3.2.1 VECTOR and I/O Range Settings.....8
3.2.2 Adaptor Memory Ranges.....9

3.3 Loading the Device Driver.....10

3.4 Using the Privileged Processes.....11
3.4.1 BIT3_MAP.....11
3.4.2 BIT3_UNMAP.....15

3.5 Adjusting System Parameters.....15
3.5.1 GBLSECTIONS/GBLPAGES Using SYSGEN...15
3.5.2 WSQUOTA Using AUTHORIZE.....17

3.6 Changing the System Startup Procedure....17

Chapter Four: Using the Interface Library

4.0 Introduction.....19

4.1 Using the Mapping Routines.....20

4.2 Accessing Node I/O and Remote Bus I/O....21

Chapter Five: Interrupt Handling

5.0 Introduction.....23

5.1 Sending Interrupts.....23
5.1.1 PT interrupts.....23
5.1.2 PR Interrupts.....24

5.2 Receiving Interrupts.....25
5.2.1 Registering for an AST.....26
5.2.2 Processing the AST.....26
5.2.3 Communicating Outside an AST.....27

Chapter Six: Using the Device Driver

6.0 Introduction.....28

6.1 Using the $QIO Interface.....28
6.1.1 EFN and CHAN Parameters.....29
6.1.2 Valid FUNC Parameters.....29
6.1.3 IOSB Status Block.....30
6.1.4 Defining an AST Handler.....30
6.1.5 Data Buffer Address and Length.....30
6.1.6 Specifying a Timeout Interval.....31
6.1.7 Specifying Transfer Type/Direction..31

Chapter Seven: Example Programs

7.0 Introduction.....33

7.1 DUMPPORT.C Example Program.....33

7.2 DESTIN.MAR Example Program.....34

7.3 HOST.MAR Example Program.....35
7.3.1 Task One: Remote Bus Memory.....35
7.3.2 Task Two: Dual-Port Memory.....36
7.3.3 Task Three: DMA Device Transfers....36
7.3.4 Task Four: Interrupt Handling.....36

Chapter Eight: Upgrading to Multiple Q22-bus Adaptors

8.0 Introduction.....37

8.1 Changes to BIT3_MAP.....37

8.2 Changes for Q22-bus Device Driver.....38

8.3 Changes to BIT3_IFLIB.....39

APPENDIX A: GLOSSARY.....A-1

APPENDIX B: BIT3_IFLIB Documentation.....A-4

APPENDIX C: Program Listings.....A-26

APPENDIX D: Fortran Programming Example.....A-32

Chapter One -- Support Software Functions

1.0 Executive Overview

Model 400-901/902 Support Software provides a device driver and interface library for the VMS operating system and example programs to help applications programmers with adaptor and system configuration. It currently supports Model 431 Q22-bus to MULTIBUS I, Model 432 Q22-bus to VMEbus, Model 433 Q22-bus to Q22-bus, and Model 436 Q22-bus to A32 VMEbus adaptors.

The software package provides a VMS device driver and interface library, plus all the tools, including memory mapping, to access dual-port and/or remote bus memory space from an application. This allows memory sharing between a VMS Q22-bus system (such as the MicroVAX II and MicroVAX III) and another system.

Bit 3's VMS device driver provides an interface similar to the DRV11-WA, allowing block transfers between application data buffers and remote memory or a remote Bit 3 Dual Port RAM card. Is also allows the registration of interrupts via unsolicited Asynchronous System Traps (ASTs) for use in receiving interrupts from the other system.

Bit 3's interface library allows direct mapping to dual-port and/or remote bus memory, without software overhead. After setup, all access is handled by hardware.

The example programs included in the Support Software demonstrate features of the adaptor hardware and software, and are useful tools for:

* Debugging.
* Testing the hardware.
* Receiving and sending programmed interrupts.
* Displaying the contents of dual-port memory.

Example programs may be modified for your specific hardware configuration.

1.1 Components

Model 400-901/902 Support Software consists of the following components:

* Privileged level routines that set up and delete the system resources used by the interface library routines. These can be added to the system startup/shutdown procedures.

* An interface library to allow access to the device registers and memory mapped access to the remote bus and/or Dual Port RAM.

* A VMS device driver that allows block data transfers and registration for ASTs.

* Example programs in MACRO32 and C.

1.1.1 Privileged Processes

These components require special privileges to operate. The privileged processes are the only parts of the software that need be configured with system-dependent hardware settings, such as the memory windows and the interrupt vector used. Once the system is configured, these procedures can be automatically executed at system startup and shutdown.

If the interface library is used by an application, the BIT3_MAP process must be run to create the system resources required by the interface library. The BIT3_UNMAP process can be executed to deallocate these same system resources, after the application completes. Because the BIT3_MAP and BIT3_UNMAP procedures allocate and deallocate system resources, an application program can be created that does not require any special privileges or configuration information.

If an application will perform DMA transfers or receive interrupts (ASTs), the device driver (BTDRIVER.EXE) must be loaded by SYSGEN, a special application included as part of VMS. This requires special privileges, as well as a knowledge of the adaptor I/O window and the interrupt vector used.

1.1.2 Interface Library

An interface library is provided to directly access and control the adaptor. It provides procedures to:

1) Map the remote bus, dual-port, or local bus memory into the application's virtual address space.
2) Access the adaptor node I/O registers.
3) Access the remote bus I/O registers.

Once an adaptor memory window is mapped into application space, an application can directly access memory in that window. Each of the map procedures returns a pointer to the beginning of the window, and optionally a pointer to the end of the window created by that procedure. By referencing off these pointers (or virtual addresses), the application can directly access the memory window.

Separate procedures are provided to access adaptor or remote bus I/O space. The procedures for accessing adaptor node I/O check that the reference is not to a DMA control register or remote bus I/O. The remote bus I/O procedures also allows access to the node I/O, but not the DMA registers. Access to the DMA Control and Status Registers is prevented so that an application can not interfere with the device driver.

1.1.3 Device Driver

The DMA device driver, BTDRIVER.EXE, provides two functions. It allows for the registration and receipt of interrupts (or ASTs), and it allows a process to perform a block transfer of up to 65534 bytes (32767 words) of data. Both of these services use the system QIO interface. The operation is very similar to that of a DRV11-type device, with the addition of a parameter defining the cable address.

The device driver can only transfer data to/from remote RAM or a remote Dual Port RAM card. It can not transfer data when both the source and destination memories are on the same side of the interface cable.

1.1.4 Example Programs

The example programs provided with Support Software are meant to serve two separate but related purposes. First, they can be used as a simple test to make sure that the hardware and software are configured correctly. Second, they demonstrate how to use various features of the adaptor with the interface library and device driver.

Before running any example program, we recommend that you read this manual and understand what the example programs do.

The example programs read and write to the remote bus. If the other chassis contains a running system (such as another MicroVAX running VMS or a VMEbus running UNIX, with other users currently logged in and using the system), it is possible that running the example programs could interfere with the other system's operation.

1.2 Multiple Adaptor Unit Support

Bit 3 Support Software provides support for one to four adaptors, identified as Units 0, 1, 2, or 3. See Chapter Eight for a detailed description of configuring and programming multiple adaptors. This added functionality is downwards compatible with existing Bit 3 hardware and software configurations.

Chapter Two -- Getting Started

2.0 Software Package Contents

The Support Software package contains the items listed below. Please identify each item and notify Bit 3 (612-881-6955) if any are missing.

Model 400-901 Support Software:

* Two 5.25" RX50 diskettes; Part Number 82602210;
* Q22-bus Adaptor VMS Support Software manual; Pub. number 82602165;
* Software license agreement.

Model 400-902 Support Software:

* Two TK50 compact tapes; Part Number 82602220;
* Q22-bus Adaptor VMS Support Software manual; Pub. number 82602165;
* Software license agreement.

2.1 Help!

Technical support is available from 9 a.m. - 5 p.m. (Central Standard Time) Monday - Friday, excluding holidays.

Contact Bit 3 at:

Mailing Address:	Bit 3 Computer Corporation 8120 Penn Avenue South Minneapolis, MN 55431-1393
Phone:	612-881-6955
FAX:	612-881-9674

Chapter Three -- Installation

3.0 Introduction

This chapter contains information for installing the software and installing/configuring a system with the adaptor hardware. Suggestions for determining the jumper settings on the Q22-bus adaptor board for the IO LO / IO HI (I/O range), REM RAM (Remote RAM), and PORT RAM (dual-port memory) windows, the BIAS (local memory Bias), and the VECTOR (interrupt vector) jumpers are included. Chapter Three also contains information on how to use the privileged level routines.

This chapter is not intended be a complete reference to all available jumpers or as an instruction guide for setting individual jumpers. To correctly configure the Q22-bus adaptor board, you need to read and be familiar with the adaptor hardware manual.

3.1 Installing the Software

Bit 3 Support Software uses the standard VMSINSTAL procedure. The installation kit includes release notes describing the changes made to the software since the last major release.

The software comes on two separate media. One, labeled VMS4, is for use on systems running VMS version 4.4 to 4.7. The other, labeled VMS5, is for systems running VMS version 5.0 or later.

To begin installation, log into the system manager's account and enter the following commands:

```
$ SET DEFAULT SYS$UPDATE
$ ALLOCATE ddau:
$ @VMSINSTAL BIT30nn ddau: OPTIONS N
```

(Where 'nn' is the software version number (ex: BIT3022), and 'ddau:' is the name of the device (ex: MUA0:) in which the media is installed.)

VMSINSTAL will display the current system date and time, and ask if you are satisfied with the the backup of your system disk. If you answer no (N), VMSINSTAL will exit without installing any of the software. This allows you to gracefully exit the VMSINSTAL procedure.

If you answer yes (Y), it will ask you to mount the media in your device. This is either the RX50 floppy diskette or the TK50 tape provided by Bit 3. Once the media is inserted into the proper device, answer yes (Y) and VMSINSTAL will begin software installation.

Assuming you specified 'OPTIONS N' on the VMSINSTAL command line, you will be given a chance to display and/or print the release notes provided with the software. After providing the release notes, the installation procedure will give you a chance to gracefully exit without installing the software.

Once installation is complete, remove the media and DEALLOCATE the drive. You do not need to reboot the system. You will probably want to change the system startup (SYSTARTUP.COM for VMS versions before 5.0, SYSTARTUP_V5 for VMS 5.0 or above) to setup the adaptor. Also, you may need to change some SYSGEN parameters, in which case it will be necessary to reboot the system. The rest of this chapter details these procedures.

3.2 Hardware Jumper Settings

There are five jumper blocks on the Q22-bus adaptor board relevant to configuring the Support Software. The other jumper blocks still need to be set for proper hardware operation, but they do not directly affect the Support Software configuration.

The five jumper blocks and their factory settings are:

	Octal	Hexadecimal
1) Interrupt vector (VECTOR)	0524	0x154
2) I/O Range (IO LO / IO HI)	0762000	0xE400
	0762040	0xE420
3) Remote RAM (REM RAM)	010000000	0x200000
	014000000	0x300000
4) Dual Port RAM (PORT RAM)	004000000	0x100000
	010000000	0x200000
5) Address bias (BIAS)	pass-through	pass-through

The values of the interrupt vector and I/O range are needed before the device driver (BTDRIVER) can be connected by SYSGEN. Usually, the system startup procedure does this.

The values of the I/O range, Remote Bus RAM, Dual Port RAM, and local bus memory are needed before BIT3_MAP can run. Any of these windows except the I/O range can be disabled. There is no software default setting for the I/O range, and the default settings for the other ranges are listed in the table below. All numbers are in hexadecimal, and are the same as the current factory configuration of the Q22-bus adaptor hardware.

Remote bus RAM	0x200000 to 0x300000
Dual Port RAM	0x100000 to 0x200000
Local bus memory	Disabled

3.2.1 VECTOR and I/O Range Settings

Use SYSGEN to determine I/O addresses and interrupt vectors already in use on your system. Usually this is done from the SYSTEM account, although any account with the CMEXEC is able to view the I/O database. To actually load the device driver, you need CMKRNL privilege.

There are 16 possible I/O (or CSR) start addresses for the Bit 3 Q22-bus adaptor board, from 0760000 through 0777000. The minimum I/O window is 32 bytes (040 octal), when IO LO jumpers are set equal to IO HI. If you plan to perform remote bus I/O, the minimum window size is 512 bytes, or 01000 octal.

By eliminating any addresses that conflict with devices listed on the SYSGEN Device Table contained in Appendix B of the *VMS System Generation Utility Manual* (DEC order number AA-LA30A-TE), we can narrow the list of potential I/O addresses to the following:

0760000	0761000	0762000	0763000
0765000	0766000	0771000	0773000
0767000	0770000		

Please note that the last two addresses (0767000 and 0770000) will conflict with other devices on the device table unless the adaptor is set for the minimum (32 byte) I/O window.

Since your system may contain devices not listed in the device table, use SYSGEN to check your own system's configuration. By executing the following commands, SYSGEN will list all the devices, their starting I/O address (the CSR), and the interrupt vector(s) used:

```
$ MCR SYSGEN
SYSGEN> SHOW/CONFIGURATION

     System CSR and Vectors on  1-APR-1989 12:34:56.78

Name: PUA Units:1 Nexus:0 (UBA) CSR:772150 Vector1:774 Vector2:00
Name: LPA Units:1 Nexus:0 (UBA) CSR:777514 Vector1:200 Vector2:00
Name: PTA Units:1 Nexus:0 (UBA) CSR:774500 Vector1:260 Vector2:00

SYSGEN> EXIT
$
```

All CSR and VECTOR addresses will be displayed in octal. You should be able to use the factory setting of 0524 for the interrupt vector. Compare the CSR values listed by SYSGEN with the list of available CSR above, eliminating any conflicts. The factory hardware configuration is 0762000 to 0762040 (Node I/O only, 32 bytes).

3.2.2 Adaptor Memory Ranges

There are 4M bytes of Q22-bus memory space. Of this, the first 248K bytes (496 map registers * 512 bytes) are reserved by VMS for DMA device memory buffers. The MicroVAX II GPX, VAX Station 2000, and other similar graphic systems use the last 256K for a device frame buffer. Adaptor memory windows must fall on 64K byte boundaries (hexadecimal 0x10000), this leaves the space from 0x040000 to 0x3C0000 available for use by the Bit 3 adaptor.

Please note that the Q22-bus can map up to 4M bytes of memory, but VMS requires a number of Q22-bus mapping registers for its own internal operations. If you are installing more than one adaptor in a single system, you must allocate the resources for all of the adaptors within 3.5M bytes for your applications. None of the ranges for any of the adaptors may overlap. If there is not enough space to directly access all resources, you may need to use Page Mode to access some of the remote memory or remote dual-port memory.

With the release of VMS 5.0, the system boot process polls all Q22-bus memory address space to locate any hardware residing there. Any locations that do not respond are put in a list of free pages, which can also be used by a DMA device to transfer a data buffer. To ensure proper system operation, the AWAKEN configuration on the SYST jumper block must always be set for HARDWARE AWAKEN.

There are no programs available to list the parts of the Q22-bus address space currently in use. Most interface boards do not use the Q22-bus memory space. The types of boards that may use this space are graphics boards, memory boards, and other boards that appear as memory boards. For most systems, the factory configuration should not cause any conflicts.

The REM RAM and PORT RAM jumpers on the Q22-bus adaptor board control where remote bus and dual-port memory appear.

To access a local bus memory card, set the BIAS jumpers that translate (or bias) the address from the remote adaptor into a window on the Q22-bus. This allows access to the local bus memory card from the remote adaptor board but restricts the remote adaptor board from accessing other parts of the Q22-bus memory. It alse allows the remote system to function without knowing the hardware address of the Q22-bus memory board.

3.3 Loading the Device Driver

The device driver is usually loaded at the time the system is booted. Before loading the device driver, you must know the location of your I/O window (also referred to as the CSR address) and the interrupt vector.

Initially, log into the SYSTEM account to load the device driver. After making sure that the device driver is configured correctly, edit the system startup procedure to have the device driver automatically loaded.

To be consistent with the SYSGEN utility, this section uses octal addresses for the I/O and interrupt vector, unless otherwise noted. Furthermore, the examples show the commands assuming the factory configuration of IO LO 0762000 and VECTOR 0524.

Log into the SYSTEM account, and enter the following commands:

```
$ MCR SYSGEN
SYSGEN> LOAD BTDRIVER
SYSGEN> CONNECT BTA0 /ADAPTER=UB0/CSR=%0762000/VECTOR=%0524
SYSGEN> EXIT
$
```

The CONNECT command uses the letter 'O' ('=%O') to prefix octal addresses. Be sure to substitute your own I/O address and interrupt vector settings for the CSR and VECTOR addresses.

See Chapter Eight for information about SYSGEN support for multiple adaptors.

Enter the command "**SHOW DEVICE BT /FULL**" to check that the driver loaded correctly. If the device is off-line, either the adaptor hardware is not installed or the wrong I/O address was entered. Use the SHOW/CONFIGURATION command from SYSGEN to check the CSR and VECTOR settings. Refer to the Q22-bus adaptor board and the hardware manual to check that the jumper settings are correct.

3.4 Using the Privileged Processes

The BIT3_MAP and BIT3_UNMAP privileged processes allocate and deallocate system resources required by the interface library. To allocate these resources, an account must have the following privileges:

* PFNMAP -- Map to specific (memory) physical pages
* SYSGBL -- Create system-wide global sections
* PRMGBL -- Create permanent global sections

Do not run BIT3_UNMAP while applications using the memory sections are running. An easy way to make sure these restrictions are obeyed is to only execute BIT3_MAP in the system startup command, at system boot.

You may also want to modify the MODPARAMS.DAT file and run AUTOGEN so that the system can adjust for the resources allocated by BIT3_MAP. Section 3.5 details which parameters may need to be changed and the simplest way to do so.

3.4.1 BIT3_MAP

There are two ways to run BIT3_MAP. The first method assumes the location and size of the dual-port and remote memory windows, and only allows you to adjust the starting address of the I/O window, assuming the size of the I/O window is 32 bytes. It assumes there is no local memory window. The second method allows you to configure any or all of these settings based on your system. Both methods can be used either interactively or from inside a script file.

When the BIT3_MAP process is run, it first asks whether it should create a remote memory window. If you want to configure the software for settings other than the default, answer "C" to this question. The later part of this section explains the available configuration commands. Chapter Eight contains additional information for configuring multiple adaptors.

When you answer "Y" to the remote memory prompt, the program will configure itself for a remote memory window from 0x200000 to 0x300000. It then prompts for a starting I/O or CSR address. If you answer "N", the program does not create a remote memory window.

The CSR address BIT3_MAP requests is the same address used when loading the device driver. See section 3.2.1 for more information on how to determine an I/O window. The address is assumed to be octal, unless prefixed with 'x' in which case hexadecimal notation is assumed.

The next question BIT3_MAP asks is whether you want to create a dual-port memory section. If you answer "Y", the program configures itself for a dual-port memory section from 0x100000 to 0x200000. If you have not already been asked, the program will prompt you for a CSR address.

Finally, BIT3_MAP creates the I/O and memory sections requested. Below is an example session where BIT3_MAP was asked to create the default remote memory and dual-port memory windows, with an I/O window at 0762000 octal.

```
$ RUN BIT3_MAP
BIT3_MAP.EXE  V2.0
Copyright (c) 1988, 1989, 1990 by Bit 3 Computer Corporation
All rights reserved.

If you want to configure the adaptor to locations other than the
factory settings, answer C (CONFIGURE) to the first question.

Would you like to map the remote memory (Y/N/C) ? Y
Valid CSR addresses for the Bit 3 device range from 760000 to 777
What is the CSR address of the Bit 3 device (e.g. 762000) ? 76200
Would you like to map the Dual-Port memory (Y/N) ? Y

Remote RAM mapping established.
Dual-Port memory mapping established.

All done.
$
```

When you answer "C" to the first question (remote memory: Y/N/C), the program displays a "Bit 3>" prompt and waits for a command. The "HELP" command displays a short help screen that lists each of the commands available and their function. Please note that it is not possible to abbreviate the commands.

Many of the commands require an address or range of addresses as parameters. All of these commands have a two-character alias that performs the exact same function. All addresses are assumed to be entered in octal, unless prefixed with 'x' in which case hexadecimal notation is assumed.

The **CSR** (or IO) command allows you to set the starting I/O address. If you enter two addresses, it interprets this as an I/O window, and creates a section larger than the default I/O window of 32 bytes. Initially, there is no default I/O window set. The program will not allow you to "**EXIT**" and create the memory sections unless you set an I/O window.

The I/O window must start on a 512 byte (01000 octal, 0x200 hexadecimal) boundary. The end address must also be on a 512 byte boundary, unless the minimal I/O window is requested. In this case the end address is equal to the start address plus 32. The end address is the first address greater than the last I/O location the adaptor responds to. This is consistent with the way the jumpers are set on the adaptor board.

All of the following commands work in the same way -- to set one of the three adaptor memory windows:

REMOTERAM	RR	The remote RAM window
DUALPORT	DP	The dual-port memory window
LOCALMEM	LM	The local memory window

Each command requires a start and end address, or the word "**DISABLE**", as a parameter. The start and end address must be on a 64K byte boundary (0200000 octal, 0x10000 hexadecimal), and the start address must be less than the end address. The end address is the first location greater than the last memory location the adaptor window responds to. Once again, this is done to be consistent with the way the address ranges are described for the adaptor hardware.

Please refer to section 3.2 of this manual for other restrictions on the I/O and memory windows. The BIT3_MAP process will not accept what it considers to be an invalid address range, but does no further checking to make sure the settings are valid. It is your responsibility to make sure the settings match those of the adaptor hardware, and that no two memory sections overlap.

The "**AU**" and "**ADAPTOR**" commands are used when multiple adaptor units are used. They use the current settings to create memory sections for the specified unit number. Valid unit numbers range from zero to three, inclusive. See Chapter Eight for more information.

Other commands available from the "Bit 3>" prompt are:

HELP	Display a short help screen.
SHOW	Display the current settings for the I/O and memory windows.
EXIT	Exit the program, using the current settings to create any requested memory sections.
QUIT	Exit the program without creating any memory sections.

The "**HELP**" screen provides a brief description of each command. It is not meant to be a complete reference on the use of BIT3_MAP. However, if you already understand what BIT3_MAP is used for, it does give enough information for you to use all of the commands.

The "**SHOW**" command can be used to show any and/or all memory window settings. Entering "**SHOW**" with no parameters gives you information on all of the window settings. Otherwise, enter any combination of the parameters "**RR**", "**DP**", "**LM**", and/or "**IO**" to display only the remote RAM, dual-port memory, local bus memory, or I/O windows, respectively. The settings for each window are displayed in both octal and hexadecimal notation.

After displaying the current settings, the SHOW command also displays a set of recommended settings for the MODPARAMS.DAT file. These lines reflect the system resources BIT3_MAP allocates using the current settings. See section 3.5 for more information about how to use this information.

Please note: BIT3_MAP does not keep a record of the window settings used. The "**SHOW**" command shows you the settings as they are configured for that run. It will not show you the previous settings.

Both the "**EXIT**" and "**QUIT**" commands allow you to leave the program. The "**QUIT**" command exits without attempting to create any I/O or memory windows. The "**EXIT**" command uses the current settings and attempts to create the requested windows. Please note that you must specify an I/O window before the "**EXIT**" command will allow you to leave the program.

3.4.2 BIT3_UNMAP

The BIT3_UNMAP process deallocates any memory sections BIT3_MAP may have created. It is important that this process is NEVER run while an application that uses those memory sections is executing. There is no checking from within the BIT3_MAP or BIT3_UNMAP processes to prevent this type of incorrect usage. It is up to you to make sure this does not happen. Run BIT3_UNMAP to free the system resources for other processes when there is no process running that needs them.

3.5 Adjusting System Parameters

The BIT3_MAP process allocates certain global system resources to allow the mapping procedures contained in the interface library to work. Because it is possible to create sections that are quite large (slightly more than 3M byte total), the system may need tuning to take these new demands into account. This section discusses only the parameters that may need adjustments and the most basic ways to make these adjustments. For a more advanced discussion of these parameters and how they affect system operations, refer to the DEC VMS System Manager documentation.

3.5.1 GBLSECTIONS/GBLPAGES Using SYSGEN

The resources BIT3_MAP allocates require approximately 2100 GBLPAGES per mapped Megabyte and 1 GBLSECTIONS per type of window. Even for a small (64K byte) memory window, about 130 GBLPAGES are required. System performance could be adversely affected if these changes are not taken into account. Setting the values too low could prevent BIT3_MAP from operating, whereas setting the values too high would waste physical memory.

To find exactly how many GBLSECTIONS and GBLPAGES are required, run BIT3_MAP and answer "C" to the initial remote memory query. If you are used to using the configuration abilities of BIT3_MAP, adjust the I/O and memory window settings to your standard configuration.

If you normally use the default settings for the remote memory and dual-port sections, use the "**CSR**" command to set the I/O window. After answering "C" to the remote memory query, the "Bit 3> " prompt appears. To set the the I/O starting address to 0762000 octal, enter "**CSR 762000**". If you normally do not use the remote or dual-port memory, also enter the commands "**RR DISABLE**" or "**DP DISABLE**" to disable these sections.

Once your configuration is set the way it is normally used, enter the SHOW command. If you have more than one adaptor, enter the SHOW command after each adaptor is configured. SHOW gives only the system resources for the current settings. When used with multiple adaptors, note the requirements of each adaptor. After displaying the settings for each section, it will display the number of global pages and global sections these settings require. To leave the program, either enter "**QUIT**" (to leave without creating any sections) or "**EXIT**" (to attempt to use the current settings to create the requested sections).

To configure the system with these new settings, first add this information to the MODPARAMS.DAT file in SYS$SYSTEM. Assuming that you have a single 1M byte window and a 512 byte I/O window, add the following lines to MODPARAMS.DAT:

```
ADD_GBLPAGES    =    2054       ! BIT3_MAP resources
ADD_GBLSECTIONS =    2
```

Substitute the numbers shown by BIT3_MAP for your own configuration. After adding these lines to the MODPARAMS.DAT file, run the AUTOGEN procedure. It is best if the system has been operating at least 24 hours before running AUTOGEN. This allows AUTOGEN to use the system feedback information to better calculate new values for the various system parameters.

The first step is to use AUTOGEN to compute new values for the system parameters. This can be done with the following commands:

```
$ SET DEFAULT SYS$UPDATE
$ @AUTOGEN SAVPARAMS GENPARAMS
```

This procedure invokes AUTOGEN which uses the current feedback information and the MODPARAMS data to calculate new values for all parameters. It creates a new SETPARAMS.DAT file containing the new parameters, and a AGEN$FEEDBACK.REPORT that tells which parameters were changed along with their old and new values. If you are not satisfied with the calculated results, edit the MODPARAMS.DAT files and execute the first AUTOGEN again.

When you are satisfied with the system parameters that AUTOGEN calculates, make sure everyone is logged off the system, then enter the following commands:

```
$ SET LOGIN/INTERACTIVE=0
$ SET DEFAULT SYS$UPDATE
$ @AUTOGEN SETPARAMS REBOOT
```

This tells AUTOGEN to use the newly calculated values for the system parameters, and to reboot the system. It is necessary to reboot the system before the new system parameters will take affect.

If you require more information about SYSGEN, AUTOGEN, or system balancing, see the DEC VMS System Manager documentation.

3.5.2 WSQUOTA Using AUTHORIZE

In addition to adjusting the system parameters, it may be necessary to adjust account parameters so that a process has enough virtual address space in which to execute. This can be done with the AUTHORIZE procedure. One indication that this is necessary is when one of the interface library mapping procedures returns the SS$_INSFWSL error code indicating an insufficient working set limit.

Parameters that you may want to change are WSDEFAULT, WSQUOTA, and WSEXTENT. The WSDEFAULT value is the initial value a process has at image startup. Although the working set size will grow, by setting the initial value closer to the required value you can decrease the number of times VMS has to expand the image size. The WSQUOTA value is the maximum size the working set is allowed to grow when the system is loaded. The WSEXTENT is the absolute working set limit allowed when the system has excess free pages of memory. The working set size will be reduced to WSQUOTA if the system later requires the additional memory for another process.

Please read the VMS documentation on AUTHORIZE before attempting to change any of these parameters.

3.6 Changing the System Startup Procedure

If you are using the BTDRIVER device driver, we strongly recommended that you add lines to your system startup procedure (SYSTARTUP.COM for versions of VMS before 5.0, SYSTARTUP_V5.COM for VMS 5.0 or later) to load and connect the driver at system boot.

Also, if applications will be using the interface library, it may be easier to have the system startup procedure execute the BIT3_MAP process as well. This way, the resources are always available to any applications that may use the adaptor and the accounts will not need a special login script and privileges to run the BIT3_MAP process.

To add either of these to your system startup script, log into the system manager's account and edit either the SYSTARTUP.COM file (for VMS versions less than 5.0) or SYSTARTUP_V5.COM (for VMS versions 5.0 and greater) and add the text you normally enter in response to prompts from SYSGEN (loading the device driver) or BIT3_MAP (allocating system resources.)

The following is a simple example script that loads the device driver with the default settings, and has BIT3_MAP create a 1024 byte I/O section (for doing remote bus I/O accesses), a 1.5M byte remote memory section, and no dual-port or local bus memory section.

```
$ !  Set up the Bit 3 adaptor
$ RUN SYS$SYSTEM:SYSGEN
LOAD BTDRIVER
CONNECT BTA0 /ADAPTER=UB0 /CSR=%O762000/VECTOR=%O524
EXIT
$ RUN BIT3_MAP
C
DUALPORT DISABLE
LOCALMEM DISABLE
REMOTERAM x200000 x380000
IO 762000 764000
EXIT
$
```

Chapter Four -- Using the Interface Library

4.0 Introduction

The BIT3_IFLIB interface library is a standard object module library containing the routines that allow you to map the various memory sections into an application's virtual address space, access the adaptor node I/O registers, and access remote bus I/O registers.

The following procedures are contained in the library:

Mapping procedures:
- BIT3_MAP_RR
- BIT3_MAP_DP
- BIT3_MAP_LM

Node I/O procedures:
- BIT3_READ_REGISTER
- BIT3_WRITE_REGISTER
- BIT3_IACK_READ

Remote bus I/O procedures:
- BIT3_OUT_BYTE
- BIT3_OUT_WORD
- BIT3_IN_WORD

Before any other procedures can be used, one of the BIT3_MAP_ procedures must have successfully completed.

None of the BIT3_MAP_ procedures are re-entrant. All of the other procedures are AST re-entrant.

Appendix B defines each of the procedures, their parameters, and the error codes returned by the procedures.

An application using these procedures would have to include the language-dependent external procedure declarations in their application code, and then link the program with BIT3_IFLIB.

For example, to compile and link the DUMPPORT.C example program, copy both the DUMPPORT.C and DUMPPORT.H files to your own account and execute the following commands:

```
$ CC DUMPPORT
$ LINK DUMPPORT,SYS$LIBRARY:BIT3_IFLIB/LIBRARY
```

The executable file contains only those procedures contained in BIT3_IFLIB that were actually called by the DUMPPORT application.

4.1 Using the Mapping Routines

There are three separate routines that allow an application to directly access sections of the Q22-bus memory space. These sections are the remote bus memory, dual-port memory, and local bus memory windows that the BIT3_MAP process created. Few, if any, applications need to use all three sections.

VMS is a virtual memory system. A combination of hardware and system level software allow the virtual address an application uses to be redirected to a different physical address, or even loaded from mass storage. If the page of virtual memory is already loaded into physical memory, the process is completely handled by hardware support.

The map procedures make use of these same virtual memory mechanisms to create a section of the application's address space which references the Q22-bus memory space. The map procedures return a pointer to the beginning of this special address space. The hardware support for virtual memory will cause any program access to this address space to automatically access the adaptor board on the Q22-bus.

An application needs only to execute any of the mapping procedures once. The special memory section is available to any part of that process until the application completes and returns to VMS.

At least one of the three procedures must be called before any other procedures in the interface library (for node I/O or remote bus I/O) will complete. The first map procedure to be called will automatically do the initialization needed by the I/O routines.

If the error code after calling one of these procedures is SS$_INSFWSL, you have run out of working set space. You will need to have your WSQUOTA and WSEXTENT increased. Please contact the system manager to arrange to have your working set limits increased.

The error code SS$_NOSUCHSEC indicates one of the following conditions:

1) BIT3_MAP was never run, or BIT3_UNMAP has removed the sections that BIT3_MAP created.
2) BIT3_MAP was run, but told not to create that type of memory section.
3) BIT3_MAP was run, but an error prevented it from successfully creating the memory sections.
4) BIT3_MAP was run, but that adaptor Unit number was not configured.

If the board and the software are not configured identically, any access to an adaptor memory window will cause an MCHECK (Machine Check, also known as Q22-bus timeout). This could be caused by configuring the window to a larger window, or an incorrect window, with BIT3_MAP. It is your responsibility to make sure that the software is properly configured.

4.2 Accessing Node I/O and Remote Bus I/O Registers

These procedures are meant to allow an application to read/write to the I/O registers located either on the adaptor board, or on a device existing on the remote bus. They will not allow access to the DMA device registers.

These procedures are AST re-entrant. None will work until one of the three memory mapping procedures has successfully completed. All these procedures accept an optional adaptor Unit parameter, specifying which adaptor to address. This support for multiple adaptors is described in Chapter Eight. The default is adaptor Unit 0 for use with software that only requires a single adaptor.

The BIT3_IACK_READ procedure reads the interrupt acknowledge register, returning the 16-bit interrupt vector.

All other procedures have a register offset parameter that defines which of the registers is being accessed. Please note that this offset starts with the Local Command Register at offset zero. Access to the DMA registers is not allowed, to prevent any possible interference between the device driver and the interface library. A register offset of 16 would be the first remote bus register.

To access remote node or remote bus I/O, the SYST jumper block on the Q22-bus adaptor card must have the "TRANSMITTER" jumper installed. If not, the procedures will return a SS$_DEVINACT error code.

The BIT3_READ_REGISTER and BIT3_WRITE_REGISTER allow an eight-bit value to be read/written to either a local node or remote node adaptor register. If you attempt to access beyond the node I/O space, the procedure will return a value of SS$_BADPARAM.

The BIT3_IN_WORD, BIT3_OUT_WORD, and BIT3_OUT_BYTE procedures allow 16-bit read/write and eight-bit write access to either the adaptor node I/O space or the remote bus I/O space. Accessing the remote bus I/O space allows you to control the registers of a device located on the remote bus. Also, the IO HI jumpers must be set to a value greater than IO LO.

If not enough I/O space is available to position the remote device at an I/O address within this window, the adaptor also allows a memory cycle to be converted into a remote bus I/O access. For the Model 431 Q22-bus to MULTIBUS I and Model 433 Q22-bus to Q22bus adaptors, this is done by setting the select I/O page bit in the Remote Node Command Register. For the Model 432 Q22-bus to VMEbus or Model 436 Q22-bus to A32 VMEbus, this is done by changing the value in the address modifier register. By adding the absolute device register address to the base of the remote memory section (as returned by BIT3_MAP_RR), you can access the device using the remote memory address window. By clearing the bit (Models 431, 433 adaptors) or restoring the normal address modifier (Models 432, 436 adaptors), the remote memory window will be restored to its normal function.

Chapter Five -- Interrupt Handling

5.0 Introduction

The Q22-bus adaptor board is capable of sending out one of two types of programmed interrupts. It can receive and distinguish between any one of eight different interrupts, one of which the device driver software reserves for the DMA Done interrupt.

The RINT and TINT jumper blocks on the Q22-bus adaptor board control which interrupts it sends/receives. Please refer to the sections of the hardware manual for a discussion of how to configure these jumper blocks. We strongly recommend that you leave the RINT jumper block wire-wrap line from DMA DONE to External Interrupt 3 in place. The driver software assumes this configuration.

The example programs demonstrate the concepts described here. It will be helpful to have a listing of the example program to reference while reading this chapter.

5.1 Sending Interrupts

When used on a MicroVAX processor, the only type of interrupts the adaptor can send are programmed interrupts to the remote bus. The BIRQ4-BIRQ7 lines should never be connected on the TINT jumper block when used in these type systems.

The type of programmed interrupt used depends on which adaptor is allowed to transmit across the cable. The SYST "TRANSMITTER" jumper determines if this is allowed.

The PR (Programmed to Receiver) interrupt is used when the "TRANSMITTER" jumper is installed.

The PT (Programmed to Transmitter) interrupt is used when the "TRANSMITTER" jumper is not installed.

5.1.1 PT Interrupts

This interrupt requires a jumper connection on the TINT jumper block. It allows a receiver board (an adaptor board that is not allowed to access the other adaptor board across the cable) to send an interrupt to the remote system.

All processing for sending the PT interrupt is done with Local Node I/O Register accesses. The interrupt handler on the remote system writes to its Remote Node Registers to clear the PT interrupt. This way, the "receiver" board can send an interrupt to the other system without having cable access rights.

To cause a PT interrupt, the PT interrupt flip-flop in the Local Node Command Register must be set. By reading the Local Node Status Register, the application can determine when the remote system has cleared the PT interrupt.

The remote system clears the PT flip-flop in its Remote Command Register. This causes the interrupt signal to go away, and the PT status bit in the Local Status Register to be cleared.

Before the interrupt can occur on the remote system, the TINT jumper block on the Q22-bus adaptor board and an interrupt jumper block on the remote system must be properly configured. The registers show the PT interrupt as set, but no interrupt will be generated until both adaptor boards are jumpered correctly. See the adaptor hardware manual for the location and further explanation of the jumpers controlling the PT interrupt.

If a system sends PT interrupts, it should receive PR interrupts.

5.1.2 PR Interrupts

This interrupt allows a transmitter board (an adaptor board that is allowed to access the other adaptor board across the cable) to send an interrupt to the remote system.

All processing for sending the PR interrupt is done with Remote Node I/O Register accesses. The interrupt handler on the remote system would write to its Local Node Registers in order to clear the PR interrupt. This way, the "receiver" board can acknowledge and clear the interrupt from the other system without having cable access rights.

To cause a PR interrupt, the PR interrupt flip-flop in the Remote Node Command Register must be set. By reading the Remote Node Status Register, the application can determine when the remote system has cleared the PR interrupt.

The remote system clears the PR flip-flop in its Local Command Register. This causes the interrupt signal to go away, and the PR status bit in the Remote Status Register to be cleared. With some of the faster processors now available, the remote system can process the interrupt and clear the status bit between the time the MicroVAX processor causes the interrupt and reads back the status register. This would look to the MicroVAX system as if the PR interrupt status bit was never set.

All jumpers for the PR interrupt are on the remote adaptor board. There are no jumpers on the Q22-bus adaptor board that control PR interrupts. Please refer to the hardware manual for information about these jumper blocks. It is still necessary to have the "TRANSMITTER" jumper in the SYST block installed.

If a system sends PR interrupts, it should receive PT interrupts.

5.2 Receiving Interrupts

To register to receive interrupt messages, the application must use the BTDRIVER.EXE device driver. The application uses a special QIO device call to register for each interrupt, or unsolicited AST it wants to receive.

When an interrupt occurs, one of the parameters passed to the application's AST handler is the DMA Control and Status Register. By looking at bits 9-11 of this register, you can determine which of the eight possible interrupts was passed.

The registration and handling of unsolicited ASTs is similar to the method used by the DRV11-W device driver. If you need more programming examples, we suggest that you look at code for that device.

5.2.1 Registering for an Unsolicited AST

Before registering for an Unsolicited AST, you must have used the $ASSIGN system service to open a communications channel between the device driver and the application.

The code to register for an unsolicited AST in MACRO-32 is:

```
; Register routine AST_TRAP for unsolicited AST from BTA0:

$QIO_S  CHAN = BIT3_CHAN,-          ; Device BTA0:
        FUNC = #<IO$_SETMODE!IO$M_ATTNAST>,-
        P1 = AST_TRAP               ; Address of AST routine
```

where:

CHAN is the channel number returned by the $ASSIGN system service.

FUNC is always the constant value IO$_SETMODE+IO$M_ATTNAST.

P1 is the address of the AST handling routine. This routine is called when the interrupt occurs. If the P1 address is equal to zero, all unsolicited ASTs in the queue are canceled.

You can register for the next AST from within an AST handler to continue to receive the messages. When you no longer want to receive AST messages, perform the ATTNAST call with P1 equal to zero. This cancels any pending requests for unsolicited ASTs.

5.2.2 Processing the AST

In addition to registering for the AST, you need to write an AST handler procedure. Usually the AST handler uses the DMA CSR register value, extracting bits 9-11, and uses this value in a CASE statement. Thus, the handler can immediately determine the type of interrupt that occurred and do only the processing for that interrupt type.

Cases 0, 1, and 2 indicate the receipt of a PR interrupt. The first part of the processing in this case is to write to the Local Node Command Register and clear the interrupt flip-flop.

Case 3 should always be a do-nothing case. Any time a DMA completes, you would receive this message.

Cases 4, 5, 6, and 7 indicate external interrupts. These can be PT programmed interrupts, error interrupts, and/or remote bus interrupts.

If you will be receiving PT interrupts, assign one of the external interrupts to do this. The first part of that section would do a remote command write to clear the PT interrupt.

If you will be receiving error interrupts, assign one of the external interrupts to do this. The adaptor would not require you to do any processing, but you may want to read the Local Node Status Register and then clear the status register by writing to the Local Node Command Register.

If the interrupt was not a PT or error interrupt, a device on the remote bus caused the interrupt. In the case of the MULTIBUS I adaptor, the handler would immediately start processing the interrupt. For the VMEbus and Q22-bus adaptors, execute the BIT3_IACK_READ procedure to acknowledge receipt of the interrupt. Before performing the interrupt vector read on a VMEbus adaptor, the IACK CODE bits must be set in the Remote Node Command Register. These bits correspond to interrupt level being acknowledged. The BIT3_IACK_READ procedure returns the interrupt vector number.

5.2.3 Communicating Outside an AST Routine

Only global variables should be accessed from inside the AST procedure. By using two linked list data structures, you can allocate a number of message buffers for communications between the AST routine and the main program. The AST procedure removes one entry from the free list, fills the buffer with information, and adds the buffer to the received list. The application can look in the received list to check for messages, and move the buffers back to the free list once it has processed the message.

The system service event flag routines are also useful for indicating if an interrupt message came in. By having the interrupt routine set an event flag, the application can either periodically poll that event flag or wait until after the event flag is set. This is also an easy way to synchronize operation between the local and remote systems.

Chapter Six -- Using the Device Driver

6.0 Introduction

In addition to allowing the receipt of interrupts, the BTDRIVER device driver allows an application to transfer large blocks (up to 65534 bytes) of data across the adaptor link. The $QIO system service provides the application interface to the device driver.

This chapter describes the $QIO functions and parameters that the device driver recognizes. It does not repeat the information on registering for unsolicited ASTs (receiving interrupts). Please refer to Chapter Five for this information.

Before using any of the $QIO calls, you must use the $ASSIGN system service to open a communications channel between the device driver and the application.

6.1 Using the $QIO Interface

There are two separate but related $QIO system services -- $QIO and $QIOW. An application can use either of these calls when using the device driver. Both have the same parameters and options.

The $QIO call performs preliminary processing of the parameters and queues the request to the device driver. If the service reports an error, the request is not queued to the driver. It does NOT wait for the transfer to complete before returning control to the application. This allows the application to have several outstanding DMA transfers queued simultaneously. Please note that a success condition value does not mean that the transfer was completed successfully.

The $QIOW waits until the transfer is complete before returning to the application. When this call is used, it is not possible to do any other processing (such as queuing another transfer) until the device driver completes this request. A condition value of success means that the whole operation completed successfully.

The formats of the $QIO and $QIOW system service calls are:

```
SYS$QIO   EFN=EventFlag,          CHAN=DeviceChannel, -
          FUNC=DeviceCommand,     IOSB=StatusBlock, -
          ASTADR=Handler,         ASTPRM=ASTparameter, -
          P1=BufferAddress,       P2=BufferLength, -
          P3=TimeoutSeconds,      P4=TransferType, -
          P5=CableAddress
```

Most of this chapter assumes that FUNC is a device read or write. Unless otherwise indicated, you should assume all parameter descriptions refer to their use with these functions.

6.1.1 EFN and CHAN Parameters

The EFN parameter indicates which event flag to set when the request completes. The $QIO system service can set an event flag upon request completion. Providing an event flag is equivalent to having an AST handler that always issues the $SETEF system service. This is often used with the $QIO system service, allowing an application to detect completion without requiring an AST procedure.

The CHAN parameter points to the device channel returned by the $ASSIGN system service.

6.1.2 Valid FUNC Parameters

The following FUNC device commands are recognized:

WRITEPBLK	WRITEVBLK	WRITELBLK
READPBLK	READVBLK	READLBLK
SETMODE	SETCHAR	
SENSEMODE	SENSECHAR	

The WRITEVBLK, WRITELBLK, READVBLK, and READLBLK functions are all handled the same way. They indicate that this is a device DMA transfer request. It does not matter whether the function is a read or a write. The P4 parameter is used to determine the direction of the transfer.

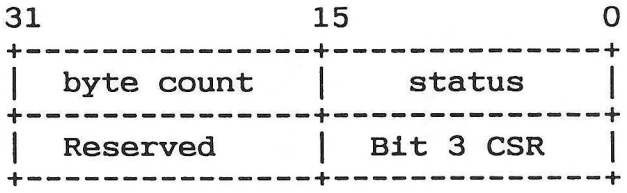
The WRITEPBLK and READPBLK functions can be used the same way as the above functions. The only difference is that they require a special privilege in order to execute.

The SETMODE and SETCHAR functions are used to register for unsolicited ASTs -- such as programmed interrupts, error interrupts, and/or remote bus interrupts. Chapter Five contains an explanation of these functions.

The SENSEMODE and SENSECHAR functions return the standard information any device returns in response to these functions. The device driver does not do any special processing for these functions.

6.1.3 IOSB Status Block

The IOSB parameter points to an aligned quad-word status block. The first word of this value is the final condition value of the request. The complete format of this parameter is defined as follows:



As noted above, the status field is the final condition value indicating the status of the request. The byte count is the number of bytes transferred by the driver. In the case of a successful transfer, this should always match the requested transfer size. In the case of an error, it is possible for this value to slightly exceed the actual number of bytes transferred. The CSR value is the value read from the DMA Control/Status Register at the end of the transfer. The Reserved field is always zero.

6.1.4 Defining an AST Handler

The ASTADR parameter is the address of the entry mask for an AST handler. The ASTPRM value is passed as the first parameter to the AST handler, allowing the application to pass additional information to the AST handler.

This AST handler is executed when the I/O request completes. The application is suspended and the AST handler is called with the ASTPRM value as the first parameter. The AST handler can then use this information to do any special post-processing that may be required. Program execution is suspended until the AST handler completes.

6.1.5 Data Buffer Address and Length

The P1 parameter gives the virtual address of the application data buffer to be used. The P5 parameter gives the physical cable address for the transfer. Both of these values must be word aligned. The P2 parameter is the number of bytes to transfer. This must be an even value.

The device driver transfers data as words. The buffer address (parameter P1) is the virtual address of the application's data buffer. The cable address is the physical address on the remote system where the transfer is to take place.

If both the device driver and the interface library are used, you can convert the virtual address used to access data in the memory window to the physical cable address. By subtracting the base virtual address (as returned in the first parameter by BIT3_MAP_DP or BIT3_MAP_RR) from the address at which the data is located, the result is the cable address the device driver would use.

The device driver can only perform DMA transfers across the cable. The source and destination of the transfer can not be on the same side of the cable. For example, an attempt to perform a DMA transfer to a local dual-port memory would result in a SS$_DRVERR condition value.

6.1.6 Specifying a Timeout Interval

The P3 parameter can be used to specify a timeout interval, in seconds, for the device to wait until the transfer is complete. The timer is accurate plus or minus one second, making the minimum interval two seconds. To specify a timeout interval, add the IO$M_TIMED modifier to the FUNC value. Otherwise the default value of 10 seconds is used.

Since the Bit 3 Q22-bus adaptor board is capable of transferring data in excess of 500K bytes/second, it usually takes less than one second to transfer the maximum size buffer of data. Therefore, it is unlikely that you will need to use this parameter.

6.1.7 Specifying Transfer Type/Direction

The P4 parameter determines in which direction the transfer will be, and whether the transfer will be to remote bus memory or remote dual-port memory. The parameter is the same value that the device driver moves into bits 1-3 of the DMA Control/Status Register.

The following values can be used to determine the direction and type of transfer:

0 = Transfer from application data buffer to remote bus memory
1 = Transfer from remote bus memory to application data buffer
4 = Transfer from application data buffer to dual-port memory
5 = Transfer from dual-port memory to application data buffer

By adding two to the value, the device can be reset before beginning the transfer.

Another way to view these values is as additive bits, each bit defining a function and the sum giving the complete code. For example, bit 0 indicates the direction of transfer, when set the transfer would be a write from the application buffer to the remote. Bit 1 causes the adaptor DMA registers to be reset before the transfer is begun. Bit 2 determines whether the remote memory is bus memory or dual-port memory, when set the transfer will use remote dual-port memory. These bits are described in the adaptor hardware manual.

Chapter Seven -- Example Programs

7.0 Introduction

Three example programs are provided with the Bit 3 Support Software. All of these programs are located in the BIT3 sub-directory of the SYS$EXAMPLES account. A listing of DUMPPORT is also contained in Appendix C of this manual. Please refer to the actual source code provided rather than depend on the listings in Appendix C; the programs may have been updated since the last printing of this manual.

These programs support multiple adaptors. If your system is configured with one adaptor, that adaptor is Unit 0. You can specify one of four possible adaptors. See Chapter Eight for multiple Q22-bus adaptor support.

The programs included with the software are:

DUMPPORT.C	Displays the first few characters of the dual-port memory both as hexadecimal bytes and characters.
DESTIN.MAR	Example program demonstrating the software and hardware features that the "RECEIVER" system may use.
HOST.MAR	Example program demonstrating the software and hardware features that the "TRANSMITTER" system may use.

For your convenience, both the source and executable files for each of these programs is provided.

Each of these programs is described individually in this chapter.

7.1 DUMPPORT.C Example Program

This program consists of two files, DUMPPORT.H and DUMPPORT.C. It uses the BIT3_IFLIB library to access a dual-port memory.

To compile and link the program, you must have a C compiler. We use the VAX C product available from DEC. Using VAX C, the commands to compile and link the program would be:

```
$ CC DUMPPORT
$ LINK DUMPPORT,SYS$LIBRARY:BIT3_IFLIB/LIBRARY
```

If you have the optional dual-port memory, this is the first program to run after installing the hardware and running the privileged process BIT3_MAP. There is nothing in this application that should interfere with the remote system's operation.

If this program is able to complete successfully, the I/O and dual-port starting addresses are set correctly. Assuming you have some way to write into the dual-port from the other system, this provides an easy way to dump the beginning of the section from VMS.

This program also demonstrates how to use pointers in the C language to locate data contained in dual-port memory. Although it only references bytes of data, any data type or structure can be put into dual-port memory.

Note that the data format of DEC floating-point numbers does not conform to the IEEE 754 standard that many other systems use. If floating-point values are passed between systems, it may be necessary to write a routine that converts the format between the DEC and IEEE floating types.

In addition, some processors use a different byte ordering scheme than DEC VAX. The adaptor has a byte and word swapping capability to aid in any conversions that may be needed. Please refer to the adaptor hardware manual for more information about byte and word swapping.

7.2 DESTIN.MAR Example Program

This program demonstrates many of the features that a "RECEIVER" board usually uses. None of the code in this example requires an access across the cable. The program assumes that the dual-port memory is on the local Q22-bus adaptor board.

This program performs three separate tasks:

1) Access local bus memory
2) Access dual-port memory
3) Send PT and receive PR interrupts

Of these, only the third task requires special setup and jumpering of the adaptor boards. Task Three is also the only one likely to cause any interference with the operation of the remote system. Since it sends a PT interrupt and expects the remote system to clear it, do not execute this task until you are writing interrupt code and are ready for preliminary testing.

The third task also requires that the device driver be loaded correctly to receive the interrupt.

7.3 HOST.MAR Example Program

This program is by far the longest and most detailed of the example programs. It demonstrates the features that a "TRANSMITTER" board may want to use. It is also the most likely to interfere with the operation of the other system.

The program performs the following tasks:

1) Perform several different types of remote bus access, including both direct and Page Mode addressing
2) Access the dual-port memory
3) Send and read back data to both the remote bus and dual-port memory using the DMA device driver.
4) Send a PR and receive a PT interrupt

7.3.1 Task One: Remote Bus Memory

First, the remote bus memory is mapped into applications space, and the adaptor board is set up.

The remote bus memory is byte-filled with a pattern of 0x55 hexadecimal. After the whole window is filled, it is read back to make sure that nothing changed. If the remote system used this memory to store operational data (such as the vector table most processors put in low memory), it is likely that running this section will interfere with the other system's operation.

The program reports when it finishes the byte test, and gives an error message if the data did not match what was written.

The next section of Task One uses Page Mode to access the remote system. It inverts the value located at address 0x1000, which once again could interfere with a running system on the remote side. It attempts to restore the value (inverts it again) when it is done with the initial test of Page Mode.

After the initial test of Page Mode, the example uses Page Mode to determine (by accessing at 4K byte boundaries and testing for errors) how much contiguous memory the remote system has starting at address zero.

Since it starts the test at address zero, some systems may report OK of remote memory. This is not an error -- it just means that there is no memory card at address zero.

The final section of Task One demonstrates use of the Handshake Mode. Handshake Mode is a special type of access that allows the Q22-bus system to read and write to devices that can not be accessed fast enough to prevent a Q22-bus timeout.

Most applications will not need to use this mode of operation. However, since Handshake Mode access is more involved than a normal access, we felt that an example of its use was in order.

7.3.2 Task Two: Dual-Port Memory

The only real difference between this task and Task Two of the DESTIN.MAR example program is in the SETUP procedure. Since the HOST.MAR example program is for TRANSMITTER systems, the SETUP procedure is more complex.

There is nothing in Task Two that should interfere with the operation of a remote system.

7.3.3 Task Three: DMA Device Transfers

Task Three writes over the same section of remote bus memory that Task One initially filled with the hexadecimal 0x55 pattern. It also attempts to transfer data to a remote dual-port memory.

If you could safely run the first section of Task One, this section should not cause any problems. If you do not have a dual-port memory, or it is located on the local Q22-bus adaptor board, the program will exit with a fatal device error. Other than causing the program to exit, this should not cause any problems.

Task Three requires the use of the device driver, but does not require the interface library.

7.3.4 Task Four: Interrupt Handling

Since Task Four sends and receives interrupts, it could interfere with the operation of the remote system. It also requires that both adaptors be correctly jumpered to send/receive interrupts.

Because it sends a PR interrupt and expects the remote system to clear it, do not execute this task until you are writing interrupt code and want to do preliminary testing.

Also, users of the VMEbus adaptor board should pay special attention to the interrupt handler. This demonstrates how to set the IACK bits of the Remote Node Command Register, by assuming that VMEbus IRQ3 is jumpered to External Interrupt 7.

CHAPTER EIGHT -- Upgrading to Multiple Q22-bus Adaptors

8.0 Introduction

Bit 3 Support Software release 2.2 supports up to four Q22-bus adaptors. Earlier versions supported a single adaptor. This version of the Support Software is downwards compatible with earlier releases. The software now allows the user application to optionally specify an adaptor Unit in the range from 0 to 3 for one of four possible adaptors. If you do not specify an adaptor Unit, Unit 0 is assumed. This allows existing applications using a single adaptor to function correctly without change or recompilation.

8.1 Changes to BIT3_MAP

The adaptor Unit is specified during system configuration by running the BIT3_MAP program. This program has been modified to support the menu options "AU [unit]" and "ADAPTOR [unit]". Use the SHOW command to see all menu options available for the system administrator to configure one or more adaptors. After configuring the various memory partitions, use either the ADAPTOR [unit], or shortened form AU [unit] command, to allocate and map the memory, that was previously specified.

The following example demonstrates how to allocate and map memory for two adaptors via the CONFIGURE option:

```
$RUN BIT3_MAP

Bit 3> CSR 762000
Bit 3> SHOW
                                Octal                Hexadecimal
I/O range:                   162000   162040      0xE400    0xE420
Remote RAM range:          10000000 14000000    0x200000  0x300000
Dual-Port range:            4000000 10000000    0x100000  0x200000
Local bus memory range:    DISABLED.

Recommended MODPARAMS.DAT settings for current configuration:

                    ADD_GLBPAGES:          4122
                    ADD_GBLSECTIONS:          3

Bit 3> ADAPTOR 0

Remote RAM (Adaptor 0) mapping established.
Dual-Port  (Adaptor 0) mapping established.
```

```
Bit 3> CSR 763000
Bit 3> DP X300000 X320000
Bit 3> RR X380000 X3C0000

Bit 3> SHOW

                                Octal                Hexadecimal
I/O range:                   163000   163040      0xE600    0xE620
Remote RAM range:          16000000 17000000    0x380000  0x3C0000
Dual-Port range:           14000000 14400000    0x300000  0x320000
Local bus memory range:     DISABLED.


Recommended MODPARAMS.DAT settings for current configuration:

                    ADD_GLBPAGES:          782
                    ADD_GBLSECTIONS:         3

Bit 3> AU 1

Remote RAM (Adaptor 1) mapping established.
Dual-Port  (Adaptor 1) mapping established.

Bit 3> EXIT
```

In this example, the system has to be configured to allocate 4904 global pages and six global sections. The menu selections took the default settings for the first board, and specified a 128K byte dual-port window, as well as a 256K byte remote memory window. See Section 3.5 for a description of how to adjust system parameters.

8.2 Changes for Q22-bus Device Driver Configuration

During system configuration, the system administrator needs to SYSGEN as many Bit 3 Device Drivers as there are adaptor Units. In the start up procedure SYSTARTUP.COM, the following commands need to be added to support four adaptors. The example assumes the CSRs and the Vectors are available:

```
SYSGEN> LOAD BTDRIVER

SYSGEN> CONNECT BTA0 /ADAP=UBA0/CSR=%O762000/VECTOR=%O524

SYSGEN> CONNECT BTB0 /ADAP=UBA0/CSR=%O763000/VECTOR=%O530

SYSGEN> CONNECT BTC0 /ADAP=UBA0/CSR=%O764000/VECTOR=%O534

SYSGEN> CONNECT BTD0 /ADAP=UBA0/CSR=%O765000/VECTOR=%O540

SYSGEN> EXIT
```

8.3 Changes to BIT3_IFLIB

At run time, the application interfaces to the Bit 3 adaptor board by way of interface library routines in BIT3_IFLIB.OLB. In the absence of a unit specification, the library routines default to Unit 0. The calling syntax is illustrated below with examples in Pascal and Macro32 that use adaptor Unit 2. See Appendix D for a Fortran programming example.

Pascal:

```
STATUS := BIT3_MAP_DP( VAR FWA : POINTER;
                       VAR LWA : POINTER;
                           UNT : INTEGER := 2);
```

Macro32:

```
CALLG  ARGS,BIT3_MAP_DP

ARGS:  .LONG 3
       .ADDRESS FWA
       .ADDRESS LWA
       .LONG    2                    ; Adaptor Unit
```

Appendix A: Glossary

The following terms are used throughout this manual to describe the components of a system using the Bit 3 Q22-bus adaptor board.

"0":
Zero.

"1":
One.

Adaptor Node Input/Output:
Any access to the I/O registers contained on either the local or remote adaptor board. These are referred to as local note I/O and remote node I/O, respectively.

AST:
Asynchronous System Trap. A VMS mechanism that allows a routine to be called outside the normal linear flow of the program. The Bit 3 device driver uses unsolicited ASTs to send a software interrupt to applications when the device interrupts the Q22-bus system.

Byte:
8 bits.

Cable Interrupt:
An interrupt sent from a device on the remote system across the interface cable. The PT programmed interrupt also comes across the interface cable, but is considered as separate from the other cable interrupts.

Direct Memory Access Transfers (DMA):
The adaptor may be programmed to transfer large blocks of data across the cable to or from the remote bus, rather than requiring a processor to move data.

Dual Port Memory:
An optional Dual Port RAM board attached to either the local or remote adaptor board.

Exchanging Interrupts:
Sending interrupts to and receiving interrupts from the remote bus. This would also include any processing an application should do to acknowledge the receipt of an interrupt.

Hex:
Hexadecimal notation. A numbering system that uses 16 digits (0123456789ABCDEF) to denote a number. This manual follows the Standard C notation conventions of preceding hexadecimal numbers with '0x' to indicate hexadecimal notation.

K byte:
Kilobyte. Two to the tenth power (exactly 1024) bytes.

Local:
Pertaining to the system accessing either of the adaptor boards. Implies that it is not necessary to go across the interface cable to access the resource.

Local Bus Memory:
Any access to a memory board on the Q22-bus which the other system is allowed access to. The other system would consider this memory as remote bus memory. Note that this must be a Q22-bus memory board, not an LMI memory board. An example of such a memory board would be the DEC MSV11-PK 256K Q22-bus memory board.

M byte:
Megabyte. Two to the twentieth power (exactly 1,048,576) bytes.

MB/sec:
Megabytes per second. Example 1,000,000 bytes per second.

Octal:
Octal notation. A numbering system that uses eight digits (01234567) to denote a number. This manual follows the Standard C notation convention of preceding octal numbers with '0' to indicate octal notation.

Programmed Interrupts:
Interrupts caused by setting a flip-flop in one of the adaptor node I/O registers. The two types of programmed interrupts are the PT (Programmed to Transmitter) interrupt and the PT (Programmed to Receiver) interrupt.

Receiver:
An adaptor board that is not allowed to transmit messages across the interface cable. This would prevent it from accessing the Remote Node I/O, Remote Bus I/O, and Remote Bus memory, or a remotely-installed Dual Port RAM board.

Remote:
Pertaining to the systems accessing either of the adaptor boards. Implies that the resource is located at the other end of the adaptor interface cable.

Remote Bus Input/Output:
Any access to the I/O registers of devices that are physically located on the remote system bus (NOT the remote adaptor board).

Remote Bus Interrupts:
Interrupts generated by devices on the remote bus that are passed, via cable interrupt lines, to software residing on the local system.

Remote Bus Memory:
Any access to the memory space on the remote bus. This may be a shared memory section, a device buffer, or any device that responds to a memory access. This does not include Dual Port RAM located on the remote adaptor board.

Transmitter:
An adaptor board that is allowed to initiate message transfers across the interface cable. There must always be at least one transmitter in any pair of adaptor boards.

Word:
16 bits.

Appendix B: BIT3_IFLIB Documentation

BIT3_MAP_RR

Maps the remote bus memory window into the process' virtual address space, and returns a pointer to the beginning of this space. The adaptor I/O registers are also mapped at this time if they were not previously mapped.

VMS PASCAL interface:

```
FUNCTION BIT3_MAP_RR(VAR RR_ADDRESS : POINTER;
                     [VAR RR_ENDADDR : POINTER;]
                     [UNIT : INTEGER]) : INTEGER; EXTERNAL;
```

Returns:

type: cond_value
access: write only
mechanism: by value

Longword condition value, returned in R0. See "Condition Values Returned" below for a list of possible values.

Arguments:

RR_ADDRESS
type: address
access: write only
mechanism: by reference

The starting address of the remote memory window. The value returned will be zero if an error occurs in the procedure call. The address will be in the process' P0 region.

Optional Arguments:

RR_ENDADDR
type: address
access: write only
mechanism: by reference

The ending address of the remote memory window. The value is undefined if an error occurs on the procedure call.

The size of the remote memory window can be computed by:

```
RR_SIZE = 1 + RR_ENDADDR - RR_ADDRESS
```

UNIT
type: longword_unsigned
access: read only
mechanism: by value

The adaptor Unit used to access remote memory. In the absence of this parameter, this procedure assumes Unit 0.

Description:

This procedure maps the global sections created by the privileged process BIT3_MAP into an applications virtual address space. A pointer to the beginning of this address space is returned by the procedure.

The Remote RAM global section is mapped into the P0 address space. The I/O register section is mapped into the P0 address space.

When an application needs to reference a section of the remote memory, it is only necessary to index off of the pointer. The address translation and adaptor access are transparent to the application.

Condition Values Returned:

SS$_NORMAL	Successful execution.
SS$_NOSUCHSEC	The privileged application, BIT3_MAP, has not successfully completed, or was not told to create a remote memory section.
SS$_INSFWSL	The working set limit of the process is not large enough to accommodate the increased virtual address space. Increase the WSQUOTA for the process.
SS$_VASFULL	The virtual address space for the process is full; no space is available to map the section.
SS$_BADPARAM	An invalid adaptor Unit was specified.

BIT3_MAP_DP

Maps the dual-port memory window into the process' virtual address space, and returns a pointer to the beginning of this space. The adaptor I/O registers are also mapped at this time if they were not previously mapped.

VMS PASCAL interface:

```
FUNCTION BIT3_MAP_DP(VAR DP_ADDRESS : POINTER;
                     [VAR DP_ENDADDR : POINTER;]
                     [UNIT : INTEGER]) : INTEGER; EXTERNAL;
```

Returns:

type: cond_value
access: write only
mechanism: by value

Longword condition value, returned in R0. See "Condition Values Returned" below for a list of possible values.

Arguments:

DP_ADDRESS
type: address
access: write only
mechanism: by reference

The starting address of the dual-port memory window. The value returned will be zero if an error occurs in the procedure call. The address will be in the process' P0 region.

Optional Arguments:

DP_ENDADDR
type: address
access: write only
mechanism: by reference

The ending address of the dual-port memory window. The value is undefined if an error occurs on the procedure call.

The size of the dual-port memory window can be computed by:

```
DP_SIZE = 1 + DP_ENDADDR - DP_ADDRESS
```

UNIT
type: longword_unsigned
access: read only

mechanism: by value

The adaptor Unit used to access dual-port memory. This procedure assumes Unit 0 by default.

Description:

This procedure maps the global sections created by the privileged process BIT3_MAP into an application's virtual address space. A pointer to the beginning of this address space is returned by the procedure.

The dual-port global section is mapped into the P0 address space. The I/O register section is mapped into the P0 address space.

When an application needs to reference a section of the dual-port memory, it is only necessary to index off of the pointer. The address translation and adaptor access are transparent to the application.

Condition Values Returned:

SS$_NORMAL	Successful execution.
SS$_NOSUCHSEC	The privileged application, BIT3_MAP, has not successfully completed, or was not told to create a dual-port memory section.
SS$_INSFWSL	The working set limit of the process is not large enough to accommodate the increased virtual address space. Increase the WSQUOTA for the process.
SS$_VASFULL	The virtual address space for the process is full; no space is available to map the section.
SS$_BADPARAM	An invalid adaptor Unit was specified.

BIT3_MAP_LM

Maps the local bus memory window into the process' virtual address space, and returns a pointer to the beginning of this space. The adaptor I/O registers are also mapped at this time if they were not previously mapped.

VMS PASCAL interface:

```
FUNCTION BIT3_MAP_LM(VAR LM_ADDRESS : POINTER;
                     [VAR LM_ENDADDR : POINTER;]
                     [UNIT : INTEGER]) : INTEGER; EXTERNAL;
```

Returns:

type: cond_value
access: write only
mechanism: by value

Longword condition value, returned in R0. See "Condition Values Returned" below for a list of possible values.

Arguments:

LM_ADDRESS
type: address
access: write only
mechanism: by reference

The starting address of the local bus memory window. The value returned will be zero if an error occurs in the procedure call. The address will be in the process' P0 region.

Optional Arguments:

LM_ENDADDR
type: address
access: write only
mechanism: by reference

The ending address of the local bus memory window. The value is undefined if an error occurs in the procedure call.

The size of the local bus memory window can be determined by:

```
LM_SIZE = 1 + LM_ENDADDR - LM_ADDRESS
```

UNIT
type: longword_unsigned
access: read only
mechanism: by value

The adaptor Unit used to access local memory. This procedure assumes Unit 0 by default.

Description:

This procedure maps the global sections created by the privileged process BIT3_MAP into an application's virtual address space. A pointer to the beginning of this address space is returned by the procedure.

The local bus memory section is mapped into the P0 address space. The I/O register section is mapped into the P0 address space.

When an application needs to reference a section of the local bus memory, it is only necessary to index off of the pointer. The address translation and adaptor access are transparent to the application.

Condition Values Returned:

SS$_NORMAL	Successful execution.
SS$_NOSUCHSEC	The privileged application, BIT3_MAP, has not successfully completed, or was not told to create a local memory section.
SS$_INSFWSL	The working set limit of the process is not large enough to accommodate the increased virtual address space. Increase the WSQUOTA for the process.
SS$_VASFULL	The virtual address space for the process is full; no space is available to map the section.
SS$_BADPARAM	An invalid adaptor Unit was specified.

BIT3_READ_REGISTER

Reads data from one of the local or remote Bit 3 adaptor registers. All registers are accessed as byte size.

VMS PASCAL interface:

```
FUNCTION BIT3_READ_REGISTER(VAR REGISTER_OFFSET : INTEGER;
            VAR REGISTER_VALUE : INTEGER;
            [UNIT : INTEGER]) : INTEGER; EXTERNAL;
```

Returns:

type: cond_value
access: write only
mechanism: by value

Longword condition value, returned in R0. See "Condition Values Returned" below for a list of possible values.

Arguments:

REGISTER_OFFSET
type: longword_unsigned
access: read only
mechanism: by reference

Offset from the address of the Local Command Register to the register to be accessed. Can vary from 0 to 15 decimal.

See the adaptor hardware manual for a list of the specific registers available and an explanation of what they do.

In order to read from remote bus I/O registers, use the BIT3_IN_WORD procedure instead.

REGISTER_VALUE
type: longword_unsigned
access: write only
mechanism: by reference

The value read from the adaptor register. Only the lower eight bits are from the register. The upper bits are always zero.

Optional Arguments:

UNIT
type: longword_unsigned
access: read only
mechanism: by value

Adaptor Unit used for read register operations. This prodecure assumes Unit 0 by default.

Description:

The application uses this procedure to read from the registers of the Bit 3 adaptor. To access the registers on the other bus, use the BIT3_IN_WORD procedure.

For a description of available local and remote node adaptor registers, refer to the adaptor hardware manual.

When a function call returns the value SS$_DEVINACT, the procedure was unable to access the adaptor register. The most likely causes are:

* There is no adaptor board in the system.
* The I/O space selected via the IO jumpers at location B9 does not agree with the I/O address range given BIT3_MAP.
* If the access was to the remote registers (REGISTER_OFFSET >= 8), the transmitter jumper in the SYST block (location C1) is not installed.

Condition Values Returned:

SS$_NORMAL	Successful execution.
SS$_OPINCOMPL	None of the _MAP_ procedures (BIT3_MAP_RR, BIT3_MAP_DP, or BIT3_MAP_LM) successfully completed. One of these procedures must complete before the BIT3_READ_REGISTER procedure can complete.
SS$_DEVINACT	The routine could not access the adaptor hardware.
SS$_BADPARAM	The value of REGISTER_OFFSET was greater than 15, or an invalid adaptor Unit was specified.

BIT3_WRITE_REGISTER

Writes data to one of the local or remote Bit 3 adaptor registers. All registers are accessed as byte size.

VMS PASCAL interface:

```
FUNCTION BIT3_WRITE_REGISTER( VAR REGISTER_OFFSET : INTEGER;
            VAR REGISTER_VALUE : INTEGER;
            [UNIT : INTEGER]) : INTEGER; EXTERNAL;
```

Returns:

Type: cond_value
access: write only
mechanism: by value

Longword condition value, returned in R0. See "Condition Values Returned" below for a list of possible values.

Arguments:

REGISTER_OFFSET
type: longword_unsigned
access: read only
mechanism: by reference

Offset from the address of the Local Command Register to the register to be accessed. Can vary from 0 to 15 decimal.

See the adaptor hardware manual for a list of the specific registers available and an explanation of what they do.

To write to remote bus I/O registers, use the BIT3_OUT_BYTE or BIT3_OUT_WORD procedures.

REGISTER_VALUE
type: longword_unsigned
access: read only
mechanism: by reference

The value to write to the adaptor register. Only the lower eight bits of the value are actually written to the register.

Optional Arguments:

UNIT
type: longword_unsigned
access: read only
mechanism: by value

Adaptor Unit used for write register operations. This procedure assumes Unit 0 by default.

Description:

The application uses this procedure to write to the registers of the Bit 3 adaptor. To access the registers on the other bus, use the BIT3_OUT_BYTE or BIT3_OUT_WORD procedures.

For a description of available local and remote node adaptor registers, refer to the adaptor hardware manual.

When a function call returns the value SS$_DEVINACT, the procedure was unable to access the adaptor register. The most likely causes are:

* There is no adaptor board in the system.
* The I/O space selected via the IO jumpers at location B9 does not agree with the I/O address range given BIT3_MAP.
* If the access was to the remote registers (REGISTER_OFFSET >= 8), the transmitter jumper in the SYST block (location C1) is not installed.

Condition Values Returned:

SS$_NORMAL	Successful execution.
SS$_OPINCOMPL	None of the mapping procedures (MAP_RR, MAP_DP, or MAP_LM) has successfully completed.
SS$_DEVINACT	The routine could not access the adaptor hardware.
SS$_BADPARAM	The value of REGISTER_OFFSET was greater than 15, or an invalid adaptor Unit was specified.

BIT3_IACK_READ

Causes the remote adaptor to perform an interrupt acknowledge cycle and returns the interrupt vector generated on the remote bus.

VMS PASCAL interface:

```
FUNCTION BIT3_IACK_READ(VAR VECTOR : INTEGER;
                        [UNIT : INTEGER]) : INTEGER; EXTERNAL;
```

Returns:

Type: cond_value
access: write only
mechanism: by value

Longword condition value, returned in R0. See "Condition Values Returned" below for a list of possible values.

Arguments:

VECTOR
type: longword_unsigned
access: write only
mechanism: by reference

The 16-bit interrupt vector returned from the remote bus. The precise meaning of this number is system dependent.

Usually, it indicates which of the boards on the system caused the interrupt. However, it is possible for more than one board to return identical vectors.

Optional Arguments:

UNIT
type: longword_unsigned
access: read only
mechanism: by value

Adaptor Unit used to send and receive interrupts. This procedure assumes Unit 0 by default.

Description:

An application normally uses this procedure only when bus arbitration and control are being driven by the remote Bit 3 adaptor board. Normally, the processor in a system handles all interrupt acknowledge cycles in a system-dependent way.

This function causes the remote adaptor to perform an interrupt acknowledge cycle, and returns a 16-bit interrupt vector. Usually, this is done in response to receiving an interrupt across the adaptor interface.

This function is equivalent to doing a BIT3_IN_WORD call with the REGISTER_OFFSET set to 14.

Note that before performing a BIT3_IACK_READ to a VMEbus system (Model 432 or 436 adaptor), it is necessary to set the IACK code bits (bits 0,1,2) of the Remote Node Command Register to the VMEbus interrupt level being acknowledged. See the interrupt handler section of the example program provided with the Bit 3 software package for an example.

The MULTIBUS I adaptor set does not support the BIT3_IACK_READ feature.

When a function call returns the value SS$_DEVINACT, the procedure was unable to access the adaptor register. The most likely causes are:

* There is no adaptor board in the system.
* The I/O space selected via the IO jumpers at location B9 does not agree with the I/O address range given to BIT3_MAP.
* The transmitter jumper in the SYST block (location C1) is not installed.

Condition Values Returned:

SS$_NORMAL	Successful execution.
SS$_OPINCOMPL	None of the _MAP_ procedures (BIT3_MAP_RR, BIT3_MAP_DP, or BIT3_MAP_LM) successfully completed. One of these procedures must complete before the BIT3_IACK_READ procedure can complete.
SS$_DEVINACT	The routine could not access the adaptor hardware.
SS$_BADPARAM	An invalid adaptor Unit was specified.

BIT3_IN_WORD

Reads data from one of the Bit 3 local or remote node adaptor registers, or a remote bus I/O register. The registers are read as 16-bit quantities.

VMS PASCAL interface:

```
FUNCTION BIT3_IN_WORD(VAR REGISTER_OFFSET : INTEGER;
                      VAR REGISTER_VALUE : INTEGER;
                      [UNIT : INTEGER]) : INTEGER; EXTERNAL;
```

Returns:

Type: cond_value
access: write only
mechanism: by value

Longword condition value, returned in R0. See "Condition Values Returned" below for a list of possible values.

Arguments:

REGISTER_OFFSET
type: longword_unsigned
access: read only
mechanism: by reference

Offset from the address of the Local Command Register to the register to be accessed.

If the value is less than 16, one of the local or remote node registers on the adaptor set will be accessed. If the value is greater than 15, a word I/O will be done on the remote bus.

Note that the REGISTER_OFFSET must be greater than 15 in order to skip the local and remote adaptor registers.

See the adaptor hardware manual for more information.

REGISTER_VALUE
type: longword_unsigned
access: write only
mechanism: by reference

The value read from the register. The upper 16-bits of the longword value returned are always cleared.

Optional Arguments:

UNIT
type: longword_unsigned
access: read only
mechanism: by value

Adaptor Unit used for input word operations. This prodecure assumes Unit 0 by default.

Description:

The application uses this procedure to read from the Bit 3 adaptor or remote bus I/O registers.

When the value of REGISTER_OFFSET is less than 16, the I/O reference is to a Bit 3 local or remote node adaptor register. Otherwise, the remote adaptor causes an I/O read cycle on the remote bus.

To compute the physical I/O address of the access, determine the IO LO setting of the jumpers at location B9 of the Q22-bus adaptor board. Add 0x10 to this base address, plus the REGISTER_OFFSET value.

When a function call returns the value SS$_DEVINACT, the procedure was unable to access the register. The most likely causes are:

* There is no adaptor board in the system.
* The I/O space selected via the IO jumpers at location B9 does not agree with the I/O address range given BIT3_MAP.
* The IO LO and IO HI settings are the same at location B9. To access the remote bus I/O using this function, set IO HI greater than IO LO. See the adaptor hardware manual for more information.
* If the access was to the remote registers (REGISTER_OFFSET >= 8), the transmitter jumper in the SYST block (location C1) is not installed.

Condition Values Returned:

SS$_NORMAL	Successful execution.
SS$_OPINCOMPL	None of the _MAP_ procedures (BIT3_MAP_RR, BIT3_MAP_DP, or BIT3_MAP_LM) successfully completed. One of these procedures must complete before the BIT3_IN_WORD procedure can complete.

SS$_DEVINACT	The routine could not access the adaptor hardware.
SS$_BADPARAM	The value of REGISTER_OFFSET was greater than the size of the I/O section, or an invalid adaptor Unit was specified.

BIT3_OUT_BYTE

Writes data from one of the local or remote node Bit 3 adaptor registers, or a remote bus I/O register. The registers are written as 8-bit quantities.

VMS PASCAL interface:

```
FUNCTION BIT3_OUT_BYTE(VAR REGISTER_OFFSET : INTEGER;
                       VAR REGISTER_VALUE : INTEGER;
                       [UNIT : INTEGER]) : INTEGER; EXTERNAL;
```

Returns:

Type: cond_value
access: write only
mechanism: by value

Longword condition value, returned in R0. See "Condition Values Returned" below for a list of possible values.

Arguments:

REGISTER_OFFSET
type: longword_unsigned
access: read only
mechanism: by reference

Offset from the address of the Local Command Register to the register which should be accessed.

If the value is less than 16, one of the local or remote node registers on the Bit 3 adaptor will be accessed. If the value is greater than 15, a byte I/O operation will occur on the remote bus.

Note that the REGISTER_OFFSET must be greater than 15 in order to skip the local and remote node adaptor registers.

See the adaptor hardware manual for more information.

REGISTER_VALUE

type: longword_unsigned
access: read only
mechanism: by reference

The value to write to the register. Only the lower 8-bits of the value are actually written to the register.

Optional Arguments:

UNIT
type: longword_unsigned
access: read only
mechanism: by value

Adaptor Unit used for output byte operations. This procedure assumes Unit 0 by default.

Description:

The application uses this procedure to write to the Bit 3 adaptor node I/O or remote bus I/O registers.

When the value of REGISTER_OFFSET is less than 16, the I/O reference is to a Bit 3 adaptor register. Otherwise, the remote adaptor causes an I/O write cycle on the remote bus.

To compute the physical I/O address of the access, determine the IO LO setting of the jumpers at location B9 of the Q22-bus adaptor board. Add 0x10 to this base address, plus the REGISTER_OFFSET value.

When a function call returns the value SS$_DEVINACT, the procedure was unable to access the register. The most likely causes are:

* There is no adaptor board in the system.
* The I/O space selected via the IO jumpers at location B9 does not agree with the I/O address range given BIT3_MAP.
* The IO LO and IO HI settings are the same at location B9. To access the remote bus I/O using this function, set IO HI greater than IO LO. See the adaptor hardware manual for more information.
* If the access was to the remote registers (REGISTER_OFFSET >= 8), the transmitter jumper in the SYST block (location C1) is not installed.

Condition Values Returned:

SS$_NORMAL	Successful execution.
SS$_OPINCOMPL	None of the _MAP_ procedures (BIT3_MAP_RR, BIT3_MAP_DP, or BIT3_MAP_LM) successfully completed. One of these procedures must complete before the BIT3_OUT_BYTE procedure can complete.
SS$_DEVINACT	The routine could not access the adaptor hardware.
SS$_BADPARAM	The value of REGISTER_OFFSET was greater than the size of the I/O section, or an invalid adaptor Unit was specified.

BIT3_OUT_WORD

Writes data to one of the local or remote node Bit 3 adaptor registers, or a remote bus I/O register. The registers are written as 16-bit quantities.

VMS PASCAL interface:

```
FUNCTION BIT3_OUT_WORD(VAR REGISTER_OFFSET : INTEGER;
                       VAR REGISTER_VALUE : INTEGER;
                       [UNIT : INTEGER]) : INTEGER; EXTERNAL;
```

Returns:

Type: cond_value
access: write only
mechanism: by value

Longword condition value, returned in R0. See "Condition Values Returned" below for a list of possible values.

Arguments:

REGISTER_OFFSET
type: longword_unsigned
access: read only
mechanism: by reference

Offset from the address of the Local Command Register to the register which should be accessed.

If the value is less than 16, one of the local or remote node registers on the Bit 3 adaptor set will be accessed. If the value is greater than 15, a word I/O will be done on the remote bus.

Note that the REGISTER_OFFSET must be greater than 15 in order to skip the local and remote node adaptor registers.

See the adaptor hardware manual for more information.

REGISTER_VALUE
type: longword_unsigned
access: read only
mechanism: by reference

The value written to the register. The upper 16-bits of the longword value are ignored.

Optional Arguments:

UNIT
type: longword_unsigned
access: read only
mechanism: by value

Adaptor Unit used for output word operations. This procedure assumes Unit 0 by default.

Description:

The application uses these procedures to write to the Bit 3 adaptor or remote bus I/O registers.

When the value of REGISTER_OFFSET is less than 16, the I/O reference is to a Bit 3 adaptor register. Otherwise, the remote adaptor causes an I/O read cycle on the remote bus.

To compute the physical I/O address of the access, determine the IO LO setting of the jumpers at location B9 of the Q22-bus adaptor board. Add 0x10 to this base address, plus the REGISTER_OFFSET value.

When a function call returns the value SS$_DEVINACT, the procedure was unable to access the register. The most likely causes are:

* There is no adaptor board in the system.
* The I/O space selected via the IO jumpers at location B9 does not agree with the I/O address range given BIT3_MAP.
* The IO LO and IO HI settings are the same at location B9. To access the remote bus I/O using this function, set IO HI greater than IO LO. See the adaptor hardware manual for more information.
* If the access was to the remote registers (REGISTER_OFFSET >= 8), the transmitter jumper in the SYST block (location C1) is not installed.

Condition Values Returned:

SS$_NORMAL	Successful execution.
SS$_OPINCOMPL	None of the _MAP_ procedures (BIT3_MAP_RR, BIT3_MAP_DP, or BIT3_MAP_LM) successfully completed. One of these procedures must complete before the BIT3_OUT_WORD procedure can complete.

SS$_DEVINACT	The routine could not access the adaptor hardware.
SS$_BADPARAM	The value of REGISTER_OFFSET was greater than the size of the I/O section, or an invalid adaptor Unit was specified.

Appendix C: Example Program Listings

Program listings for:

DumpPort.H	Include file for DumpPort.C
DumpPort.C	C language example, dumps beginning of dual-port memory

```
/*****************************************************************************
;*  DumpPort.h                                                               *
;*  Copyright (c) 1989, 1990 by                                              *
;*  Bit 3 Computer Corporation, Minneapolis, Minnesota.                      *
;*  All rights reserved.                                                     *
;*                                                                           *
;*  This software is furnished under a license and may be used and copied    *
;*  only in accordance with the terms of such license and with the           *
;*  inclusion of the above copyright notice.  This software or any other     *
;*  copies thereof may not be provided or otherwise made available to any    *
;*  other person.  No title to and ownership of the software is hereby       *
;*  transferred.                                                             *
;*                                                                           *
;*  The information in this software is subject to change without notice     *
;*  and should not be construed as a commitment by Bit 3 Computer            *
;*  Corporation.                                                             *
;*                                                                           *
;*****************************************************************************

    DumpPort.h
        Include file for DumpPort.C example program.

*/

#define MAXLINE         512
/* Maximum number of characters in a buffer */

#define INCDUMP         16
/* How much data to dump on one line */

#define MAXDUMP         (8 * INCDUMP)
/* How much data to dump.  Number of lines * data per line */


/*
                Definitions used for BIT3_IFLIB procedures
*/

#define BIT3_GK_CMD             0
#define BIT3_GK_CMD_EXT         1
#define BIT3_GK_STATUS          2

/*      Local Status register error bits        */

#define BIT3_M_NOPOWER  0x01
#define BIT3_M_PARITY   0x80
#define BIT3_M_BERR     0x40
#define BIT3_M_TIMEOUT  0x04

#define BIT3_GK_ERROR   \
        (BIT3_M_NOPOWER|BIT3_M_PARITY|BIT3_M_BERR|BIT3_M_TIMEOUT)


/*      Function prototypes for BIT3_IFLIB */

int     BIT3_MAP_DP(void ** p0_addr, ...);

int     BIT3_WRITE_REGISTER(int * offset, unsigned * value, int unit);
int     BIT3_READ_REGISTER(int * offset, unsigned * value, int unit);
```

```
/*****************************************************************************
;*  DumpPort.c                                                               *
;*  Copyright (c) 1989, 1990 by                                              *
;*  Bit 3 Computer Corporation, Minneapolis, Minnesota.                      *
;*  All rights reserved.                                                     *
;*                                                                           *
;*  This software is furnished under a license and may be used and copied    *
;*  only in accordance with the terms of such license and with the           *
;*  inclusion of the above copyright notice.  This software or any other     *
;*  copies thereof may not be provided or otherwise made available to any    *
;*  other person.  No title to and ownership of the software is hereby       *
;*  transferred.                                                             *
;*                                                                           *
;*  The information in this software is subject to change without notice     *
;*  and should not be construed as a commitment by Bit 3 Computer            *
;*  Corporation.                                                             *
;*                                                                           *
;*****************************************************************************

    DumpPort.C
        A simple example program written in VAX C.
        Demonstrates the use of the BIT3_IFLIB in order to dump the first
        few characters contained in the Dual-Port memory window.
*/

#include        <stdio.h>
#include        <stdlib.h>
#include        <ctype.h>
#include        <string.h>

#include        <ssdef.h>
#include        <descrip.h>

#include        "DumpPort.h"

/*****************************************************************************

        Global Variables

*****************************************************************************/

    int AdUnit;         /* Adaptor Unit for Dump Port exercise */

/*****************************************************************************

        Function prototypes

*****************************************************************************/

void    PrintError(int ErrorCode);      /* Prints VMS error message */

char *  GetStatusMsg(int status);       /* Returns string error message */
int     Setup(int *status);             /* Sets up the Q22bus Adaptor */
int     ErrorStatus(int *status);       /* Reads the local node status */
int     ClearStatus(void);              /* Clears the local node status */

/*****************************************************************************

        Main program

*****************************************************************************/

main()
{
    unsigned char       *DualPortLow,*DualPortHigh;
                        /* Start and end address of Dual-Port memory */

    int ErrorCode;      /* VMS error code */

    int Status;         /* Status register error bits */

    unsigned char       *DumpAddr;      /* Temporary pointers into Dual-Port */
    unsigned char       *Ptr;
```

```
/*      Map the Dual-Port RAM window */

    printf("On which Adaptor do you want to dump Dual Port RAM (0-3)? ");
    scanf("%d", &AdUnit);
    ErrorCode = BIT3_MAP_DP(&DualPortLow,&DualPortHigh,AdUnit);

    if (ErrorCode != SS$_NORMAL) {
        if (ErrorCode == SS$_NOSUCHSEC)
            printf("Please use BIT3_MAP to create a Dual-Port window.");
        exit(ErrorCode);
    }


/*      Set up the link, abort if status error */

    if ( (ErrorCode = Setup(&Status)) != SS$_NORMAL) {
        exit(ErrorCode);
    } else if ( Status != 0) {
        printf("\n%s\n",GetStatusMsg(Status));
        exit(SS$_ABORT);
    }


/*      Dump the Dual-Port section as ASCII characters,
        printing a period in place of non-printable or
        control characters.                                 */

    putchar('\n');
    for (DumpAddr = DualPortLow;
            ((DumpAddr <= DualPortHigh) && (DumpAddr < DualPortLow + MAXDUMP));
            DumpAddr += INCDUMP) {

        for (Ptr = DumpAddr; ( (Ptr < DumpAddr + INCDUMP) &&
                (Ptr < DualPortHigh) ); Ptr++)
            printf("%02x ",*Ptr);

        putchar('\t');

        for (Ptr = DumpAddr; ( (Ptr < DumpAddr + INCDUMP) &&
                (Ptr < DualPortHigh) ); Ptr++) {
            if (isprint(*Ptr) && (!iscntrl(*Ptr)) )
                putchar(*Ptr);
            else putchar('.');
        }
        putchar('\n');
    }


/*      Check again for Status errors */

    if ( (ErrorCode = ErrorStatus(&Status)) != SS$_NORMAL) {
        exit(ErrorCode);
    } else if ( Status != 0) {
        printf("\n%s\n",GetStatusMsg(Status));
        exit(SS$_ABORT);
    }

    printf("\n\nAll done.\n");
    exit(SS$_NORMAL);
}


/*****************************************************************************

    Setup
        Initializes the Adaptor registers.  Checks the local status register
        for errors.

*****************************************************************************/

int Setup(Sts)
int *Sts:               /* Returned: Local Status Register error bits */
{
    int Error:          /* Returned: VMS Error Code */
    int Value;          /* Temporary: Data register read/write area */
```

```
    Error = BIT3_WRITE_REGISTER(&BIT3_GK_CMD_EXT,&2,AdUnit);  /* AWAKEN Adaptor */
    if (Error != SS$_NORMAL) return(Error);

    Error = BIT3_READ_REGISTER(&BIT3_GK_STATUS,&Value,AdUnit); /* Flush status register */
    if (Error != SS$_NORMAL) return(Error);


    Error = BIT3_WRITE_REGISTER(&BIT3_GK_CMD,&0,AdUnit);  /* Clear local command register */
    if (Error != SS$_NORMAL) return(Error);


    Error = ClearStatus();              /* Clear status errors */
    if (Error != SS$_NORMAL) return(Error);

    Error = ErrorStatus(&Value);        /* Get status */
    if (Error == SS$_NORMAL)
        *Sts = Value;

    return(Error);
}


/*****************************************************************************

    ClearStatus
        Clears the error status bits in the Local Node Status Register.

*****************************************************************************/

int ClearStatus()
{
    int value;
    int Error;

    Error = BIT3_READ_REGISTER(&BIT3_GK_CMD,&value,AdUnit);
    if (Error != SS$_NORMAL) return(Error);

    value |= 0x80;

    return(BIT3_WRITE_REGISTER(&BIT3_GK_CMD,&value,AdUnit));
}


/*****************************************************************************

    ErrorStatus
        Reads the Local Node Status Register, returns only the error bits
        of that register.

*****************************************************************************/

int ErrorStatus(Sts)
int *Sts;               /* Returned: Local Status Register error bits */
{
    int Error;          /* Returned: VMS error status code */


    Error = BIT3_READ_REGISTER(&BIT3_GK_STATUS,Sts,AdUnit);
    *Sts &= BIT3_GK_ERROR;

    return(Error);
}


/*****************************************************************************

    GetStatusMsg
        Creates a character string containing a message appropriate to
        the status bits set in the procedures parameter.

*****************************************************************************/

char * GetStatusMsg(Status)
int     Status;         /* Local Status Register error bits */
{
    static char Buffer[MAXLINE];        /* Message buffer, local to procedure */
    char        *message = &Buffer[0];  /* Easier to work with a pointer */
```

```
    if ((Status & BIT3_GK_ERROR) == 0)
        strcpy(message,"No Adaptor errors.");
    else {
        sprintf(message,"Adaptor status error 0x%02x:",Status);
        if (Status & BIT3_M_NOPOWER)
            strcat(message,"\n\tCable disconnected or remote power off.");
        else {
            if (Status & BIT3_M_PARITY)
                strcat(message,"\n\tInterface parity error.");
            if (Status & BIT3_M_BERR)
                strcat(message,"\n\tRemote bus error.");
            if (Status & BIT3_M_TIMEOUT)
                strcat(message,"\n\tInterface timeout.");
        }
    }

    return(message);
}
```

Appendix D: Fortran Programming Example

The following VMS Fortran program demonstrates how to access (write and read) dual-port memory. This program could easily be changed to use Remote Bus memory instead.

The main program maps to dual-port memory using the library procedure BIT3_MAP_DP. This program uses adaptor Unit 2. Appendix B describes the detailed interface for all the Bit 3 library procedures.

If the mapping operation was successful, the program computes the size of memory (in bytes) mapped. It then calls the MODIFYDP subroutine, passing the base address and size of the dual-port memory mapped.

The subroutine uses this input pointer as the base address of an adjustable size local array. Writing and reading to and from the local array results in a write/read operation to the dual-port memory.

Although this program uses dual-port memory, it could have easily used Remote Bus memory instead.

```
      PROGRAM TESTDP
      EXTERNAL BIT3_MAP_DP

      INTEGER*4 IPTR_BGN, IPTR_END, ISTAT, ISIZE, IUNIT

      IUNIT = 2

      ISTAT = BIT3_MAP_DP(IPTR_BGN, IPTR_END, %VAL(IUNIT) )

      ISIZE = (IPTR_END - IPTR_BGN) * 4

      IF (ISTAT .NE. SS$_NORMAL) THEN
         CALL EXIT(ISTAT)
      END IF

      CALL MODIFYDP ( %VAL(IPTR_BGN), ISIZE)
      END


      SUBROUTINE MODIFYDP (ARRAY,BSIZE)

      BYTE ARRAY (BSIZE)
      INTEGER * 2 I

      DO 1000 I = 1, BSIZE
         ARRAY(I) = I
1000  CONTINUE
      RETURN
      END
```

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP