MicroVAX/DRQ3B Device Driver User's Guide

Order Number: AA–KM26C–TN

July 1990

This document describes the MicroVAX/DRQ3B software driver for control of the DRQ3B parallel input/output module under the MicroVMS and VMS operating systems.

First Printing, October 1987
Second Printing, September 1988
Third Printing, July 1990

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Restricted rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Copyright ©Digital Equipment Corporation 1987, 1988, 1990

All rights reserved.
Printed in U.S.A.

The Reader's Comments form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation: DEC, MicroVAX, MicroVMS, PDP-11, Q-bus, VAX, VMS, and the DIGITAL logo.

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA*	**CANADA**	**INTERNATIONAL**
Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario K2K 2A6 Attn: Direct Order Desk	Digital Equipment Corporation PSG Business Manager c/o Digital's local subsidiary or approved distributor

In Continental USA, Alaska, and Hawaii call 800–DIGITAL.

In Canada call 800-267-6215.

*Any order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Supply Business (SSB), Digital Equipment Corporation, Westminster, Massachusetts 01473.

This document was prepared using VAX DOCUMENT, Version 1.2

Contents

PREFACE ix

CHAPTER 1 INTRODUCTION 1–1

1.1 THE DRQ3B PARALLEL DMA I/O MODULE 1–1

1.2 THE MICROVAX/DRQ3B DEVICE DRIVER 1–2

CHAPTER 2 INSTALLATION 2–1

2.1 INSTALLATION PREREQUISITES 2–1

2.2 INSTALLATION TIME 2–1

2.3 INSTALLATION PROCEDURE 2–2

2.4 INSTALLATION VERIFICATION PROCEDURE — IVP 2–6

2.5 POST-INSTALLATION TASKS — MICROVMS SYSTEMS ONLY 2–7

2.6 SAMPLE INSTALLATION DIALOGUE — MICROVMS V4.7 2–8

2.7 SAMPLE INSTALLATION DIALOGUE — VMS V5.0 2–10

CHAPTER 3 OPERATING FEATURES 3–1

3.1 BLOCK OR NONBLOCK DMA TRANSFERS 3–1

3.1.1 Extended Block Mode Option 3–2

3.2 DOUBLE BUFFERING 3–3

Contents

3.3 DMA I/O TRANSFER MODES — LOGICAL AND PHYSICAL 3–3
3.3.1 Logical I/O Operation 3–4
3.3.2 Physical I/O Operation 3–4
3.3.3 Advantages of Physical I/O 3–6

3.4 USING LOGICAL I/O 3–6
3.4.1 Logical I/O Summary 3–6

3.5 USING PHYSICAL I/O 3–7
3.5.1 Physical I/O Read and Write Logical Blocks 3–7
3.5.2 Physical I/O to or from Local VAX Memory 3–7
3.5.2.1 Page Locking • 3–8
3.5.2.2 Allocating Mapping Registers • 3–8
3.5.2.3 Q-bus Address Descriptor • 3–8
3.5.2.4 User Mapping Register Descriptor • 3–9
3.5.2.5 Summary of Physical I/O to Local VAX Memory • 3–9
3.5.3 Physical I/O to or from Q-bus Memory 3–10
3.5.3.1 Summary of Physical I/O to Q-bus Memory • 3–10

3.6 SELECTING I/O MODE 3–11

3.7 INTERRUPTS AND ATTENTION AST'S 3–11
3.7.1 End-of-Process 3–12
3.7.2 Channel 0 EOP 3–12
3.7.3 Nonexistent Memory EOP 3–13
3.7.4 External Interrupt 3–13
3.7.5 FIFO Underflow 3–13
3.7.6 FIFO Overflow 3–14

CHAPTER 4 FUNCTION CODES AND I/O STATUS BLOCKS 4–1

4.1 DEFINITION FILES 4–1

4.2 FUNCTION CODES, ARGUMENTS, AND MODIFIERS 4–2
4.2.1 Function Codes 4–2
4.2.2 Function Code Arguments 4–3
4.2.2.1 Logical I/O Function Code Arguments • 4–3
4.2.2.2 Physical I/O Function Code Arguments • 4–4

4.2.3 **Function Modifiers** 4–4
4.2.3.1 IO$M_HX_NOBLOCK_MODE and IO$M_HX_NODOUBLE_BUFF • 4–6
4.2.3.2 IO$M_HX_FIFO_CLEAR • 4–6
4.2.3.3 IO$M_HX_NOSTART_DMA • 4–6
4.2.3.4 IO$M_HX_RUN_DOWN • 4–6
4.2.3.5 IO$M_HX_FUNCT_BITS and IO$M_HX_ATTN_ENABLE • 4–7

4.3 SET MODE AND SENSE MODE 4–7
4.3.1 Controlling the Function Bits 4–7
4.3.1.1 Writing to the Output Function Bits • 4–8
4.3.1.2 Reading the Input Function Bits • 4–8
4.3.2 Enabling and Disabling Attention AST's 4–8
4.3.2.1 Interrupt Setting Summary • 4–11

4.4 I/O STATUS BLOCKS 4–11
4.4.1 DMA Initialization I/O Status Block 4–12
4.4.2 Read/Write I/O Status Block 4–13
4.4.3 Set Mode/Sense Mode I/O Status Block 4–14

4.5 DRQ3B STATUS REGISTERS 4–17
4.5.1 Status Register 4–18
4.5.2 DMA Status Register 4–19

4.6 BUFFER COMPLETION 4–22
4.6.1 AST Routines 4–22
4.6.2 Event Flags 4–22
4.6.3 SYS$QIOW 4–23

4.7 USING BOTH PORTS SIMULTANEOUSLY 4–23

4.8 AST AND DIO LIMITS 4–24

4.9 TIMEOUTS 4–24

4.10 CANCELING I/O 4–24

Contents

CHAPTER 5 DEVICE INFORMATION 5–1

APPENDIX A EXAMPLE PROGRAM A–1

A.1 EXAMPLE PROGRAM A–1

APPENDIX B DRQ3B CONNECTORS B–1

B.1 DRQ3B CONNECTORS B–1

INDEX

FIGURES

4–1 Attention Mask Format 4–10
4–2 DMA Initialization I/O Status Block 4–12
4–3 Read/Write I/O Status Block 4–13
4–4 SET MODE/SENSE MODE IOSB 4–15
4–5 High Word of the First Longword in SETMODE IOSB 4–15
4–6 Status Register 4–18
4–7 DMA Status Register 4–20
B–1 Input Pins on J2 of the DRQ3B Connector Kits B–1
B–2 Output Pins on J1 of the DRQ3B Connector Kits B–2

TABLES

4–1 Definition Files 4–1
4–2 Function Codes for Logical I/O 4–2
4–3 Function Codes for Physical I/O 4–2
4–4 Function Code Arguments for Logical I/O 4–3
4–5 Function Code Arguments for Physical I/O 4–4
4–6 Function Modifiers for Logical and Physical I/O 4–5
4–7 Correspondence Between P1 Bits and Function Bits 4–8
4–8 Arguments for the IO$M_HX_ATTN_ENABLE Function Modifier 4–10
4–9 Attention AST Bit Masks 4–10
4–10 Attention AST Constants 4–11
4–11 Status Codes Returned by IO$_HX_DMA_INIT 4–12
4–12 Status Codes Returned by Read/Write Logical or Physical Block 4–13

4–13	Function Bit Status in SETMODE IOSB	4–15
4–14	Status Code Returned by IO$M_HX_FUNCT_BITS	4–16
4–15	Status Codes Returned by IO$M_HX_ATTN_ENABLE	4–16
4–16	Status Register	4–18
4–17	DMA Status Register	4–20
5–1	Device Information	5–1

Preface

Manual Objectives

This manual describes the MicroVAX/DRQ3B Device Driver. This software controls the DMA parallel input/output module in a MicroVMS or VMS operating system environment. The driver is accessed through the VMS system service routine, SYS$QIO, but this routine and system service routines in general are not described. The *VAX/VMS System Service Manual* covers this topic.

Intended Audience

This manual is intended for all programmers writing code for the DRQ3B module. It describes the driver's features and requirements. This manual assumes that the reader is familiar with programming terms and understands DIGITAL addressing conventions and VAX architecture. In particular, the manual assumes that the user is already a competent high level language or MACRO programmer. The manual assumes that the user has some familiarity with the VMS I/O structure, particularly the use of the SYS$QIO system service routine. Some explanation is given of concepts such as AST routines and event flags, but a detailed description of these VMS features is not covered.

Document Structure

The manual contains five chapters and two appendixes. The following material is discussed:

Chapter	Title	Contents
Chapter 1	Introduction	A brief overview of the driver and the module.
Chapter 2	Installation	Templates of successful installations.
Chapter 3	Operating Features	A detailed conceptual description of the features available under the driver.
Chapter 4	Function Codes and I/O Status Blocks	Descriptions of all the available function codes, code modifiers, arguments and descriptions of the type of information returned in the I/O status blocks.
Chapter 5	Device Information	Description of the use of the SYS$GETDVI system service routine.
Appendix A	Example Program	An example program showing the use of both ports.
Appendix B	Connectors	Diagrams of the signals carried on each pin in the DRQ3B connectors.

Associated Documents

The following documents may be of interest to DRQ3B users.

- *DRQ3B Parallel DMA I/O Module User's Guide*
- *Laboratory Interfacing Handbook*
- *VAX Realtime User's Guide*
- *VAX/VMS System Services Reference Manual*
- *VAX/VMS I/O User's Reference Manual*

1 Introduction

1.1 The DRQ3B Parallel DMA I/O Module

The DRQ3B parallel direct memory access (DMA) input/output module allows real-time collection of parallel digital data at transfer rates of up to 1.3 megahertz of 16-bit words.[1] The module provides two ports for connection to external devices, an input port and an output port.

The DRQ3B incorporates the following features:

- DMA data transfer using logical or physical I/O
- One 16-bit digital data input port
- One 16-bit digital data output port
- Data transfer rates up to 1.3 megahertz (words)
- One 512-word first-in/first-out (FIFO) buffer on each port for increased data throughput
- Six input and six output general purpose function bits
- Selectable burst or block mode DMA

The DRQ3B is designed to provide maximum data transfer rates with a minimum of system bus interaction. This is accomplished through the use of DMA channels associated with the two buffered data ports. Each data port is associated with one of the DMA channels.

The input data port is called port 0 and is associated with DMA channel 0. The output data port is called port 1 and is associated with DMA channel 1. Each data port includes a 512-word first-in/first-out (FIFO) buffer to prevent slower devices from impeding transfers to or from system memory. The data ports use a two-wire handshaking scheme to synchronize data transfers with external devices.

The DMA channels are also designed to provide a minimum of system bus interaction. Direct memory access (DMA) refers to the DRQ3B's capability to access memory locations automatically. The DRQ3B driver loads a starting address into the DRQ3B's address register. When the DRQ3B starts data transfers, it can then access that address automatically and asynchronously from the executing program. The DRQ3B increments its address counters automatically to access successive address locations. Direct memory access operations greatly increase data transfer rates and allow the DRQ3B to transfer data asynchronously from your executing program.

[1] Rates of up to 1.3 megahertz of 16 bit words are obtainable when the DRQ3B is operated in extended block mode with a 1 microsecond holdoff time and a minimum of other system activity.

1.2 The MicroVAX/DRQ3B Device Driver

The MicroVAX/DRQ3B device driver provides VMS support for the DRQ3B. The driver supports the input and output ports on the DRQ3B as separate devices.

The device name for the DRQ3B is ''HX.'' Each DRQ3B installed in a system is assigned a controller name when the device driver is installed into a VMS system. This controller name follows the format HXa, where ''a'' is an alphabetic character designating each separate DRQ3B in a system. The first DRQ3B installed is assigned a controller name of ''A'' giving it a device name of HXA. In addition, since each DRQ3B has an input port and output port, and each of these ports is considered a separate device, the input port is named ''0'' and the output port is named ''1''. Under this scheme, the input port on the first DRQ3B is named HXA0; the output port on the first DRQ3B is named HXA1. On the second DRQ3B installed, the input port is named HXB0, and the output port is named HXB1.

Both ports on a single DRQ3B can be used simultaneously. However, this requires the use of asynchronous system traps (AST's) or event flags. These techniques require some knowledge of VMS I/O architecture.

The MicroVAX/DRQ3B device driver is used by calling system service routines from your application program. Briefly, the following steps are taken to invoke the driver:

- The port to be used must be assigned a ''channel'' using the SYS$ASSIGN system service call. The channel number assigned to this port is later used as an argument to the SYS$QIO system service call.
- The SYS$QIOW system service routine is called using the IO$_HX_DMA_INIT function code which initializes the DRQ3B for DMA data transfers.
- Data is queued to be transferred by calling the SYS$QIO system service routine using the appropriate function code. For example, on the input port (HXA0) the IO$_READLBLK function code is specified. The data transfer function codes take arguments which specify the address and size of the data buffers.
- The driver begins transferring the specified buffer.
- Subsequent SYS$QIO calls can be made to queue additional buffers.
- After all data buffers are transferred, the DRQ3B port is disassociated from the channel by issuing the SYS$DASSGN system service call.

The above sequence illustrates the general outline of steps taken for logical I/O transfers. Additional steps must be taken if the physical I/O mode is desired. These are described in Chapter 3.

Some of the additional features of the driver allow you to queue buffers without starting DMA transfers or select that double buffering not be used. These additional features are described in Chapter 4.

2 Installation

This chapter describes how to install the MicroVAX/DRQ3B Device Driver from a tape cartridge or diskette. Instructions for running the Installation Verification Procedure (IVP) are also provided. A sample dialogue of the installation procedure is included at the end of this chapter.

2.1 Installation Prerequisites

Before installing the MicroVAX/DRQ3B Device Driver, check the contents of your kit against the bill of materials included in your shipment. If any items are missing, do not proceed with the installation.

Check that you have all of the following prerequisites before beginning the installation procedure.

- A supported MicroVAX II, MicroVAX 3000 series, or VAXstation II computer system. Refer to the MicroVAX/DRQ3B Device Driver SPD/SSA for a list of supported computer systems.
- The MicroVMS V4.7 operating system, OR the VMS V5.0 or later operating system
- A DRQ3B installed in the card cage
- 700 free disk blocks
- The MicroVAX/DRQ3B Device Driver on tape cartridge or diskette

Before installing the DRQ3B driver, the DRQ3B module should be installed in the machine. You must know the address of the DRQ3B as represented on its address switches. The installation procedure prompts you for this address.

On VMS V5.0 or later systems, you must install the license registration product authorization key (PAK) before installing the software. From the SYSTEM account, type the following at the "$" prompt:

```
$  @SYS$UPDATE:VMSLICENSE
```

This command file prompts you for all the information needed to install the PAK. Refer to the paper PAK sheets shipped with the software for all of the necessary information. Full information on the license registration procedure is found in the *VMS License Management Utility Reference Manual.*

2.2 Installation Time

The entire installation procedure takes about 15 minutes.

2.3 Installation Procedure

The DRQ3B device driver is installed using the VMSINSTAL command. This command can only be issued from the system manager's account.

If you are running MicroVMS Version 4.7, logging into a system manager's account may invoke the Manager Menu. This installation procedure instructs you to first exit the menu. If your system account LOGIN.COM file does not invoke the Manager Menu, or if you are running VMS V5.0 or later, the Manager Menu is not invoked and you should proceed from step 3 below.

To install the MicroVAX/DRQ3B Device Driver on a system running MicroVMS Version 4.7, using option 1 on the Manager Menu, take the following steps:

Step 1. Invoke VMSINSTAL.

1 Log in to a SYSTEM account.

The system displays a welcome message and the Manager Menu:

```
       Welcome to MicroVMS V4.7

   Last interactive login on Xday, dd-mmm-yyyy hh:mm

Main Menu

        1 - Exit to DCL
        2 - Log out of the SYSTEM account
        3 - Invoke the MAIL utility
        4 - Invoke the PHONE utility
        5 - Add a user account to the system
        6 - Install optional software
        .
        .
        .
       14 - SHUT DOWN the system

Enter a number (? or ?# for HELP):  1

If you wish to use the menus again, type the following:
        $ @SYS$MANAGER:MGRMENU
```

2 Select option 1. The system displays the DCL prompt.

3 Set your default directory to SYS$UPDATE:

```
$  SET DEFAULT SYS$UPDATE
```

4 Type the following command line to run the VMSINSTAL utility:

```
$  @VMSINSTAL hx013 device_name OPTIONS N
```

If you are installing the device driver from a tape cartridge, specify the device_name in the preceding command line as **MUA0:**. If you are installing the device driver from a diskette, specify the device_name in the preceding command line as **$FLOPPY1:**.

Specifying the OPTIONS N parameter causes VMSINSTAL to prompt you about the *Release Notes*. Print the *Release Notes* and read them before proceeding further with the installation. The release notes may contain last-minute instructions or changes to the installation procedures described in this guide.

The system displays the following informational messages:

```
  VAX/VMS Software Product Installation Procedure 4.7

It is dd-mmm-yyyy at hh:mm.
Enter a question mark (?) at any time for help.
```

The VMSINSTAL utility proceeds by displaying the following prompts:

```
* Are you satisfied with the backup of your system disk [YES]?
```

If you are satisfied with the backup of your system disk, press RETURN after this prompt. If you do not have a current backup of your system disk, answer NO to terminate this installation procedure. Back up your system disk and then begin the installation again. If your SYSTEM disk fails for any reason, you will need a copy of your previous system disk.

For information about the backup procedure, see the *MicroVMS V4.7 Release Notes*, and *Installing MicroVMS on MicroVAX II From a Tape Cartridge*, *Installing MicroVMS on MicroVAX I/II From Diskettes*.

Step 2. Load the distribution medium into the drive.

If you are installing the MicroVAX/DRQ3B Device Driver from a tape cartridge, VMSINSTAL displays the following instruction and prompt:

```
Please mount the first volume of the set on MUA0:.
* Are you ready?
```

If you are installing the MicroVAX/DRQ3B Device Driver from a diskette, VMSINSTAL displays the following instruction and prompt:

```
Please mount the first volume of the set on $FLOPPY1:.
* Are you ready?
```

Load the distribution media in the appropriate place.

Take the following steps to load or unload a tape cartridge:

1. If the cartridge-release handle of the tape drive is not already in the down or right position (depending on whether the drive is in the horizontal or vertical position), put it in this position.
2. Check that the red light (the LOAD/UNLOAD switch) on the right of (or above) the tape drive is in the out position and not glowing.
3. Check that the small green (activity) light on the left of (or below) the tape drive is glowing steadily.
4. Lift the cartridge-release handle.
5. Insert the tape cartridge, with the arrow on the cartridge facing toward the drive (away from the DIGITAL logo).
6. Put the cartridge-release handle down (or to the right) on the drive, and make sure that the handle is locked solidly in place.
7. Push the red light button in.

8 Check that you have successfully loaded the tape cartridge by seeing that the red light button is in and glowing steadily, and the green light is also glowing steadily.

Take the following steps to load a diskette:

- Open the diskette door by pressing on its outer edge.
- Align the orange arrow on the diskette with the orange stripe on the drive and insert the diskette. The write-protect notch is down in drive 1 and up in drive 2.
- After inserting the diskette, press the drive door to close it.

Enter YES after the ''Are you ready?'' prompt when you have loaded your distribution medium.

When your distribution medium has been successfully loaded into the drive, VMSINSTAL displays an informational message telling you that the medium has been mounted and that installation has begun.

```
%MOUNT-I-MOUNTED, HX mounted on _MUA0:
The following products will be processed:
  HX V1.3

        Beginning installation of HX V1.3 at 11:38
%VMSINSTAL-I-RESTORE, Restoring product saveset A...
```

Step 3. Select the release notes option.

VMSINSTAL now prompts you to select a release notes option:

```
Release Notes Options:

        1. Display release notes
        2. Print release notes
        3. Both 1 and 2

* Select option [3]:
```

You can display the release notes on the screen, print them, or do both. Then, you can continue the installation procedure by entering YES after the following prompt:

```
Do you want to continue the installation [NO]? yes
```

The installation then proceeds with the following message.

```
%VMSINSTAL-I-RELMOVED, The products release notes have been successfully moved to SYS$HELP.

$!  Copyright (c) 1990 Digital Equipment Corporation.  All rights reserved.

%HX-I-VMSOK, Installing HXDRIVER V1.3-008 on VMS Version V4.7
```

Step 4. Specify DRQ3B Base Address and Vector Address

VMSINSTAL prompts you to select if files replaced by this installation are to be purged. Answer YES to this question.

```
* Do you want to purge files replaced by this installation [YES]?:
```

VMSINSTAL now prompts you for the base address and the vector address of each DRQ3B that you have in your system. The DRQ3B is a floating device and may or may not be configured at the factory configurations.

The following message is printed by VMSINSTAL on MicroVMS V4.7 systems.

```
The DRQ3B device (HXDRIVER) is not currently supported under
the VMS AUTOCONFIGURATION utility.  In order to run the
installation procedure, you must answer questions
concerning the hardware installation. This procedure will
then create the SYS$MANAGER:HX$LOAD.COM procedure to load
the driver for the installation verification procedure.

If you do not know these settings at this time, then use
the default settings and do not request execution of the
IVP.  You may then edit SYS$MANAGER:HX$LOAD.COM at your
convenience to set the correct hardware parameters and to
add additional DRQ3B interface units.
```

If you do not currently know the address setting(s) of the DRQ3B(s) in your system, use the default address settings in the installation prompt. This will accomplish the task of loading the driver, but to use the driver, the correct addresses must be be entered into the HX$LOAD.COM file.

The VMSINSTAL procedure then prompts you for the addresses of the DRQ3B devices. Enter the base address and vector address of the first DRQ3B in your system.

```
* Enter the first DRQ3B Q-Bus address (octal) [17760740]: 17760740
* Enter the first DRQ3B Q-bus vector (octal) [300]: 300
```

The VMSINSTAL procedure then asks if you have additional DRQ3B's.

```
* Do you have additional DRQ3B's [NO]? no
```

If you have additional DRQ3B's, answer YES. The VMSINSTAL program then prompts you for the base address and vector address of the second DRQ3B. This procedure continues until you answer NO to the prompt for additional DRQ3B's.

On VMS V5.0 or later systems the VMSINSTAL procedure then displays product registration information asks if you have registered and loaded the product authorization key (PAK) which you received with your kit.

```
        Product:      DRQ3B-DRIVER
        Producer:     DEC
        Version:      1.3
        Release Date: 30-MAY-1990

* Does this product have an authorization key registered and loaded? YES
```

If you have registered and loaded the product authorization key for the MicroVAX/DRQ3B Device Driver answer YES. The procedure will then continue.

If you have not registered and loaded the product authorization key answer NO. VMSINSTAL will then print out the following informational messages and you will not be allowed to run the IVP. See Section 2.1 for information on how to install the PAK.

```
An authorization key for DRQ3B HXDRIVER has not been registered and loaded.

    The DRQ3B IVP will NOT be run as part of this installation.
    After the DRQ3B license has been registered and loaded, it
    is recommended that the IVP be run with the following command:

      $ @SYS$TEST:HX$IVP
```

VMSINSTAL then prints the following informational message.

```
    The installation will check for the VMS driver already installed
    in the system.  If there is no driver installed in the system you
    will receive the following message:

    "%SYSTEM-W-NOSUCHDEV, no such device available"

    This message can be ignored, the installation will proceed normally.
    Do not run the IVP if you must edit HX$LOAD.COM to insert the
    correct CSR and vector for your DRQ3B.

%SYSTEM-W-NOSUCHDEV, no such device available
```

In this example, there is no preexisting version of the MicroVAX/DRQ3B Device Driver, and the ‘‘no such device available’’ message is printed.

Step 5. Run the Installation Verification Procedure

The final question asked in the installation procedure determines whether the Installation Verification Procedure (IVP) is executed. This question will not be asked on VMS V5.0 or later systems if you have not registered and loaded the Product Registration Key (PAK).

```
* Do you want to run the IVP after the installation [YES]? yes
```

You should run the IVP to verify that the installation of the device driver was successful. Do not run the IVP only if you are unsure of the base address and vector address of the DRQ3B installed in your system. Failure of the IVP when you do not know the address of your DRQ3B does not indicate that the driver installation failed.

At this point, no further action is required of you. Informational messages are printed as the installation proceeds. Section 2.6 and Section 2.7 provide complete examples of installation procedures on MicroVMS V4.7 and VMS V5.0 systems with the informational messages printed.

2.4 Installation Verification Procedure — IVP

The installation verification procedure checks to see that the MicroVAX/DRQ3B Device Driver was successfully installed. To manually invoke the installation verification procedure execute the following command:

```
$  @SYS$TEST:HX$IVP
```

A successful execution of the IVP is shown below.

```
Beginning the HX V1.3-008 Installation Verification Procedure

HXA configuration:
   BIRQ level: 4
   Block Mode: extended
   Loopback cable: installed

Verification of MicroVAX/DRQ3B Device Driver HX V1.3-008 was successful

Completion of the HX V1.3-008 Installation Verification Procedure
```

The IVP reads the installed DRQ3B’s registers to obtain the above information. If you do not have a loopback cable installed from port 0 to port 1, the loopback cable is reported as not installed.

If the IVP fails, the successful message is not printed and a failure message and error report is generated. The following guidelines can help to isolate the problem.

- If the IVP does not report information about the DRQ3B as shown above (BR level, etc.) and reports a ''no such device'' (NOSUCHDEV) error message, the base address and/or vector address specified for the DRQ3B are probably incorrect. The driver was probably installed correctly, but since it cannot find a device at the address you specify, the IVP is not successful. Try rerunning the IVP after editing the HX$LOAD.COM file to specify the correct base address.
- If you are certain that the base address was specified correctly, but the IVP still fails to report information about the DRQ3B, the DRQ3B may have failed or not be properly seated in the card cage. Run the MDM diagnostics to test that the DRQ3B is properly seated in the card cage and is functioning.
- If the IVP does report information about the DRQ3B as shown above, but still reports an error message, the driver installation is unsuccessful. Try reinstalling the driver by starting over at the beginning. If the IVP fails a second time, call DIGITAL at 1-800-TEAM-CSS to report your problem.

The IVP only tests for the first DRQ3B you specified. If you want to run the IVP on second (or subsequent) DRQ3B's installed in your system, use the following command lines.

```
$ hx_ivp :== $sys$sysroot:[systest.hx]hx$ivp
$ hx_ivp hxb
```

The first command sets a symbol to be used to run the IVP. The second command line takes as an argument the device name for the subsequent DRQ3B's installed in your system. The third DRQ3B, for example, is named HXC.

2.5 Post-Installation Tasks — MicroVMS Systems Only

For VMS V5.0 or later systems, the DRQ3B is supported by AUTOCONFIGURE and there are no required post-installation tasks.

The following post-installation tasks apply to installations on MicroVMS V4.7 only.

If you did not specify the correct base addresses and vector addresses for your DRQ3B's during the installation process, you need to edit the HX$LOAD.COM file to enter the correct addresses. This must be done before using the driver. The HX$LOAD.COM file is located in SYS$MANAGER. If you specified the correct base addresses and vector addresses for your DRQ3B's during the installation process, the HX$LOAD.COM does not have to be altered.

If you want the MicroVAX/DRQ3B Device Driver to be initialized at every system startup, edit the SYS$MANAGER:SYCONFIG.COM file and add the following line to it:

```
$ @SYS$MANAGER:HX$LOAD
```

If you do not add this line to the SYCONFIG.COM file, you must use the above command line before you want to use a program that accesses the driver.

2.6 Sample Installation Dialogue — MicroVMS V4.7

The following is a sample installation dialogue for installation of the MicroVAX/DRQ3B Device Driver under MicroVMS 4.7.

```
Username: SYSTEM
Password:

    Last interactive login on Friday, 22-JUL-1988 11:36
    Last non-interactive login on Friday, 22-JUL-1988 11:37

$ show default
  SYS$SYSROOT:[SYSMGR]
$ set default sys$update
$ @vmsinstal hx013 mua0:

        VAX/VMS Software Product Installation Procedure V4.7


It is 22-JUL-1988 at 11:38.
Enter a question mark (?) at any time for help.
* Are you satisfied with the backup of your system disk [YES]? yes

Please mount the first volume of the set on MUA0:.
* Are you ready? yes
%MOUNT-I-MOUNTED, HX mounted on _MUA0:

The following products will be processed:

  HX V1.3

        Beginning installation of HX V1.3 at 11:38

%VMSINSTAL-I-RESTORE, Restoring product saveset A...
%VMSINSTAL-I-RELMOVED, The products release notes have been successfully moved to SYS$HELP.

$!  Copyright (c) 1990 Digital Equipment Corporation.  All rights reserved.

%HX-I-VMSOK, Installing HXDRIVER V1.3-008 on VMS Version V4.7
* Do you want to purge files replaced by this installation [YES]? yes

        The DRQ3B device (HXDRIVER) is not currently supported under
        the VMS AUTOCONFIGURATION utility.  In order to run the
        installation procedure, you must answer questions
        concerning the hardware installation. This procedure will
        then create the SYS$MANAGER:HX$LOAD.COM procedure to load
        the driver for the installation verification procedure.


        If you do not know these settings at this time, then use
        the default settings and do not request execution of the
        IVP.  You may then edit SYS$MANAGER:HX$LOAD.COM at your
        convenience to set the correct hardware parameters and to
        add additional DRQ3B interface units.

* Enter the first DRQ3B Q-Bus address (octal) [17760740]: 17760740
* Enter the first DRQ3B Q-bus vector (octal) [300]: 300
* Do you have additional DRQ3B's [NO]? no

        The installation will check for the VMS driver already installed
        in the system.  If there is no driver installed in the system you
        will receive the following message:

        "%SYSTEM-W-NOSUCHDEV, no such device available"

        This message can be ignored, the installation will proceed normally.
        Do not run the IVP if you must edit HX$LOAD.COM to insert the
        correct CSR and vector for your DRQ3B.
```

```
%SYSTEM-W-NOSUCHDEV, no such device available
* Do you want to run the IVP after the installation [YES]? yes

        The installation will now proceed with no further questions.
        A few more messages will be printed that contain information
        about the progress of the installation; the text of this
        information is duplicated in the installation section of the
        MicroVAX/DRQ3B Device Driver User's Guide.


        Linking the driver.  The linker will produce a warning
        message that the driver has no transfer address.   Drivers do not
        have a transfer address,  and this  message  should  be  ignored.

%LINK-W-USRTFR, image VMI$ROOT:[SYSUPD.HX013]HXDRIVER.EXE;1 has no user transfer address

        Linking the IVP program (HX$IVP).


        Linking the demo program (HX$TSTQ).

This installation creates the following directories/files:

SYS$SYSTEM:     HXDRIVER.EXE    ; DRQ3B device driver
                HXDRIVER.STB    ;   symbol table
                HXDRIVER.MAP    ;   link map

SYS$MANAGER:    HX$LOAD.COM     ; Command file to load the DRQ3B driver

SYS$LIBRARY:    HX$DEF.*        ; Definition files (.ADA, .BAS, .FOR, .H,
                                ;  .MLB, .PAS, .PLI, .REQ)

SYS$HELP:       HX$013.RELEASE_NOTES

SYS$TEST:       HX$IVP.COM      ; Command file to execute IVP

SYS$SYSROOT:[SYSTEST.HX]
                HX$IVP.EXE      ; The installation Verification Program (IVP)
                HX$TSTQ.EXE     ; A complicated DRQ3B QIO test program

SYS$SYSROOT:[SYSHLP.EXAMPLES.HX] (HX$EXAMPLES: after execution of HX$LOAD.COM)
                HX$EXAMPLE.C    ; A simple example of doing read/write
                HX$VER.H        ; Include file for HX$IVP and HX$TSTQ
                HX$IVP.C        ; Source for the IVP
                HX$IVP.OBJ
                HX$TSTQ.C       ; Source for the DRQ3B QIO test program
                HX$TSTQ.HLP     ; Help file for the DRQ3B test program
                HX$TSTQ.OBJ

%VMSINSTAL-I-SYSDIR, This product creates system directory [SYSHLP.EXAMPLES.HX].
%VMSINSTAL-I-SYSDIR, This product creates system directory [SYSTEST.HX].

   System Manager:
        Upon completion of this installation, please be sure to edit
        the SYS$MANAGER:HX$LOAD.COM procedure to conform to your
        installation.

        If you wish automatic driver installation at system startup,
        add the line:

                $ @SYS$MANAGER:HX$LOAD

        to your SYS$MANAGER:SYCONFIG.COM command procedure.

        If you have previously installed VAXlab/LabStar V1.1 with the
        DRQ3B, you should remove the line "$ @SYS$MANAGER:HXLOAD" from
        your SYCONFIG file (note the absence of the "$" in "HXLOAD")
        and only use HX$LOAD.COM.

        The release notes are located in SYS$HELP:HX$013.RELEASE_NOTES.

%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target directories...

Device                  Device          Error
 Name                   Status          Count
HXA0:                   Online              0
HXA1:                   Online              0
```

```
Beginning the HX V1.3-008 Installation Verification Procedure

HXA configuration:
   BIRQ level: 4
   Block Mode: extended
   Loopback cable: installed

Verification of MicroVAX/DRQ3B Device Driver HX V1.3-008 was successful

Completion of the HX V1.3-008 Installation Verification Procedure

        Installation of HX V1.3 completed at 11:43

        VMSINSTAL procedure done at 11:43

$ logo

  SYSTEM       logged out at 22-JUL-1988 11:44
```

2.7 Sample Installation Dialogue — VMS V5.0

The following is a sample installation dialogue for installation of the MicroVAX/DRQ3B Device Driver under VMS 5.0.

```
Username: SYSTEM
Password:


    Last interactive login on Friday, 22-JUL-1988 11:36
    Last non-interactive login on Friday, 22-JUL-1988 11:37

$ show default
  SYS$SYSROOT:[SYSMGR]
$ set default sys$update
$ @vmsinstal hx013 mua0:

        VAX/VMS Software Product Installation Procedure V5.0


It is 22-JUL-1988 at 11:38.
Enter a question mark (?) at any time for help.
* Are you satisfied with the backup of your system disk [YES]? yes

Please mount the first volume of the set on MUA0:.
* Are you ready? yes
%MOUNT-I-MOUNTED, HX mounted on _MUA0:

The following products will be processed:

  HX V1.3

        Beginning installation of HX V1.3 at 11:38

%VMSINSTAL-I-RESTORE, Restoring product saveset A...
%VMSINSTAL-I-RELMOVED, The products release notes have been successfully moved to SYS$HELP.

$!  Copyright (c) 1990 Digital Equipment Corporation.  All rights reserved.

%HX-I-VMSOK, Installing HXDRIVER V1.3-008 on VMS Version V5.0
* Do you want to purge files replaced by this installation [YES]? yes

        The DRQ3B device (HXDRIVER) is currently supported under
        the VMS AUTOCONFIGURATION utility.  In order to run the
        installation procedure, you must answer questions
        concerning the hardware installation. This procedure will
        then create the SYS$STARTUP:HX$LOAD.COM procedure to load
        the driver for the installation verification procedure.


        If you do not know these settings at this time, then use
        the default settings and do not request execution of the
        IVP.  You may then edit SYS$STARTUP:HX$LOAD.COM at your
        convenience to set the correct hardware parameters and to
        add additional DRQ3B interface units.
```

```
* Enter the first DRQ3B Q-Bus address (octal) [17760740]: 17760740
* Enter the first DRQ3B Q-bus vector (octal) [300]: 300
* Do you have additional DRQ3B's [NO]? no

        Product:      DRQ3B-DRIVER
        Producer:     DEC
        Version:      1.3
        Release Date: 30-MAY-1990


* Does this product have an authorization key registered and loaded? YES

        The installation will check for the VMS driver already installed
        in the system.  If there is no driver installed in the system you
        will receive the following message:

        "%SYSTEM-W-NOSUCHDEV, no such device available"

        This message can be ignored, the installation will proceed normally.
        Do not run the IVP if you must edit HX$LOAD.COM to insert the
        correct CSR and vector for your DRQ3B.

%SYSTEM-W-NOSUCHDEV, no such device available
* Do you want to run the IVP after the installation [YES]? yes

        The installation will now proceed with no further questions.
        A few more messages will be printed that contain information
        about the progress of the installation; the text of this
        information is duplicated in the installation section of the
        MicroVAX/DRQ3B Device Driver User's Guide.


        Linking the driver.  The linker will produce a warning
        message that the driver has no transfer address.   Drivers do not
        have a transfer address,  and this  message  should  be  ignored.

%LINK-W-USRTFR, image VMI$ROOT:[SYSUPD.HX013]HXDRIVER.EXE;1 has no user transfer address

        Linking the IVP program (HX$IVP).


        Linking the demo program (HX$TSTQ).

This installation creates the following directories/files:

SYS$LOADABLE_IMAGES:
                HXDRIVER.EXE    ; DRQ3B device driver
                HXDRIVER.STB    ;   symbol table
                HXDRIVER.MAP    ;   link map

SYS$STARTUP:    HX$LOAD.COM     ; Command file to load the DRQ3B driver

SYS$LIBRARY:    HX$DEF.*        ; Definition files (.ADA, .BAS, .FOR, .H,
                                ;  .MLB, .PAS, .PLI, .REQ)

SYS$HELP:       HX$013.RELEASE_NOTES

SYS$TEST:       HX$IVP.COM      ; Command file to execute IVP

SYS$SYSROOT:[SYSTEST.HX]
                HX$IVP.EXE      ; The installation Verification Program (IVP)
                HX$TSTQ.EXE     ; A complicated DRQ3B QIO test program

SYS$SYSROOT:[SYSHLP.EXAMPLES.HX] (HX$EXAMPLES: after execution of HX$LOAD.COM)
                HX$EXAMPLE.C    ; A simple example of doing read/write
                HX$VER.H        ; Include file for HX$IVP and HX$TSTQ
                HX$IVP.C        ; Source for the IVP
                HX$IVP.OBJ
                HX$TSTQ.C       ; Source for the DRQ3B QIO test program
                HX$TSTQ.HLP     ; Help file for the DRQ3B test program
                HX$TSTQ.OBJ

%VMSINSTAL-I-SYSDIR, This product creates system directory [SYSHLP.EXAMPLES.HX].
%VMSINSTAL-I-SYSDIR, This product creates system directory [SYSTEST.HX].
```

```
   System Manager:
        Upon completion of this installation, please be sure to edit
        the SYS$STARTUP:HX$LOAD.COM procedure to conform to your
        installation.

        If you have previously installed VAXlab/LabStar V1.1 with the
        DRQ3B, you should remove the line "$ @SYS$MANAGER:HXLOAD" from
        your SYCONFIG file (note the absence of the "$" in "HXLOAD")
        and only use HX$LOAD.COM.

        The release notes are located in SYS$HELP:HX$013.RELEASE_NOTES.

%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target directories...

Device                  Device          Error
 Name                   Status          Count
HXA0:                   Online              0
HXA1:                   Online              0


Beginning the HX V1.3-008 Installation Verification Procedure

HXA configuration:
   BIRQ level: 4
   Block Mode: extended
   Loopback cable: installed

Verification of MicroVAX/DRQ3B Device Driver HX V1.3-008 was successful

Completion of the HX V1.3-008 Installation Verification Procedure

        Installation of HX V1.3 completed at 11:43

        VMSINSTAL procedure done at 11:43

$ logo

  SYSTEM       logged out at 22-JUL-1988 11:44
```

3 Operating Features

Some of the operating features of the DRQ3B that you can control with the device driver include the following:

- Block or nonblock DMA transfers
- Double buffering
- Logical I/O or Physical I/O
- Interrupts enabled or disabled

These features are described in the following sections.

3.1 Block or Nonblock DMA Transfers

The DRQ3B device driver supports two types of DMA transfers across the Q-bus. These are:

- Burst mode (also called nonblock mode) transfer
- Block mode transfer

Block and nonblock transfers differ in regard to the addressing information passed across the Q-bus with data. In nonblock mode, each data word, when placed on the Q-bus, is accompanied by an address location. In block mode, only the first address in each block of data is asserted on the bus.

In block mode, the address location of the first data word is placed on the Q-bus followed by the data word. This is the only address information sent in a block of data. The next data word is sent, followed by the third data word and subsequent data words. In block mode, the receiving memory device or peripheral must be capable of incrementing the specified address location for each data word transferred. Sixteen data words are transferred before the DRQ3B gives up the bus. It does, however, monitor the bus halfway through the block of 16 data words. In block mode, the DRQ3B transfers seven data words and then monitors the bus request line to see if any other device is requesting the bus. If a device is requesting the bus, the DRQ3B transfers the eighth word, and then releases the bus to allow the other device to perform its task. If no other device is requesting the bus, the DRQ3B does not release the bus and transfers the remaining data words. After all 16 data words are transferred, the DRQ3B releases the bus. If there is still data to be transferred, the DRQ3B reasserts the bus request line. If no other device is requesting the bus, the DRQ3B regains the bus for transfer of the next block.

In block mode, when both channels of the DRQ3B have data to transfer, the first channel transfers eight words. The DRQ3B retains bus mastership and the second channel is allowed to transfer eight words. The DRQ3B then releases the bus. This allows both channels access to the bus.

In nonblock mode (also called burst mode), the DRQ3B transfers an address location with each data word transferred. Data transfer speeds are therefore approximately half that attainable in block mode, but transfers can be made to devices which cannot increment their own addresses. In burst mode, four address/data word combinations are transferred before the DRQ3B releases the bus. If additional data remains to be transferred, the DRQ3B again asserts its bus request line.

Block mode is the default mode for the MicroVAX/DRQ3B device driver. To select burst mode, use the function modifier, IO$M_NOBLOCK_MODE with the IO$_HX_DMA_INIT function code. However, if block mode is selected, and the memory does not support block mode transfers, the driver will automatically switch to burst mode.

The type of DMA transfer selected applies to both channels. If burst mode is selected, both channels are operated in burst mode. If block mode is selected both channels are operated in block mode.

3.1.1 Extended Block Mode Option

The extended block mode option allows higher data transfer rates, but is designed for environments where other modules are not frequently competing with the DRQ3B for bus mastership. In addition, extended block mode can only be used on MicroVAX based Q-bus systems. PDP-11 based systems do not support the extended block mode option.

Using the extended block mode option and the 1 microsecond holdoff time provides data transfer rates of up to 1.3 megahertz (words).

When extended block mode operation is selected, the normal release of the bus after 16 transfers only occurs if another device is currently requesting the bus. In normal block mode, the DRQ3B, once it acquires the bus, performs eight data transfers, but releases the bus if any other device is requesting it. If no other device is requesting the bus, the second block of eight data words is transferred and the bus is released. The DRQ3B then requests the bus again.

In extended block mode, the sequence is altered so that the bus is released only if another device is asserting its bus request line. The DRQ3B in extended block mode checks the bus after every eight data transfers and if the bus is not being requested by any device, the DRQ3B continues to transfer data. In extended block mode, the DRQ3B releases the bus when one of the following conditions occurs:

- When another device requests the bus
- When it has no more data to transfer
- When a match condition occurs
- When an end-of-process (EOP) condition occurs (see Section 3.7.1)

To use the extended block mode option, the extended block mode switch must be set on switch pack 1. This is the factory configuration. When the extended block mode switch is set, the DRQ3B uses extended block mode operation any time it is programmed for block mode transfers.

3.2 Double Buffering

Double buffering means that the DRQ3B is capable of storing pointers to a second buffer to be transferred. When more than one buffer is queued, using two or more SYS$QIO calls, the DRQ3B uses the address and length parameters of the first buffer to begin data transfer. The address and length parameters of the second buffer are stored *on the board* in secondary registers. When the first buffer completes transfer, the DRQ3B hardware uses the stored address and length information about the second buffer to immediately beginning transferring the second buffer. Double buffering is the default transfer method used by the driver. To disable double buffering, specify the IO$M_HX_NODOUBLE_BUFF function modifier when initializing the port with the IO$_HX_DMA_INIT function.

The double buffering capability allows the DRQ3B to transfer data continuously. As long your program continues to issue SYS$QIO service calls enqueuing new buffers before the previous buffer finishes, data is transferred continuously. Obviously, this is easier to accomplish with larger size data buffers. If very small data buffers are enqueued, your program may not have time to queue the next data buffer before the previous buffer finishes.

Double buffering is assured by queuing several buffers of data without starting DMA. All but the last data buffer must be queued with the IO$M_HX_NOSTART_DMA function modifier. When the last data buffer is queued the IO$M_HX_NOSTART_DMA modifier is left out of the request and DMA transfers start immediately, beginning with the first data buffer queued.

3.3 DMA I/O Transfer Modes — Logical and Physical

The DRQ3B driver can be operated in logical I/O mode or physical I/O mode. Logical I/O mode is less complex and is probably more convenient for most users. However, physical I/O mode uses less VMS overhead and may be desirable in situations where multiple data buffers are being transferred and the amount of VMS overhead being used is critical.

Physical I/O also allows data transfer directly to Q-bus devices.

In all transfers, the VMS operating system sets up a correspondence between the virtual addresses used in your program and the actual physical addresses used in system memory. The addresses that you use in your program to identify data structures or program components are virtual addresses. The VMS operating system automatically creates page tables that draw a correspondence between your virtual addresses and actual system memory addresses. When you create two data buffers that appear contiguous in your program, VMS maps the buffers into system memory, where they may or, more likely, may not be contiguous. VMS keeps track of the buffers by using the page table. The virtual address of your data buffer corresponds to an address in the page table. The content of the address in the page table is the address in system memory where your buffer resides. Entries in the VMS page table are called page frame numbers (PFN's).

For transfers between your process and an external device such as the DRQ3B, the operating system also uses mapping registers to map physical pages in VAX memory to Q-bus addresses. The I/O buffers in your process' virtual address space are designated as Q-bus memory space. Normally, local VAX memory cannot be accessed by ''external'' Q-bus devices. Since the DRQ3B operating under DMA is an external device from the vantage point of your process, it is prevented from writing or reading in your process space. However, a section of your process memory can be mapped to the Q-bus address area. In other words, your process (the executing program) and the DRQ3B can both access this local VAX memory. The DRQ3B accesses it using a Q-bus address. Your program accesses it using the virtual address defined in your program.

The Q-bus mapping registers create a correspondence between VAX memory page addresses and actual Q-bus addresses. Each mapping register corresponds to an area of Q-bus memory. By writing the contents of the page table entries into the mapping registers, both the page table entry and the Q-bus mapping registers point to the buffer in system memory. This means that when the DRQ3B specifies a Q-bus address (by placing the address on the Q-bus), the VAX responds to this address by accessing the Q-bus memory area of your process's I/O space. The Q-bus mapping registers identified with this Q-bus memory area point to the area in system memory which is also pointed to by the page table. Thus, your process and the DRQ3B can access the same buffers.

The mapping correspondences described above occur in both logical I/O mode and physical I/O mode when transferring buffers from your process to or from the DRQ3B. The difference between the two modes lies in when the allocation and loading of the mapping registers occurs.

3.3.1 Logical I/O Operation

In logical I/O, allocation and loading of Q-bus mapping registers occurs each time a buffer is queued for transfer. In other words, the driver and VMS must lock the buffers into memory, allocate mapping registers, and load them with the PFN from the page table before data transfer begins. There is a fair amount of overhead time involved. However, logical I/O is extremely convenient. You do not need to know in advance how many buffers you are going to transfer. You do not need to reserve space in your program's data area for the driver to load with Q-bus addresses. In addition, you do not need to worry about possibly writing or reading data from invalid areas. Your program does not need any special privileges assigned by the system manager. Lastly, though not insignificantly, you do not need to understand the VMS mapping process briefly described above.

3.3.2 Physical I/O Operation

When physical I/O mode is used to transfer buffers between your process and the DRQ3B, all of the overhead involved in locking buffers, allocating, and loading of mapping registers is done *before any data buffer is queued for transfer.* This means that when the SYS$QIO call executes and queues a buffer, the VMS I/O system has already done its overhead

processing. This has a beneficial effect on transfer rates, but it has an even greater effect on CPU utilization. The CPU spends less time queuing and transferring physical I/O buffers than it does transferring logical I/O buffers.

However, physical I/O transfers require more complex programming. You must specify the maximum number of buffers to be transferred in order to allocate enough mapping registers. You also must have the PHY_IO privilege, which is assigned by the system manager. You also must take great care in programming, since the operating system will now permit you to access any physical address, whether it belongs to you or not. Specification of the wrong address will most likely cause the system to crash.

In addition, to use physical I/O transfers to mapped Q-bus memory, you need to specify the actual Q-bus address which identifies your buffer, not the virtual address. However, the DRQ3B driver calculates automatically the Q-bus address used to access local VAX memory which you have mapped. You do not need to calculate or be concerned with the actual number. You do however, have to specify a variable in your program to hold the Q-bus address of your buffer.

You also need to allocate a data structure in your program's data area called the user mapping register descriptor, or UMRD. This data structure is used by the driver and should not be accessed by your program, although the data structure must be in your program.

The virtual address of the Q-bus address variable is passed as an argument to the IO$_HX_ALLOC_MAP function code and the IO$_READPBLK or IO$_WRITEPBLK function code. The virtual address of the UMRD data structure is passed as an argument to the IO$_HX_ALLOC_MAP function code.

The driver writes to and reads from the UMRD data structure and Q-bus address descriptor during execution. It is extremely important that your program not modify or access these structures during execution. If it does, the results are unpredictable. However, it is likely that the driver, if it uses incorrect information obtained from a modified UMRD structure, would crash the system.

More information on the data structures you must allocate are described in Sections 3.5.2.3 and 3.5.2.4.

A second type of physical I/O involves transfers to or from a Q-bus device. In this scenario, your process only sets up the DRQ3B for operation. The buffers transferred by the DRQ3B are not part of your process. Once the DRQ3B starts, it transfers data to the specified Q-bus address without any intervention by your process. Locking of buffers, and allocation and loading of mapping registers does not occur. The actual Q-bus address of the data to be transferred is given to the driver. It is assumed that you already know the destination's or source's Q-bus address. For example, many Q-bus devices, such as an array processor, have a data window register with a defined Q-bus address. The address of this register can be directly specified in physical I/O.

3.3.3 Advantages of Physical I/O

Physical I/O's primary advantage over logical I/O is CPU utilization rather than transfer rates. Although physical I/O can achieve faster transfer rates than logical I/O, given the same size buffer, the increased transfer rate is only a small percentage for large buffers. However, physical I/O uses far less CPU time than logical I/O to transfer equivalent size buffers. In other words, the CPU has more time to handle other tasks when the DRQ3B driver is operated in physical I/O mode.

This advantage increases as the buffer size increases. With larger buffers, in the range of 32K and 64K (bytes), physical I/O uses one-fourth to one-fifth of the CPU time required for logical I/O. However, this advantage is diminished when small buffer sizes (less than 4K) are used.

If you use your system in a multi-tasking environment, and have many processes active at the same time and are transferring large buffers, physical I/O may be of great benefit to you. If, however, your system is dedicated to data collection and is not performing other tasks while a single driver process is executing, there is no real advantage achieved by using physical I/O.

If you want to transfer data to or from another Q-bus device, you must use physical I/O. Logical I/O cannot be used for these situations.

3.4 Using Logical I/O

The logical I/O mode is selected by using the IO$_READLBLK and IO$_WRITELBLK function codes. Each takes two arguments, P1 and P2. P1 specifies the starting address of the buffer. The virtual address of the buffer is entered. For example, if you have labeled your buffer BUF1, the virtual address BUF1, is entered as P1. P2 specifies the size of the data buffer in bytes. If you calculate the length of the buffer dynamically and store the length in a symbol, this symbol name can be entered as P2.

The IO$_READLBLK can only be issued to the HXa0 input port. The IO$_WRITELBLK function code can only be issued to the HXa1 output port.

Logical I/O does not require any process privileges, provided that the device protection is set to allow access (this is the default).

3.4.1 Logical I/O Summary

The following summarizes the steps you need to take to perform logical I/O data transfers.

1. Call SYS$ASSIGN to assign a channel to the port.
2. Call SYS$QIOW with the IO$_HX_DMA_INIT function code to initialize the module. Only one argument, P1, the size of the largest buffer, is specified. P2 through P6 are not used and cannot be specified as zero.

3. Call SYS$QIO with the IO$_READLBLK function code to queue a data buffer for input. (IO$_WRITELBLK is used for output.) The logical address of the data buffer is specified in P1 and the size in bytes is specified as P2.
4. Repeat step 3 to queue subsequent data buffers.
5. Call SYS$DASSGN when all data transfers are finished.

3.5 Using Physical I/O

Physical I/O involves transfer using a Q-bus address. There are two situations in which this can occur.

Physical I/O can be used for transfer between your process and the DRQ3B. In this situation, physical I/O is generally selected because of its better CPU utilization performance over logical I/O.

Physical I/O can also be used for transferring data directly between the DRQ3B and another Q-bus device. In this situation, physical I/O must be used; logical I/O does not support this type of transfer.

In both types of physical I/O your process must have the PHY_IO privilege. In addition, because physical I/O allows you to directly access any address on the Q-bus, intentionally or otherwise, it requires that you be extremely careful.

Note: Physical I/O mode bypasses many of the protective features offered by VMS. Using it requires a detailed knowledge of the VMS architecture and memory layout.

3.5.1 Physical I/O Read and Write Logical Blocks

Both types of physical I/O transfers are done by using the IO$_READPBLK and IO$_WRITEPBLK function codes. Each takes two arguments, P1 and P2. P1 specifies the starting address of the buffer. This is the Q-bus address of the buffer. P2 specifies the size of the data buffer in bytes. If you calculate the length of the buffer dynamically and store the length in a symbol, this symbol name can be entered as P2.

As with the logical I/O mode, the IO$_READPBLK can only be issued to the HXa0 input port. The IO$_WRITEPBLK function code can only be issued to the HXa1 output port.

3.5.2 Physical I/O to or from Local VAX Memory

By mapping a section of your process I/O space to the Q-bus address range, both your process and the DRQ3B can access the same buffers. This type of physical I/O requires that you allocate mapping registers for each buffer to be transferred. It also requires that you allocate data structures within your program's data area to be used by the driver. The driver uses these areas to keep track of the mapping registers allocated and to calculate and store the Q-bus addresses.

3.5.2.1 Page Locking

When using physical I/O, the MicroVAX II mapping registers are used to define a correspondence (map) between addresses in your virtual memory with actual addresses on the Q-bus. To make certain that this correspondence does not change while your process is running, you must lock your process pages in memory prior to allocating the mapping registers. This is done using the system service routine SYS$LCKPAG. Your process needs the PSWPM privilege to successfully issue this routine.

Further information on this system service routine can be found in the *VAX/VMS System Services Reference Manual*.

3.5.2.2 Allocating Mapping Registers

When using physical I/O for mapped Q-bus addresses, you must explicitly allocate the mapping registers used for each buffer. The driver provides a function code, IO$_HX_ALLOC_MAP, which performs this function. The mapping registers should be deallocated before your program terminates using the IO$_HX_DEALLOC_MAP function code.

3.5.2.3 Q-bus Address Descriptor

In physical I/O mode, the actual 22-bit Q-bus address of a buffer is passed as an argument to the IO$_READPBLK and IO$_WRITEPBLK function codes. You are not expected to know this address; the driver calculates it for you when the IO$_HX_ALLOC_MAP function code executes. Therefore, you specify the Q-bus address using a symbol. For each buffer to be transferred, you must allocate an area in your program for the Q-bus address of the buffer. This data space in your data area must be one longword and described by a virtual address. The virtual address you choose is specified as the P3 argument to the IO$_HX_ALLOC_MAP function code.

When the IO$_HX_ALLOC_MAP function code is called, the driver determines the Q-bus address of the buffer and writes it into the data area you have specified (the Q-bus address descriptor). When you call IO$_READPBLK or IO$_WRITEPBLK, the virtual address of the Q-bus address descriptor is passed as an argument. Since this Q-bus descriptor has already been written with the Q-bus address of the buffer, the driver has, in effect, been passed the physical address which it uses to directly access the buffer.

Note: The Q-bus address descriptor data space is used by the driver to write and read the Q-bus address of the buffer. You do not need to access the data area and should not write anything into it.

You need to allocate a Q-bus address descriptor for each buffer to be transferred.

3.5.2.4 User Mapping Register Descriptor

In physical I/O mode, the MicroVAX/DRQ3B device driver needs to keep track of the mapping registers assigned and the values written to them. It does this by reading and writing an area in your program's virtual address space. This area is called the User Mapping Register Descriptor. When you write your program, you need to declare a data area to contain this information. The size of the data can be specified using a length parameter (which is defined in the UMRD definition file) to declare an array or structure of that size. The available length parameters include the following:

Length Parameter	Meaning
HX$K_UMRD_BYTE_LEN	Length of UMRD in bytes
HX$K_UMRD_WORD_LEN	Length of UMRD in words
HX$K_UMRD_LONG_LEN	Length of UMRD in longwords

A virtual address should be assigned to the UMRD. This label is passed as the P4 argument to the IO$_HX_ALLOC_MAP function code. The driver then writes information about the mapping registers assigned and reads the UMRD table as needed. You do not need to be concerned with the information placed in the UMRD; declaring a data area in your program for this table is sufficient. You must, however, be certain that your program itself does not address this data space or write over it during execution.

Note: The UMRD is both written and read by the driver. You do not need to access this data area and should not write anything into it.

You need to allocate a UMRD table for each buffer to be transferred.

3.5.2.5 Summary of Physical I/O to Local VAX Memory

The list below summarizes the steps you need to take to perform physical I/O transfers between the DRQ3B and buffers in your process' address space.

1. Call SYS$ASSIGN to assign a channel to the port.
2. Call SYS$QIOW with the IO$_HX_DMA_INIT to initialize the module. The P1 argument must be zero. P2 is the size in bytes of the largest buffer. P3 is the number of buffers to be transferred.
3. Call SYS$LCKPAG to lock the buffer in VAX memory into the process.
4. Call SYS$QIO with the IO$_HX_ALLOC_MAP function code to allocate individual mapping registers for buffer transfers. P1 is the virtual address of the buffer. P2 is the length of the buffer in bytes. P3 is the Q-bus address for this buffer. P4 is the UMRD virtual address for this buffer.
5. Repeat steps 3 and 4 for each subsequent data buffer to be queued.
6. Call SYS$QIO with the IO$_READPBLK function code to queue the first data buffer. P1 is the Q-bus address (same one as in P3 above). P2 is the size of the buffer in bytes (same as P2 above). P3 through P6 are not used and cannot be specified as zero.

7 Repeat step 6 for each subsequent data buffer to be queued.

8 Call SYS$QIO with IO$_HX_DEALLOC_MAP function code to release all of the mapping registers allocated. P1 is the virtual address for the UMRD for that buffer. This step should be repeated for each time IO$_HX_ALLOC_MAP was called.

9 Call SYS$DASSGN when all data transfers are finished.

To use physical I/O, you must allocate space in your program for the UMRD and the Q-bus address of each buffer. The UMRD is passed by reference to the IO$_HX_ALLOC_MAP and IO$_HX_DEALLOC_MAP function codes. The Q-bus address is passed by value to the read or write physical block function codes and is returned by the IO$_HX_ALLOC_MAP function codes. See Sections 3.5.2.3 and 3.5.2.4.

3.5.3 Physical I/O to or from Q-bus Memory

In physical I/O mode, the DRQ3B can read from or write to Q-bus memory directly. In this case, allocation of mapping registers is not needed. Instead the starting point of the buffer is considered to be the memory address in Q-bus memory.

For example, if you are reading data from an external device and want to send it directly from the DRQ3B to an array processor, the Q-bus memory address of the array processor can be specified as the P1 argument of the read physical block function code (IO$_READPBLK).

For block mode transfers to Q-bus memory, the Q-bus memory board must be able to increment its own address locations automatically. The DRQ3B hardware, however, will automatically switch to burst mode is block mode transfers are specified, and the memory is incapable of incrementing its own address.

For physical I/O transfers to Q-bus memory, mapping registers are not needed and the allocation and deallocation steps can be skipped.

Note: Version 1.3 of the MicroVAX/DRQ3B Device Driver does not support I/O page addressing.

3.5.3.1 Summary of Physical I/O to Q-bus Memory

The list below summarizes the steps you need to take to perform physical I/O transfers between the DRQ3B and Q-bus memory. In these situations, it is assumed that you know the address of the Q-bus device. The actual Q-bus address can be directly entered as the P1 argument to IO$_WRITEPBLK or IO$_READPBLK. It is not necessary to use a Q-bus address descriptor or UMRD.

1 Call SYS$ASSIGN to assign a channel to the port.

2 Call SYS$QIOW with the IO$_HX_DMA_INIT to initialize the module. The P1 argument must be zero. P2 is the size in bytes of the largest buffer. The P3 argument must be zero.

3 Call SYS$QIO with the IO$_READPBLK function code to queue the first data buffer. P1 is the actual Q-bus address of the destination or source. P2 is the size of the buffer in bytes. P3 through P6 are not used and cannot be specified as zero.

4 Repeat step 4 for each subsequent data buffer to be queued.

5 Call SYS$DASSGN when all data transfers are finished.

Although not required, it may be useful to use the system service routine, SYS$CRMPSC, Create and Map Section, when using physical I/O for transfers between the DRQ3B and Q-bus devices. The SYS$CRMPSC routine allows a process to associate a section of its address space with physical addresses represented by page frame numbers. Further information on this system service routine can be found in the *VAX/VMS System Services Reference Manual*.

3.6 Selecting I/O Mode

Logical or physical I/O is selected by using the IO$_HX_DMA_INIT function code. When this function code is called with P1 not equal to zero, logical I/O is selected. P1 specifies the size of the largest buffer to be transferred and the driver expects that only logical read and write function codes (IO$_READLBLK and IO$_WRITELBLK) will be called.

If the IO$_HX_DMA_INIT function code is called with P1 equal to zero, physical I/O is selected and P2 and P3 must be specified. P2 specifies the size of the largest buffer to be transferred and P3 specifies the number of buffers. The driver expects that only physical read and write function codes (IO$_READPBLK and IO$_WRITEPBLK) will be called.

3.7 Interrupts and Attention AST's

There are five types of conditions which can cause the DRQ3B to generate a hardware interrupt. These include the following:

- Channel 0 end-of-process (EOP)
- Nonexistent memory end-of-process (EOP)
- External interrupt
- FIFO Underflow (input only)
- FIFO Overflow (output only)

These conditions, with the exception of the channel 0 EOP condition, are considered to be undesired error terminations. The device driver allows you to enable each condition individually. When enabled, you specify an attention AST (asynchronous system trap) routine which executes when the condition occurs.

The channel 0 EOP is considered to be an intentional end-of-process and is handled in the same way as a successful buffer completion.

The channel 0 EOP is always enabled. You can select whether to enable or disable the other conditions. Interrupt conditions are enabled using the IO$_SETMODE function code and the IO$M_HX_ATTN_ENABLE function modifier. These are described in Section 4.3.2.

Except for the channel 0 EOP and nonexistent memory error, when the interrupt conditions occur, data transfer does not stop, but the attention AST that you have specified executes. How you handle the condition is up to you.

These conditions are described individually in the following sections.

3.7.1 End-of-Process

The term end-of-process refers to a condition where the transfer terminates before the complete buffer transfers.

A channel 0 EOP occurs when a high to low transition is detected on the EOP pin present on the channel 0 port connector. A nonexistent memory EOP occurs when the DRQ3B attempts to access nonexistent memory.

Both types of EOP conditions set the EOP bit (bit 1) in the DMA status register. However, only the nonexistent memory EOP sets the nonexistent memory bit (bit 15) in the status register.

3.7.2 Channel 0 EOP

The input connector on the DRQ3B includes an EOP pin. This pin can be used by external devices to stop data transfer and flag error conditions. When a signal is asserted low (less than 0.5 volts) on the EOP pin, the data word present on the data lines at that time is flagged. (The EOP line should be synchronized with the handshaking line STROBE; EOP should go low when STROBE goes low.) When the flagged data word is transferred out of the FIFO buffer and onto the Q-bus, the DRQ3B terminates the current buffer transfer. The FIFO remains full with any subsequent words that were transferred after the EOP data word. If there are additional buffers available, the DRQ3B will will begin filling the next buffer with the data in the FIFO.

When a buffer terminates because of a channel 0 EOP, the IOSB status code indicates this with the successful status code, SS$_WASSET. The word count is probably less than what you expected, but it is assumed that the assertion of the EOP line was intentional and therefore, the buffer transfer is considered successful.

The driver uses chaining to process a channel 0 EOP interrupt in order to read the transfer count register and determine where in the buffer the EOP occurred. As a result, the EOP bit will not be set in the DMA status register returned in the read/write I/O status block.

There is no equivalent EOP pin for the channel 1 output of the DRQ3B.

This interrupt is a function of the DRQ3B hardware and is always enabled.

When the channel 0 EOP occurs, the buffer transfer terminates and the read/write I/O status block is written with a successful status code indicating the EOP condition. If you specified an AST routine or event flag to indicate completion of the buffer, the AST routine executes or the event flag is set.

3.7.3 Nonexistent Memory EOP

The nonexistent memory EOP is generated when the DRQ3B cannot find the specified memory location within the Q-bus time limit. Generally this means that the location does not exist. Either the address currently being used is not on the system, or the memory is not responding for some reason.

Data transfer stops when this error occurs. SS$_ABORT is returned in IOSB for this buffer.

This condition can be enabled or disabled using the IO$_SETMODE function code and IO$M_HX_ATTN_ENABLE function modifier.

3.7.4 External Interrupt

The general purpose external interrupt occurs when there is a high to low transition on the external interrupt pin of the port connectors. There is only one external interrupt signal, but it is presented on a pin in both connectors. This interrupt condition can be used for customized purposes.

The external interrupt pin differs from the channel 0 EOP interrupt. DMA transfers are not stopped when an external interrupt occurs. An interrupt condition occurs, but the DRQ3B continues to transfer the current data buffer.

This interrupt can be enabled or disabled using the IO$_SETMODE function code and IO$M_HX_ATTN_ENABLE function modifier.

3.7.5 FIFO Underflow

The FIFO underflow condition occurs when the port 1 FIFO is empty, but the external device attempted to read an additional word from the FIFO. This generally indicates that the external device is receiving data at a much faster rate than the DRQ3B is capable of sending it. This condition usually only occurs when a one-wire handshaking scheme is used, or the external device's two-wire handshaking scheme is not operating properly.

This condition can be enabled or disabled using the IO$_SETMODE function code and IO$M_HX_ATTN_ENABLE function modifier.

3.7.6 FIFO Overflow

The FIFO overflow condition occurs when the FIFO is full and the external device attempts to strobe an additional data word into the FIFO. This condition usually only occurs when a one-wire handshaking scheme is used, or the external device's two-wire handshaking scheme is not operating properly.

This condition can be enabled or disabled using the IO$_SETMODE function code and IO$M_HX_ATTN_ENABLE function modifier.

4 Function Codes and I/O Status Blocks

The MicroVAX/DRQ3B device driver is invoked using the SYS$QIO system service call. Complete details on the SYS$QIO system service routine can be found in the *VAX/VMS System Services Manual*.

The driver is controlled by passing function codes and function code modifiers to the SYS$QIO service routine. The following sections describe the function codes and function modifiers for this service routine.

4.1 Definition Files

The definitions of the function codes and modifiers are contained in language specific files. Your program should include the definition file appropriate for your language. The definition file is named HX$DEF.*, where * is the extension appropriate for your language. The language extension names are shown in Table 4–1.

Table 4–1 Definition Files

File Extension	Language
HX$DEF.ADA	ADA
HX$DEF.BAS	BASIC
HX$DEF.FOR	Fortran
HX$DEF.H	C
HX$DEF.MLB	Macro Library
HX$DEF.PAS	Pascal
HX$DEF.PLI	PL/I
HX$DEF.R32	Bliss

Note: When you include the HX$DEF file, this file should appear first in any list of included files you have.

4.2 Function Codes, Arguments, and Modifiers

The following sections list the function codes, their arguments, and the function modifiers that can be used with each mode of operation.

4.2.1 Function Codes

Table 4–2 shows the function codes used with logical I/O transfers.

Table 4–2 Function Codes for Logical I/O

Function Code	Meaning
IO$_HX_DMA_INIT	Initializes the port; should be called using SYS$QIOW.
IO$_READLBLK	Read data in through port 0.
IO$_WRITELBLK	Write data out through port 1.
IO$_SETMODE	Write or clear the function bits on this port and enable or disable the external and FIFO overflow/underflow interrupts.
IO$_SENSEMODE	Read all the function bits.

Table 4–3 shows the function codes used with physical I/O transfers.

Table 4–3 Function Codes for Physical I/O

Function Code	Meaning
IO$_HX_DMA_INIT	Initializes the port; should be called using SYS$QIOW.
IO$_HX_ALLOC_MAP	Allocate mapping registers.
IO$_HX_DEALLOC_MAP	Deallocate mapping registers.
IO$_READPBLK	Read data in through port 0.
IO$_WRITEPBLK	Write data out through port 1.
IO$_SETMODE	Write or clear the function bits on this port and enable or disable the external and FIFO overflow/underflow interrupts.
IO$_SENSEMODE	Read all the function bits.

4.2.2 Function Code Arguments

All of the function codes listed above take arguments. The sections and tables below describe the arguments used by the function codes for each mode.

The SYS$QIO call requires that six arguments, P1 through P6, be specified. Arguments not used by a function code should be specified as null arguments and are not listed in the following tables. Null arguments are indicated by commas only. For example, in the following line of code, arguments P3 through P6 are null arguments:

```
status = sys$qio(0,chan,IO$_READLBLK,iosb,ast,ast_param,BUF1,3000,,,,)
```

4.2.2.1 Logical I/O Function Code Arguments

Arguments for all of the function codes used with logical I/O are listed in Table 4–4.

Table 4–4 Function Code Arguments for Logical I/O

Function Code	Argument	Passed by	Meaning
IO$_HX_DMA_INIT	P1	Value	Size in bytes of the largest buffer to be transferred. Must be even.
IO$_READLBLK	P1	Reference	Starting address (virtual) of the buffer.
	P2	Value	Size of the buffer in bytes; maximum is 128K.
IO$_WRITELBLK	P1	Reference	Starting address (virtual) of the buffer.
	P2	Value	Size of the buffer in bytes; maximum is 128K.
IO$_SETMODE	P1	Value	With IO$M_HX_FUNCT_BITS, P1 specifies the longword containing the bits to be set or cleared. With IO$M_HX_ATTN_ENABLE, P1 specifies the address of the AST routine to be executed when interrupt condition occurs.
	P2	Reference	With IO$M_HX_ATTN_ENABLE only, P2 specifies the address of an attention mask.
	P3	Value	With IO$M_HX_ATTN_ENABLE only, P3 specifies the access mode to deliver the AST.
IO$_SENSEMODE			No arguments.

4.2.2.2 Physical I/O Function Code Arguments

Arguments for all of the function codes used with physical I/O are listed in Table 4–5.

Table 4–5 Function Code Arguments for Physical I/O

Function Code	Argument	Passed by	Meaning
IO$_HX_DMA_INIT	P1	Value	Always zero in physical I/O.
	P2	Value	Size in bytes of the largest buffer to be transferred. Must be even.
	P3	Value	Number of buffers to be transferred, for physical I/O to or from local VAX memory; Always zero for physical I/O to or from Q-bus memory
IO$_HX_ALLOC_MAP	P1	Reference	The virtual address of the buffer.
	P2	Value	The length of the buffer in bytes.
	P3	Reference	The returned Q-bus address for this buffer.
	P4	Reference	The virtual address of the user mapping register descriptor (UMRD) for this buffer.
IO$_HX_DEALLOC_ MAP	P1	Reference	The virtual address of the user mapping register descriptor (UMRD) for this buffer.
IO$_READPBLK	P1	Value	Starting Q-bus address (physical) of the buffer.
	P2	Value	Size of the buffer in bytes; maximum is 128K.
IO$_WRITEPBLK	P1	Value	Starting Q-bus address (physical) of the buffer.
	P2	Value	Size of the buffer in bytes; maximum is 128K.
IO$_SETMODE	P1	Value	With IO$M_HX_FUNCT_BITS, P1 specifies the longword containing the bits to be set or cleared. With IO$M_HX_ATTN_ENABLE, P1 specifies the address of the AST routine to be executed when interrupt condition occurs.
	P2	Reference	With IO$M_HX_ATTN_ENABLE only, P2 specifies the address of a quadword attention mask.
	P3	Value	With IO$M_HX_ATTN_ENABLE only, P3 specifies the access mode to deliver the AST.
IO$_SENSEMODE			No arguments.

4.2.3 Function Modifiers

Function modifiers are used with the function codes to provide additional control over how the function code operates.

Table 4–6 shows the function modifiers used for logical I/O and physical I/O transfers. The modifiers have the same meaning in all modes of transfer.

Table 4–6 Function Modifiers for Logical and Physical I/O

Function Code	Modifiers	Meaning
IO$_HX_DMA_INIT	IO$M_HX_NOBLOCK_MODE	Use burst mode DMA.
	IO$M_HX_NODOUBLE_BUFF	Wait for each DMA transfer to complete before starting the next DMA transfer. This does not mean that the data must also be out of the FIFO before starting the next write operation, only that the last word of the transfer has been placed in the FIFO. Use the IO$M_HX_RUN_DOWN modifier to ensure that the FIFO is empty before starting the next write operation. For read operations you have no control over the FIFO state.
IO$_READLBLK	IO$M_HX_FIFO_CLEAR	Clear the FIFO before starting this request. If specified during a double buffered transfer, this modifier is ignored for all but the initial buffer of the transfer.
	IO$M_HX_NOSTART_DMA	Queue request but do not start transfer.
IO$_WRITELBLK	IO$M_HX_RUN_DOWN	Wait for DRQ3B FIFO to empty before issuing I/O complete status.
	IO$M_HX_FIFO_CLEAR	Clear the FIFO before starting this request. If specified during a double buffered transfer, this modifier is ignored for all but the initial buffer of the transfer.
	IO$M_HX_NOSTART_DMA	Queue request but do not start transfer.
IO$_READPBLK	IO$M_HX_FIFO_CLEAR	Clear the FIFO before starting this request. If specified during a double buffered transfer, this modifier is ignored for all but the initial buffer of the transfer.
	IO$M_HX_NOSTART_DMA	Queue request but do not start transfer.
IO$_WRITEPBLK	IO$M_HX_RUN_DOWN	Wait for DRQ3B FIFO to empty before issuing I/O complete status.
	IO$M_HX_FIFO_CLEAR	Clear the FIFO before starting this request. If specified during a double buffered transfer, this modifier is ignored for all but the initial buffer of the transfer.
	IO$M_HX_NOSTART_DMA	Queue request but do not start transfer.
IO$_SETMODE	IO$M_HX_FUNCT_BITS	Set or clear the function bits as specified in argument P1.
	IO$M_HX_ATTN_ENABLE	Enable attention AST's for the interrupt conditions defined in the attention mask specified in argument P2. The AST address is specified in argument P1 and the AST access mode in argument P3.
IO$_SENSEMODE		No modifiers.
IO$_HX_ALLOC_MAP		No modifiers.
IO$_HX_DEALLOC_MAP		No modifiers.

4.2.3.1 IO$M_HX_NOBLOCK_MODE and IO$M_HX_NODOUBLE_BUFF

For the IO$_HX_DMA_INIT function code, double buffering and block mode transfers are the default. The IO$M_HX_NOBLOCK_MODE overrides the default and uses burst mode to transfer all of the buffers queued under this initialization. This might be desired if the memory being used is not capable of incrementing its own address. However, the DRQ3B hardware will switch the transfer mode to burst mode if block mode is selected and the memory is not capable of incrementing its own address.

The IO$M_HX_NODOUBLE_BUFF function modifier overrides the double buffering default for all of the buffers queued under this initialization.

4.2.3.2 IO$M_HX_FIFO_CLEAR

For the read and write logical block and read and write physical block function codes, the IO$M_HX_FIFO_CLEAR function code clears the input FIFO buffer prior to reading in data. The default value is not to clear the FIFO before transferring data. If data words are in the buffer when the first transfer begins, and IO$M_HX_FIFO_CLEAR is not specified, these words will be read into the users first data buffer. If specified during a double buffered transfer, this modifier is ignored for all but the initial buffer of the transfer.

4.2.3.3 IO$M_HX_NOSTART_DMA

The IO$M_HX_NOSTART_DMA function modifier is used to queue up several data buffers without beginning transfer. Each buffer, except the final buffer, includes this function modifier as part of the SYS$QIO call. The final buffer to be queued leaves this function modifier off of the request; the DRQ3B then begins transferring data, starting with the first data buffer queued.

4.2.3.4 IO$M_HX_RUN_DOWN

This modifier, valid only with the write logical and write physical function codes, specifies that the transfer complete notification occurs when the last data word is transferred out of the data port FIFO. Normally transfer complete notification occurs when the last data word is placed in the port FIFO by the DMA engine. With this modifier, the driver will wait for the FIFO empty interrupt at the end of the transfer before returning to the user.

Use of this modifier effectively disables double buffering for the specific data buffer specified in the SYS$QIO call. Without this modifier, the second data buffer is started as soon as the last data word is placed in the port FIFO. When the IO$M_HX_RUN_DOWN modifier is used, the second data buffer is not started until the first data buffer has been transferred out of the port FIFO. There is, therefore, a brief hiatus between buffers.

4.2.3.5 IO$M_HX_FUNCT_BITS and IO$M_HX_ATTN_ENABLE

There are two function modifiers for the set mode function code. The modifiers specify whether the function bits are controlled or the interrupts are enabled or disabled.

The IO$M_HX_FUNCT_BITS function modifier must be specified if it is desired to write the general purpose function bits. When this modifier is specified, the P1 argument contains a data word which sets or clears the three output function bits on this port (each port has three output function bits).

The IO$M_HX_ATTN_ENABLE function modifier must be specified when it is desired to enable or disable the supported interrupts.

4.3 Set Mode and Sense Mode

The standard SYS$QIO function codes, IO$_SETMODE and IO$_SENSEMODE, are used to control the general purpose I/O function bits on the DRQ3B port connectors, and enable or disable the external interrupt and FIFO overflow and underflow interrupts.

4.3.1 Controlling the Function Bits

The general purpose function bits on the DRQ3B are controlled using the set mode and sense mode function codes. These standard function codes for SYS$QIO allow you to read the input function bits and write to the output function bits.

There are six input and six output general purpose bits. Three input pins and three output pins are located on the connector for each port. These pins are labeled FUNCT OUT or FUNCT IN in Figures B–1 and B–2. The number of each pin corresponds to the bit in the function registers. The pins are completely independent of the port's operation and are available for customized use. Examples of such use might be remote power-on of external devices, or additional information transfer, such as monitoring the status of an external device. The type of uses possible with the output function bits depends upon the signals present on the external device.

The DRQ3B has an output function register which is used to write data to the output function bits. The IO$_SETMODE function code and IO$M_HX_FUNCT_BITS modifier is used to write data to this register. Data written to the output function bits is latched, that is, the output bits continually assert the signals written until new data is written.

The DRQ3B also has an input function register for reading the input function bits. The IO$_SENSEMODE function code is used to read the input function bits. The input function bits are not latched and continually reflect the signals present on the input pins.

4.3.1.1 Writing to the Output Function Bits

Writing data to the output function bits is done by issuing a SYS$QIO system service call with IO$_SETMODE as the function code and IO$M_HX_FUNCT_BITS as a function modifier. There is one argument. The P1 argument points to a data word that contains the data to be written to the three function bits. Generally the label of the longword is supplied as the argument. The low order bits of the longword specify which of the function bits to set or clear. A one in a bit position sets the FUNCT OUT output high. Bit 0 corresponds to the lowest order function bit; bit 2 corresponds to the high order function bit. Thus a data word that specifies 110 (binary) sets FUNCT OUT 4 and 5, but leaves FUNCT OUT 3 clear, when the IO$_SETMODE call is issued to HXA1.

Table 4–7 illustrates the correspondence between the bits in the P1 argument and the pins on the port connector.

Table 4–7 Correspondence Between P1 Bits and Function Bits

P1 bit	HXa0 Input	HXa1 Output
0	FUNCT OUT 0	FUNCT OUT 3
1	FUNCT OUT 1	FUNCT OUT 4
2	FUNCT OUT 2	FUNCT OUT 5

4.3.1.2 Reading the Input Function Bits

The input function bits are read by issuing the SYS$QIO system service routine with the IO$_SENSEMODE function code. The status of the input bits, as well as the output bits, is returned in the sense mode I/O status block. See Section 4.4.3.

4.3.2 Enabling and Disabling Attention AST's

The DRQ3B driver allows you to enable or disable the following conditions for attention AST's.

- External interrupt
- Nonexistent memory interrupt
- FIFO overflow
- FIFO underflow

When you enable these conditions, you specify the address of an attention AST routine. When the condition occurs, the attention AST executes.

The interrupt conditions are described in Section 3.7. When one of these conditions occurs, and the condition was previously enabled, data transfer continues, but the driver invokes execution of a special asynchronous system traps (AST), called attention AST's.

An AST is a routine which operates at a higher priority than your calling program and is used to handle specific conditions. AST's are used to handle expected and normal conditions such as buffer completion. See Section 4.6. Attention AST is a general name given to an AST which is designed to handle abnormal or unexpected conditions.

Note: There is a fifth hardware interrupt called the channel 0 EOP. This condition differs from the above interrupts. When a channel 0 EOP occurs, data transfer stops. Although the complete buffer was probably not transferred, the channel 0 EOP is considered a successful transfer and the buffer completion AST executes instead of an attention AST. The channel 0 EOP condition is always enabled. The channel 0 EOP is further described in Section 3.7.2.

The MicroVAX/DRQ3B device driver uses attention AST's to allow your program to react to any of the conditions listed above. You write the AST routine and pass its name to a SYS$QIO call with the set mode function code. This AST routine is then called whenever the enabled condition occurs. You can enable one or more of the conditions in any combination.

Within the AST routine, you write code which appropriately handles each type of interrupt. If you enable more than one attention AST condition, the first thing you might want to do is determine which type of condition occurred. This can be done using AST parameters. When the AST routine executes, the driver places an AST argument called the AST parameter on the argument stack. The AST parameter returned on this stack identifies the interrupt that occurred. The driver provides four constants, listed in Table 4–10, which can be used for comparison with the AST parameter. The AST parameter is returned at the address equal to four plus the contents of the argument pointer register (expressed as 4(AP) in MACRO).

All AST's place the AST parameter, R0, R1, the PC, and the PSL, on the argument pointer stack. This VMS convention is described in the *VAX/VMS System Services Manual*.

Since most of the interrupts indicate a situation where data has been lost or corrupted, you may wish to call SYS$CANCEL, which stops transfer of this buffer and dequeues it (as well as all pending buffers on this channel). However, the decision on how to handle the interrupt is up to you.

The interrupts are enabled by specifying the IO$_SETMODE function code with the IO$M_HX_ATTN_ENABLE function modifier.

The arguments to the IO$M_HX_ATTN_ENABLE specify the address of the AST routine, the address of the attention mask (whose contents specify the interrupt to be enabled), and the access mode to deliver the AST.

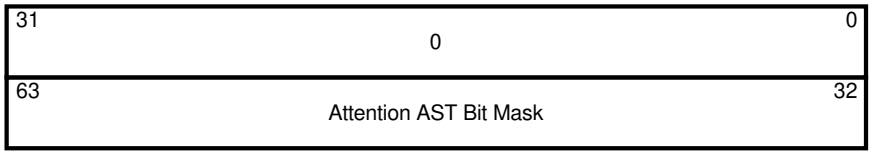
Table 4–8 summarizes the three arguments.

Table 4–8 Arguments for the IO$M_HX_ATTN_ENABLE Function Modifier

Argument	Passed By	Meaning
P1	Reference	The address of the AST routine.
P2	Reference	The address of a quadword attention mask. This mask is contained in your table and identifies the interrupt to be enabled.
P3	Value	The access mode to deliver the AST. This mode cannot be higher than the access mode of your program. An access mode of 3 specifies user mode. To specify a higher mode, your process must have privileges, assigned by the system manager.

The attention mask is illustrated in Figure 4–1. This mask is generally allocated as a table in the data section of your program. The driver provides bit masks which you can use when allocating the table to specify the interrupts to be enabled. Each bit mask sets a single bit in the attention mask. The attention mask is then specified as part of the set mode call.

Figure 4–1 Attention Mask Format



MR–4716–RA

The attention AST bit masks supported for the DRQ3B are shown in Table 4–9.

Table 4–9 Attention AST Bit Masks

Bit Mask	Meaning
HX$M_EXTERNAL	Enable the external interrupt.
HX$M_NXM	Enable the nonexistent memory interrupt.
HX$M_OVERRUN	Enable the port 0 FIFO overrun interrupt.
HX$M_UNDERRUN	Enable the port 1 FIFO underrun interrupt.

When an AST routine executes, the driver places an AST parameter on the argument stack. The parameter is stored at four plus the contents of the AP register. The driver provides the constants shown in Table 4–10 for use in identifying the interrupt which occurred.

Table 4–10 Attention AST Constants

Constants	Meaning
HX$K_EXTERNAL	The external interrupt occurred.
HX$K_NXM	The nonexistent memory interrupt occurred.
HX$K_OVERRUN	The port 0 FIFO overrun interrupt occurred.
HX$K_UNDERRUN	The port 1 FIFO underrun interrupt occurred.

4.3.2.1 Interrupt Setting Summary

To set the interrupts, then, you should create a table in your program, perhaps named INT_ENABLE. When allocating the table, use the bit masks to set the desired interrupts. For example, the MACRO code below sets the external interrupt and the nonexistent memory interrupt.

```
              .psect data
INT_ENABLE:   .long 0
              .long HX$M_EXTERNAL!-
                    HX$M_NXM
              .
              .
              .
              .psect code
              status = sys$qio(0,chan,IO$_SETMODE|IO$M_HX_ATTN_ENABLE,
                                    iosb,,,ast_addr,INT_ENABLE,3,,,)
```

The bit clear and bit set instructions can be used to alter the contents of the INT_ENABLE table during execution, if desired. Calling the now altered table allows you to enable or disable interrupts on the fly.

Note: The design of the attention AST mask and SYS$QIO call mirrors the structure used for out-of-band AST's in the VMS terminal driver. Further information on the terminal driver can be found in the *VAX/VMS I/O User's Reference Manual*.

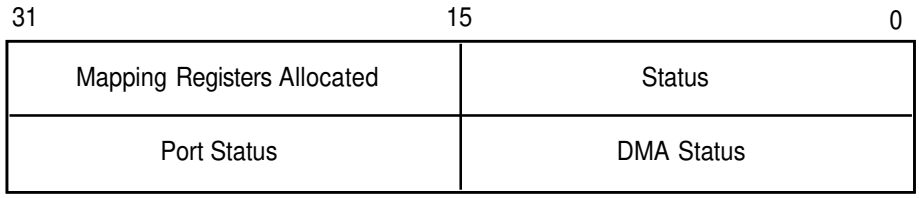
4.4 I/O Status Blocks

I/O status blocks (IOSB) are returned for all SYS$QIO calls. Each function code, read or write logical block, DMA initialization, or set/sense mode, returns different type of information in the IOSB. This section describes the information returned in each IOSB.

4.4.1 DMA Initialization I/O Status Block

The IO$_HX_DMA_INIT function code returns an I/O status block of two longwords. The DMA initialization IOSB is written immediately after the initialization function code call. It can be tested in program code that immediately follows the IO$_HX_DMA_INIT call. This IOSB is illustrated in Figure 4–2.

Figure 4–2 DMA Initialization I/O Status Block



The high word in the first longword contains the number of mapping registers allocated. At least two mapping registers are assigned even if a zero length buffer is specified.

The standard VMS I/O status codes returned in the low word of the first longword of the IOSB and in the SYS$QIO call are shown in Table 4–11.

Table 4–11 Status Codes Returned by IO$_HX_DMA_INIT

I/O Status Code	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	The initialization completed successfully.
SS$_ABORT	IOSB	An error occurred while trying to initialize this port. This may be due to a lack of mapping registers available at the time of the system service routine call.
SS$_IVBUFLEN	QIO	This status code is returned if the buffer size argument (P1 for logical I/O or P2 for physical I/O) exceeds 128K bytes. Buffer sizes are specified in bytes.

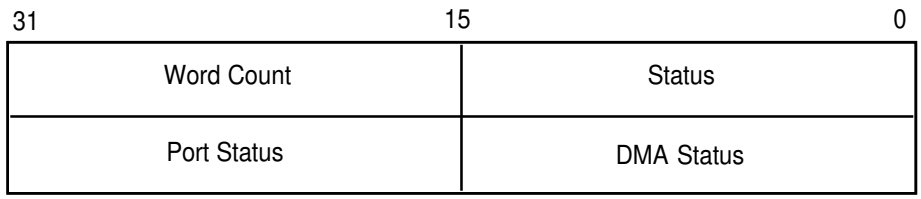
The second longword of the DMA initialization I/O status block contains the values of the DRQ3B's status register and DMA status register at the time the buffer completed transfer. These registers are described in Section 4.5. Note that the status register actually contains information about both the input and output port on the DRQ3B, while the DMA status register is specific to the port being used.

4.4.2 Read/Write I/O Status Block

The read and write block function codes for logical and physical I/O all return the same I/O status block. The read/write IOSB is written when a buffer completes transfer (or fails to transfer), not when the buffer is queued. Normally, it is tested or read by your buffer completion AST routine or read after the event flag for this buffer becomes set. This IOSB consists of two longwords and is illustrated in Figure 4–3.

Note: When doing asynchronous reads or writes, a unique I/O status block should be declared for each data buffer.

Figure 4–3 Read/Write I/O Status Block



The first longword of the I/O status block contains I/O status codes and the number of the words actually transferred.

The number of words (not bytes) actually transferred is returned in the high word of the first longword. If the maximum buffer size of 128K bytes was specified, the word count reads zero on successful completion. If a transfer terminates unsuccessfully, the word count contains the number of words actually transferred across the Q-bus, but does not include any data which may still be in the FIFO.

The standard VMS I/O status codes returned in the low word of the first longword of the IOSB and by the SYS$QIO call are shown in Table 4–12.

Table 4–12 Status Codes Returned by Read/Write Logical or Physical Block

I/O Status Code	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	The request completed successfully.
SS$_IVBUFLEN	QIO	The buffer size is greater than the maximum buffer size specified in IO$_HX_DMA_INIT.
SS$_NOLOG_IO	QIO	IO$_HX_DMA_INIT was not previously called for logical I/O.

Table 4–12 (Cont.) Status Codes Returned by Read/Write Logical or Physical Block

I/O Status Code	Returned by	Meaning
SS$_NOPHY_IO	QIO	IO$_HX_DMA_INIT was not previously called for physical I/O.
SS$_ABORT	IOSB	An error occurred while trying to perform the request. The most probable error is a nonexistent memory error.
SS$_CANCEL	IOSB	The SYS$CANCEL service routine was called while this buffer was being transferred. This status code is also returned if the calling programmed exited before this buffer completed transfer.
SS$_WASSET	IOSB	The transfer was terminated because the channel 0 EOP pin was asserted. The word count reflects the number of words which were actually input. The last word read is the word that was being transferred when the EOP occurred. This code occurs for channel 0 only.

The second longword of the read/write I/O status block contains the values of the DRQ3B's status register and DMA status register at the time the buffer completed transfer. These registers are described in Section 4.5. Note that the status register actually contains information about both the input and output port on the DRQ3B, while the DMA status register is specific to the port being used.

For transfers terminated by a channel 0 EOP interrupt the EOP bit (bit 1) of the DMA status register will not be set. This is due to the chaining method used by the driver to determine the transfer count at the time the EOP occurred.

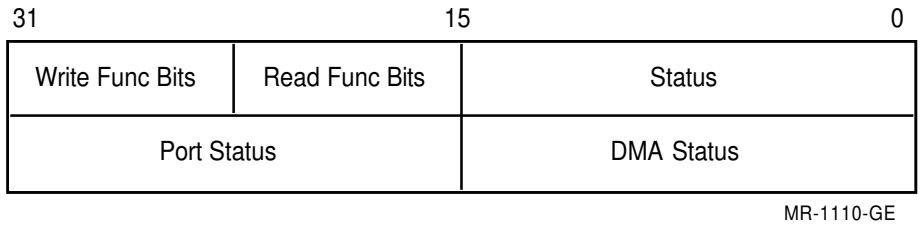
4.4.3 Set Mode/Sense Mode I/O Status Block

The I/O status block returned for the IO$_SETMODE and IO$_SENSEMODE function codes is the same. The IOSB returned for the IO$_SETMODE function code is the same regardless of which function modifier is specified. The status codes returned, however, are specific to the function modifiers, IO$M_HX_ATTN_ENABLE and IO$M_HX_FUNCT_BITS. The IOSB also returns the status of all of the function bits on both ports.

The set mode/sense mode IOSB is written as soon as the function code call executes and tested in program code immediately following the set mode or sense mode call.

Figure 4–4 shows the IOSB returned for the IO$_SETMODE function code.

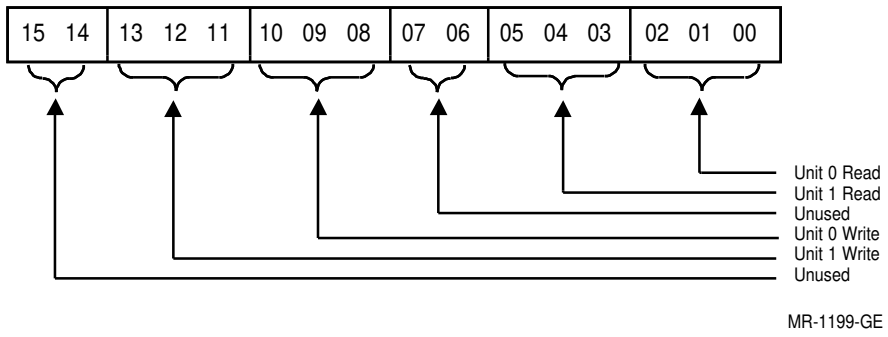
Figure 4–4 SET MODE/SENSE MODE IOSB



The first longword contains the I/O status code in the low word and the current setting of all of the DRQ3B's function bits in the high word. The second longword contains the current contents of the DRQ3B status register and DMA status register, as described in Section 4.5.

Figure 4–5 shows the contents of the high word in the SETMODE IOSB.

Figure 4–5 High Word of the First Longword in SETMODE IOSB



The correspondence between this longword and the actual pins on the DRQ3B connectors is shown in Table 4–13.

Table 4–13 Function Bit Status in SETMODE IOSB

Unit	Bit Number	Function Bit Pin Number
Unit 0 Read	00	FUNCT IN 0
	01	FUNCT IN 1
	02	FUNCT IN 2
Unit 1 Read	03	FUNCT IN 3
	04	FUNCT IN 4
	05	FUNCT IN 5

Table 4–13 (Cont.) Function Bit Status in SETMODE IOSB

Unit	Bit Number	Function Bit Pin Number
Unused	06	
Unused	07	
Unit 0 Write	08	FUNCT OUT 0
	09	FUNCT OUT 1
	10	FUNCT OUT 2
Unit 1 Write	11	FUNCT OUT 3
	12	FUNCT OUT 4
	13	FUNCT OUT 5
Unused	14	
Unused	15	

The connectors and the signals carried on each pin are illustrated in Figures B–1 and B–2.

When the IO$_SETMODE is called with the IO$M_HX_FUNCT_BITS function modifier, the standard VMS I/O status code returned in the low word of the first longword of the IOSB and in the SYS$QIO call is shown in Table 4–14.

Table 4–14 Status Code Returned by IO$M_HX_FUNCT_BITS

I/O Status Code	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	The request completed successfully.

When the IO$_SETMODE is called with the IO$M_HX_ATTN_ENABLE function modifier, there are six I/O status codes that can be returned in the low word of the first longword of the IOSB or by the SYS$QIO call. These are shown in Table 4–15.

Table 4–15 Status Codes Returned by IO$M_HX_ATTN_ENABLE

SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_BADPARAM	QIO	A bad attention AST condition was specified or the mask was specified in an invalid format. Make sure that the mask is in a quadword data item.

Table 4–15 (Cont.) Status Codes Returned by IO$M_HX_ATTN_ENABLE

Status Codes	Returned by	Meaning
SS$_ILLIOFUNC	QIO	More than one function modifier was specified.
SS$_ACCVIO	QIO	The address of the AST routine or specified bit mask is not accessible. Make sure you are passing the parameters correctly.
SS$_EXQUOTA	QIO	The buffered I/O quota or the AST quota was exceeded. Both of these quotas are set by your system manager.
SS$_INSUFMEM	QIO	The AST control block (created by VMS when an AST routine is specified) could not be created. This means that you have insufficient dynamic memory resources. This resource can also be raised by your system manager.

The second longword of the set mode/sense mode I/O status block contains the values of the DRQ3B's status register and DMA status register at the time the buffer completed transfer. These registers are described in Section 4.5. Note that the status register actually contains information about both the input and output port on the DRQ3B, while the DMA status register is specific to the port being used.

4.5 DRQ3B Status Registers

Two status registers are returned in the I/O status blocks used by this driver. These are the status register, which contains general information about the DRQ3B ports and FIFO's, and the DMA status register, which contains information about the DMA channel on the DRQ3B. There is only one status register on the DRQ3B and therefore it contains information about both ports, even though the driver is connected to only of the ports on the DRQ3B. The DMA status register is specific to the port to which the driver is connected.

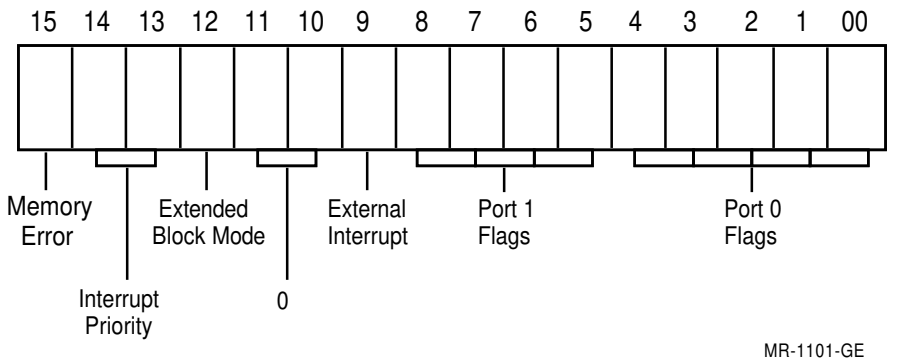
Both of these registers are described in the following sections.

4.5.1 Status Register

The status register is separate from the DMA status register.

Figure 4–6 shows the status register. Table 4–16 describes the bits in the status register.

Figure 4–6 Status Register



The interrupt flags in the status register must have previously been enabled in the port configuration register. In other words, if a FIFO full condition occurs, but the interrupt was not enabled in the port configuration register, the FIFO full interrupt status bit is not set.

Table 4–16 Status Register

Bit	Name	Description
0	Port 0 not empty	This bit is set if the port 0 FIFO has data in it.
1	Port 0 not full	This bit is set if the port 0 FIFO is not full.
2	Port 0 full interrupt	This bit is set when the port 0 FIFO becomes full and an interrupt is generated.
3	Port 0 overflow interrupt	This bit is set when the port 0 FIFO is full and the external device tries to write another data item into the FIFO; an interrupt is generated.
4	Port 0 not empty interrupt	This bit is set when the port 0 FIFO has data in it and an interrupt is generated.
5	Port 1 not empty	This bit is set if the port 1 FIFO has data in it.
6	Port 1 not full	This bit is set if the port 1 FIFO is not full.
7	Port 1 empty interrupt	This bit is set after port 1 transfers the last data item in the FIFO to the external device and an interrupt is generated.

Table 4–16 (Cont.) Status Register

Bit	Name	Description
8	Port 1 underflow interrupt	This bit is set when the external device asserts the strobe signal LOW, indicating it is ready to receive data, but there is no data word in the FIFO to be transferred; an interrupt is generated.
9	External interrupt	This bit is set when the external interrupt signal is asserted LOW by the external device.
10-11	Not used	These two bits always read zero.
12	Extended block mode	When this bit is clear it indicates that the extended block mode switch on switch pack 1 is set. This is the factory configuration.
13-14	Interrupt priority	These two bits together show the current setting in switch pack 1 for the interrupt priority level, as shown below.
	14-13	Interrupt Priority Level
	00	Interrupt level BR4
	01	Interrupt level BR5
	10	Interrupt level BR6
	11	Interrupt level BR7
15	Nonexistent memory	This bit is set if the DRQ3B DMA logic attempts to access nonexistent memory.

4.5.2 DMA Status Register

The DMA status register provides information about the DMA logic for the channel. Figure 4–7 shows the DMA status register. Table 4–17 defines the use of the bits in the DMA status register.

Note: Due to the nature of double buffered I/O operations, this register may contain the status of the current transfer when returned in the IOSB, rather than the status of the just-completed buffer.

Figure 4–7 DMA Status Register

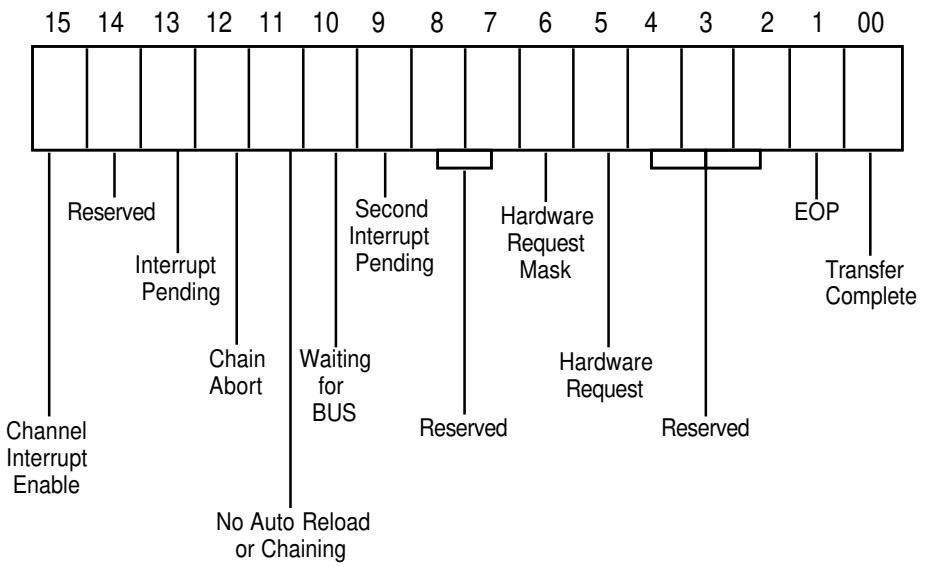


Table 4–17 DMA Status Register

Bit	Name	Description
0	Transfer complete	This bit is set when the operation count reaches zero, ending the DMA transfer.
1	EOP	This bit is set when an end-of-process occurs during a data transfer. An end-of-process occurs when the DMA machine cannot access the specified system memory address, or, on channel 0 only, the EOP pin input went low. If the nonexistent memory EOP occurred, the nonexistent memory bit (bit 15) in the status register is also set. The EOP bit is not set when an end-of-process occurs during chaining.
2-4	Reserved	Reserved for future use. No current meaning is attributed to these bits when using the driver and they may read zero or one.
5	Hardware request	This bit is dynamic and always reflects the current status of the port's hardware request line input. The hardware request line is used to synchronize data flow between the DMA logic and the FIFO. When the port 0 FIFO has data to be transferred to the DMA logic, it asserts this line to signal the DMA logic to transfer data to or from the FIFO.

Table 4–17 (Cont.) DMA Status Register

Bit	Name	Description
6	Hardware request mask	This bit reflects the status of the hardware request mask bit in the channel mode low register on the DRQ3B. The hardware request mask bit is not directly controllable using the DRQ3B driver, but the information is provided here for diagnostic purposes. This bit will generally read 0. When the hardware request mask bit is set, the automatic communication between the FIFO buffer and the DMA logic is disabled. The hardware request line status bit, bit 5, still reports the current status of the request line, however.
7-8	Reserved	Reserved for future use. No current meaning is attributed to these bits and they may read zero or one.
9	Second interrupt pending	This bit is set when a second interrupt occurs before the previous interrupt request has been serviced; that is, before the interrupt pending bit has been cleared. When this bit is set, and the interrupt pending bit is cleared, the second interrupt pending bit will immediately set the interrupt pending bit again, causing a second interrupt. The DRQ3B will not request the bus while this bit is set and no data will be transferred.
10	Waiting for bus	This bit is set when the DRQ3B has requested the bus, but has not yet been granted bus control.
11-12	Reserved	Reserved for future use. No current meaning is attributed to these bits and they may read zero or one.
13	Interrupt pending	This bit is set when an enabled DMA interrupt condition occurs. If the second interrupt pending flag is set when this bit is cleared, this bit is immediately reset, and the second interrupt pending flag is cleared automatically.
14	Reserved	Reserved for future use. No current meaning is attributed to this bit and it may read zero or one.
15	Channel interrupt enable	When this bit is set, an interrupt request is asserted on the bus whenever the interrupt pending bit becomes set. When this bit is cleared, the interrupt pending bit still becomes set if an interrupt condition occurs which has been enabled in the channel mode register. However, the DRQ3B will not assert an interrupt request. The channel interrupt bit is not directly controllable using the DRQ3B driver, but the information is provided here for diagnostic purposes. This bit will generally read 1.

Further information about these registers can be found in the *DRQ3B Parallel DMA I/O Module User's Guide*.

4.6 Buffer Completion

Most applications benefit from knowing when a buffer has completed transfer. As part of the SYS$QIO call, you can specify an asynchronous system trap (AST) routine or event flag. Either of these features can be used to notify your program when a buffer completes transfer. When the queued buffer completes, the specified AST routine executes, or the specified event flag is set. You can use these to notify your program that a buffer has completed.

4.6.1 AST Routines

Asynchronous system traps are routines written by you. The address of the routine is specified in the SYS$QIO call along with an AST parameter. When the buffer completes, the VMS operating system calls the AST routine. The AST parameter, R0, R1, the PC (program counter register), and the PSL (processor status longword), are all placed in an argument list pointed to by the argument pointer (AP) register.

Typically, the AST parameter is used to identify the buffer that terminated. For example, buffer 1 might be queued with an AST parameter of 1. Your AST routine can then examine the AST parameter (pointed to by the address 4(AP), or four plus the contents of AP). Since this value will be 1, you know that buffer 1 completed. If multiple buffers are queued with AST's and you are using the AST parameter to distinguish between buffers, the AST parameter must have a unique address for each value to be passed to the AST routine.

Complete information on AST's and the argument list created when an AST executes can be found in the *VAX/VMS System Services Manual*.

4.6.2 Event Flags

Event flags can also be used to identify buffer completion. The event flags are status posting bits maintained by VMS. The SYS$QIO format contains an optional argument for the event flag number, or *efn*. This flag bit becomes set when the queued buffer completes transfer. By testing the event flag, your program can determine when buffer completion occurs.

Event flags are fundamentally different from AST routines. Your program must explicitly test the event flag to determine whether the queued buffer has terminated. However, event flags have one advantage over AST's. Event flags can be used by more than one process (as long as the cooperating processes are in the same group). This provides a convenient way to synchronize two or more executing programs.

Event flags are numbered between 0 and 127. Flags 0-23 are always available to your local program. Flags 24-31 are reserved for system use. Flags 64-127 are common event flag clusters and are used when an event flag is shared between processes.

Typically, you might set a separate event flag for each buffer queued. By examining the specified event flags, you can determine which buffer completed. In addition, you can optionally specify that your program wait until a specified event flag is set. VMS also provides system services for clearing, setting, waiting for, and reading event flags.

Because of the extensive support for event flags under VMS, it is beyond the range of this document to fully describe their use. The *VAX/VMS System Services Manual* contains a complete chapter on event flags and their use.

4.6.3 SYS$QIOW

An alternative method for knowing when buffers complete transfer is to issue the SYS$QIOW system service routine instead of the SYS$QIO system service routine. The SYS$QIOW routine waits for the specified buffer to transfer before the next program line is executed. While this guarantees synchronization, it sacrifices many of the valuable features of the DRQ3B such as double buffering and simultaneous use of the ports.

It should be noted that the device driver does not have a timeout mechanism for use with SYS$QIOW. If an external device connected to the DRQ3B fails to respond to handshaking signals, the queued buffer will not be transferred. Since your program does not proceed until the buffer completes transfer, this situation would cause your program to hang.

The CTRL/Y key sequence can be used to abort the current transfer (providing that CTRL/Y has not been disabled).

4.7 Using Both Ports Simultaneously

Using both ports from a single program requires some extra thought. You need to be able to separately identify which port's buffers terminate. There are two ways to do this. One is to specify an asynchronous system trap (AST) routine as part of the SYS$QIO routine call. When a queued buffer terminates, the AST routine begins execution. By using AST parameters specific to the buffer, you can check to see which buffer terminated. You may wish to use separate AST's for each port; that is, have one AST which executes when an input buffer terminates, and one which executes when an output buffer terminates.

An alternative way to be notified when a buffer transfer is complete is to specify an event flag as part of the SYS$QIO call. The event flag becomes set when the queued buffer transfer terminates. By specifying separate event flags for buffers on each port, you can differentiate which buffers have been transferred. Both of these items, AST routines and event flags, are specified as arguments to the SYS$QIO system service routine call.

4.8 AST and DIO Limits

VMS provides a mechanism for setting limits for the number of AST's and direct I/O's (DIO) permitted for each process. These limits must be high enough for your process for proper operation of the driver.

The AST limit, called ASTLM, should be set to at least the number of buffers you are transferring, plus any other AST overhead used by your process. The DIO limit, called DIOLM, should be set to the same value as the ASTLM. If these limits are not high enough, the process may become hung and enter the RWAST state (resource wait, asynchronous system traps). Issuing the following command would result in an error being returned if your process exceeded its direct I/O or buffered I/O quota:

```
$  SET PROCESS/NORESOURCE_WAIT
```

Further information on these parameters and how to set them can be found in the *VAX/VMS System Services Manual*.

4.9 Timeouts

The DRQ3B device driver does not have a built in timeout feature. However, the SYS$CANCEL system service routine can be used to cancel current and pending I/O requests. It is possible, therefore, to construct a timeout routine of your own using the timing services provided under VMS in conjunction with the SYS$CANCEL service routine.

4.10 Canceling I/O

If it is necessary to cancel I/O at any time, the SYS$CANCEL system service routine can be used. This system service call takes as an argument the channel number assigned to the HX device by the SYS$ASSIGN routine. If a buffer was queued before the SYS$CANCEL system service routine was issued, the SS$_CANCEL status code is returned in the read/write IOSB.

All current and pending I/O request the specified channel will be canceled when SYS$CANCEL is called.

More information on the SYS$CANCEL system service routine can be found in the *VAX/VMS System Services Manual.*

5 Device Information

You can obtain information about the DRQ3B characteristics by using the get device system service routine, SYS$GETDVI. For complete information on this routine, see the *VAX/VMS System Services Manual*.

For the DRQ3B, the SYS$GETDVI system returns device independent characteristics only. The type of information returned by this system service routine is shown in Table 5–1.

Table 5–1 Device Information

Characteristic	Meaning
DEV$M_AVL	Device is online and available
DEV$M_IDV	Input device
DEV$M_ODV	Output device
DEV$M_RTM	Realtime device

A Example Program

A.1 Example Program

The following example program shows how a program issues SYS$QIO system service calls to initialize the DRQ3B and queue buffers for data input.

```
/*
        This is an example using the MicroVAX/DRQ3B device driver
        to transfer data in logical I/O mode over a loopback. The output
        buffer, buff_out, is transfered out of port 1, and read in through
        port 0, into the buffer named buff_in. An AST is used to indicate
        completion of the write SYS$QIO. This AST uses the iosb as an AST
        parameter, and indicates completion by printing a message to the
        terminal screen.

        The program does the following :

                1) assigns a VMS channel to the input port
                2) assigns a VMS channel to the output port
                3) initialize DMA on both ports
                4) issues an asynchronous write request to the output port,
                   with an AST to print a completion message at the terminal
                5) issues a synchronous read request from the input port
                6) prints a completion message at the terminal
*/

/*        Include Files  */

#include <hx$def>          /* DRQ3B specific defintions */
#include ssdef            /* system service definitions */
#include stsdef           /* VMS  status codes */
#include iodef            /* I/O definitions */
#include stdio            /* C I/O defintions */
#include descrip          /* character string definitions */

#define  BUFF_SIZE        2048    /* buff size in bytes */
main()
{
/*
        variable declarations
*/
int     status,                 /* for checking completion status */
        iosb[2],                /* for I/O completion */
        iosb1[2],               /* for I/O completion */
        in_chan,                /* input VMS channel */
        out_chan,               /* output VMS channel */
        i                       /* counter */
        ;
extern        int drq_ast();             /* declare the AST routine */
short int     buff_out[BUFF_SIZE/2],     /* buffer for output */
              buff_in[BUFF_SIZE/2]     /* buffer for input  */
                ;
static $DESCRIPTOR( dev_name_in, "HXA0:");      /* input port name */
static $DESCRIPTOR( dev_name_out, "HXA1:");     /* output port name*/

        /* assign a channel to input port */
        status = sys$assign( &dev_name_in, &in_chan, NULL, NULL);
        if(!(status&STS$M_SUCCESS)) lib$signal( status );

        /* assign a channel to output port */
        status = sys$assign( &dev_name_out, &out_chan, NULL, NULL);
        if(!(status&STS$M_SUCCESS)) lib$signal( status );
```

Example Program

```
/* initialize DMA on the input channel */
status = sys$qiow(      NULL,            /* no event flag */
                        in_chan,         /* vms channel  */
                        IO$_HX_DMA_INIT, /* INIIALIZE DMA */
                        iosb,            /* io status */
                        NULL, NULL,      /* no ast */
                        BUFF_SIZE,       /* max buffer size */
                        NULL, NULL,NULL, NULL, NULL); /* skip p2-6*/
if(!(status&STS$M_SUCCESS)) lib$signal( status ); /* check QIO status*/
if(!(iosb[0]&STS$M_SUCCESS)) lib$signal( iosb[0] );/* check I/O status*/

/* initialize DMA on the output channel */
status = sys$qiow(      NULL,             /* no event flag */
                        out_chan,         /* vms channel  */
                        IO$_HX_DMA_INIT,  /* INIIALIZE DMA */
                        iosb,             /* io status */
                        NULL, NULL,       /* no ast */
                        BUFF_SIZE,        /* max buffer size */
                        NULL, NULL,NULL, NULL, NULL); /* skip p2-6*/
if(!(status&STS$M_SUCCESS)) lib$signal( status ); /* check QIO status*/
if(!(iosb[0]&STS$M_SUCCESS)) lib$signal( iosb[0] );/* check I/O status*/

/* fill output buffer with data */
for (i=0;i<BUFF_SIZE/2;i++) buff_out[i] = i;

/* issue an asynchronous output request with an AST */
status = sys$qio(       NULL,           /* no event flag */
                        out_chan,       /* vms channel */
                        IO$_WRITELBLK,  /* write */
                        iosb,           /* I/O status */
                        drq_ast,        /* AST address */
                        &iosb[0],       /* pass iosb addres as parameter */
                        buff_out,       /* buff address */
                        BUFF_SIZE,      /* size in bytes */
                        NULL, NULL, NULL, NULL); /* p3-6 */

if(!(status&STS$M_SUCCESS)) lib$signal( status ); /* check QIO status*/

/* issue a synchronous read request */
status = sys$qiow(      NULL,           /* no event flag */
                        in_chan,        /* vms channel */
                        IO$_READLBLK|IO$M_HX_FIFO_CLEAR,  /* read */
                        iosb1,          /* I/O status */
                        NULL,           /* no AST address */
                        NULL,           /* no ast parametr */
                        buff_in,        /* buff address */
                        BUFF_SIZE,      /* size in bytes */
                        NULL, NULL, NULL, NULL); /* p3-6 */

if(!(status&STS$M_SUCCESS)) lib$signal( status ); /* check QIO status*/
if(!(iosb1[0]&STS$M_SUCCESS)) lib$signal( iosb1[0] );/* check I/O status*/

printf("\n Read request completed in main program");

/* free the channels for vms */
status = sys$dassgn(in_chan);
if(!(status&STS$M_SUCCESS)) lib$signal( status );

status = sys$dassgn(out_chan);
if(!(status&STS$M_SUCCESS)) lib$signal( status );
```

```
}
int drq_ast( iosb )      /* This is the AST routine */
int iosb[];              /* IO status block is the parameter */
{
        if (iosb[0]&STS$M_SUCCESS)
                {
                printf("\n Output Request was successful. ");
                }
        else
                {
                printf("\n Output Request failed. ");
                lib$signal( iosb[0] );
                }
}
```

The example program above was compiled and linked using the following commands:

```
$ cc hx$example
$ link hx$example, sys$input/opt
  sys$library:vaxcrtl.exe/share
```

B DRQ3B Connectors

B.1 DRQ3B Connectors

Figures B–1 and B–2 show the signals carried on the pins available on the DRQ3B input and output connectors. These figures apply both to the connectors actually on the board and to the connectors in the CK-DRQ3B connector kits.

Figure B–1 Input Pins on J2 of the DRQ3B Connector Kits

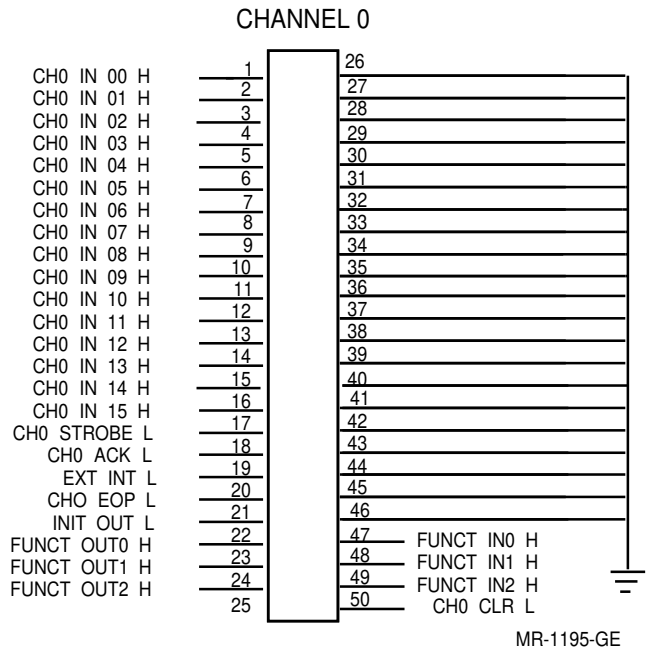
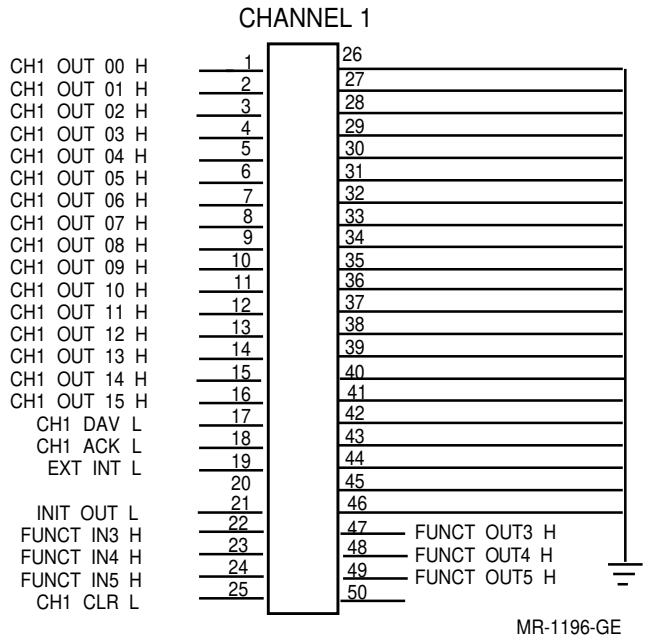


Figure B–2 Output Pins on J1 of the DRQ3B Connector Kits



For further information, please refer to the *DRQ3B Parallel DMA I/O Module User's Guide*.

Index

A

AST • 1–2, 3–11
Asynchronous system trap • 1–2, 3–11, 4–22, 4–23
  ASTLM • 4–23 to 4–24
Attention AST's
  enabling and disabling • 4–8 to 4–11
  interrupts • 3–11 to 3–14

C

Canceling I/O • 4–24
Chain loading
  abort during • 4–20

D

Direct memory access • 1–1
DMA • 1–1, 3–1 to 3–2
DMA modes
  block • 1–1, 3–1 to 3–2
  burst • 1–1, 3–1 to 3–2
  extended block mode • 3–2
DMA status register • 4–18, 4–19 to 4–21

E

End-of-process • 3–2, 4–20
Event flags • 1–2, 4–22 to 4–23

F

Factory configuration • 4–19
FIFO buffer • 1–1
Function bits • 1–1

H

Handshaking signals • 1–1

I

Interrupt priority level • 1–1, 4–19
Interrupts
  FIFO • 4–18 to 4–21

L

Limits
  ASTLM • 4–23 to 4–24
  DIOLM • 4–23 to 4–24

M

Match condition • 3–2

P

Port configuration register • 4–18

S

Status register • 4–18 to 4–19
Switch pack 1
  extended block mode • 3–2, 4–19

T

Timeout • 4–23, 4–24
Transfer complete • 4–20
Transfer rates • 1–1, 3–2