

VMS

digital

VMS Version 5.3 Small Computer System
Interface (SCSI) Device Support Manual

Order Number: AA-PAJ2A-TE

VMS Version 5.3 Small Computer System Interface (SCSI) Device Support Manual

Order Number: AA-PAJ2A-TE

October 1989

This manual describes the mechanisms the VMS operating system provides that allow a non-Digital-supplied SCSI device to be attached to a VAXstation or MicroVAX system.

Revision/Update Information: This is a new manual.

Software Version: VMS Version 5.3

**digital equipment corporation
maynard, massachusetts**

October 1989

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.

© Digital Equipment Corporation 1989.

All Rights Reserved.
Printed in U.S.A.

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDA	LiveLink	VAXcluster
DDIF	LN03	VAX RMS
DEC	MASSBUS	VAXserver
DECnet	MicroVAX	VAXstation
DECUS	PrintServer 40	VMS
DECwindows	Q-bus	VT
DECwriter	ReGIS	XUI
DEQNA	ULTRIX	
DIGITAL	UNIBUS	
GIGI	VAX	

PostScript is a registered trademark of Adobe Systems, Inc.

ZK5369

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by Digital. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use Digital-supported devices, such as the LN03 laser printer and PostScript printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

Contents

PREFACE		xi
---------	--	----

CHAPTER 1	INTRODUCTION	1-1
-----------	--------------	-----

1.1	VMS SCSI CLASS/PORT ARCHITECTURE	1-1
1.2	VMS SCSI THIRD-PARTY DEVICE SUPPORT MECHANISMS	1-3
1.3	HARDWARE CONSIDERATIONS	1-4
1.3.1	Connecting a Non-Digital-Supplied SCSI Disk or Tape Drive to a VMS SCSI Port _____	1-8

CHAPTER 2	USING THE VMS GENERIC SCSI CLASS DRIVER	2-1
-----------	-----------------------------------------	-----

2.1	OVERVIEW OF THE VMS GENERIC SCSI CLASS DRIVER	2-1
2.2	ACCESSING THE VMS GENERIC SCSI CLASS DRIVER	2-2
2.3	SCSI PORT FEATURES UNDER APPLICATION CONTROL	2-4
2.3.1	Setting the Data Transfer Mode _____	2-4
2.3.2	Enabling Disconnection and Reselection _____	2-5
2.3.3	Disabling Command Retry _____	2-5
2.3.4	Setting Command Timeouts _____	2-6
2.4	CONFIGURING A DEVICE USING THE GENERIC CLASS DRIVER	2-6
2.4.1	Disabling the Autoconfiguration of a SCSI Device _____	2-7
2.5	ASSIGNING A CHANNEL TO GKDRIVER	2-8
2.6	ISSUING A \$QIO REQUEST TO THE GENERIC CLASS DRIVER	2-8

Contents

2.7	OBTAINING DEVICE INFORMATION	2-12
2.8	PROGRAMMING EXAMPLE	2-13
CHAPTER 3 WRITING A VMS SCSI CLASS DRIVER		3-1
3.1	SCSI CLASS/PORT ARCHITECTURE	3-1
3.1.1	SCSI Port Interface	3-4
3.1.2	SCSI-Specific Data Structures	3-6
3.1.3	SCSI Template Class Driver	3-8
3.2	CONNECTING TO A SCSI DEVICE	3-8
3.3	SETTING UP A SCSI COMMAND	3-9
3.3.1	Preparing a SCSI Command Descriptor Block	3-9
3.3.2	Setting Command Timeouts	3-10
3.3.3	Disabling Command Retry	3-11
3.4	PERFORMING A SCSI DATA TRANSFER	3-12
3.4.1	Setting the Data Transfer Mode	3-12
3.4.2	Enabling Disconnection and Reselection	3-13
3.4.3	Determining the Maximum Data Transfer Size	3-13
3.4.4	Initializing the SCDRP to Reflect Class Driver Data Buffering Mechanisms	3-14
3.4.5	Making a Class Driver Data Buffer Accessible to the Port	3-15
3.4.6	Examining Port and SCSI Status	3-16
3.4.6.1	Examining Port Status •	3-16
3.4.6.2	Examining the SCSI Status Byte •	3-17
3.4.6.3	Testing for Incomplete Transfers •	3-18
3.5	OTHER SCSI CLASS DRIVER ISSUES	3-18
3.5.1	Preserving Local Context	3-18
3.5.2	Error Logging	3-19
3.6	FLOW OF A READ I/O REQUEST THROUGH THE SCSI CLASS AND PORT DRIVERS	3-21

3.7	COMPONENTS OF A SCSI CLASS DRIVER	3-23
3.7.1	Data Definitions _____	3-23
3.7.2	Driver Prologue Table _____	3-24
3.7.3	Driver Dispatch Table _____	3-24
3.7.4	Function Decision Table and FDT Routines _____	3-24
3.7.5	Controller Initialization Routine _____	3-24
3.7.6	Unit Initialization Routine _____	3-25
3.7.7	Start-I/O Routine _____	3-26
3.7.8	Cancel-I/O Routine _____	3-27
3.7.9	Register Dumping Routine _____	3-27
<hr/>		
3.8	SERVICING ASYNCHRONOUS EVENTS FROM A SCSI DEVICE	3-27
<hr/>		
3.9	CONFIGURING A SCSI THIRD-PARTY DEVICE	3-29
3.9.1	Disabling the Autoconfiguration of a SCSI Device _____	3-30
<hr/>		
3.10	DEBUGGING A SCSI CLASS DRIVER	3-30
3.10.1	Selecting a SCSI Bus Analyzer _____	3-31
<hr/>		
APPENDIX A	SCSI DEVICE DRIVER DATA STRUCTURES	A-1
<hr/>		
A.1	SCSI CLASS DRIVER REQUEST PACKET (SCDRP)	A-1
<hr/>		
A.2	SCSI CONNECTION DESCRIPTOR TABLE (SCDT)	A-9
<hr/>		
A.3	SCSI PORT DESCRIPTOR TABLE (SPDT)	A-15
<hr/>		
APPENDIX B	VMS MACROS INVOKED BY SCSI CLASS DRIVERS	B-1
<hr/>		
B.1	STANDARD SCSI PORT INTERFACE MACROS	B-1
	SPI\$ABORT_COMMAND	B-2
	SPI\$ALLOCATE_COMMAND_BUFFER	B-3
	SPI\$CONNECT	B-4
	SPI\$DEALLOCATE_COMMAND_BUFFER	B-6
	SPI\$DISCONNECT	B-7
	SPI\$GET_CONNECTION_CHAR	B-8
	SPI\$MAP_BUFFER	B-10

Contents

	SPI\$RESET	B-13
	SPI\$SEND_COMMAND	B-14
	SPI\$SET_CONNECTION_CHAR	B-17
	SPI\$UNMAP_BUFFER	B-19
<hr/>		
B.2	SCSI PORT INTERFACE EXTENSION MACROS FOR ASYNCHRONOUS EVENT NOTIFICATION	B-20
	SPI\$FINISH_COMMAND	B-21
	SPI\$RECEIVE_BYTES	B-22
	SPI\$RELEASE_BUS	B-23
	SPI\$SEND_BYTES	B-24
	SPI\$SENSE_PHASE	B-25
	SPI\$SET_PHASE	B-26
<hr/>		
	APPENDIX C VMS TEMPLATE SCSI CLASS DRIVER	C-1
<hr/>		
	APPENDIX D INTERPRETING SCSI DRIVER ERROR LOG ENTRIES	D-1
<hr/>		
D.1	SCSI PORT DRIVER ERROR LOG ENTRIES	D-1
<hr/>		
D.2	SCSI CLASS DRIVER ERROR LOG ENTRIES	D-5
<hr/>		
D.3	RESOLVING SCSI CLASS DRIVER PROBLEMS USING ERROR LOGS	D-6
<hr/>		
	APPENDIX E VMS REQUIREMENTS AND RESTRICTIONS FOR NON-DIGITAL-SUPPLIED SCSI DEVICES	E-1
<hr/>		
E.1	VMS REQUIREMENTS	E-1
E.1.1	Conformance to Standards _____	E-1
E.1.2	Cabling _____	E-2
E.1.3	Connector Requirements _____	E-2
E.1.4	SCSI Bus Termination _____	E-2
E.1.5	Terminator Power _____	E-2
E.1.6	Dynamic Reconfiguration of Devices _____	E-2
E.1.7	External Boxes _____	E-3
E.1.8	Device Behavior Following Power-On _____	E-3
E.1.9	Device Behavior Following Bus Reset _____	E-3

E.1.10	Data Transfer _____	E-4
E.1.11	Initiator/Target Operation _____	E-5
E.1.12	SCSI IDs and Logical Unit Numbers _____	E-5
E.1.13	Bus Phases _____	E-6
E.1.14	Disconnect and Reselection _____	E-6
E.1.15	Messages _____	E-7
E.1.16	Commands _____	E-8
E.1.17	INQUIRY Command _____	E-8
E.1.18	Status _____	E-9
E.1.19	Unsupported Features _____	E-9

GLOSSARY

Glossary-1

INDEX

EXAMPLES

D-1	SCSI Bus Phase Error Port Driver Error Log Entry _____	D-7
D-2	SCSI Bus Reset Port Driver Error Log Entry _____	D-8
D-3	SCSI Bus Reset Class Driver Error Log Entry _____	D-9

FIGURES

1-1	VMS SCSI Class/Port Interface _____	1-2
1-2	MicroVAX/VAXstation 3100 System Configuration _____	1-5
1-3	VAXstation 3520/3540 System Configuration _____	1-6
1-4	SCSI Bus Configuration _____	1-7
2-1	Generic SCSI Class Driver Flow _____	2-3
2-2	VMDS2 System Parameter _____	2-8
3-1	VMS SCSI Class/Port Interface _____	3-2
3-2	VMS SCSI Port Driver Configuration _____	3-3
3-3	VMS SCSI Class Driver Configuration _____	3-4
3-4	SCSI Driver Data Structures _____	3-7
3-5	VMDS2 System Parameter _____	3-30
A-1	SCSI Class Driver Request Packet (SCDRP) _____	A-1
A-2	SCSI Connection Descriptor Table (SCDT) _____	A-10
A-3	SCSI Port Descriptor Table (SPDT) _____	A-15

Contents

B-1	SCSI Bus Phase Longword Returned to SPI\$SENSE_PHASE _____	B-25
B-2	SCSI Bus Phase Longword Supplied to SPI\$SET_PHASE ____	B-26

TABLES

3-1	SCSI Port Interface (SPI) Macros _____	3-5
3-2	Data Structures _____	3-7
3-3	SCSI Status Byte Format _____	3-17
3-4	Error Message Buffer Extension for SCSI Class Drivers ____	3-19
3-5	SPI Extension Macros Supporting Asynchronous Event Notification _____	3-28
A-1	Contents of SCSI Class Driver Request Packet _____	A-4
A-2	Contents of SCSI Connection Descriptor Table _____	A-11
A-3	Contents of SCSI Port Descriptor Table _____	A-18
D-1	Key to Port Driver Error Log Entries _____	D-2
D-2	Key to Class Driver Error Log Entries _____	D-5

Preface

The American National Standard for information systems—Small Computer System Interface–2 (SCSI–2) specification defines mechanical, electrical and functional requirements for connecting small computers to a wide variety of intelligent devices, such as rigid disks, flexible disks, magnetic tape devices, printers, optical disks, and scanners. It specifies standard electrical bus signals, timing, and protocol, as well as a standard packet interface for sending commands to devices on the SCSI bus.

Certain VAXstation and MicroVAX systems employ the SCSI bus as an I/O bus. For these systems, Digital offers SCSI-compliant disk and tape drives, such as the RZ55 300MB read/write disk, the RRD40 600MB compact disk, and the TZK50 95MB streaming tape drive. The VMS operating system also allows non-Digital-supplied devices including disk drives, tape drives, and scanners to be connected to the SCSI bus of such a system. This manual describes the VMS software interfaces that must be used to establish this connection and control the device's operation within the VMS operating system.

SCSI has been widely adopted by manufacturers for a variety of peripheral devices. However, because the ANSI SCSI standard is broad in scope, not all devices that implement its specifications can fully interrelate on the bus. Digital fully supports SCSI-compliant equipment sold or supplied by Digital. Proper operation of products not sold or supplied by Digital cannot be assured.

Digital intends to maintain the interfaces described in this manual, although some unavoidable changes may occur as new features are added. The use of internal executive interfaces other than those described in this manual is discouraged.

Intended Audience

Programmers responsible for supporting non-Digital-supplied SCSI devices on MicroVAX/VAXstation systems require the information presented in this manual. They should be familiar with the VMS operating system and with the ANSI SCSI specification.

Programmers of a high-level application interface to SCSI devices on MicroVAX/VAXstation systems should understand how to use the Queue I/O Request (\$QIO) system service to initiate I/O operations and how to manage device status return values. Programmers of a SCSI device driver must be fluent in VAX MACRO and have some experience writing a VMS device driver.

Document Structure

Chapter 1 introduces some general SCSI concepts and defines those terms that are used in discussions in subsequent chapters. It presents an overview of MicroVAX/VAXstation system configurations that incorporate the SCSI bus, summarizing and contrasting the mechanisms by which a third-party SCSI device may be connected to these systems.

Chapter 2 describes the features and capabilities of the VMS generic SCSI class driver. It discusses the means by which a programmer can support a non-Digital-supplied SCSI device by writing an application that uses the generic SCSI class driver interface.

Chapter 3 provides information on creating a third-party SCSI class driver to support a non-Digital-supplied SCSI device. It describes the components of the VMS SCSI class/port interface and follows the flow of an I/O operation through the class and port drivers. It introduces the SCSI port interface (SPI) functions, SCSI-specific VMS data structures, and the VMS template SCSI class driver—all tools that aid in the development of a device-specific class driver. It describes the actions of the components of such a driver, as well as the means by which the driver may be configured and debugged. It concludes with a description of the optional asynchronous event notification feature.

Appendix A contains a set of figures and tables that describe each data structure referenced by SCSI class and port drivers.

Appendix B describes the macros that compose the SCSI port interface and the extensions to the SCSI port interface for asynchronous event notification.

Appendix C contains a listing of the VMS template SCSI class driver.

Appendix D provides a guide to reading SCSI class and port driver error log entries.

Appendix E describes the requirements and restrictions that are necessary for a non-Digital-supplied SCSI device to be connected to the SCSI bus of a MicroVAX or VAXstation system.

The glossary at the end of this manual defines the vocabulary that pertains to SCSI device drivers and their environment.

Associated Documents

Before reading the *VMS Version 5.3 Small Computer System Interface (SCSI) Device Support Manual*, you should have an understanding of the material discussed in the following documents:

- *VMS Device Support Manual*, which describes the components of a VMS device driver and the basic rules to which non-Digital-supplied device drivers must adhere
- American National Standard for Information Systems—Small Computer System Interface—2 (SCSI-2) specification (X3T9.2/86-109)

The SCSI-2 specification is a draft of a proposed standard. Until it is finally approved, copies of this document may be purchased from: Global Engineering Documents, 2805 McGaw, Irvine, California 92714, United States; or (800) 854-7179 or (714) 261-1455. Please refer to document X3.131-198X.

- American National Standard for Information Systems—Small Computer System Interface specification (X3.131-1986)

Copies of this document may be obtained from: American National Standards Institute, Inc., 1430 Broadway, New York, New York, 10018. This document is now known as the SCSI-1 standard.

Digital publishes two additional documents to help third-party vendors prepare SCSI peripherals and peripheral software for use with Digital's workstations and MicroVAX systems.

- *The Small Computer System Interface: An Overview* (EK-SCSISOV-001) provides a general description of Digital's SCSI third-party support program.
- *The Small Computer System Interface: A Developer's Guide* (EK-SCSIS-SP-001) presents the details of Digital's implementation of SCSI within its operating systems.

You may need to refer to material in the following manuals for help in certain aspects of application and driver programming:

- *VMS System Services Reference Manual* for a description of the high-level language interface to the I/O subsystem of the VMS operating system
- *VMS System Dump Analyzer Utility Manual* for assistance in investigating system failures
- *VMS Delta / XDelta Utility Manual* for information on debugging device driver code

Conventions

This manual describes code transfer operations in three ways:

- 1 The phrase "issues a system service call" implies the use of a CALL instruction.
- 2 The phrase "calls a routine" implies the use of a JSB or BSB instruction.
- 3 The phrase "transfers control to" implies the use of a BRB, BRW, or JMP instruction.

Typographical conventions used in this book include the following:

- Generally, terms that are further explained in the glossary of this manual first appear in italic print. For example:

Under the VMS operating system, a *device driver* is a set of routines and tables that the system uses to process an I/O request for a particular device type.

Preface

- Terms that serve as arguments to macros appear in boldface in the text of the manual. For example:

If an at-sign character (@) precedes the **oper** argument, then the **exp** argument describes the address of the data with which to initialize the field.

- In examples, a symbol with a 1- to 6-character abbreviation indicates that you press a key on the terminal. For instance:

```
driver-base-address,0;X Ret
```

- In examples, the symbol **Ctrl/x** indicates that you must press the key labeled Ctrl while you simultaneously press another key. For instance:

```
$ CREATE MYDRIVER.OPT
BASE=0
CTRL/Z
```

- A horizontal ellipsis indicates that additional parameters, values, or information can be entered. For example:

```
$ LINK /NOTRACE MYDRIVER1[,MYDRIVER2,...],-
MYDRIVER.OPT/OPTIONS,-
SYS$SYSTEM:SYS.STB/SELECTIVE_SEARCH
```

- Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)

```
DSBINT [ip] [dst]
```

- In interactive examples in printed editions of this book, all output lines or prompting characters that the system prints or displays appear in black letters. All user-entered commands are shown in red letters. In online editions of this book, all user-entered commands are shown in boldface type.

For example:

```
>>> DEPOSIT R3 0
>>> @DMAXDT
SYSBOOT>
SYSBOOT> CONTINUE
```

- A vertical ellipsis means either that not all the data that the system would display in response to the particular command is shown or that not all the data a user would enter is shown. For example:

```
JSB      @UCB$L_FPC(R5)          ; Restore the driver process.
.
.
.
;Between these instructions, the interrupt service routine
;can make no assumptions about the contents of R0 through R4.
.
.
.
POPR     #^M<R0,R1,R2,R3,R4,R5> ; Restore interrupt registers.
```

1

Introduction

The Small Computer System Interface (SCSI) provides a standard means by which small computers and intelligent peripherals may be interconnected.

The VMS operating system offers a native mode implementation of the ANSI SCSI bus on its MicroVAX/VAXstation 3100 and VAXstation 3520/3540 system configurations. Although this implementation is currently based on the SCSI-1 standard, the SCSI-1 standard is upwardly compatible with SCSI-2, the SCSI-2 standard clarifying many of the details specified in the SCSI-1 standard. Any non-Digital-supplied device to be attached to the SCSI bus of a MicroVAX/VAXstation system must implement all mandatory features of the SCSI-2 standard as described in the specification. The device is permitted to implement any optional features, as long as they are implemented according to the SCSI-2 standard. The device may implement vendor-unique features, as long as they are implemented in areas clearly designated as such by the standard.

The ANSI SCSI specification is, in places, very broad and flexible. In some cases, it is possible for a SCSI device to conform to the specification, but be unsupported by the VMS operating system. For instance, it is possible that a SCSI device may implement a maximum timeout value that is incompatible with a value required by the VMS operating system. The requirements and restrictions adopted by the VMS operating system in support of SCSI devices appear in Appendix E. Consult this appendix prior to attaching a non-Digital-supplied device to a VMS system.

1.1 VMS SCSI Class/Port Architecture

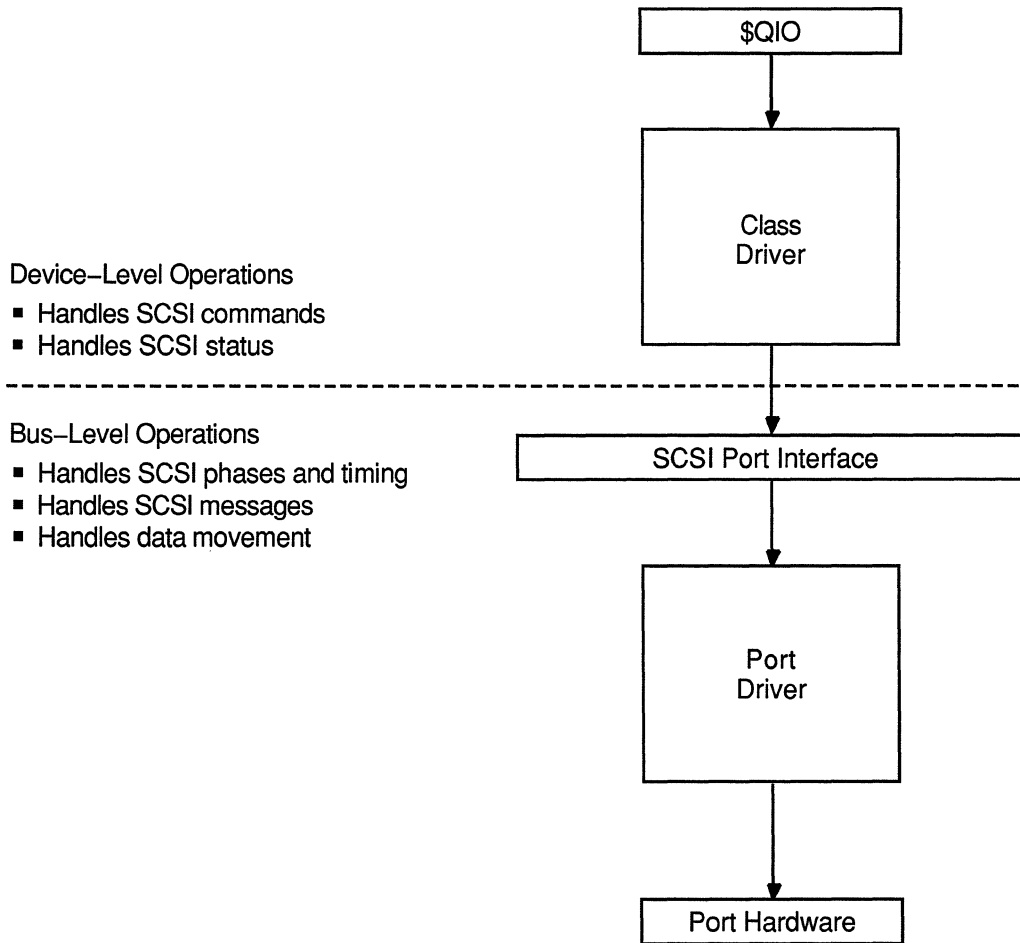
The VMS operating system employs a class/port driver architecture to communicate with devices on the SCSI bus. The class/port design allows the responsibilities for communication between the operating system and the device to be cleanly divided between two separate driver modules (see Figure 1-1).

The *SCSI port driver* transmits and receives SCSI commands and data. It knows the details of transmitting data from the local processor's SCSI port hardware across the SCSI bus. Although it understands SCSI bus phases, protocol, and timing, it has no knowledge of which SCSI commands the device supports, what status messages it returns, or the format of the packets in which this information is delivered. Strictly speaking, the port driver is a communications path. When directed by a SCSI class driver, the port driver forwards commands and data from the class driver onto the SCSI bus to the device. On any given MicroVAX/VAXstation system, a single SCSI port driver handles bus-level communications for all SCSI class drivers that may exist on the system.

Introduction

1.1 VMS SCSI Class/Port Architecture

Figure 1-1 VMS SCSI Class/Port Interface



ZK-1366A-GE

The *SCSI class driver* acts as an interface between the user and the SCSI port, translating an I/O function as specified in a user's \$QIO request to a SCSI command targeted to a device on the SCSI bus. Although the class driver knows about SCSI command descriptor buffers, status codes, and data, it has no knowledge of underlying bus protocols or hardware, command transmission, bus phases, timing, or messages. A single class driver can run on any given MicroVAX/VAXstation system, in conjunction with the SCSI port driver that supports that system. The VMS operating system supplies a standard SCSI disk class driver and a standard SCSI tape class driver to support its disk and tape SCSI devices.

1.2 VMS SCSI Third-Party Device Support Mechanisms

The VMS operating system provides the following three mechanisms to allow a non-Digital-supplied SCSI device to be attached to a MicroVAX/VAXstation system. The implementor of support for a non-Digital-supplied SCSI device can select the most appropriate method, based on the capabilities of the device, the needs of its end users, and available programming resources.

- A SCSI disk or tape drive may function properly using the standard VMS SCSI disk or tape class driver and the VMS SCSI port driver, given the restrictions and cautions presented in Section 1.3.1.
- An application program can send commands to, receive status from, and exchange data with a device on the SCSI bus by using the VMS generic SCSI class driver. The VMS operating system defines a special Queue I/O Request (\$QIO) system service interface that allows an application to pass SCSI command packets to the device through the generic SCSI class driver and the VMS SCSI port driver.
- A third-party SCSI class driver, in conjunction with the VMS SCSI port driver, can supply the level of support most closely tailored to the capabilities of the device. By writing a SCSI class driver, a system programmer can implement device-specific error handling and a simple, robust \$QIO interface.

Because the VMS operating system provides a special set of macros that initialize the SCSI port and transfer commands and data to a SCSI device, the programmer of a SCSI class driver can focus on coding details related to device capabilities. The VMS operating system further facilitates the writing of a SCSI class driver by including the online sources of a template SCSI class driver.

When selecting between writing an application program that uses the VMS generic SCSI class driver and writing a third-party SCSI class driver, the implementor of SCSI device support should consider the following factors:

- Because the VMS generic SCSI class driver provides access to the SCSI device from application code, the programmer of an application that uses it must be familiar with a high-level language and have some I/O programming skill. Because VMS device drivers cannot be written in a high-level language, the programmer of a third-party SCSI class driver must have a thorough understanding of VAX MACRO and VMS driver internals.
- The VMS generic SCSI class driver uses a fixed \$QIO interface to the SCSI port, requiring an application to pass a SCSI command descriptor block to the device by means of a single I/O function, IO\$_DIAGNOSE. By contrast, a SCSI class driver can define a unique \$QIO interface that conceals the details of SCSI command format from application programs.

Introduction

1.2 VMS SCSI Third-Party Device Support Mechanisms

- A programmer typically can develop an interface to a SCSI device more quickly by using the VMS generic SCSI class driver than by developing a third-party SCSI class driver. Because device drivers are tightly integrated into the VMS operating system and run in a privileged mode at high IPL, coding errors can result in system crashes. Because the VMS generic SCSI class driver is an established system interface, a programmer using it can spend less time integrating the code into the operating system and more time working on the interface.
- A third-party SCSI class driver can write entries to an error log buffer, thus allowing the programmer to use the VMS Error Log Utility as a debugging aid.
- A third-party SCSI class driver can implement error recovery mechanisms that are closely associated with the abilities of the device. It can service a device error within the context of the single \$QIO request that initiated the transaction to the device.

Because the generic SCSI class driver has no knowledge of specific device errors, an application using that driver must manage device-specific errors itself. To service an error returned on a single transaction, the application must issue additional \$QIO requests and initiate further transactions to the device.

- The SCSI asynchronous event notification (AEN) protocol is available only to third-party SCSI class drivers (see Section 3.8).

For information on how to program to the VMS generic SCSI class driver's \$QIO interface, see Chapter 2. Chapter 3 describes the means by which you can write a third-party SCSI class driver.

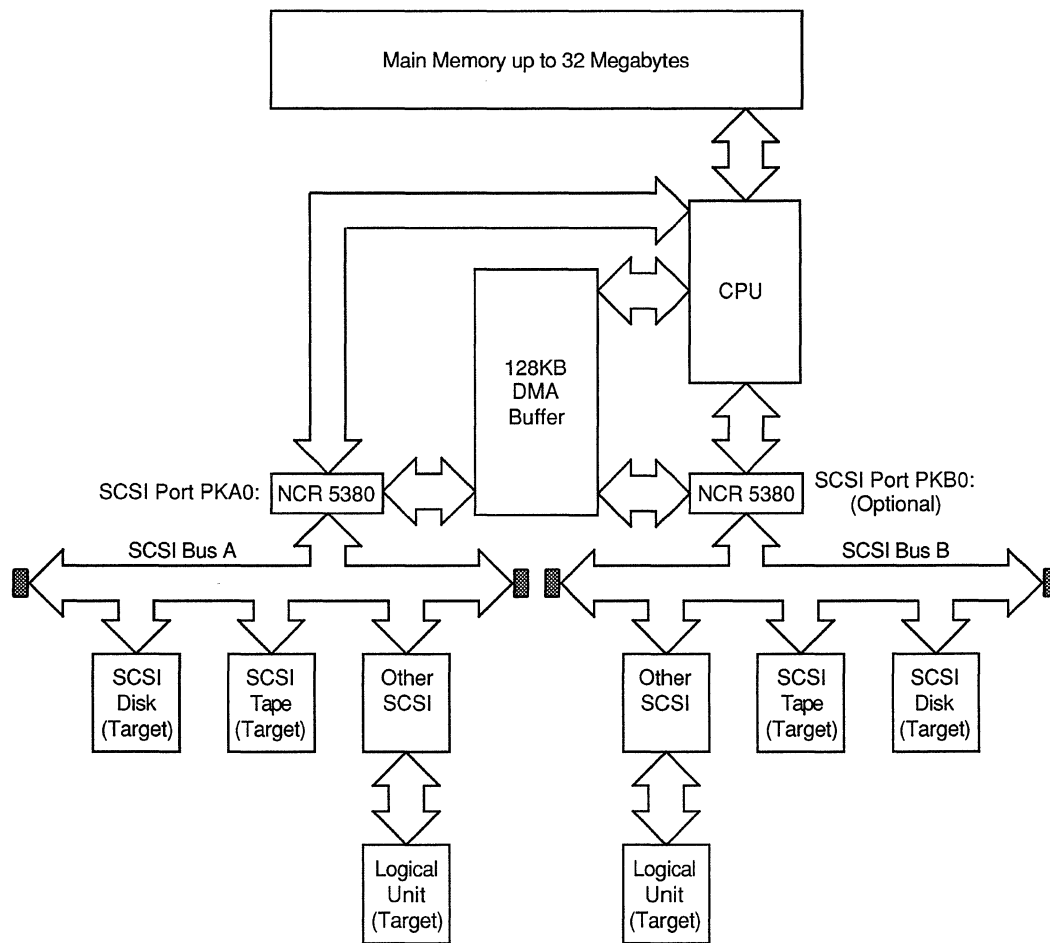
1.3 Hardware Considerations

MicroVAX/VAXstation 3100 systems are uniprocessing systems, providing access to one or two SCSI buses, each under the control of an NCR 5380 SCSI controller chip that supports asynchronous data transfers. MicroVAX/VAXstation 3100 systems support the SCSI asynchronous event notification feature. Figure 1-2 shows a representative configuration of a MicroVAX/VAXstation 3100 system.

Introduction

1.3 Hardware Considerations

Figure 1-2 MicroVAX/VAXstation 3100 System Configuration



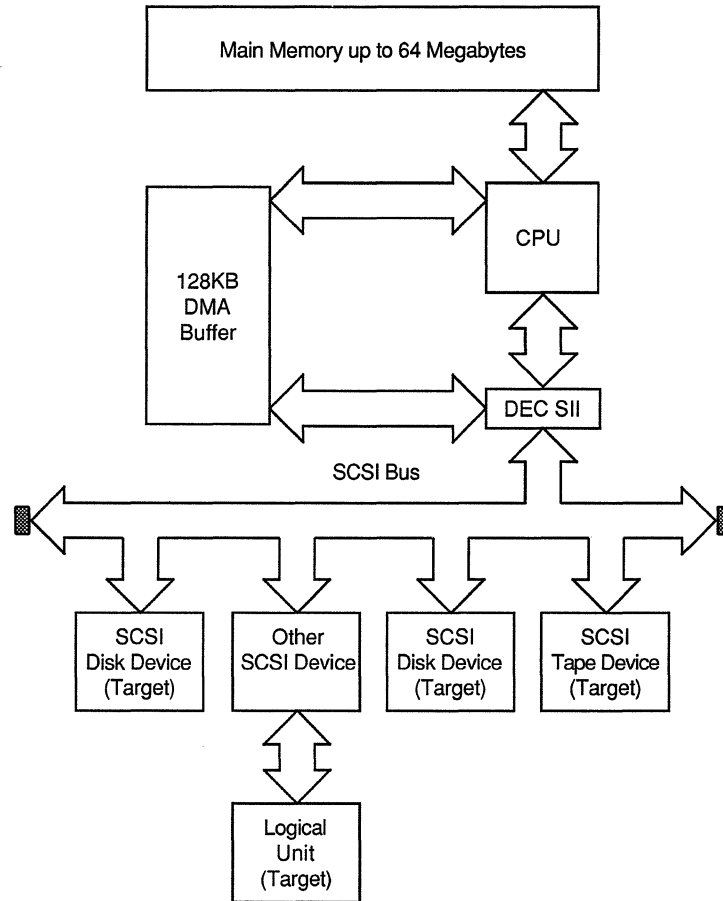
ZK-1367A-GE

The VAXstation 3520/3540 systems are multiprocessing systems, providing access to a single SCSI bus by means of Digital's SII SCSI controller chip. The SII chip supports both asynchronous and synchronous data transfers. VAXstation 3520/3540 systems do not support the SCSI asynchronous event notification feature. Figure 1-3 shows a representative configuration of the VAXstation 3520/3540 system.

Introduction

1.3 Hardware Considerations

Figure 1-3 VAXstation 3520/3540 System Configuration



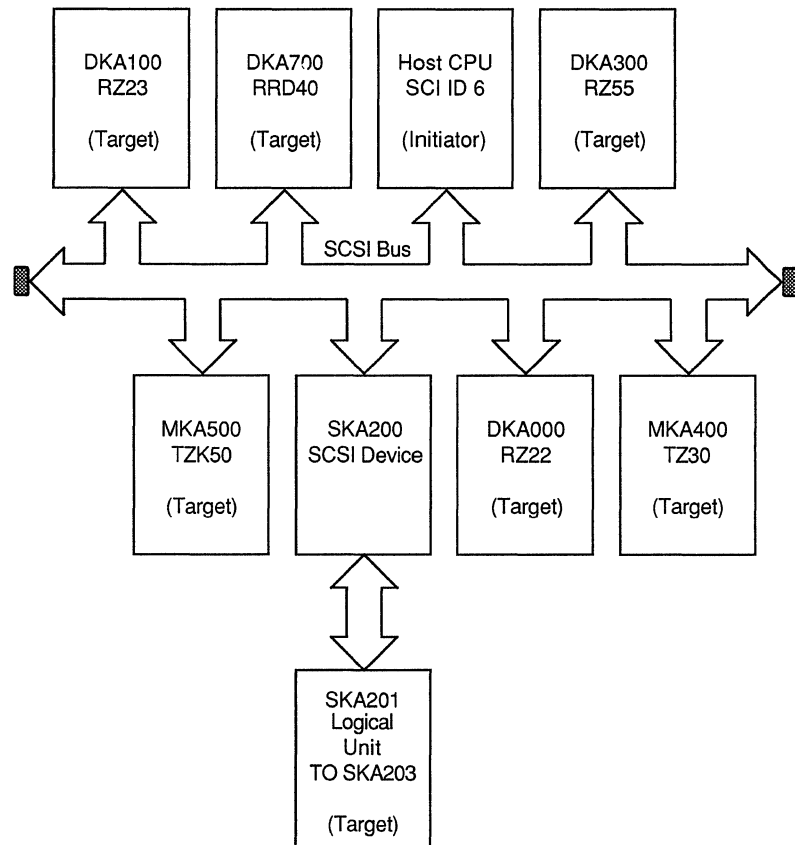
ZK-1368A-GE

Each SCSI bus in the system is identified by a *SCSI port ID* (*A* or *B*) (see Figure 1-4). The SCSI port ID uniquely identifies a *SCSI port*: that is, the SCSI controller channel that controls communications to and from a specific SCSI bus on the system.

Introduction

1.3 Hardware Considerations

Figure 1-4 SCSI Bus Configuration



ZK-1369A-GE

Each SCSI bus supports seven devices and a processor, at SCSI IDs 0 through 7. As defined by the ANSI SCSI specification, a *SCSI ID* refers to a line on the SCSI data bus (DB) on which the device uniquely asserts itself. The VMS operating system uses the term *SCSI device ID* to represent this value. Typically, a MicroVAX/VAXstation 3100 system processor is assigned device ID 6 and asserts itself at DB(6); a VAXstation 3520/3540 system processor is assigned device ID 7 and asserts itself at DB(7).

According to the ANSI SCSI specification, a *logical unit* is a physical or virtual device accessible by means of a SCSI device. For instance, if a peripheral controller resides on the SCSI bus, it, in turn, can control up to eight devices. A *logical unit number* (LUN), an integer from 0 to 7, uniquely identifies the device with respect to the controller's SCSI device ID.

Transactions on the SCSI bus are between an *initiator* and a *target*. The initiator, usually the host processor, requests that another SCSI device, the target, perform a certain operation. In situations in which the host processor requires notification of some unexpected event on the SCSI bus, the ANSI specification defines the *asynchronous event notification* (AEN)

Introduction

1.3 Hardware Considerations

protocol. AEN allows a SCSI device that is usually a target to inform the processor that an event has occurred asynchronously with respect to the processor's current stream of execution. (Certain MicroVAX/VAXstation implementations make the AEN protocol available to non-Digital-supplied SCSI class drivers, as described in Section 3.8.)

As Figures 1-2 and 1-3 illustrate, the MicroVAX/VAXstation 3100 and VAXstation 3520/3540 port hardware cannot directly access data in main memory. In order to access command, status, and data buffers involved in an operation on the SCSI bus, the MicroVAX/VAXstation port hardware must refer to its own direct-memory-access (DMA) buffer. Whenever the port hardware requires access to buffered information, the standard VMS port driver dynamically allocates a segment of the *port DMA buffer* and maps to it the pages of the buffer in main memory in a system-dependent manner.

1.3.1 **Connecting a Non-Digital-Supplied SCSI Disk or Tape Drive to a VMS SCSI Port**

The System Generation Utility (SYSGEN) automatically loads and autoconfigures the SCSI port driver at system initialization. As part of autoconfiguration, SYSGEN polls each device on each SCSI bus. If the device identifies itself as a direct-access device, direct-access CDROM device, or flexible disk device, SYSGEN automatically loads the VMS disk class driver (DKDRIVER); if the device identifies itself as a sequential-access device, SYSGEN automatically loads the VMS tape class driver (MKDRIVER). If the autoconfiguration facility does not recognize the type of the SCSI device, it loads no driver.

The design of the standard VMS disk class driver allows it to control most disk drives that conform to the ANSI SCSI-2 specification. Because the ANSI SCSI specification is not as well defined for tapes as for disks, the standard VMS tape class driver may or may not work with a specific non-Digital-supplied tape drive.

The ANSI SCSI specification allows some flexibility in certain implementation details and omits others. As a result, implementations of the SCSI standard may differ from manufacturer to manufacturer and from device to device. (Appendix E lists Digital's requirements for SCSI device hardware.) Although the VMS operating system allows the use of non-Digital-supplied devices with the standard VMS disk and tape class drivers, the fact that a specific device is less likely to operate correctly in this manner does not imply VMS support of the device. Digital cannot guarantee that a non-Digital-supplied device that does currently run with a VMS class driver will continue to work with subsequent releases of the VMS operating system.

To ensure that non-Digital-supplied devices work properly in a VMS environment, Digital encourages the use of an established and supported VMS interface, such as the generic SCSI class driver (see Chapter 2) or a third-party SCSI class driver (see Chapter 3).

2

Using the VMS Generic SCSI Class Driver

The VMS generic SCSI class driver provides a mechanism by which an application program can control a non-Digital-supplied SCSI device that cannot be controlled by the standard VMS disk and tape class drivers. By means of a Queue I/O Request (\$QIO) system service call, a program can pass to the generic SCSI class driver a preformatted SCSI command descriptor block. The generic SCSI class driver, in conjunction with the standard VMS SCSI port driver, delivers this SCSI command to the device, manages any transfer of data from the device to a user buffer, and returns SCSI status to the application.

2.1 Overview of the VMS Generic SCSI Class Driver

In effect, an application using the generic SCSI class driver implements details of device control usually managed within device driver code. The programmer of such an application must understand which SCSI commands the device supports and which SCSI status values the device returns. The programmer must also be aware of the device's timeout requirements, data transfer capabilities, and command retry behavior.

The application program sets up the characteristics of the connection the generic SCSI class driver uses when delivering commands to, exchanging data with, and receiving status from the device. The program associates each I/O operation the device can perform with a specific SCSI command. When it receives a request for a particular operation, the application program creates the specific command descriptor block that, when passed to the device, causes it to perform that operation.

The application initiates all transactions to the SCSI device by means of a \$QIO call to the generic SCSI class driver, supplying the address and length of the SCSI command descriptor block, plus the parameters of any data transfer operation, in the call. When the transaction completes and the application program regains control, it interprets the returned status value, processes any returned data, and services any failure. To avoid conflicts with other applications accessing the same device, an application may need to explicitly allocate the device.

Because the generic SCSI class driver has no knowledge of specific device errors, it neither logs device errors nor implements error recovery. An application using the driver must manage device-specific errors itself. To service an error returned on a single transaction, the application must issue additional \$QIO requests and initiate further transactions to the device. If more precise or more efficient error recovery is required for a device, the developer should consider writing a third-party SCSI class driver, as described in Section 3.1. A third-party SCSI class driver can log errors associated with device activity by using the method described in Section 3.5.2.

Using the VMS Generic SCSI Class Driver

2.1 Overview of the VMS Generic SCSI Class Driver

A third-party class driver is the only means of supporting devices that themselves generate transactions on the SCSI bus, such as notification of a device selection or deselection event to the host processor. See the description of asynchronous event notification (AEN) in Section 3.8.

Figure 2–1 illustrates the flow of a \$QIO request through the generic SCSI class driver and the port driver.

When direct access to a target device on the SCSI bus is required, the generic SCSI class driver is loaded for that device, as described in Section 2.4. An application program using the generic class driver performs the following tasks to issue a command to the target device:

- 1 Calls the Assign I/O Channel (\$ASSIGN) system service to assign a channel to the generic SCSI class driver, and allocate the device for the application's exclusive use
- 2 Formats a SCSI command descriptor block
- 3 Formats any data to be transferred to the device
- 4 Calls the Queue I/O Request (\$QIO) system service to request the generic SCSI class driver to send the SCSI command descriptor block to the port driver
- 5 Upon completion of the I/O request, interprets the SCSI status byte and any data returned from the target device

These operations are described in subsequent sections.

Note: Because incorrect or malicious use of the generic SCSI class driver can result in SCSI bus hangs and lead to SCSI bus resets, DIAGNOSE and PHY_IO privileges are required to access the driver. An application program can be designed in such a way as to filter user I/O requests, thus allowing nonprivileged users access to some device functions.

2.2 Accessing the VMS Generic SCSI Class Driver

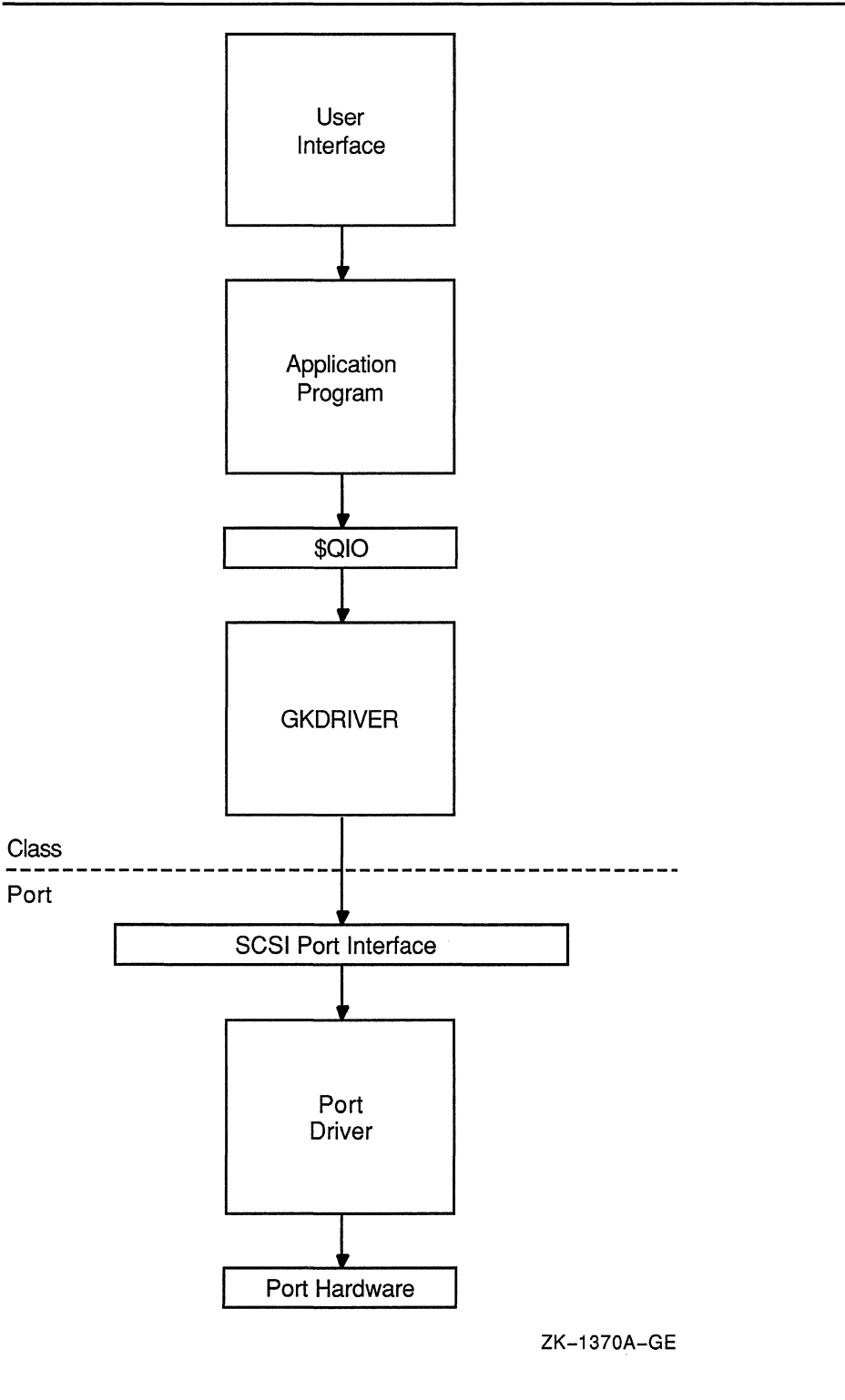
Interactive commands and procedure calls can use the VMS generic SCSI class driver to access devices on the SCSI bus. However, it is unlikely that a user application would access a device on the SCSI bus by directly using the \$QIO interface of the generic SCSI class driver. First of all, any user process directly using the \$QIO interface would require DIAGNOSE and PHY_IO privileges. Under normal circumstances, it would be a system security risk to grant DIAGNOSE and PHY_IO privileges to many system users. Secondly, it would be cumbersome for end users of the device to identify, format, and issue SCSI commands to the device. Rather, it would be more efficient to develop an interface that hides these details.

A utility program, installed with the DIAGNOSE and PHY_IO privileges, can provide nonprivileged users with a command line interface to a SCSI device. The utility translates interactive commands provided by the user into the appropriate set of SCSI commands and sends them to the device

Using the VMS Generic SCSI Class Driver

2.2 Accessing the VMS Generic SCSI Class Driver

Figure 2-1 Generic SCSI Class Driver Flow



ZK-1370A-GE

Using the VMS Generic SCSI Class Driver

2.2 Accessing the VMS Generic SCSI Class Driver

using the \$QIO interface provided by the generic SCSI class driver. The utility checks user commands to ensure that only valid SCSI commands are sent to the device. See the *Guide to VMS Programming Resources* and the *VMS Install Utility Manual* for information about installing images with privileges.

A privileged shareable image can provide system applications with a procedure interface to a SCSI device. The image contains a set of procedures that translate operations specified by the caller into the appropriate set of SCSI commands. The SCSI commands are sent to the device through the \$QIO interface of the generic SCSI class driver. The privileged shareable image checks its caller's parameters to ensure that only valid SCSI commands are sent to the device. See the *Introduction to VMS System Services* for information about creating shareable images.

2.3 SCSI Port Features Under Application Control

The standard VMS SCSI port driver provides mechanisms by which the generic SCSI class driver can control the nature of data transfers and command transmission across the SCSI bus. An application uses the \$QIO interface to tailor these mechanisms to the specific device it supports. Among the features under application program control are the following:

- Data transfer mode
- Disconnection and reselection
- Command retry
- Command timeouts

The following sections discuss these features.

2.3.1 Setting the Data Transfer Mode

The SCSI bus defines two data transfer modes, asynchronous and synchronous. In asynchronous mode, for each REQ from a target there is an ACK from the host prior to the next REQ from the target. Synchronous mode allows higher data transfer rates by allowing a pipelined data transfer mechanism where, for short bursts (defined by the REQ-ACK offset), the target can pipeline data to an initiator without waiting for the initiator to respond.

Whether or not a port or a target device allows synchronous data transfers, it is harmless for the program to set up the connection to use such transfers. If synchronous mode is not supported, the port driver automatically uses asynchronous mode.

To use synchronous mode in a transfer, a programmer using the generic SCSI class driver must ensure that both the SCSI port and the SCSI device involved in the transfer support synchronous mode. The SCSI port of the VAXstation 3520/3540 system allows both synchronous and asynchronous transfers, whereas that of MicroVAX/VAXstation 3100 systems supports only asynchronous transfers.

Using the VMS Generic SCSI Class Driver

2.3 SCSI Port Features Under Application Control

To set up a connection to use synchronous data transfer mode, a program using the generic SCSI class driver sets the **syn** bit in the **flags** field of the generic SCSI descriptor, the address of which is passed to the driver in the **p1** argument to the \$QIO request.

2.3.2 Enabling Disconnection and Reselection

The ANSI SCSI specification defines a disconnection facility that allows a target device to yield ownership of the SCSI bus while seeking or performing other time-consuming operations. When a target disconnects from the SCSI bus, it sends a sequence of messages to the initiator that cause it to save the state of the I/O transfer in progress. Once this is done, the target releases the SCSI bus. When the target is ready to complete the operation, it reselects the initiator and sends to it another sequence of messages. This sequence uniquely identifies the target and allows the initiator to restore the context of the suspended I/O operation.

Whether disconnection should be enabled or disabled on a given connection depends on the nature and capabilities of the device involved in the transfer, as well as on the configuration of the system. In configurations where there is a slow device present on the SCSI bus, enabling disconnection on connections that transfer data to the device can increase bus throughput. By contrast, systems where most of the I/O activity is directed towards a single device for long intervals can benefit from disabling disconnection. By disabling disconnection when there is no contention on the SCSI bus, port drivers can increase throughput and decrease the processor overhead for each I/O request.

By default, the VMS class/port interface disables the disconnect facility on a connection. To enable disconnection, an application program using the generic SCSI class driver sets the **dis** bit of the **flags** field of the generic SCSI descriptor, the address of which is passed to the driver in the **p1** argument to the \$QIO call.

2.3.3 Disabling Command Retry

The SCSI port driver implements a command retry mechanism, which is enabled on a given connection by default.

When the command retry mechanism is enabled, the port driver retries up to three times any I/O operation that fails during the COMMAND, Message, Data, or STATUS phases. For instance, if the port driver detects a parity error during the Data phase, it aborts the I/O operation, logs an error, and retries the I/O operation. It repeats this sequence twice more, if necessary. If the I/O operation completes successfully during a retry attempt, the port driver returns success status to the class driver. However, if all retry attempts fail, the port driver returns failure status to the class driver.

An application may need to disable the command retry mechanism under certain circumstances. For example, repeated execution of a command on a sequential device may produce different results than are intended by a

Using the VMS Generic SCSI Class Driver

2.3 SCSI Port Features Under Application Control

single command request. A tape drive could perform a partial write and then repeat the write without resetting the tape position.

An application program using the generic SCSI class driver can disable the command retry mechanism by setting the **dpr** bit of the **flags** field of the generic SCSI descriptor, the address of which is passed to the driver in the **p1** argument to the \$QIO request.

2.3.4 Setting Command Timeouts

The SCSI port driver implements several timeout mechanisms, some governed by the ANSI SCSI specification and others required by the VMS operating system. The timeouts required by the VMS operating system include the following:

Timeout	Description
Phase change timeout	Maximum number of seconds for a target to change the SCSI bus phase or complete a data transfer. (This value is also known as the <i>DMA timeout</i> .) Upon sending the last command byte, the port driver waits this many seconds for the target to change the bus phase lines and assert REQ (indicating a new phase). Or, if the target enters the DATA IN or DATA OUT phase, the transfer must be completed within this interval.
Disconnect timeout	Maximum number of seconds, from the time the initiator receives the DISCONNECT message, for a target to reselect the initiator so that it can proceed with the disconnected I/O transfer.

An application program using the generic SCSI class driver is responsible for maintaining both of these timeout values. It has the following options:

- Accepting a connection's default value. The default value for both timeouts is 4 seconds.
- Altering the connection's default value. To modify the default values, the class driver specifies nonzero values for the **phase change timeout** and **disconnect timeout** fields of the generic SCSI descriptor, the address of which is passed to the driver in the **p1** argument to the \$QIO system service call.

2.4 Configuring a Device Using the Generic Class Driver

The System Generation Utility (SYSGEN) loads the generic SCSI class driver into system virtual memory, creates additional data structures for the device unit, and calls the driver's controller initialization routine and unit initialization routine. SYSGEN automatically loads and autoconfigures the SCSI port driver at system initialization. As part of autoconfiguration, SYSGEN polls each device on each SCSI bus. If the device identifies itself as a direct-access device, direct-access CDROM device, or flexible disk device, SYSGEN automatically loads the VMS disk

Using the VMS Generic SCSI Class Driver

2.4 Configuring a Device Using the Generic Class Driver

class driver (DKDRIVER); if the device identifies itself as a sequential-access device, SYSGEN automatically loads the VMS tape class driver (MKDRIVER). If the autoconfiguration facility does not recognize the type of the SCSI device, it loads no driver.

Consequently, if a non-Digital-supplied SCSI device requires that the generic class driver be loaded, it must be configured by an explicit SYSGEN CONNECT command, as follows:

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> CONNECT GKpd0u /NOADAPTER
```

In this command, *GK* is the device mnemonic for the generic SCSI class driver (GKDRIVER); *p* represents the SCSI port ID (for instance, the controller ID *A* or *B*); *d* represents the SCSI device ID (a digit from 0 to 7); 0 signifies the digit zero; and *u* represents the SCSI logical unit number (a digit from 0 to 7).

Multiple devices residing on any SCSI bus in the system can share GKDRIVER as a class driver, as long as a SYSGEN CONNECT command is issued for each target device that requires the driver.

Because just one connection can exist through the SCSI port driver to each target, the generic class driver cannot be used for a target if a different SCSI class driver is already connected to that target. For example, if the SCSI disk class driver has a connection to device ID 2 on the SCSI bus identified by SCSI port ID *B* (DKB200), the generic class driver cannot be used to communicate with this disk. An attempt to connect GKDRIVER for this target results in GKB200 being placed off line.

2.4.1 Disabling the Autoconfiguration of a SCSI Device

Note that, in special cases, you may need to prevent SYSGEN's autoconfiguration facility from loading the VMS disk or tape class driver for a device with a specific port ID and device ID. This would be the case if a non-Digital-supplied SCSI device should identify itself as either a random-access or sequential-access device and were to be controlled by the generic SCSI class driver.

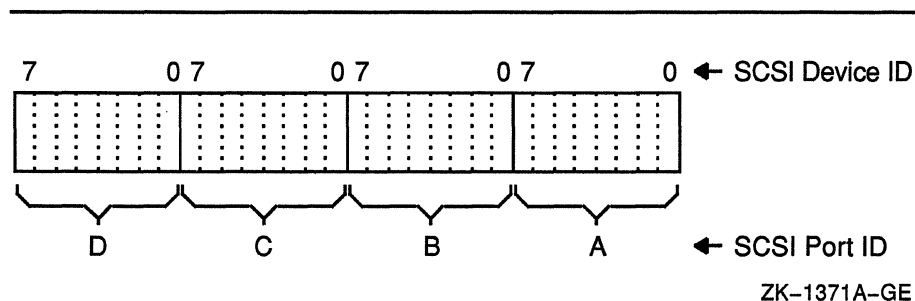
To disable the loading of a VMS disk or tape driver for any given device ID, VMS Version 5.3 temporarily defines the special SYSGEN parameter **VMDS2**.

The **VMDS2** system parameter, as shown in Figure 2-2, stores a bit mask of 32 bits in which the low-order byte corresponds to the first SCSI bus (PKA0), the second byte corresponds to the second SCSI bus (PKB0), and so on. For each SCSI bus, setting the low-order bit inhibits automatic configuration of the device with SCSI device ID 0; setting the second low-order bit inhibits automatic configuration of the device with SCSI device ID 1, and so forth. For instance, the value 00002000_{16} would prevent the device with SCSI ID 5 on the bus identified by SCSI port ID *B* from being configured. By default, all of the bits in the mask are cleared, allowing all devices to be configured.

Using the VMS Generic SCSI Class Driver

2.4 Configuring a Device Using the Generic Class Driver

Figure 2-2 VMUSD2 System Parameter



Note: A future release of VMS will provide a different mechanism for preventing the configuration of a VMS SCSI class driver for a given device ID. At that time, the VMUSD2 system parameter will revert to its status of a special parameter reserved to Digital.

2.5 Assigning a Channel to GKDRIVER

An application program assigns a channel to the generic SCSI class driver using the standard call to the \$ASSIGN system service, as described in the *VMS System Services Reference Manual*. The application program specifies a device name and a word to receive the channel number.

2.6 Issuing a \$QIO Request to the Generic Class Driver

The format of the Queue I/O Request (\$QIO) system service that initiates a request to the SCSI generic class driver is as follows. This explanation concentrates on the special elements of a \$QIO request to the SCSI generic class driver. For a detailed description of the \$QIO system service, see the *VMS System Services Reference Manual*.

VAX MACRO Format

```
$QIO [efn] ,chan ,func ,iosb ,[astadr] ,[astprm] -
      ,p1 ,p2 [,p3] [,p4] [,p5] [,p6]
```

High-Level Language Format

```
SYS$QIO ([efn] ,chan ,func ,iosb ,[astadr] ,[astprm]
          ,p1 ,p2 [,p3] [,p4] [,p5] [,p6])
```

Arguments

chan

I/O channel assigned to the device to which the request is directed. The **chan** argument is a word value containing the number of the channel, as returned by the Assign I/O Channel (\$ASSIGN) system service.

func

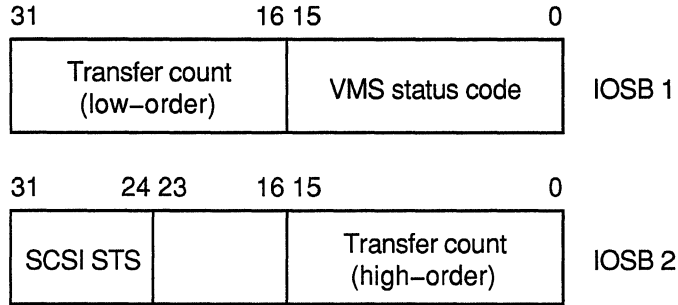
Longword value containing the IO\$_DIAGNOSE function code. Only the IO\$_DIAGNOSE function code is implemented in the generic SCSI class driver.

Using the VMS Generic SCSI Class Driver

2.6 Issuing a \$QIO Request to the Generic Class Driver

iosb

I/O status block. The **iosb** argument is required in a request to the generic SCSI class driver; it has the following format:



ZK-1372A-GE

The VMS status code provides the final status indicating the success or failure of the SCSI command. The SCSI status byte contains the status value returned from the target device, as defined in the ANSI SCSI specification. The transfer count field specifies the actual number of bytes transferred during the SCSI bus DATA IN or DATA OUT phase.

- [efn]**
- [astadr]**
- [astprm]**

These arguments apply to \$QIO system service completion. For an explanation of these arguments, see the *VMS System Services Reference Manual*.

Using the VMS Generic SCSI Class Driver

2.6 Issuing a \$QIO Request to the Generic Class Driver

p1

Address of a generic SCSI descriptor of the following format:

31	0	
	0	opcode
	4	flags
	8	SCSI command address
	12	SCSI command length
	16	SCSI data address
	20	SCSI data length
	24	SCSI pad length
	28	phase change timeout
	32	disconnect timeout
	36	reserved
	56	

ZK-1373A-GE

p2

Length of the generic SCSI descriptor.

Descriptor Fields

opcode

Currently, the only supported opcode is 1, indicating a pass-through function. Other opcode values are reserved for future expansion.

flags

Bit map having the following format:

31	4	3	2	1	0
	reserved	dpr	syn	dis	dir

ZK-1374A-GE

Using the VMS Generic SCSI Class Driver

2.6 Issuing a \$QIO Request to the Generic Class Driver

Bits in the flags bit map are defined as follows:

Field	Definition
dir	<p>Direction of transfer.</p> <p>If this bit is set, the target is expected at some time to enter the DATA IN phase to send data to the host. To facilitate this, the port driver maps the specified data buffer for write access.</p> <p>If this bit is clear, the target is expected at some time to enter the DATA OUT phase to receive data from the host. To facilitate this, the port driver maps the specified data buffer for read access.</p> <p>The generic SCSI class driver ignores the dir flag if either the SCSI data address or SCSI data length field of the generic SCSI descriptor is zero.</p>
dis	<p>Enable disconnection.</p> <p>If this bit is set, the target device is allowed to disconnect during the execution of the command.</p> <p>If this bit is clear, the target cannot disconnect during the execution of the command.</p> <p>Note that targets that hold on to the bus for long periods of time without disconnecting can adversely affect system performance. See Section 2.3.2 for additional information.</p>
syn	<p>Enable synchronous mode.</p> <p>If this bit is set, the port driver uses synchronous mode for data transfers, if both the host and target allow this mode of operation.</p> <p>If this bit is clear, or synchronous mode is not supported by either the host or target, the port driver uses asynchronous mode for data transfers.</p> <p>See Section 2.3.1 for additional information.</p>
dpr	<p>Disable port retry.</p> <p>If this bit is clear, the port driver retries, up to three times, any command that fails with a timeout, bus parity, or invalid phase transition error.</p> <p>If this bit is set, the port driver does not retry commands for which it detects failure.</p> <p>See Section 2.3.3 for additional information.</p>

SCSI command address

Address of a buffer containing a SCSI command.

SCSI command length

Length of the SCSI command. The maximum length of the SCSI command is 128 bytes.

SCSI data address

Address of a data buffer associated with the SCSI command.

If the **dir** bit is set in the **flags** field, data is written into this buffer during the execution of the command. Otherwise, data is read from this buffer and sent to the target device.

If the SCSI command requires no data to be transferred, then the **SCSI data address** field should be clear.

Using the VMS Generic SCSI Class Driver

2.6 Issuing a \$QIO Request to the Generic Class Driver

SCSI data length

Length in bytes of the data buffer pointed to by the **SCSI data address** field. For the MicroVAX/VAXstation 3100 and VAXstation 3520/3540 systems, the maximum data buffer size is 65,535 bytes.

If the SCSI command requires no data to be transferred, then this field should be clear.

SCSI pad length

This field is used to accommodate SCSI device classes that require that the transfer length be specified in terms of a larger data unit than the count of bytes expressed in the **SCSI data length** field. If the total amount of data requested in the SCSI command does not match that specified in the **SCSI data length** field, this field must account for the difference.

For example, suppose an application program is using the generic class driver to read the first 2 bytes of a disk block. The length field in the SCSI READ command contains 1 (indicating one logical block, or 512 bytes), while the **SCSI data length** field contains a 2. The **SCSI pad length** field must contain the difference, 510 bytes.

For most transfers, this field should contain 0. Failure to initialize the **SCSI pad length** field properly causes port driver timeouts and SCSI bus resets.

phase change timeout

Maximum number of seconds for a target to change the SCSI bus phase or complete a data transfer. A value of 0 causes the SCSI port driver's default phase change timeout value of 4 seconds to be used.

See Section 2.3.4 for additional information.

disconnect timeout

Maximum number of seconds for a target to reselect the initiator to proceed with a disconnected I/O transfer. A value of 0 causes the SCSI port driver's default disconnect timeout value of 4 seconds to be used.

See Section 2.3.4 for additional information.

2.7 Obtaining Device Information

A call to the Get Device/Volume Information (\$GETDVI) system service returns the following information for any device serviced by the generic SCSI class driver. (See the description of the \$GETDVI system service in *VMS System Services Reference Manual*.)

\$GETDVI returns the following device characteristics when you specify the item code DVI\$_DEVCHAR:

DEV\$_AVL	Available device
DEV\$_IDV	Input device
DEV\$_ODV	Output device
DEV\$_SHR	Shareable device
DEV\$_RND	Random-access device

DVI\$DEVCLASS returns the device class, which is DC\$_MISC;
DVI\$DEVTYPE returns the device type, which is DT\$_GENERIC SCSI.

Using the VMS Generic SCSI Class Driver

2.8 Programming Example

2.8 Programming Example

The following application program uses the generic SCSI class driver to send a SCSI INQUIRY command to a device.

```
/*
GKTEST.C

This program uses the SCSI generic class driver to send an inquiry command
to a device on the SCSI bus.
*/

#include ctype

/* Define the descriptor used to pass the SCSI information to GKDRIVER */

#define OPCODE 0
#define FLAGS 1
#define COMMAND_ADDRESS 2
#define COMMAND_LENGTH 3
#define DATA_ADDRESS 4
#define DATA_LENGTH 5
#define PAD_LENGTH 6
#define PHASE_TIMEOUT 7
#define DISCONNECT_TIMEOUT 8

#define FLAGS_READ 1
#define FLAGS_DISCONNECT 2

#define GK_EFN 1

#define SCSI_STATUS_MASK 0X3E

#define INQUIRY_OPCODE 0x12
#define INQUIRY_DATA_LENGTH 0x30

globalvalue
    IO$_DIAGNOSE;

short
    gk_chan,
    transfer_length;

int
    i,
    status,
    gk_device_desc[2],
    gk_iosb[2],
    gk_desc[15];

char
    scsi_status,
    inquiry_command[6] = {INQUIRY_OPCODE, 0, 0, 0, INQUIRY_DATA_LENGTH, 0},
    inquiry_data[INQUIRY_DATA_LENGTH],
    gk_device[] = {"GKA0"};

main ()
{
    /* Assign a channel to GKA0 */

    gk_device_desc[0] = 4;
    gk_device_desc[1] = &gk_device[0];
    status = sys$assign (&gk_device_desc[0], &gk_chan, 0, 0);
    if (!(status & 1)) sys$exit (status);

    /* Set up the descriptor with the SCSI information to be sent to the target */

```


Using the VMS Generic SCSI Class Driver

2.8 Programming Example

```
gk_desc[OPCODE] = 1;
gk_desc[FLAGS] = FLAGS_READ + FLAGS_DISCONNECT;
gk_desc[COMMAND_ADDRESS] = &inquiry_command[0];
gk_desc[COMMAND_LENGTH] = 6;
gk_desc[DATA_ADDRESS] = &inquiry_data[0];
gk_desc[DATA_LENGTH] = INQUIRY_DATA_LENGTH;
gk_desc[PAD_LENGTH] = 0;
gk_desc[PHASE_TIMEOUT] = 0;
gk_desc[DISCONNECT_TIMEOUT] = 0;
for (i=9; i<15; i++) gk_desc[i] = 0;    /* Clear reserved fields */

/* Issue the QIO to send the inquiry command and receive the inquiry data */
status = sys$qiow (GK_EFN, gk_chan, IO$_DIAGNOSE, gk_iosb, 0, 0,
                 &gk_desc[0], 15*4, 0, 0, 0, 0);

/* Check the various returned status values */

if (!(status & 1)) sys$exit (status);
if (!(gk_iosb[0] & 1)) sys$exit (gk_iosb[0] & 0xffff);
scsi_status = (gk_iosb[1] >> 24) & SCSI_STATUS_MASK;
if (scsi_status) {
    printf ("Bad SCSI status returned: %02.2x\n", scsi_status);
    sys$exit (1);
}

/* The command succeeded. Display the SCSI data returned from the target */

transfer_length = gk_iosb[0] >> 16;
printf ("SCSI inquiry data returned: ");
for (i=0; i<transfer_length; i++) {
    if (isprint (inquiry_data[i]))
        printf ("%c", inquiry_data[i]);
    else
        printf (".");
}
printf ("\n");
}
```

3

Writing a VMS SCSI Class Driver

The VMS operating system defines a mechanism by which a system programmer can write a class driver that, in conjunction with a standard VMS SCSI port driver, exchanges data, commands, and status with a third-party device on the SCSI bus. Given the particular requirements of the device, or the expectations of application programs accessing the device, the programmer may choose to create a SCSI class driver rather than employ the VMS generic SCSI class driver discussed in Chapter 2.

By writing a device-specific SCSI class driver, a programmer can define a unique, simple, robust \$QIO interface to a SCSI device. The generic SCSI class driver, by contrast, provides a more complex \$QIO interface, requiring the application program to have some knowledge of the data transfer mode and capabilities of the target device and to construct in memory the SCSI commands to be passed to the SCSI port. A third-party SCSI class driver conceals these details from the application program. Additionally it can provide device-specific error recovery, full error logging, and notification of asynchronous events from the device.

Note: A non-Digital-supplied SCSI disk device residing on the local node and controlled by a SCSI third-party class driver cannot be served to other nodes of the local area VAXcluster.

This chapter introduces the VMS SCSI class/port interface and discusses the mechanisms VMS provides to facilitate the creation of a SCSI class driver. It describes the capabilities and components of such a driver and suggests some coding strategies. It also includes sections on driver naming conventions, driver loading, and driver debugging techniques. It concludes with descriptions of class driver error logging protocol and the asynchronous event notification facility.

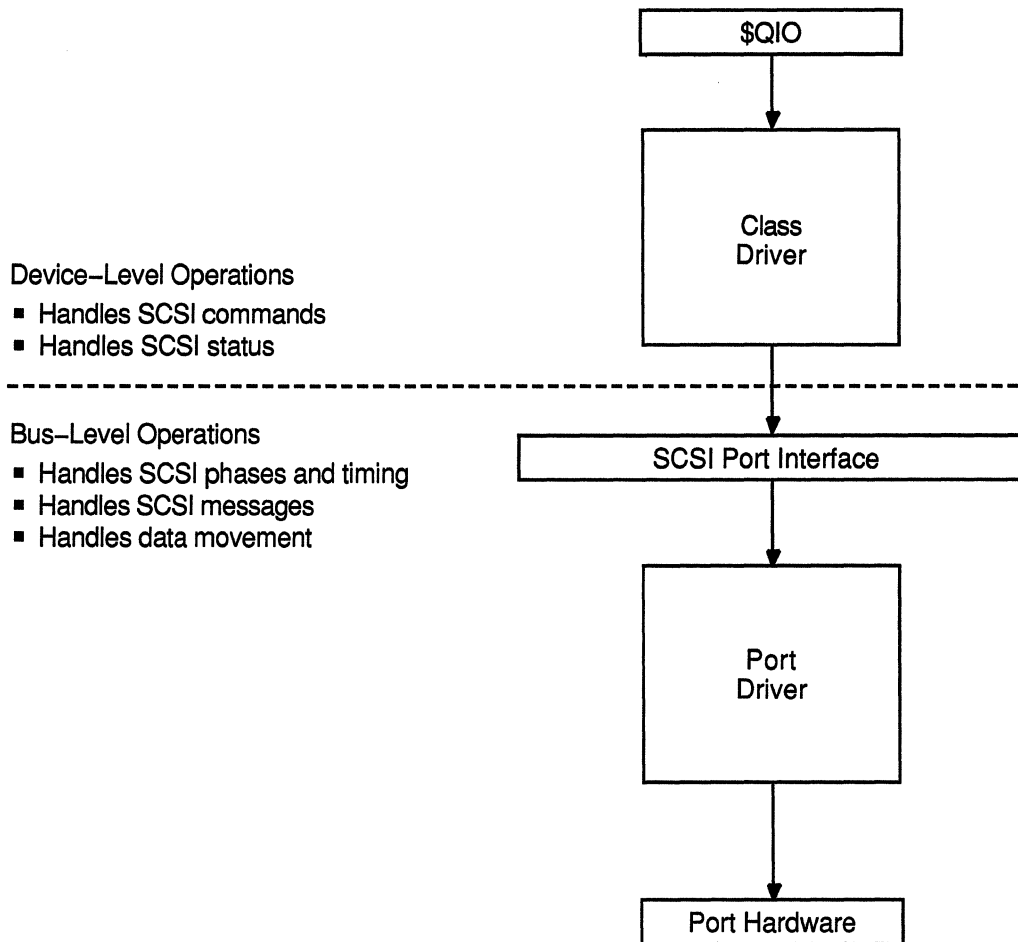
3.1 SCSI Class/Port Architecture

VMS uses a class/port driver architecture to communicate with devices on the SCSI bus. The class/port design allows the responsibilities for communication between the operating system and the device to be cleanly divided between two separate driver images (see Figure 3-1).

Writing a VMS SCSI Class Driver

3.1 SCSI Class/Port Architecture

Figure 3-1 VMS SCSI Class/Port Interface



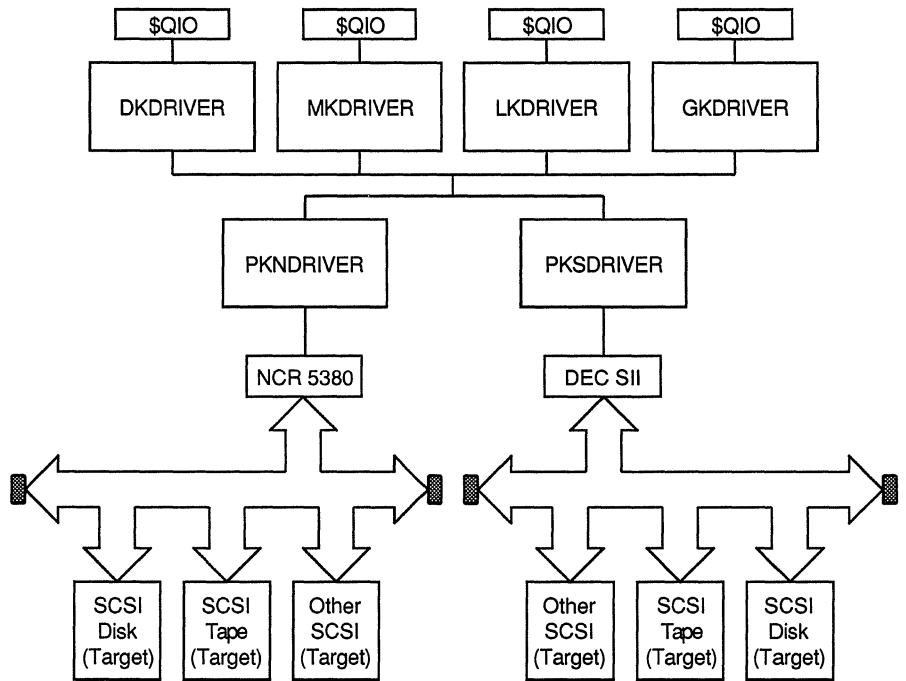
ZK-1366A-GE

The *SCSI port driver* transmits and receives SCSI commands and data. It knows the details of transmitting data from the local processor's SCSI port hardware across the SCSI bus. Although it understands SCSI bus phases, protocol, and timing, the SCSI port driver has no knowledge of the SCSI commands the device supports, the status messages it returns, or the format of the packets in which this information is delivered. Strictly speaking, the port driver is a communications path. When directed by a SCSI class driver, the port driver forwards commands and data from the class driver onto the SCSI bus to the device. On a single MicroVAX/VAXstation system, a single SCSI port driver handles bus-level communications for all SCSI class drivers that may exist on the system (see Figure 3-2).

Writing a VMS SCSI Class Driver

3.1 SCSI Class/Port Architecture

Figure 3-2 VMS SCSI Port Driver Configuration



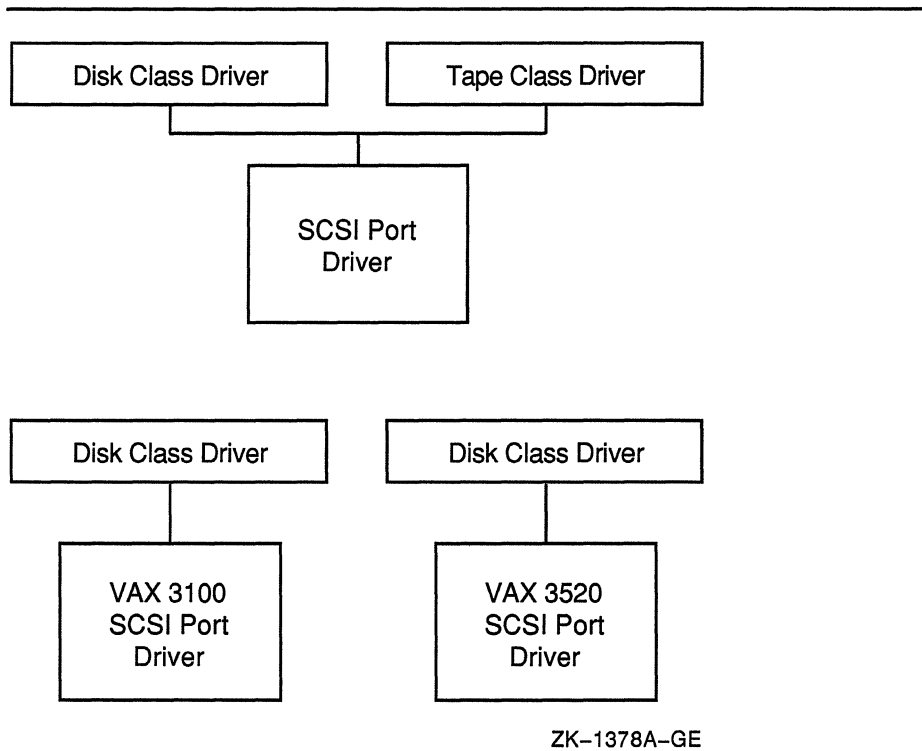
ZK-1379A-GE

The *SCSI class driver* acts as an interface between the user and the SCSI port, translating I/O functions as specified in a user's \$QIO request to a SCSI command directed to a device on the SCSI bus. Although the class driver knows about SCSI command descriptor buffers, status codes, and data, it has no knowledge of underlying bus protocols or hardware, command transmission, bus phases, timing, or messages (except in asynchronous event notification mode, as described in Section 3.8). A single SCSI class driver can run with the SCSI port driver of any MicroVAX/VAXstation system, controlling the same set of devices on each system (see Figure 3-3).

Writing a VMS SCSI Class Driver

3.1 SCSI Class/Port Architecture

Figure 3-3 VMS SCSI Class Driver Configuration



ZK-1378A-GE

The design of the SCSI driver class/port interface allows a programmer to write a class driver that is independent of any concern about the underlying hardware. VMS supplies software tools that facilitate the development of SCSI class drivers, including the following:

- A standard interface that all SCSI class drivers use to request work from and transfer control to the port driver. This interface is known as the *SCSI port interface* (SPI).
- SCSI-specific data structures that class and port drivers use to exchange information and monitor the state of the device connection or SCSI port.
- A template SCSI class driver that can serve as the basis for a third-party SCSI class driver.

3.1.1 SCSI Port Interface

The SCSI port interface (SPI) consists of a group of routines within the SCSI port driver that create and manage the connection between a SCSI class driver and a device unit. Across this connection, SPI routines exchange control information and data between the class driver and the port.

Writing a VMS SCSI Class Driver

3.1 SCSI Class/Port Architecture

When a connection must be established, a SCSI command transmitted, or data transferred, a SCSI class driver calls the appropriate routine within the port driver by invoking one of a series of macros, defined in `SYS$LIBRARY:LIB.MLB`. Each macro corresponds to a vector in the SCSI port descriptor table (SPDT) that supplies the address of the port routine that performs the applicable function. Table 3–1 lists the standard SPI macros and their functions.

Table 3–1 SCSI Port Interface (SPI) Macros

Macro	Description
<code>SPI\$ABORT_COMMAND</code>	Aborts the execution of an outstanding SCSI command over a specified connection
<code>SPI\$ALLOCATE_COMMAND_BUFFER</code>	Allocates a buffer in which a class driver passes a SCSI command descriptor to the port driver
<code>SPI\$CONNECT</code>	Creates a connection from a class driver to a SCSI device unit
<code>SPI\$DEALLOCATE_COMMAND_BUFFER</code>	Deallocates a SCSI command buffer
<code>SPI\$DISCONNECT</code>	Breaks the connection between a class driver and a SCSI device unit
<code>SPI\$GET_CONNECTION_CHAR</code>	Obtains the characteristics of a specified connection and places them in the buffer specified by the class driver
<code>SPI\$MAP_BUFFER</code>	Makes the process buffer involved in a data transfer available to the port driver
<code>SPI\$RESET</code>	Resets the port hardware and SCSI bus
<code>SPI\$SEND_COMMAND</code>	Delivers a SCSI command descriptor buffer to a SCSI device, returning status and data, if applicable
<code>SPI\$SET_CONNECTION_CHAR</code>	Sets up the characteristics of a specified connection
<code>SPI\$UNMAP_BUFFER</code>	Releases the SCSI port's DMA buffer space and the system page-table entries that double-mapped a user buffer involved in a transfer

A SCSI class driver invokes SPI macros at fork IPL, holding the fork lock. Because the port driver routines called by SPI macros may fork or stall, a class driver must preserve local context and local return addresses across an SPI macro invocation. It must also ensure that the address of its caller is at the top of the stack at the time the macro is invoked. (These issues are more fully discussed in Section 3.5.1.)

Detailed descriptions of the functions provided by the SPI macros appear where pertinent in the discussions of SCSI class driver operations that follow in this chapter. Appendix B provides a condensed description of the calling interface, functions, inputs, and returned values of each macro.

An extension to the SPI interface includes several additional macros that enable the host to receive an asynchronous event notification from a target on the SCSI bus. Section 3.8 describes the asynchronous event notification (AEN) feature in greater detail, and introduces each of the macros in the SPI interface extension.

Writing a VMS SCSI Class Driver

3.1 SCSI Class/Port Architecture

3.1.2 SCSI-Specific Data Structures

The SCSI class/port interface must maintain status and control information relevant to each participating connection and port. Moreover, SCSI class drivers and port drivers require a means of sharing information about each I/O request that involves the port. The following data structures accommodate these needs:

- SCSI connection descriptor table (SCDT)
- SCSI port descriptor table (SPDT)
- SCSI class driver request packet (SCDRP)
- Device and port unit control blocks (UCBs)

The *SCSI connection descriptor table* (SCDT) contains information specific to a connection established between a SCSI class driver and the port, such as phase records, timeout values, and error counters. The SCSI port driver creates an SCDT each time a SCSI class driver, by invoking the `SPI$CONNECT` macro, connects to a device on the SCSI bus. The class driver stores the address of the SCDT in the SCSI device's UCB.

The port driver has exclusive access to the SCDT; it is not accessed by the class driver. (The structure of the SCDT is illustrated in Figure A-2; its contents are described in Table A-2.)

The *SCSI port descriptor table* (SPDT) contains information specific to a SCSI port, such as the port driver connection database. The SPDT also includes a set of vectors, corresponding to the SPI macros invoked by SCSI class drivers, that point to service routines within the port driver. The SCSI port driver's unit initialization routine creates an SPDT for each SCSI port defined for a specific MicroVAX/VAXstation system and initializes each SPI vector.

The port driver reads and writes fields in the SPDT. The class driver does not write SPDT fields, but reads the SPDT indirectly when it invokes an SPI macro. (The structure of the SPDT is illustrated in Figure A-3; its contents are described in Table A-3.)

A SCSI class driver creates a *SCSI class driver request packet* (SCDRP) to deliver to the port driver information specific to an I/O request, such as the address of the SCSI command descriptor buffer. The class driver also places in the SCDRP some of the data it originally received in the I/O-request packet (IRP), such as the `$QIO` system service parameters, the I/O function, and the length and location of any user-specified buffer involved in a transfer. The port driver returns the actual data transfer byte count and status information to the class driver in the SCDRP.

Both class and port drivers read and write fields in the SCDRP; the port driver may modify fields written by the class driver. (The structure of an SCDRP is illustrated in Figure A-1; its contents are described in Table A-1.)

Writing a VMS SCSI Class Driver

3.1 SCSI Class/Port Architecture

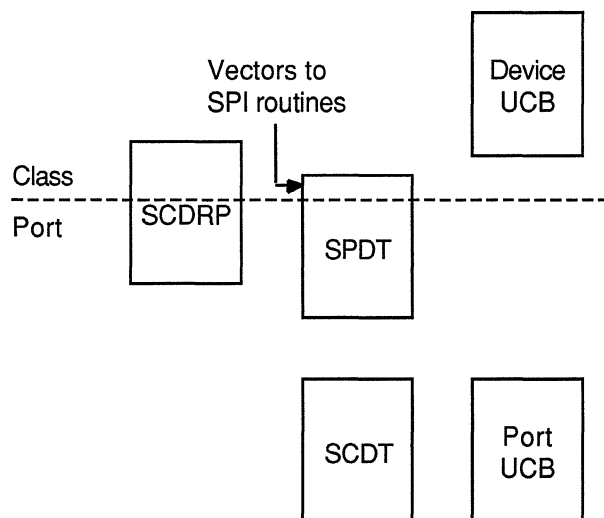
Two unit control blocks (UCBs) are involved in any interaction between the class driver and the port. The SCSI class driver maintains information in the *SCSI device UCB*, such as the device type, class, and characteristics; maximum transfer size; the address of the current SCDRP; and the addresses of the associated SPDT and SCDT. The SCSI port driver maintains similar information in the *SCSI port UCB*.

Table 3-2 summarizes the ownership of and access to these structures; their interrelationships are pictured in Figure 3-4.

Table 3-2 Data Structures

Structure	Allocation	Owner	Port Access	Class Access
SCDRP	One per I/O transfer request	Class driver	Read/write	Read/write
SCDT	One per SCSI connection	Port driver	Read/write	None
SPDT	One per SCSI port	Port driver	Read/write	Read
SCSI device UCB	One per SCSI device unit	Class driver	None	Read/write
SCSI port UCB	One per SCSI controller port	Port driver	Read/write	None

Figure 3-4 SCSI Driver Data Structures



ZK-1375A-GE

Writing a VMS SCSI Class Driver

3.1 SCSI Class/Port Architecture

3.1.3 SCSI Template Class Driver

The VMS operating system supplies a model for a third-party class driver in the SCSI template class driver, located in `SYS$EXAMPLES:SKDRIVER.MAR` and listed in Appendix C.

SKDRIVER is a simplified, self-documenting driver that supports the I/O functions `IO$_AVAILABLE`, `IO$_DIAGNOSE`, and `IO$_READLBLK` on a generalized device. SKDRIVER performs most operations required of a typical SCSI class driver to process a typical I/O request, including the appropriate SPI interface macro calls to establish a connection to the port, allocate port resources, and accomplish a transfer to the SCSI device. SKDRIVER also allocates pool for two SCDRPs. It uses one to send a SCSI command to the device, and it uses the other to issue a SCSI REQUEST SENSE command in the event the SCSI device returns failure status on the original command (see Section 3.4.6.2 for information on the interpretation of port and SCSI status return values).

In addition, SKDRIVER defines local macros that simplify common operations, including the following:

- Preserving register contents and return addresses within the class driver across calls to executive and port routines that may destroy this context (`INIT_UCB_STACK`, `SUBPOP`, `SUBPUSH`, `SUBRETURN`).
- Assembling the information relevant to a supported SCSI command such that a driver routine can easily construct the SCSI command descriptor buffer and initialize the SCDRP fields describing transfer buffer characteristics and timeout values (`SCSI_CMD`). SKDRIVER uses this macro to define the SCSI commands TEST UNIT READY, INQUIRY, REQUEST SENSE, and MODE SELECT. (It “invents” a fifth command, QIO INQUIRY, to provide a device-independent read operation servicing an `IO$_READLBLK` I/O function.)

SKDRIVER extends the device UCB to accommodate its context-saving macros, the allocation of SCDRPs, and per-request timeout values (see Section 3.3.2), SCSI-specific device characteristics, and the addresses of the SCDT and the current SCDRP.

Code in the template SCSI class driver can serve as a good starting point for the development of a third-party SCSI class driver. Subsequent sections of this chapter refer to the template SCSI class driver, as appropriate, to explain certain class driver concepts or possible implementation strategies.

3.2 Connecting to a SCSI Device

As defined by the VMS SCSI class/port interface, a *connection* is a logical link between a SCSI class driver and a SCSI device unit. In MicroVAX/VAXstation systems, a SCSI device is identified by its device mnemonic (for instance, *SK*), its SCSI port ID (*A* or *B*), its SCSI device ID (an integer from 0 to 7), and its logical unit number (an integer from 0 to 7).

Writing a VMS SCSI Class Driver

3.2 Connecting to a SCSI Device

Before a SCSI class driver can issue commands to a target device on the SCSI bus and transfer data across the bus, it must establish a logical connection to that device. The `SPI$CONNECT` macro connects a SCSI class driver with a target device, thereby establishing a linkage between the SCSI class driver and the SCSI port driver. Once the SCSI connection exists, the class and port drivers can intercommunicate.

A SCSI class driver's unit initialization routine invokes the `SPI$CONNECT` macro at fork level, specifying the SCSI port ID (in numeric form), the SCSI device ID, and the SCSI logical unit number of the device to which it needs to connect. (More detailed information about the use and functions of the `SPI$CONNECT` macro appears in Section 3.7.6 and Appendix B.)

Normally a connection lasts throughout the runtime life of a system; a SCSI class driver should never need to break a connection.

3.3 Setting Up a SCSI Command

This section describes the procedures a SCSI class driver follows to set up a SCSI command for transmission to the SCSI port driver. Although it discusses the aspects of the setup of a data transfer over the SCSI bus that relate to the preparation of a SCSI command, you should refer to Section 3.4 for a more complete discussion of SCSI data transfers.

3.3.1 Preparing a SCSI Command Descriptor Block

In preparation for sending a SCSI command to a device on the SCSI bus, a SCSI class driver must first determine which SCSI commands it supports. For each supported SCSI command, the driver programmer must perform the following tasks:

- Determine the correct size and format for the command
- Define the appropriate contents for all command bytes
- Allocate a SCSI port command buffer to make the command descriptor block and status buffer available to the port
- Create a SCSI command descriptor block in the SCSI port command buffer
- Create a 1-byte SCSI status buffer in the SCSI port command buffer
- Establish pointers in the `SCDRP` to the command descriptor block and the status buffer
- If the command involves a data transfer, store the parameters of the transfer in the `SCDRP`

The SCSI template class driver (`SKDRIVER`) performs these operations by means of the locally-defined `SCSI_CMD` macro and the `SETUP_CMD` subroutine. Each invocation of the `SCSI_CMD` macro establishes a data area within the driver to contain information about a specific SCSI command, including its length and the contents of its command bytes, plus the size, direction, and timeout values for any associated data transfer. The SCSI template class driver uses the `SCSI_CMD` macro to define

Writing a VMS SCSI Class Driver

3.3 Setting Up a SCSI Command

the parameters of five 6-byte SCSI commands, although the macro can describe commands of any length.

The `SETUP_CMD` subroutine of the template class driver `SK_STARTIO` routine repackages command data into a SCSI command descriptor block. Because both the command descriptor block and the SCSI status buffer must be accessed by both the class and port drivers, it is useful to account for the status buffer in the request to allocate the SCSI port command buffer. Thus, the `SETUP_CMD` subroutine adds 2 longwords of overhead—one for the SCSI status byte and one for the SCSI command size—to the SCSI command size. It then invokes `SPI$ALLOCATE_COMMAND_BUFFER`, causing the port driver to allocate a port command buffer and return its address and size.

The class driver initializes the status longword to `-1` and stores its address in `SCDRP$L_STS_PTR`. It places the size (in bytes) of the SCSI command in the next longword, and then constructs a SCSI command descriptor block in the buffer, copying the command to the buffer byte by byte. It places the address of the size longword in `SCDRP$L_CMD_PTR`.

Prior to invoking `SPI$SEND_COMMAND` to transmit the command descriptor block to the port driver, the class driver may perform several optional tasks to set up a data transfer operation, such as the following:

- Initializing the phase change (DMA) timeout and/or disconnect timeout fields in the `SCDRP` (`SCDRP$L_DMA_TIMEOUT` and `SCDRP$L_DISCON_TIMEOUT`), thus providing command-specific timeout values (see Section 3.3.2 for information on how to set up timeout values)
- For a data transfer involving a user buffer, initializing fields in the `SCDRP` to reflect the parameters of the buffer, and acquiring a port mapping of that buffer
- For a data transfer requiring a system buffer, allocating the buffer from nonpaged pool, initializing fields in the `SCDRP` to reflect the parameters of the buffer, and acquiring a port mapping of that buffer

When the command has completed and the SCSI port command buffer is no longer required, the class driver checks the command status, as described in Section 3.4.6, and invokes the `SPI$DEALLOCATE_COMMAND_BUFFER` macro to deallocate the buffer.

3.3.2 Setting Command Timeouts

The SCSI port driver implements several timeout mechanisms, some governed by the ANSI SCSI specification and others required by VMS. The timeouts required by VMS include the following:

Writing a VMS SCSI Class Driver

3.3 Setting Up a SCSI Command

Timeout	Description
Phase change timeout	<p>Maximum number of seconds for a target to change the SCSI bus phase or complete a data transfer. (This value is also known as the <i>DMA timeout</i>.)</p> <p>Upon sending the last command byte, the port driver waits this many seconds for the target to change the bus phase lines and assert REQ (indicating a new phase). Or, if the target enters the DATA IN or DATA OUT phase, the transfer must be completed within this interval.</p>
Disconnect timeout	<p>Maximum number of seconds, from the time the initiator receives the DISCONNECT message, for a target to reselect the initiator so that it can proceed with the disconnected I/O transfer.</p>

The SCSI class driver is responsible for maintaining both of these timeout values. It has the following three options:

- Accepting a connection's default value. The default value for both timeouts is 4 seconds.
- Altering the connection's default value. To modify the default values, the class driver specifies nonzero values in the **phase change timeout** and **disconnect timeout** longwords of the connection characteristics buffer and invokes the `SPI$SET_CONNECTION_CHAR` macro.
- Establishing timeouts for individual commands that override the connection's default value. If, prior to invoking the `SPI$SEND_COMMAND` macro, the class driver supplies a nonzero value in either `SCDRP$L_DMA_TIMEOUT` or `SCDRP$L_DISCON_TIMEOUT`, the port driver uses that value, instead of the default, for the course of that data transfer.

3.3.3 Disabling Command Retry

The SCSI port driver implements a command retry mechanism, which is enabled on a given connection by default.

When the command retry mechanism is enabled, the port driver retries up to three times any I/O operation that fails during the `COMMAND`, `Message`, `Data`, or `STATUS` phases. For instance, if the port driver detects a parity error during the Data phase, it aborts the I/O operation, logs an error, and retries the I/O operation. It repeats this sequence twice more, if necessary. If the I/O operation completes successfully during a retry attempt, the port driver returns success status to the class driver. However, if all retry attempts fail, the port driver returns failure status to the class driver.

When command retry is enabled on a connection, a SCSI class driver can control the number of retries the port attempts by supplying nonzero values in the **command retry count**, **busy retry count**, **arbitration retry count**, and **select retry count** longwords of the connection characteristics buffer, and invoking the `SPI$SET_CONNECTION_CHAR` macro.

Writing a VMS SCSI Class Driver

3.3 Setting Up a SCSI Command

A SCSI class driver may need to disable the command retry mechanism under certain circumstances. For instance, repeated execution of a command on a sequential device may produce different results than are intended by a single command request. A tape drive could perform a partial write and then repeat the write without resetting the tape position.

A SCSI class driver can disable this mechanism by setting bit 1 of the **connection flags** longword of the connection characteristics buffer, and invoking the `SPI$SET_CONNECTION_CHAR` macro.

3.4 Performing a SCSI Data Transfer

This section describes the procedures a SCSI class driver follows to set up and accomplish a data transfer over the SCSI bus.

3.4.1 Setting the Data Transfer Mode

The SCSI bus defines two data transfer modes, asynchronous and synchronous. In asynchronous mode, for each REQ from a target there is an ACK from the host prior to the next REQ from the target. Synchronous mode allows higher data transfer rates by allowing a pipelined data transfer mechanism where, for short bursts (defined by the REQ-ACK offset), the target can pipeline data to an initiator without waiting for the initiator to respond.

A class driver can determine the transfer modes supported by a device from the port capabilities longword returned from its invocation of the `SPI$CONNECT` macro. Whether or not a port or a target device supports synchronous data transfers, it is harmless for a class driver to set up the connection to use such transfers. If synchronous mode is not supported, the port driver automatically uses asynchronous mode.

To use synchronous mode in a transfer, the programmer of a SCSI class driver must ensure that both the SCSI port and the SCSI device involved in the transfer support synchronous mode. The SCSI port of the MicroVAX 3520/3540 systems supports both synchronous and asynchronous transfers, whereas that of the MicroVAX/VAXstation 3100 supports only asynchronous transfers.

To set up a connection to use synchronous data transfer mode, the SCSI class driver specifies a nonzero value in the **synchronous** longword of the connection characteristics buffer, and invokes the `SPI$SET_CONNECTION_CHAR` macro. The driver can also control the protocol of synchronous data transfers by supplying nonzero values for the **transfer period** and **REQ-ACK offset** longwords of the connection characteristics buffer and invoking the macro.

3.4.2 Enabling Disconnection and Reselection

The ANSI SCSI specification defines a disconnection facility that allows a target device to yield ownership of the SCSI bus while seeking or performing other time-consuming operations. When a target disconnects from the SCSI bus, it sends a sequence of messages to the initiator that cause it to save the state of the I/O transfer in progress. Once this is done, the target releases the SCSI bus. When the target is ready to complete the operation, it reselects the initiator and sends to it another sequence of messages. This sequence uniquely identifies the target and allows the initiator to restore the context of the suspended I/O operation.

Whether disconnection should be enabled or disabled on a given connection depends on the nature and capabilities of the device involved in the transfer, as well as on the configuration of the system. In configurations where there is a slow device present on the SCSI bus, enabling disconnection on connections that transfer data to the device can increase bus throughput. By contrast, systems where most of the I/O is directed toward a single device for long intervals can benefit from disabling disconnection. By disabling disconnection when there is no contention on the SCSI bus, port drivers can increase throughput and decrease the processor overhead for each I/O transfer.

By default, the VMS class/port interface disables the disconnect facility on a connection. To enable disconnection, the SCSI class driver sets bit 0 of the **connection flags** longword of the connection characteristics buffer, and invokes the `SPI$SET_CONNECTION_CHAR` macro.

3.4.3 Determining the Maximum Data Transfer Size

There are two factors governing the maximum data transfer size that any given SCSI device can accommodate.

First, there is the maximum size supported by the device; this can be determined from an inspection of the device's functional specification. The SCSI class driver writes the maximum device byte count to the device's UCB (`UCB$L_MAXBCNT`), usually by invoking the `DPT_STORE` macro when initializing the driver prologue table (`DPT`).

Secondly, there is the maximum value supported by the SCSI port. The port driver returns this value to the class driver in response to the class driver's invocation of the `SPI$CONNECT` macro.

The class driver may need to adjust the value in `UCB$L_MAXBCNT` to reflect the smaller of the device-specific and port-specific values.

The class driver compares the value supplied in `IRP$L_BCNT` with `UCB$L_MAXBCNT` to determine whether to accept, reject, or segment an I/O data transfer request.

Writing a VMS SCSI Class Driver

3.4 Performing a SCSI Data Transfer

3.4.4 Initializing the SCDRP to Reflect Class Driver Data Buffering Mechanisms

A standard data transfer, using direct I/O, involves the buffer specified in the \$QIO system service call as the source or destination of the data involved in the transfer. Typically this buffer is in process space (P0 space) and mapped by the process's P0 page table. To access this buffer at elevated IPL, a driver calls a VMS-supplied FDT routine (such as EXE\$READ or EXE\$MODIFY) that locks the buffer into memory and returns the system virtual address of the first P0 page-table entry that maps the buffer. The servicing of the QIO_INQUIRY SCSI command by the SCSI template class driver follows this approach. (Note that the QIO_INQUIRY command is the means by which the template driver illustrates the transfer of data from a SCSI device to a process buffer. Ordinarily, for a specific SCSI device, a class driver would use a SCSI READ command.)

Other transfer operations may require that the class driver itself operate upon the contents of the data buffer, or maintain its own data buffer. For these operations, the class driver must allocate a system buffer from nonpaged pool. The servicing of the INQUIRY SCSI command by the SCSI template class driver follows this approach.

Depending upon the local buffering mechanism it uses to service an I/O request, a SCSI class driver must initialize the SCDRP with the parameters of the transfer. When a process buffer is involved in the transfer, the class driver initializes the following fields:

Field	Contents
SCDRP\$L_ABCNT	0
SCDRP\$W_FUNC	IRP\$W_FUNC
SCDRP\$W_STS	IRP\$W_STS
SCDRP\$L_MEDIA	IRP\$L_MEDIA
SCDRP\$L_SVAPTE	IRP\$L_SVAPTE
SCDRP\$W_BOFF	IRP\$W_BOFF
SCDRP\$L_BCNT	IRP\$L_BCNT
SCDRP\$L_PAD_COUNT	0
SCDRP\$L_SCSI_FLAGS	SCDRP\$V_S0BUF bit cleared

When a system buffer is involved in the transfer, the class driver initializes the following fields:

Field	Contents
SCDRP\$L_SVA_USER	System virtual address of system buffer
SCDRP\$L_SVAPTE	System virtual address of the system page table entry mapping the first page of the system buffer
SCDRP\$L_BCNT	Length of the transfer
SCDRP\$L_PAD_COUNT	0

Writing a VMS SCSI Class Driver

3.4 Performing a SCSI Data Transfer

Field	Contents
SCDRP\$W_BOFF	Byte offset within page
SCDRP\$W_STS	IRP\$V_FUNC set for a read operation; clear for a write operation
SCDRP\$L SCSI_FLAGS	SCDRP\$V_S0BUF bit set

3.4.5 Making a Class Driver Data Buffer Accessible to the Port

Regardless of the local buffering mechanism it requires to fulfill the I/O transfer, a SCSI class driver must make the buffer available to the SCSI port hardware. A SCSI class driver accomplishes this by invoking the `SPI$MAP_BUFFER` macro.

The `SPI$MAP_BUFFER` macro causes the SCSI port driver to reserve sufficient pages of the port's DMA buffer to accomplish the transfer, plus sufficient mapping resources, if required, to map the class driver's data buffer to system virtual addresses. Certain ports require this mapping so that the port driver can access a process space buffer when setting up or completing a transfer for the SCSI port. When the class driver initiates a write operation, the SCSI port driver uses its mapping resources to copy the data from the class driver's user or system data buffer to this intermediate DMA buffer, from which the SCSI port can access it. When the class driver initiates a read operation, the SCSI device transfers the data to the DMA buffer, from which the port driver copies it to the class driver's data buffer.

Other ports do not require this mapping and can access the class driver's data buffer using system page-table entries.

By convention, a SCSI class driver sets the `SCDRP$V_BUFFER_MAPPED` bit in `SCDRP$L SCSI_FLAGS` when it invokes `SPI$MAP_BUFFER` to map a buffer; if the buffer involved in the transfer is a system buffer, it also sets the `SCDRP$V_S0BUF` bit. The `SCDRP$V_S0BUF` flag prevents the `SPI$MAP_BUFFER` port routine from double-mapping a system buffer.

The `SPI$MAP_BUFFER` port routine initializes the following fields in the `SCDRP`:

Field	Contents
SCDRP\$L_SVA_USER	System virtual address of the system buffer. When the class driver's local buffer is a system buffer, the contents of this field are unchanged by <code>SPI\$MAP_BUFFER</code> .
SCDRP\$L_SPTTE_SVAPTE	System virtual address of the system page-table entry mapping the first page of the system buffer. When the class driver's local buffer is a system buffer, the contents of this field and <code>SCDRP\$L_SVAPTE</code> are identical.

Writing a VMS SCSI Class Driver

3.4 Performing a SCSI Data Transfer

Field	Contents
SCDRP\$W_NUMREG	Number of pages of the port's DMA buffer allocated for this transfer.
SCDRP\$W_MAPREG	Starting page number of the first DMA buffer page allocated for this transfer.

Once the SCSI command has been prepared, the SCSI class driver issues the command to the SCSI device by invoking the `SPI$SEND_COMMAND` macro.

When the data transfer has completed (or its failure has been serviced) and the port DMA buffer and mapping resources are no longer required, the class driver invokes the `SPI$UNMAP_BUFFER` macro to deallocate these resources.

3.4.6 Examining Port and SCSI Status

Whether a SCSI command completes or fails, the port driver returns to the class driver the following status values:

- Port status in `R0`
- SCSI command status in the low byte of the status buffer pointed to by `SCDRP$L_STS_PTR`
- Actual number of bytes transferred in `SCDRP$L_TRANS_CNT`

The class driver should examine these returned values to determine the success or failure of a SCSI command. If a SCSI command fails, the class driver can pursue its recovery or retry the command, depending upon the type and severity of the error and the nature of the device.

3.4.6.1 Examining Port Status

The port status is the primary indicator of the failure of a SCSI command; that is, if the port failed during command preparation or transmission, it is unlikely that the SCSI command status byte contains meaningful information.

The port driver returns one of the following status values in `R0`:

Status	Meaning
<code>SS\$_NORMAL</code>	Normal successful completion
<code>SS\$_TIMEOUT</code>	Failed during selection or arbitration
<code>SS\$_CTRLERR</code>	Controller error or port hardware failure
<code>SS\$_BADPARAM</code>	Bad parameter specified by the class driver
<code>SS\$_LINKABORT</code>	Connection no longer exists
<code>SS\$_DEVACTIVE</code>	Command outstanding on this connection

If `R0` contains anything but success status, the class driver may want to examine it for specific status values and attempt error recovery, retry the operation, or return a special error status to the original `$QIO` call. At

Writing a VMS SCSI Class Driver

3.4 Performing a SCSI Data Transfer

the very least, the class driver should log a device error, according to the method described in Section 3.5.2.

3.4.6.2 Examining the SCSI Status Byte

If the port driver returns `SS$_NORMAL` status in `R0`, the class driver should proceed to check the SCSI command status in the low byte of the longword buffer pointed to by `SCDRP$L_STS_PTR`.

The format of a SCSI status byte is illustrated in Table 3–3. Interpretation of the bits in this status byte is device-specific. The VMS SCSI template driver (`SKDRIVER`) first clears reserved bits 0, 6, and 7. It compares the resulting value with the `CHECK CONDITION` status value, to determine if `CHECK CONDITION` status has been returned.

Table 3–3 SCSI Status Byte Format

Bits of Status Byte ¹								Status Represented
7	6	5	4	3	2	1	0	
R	R	0	0	0	0	0	R	GOOD
R	R	0	0	0	0	1	R	CHECK CONDITION
R	R	0	0	0	1	0	R	CONDITION MET/GOOD
R	R	0	0	1	0	0	R	BUSY
R	R	0	1	0	0	0	R	INTERMEDIATE/GOOD
R	R	0	1	0	1	0	R	INTERMEDIATE/CONDITION MET/GOOD
R	R	0	1	1	0	0	R	RESERVATION CONFLICT
R	R	1	0	1	0	0	R	QUEUE FULL (not implemented)

¹All other codes reserved.

When `CHECK CONDITION` status is returned, `SKDRIVER` initiates a `REQUEST SENSE` SCSI command to determine the specific nature of the SCSI error. To do so, it must save the address of the `SCDRP` associated with the original command (in an extension to the device UCB), and allocate a new one for use with the `REQUEST SENSE` command. It prepares and issues the command according to the procedures described in Section 3.3. When the port driver returns status from the `REQUEST SENSE` command, `SKDRIVER` examines its status. If the port returns failure status or if the SCSI status byte has any error bit set, `SKDRIVER` completes the I/O request, deallocating both `SCDRPs` and its command and data buffers; and returns error status to the `$QIO` system service.

If the port returns success status from the `REQUEST SENSE` command, `SKDRIVER` examines the request sense key in its local system buffer (at `SCDRP$L_SVA_USER`). The actions of any class driver in response to any specific request sense key are device specific. `SKDRIVER` merely translates the value into a VMS success or failure status code and returns this code in `R0`. For sense keys indicating fatal errors, `SKDRIVER` logs a device error.

Writing a VMS SCSI Class Driver

3.4 Performing a SCSI Data Transfer

3.4.6.3 Testing for Incomplete Transfers

If both the port status value and the SCSI command status byte indicate successful completion, the class driver performs one last test to determine the success of any data transfer associated with the SCSI command.

The port driver returns the actual number of bytes transferred during command processing in `SCDRP$L_TRANS_CNT`. The class driver should compare the value in this field with the requested transfer size in `SCDRP$L_BCNT`. If they are not equal, the class driver may return successfully or investigate a possible error.

3.5 Other SCSI Class Driver Issues

The writer of a third-party SCSI class driver must deal with several issues that are not specifically related to the tasks of setting up a SCSI command or data transfer, but rather relate to the definition of the class/port interface. Among these issues are the following:

- Preserving the local context of the driver across calls to the port driver
- Logging errors detected by the class driver

Subsequent sections discuss each of these issues in detail.

3.5.1 Preserving Local Context

VMS SCSI port drivers contain routines that execute in response to a class driver's invocation of an SPI macro. A class driver should take into account the fact that any SPI macro invocation may cause the port driver routine to fork or stall while waiting for a port resource, and return to its caller's caller. These actions eradicate the local context of the class driver at the time it invoked the macro.

Therefore, a SCSI class driver routine must take special steps to ensure the following:

- The address of its caller is on the top of the stack.
- All significant local context currently in registers is preserved.
- Any local return address currently on the stack is preserved.

The SCSI template class driver (`SKDRIVER`) resolves these needs by allocating a 10-longword stack within its extension to the SCSI device UCB. The symbol `UCB$L_STACK_PTR` functions as a stack pointer. The template class driver defines macros that initialize the UCB stack (`INIT_UCB_STACK`), push and pop registers (or data) from the UCB stack (`SUBPUSH` and `SUBPOP`), push the return address from the top of the interrupt stack onto the UCB stack (`SUBSAVE`), and pop the return address from the UCB stack onto the interrupt stack and RSB (`SUBRETURN`).

CAUTION: The class driver must be careful not to overflow its local stack. Unless it takes precautions, it could overwrite data integral to a transfer in progress and cause unpredictable results.

Writing a VMS SCSI Class Driver

3.5 Other SCSI Class Driver Issues

Prior to calling any routine that may destroy its context, the template class driver issues a SUBSAVE to preserve its return address (before any additional data is pushed on the interrupt stack), and invokes the SUBPUSH macro for each register that must be preserved across the call. When execution in the class driver resumes, the driver issues the SUBPOP macro to restore the saved registers and the SUBRETURN macro to return to its caller.

3.5.2 Error Logging

A SCSI class driver establishes error logging and uses the system error logging routines (ERL\$DEVICERR, ERL\$DEVICTMO, and ERL\$DEVICEATTN) as described in the *VMS Device Support Manual*.

The VMS SCSI class/port interface defines SCSI port-driver and SCSI class-driver extensions to the error message buffer, which are interpreted and formatted by the VMS Error Log Utility. A SCSI class driver and the associated port driver log errors independently, each supplying SCSI-specific information as defined in its extension to the error message buffer.

The class driver extension to the error message buffer includes the information listed in Table 3-4.

Table 3-4 Error Message Buffer Extension for SCSI Class Drivers

Field	Length (in bytes)	Contents															
Longword count	4	Number of longwords that follow in the error message buffer (not including this one).															
Revision	1	Revision level of the error message buffer. The class driver must set this field to 1.															
Hardware revision	4	Hardware revision information, returned by the SCSI INQUIRY command in ASCII format.															
Error type	1	Type of error detected by the class driver. A SCSI class driver defines device-specific error types according to the nature of the device it services. The following error types are used by the VMS disk and tape class drivers and, as such, have defined values that are interpreted by the VMS Error Log Utility: <table border="1" style="margin-left: 20px; width: 100%;"> <thead> <tr> <th style="text-align: left;">Error</th> <th style="text-align: left;">Name</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>01</td> <td>CON_ERR</td> <td>Attempt to connect to the port driver failed.</td> </tr> <tr> <td>02</td> <td>MAP_ERR</td> <td>Attempt to map a user buffer failed.</td> </tr> <tr> <td>03</td> <td>SND_ERR</td> <td>Attempt to send a SCSI command failed.</td> </tr> <tr> <td>04</td> <td>INV_INQ</td> <td>Invalid inquiry data was received.</td> </tr> </tbody> </table>	Error	Name	Description	01	CON_ERR	Attempt to connect to the port driver failed.	02	MAP_ERR	Attempt to map a user buffer failed.	03	SND_ERR	Attempt to send a SCSI command failed.	04	INV_INQ	Invalid inquiry data was received.
Error	Name	Description															
01	CON_ERR	Attempt to connect to the port driver failed.															
02	MAP_ERR	Attempt to map a user buffer failed.															
03	SND_ERR	Attempt to send a SCSI command failed.															
04	INV_INQ	Invalid inquiry data was received.															

(continued on next page)

Writing a VMS SCSI Class Driver

3.5 Other SCSI Class Driver Issues

Table 3-4 (Cont.) Error Message Buffer Extension for SCSI Class Drivers

Field	Length (in bytes)	Contents															
		<table border="1"><thead><tr><th>Error</th><th>Name</th><th>Description</th></tr></thead><tbody><tr><td>05</td><td>EXT_SNS_DAT</td><td>Extended sense data was returned from the SCSI device.</td></tr><tr><td>06</td><td>INV_MOD_SNS</td><td>Invalid mode sense data returned from the SCSI device.</td></tr><tr><td>07</td><td>REASSIGN_BLK</td><td>Reassign block.</td></tr><tr><td>08</td><td>DIAG_DATA</td><td>Invalid diagnostic data returned to the VMS SCSI tape class driver.</td></tr></tbody></table>	Error	Name	Description	05	EXT_SNS_DAT	Extended sense data was returned from the SCSI device.	06	INV_MOD_SNS	Invalid mode sense data returned from the SCSI device.	07	REASSIGN_BLK	Reassign block.	08	DIAG_DATA	Invalid diagnostic data returned to the VMS SCSI tape class driver.
Error	Name	Description															
05	EXT_SNS_DAT	Extended sense data was returned from the SCSI device.															
06	INV_MOD_SNS	Invalid mode sense data returned from the SCSI device.															
07	REASSIGN_BLK	Reassign block.															
08	DIAG_DATA	Invalid diagnostic data returned to the VMS SCSI tape class driver.															
SCSI ID	1	SCSI ID of the device to which the current command was sent. The SCSI ID is an integer between 0 and 7.															
SCSI LUN	1	SCSI LUN of the device to which the current command was sent. The SCSI LUN is an integer between 0 and 7.															
SCSI SUBLUN	1	Not used. This field always contains 0.															
Port status	4	Longword status returned in R0 from the port driver. A value of -1 in this field indicates that there is no valid data in this field.															
SCSI CMD	<i>n</i>	Current SCSI command bytes. The SCSI command bytes are preceded by a byte containing the length of the command.															
SCSI STS	1	Current SCSI status byte. A status byte of -1 in this field indicates that the status byte does not yet contain valid information.															
Additional data	<i>n</i>	Additional data, preceded by a byte count of the data. A class driver defines what additional data would be meaningful in an error log entry based on the type of device it services. Additional data is displayed by the VMS Error Log Utility as untranslated longwords.															

Prior to calling `ERL$DEVICERR` to log an error associated with device activity, or `ERL$DEVICEATTN` (or `ERL$DEVICTMO`) to log an error on an inactive device, the class driver should perform the following tasks:

- Ensure that the `DDT$W_ERRORBUF` field contains a sufficient byte count to accommodate both the standard error message buffer and the SCSI class driver extension. The class driver can either supply this value in the `erlgbf` argument to the `DDTAB` macro or specifically initialize `DDT$W_ERRORBUF`.
- Initialize the device type (`UCB$B_DEVTYPE`) and device class (`UCB$B_DEVCLASS`) fields to `DT$_GENERIC_SCSI` and `DC$_MISC`. A driver normally initializes these fields by invoking the `DPT_STORE` macro. If the driver must use other device type or class values, or allows them to be changed by a user program, it may need to save and restore the real values of these fields temporarily across calls to the error logging routines.

`ERL$DEVICERR`, `ERL$DEVICTMO`, and `ERL$DEVICEATTN` all result ultimately in a call to the class driver's register dumping routine. The register dumping routine must supply all available information about the SCSI device error in the class driver's SCSI-specific extension to the error message buffer.

Writing a VMS SCSI Class Driver

3.5 Other SCSI Class Driver Issues

The VMS SCSI template class driver (SKDRIVER) defines the fields of this extension and contains a macro (LOG_ERROR) and a routine (ERROR_LOG) that are a useful basis for the implementation of error logging in a third-party SCSI class driver.

For a discussion of the interpretation of SCSI port and class driver error log entries, refer to Appendix D.

3.6 Flow of a Read I/O Request Through the SCSI Class and Port Drivers

This section describes a hypothetical read-I/O request to a SCSI device as it is serviced by a SCSI class driver and the port driver. The discussion assumes that the read operation is successful.

When it is loaded, the class driver performs a one-time initialization sequence as follows:

- 1 Its unit initialization routine invokes the SPI\$CONNECT macro. In response, the port driver forms a logical connection between the SCSI device's UCB and the target on the SCSI bus. The port driver creates an SCDT in which it inserts information describing the connection.
- 2 Its unit initialization routine optionally invokes the SPI\$SET_CONNECTION_CHAR macro to set the appropriate data transfer mode or timeout values, or to enable disconnection of the connection. In response, the port driver modifies the connection-specific characteristics it maintains in the SCDT.

When the class driver receives a read-I/O request, it performs the following operations:

- 1 Its read FDT routine verifies and interprets the parameters of the \$QIO system service call.
- 2 It calls a system FDT routine that locks the specified process buffer in memory.
- 3 When the request becomes current, the start-I/O routine dispatches to code that services the specified function.
- 4 Its start-I/O routine allocates and initializes an SCDRP and copies to it the fields from the IRP required by the port driver to complete the read operation.
- 5 Its start-I/O routine invokes the SPI\$ALLOCATE_COMMAND_BUFFER macro. In response, the port driver allocates a buffer suitable for a SCSI command descriptor buffer and a SCSI status byte.
- 6 Its start-I/O routine invokes the SPI\$MAP_BUFFER macro. In response, the port driver allocates the resources required to make the process buffer available to the port driver.
- 7 Its start-I/O routine builds a SCSI command in the command descriptor buffer, initializes the status byte, and invokes the SPI\$SEND_COMMAND macro to send the command to the port driver.

Writing a VMS SCSI Class Driver

3.6 Flow of a Read I/O Request Through the SCSI Class and Port Drivers

When the port driver receives the command, it sets up the connection characteristics (data transfer mode, timeout value, and disconnect mode) recorded in the SCDT, sends the command buffer to the device, and responds to changes in SCSI bus phases. The port driver performs the following specific actions:

- 1 It requests and obtains ownership of the port, stalling if necessary until the port is available.
- 2 It arbitrates for ownership of the SCSI bus.
- 3 It selects a target device on the SCSI bus and sends it an IDENTIFY message.
- 4 It waits for the bus COMMAND phase.
- 5 It sends the SCSI command descriptor buffer, byte by byte, to the target device.
- 6 It waits for a SCSI bus phase change. If the next phase is not DATA IN, the port driver proceeds with the next step. Otherwise, it accepts data from the target device as follows:
 - a. It sets up and starts a DMA transfer to the port's DMA buffer.
 - b. It saves its context in the port UCB and waits for the target device to interrupt, signifying the completion of the read request. If the target device does not interrupt, the port driver sets up error status and returns to the class driver.
- 7 It checks the SCSI bus phase. If the phase is unchanged, the port driver sets up the next transfer. If the phase is STATUS, the port driver reads the status and copies the status to the return status buffer.
- 8 It waits for the MESSAGE IN phase. When the phase changes to MESSAGE IN, the port driver reads the message. If the message is COMMAND COMPLETE, the port driver returns SS\$_NORMAL in R0. Otherwise, it returns the appropriate port status to the class driver.
- 9 It releases the port.
- 10 It transfers the data from the port's DMA buffer to the process buffer.
- 11 It returns to the class driver.

When it regains control from the port driver, the class driver performs the following tasks to complete the read operation:

- 1 It checks the port status in R0.
- 2 It checks the SCSI status in the SCSI status byte.
- 3 It checks that the actual transfer length agrees with the requested transfer length.
- 4 It invokes the SPI\$DEALLOCATE_COMMAND_BUFFER macro to deallocate the command buffer.

Writing a VMS SCSI Class Driver

3.6 Flow of a Read I/O Request Through the SCSI Class and Port Drivers

- 5 It invokes the `SPI$UNMAP_BUFFER` macro to release the port resources mapping the user buffer.
- 6 It initiates device-independent postprocessing of the request by invoking the `REQCOM` macro.

3.7 Components of a SCSI Class Driver

A SCSI class driver contains nearly all of the components of a traditional VMS driver, as described in the *VMS Device Support Manual*. These include the following:

- Data, macro, and constant definitions
- Driver prologue table
- Driver dispatch table
- Function decision table and FDT routines
- Controller initialization routine
- Unit initialization routine
- Start-I/O routine
- Cancel-I/O routine
- Error logging routine
- Register dumping routine

A SCSI class driver contains no interrupt service routine. Moreover, it has no access to device control and status registers (CSRs). It relies on the port driver to initiate operations on the device and to service device interrupts.

This section describes the special operations that must be performed by the components of a SCSI class driver. The standard and typical operations performed by driver routines and tables are discussed in the *VMS Device Support Manual*.

3.7.1 Data Definitions

A SCSI class driver must invoke the `$SCDRPDEF` data structure definition macros, located in `SYS$LIBRARY:LIB.MLB`. `$SCDRPDEF` defines the fields of the SCSI class driver request packet.

A SCSI class driver typically does not reference fields in the SCSI connection descriptor table and, thus, does not need to invoke the `$SCDTDEF` macro. Although fields in the SCSI port descriptor table are used by the SPI macros as vectors to routines in the port driver, a SCSI class driver need not explicitly define SPDT fields. It indirectly obtains the SPDT definitions through its invocation of the `SPI$CONNECT` macro; it is the macro that invokes `$SPDTDEF`.

Writing a VMS SCSI Class Driver

3.7 Components of a SCSI Class Driver

A SCSI class driver may define an extension to the device UCB for an internal stack or for managing the allocation of SCDRPs, depending upon the needs of the implementation. The VMS SCSI template driver (SKDRIVER), listed in Appendix C, illustrates uses of these additional UCB fields. SKDRIVER also defines symbols representing SCSI-specific data buffer offsets and status values.

3.7.2 Driver Prologue Table

A SCSI class driver must supply the NULL keyword as the **adapter** argument to the DPTAB macro. It also must specify that the DPT\$V_NO_IDB_DISPATCH flag is set in the **flags** argument. The DPT\$V_NO_IDB_DISPATCH flag indicates that the IDB\$L_UCBLIST field is not used to store the addresses of UCBs for this device.

If the class driver implements error logging, it should use the DPT_STORE macro to initialize UCB\$B_DEVTYPE to DT\$_GENERIC SCSI and UCB\$B_DEVCLASS to DC\$_MISC. If the class driver must use other device type or class values, or allows them to be changed by a user program, it may need to save and restore the real values of these fields temporarily across calls to the error logging routines.

A SCSI class driver should not initialize CRB\$L_INTD+VEC\$L_ISR or the other interrupt vectors in the CRB. A SCSI device interrupts through a vector serviced by the port driver; any interrupt service routine specified by the SCSI class driver is not used.

3.7.3 Driver Dispatch Table

There are no special requirements for a SCSI class driver's driver dispatch table.

3.7.4 Function Decision Table and FDT Routines

There are no special requirements for a SCSI class driver's function decision table.

A class driver invokes FDT routines to preprocess I/O functions in a device-specific manner. Most SCSI class drivers use the standard VMS-supplied FDT routines (such as EXE\$READ, EXE\$WRITE, and EXE\$SETMODE). However, some class drivers may need to include a special FDT routine. The VMS SCSI template class driver illustrates this approach.

3.7.5 Controller Initialization Routine

There are no special requirements for a SCSI class driver's controller initialization routine.

Writing a VMS SCSI Class Driver

3.7 Components of a SCSI Class Driver

3.7.6 Unit Initialization Routine

A SCSI class driver's unit initialization routine must perform several special actions, as follows:

- It checks the power failure bit (UCB\$V_POWER) in UCB\$W_STS to determine whether it is being called in the course of power failure recovery. If this bit is set, the unit initialization routine returns immediately.
- It forks twice, issuing the FORK macro twice in succession. The first fork ensures, during system initialization or autoconfiguration, that the SCSI port driver's initialization routines begin execution before the class driver performs its initialization. The second fork guarantees that a port driver initialization fork thread has created its SPDTs and initialized the SCSI ports.

Note that the unit initialization routine must be executing at fork IPL when it invokes the SPI\$CONNECT macro.

- It prepares for an SPI\$CONNECT request by obtaining the SCSI port ID, the SCSI device ID, and the SCSI logical unit number (LUN).

SCSI device unit numbers have the form $d0u$, where d is the device ID and u is the LUN. The unit initialization routine obtains the SCSI device unit number from UCB\$W_UNIT and divides it by 100 (using the EDIV instruction). The quotient (in R1) is the device ID and the remainder (in R2) is the LUN. Both should be values between 0 and 7.

SCSI port IDs are represented by the alphabetic characters A and B . The unit initialization routine obtains this letter from the third byte in DDB\$T_NAME (for instance, A , from SKA500) and converts it to the numeral 0 or 1.

Once it has obtained the SCSI port ID, the SCSI device ID, and the SCSI LUN, the unit initialization routine sets up the registers for the call to SPI\$CONNECT as follows:

ID	Destination
SCSI port ID	Low-order word of R1
SCSI device ID	High-order word of R1
SCSI LUN	High-order word of R2

- It invokes the SPI\$CONNECT macro. The port driver, as a result, attempts to create a connection between the class driver and the port.

If the class driver expects notification of asynchronous events from the target device, it supplies the address of a local callback routine in the **callback** argument of the SPI\$CONNECT macro. (For a discussion of asynchronous event notification (AEN) mode, see Section 3.8.)

- If the port driver returns failure status, the unit initialization routine sets the device off line.

Writing a VMS SCSI Class Driver

3.7 Components of a SCSI Class Driver

- If the port driver successfully creates the connection, the unit initialization routine initializes UCB\$L_MAXBCNT, UCB\$L_SCDT, and UCB\$L_PDT with the values returned by the port driver, sets the device unit on line (by setting UCB\$V_ONLINE in UCB\$W_STS), and returns success status to its caller.

The VMS template SCSI class driver (SKDRIVER) unit initialization routine performs such optional actions as setting up an internal stack in the UCB for context-saving purposes, and allocating nonpaged pool for a set of SCDRPs to be queued to the UCB for use by the driver's start-I/O routine. See Appendix C for a listing of the template SCSI class driver.

The unit initialization routine may also invoke the SPI\$GET_CONNECTION_CHAR and SPI\$SET_CONNECTION_CHAR macros to examine (and possibly alter) the current data transfer mode, timeout, command retry, and disconnect characteristics of the SCSI connection. (See Section 3.4 for additional information.)

3.7.7 Start-I/O Routine

A SCSI class driver's start-I/O routine must perform the following steps to prepare a SCSI command for delivery to the port driver:

- Allocate an SCDRP from nonpaged pool. (The VMS template SCSI class driver allocates SCDRPs in its unit initialization routine; its start-I/O routine simply removes a preallocated SCDRP from a queue in the device UCB.)
- Insert the address of the IRP in SCDRP\$L_IRP.
- Dispatch to a function-specific command preparation routine.

The command preparation routine performs the procedures described in Section 3.3 and Section 3.4. Its actions typically involve the following:

- Invoking SPI\$ALLOCATE_COMMAND_BUFFER to allocate a port command buffer in which it assembles the SCSI command and reserves a longword for the SCSI status byte to be returned from command execution.
- Initializing fields in the SCDRP from the corresponding fields in the IRP. For read-I/O functions, the class driver must ensure that IRP\$V_FUNC is set in SCDRP\$W_STS.
- Invoking SPI\$MAP_BUFFER to make data in the process buffer available to the port, and setting the SCDRP\$V_BUFFER_MAPPED bit in SCDRP\$L SCSI_FLAGS to indicate that the buffer has been mapped to the port. (If it maps a system buffer, it must set both the SCDRP\$V_S0BUF and the SCDRP\$V_BUFFER_MAPPED bits.)
- Invoking SPI\$SEND_COMMAND to deliver the SCSI command to the port driver.
- When the command completes, examining the port status, SCSI status, and transfer count to determine the success or failure of the I/O operation. (See Section 3.4.6 for a detailed description of the means by which a SCSI class driver typically responds to status information.)

Writing a VMS SCSI Class Driver

3.7 Components of a SCSI Class Driver

If the operation fails, the class driver may take steps to obtain additional status information from the target device, pursue error recovery and retry the operation, enter a device-specific message in the error log buffer, return error status to the \$QIO system service, or perform some combination of these actions.

- Invoking SPI\$UNMAP_BUFFER to release port mapping resources.
- Invoking SPI\$DEALLOCATE_COMMAND_BUFFER to deallocate the port command buffer.
- Deallocating the SCDRP.
- Initiating device-independent postprocessing by invoking the REQCOM macro.

3.7.8 Cancel-I/O Routine

If a SCSI class driver receives a cancel request for an I/O operation in progress on a SCSI device, its cancel-I/O routine may invoke the SPI\$ABORT_COMMAND macro to terminate the I/O operation.

Note: VAXstation 3520/3540 systems do not implement the abort-SCSI-command function.

3.7.9 Register Dumping Routine

A SCSI class driver's register dumping routine executes in the course of a driver error logging operation. The class driver calls ERL\$DEVICERR, ERL\$DEVICTMO, or ERL\$DEVICEATTN, and the system error logging routine calls the driver's register dumping routine.

The register dumping routine loads the error message buffer with all available information about a SCSI device error in the buffer extension reserved for SCSI class driver information (see Table 3-4). Detailed information on SCSI class driver error logging appears in Section 3.5.2.

3.8 Servicing Asynchronous Events from a SCSI Device

Devices can perform one of two roles on the SCSI bus; either the target role or the initiator role. Typically, the host processor serves as the initiator and peripheral devices serve as targets. However, some devices require that the host processor respond to an unsolicited event, such as when the device is selected or deselected. When such an event occurs, the target device must be capable of selecting the host and acting in the initiator mode.

Certain MicroVAX/VAXstation systems implement the SCSI asynchronous event notification (AEN) feature, allowing SCSI devices to act as initiators on given connections. When AEN is enabled and the host is selected by a target, the host

- Responds to selection
- Parses SCSI command packets

Writing a VMS SCSI Class Driver

3.8 Servicing Asynchronous Events from a SCSI Device

- Drives the SCSI bus phase as required by targets

The VMS SCSI class/port interface supports asynchronous event notification by the `SPI$CONNECT` macro and an extension to the SCSI port interface (SPI). Table 3–5 lists the SPI macros provided in the SPI extension.

Table 3–5 SPI Extension Macros Supporting Asynchronous Event Notification

<code>SPI\$FINISH_COMMAND¹</code>	Completes an I/O operation executing under the AEN feature
<code>SPI\$RECEIVE_BYTES</code>	Receives command, message, and data bytes from a device acting as an initiator
<code>SPI\$RELEASE_BUS¹</code>	Releases the SCSI bus
<code>SPI\$SEND_BYTES</code>	Sends command, message, and data bytes to a device acting as an initiator
<code>SPI\$SENSE_PHASE</code>	Reads the current SCSI bus phase
<code>SPI\$SET_PHASE</code>	Sets the SCSI bus phase

¹A SCSI class driver must invoke either the `SPI$FINISH_COMMAND` macro or the `SPI$RELEASE_BUS` macro (but not both) to complete an AEN operation.

To utilize asynchronous event notification, a SCSI class driver's unit initialization routine must provide a callback address in the call to the `SPI$CONNECT` macro. The port driver invokes the callback routine at this address in response to selection by another device, passing to it the address of the SPDT in R4 and the address of the SCDRP in R5.

If the SCSI class driver does not provide a callback address, no selections are allowed on the connection that is established. If a selection does occur on a connection that is not set up to accommodate selections, the port driver attempts to send the BUS DEVICE RESET message to the device. If that fails, the port driver resets the SCSI bus.

The flow of an AEN operation is as follows:

- 1 The class driver connects to the port driver and provides the callback address.
- 2 The port driver receives a selection on an existing connection. If selections are allowed, the port driver calls the class driver at its callback address, holding the fork lock at IPL 8. R4 contains the address of the SCDT and R5 contains the address of the SCDRP.
- 3 The class driver invokes `SPI$SET_PHASE` to set the SCSI bus to COMMAND phase.

Because the target has selected the host, the host now becomes the target. In SCSI, the target drives the phase of the SCSI bus after selection. Thus, the class driver drives the SCSI bus to the COMMAND phase to receive the command bytes from the initiator.

Writing a VMS SCSI Class Driver

3.8 Servicing Asynchronous Events from a SCSI Device

- 4 The class driver invokes `SPI$RECEIVE_BYTES`.

Because command packets are variable in size, the class driver requests the first byte of the command to determine how many additional command bytes are to be expected.

- 5 The class driver again invokes `SPI$RECEIVE_BYTES`.

Once the class driver has determined exactly how many command bytes are expected, it requests all remaining command bytes with this one call.

- 6 The class driver invokes `SPI$SET_PHASE` to set the SCSI bus phase to DATA IN.

After the class driver has received all the command bytes and parsed the command to identify it, the class driver sets the bus to the appropriate phase. For instance, if the command is a READ BUFFER, the class driver must set the bus phase to DATA IN.

- 7 The class driver invokes `SPI$SEND_BYTES` to return exactly the number of data bytes requested by the initiator.

- 8 The class driver invokes `SPI$FINISH_COMMAND`.

- 9 The class driver returns from its callback routine.

Once the data transfer has completed, the I/O operation is done from the class driver's perspective. The class driver completes the I/O operation by returning status and the COMMAND COMPLETE message to the device.

3.9 Configuring a SCSI Third-Party Device

The System Generation Utility (SYSGEN) loads a third-party SCSI class driver into system virtual memory, creates additional data structures for the device unit, and calls the driver's controller initialization routine and unit initialization routine. SYSGEN automatically loads and autoconfigures the SCSI port driver at system initialization. As part of autoconfiguration, SYSGEN polls each device on each SCSI bus. If the device identifies itself as a direct-access device, a direct-access CDROM device, or a flexible disk device, SYSGEN automatically loads the VMS disk class driver (DKDRIVER); if the device identifies itself as a sequential-access device, SYSGEN automatically loads the VMS tape class driver (MKDRIVER). If the autoconfiguration facility does not recognize the type of the SCSI device, it loads no driver.

Consequently, third-party SCSI devices must be configured and their drivers loaded by an explicit SYSGEN CONNECT command, as follows:

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> CONNECT mmpd0u /NOADAPTER
```

In this command, *mm* represents the device mnemonic (for instance, SK; *p* represents the SCSI port ID (for instance, the controller ID A or B); *d* represents the SCSI device ID (a digit from 0 to 7); 0 signifies the digit zero; and *u* represents the SCSI logical unit number (a digit from 0 to 7).

Writing a VMS SCSI Class Driver

3.9 Configuring a SCSI Third-Party Device

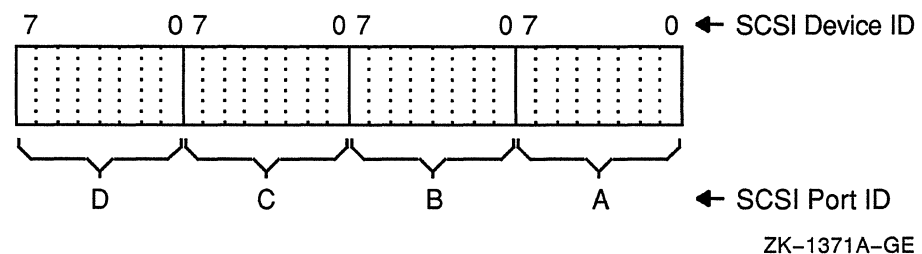
3.9.1 Disabling the Autoconfiguration of a SCSI Device

Note that, in special cases, you may need to prevent SYSGEN's autoconfiguration facility from loading the VMS disk or tape class driver for a device with a specific port ID and device ID. This would be the case if a third-party device should identify itself as either a random-access or sequential-access device and were to be supported by the VMS generic SCSI class driver.

To disable the loading of a VMS disk or tape driver for any given device ID, VMS Version 5.3 temporarily defines the special SYSGEN parameter **VMSD2**.

The **VMSD2** system parameter, as shown in Figure 3-5, stores a bit mask of 32 bits in which the low-order byte corresponds to the first SCSI bus (PKA0), the second byte corresponds to the second SCSI bus (PKB0), and so on. For each SCSI bus, setting the low-order bit inhibits automatic configuration of the device with SCSI device ID 0; setting the second low-order bit inhibits automatic configuration of the device with SCSI device ID 1, and so forth. For instance, the value 00002000_{16} would prevent the device with SCSI ID 5 on the bus identified by SCSI port ID B from being configured. By default, all of the bits in the mask are cleared, allowing all devices to be configured.

Figure 3-5 VMSD2 System Parameter



Note: A future release of VMS will provide a different mechanism for preventing the configuration of a VMS SCSI class driver for a given device ID. At that time, the VMSD2 system parameter will revert to its status of a special parameter reserved to Digital.

3.10 Debugging a SCSI Class Driver

VMS device drivers execute in kernel mode at elevated interrupt priority levels. Problems in device driver code often manifest themselves in system failures and system hangs. The *VMS Device Support Manual* describes some general methods for debugging device drivers that can also be used to debug a third-party SCSI class driver. While using the XDELTA debugger to investigate problems in a class driver, however, you should set breakpoints such that you can easily step over VMS SCSI port driver code.

Writing a VMS SCSI Class Driver

3.10 Debugging a SCSI Class Driver

As discussed in Section 3.5.2 and Appendix D, Digital strongly recommends that a third-party SCSI class driver respond to port and SCSI status return values, and that it incorporate an error logging routine that records events significant to the device. Class driver error log entries, as well as VMS port driver error log entries, can provide clues that are helpful in resolving problems that may occur during the development of a third-party SCSI class driver.

Among the problems that commonly occur in early versions of SCSI class drivers are the following:

- The class driver has failed to deallocate a port resource, such as a command buffer or port map registers. You should ensure that the class driver invokes the `SPI$DEALLOCATE_COMMAND_BUFFER` and the `SPI$UNMAP_BUFFER` macros before completing a data transfer (that is, before invoking the `REQCOM` macro).
- The class driver has sent a SCSI command to a device, but the device does not support the command. Typically, the device times out or the port driver logs an entry for a bad phase transition event.
- The class driver has sent a misformatted SCSI command packet to a device. This problem also results in a device timeout or phase error.

Hardware problems on a SCSI bus can cause a SCSI command to fail, regardless of whether the device to which the command was directed is at fault. When testing and debugging a class driver for a new device on a SCSI bus, you should ensure that bus traffic from busy or faulty devices elsewhere on a SCSI bus does not interfere with the device's operation. Isolate the device by placing it on a SCSI bus reserved for it and the processor alone or, if that is not possible, by placing it on the SCSI bus on which the system disk does *not* reside.

3.10.1 Selecting a SCSI Bus Analyzer

Finally, in debugging a SCSI class driver, you may find a SCSI bus analyzer to be a valuable aid.

A SCSI bus analyzer is a passive device that monitors all traffic on the SCSI bus to which it is connected, and displays in a useful format the data it has collected. Some analyzers can be used in an active mode to generate packets on the bus; however, this is generally more useful to developers of SCSI target devices than to writers of class drivers.

A SCSI bus analyzer is commonly used to verify that the commands the class driver generates (or should generate) are actually being transmitted across the SCSI bus. The most useful analyzers can interpret the SCSI phase lines and display the phase along with the data sent during that phase. This helps the writer of a class driver pinpoint the location of a possible coding problem. Another common use of an analyzer is to capture infrequent errors such as bus hangs or a target dropping off the bus.

Writing a VMS SCSI Class Driver

3.10 Debugging a SCSI Class Driver

Some features to look for in an analyzer are as follows:

- Ability to interpret the bus phase lines and display the data according to the phase
- A “timing mode” that displays bus signals in the form of a timing diagram
- Ability to trigger the analyzer on a specific event, such as a specific data pattern in a specific phase or a bus reset
- Ability to dump the contents of the display to a printer

A

SCSI Device Driver Data Structures

This appendix provides a condensed description of those data structures referenced by SCSI class driver code. It lists fields in the order in which they appear in the structures. All data structures discussed in this appendix exist in nonpaged system memory.

Notes: Driver code must consider fields marked by asterisks to be read-only fields. Fields marked *reserved* or *unused* are reserved for future use by Digital unless otherwise specified.

When referring to locations within a data structure, a driver should use symbolic offsets from the beginning of the structure and *not* numeric offsets. Numeric offsets are likely to change with each new release of the VMS operating system. The figures in this appendix list VMS Version 5.3 numeric offsets to aid in driver debugging.

A.1 SCSI Class Driver Request Packet (SCDRP)

The SCSI class driver allocates and builds a SCSI class driver request packet (SCDRP) for each I/O request it services, passing it to the SCSI port driver. The class driver routine initializes the SCDRP with the addresses of the UCB, SCDT, and IRP and copies to it data obtained from the IRP. The SCDRP also contains the addresses of the SCSI command buffer and status buffer.

The SCSI class driver passes the address of the SCDRP to the port driver in the call to SPI\$SEND_COMMAND.

The SCDRP is illustrated in Figure A-1 and described in Table A-1.

Figure A-1 SCSI Class Driver Request Packet (SCDRP)

SCDRP\$L_FQFL			0
SCDRP\$L_FQBL			4
SCDRP\$B_FLCK	SCDRP\$B_CD_TYPE	SCDRP\$W_SCDRPSIZE	8
SCDRP\$L_FPC			12
SCDRP\$L_FR3			16
SCDRP\$L_FR4			20
SCDRP\$L_PORT_UCB			24

(continued on next page)

SCSI Device Driver Data Structures

A.1 SCSI Class Driver Request Packet (SCDRP)

Figure A-1 (Cont.) SCSI Class Driver Request Packet (SCDRP)

SCDRP\$L_UCB		28
SCDRP\$W_STS	SCDRP\$W_FUNC	32
SCDRP\$L_SVAPE		36
reserved	SCDRP\$W_BOFF	40
SCDRP\$L_BCNT		44
SCDRP\$L_MEDIA		48
SCDRP\$L_ABCNT		52
SCDRP\$L_SAVD_RTN		56
reserved		60
SCDRP\$L_CDT		68
reserved		72
SCDRP\$L_IRP		76
SCDRP\$L_SVA_USER		80
SCDRP\$L_CMD_BUF		84
SCDRP\$L_CMD_BUF_LEN		88
SCDRP\$L_CMD_PTR		92
SCDRP\$L_STS_PTR		96
SCDRP\$L_SCSI_FLAGS		100
SCDRP\$L_DATACHECK		104
SCDRP\$L_SCSI_STK_PTR		108
≈	SCDRP\$L_SCSI_STK (32 bytes)	≈112
SCDRP\$L_CL_RETRY		144
SCDRP\$L_DMA_TIMEOUT		148
SCDRP\$L_DISCON_TIMEOUT		152

(continued on next page)

SCSI Device Driver Data Structures

A.1 SCSI Class Driver Request Packet (SCDRP)

Figure A-1 (Cont.) SCSI Class Driver Request Packet (SCDRP)

reserved	SCDRP\$W_PAD_BCNT	156
SCDRP\$B_TQE* (52 bytes)		160
SCDRP\$L_TQE_DELAY*		212
SCDRP\$L_SVA_DMA*		216
SCDRP\$L_SVA_CMD*		220
SCDRP\$W_CMD_MAPREG*	SCDRP\$W_MAPREG*	224
SCDRP\$W_CMD_NUMREG*	SCDRP\$W_NUMREG*	228
SCDRP\$L_SVA_SPT*		232
SCDRP\$L_SCSIMSGO_PTR*		236
SCDRP\$L_SCSIMSGI_PTR*		240
SCDRP\$B_SCSIMSGO_BUF*		244
SCDRP\$B_SCSIMSGI_BUF*		248
SCDRP\$L_MSGO_PENDING*		256
SCDRP\$L_MSGI_PENDING*		260
reserved	SCDRP\$B_LAST_MSGO*	264
SCDRP\$L_DATA_PTR*		268
SCDRP\$L_TRANS_CNT*		272
SCDRP\$L_SAVE_DATA_CNT*		276
SCDRP\$L_SAVE_DATA_PTR*		280
SCDRP\$L_SDP_DATA_CNT*		284
SCDRP\$L_SDP_DATA_PTR*		288
SCDRP\$L_DUETIME*		292
SCDRP\$L_TIMEOUT_ADDR*		296
SCDRP\$W_BUSY_RETRY_CNT*	SCDRP\$W_CMD_BCNT*	300

(continued on next page)

SCSI Device Driver Data Structures

A.1 SCSI Class Driver Request Packet (SCDRP)

Figure A-1 (Cont.) SCSI Class Driver Request Packet (SCDRP)

SCDRP\$W_SEL_RETRY_CNT*	SCDRP\$W_ARB_RETRY_CNT*	304
SCDRP\$W_SEL_TQE_RETRY_CNT*	SCDRP\$W_CMD_RETRY_CNT*	308
SCDRP\$L_SAVER3*		312
SCDRP\$L_SAVER6*		316
SCDRP\$L_SAVER7*		320
SCDRP\$L_SAVER3CL*		324
SCDRP\$L_SAVEPCCL*		328
SCDRP\$L_ABORTPCCL*		332
SCDRP\$L_PO_STK_PTR*		336
~	SCDRP\$L_PO_STK* (24 bytes)	~340
SCDRP\$L_TAG*		364
~	reserved (40 bytes)	~368

Table A-1 Contents of SCSI Class Driver Request Packet

Field Name	Contents
SCDRP\$L_FQFL	Fork queue forward link. This field points to the next entry in the SCSI adapter's command buffer wait queue (ADP\$L_BVPWAITFL), map register wait queue (ADP\$L_MRQFL), port wait queue (SPDT\$L_PORT_WQFL), or system fork queue.
SCDRP\$L_FQBL	Fork queue backward link. This field points to the previous entry in the SCSI adapter's command buffer wait queue (ADP\$L_BVPWAITFL), map register wait queue (ADP\$L_MRQFL), port wait queue (SPDT\$L_PORT_WQFL), or system fork queue.
SCDRP\$W_SCDRPSIZE	Size of SCDRP. A SCSI class driver, after allocating sufficient nonpaged pool for the SCDRP, writes the constant SCDRP\$C_LENGTH into this field.
SCDRP\$B_CD_TYPE	Class driver type. This field is currently unused.

(continued on next page)

SCSI Device Driver Data Structures

A.1 SCSI Class Driver Request Packet (SCDRP)

Table A-1 (Cont.) Contents of SCSI Class Driver Request Packet

Field Name	Contents
SCDRP\$B_FLCK	Index of the fork lock that synchronizes access to this SCDRP at fork level. A SCSI class driver, after allocating sufficient nonpaged pool for the SCDRP, copies to this field the value of UCB\$B_FLCK. All devices controlled by a single SCSI adapter and actively competing for shared adapter resources must specify the same value for this field.
SCDRP\$L_FPC	Address of instruction at which processing resumes when SCSI adapter resources become available to satisfy a request stalled in an adapter resource wait queue.
SCDRP\$L_FR3	Value of R3 when the request is stalled to wait for SCSI adapter resources. When the request is satisfied, this value is restored to R3 before the driver resumes execution at SCDRP\$L_FPC.
SCDRP\$L_FR4	Value of R4 when the request is stalled to wait for SCSI adapter resources. When the request is satisfied, this value is restored to R4 before the driver resumes execution at SCDRP\$L_FPC.
SCDRP\$L_PORT_UCB	SCSI adapter's UCB address. The SCSI port driver reads and writes this field in order to manage ownership of the SCSI port across bus reselection.
SCDRP\$L_UCB	SCSI device's UCB address. The SCSI class driver initializes this field to indicate that the SCDRP is active.
SCDRP\$W_FUNC	I/O function code that identifies the function to be performed for the I/O request. The SCSI class driver's start-I/O routine copies the contents of IRP\$W_FUNC to this field.
SCDRP\$W_STS	Status of I/O request. The SCSI class driver's start-I/O routine copies the contents of IRP\$W_STS to this field. Bits in the SCDRP\$W_STS field correspond to the bits in the IRP\$W_STS field that describe the type of I/O function, as follows:
	IRP\$V_BUFIO Buffered-I/O function
	IRP\$V_FUNC Read function
	IRP\$V_PAGIO Paging-I/O function
	IRP\$V_COMPLX Complex-buffered-I/O function
	IRP\$V_VIRTUAL Virtual-I/O function
	IRP\$V_CHAINED Chained-buffered-I/O function
	IRP\$V_SWAPIO Swapping-I/O function
	IRP\$V_DIAGBUF Diagnostic buffer present
	IRP\$V_PHYSIO Physical-I/O function
	IRP\$V_TERMIO Terminal I/O (for priority increment calculation)
	IRP\$V_MBXIO Mailbox-I/O function
	IRP\$V_EXTEND An extended IRP is linked to this IRP
	IRP\$V_FILACP File ACP I/O

(continued on next page)

SCSI Device Driver Data Structures

A.1 SCSI Class Driver Request Packet (SCDRP)

Table A-1 (Cont.) Contents of SCSI Class Driver Request Packet

Field Name	Contents
	IRP\$V_MVIRP Mount-verification I/O function
	IRP\$V_SRVIO Server-type I/O
	IRP\$V_KEY Encrypted function (encryption key address at IRP\$L_KEYDESC)
SCDRP\$L_SVAPTE	For a <i>direct-I/O</i> transfer, virtual address of the first page-table entry (PTE) of the I/O transfer buffer. This address is originally written to IRP\$L_SVAPTE by the FDT routine that locks process pages. For a <i>buffered-I/O transfer</i> , address of a buffer in system address space. This address is originally written to IRP\$L_SVAPTE by the class driver FDT routine that allocates the buffer. The class driver's start-I/O routine copies the address from the IRP to this field.
SCDRP\$W_BOFF	For a <i>direct-I/O</i> transfer, byte offset into the first page of the buffer; for a <i>buffered-I/O</i> transfer, number of bytes to be charged to the process requesting the transfer. FDT routines calculate this value and write it to IRP\$W_BOFF. The class driver's start-I/O routine copies the value from the IRP to this field.
SCDRP\$L_BCNT	Byte count of the I/O transfer. Class driver FDT routines calculate this value and write it to IRP\$L_BCNT. The class driver's start-I/O routine copies the value from the IRP to this field.
SCDRP\$L_MEDIA	Spare field.
SCDRP\$L_ABCNT	Accumulated count of bytes transferred. The SCSI class driver maintains this field to accomplish segmented transfers.
SCDRP\$L_SAVD_RTN	Saved return address from Level 1 JSB.
SCDRP\$L_CDT	Address of the SCSI connection descriptor table (SCDT). When the SCSI class driver's unit initialization routine invokes the SPI\$CONNECT macro, the macro returns the address of the SCDT describing the connection it established to the SCSI port. The class driver stores that address in SCDRP\$L_CDT.
SCDRP\$L_IRP	Address of I/O request block. The SCSI class driver copies the address of the IRP to this field.
SCDRP\$L_SVA_USER	System virtual address of a process buffer as mapped in system space (S0 space). The SCSI port driver initializes this field as the result of a class driver call to SPI\$MAP_BUFFER.
SCDRP\$L_CMD_BUF	Address of the port command buffer. The SCSI class driver initializes this field with the address returned from a call to SPI\$ALLOCATE_COMMAND_BUFFER.
SCDRP\$L_CMD_BUF_LEN	Length of SCSI command buffer.
SCDRP\$L_CMD_PTR	Address of the SCSI command descriptor block (its length byte) in the SCSI command buffer allocated by the SCSI port driver. The SCSI class driver initializes this field.

(continued on next page)

SCSI Device Driver Data Structures

A.1 SCSI Class Driver Request Packet (SCDRP)

Table A-1 (Cont.) Contents of SCSI Class Driver Request Packet

Field Name	Contents						
SCDRP\$L_STS_PTR	Address of SCSI status byte in the port command buffer. The SCSI class driver initializes this field.						
SCDRP\$L_SCSI_FLAGS	SCSI flags. The SCSI class and port drivers use the following bits:						
	<table style="width: 100%; border: none;"> <tr> <td style="width: 30%;">SCDRP\$V_S0BUF</td> <td>System buffer mapped. A SCSI class driver sets this bit, before invoking SPI\$MAP_BUFFER, if the data transfer buffer is in system space (S0).</td> </tr> <tr> <td>SCDRP\$V_BUFFER_MAPPED</td> <td>Data transfer buffer mapped. A SCSI class driver sets this bit, after invoking SPI\$MAP_BUFFER, to indicate that the data transfer buffer (either a system or process space buffer) has been mapped.</td> </tr> <tr> <td>SCDRP\$V_DISK_SPUN_UP</td> <td>START UNIT command issued. The VMS SCSI disk class sets this bit.</td> </tr> </table>	SCDRP\$V_S0BUF	System buffer mapped. A SCSI class driver sets this bit, before invoking SPI\$MAP_BUFFER, if the data transfer buffer is in system space (S0).	SCDRP\$V_BUFFER_MAPPED	Data transfer buffer mapped. A SCSI class driver sets this bit, after invoking SPI\$MAP_BUFFER, to indicate that the data transfer buffer (either a system or process space buffer) has been mapped.	SCDRP\$V_DISK_SPUN_UP	START UNIT command issued. The VMS SCSI disk class sets this bit.
SCDRP\$V_S0BUF	System buffer mapped. A SCSI class driver sets this bit, before invoking SPI\$MAP_BUFFER, if the data transfer buffer is in system space (S0).						
SCDRP\$V_BUFFER_MAPPED	Data transfer buffer mapped. A SCSI class driver sets this bit, after invoking SPI\$MAP_BUFFER, to indicate that the data transfer buffer (either a system or process space buffer) has been mapped.						
SCDRP\$V_DISK_SPUN_UP	START UNIT command issued. The VMS SCSI disk class sets this bit.						
SCDRP\$L_DATACHECK	Address of buffer for datacheck operations. A SCSI class driver maintains this field.						
SCDRP\$L_SCSI_STK_PTR	Stack pointer of the class driver's return address stack.						
SCDRP\$L_SCSI_STK	Class driver's return address stack. This stack is 32 bytes long.						
SCDRP\$L_CL_RETRY	Retry count.						
SCDRP\$L_DMA_TIMEOUT	<p>Maximum number of seconds for a target to change the SCSI bus phase or complete a data transfer.</p> <p>Upon sending the last command byte, the port driver waits this many seconds for the target to change the bus phase lines and assert REQ (indicating a new phase). Or, if the target enters the DATA IN or DATA OUT phase, the transfer must be completed within this interval.</p> <p>A class driver can initialize this field to specify a per-request DMA timeout value.</p>						
SCDRP\$L_DISCON_TIMEOUT	Maximum number of seconds, from the time the initiator receives the DISCONNECT message, for a target to reselect the initiator so that it can proceed with the disconnected I/O transfer. A class driver can initialize this field to specify a per-request disconnect timeout value.						
SCDRP\$W_PAD_BCNT	Pad byte count. This field contains the number of bytes required to make the size of the user buffer equal to the data length value required by a specific SCSI command. A SCSI class driver uses this field to accommodate SCSI device classes that require that the transfer length be specified in terms of a larger data unit than the count of bytes expressed in the SCDRP\$L_BCNT. If the total amount of data requested in the SCSI command does not match that specified in the SCDRP\$L_BCNT, this field must account for the difference.						

(continued on next page)

SCSI Device Driver Data Structures

A.1 SCSI Class Driver Request Packet (SCDRP)

Table A-1 (Cont.) Contents of SCSI Class Driver Request Packet

Field Name	Contents
SCDRP\$B_TQE*	Timer queue element, used by the port driver to time out pending disconnected I/O transfers. When this TQE expires, the timer thread times out expired pending I/O transfers.
SCDRP\$L_TQE_DELAY*	Delay time for next TQE delay.
SCDRP\$L_SVA_DMA*	System address of the section of the port DMA buffer allocated for the data transfer.
SCDRP\$L_SVA_CMD*	System address of the segment of the port DMA buffer allocated for the port command buffer.
SCDRP\$W_MAPREG*	Page number of the first port DMA buffer page allocated for the data transfer.
SCDRP\$W_CMD_MAPREG*	Page number of the first port DMA buffer page allocated for the port command buffer.
SCDRP\$W_NUMREG*	Number of port DMA buffer pages allocated for the data transfer.
SCDRP\$W_CMD_NUMREG*	Number of port DMA buffer pages allocated for the port DMA buffer.
SCDRP\$L_SVA_SPT*	System virtual address of the system page-table entry that maps the first page of the process buffer in S0 space.
SCDRP\$L_SCSIMSGO_PTR*	SCSI output message pointer.
SCDRP\$L_SCSIMSGI_PTR*	SCSI input message pointer.
SCDRP\$B_SCSIMSGO_BUF*	SCSI output message buffer.
SCDRP\$B_SCSIMSGI_BUF*	SCSI input message buffer.
SCDRP\$L_MSGO_PENDING*	Output message pending flags. One or more of the following bits are set in this longword if the port driver is to send the corresponding message: SCDRP\$V_IDENTIFY IDENTIFY message SCDRP\$V_SYNC_OUT SYNCHRONOUS DATA TRANSFER REQUEST (out) message SCDRP\$V_BUS_DEVICE_RESET BUS DEVICE RESET message SCDRP\$V_MESSAGE_PARITY_ERROR MESSAGE PARITY ERROR message SCDRP\$V_ABORT ABORT message SCDRP\$V_NOP NO OPERATION message SCDRP\$V_MESSAGE_REJECT MESSAGE REJECT message
SCDRP\$L_MSGI_PENDING*	Input message pending flags. The only currently defined bit is SCDRP\$V_SYNC_IN, which is set when the port driver expects to receive a SYNCHRONOUS DATA TRANSFER REQUEST (in) message.
SCDRP\$B_LAST_MSG0*	Last message sent.
SCDRP\$L_DATA_PTR*	Current data pointer address.
SCDRP\$L_TRANS_CNT*	Actual number of bytes sent or received by the port driver. The port driver returns a value in this field to the class driver when it completes a SCSI data transfer.

(continued on next page)

SCSI Device Driver Data Structures

A.1 SCSI Class Driver Request Packet (SCDRP)

Table A–1 (Cont.) Contents of SCSI Class Driver Request Packet

Field Name	Contents
SCDRP\$L_SAVE_DATA_CNT*	Running count of bytes (in two's-complement form) to be transferred. The port driver maintains this count.
SCDRP\$L_SAVE_DATA_PTR*	Pointer to current port DMA buffer segment. The SCSI port driver maintains this pointer.
SCDRP\$L_SDP_DATA_CNT*	Storage for SDP data count.
SCDRP\$L_SDP_DATA_PTR*	Storage for SDP data pointer.
SCDRP\$L_DUETIME*	Timeout time for a disconnected I/O transfer.
SCDRP\$L_TIMEOUT_ADDR*	Address of timeout routine.
SCDRP\$W_CMD_BCNT*	Command byte count.
SCDRP\$W_BUSY_RETRY_CNT*	Count of remaining busy retries.
SCDRP\$W_ARB_RETRY_CNT*	Count of remaining arbitration retries.
SCDRP\$W_SEL_RETRY_CNT*	Count of remaining selection retries.
SCDRP\$W_CMD_RETRY_CNT*	Count of remaining command retries.
SCDRP\$W_SEL_TQE_RETRY_CNT*	Count of remaining TQE retries.
SCDRP\$L_SAVER3*	Reserved to Digital.
SCDRP\$L_SAVER6*	Reserved to Digital.
SCDRP\$L_SAVER7*	Reserved to Digital.
SCDRP\$L_SAVER3CL*	Reserved to Digital.
SCDRP\$L_SAVEPCCL*	Reserved to Digital.
SCDRP\$L_ABORTPCCL*	Reserved to Digital.
SCDRP\$L_PO_STK_PTR*	Stack pointer of the port driver's return address stack.
SCDRP\$L_PO_STK*	Port driver's return address stack. This stack is 24 bytes long.
SCDRP\$L_TAG*	Reserved to Digital.

A.2 SCSI Connection Descriptor Table (SCDT)

The SCSI connection descriptor table (SCDT) contains information specific to a connection established between a SCSI class driver and the port, such as phase records, timeout values, and error counters. The SCSI port driver creates an SCDT each time a SCSI class driver, by invoking the `SPI$CONNECT` macro, connects to a device on the SCSI bus. The class driver stores the address of the SCDT in the SCSI device's UCB.

The SCSI port driver has exclusive access to the SCDT. A SCSI class driver has no access to this structure.

The SCDT is illustrated in Figure A–2 and described in Table A–2.

SCSI Device Driver Data Structures

A.2 SCSI Connection Descriptor Table (SCDT)

Figure A-2 SCSI Connection Descriptor Table (SCDT)

SCDT\$L_FLINK*		0
reserved	SCDT\$W_SIZE*	4
SCDT\$B_FLCK*	reserved	8
SCDT\$L_FPC*		12
SCDT\$L_FR3*		16
SCDT\$L_FR4*		20
SCDT\$L_STS*		24
SCDT\$W_STATE*	SCDT\$W_SCDT_TYPE*	28
SCDT\$L_SPDT*		32
SCDT\$L_SCSI_PORT_ID*		36
SCDT\$L_SCSI_BUS_ID*		40
SCDT\$L_SCSI_LUN*		44
reserved		48
SCDT\$L_SCDRP_ADDR*		56
SCDT\$L_BUS_PHASE*		60
SCDT\$L_OLD_PHASES*		64
~	SCDT\$W_PHASES* (44 bytes)	~ 68
SCDT\$L_PHASE_STK_PTR*		112
SCDT\$L_PHASE_END_STK_PTR*		116
SCDT\$L_EVENTS_SEEN*		120
SCDT\$L_ARB_FAIL_CNT*		124
SCDT\$L_SEL_FAIL_CNT*		128
SCDT\$L_PARERR_CNT*		132

(continued on next page)

SCSI Device Driver Data Structures

A.2 SCSI Connection Descriptor Table (SCDT)

Figure A-2 (Cont.) SCSI Connection Descriptor Table (SCDT)

SCDT\$L_MISPHS_CNT*	136
SCDT\$L_BADPHS_CNT*	140
SCDT\$L_RETRY_CNT*	144
SCDT\$L_RST_CNT*	148
SCDT\$L_CTLERR_CNT*	152
SCDT\$L_BUSERR_CNT*	156
SCDT\$L_CMDSENT*	160
SCDT\$L_MSGSENT*	164
SCDT\$L_BYTSENT*	168
SCDT\$L_CON_FLAGS*	172
SCDT\$L_SYNCHRONOUS*	176
SCDT\$W_TRANSFER_PERIOD*	180
SCDT\$W_REQACK_OFFSET*	180
SCDT\$W_ARB_RETRY_CNT*	184
SCDT\$W_BUSY_RETRY_CNT*	184
SCDT\$W_CMD_RETRY_CNT*	188
SCDT\$W_SEL_RETRY_CNT*	188
SCDT\$L_DMA_TIMEOUT*	192
SCDT\$L_DISCON_TIMEOUT*	196
SCDT\$L_SEL_CALLBACK*	200
reserved (40 bytes)	204

Table A-2 Contents of SCSI Connection Descriptor Table

Field Name	Contents
SCDT\$L_FLINK*	SCDT forward link. This field points to the next SCDT in the port's SCDT list (at SPDT\$L_SCDT_VECTOR). The SCSI port driver initializes this field when it creates the SCDT in response to an SPI\$CONNECT call.

(continued on next page)

SCSI Device Driver Data Structures

A.2 SCSI Connection Descriptor Table (SCDT)

Table A-2 (Cont.) Contents of SCSI Connection Descriptor Table

Field Name	Contents						
SCDT\$W_SIZE*	Size of SCDT. The port driver, after allocating sufficient nonpaged pool for the SCDT, writes the constant SCDT\$C_LENGTH into this field.						
SCDT\$B_FLCK*	Index of the fork lock that synchronizes access to this SCDT at fork level. The SCSI port driver, when creating the SCDT, initializes this field with SPL\$C_IOLOCK8. The SCDT fork block is used during an ABORT command request on the connection.						
SCDT\$L_FPC*	Address of instruction at which the suspended port driver thread is to be resumed.						
SCDT\$L_FR3*	Value of R3 when the request is stalled during disconnection. The value in R3 is restored before a suspended driver thread is resumed.						
SCDT\$L_FR4*	Value of R4 when the request is stalled during disconnection. The value in R4 is restored before a suspended driver thread is resumed.						
SCDT\$L_STS*	Connection status. This field is a bit map, maintained by the port driver. The only currently defined bit is SCDT\$V_BSY (connection busy).						
SCDT\$W_SCDT_TYPE*	Type of SCDT.						
SCDT\$W_STATE*	SCSI connection state. The VMS SCSI port driver maintains this field, using the following constants: <table data-bbox="641 997 1144 1108"> <tr> <td>SCDT\$C_CLOSED</td> <td>Closed</td> </tr> <tr> <td>SCDT\$C_OPEN</td> <td>Open</td> </tr> <tr> <td>SCDT\$C_FAIL</td> <td>Failed</td> </tr> </table>	SCDT\$C_CLOSED	Closed	SCDT\$C_OPEN	Open	SCDT\$C_FAIL	Failed
SCDT\$C_CLOSED	Closed						
SCDT\$C_OPEN	Open						
SCDT\$C_FAIL	Failed						
SCDT\$L_SPDT*	Address of port descriptor table with which this SCDT is associated.						
SCDT\$L_SCSI_PORT_ID*	SCSI port ID of the port to which this connection is established.						
SCDT\$L_SCSI_BUS_ID*	SCSI device ID of the device unit to which this connection is established.						
SCDT\$L_SCSI_LUN*	SCSI logical unit number (LUN) of the device unit to which this connection is established.						
SCDT\$L_SCDRP_ADDR*	Address of SCDRP current on the connection.						

(continued on next page)

SCSI Device Driver Data Structures

A.2 SCSI Connection Descriptor Table (SCDT)

Table A-2 (Cont.) Contents of SCSI Connection Descriptor Table

Field Name	Contents
SCDT\$L_BUS_PHASE*	Current SCSI bus phase. The VMS SCSI port driver defines the following flags in this longword bit map:
	SCDT\$V_DATAOUT DATA OUT phase
	SCDT\$V_DATAIN DATA IN phase
	SCDT\$V_CMD COMMAND phase
	SCDT\$V_STS STATUS phase
	SCDT\$V_INV1 Invalid phase 1
	SCDT\$V_INV2 Invalid phase 2
	SCDT\$V_MSGOUT MESSAGE OUT phase
	SCDT\$V_MSGIN MESSAGE IN phase
	SCDT\$V_ARB ARBITRATION phase
	SCDT\$V_SEL SELECTION phase
	SCDT\$V_RESEL RESELECTION phase
	SCDT\$V_DISCON DISCONNECT message seen
	SCDT\$V_TMODISCON Disconnect operation timed out
	SCDT\$V_CMD_CMPL COMMAND COMPLETE message received
	SCDT\$V_PND_RESEL Reselection interrupt pending
	SCDT\$V_FREE BUS FREE phase
SCDT\$L_OLD_PHASES*	Bus phase tracking information.
SCDT\$W_PHASES*	Bus phase tracking information. This field is 44 bytes long.
SCDT\$L_PHASE_STK_PTR*	Address of the top of the bus phase stack. The VMS SCSI port driver uses the bus phase stack to maintain a phase histogram.
SCDT\$L_PHASE_END_STK_PTR*	Address of the bottom of the bus phase stack. The VMS SCSI port driver uses the bus phase stack to maintain a phase histogram.
SCDT\$L_EVENTS_SEEN*	Longword bit mask of bus events seen by the VMS SCSI port driver. VMS defines the following bits:
	SCDT\$V_PARERR Parity error
	SCDT\$V_BSYERR Bus lost during command
	SCDT\$V_MISPHS Missing bus phase
	SCDT\$V_BADPHS Bad phase transition
	SCDT\$V_RST Bus reset during command
	SCDT\$V_CTLERR SCSI controller error
	SCDT\$V_BUSERR SCSI bus error
SCDT\$L_ARB_FAIL_CNT*	Count of arbitration failures.
SCDT\$L_SEL_FAIL_CNT*	Count of selection failures.
SCDT\$L_PARERR_CNT*	Count of parity errors.
SCDT\$L_MISPHS_CNT*	Count of missing phases errors.

(continued on next page)

SCSI Device Driver Data Structures

A.2 SCSI Connection Descriptor Table (SCDT)

Table A-2 (Cont.) Contents of SCSI Connection Descriptor Table

Field Name	Contents
SCDT\$L_BADPHS_CNT*	Count of bad phase errors.
SCDT\$L_RETRY_CNT*	Count of retries.
SCDT\$L_RST_CNT*	Count of bus resets.
SCDT\$L_CTLERR_CNT*	Count of controller errors.
SCDT\$L_BUSERR_CNT*	Count of bus errors.
SCDT\$L_CMDSENT*	Number of commands sent on this connection.
SCDT\$L_MSGSENT*	Number of messages sent on this connection.
SCDT\$L_BYTSENT*	Number of bytes sent during DATA OUT phase.
SCDT\$L_CON_FLAGS*	Connection-specific flags. The VMS SCSI port driver sets or clears these flags according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro. The following bits are defined: SCDT\$V_ENA_DISCON Enable disconnect SCDT\$V_DIS_RETRY Disable command retry SCDT\$V_TARGET_MODE Enable asynchronous event notification from target
SCDT\$L_SYNCHRONOUS*	Synchronous data transfer enabled field. This longword contains 1 if synchronous data transfers are enabled for this connection; otherwise it contains a 0. The VMS SCSI port driver writes this field according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro.
SCDT\$W_REQACK_OFFSET*	For synchronous data transfers, maximum number of REQs outstanding on the connection before an ACK is transmitted. The VMS SCSI port driver writes this field according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro.
SCDT\$W_TRANSFER_PERIOD*	Number of 4-nanosecond ticks between a REQ and an ACK on this connection. The VMS SCSI port driver writes this field according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro.
SCDT\$W_BUSY_RETRY_CNT*	Remaining number of retries allowed on this connection to successfully send a command to the target device. The VMS SCSI port driver initially writes this field according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro.
SCDT\$W_ARB_RETRY_CNT*	Remaining number of retries allowed on this connection while waiting for the port to win arbitration of the bus. The VMS SCSI port driver initially writes this field according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro.
SCDT\$W_SEL_RETRY_CNT*	Select retry count. Remaining number of retries allowed on this connection while waiting for the port to be selected by the target device. The VMS SCSI port driver initially writes this field according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro.

(continued on next page)

SCSI Device Driver Data Structures

A.2 SCSI Connection Descriptor Table (SCDT)

Table A-2 (Cont.) Contents of SCSI Connection Descriptor Table

Field Name	Contents
SCDT\$W_CMD_RETRY_CNT*	Remaining number of retries allowed on this connection to successfully send a command to the target device. The VMS SCSI port driver initially writes this field according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro.
SCDT\$L_DMA_TIMEOUT*	Timeout value (in seconds) for a target to change the SCSI bus phase or complete a data transfer. The VMS SCSI port driver initially writes this field according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro.
SCDT\$L_DISCON_TIMEOUT*	Disconnect timeout. Default timeout value (in seconds) for a target to reselect the initiator to proceed with a disconnected I/O transfer. The VMS SCSI port driver initially writes this field according to information the SCSI class driver supplies to the SPI\$SET_CONNECTION_CHAR macro.
SCDT\$L_SEL_CALLBACK*	Address of class driver's asynchronous event notification callback routine.

A.3 SCSI Port Descriptor Table (SPDT)

The SCSI port descriptor table (SPDT) contains information specific to a SCSI port, such as the port driver connection database. The SPDT also includes a set of vectors, corresponding to the SPI macros invoked by SCSI class drivers, that point to service routines within the port driver. The SCSI port driver's unit initialization routine creates an SPDT for each SCSI port defined for a specific MicroVAX/VAXstation system and initializes each SPI vector.

The port driver reads and writes fields in the SPDT. The class driver reads the SPDT indirectly when it invokes an SPI macro.

The SPDT is illustrated in Figure A-3 and described in Table A-3.

Figure A-3 SCSI Port Descriptor Table (SPDT)

SPDT\$L_FLINK*		0
reserved	SPDT\$W_SIZE*	
SPDT\$B_FLCK*	SPDT\$B_SCSI_INT_MSK*	8
SPDT\$L_FPC*		12
SPDT\$L_FR3*		16
SPDT\$L_FR4*		20
SPDT\$L_SCSI_PORT_ID*		24
SPDT\$L_SCSI_BUS_ID*		28

(continued on next page)

SCSI Device Driver Data Structures

A.3 SCSI Port Descriptor Table (SPDT)

Figure A-3 (Cont.) SCSI Port Descriptor Table (SPDT)

SPDT\$L_STS*	32
SPDT\$L_PORT_WQFL*	36
SPDT\$L_PORT_WQBL*	40
SPDT\$L_MAXBYTECNT*	44
reserved	48
SPDT\$L_PORT_UCB*	56
SPDT\$L_PORT_CSR*	60
SPDT\$L_PORT_IDB*	64
SPDT\$L_DMA_BASE*	68
SPDT\$L_SPTE_BASE*	72
SPDT\$L_SPTE_SVAPTE*	76
SPDT\$L_ADP*	80
SPDT\$L_PORT_RING* (64 bytes)	84
SPDT\$L_PORT_RING_PTR*	148
SPDT\$L_OWNERSCDT*	152
SPDT\$L_SCDT_VECTOR* (256 bytes)	156
SPDT\$L_DLCK*	412
SPDT\$B_DIPL*	416
reserved	
SPDT\$L_SEL_SCDRP*	424
SPDT\$L_ENB_SEL_SCDRP*	428
SPDT\$L_MAP_BUFFER*	432
SPDT\$L_UNMAP*	436

(continued on next page)

SCSI Device Driver Data Structures

A.3 SCSI Port Descriptor Table (SPDT)

Figure A-3 (Cont.) SCSI Port Descriptor Table (SPDT)

SPDT\$L_SEND*	440
SPDT\$L_SET_CONN_CHAR*	444
SPDT\$L_GET_CONN_CHAR*	448
SPDT\$L_RESET*	452
SPDT\$L_CONNECT*	456
SPDT\$L_DISCONNECT*	460
SPDT\$L_ALLOC_COMMAND_BUFFER*	464
SPDT\$L_DEALLOC_COMMAND_BUFFER*	468
SPDT\$L_ABORT*	472
SPDT\$L_SET_PHASE*	476
SPDT\$L_SENSE_PHASE*	480
SPDT\$L_SEND_BYTES*	484
SPDT\$L_RECEIVE_BYTES*	488
SPDT\$L_FINISH_CMD*	492
SPDT\$L_RELEASE_BUS*	496
~ reserved (52 bytes) ~	~500
reserved	552
BUS_HUNG_VEC*	552
~ SPDT\$B_TQE* (52 bytes) ~	~556
SPDT\$L_TQE_DELAY*	608
SPDT\$L_BUS_HUNG_CNT*	612
SPDT\$L_TARRST_CNT*	616
SPDT\$L_RETRY_CNT*	620
SPDT\$L_STRAY_INT_CNT*	624
SPDT\$L_UNEXP_INT_CNT*	628

(continued on next page)

SCSI Device Driver Data Structures

A.3 SCSI Port Descriptor Table (SPDT)

Figure A-3 (Cont.) SCSI Port Descriptor Table (SPDT)

SPDT\$L_NODISCON_CNT*				632
SPDT\$W_DISCON_CNT*		reserved		636
SPDT\$L_PORT_FLAGS*				640
SPDT\$L_VERSION_CHECK*				644
reserved (36 bytes)				648
SPDT\$B_EVENT_CNT*	SPDT\$B_MODE*	SPDT\$B_STATUS*	SPDT\$B_CUR_STAT*	684
reserved (16 bytes)				688

Table A-3 Contents of SCSI Port Descriptor Table

Field Name	Contents
SPDT\$L_FLINK*	SPDT forward link. This field points to the next SPDT in the system SPDT list. The SCSI port driver initializes this field when it creates the SPDT.
SPDT\$W_SIZE*	Size of SPDT. The VMS SCSI port driver initializes this field to SPDT\$C_PKNLENGTH or SPDT\$C_PKSLLENGTH when creating the SPDT.
SPDT\$W_SPDT_TYPE*	SPDT type. The VMS SCSI port driver initializes this field to SPDT\$C_PKN or SPDT\$C_PKS when creating the SPDT.
SPDT\$B_SCSI_INT_MSK*	Port-specific interrupt mask.
SPDT\$B_FLCK*	Index of the fork lock that synchronizes access to this SPDT at fork level. The SCSI port driver, when creating the SPDT, copies to this field the value of UCB\$B_FLCK. The SPDT fork block is used during reselection and disconnection.
SPDT\$L_FPC*	Address of instruction at which the suspended port driver thread is to be resumed.
SPDT\$L_FR3*	Value of R3 when the request is stalled during disconnection. The value in R3 is restored before a suspended driver thread is resumed.
SPDT\$L_FR4*	Value of R4 when the request is stalled during disconnection. The value in R4 is restored before a suspended driver thread is resumed.
SPDT\$L_SCSI_PORT_ID*	SCSI port ID, an alphabetic value from A to Z.
SPDT\$L_SCSI_BUS_ID*	SCSI device ID of the port, a numeric value from 0 to 7.

(continued on next page)

SCSI Device Driver Data Structures

A.3 SCSI Port Descriptor Table (SPDT)

Table A–3 (Cont.) Contents of SCSI Port Descriptor Table

Field Name	Contents
SPDT\$L_STS*	Port device status. This field is a bit map maintained by the port driver. The following bits are defined: SPDT\$V_ONLINE Online SPDT\$V_TIMEOUT Timed out SPDT\$V_ERLOGIP Error log in progress SPDT\$V_CANCEL Cancel I/O SPDT\$V_POWER Power failed while unit busy SPDT\$V_BSY Busy SPDT\$V_FAILED Port failed operation or initialization
SPDT\$L_PORT_WQFL*	Port wait queue forward link. This field points to the first SCDRP waiting for the port to be free.
SPDT\$L_PORT_WQBL*	Port wait queue backward link. This field points to the last SCDRP waiting for the port to be free.
SPDT\$L_MAXBYTECNT*	Maximum byte count for a transfer using this port.
SPDT\$L_PORT_UCB*	Address of port UCB.
SPDT\$L_PORT_CSR*	Address of the port hardware's CSR.
SPDT\$L_PORT_IDB*	Address of the port IDB.
SPDT\$L_DMA_BASE*	Base address of the port's DMA buffer.
SPDT\$L_SPTE_BASE*	System virtual address of the system page-table entry mapping the first page of the port's DMA buffer.
SPDT\$L_SPTE_SVAPTE*	System virtual address of the system page-table entry that double-maps the data transfer buffer.
SPDT\$L_ADP*	Address of the adapter control block managing port resources.
SPDT\$L_PORT_RING*	64-byte field recording the PCs of port channel request and release transactions.
SPDT\$L_PORT_RING_PTR*	Pointer to the current port channel ring buffer entry.
SPDT\$L_OWNERSCDT*	Address of the SCDT of the connection that currently owns the port.
SPDT\$L_SCDT_VECTOR*	256-byte vector, recording the SCDT addresses associated with connections active for a given SCSI device ID (0 through 7).
SPDT\$L_DLCK*	Address of device lock that—in a VMS multiprocessing environment—synchronizes access to device registers and those fields at the SPDT accessed at device IPL. The port driver initializes this field from UCB\$L_DLCK when it creates the SPDT.
SPDT\$B_DIPL*	Interrupt priority level (IPL) at which the device requests hardware interrupts. The port driver initializes this field from UCB\$L_DLCK when it creates the SPDT.
SPDT\$L_SEL_SCDRP*	SCDRP used during selection interrupt.
SPDT\$L_ENB_SEL_SCDRP*	SCDRP used to enable selection.
SPDT\$L_MAP_BUFFER*	Address of the port driver routine that executes in response to a class driver's SPI\$MAP_BUFFER macro call. The port driver initializes this field.

(continued on next page)

SCSI Device Driver Data Structures

A.3 SCSI Port Descriptor Table (SPDT)

Table A-3 (Cont.) Contents of SCSI Port Descriptor Table

Field Name	Contents
SPDT\$L_UNMAP*	Address of the port driver routine that executes in response to a class driver's SPI\$UNMAP_BUFFER macro call. The port driver initializes this field.
SPDT\$L_SEND*	Address of the port driver routine that executes in response to a class driver's SPI\$SEND_COMMAND macro call. The port driver initializes this field.
SPDT\$L_SET_CONN_CHAR*	Address of the port driver routine that executes in response to a class driver's SPI\$SET_CONNECTION_CHAR macro call. The port driver initializes this field.
SPDT\$L_GET_CONN_CHAR*	Address of the port driver routine that executes in response to a class driver's SPI\$GET_CONNECTION_CHAR macro call. The port driver initializes this field.
SPDT\$L_RESET*	Address of the port driver routine that executes in response to a class driver's SPI\$RESET macro call. The port driver initializes this field.
SPDT\$L_CONNECT*	Address of the port driver routine that executes in response to a class driver's SPI\$CONNECT macro call. The port driver initializes this field.
SPDT\$L_DISCONNECT*	Address of the port driver routine that executes in response to a class driver's SPI\$DISCONNECT macro call. The port driver initializes this field.
SPDT\$L_ALLOC_COMMAND_BUFFER*	Address of the port driver routine that executes in response to a class driver's SPI\$ALLOCATE_COMMAND_BUFFER macro call. The port driver initializes this field.
SPDT\$L_DEALLOC_COMMAND_BUFFER*	Address of the port driver routine that executes in response to a class driver's SPI\$DEALLOCATE_COMMAND_BUFFER macro call. The port driver initializes this field.
SPDT\$L_ABORT*	Address of the port driver routine that executes in response to a class driver's SPI\$ABORT_COMMAND macro call. The port driver initializes this field.
SPDT\$L_SET_PHASE*	Address of the port driver asynchronous event notification (AEN) routine that executes in response to a class driver's SPI\$SET_PHASE macro call. The port driver initializes this field.
SPDT\$L_SENSE_PHASE*	Address of the port driver AEN routine that executes in response to a class driver's SPI\$SENSE_PHASE macro call. The port driver initializes this field.
SPDT\$L_SEND_BYTES*	Address of the port driver AEN routine that executes in response to a class driver's SPI\$SEND_BYTES macro call. The port driver initializes this field.
SPDT\$L_RECEIVE_BYTES*	Address of the port driver AEN routine that executes in response to a class driver's SPI\$RECEIVE_BYTES macro call. The port driver initializes this field.
SPDT\$L_FINISH_CMD*	Address of the port driver AEN routine that executes in response to a class driver's SPI\$FINISH_COMMAND macro call. The port driver initializes this field.
SPDT\$L_RELEASE_BUS*	Address of the port driver routine that executes in response to a class driver's SPI\$RELEASE_BUS macro call. The port driver initializes this field.

(continued on next page)

SCSI Device Driver Data Structures

A.3 SCSI Port Descriptor Table (SPDT)

Table A-3 (Cont.) Contents of SCSI Port Descriptor Table

Field Name	Contents
SPDT\$B_BUS_HUNG_VEC*	Vector of suspected hung connections.
SPDT\$B_TQE*	Timer queue element (52 bytes long), used by the port driver to time out pending disconnected I/O transfers. When this TQE expires, the timer thread times out expired pending I/O transfers.
SPDT\$L_TQE_DELAY*	Delay time for next TQE delay.
SPDT\$L_BUS_HUNG_CNT*	Count of detected bus hangs.
SPDT\$L_TARRST_CNT*	Count of target-initiated bus resets.
SPDT\$L_RETRY_CNT*	Total of retry attempts.
SPDT\$L_STRAY_INT_CNT*	Count of interrupts occurring when channel is unowned.
SPDT\$L_UNEXP_INT_CNT*	Count of unexpected interrupts occurring when channel is owned.
SPDT\$L_NODISCON_CNT*	Count of reselections when port is not disconnected.
SPDT\$W_DISCON_CNT*	Count of outstanding disconnects.
SPDT\$L_PORT_FLAGS*	Port-specific flags. The following bits are defined: <ul style="list-style-type: none"> SPDT\$V_SYNCH Port supports synchronous mode data transfers. SPDT\$V_ASYNCH Port supports asynchronous mode data transfers. SPDT\$V_MAPPING_REG Port supports map registers. SPDT\$V_BUF_DMA Port supports buffered DMA transfers. SPDT\$V_DIR_DMA Port supports direct DMA transfers. SPDT\$V_AEN Port supports asynchronous event notification. SPDT\$V_LUNS Port supports logical unit numbers.
SPDT\$L_VERSION_CHECK*	Value used to check driver versions.
SPDT\$L_RSVD_LONG*	Reserved to Digital.
SPDT\$B_CUR_STAT*	Copy of CUR_STAT register.
SPDT\$B_STATUS*	Copy of STATUS register.
SPDT\$B_MODE*	Copy of MODE register.
SPDT\$B_EVENT_CNT*	Count of events while servicing current interrupt.

B VMS Macros Invoked by SCSI Class Drivers

This appendix describes the macros used by third-party SCSI class drivers. It includes the following sections:

- Standard SCSI port interface macros
- Extended SCSI port interface macros, enabling asynchronous event notification (AEN)

B.1 Standard SCSI Port Interface Macros

The macros described in this section are used by all SCSI class drivers to communicate with the SCSI port.

A SCSI class driver invokes SPI macros at fork IPL, holding the fork lock. Because the port driver routines called by SPI macros may fork or stall, a class driver must preserve local context and local return addresses across an SPI macro invocation. It must also ensure that the address of its caller is at the top of the stack at the time of the invocation. (These issues are more fully discussed in Section 3.5.1.)

VMS Macros Invoked by SCSI Class Drivers

SPI\$ABORT_COMMAND

SPI\$ABORT_COMMAND

Aborts execution of the outstanding SCSI command on a given connection.

FORMAT SPI\$ABORT_COMMAND

DESCRIPTION The SPI\$ABORT_COMMAND macro aborts the outstanding SCSI command on the connection specified in SCDRP\$L_CDT. The SCSI port driver's abort routine sends the SCSI ABORT command to the target device.

Note: VAXstation 3520/3540 systems do not implement the abort-SCSI-command function.

Inputs to the SPI\$ABORT_COMMAND macro include the following:

Location	Contents
R4	Address of the SPDT
R5	Address of the SCDRP
SCDRP\$L_CDT	Address of the SCDT

The port driver returns SS\$_NORMAL status in R0, and preserves the contents of R3, R4, and R5. The original SPI\$SEND_COMMAND call completes with SS\$_ABORT status.

VMS Macros Invoked by SCSI Class Drivers

SPI\$ALLOCATE_COMMAND_BUFFER

SPI\$ALLOCATE_COMMAND_BUFFER

Allocates a port command buffer for a SCSI command descriptor block.

FORMAT SPI\$ALLOCATE_COMMAND_BUFFER

DESCRIPTION The SPI\$ALLOCATE_COMMAND_BUFFER macro allocates a port command buffer for a SCSI command descriptor block.

Typically a SCSI class driver requests two additional longwords when specifying the size of the requested buffer, the first for the SCSI status byte and the second for the length of the SCSI command. The port command buffer allows the SCSI port driver to access both the SCSI command descriptor block and the SCSI status byte during the SCSI COMMAND and STATUS phases.

Inputs to the SPI\$ALLOCATE_COMMAND_BUFFER macro include the following:

Location	Contents
R1	Size of requested buffer. This value should include the size of the SCSI command, plus 4 bytes reserved for the SCSI status byte and 4 bytes in which the SCSI class driver places the size of the SCSI command.
R4	Address of the SPDT.
R5	Address of the SCDRP.
SCDRP\$L_CDT	Address of the SCDT.
SCDRP\$W_CMD_MAPREG	Page number of the first port DMA buffer page allocated for the port command buffer.
SCDRP\$W_CMD_NUMREG	Number of port DMA buffer pages allocated for the port DMA buffer.

The port driver returns the following values to the class driver, preserving the contents of R3, R4, and R5:

Location	Contents
R0	SS\$_NORMAL
R1	Size of port command buffer
R2	Address of port command buffer

VMS Macros Invoked by SCSI Class Drivers

SPI\$CONNECT

SPI\$CONNECT

Creates a connection from a class driver to a SCSI device.

FORMAT **SPI\$CONNECT** *[callback]*

PARAMETERS *callback*

Address of a routine in the class driver that executes in response to asynchronous event notification from the target device. When the SCSI port driver receives a selection on an existing connection, it calls the class driver at this address, holding the fork lock and no other locks at IPL 8.

If the SCSI class driver does not provide a callback address, no selections are allowed on the connection that is established.

DESCRIPTION

The SPI\$CONNECT macro establishes a connection between the class driver and a SCSI device. It also links a SCSI class driver to the port driver. Before a SCSI class driver can exchange commands and data with a SCSI device, it must invoke SPI\$CONNECT.

In response to the call to SPI\$CONNECT, the port driver allocates and links an SCDT for the connection. It marks the connection state open and initializes default connection information. If the connection already exists, it returns SS\$_DEVALLOC status to the class driver.

Inputs to the SPI\$CONNECT macro include the following:

Location	Contents
R1	SCSI device ID (bits <31:16>) and SCSI port ID (bits <15:0>). Valid SCSI device IDs are integers from 0 to 7; valid SCSI port IDs are integers 0 and 1, corresponding to controller IDs <i>A</i> and <i>B</i> .
R2	SCSI logical unit number (bits <31:16>). Bits <15:0> are reserved. Valid SCSI logical unit numbers are integers from 0 to 7.

VMS Macros Invoked by SCSI Class Drivers

SPI\$CONNECT

The port driver returns the following values to the class driver:

Location	Contents
R0	<p>Port status. The port driver returns one of the following values:</p> <p>SS\$_DEVALLOC Connection already open for this target.</p> <p>SS\$_DEVOFFLINE Port is off line and allows no connections.</p> <p>SS\$_INSFMEM Insufficient memory to allocate SCDT.</p> <p>SS\$_NORMAL Connection formed.</p> <p>SS\$_NOSUCHDEV Port not found.</p>
R2	Address of the SCDT.
R3	<p>Port capability mask. The following bits are defined by the \$SPDTDEF macro (in SYS\$LIBRARY:LIB.MLB):</p> <p>SPDT\$_M_SYNCH Supports synchronous mode.</p> <p>SPDT\$_M_ASYNCH Supports asynchronous mode.</p> <p>SPDT\$_M_MAPPING_REG Supports map registers.</p> <p>SPDT\$_M_BUF_DMA Supports buffered DMA.</p> <p>SPDT\$_M_DIR_DMA Supports direct DMA.</p> <p>SPDT\$_M_AEN Supports asynchronous event notification.</p> <p>SPDT\$_M_LUNS Supports LUNs (logical unit numbers).</p>
R4	Address of the SPDT.

VMS Macros Invoked by SCSI Class Drivers

SPI\$DEALLOCATE_COMMAND_BUFFER

SPI\$DEALLOCATE_COMMAND_BUFFER

Deallocates a port command buffer.

FORMAT SPI\$DEALLOCATE_COMMAND_BUFFER

DESCRIPTION

The SPI\$DEALLOCATE_COMMAND_BUFFER macro deallocates a port command buffer.

Inputs to the SPI\$DEALLOCATE_COMMAND_BUFFER macro include the following:

Location	Contents
R4	Address of the SPDT.
R5	Address of the SCDRP.
SCDRP\$L_CDT	Address of the SCDT.
SCDRP\$W_CMD_MAPREG	Page number of the first port DMA buffer page allocated for the port command buffer.
SCDRP\$W_CMD_NUMREG	Number of the port DMA buffer pages allocated for the port DMA buffer.

The port driver returns SS\$_NORMAL status in R0, and preserves the contents of R3, R4, and R5.

VMS Macros Invoked by SCSI Class Drivers

SPI\$DISCONNECT

SPI\$DISCONNECT

Breaks a connection between a class driver and a SCSI port.

FORMAT SPI\$DISCONNECT

DESCRIPTION

The SPI\$DISCONNECT macro breaks a connection between a class driver and a SCSI device unit and deallocates the associated SCDT. The connection must not be busy when it is being disconnected.

Normally a connection between a class driver and a SCSI device unit lasts throughout the runtime life of a system. A SCSI class driver should never need to invoke this macro.

Inputs to the SPI\$DISCONNECT macro include the following:

Location	Contents
R1	SCSI device ID (bits <31:16>) and SCSI port ID (bits <15:0>). Valid SCSI device IDs are integers from 0 to 7; valid SCSI port IDs are integers 0 and 1, corresponding to controller IDs <i>A</i> and <i>B</i> .
R2	SCSI logical unit number (bits <15:0>). Valid SCSI logical unit numbers are integers from 0 to 7.
R4	Address of the SPDT.
R5	Address of the SCDT.

The port driver returns SS\$_NORMAL status in R0, and preserves the contents of R3, R4, and R5.

VMS Macros Invoked by SCSI Class Drivers

SPI\$GET_CONNECTION_CHAR

SPI\$GET_CONNECTION_CHAR

Returns characteristics of an existing connection to a specified buffer.

FORMAT SPI\$GET_CONNECTION_CHAR

DESCRIPTION The SPI\$GET_CONNECTION_CHAR macro returns characteristics of an existing connection to a specified buffer.

The connection characteristics buffer has the following format:

Longword	Contents						
1	Number of longwords in the buffer, <i>not</i> including this longword. The value of this field must be 10.						
2	Connection flags. Bits in this longword are defined as follows: <table border="1"><thead><tr><th>Bit</th><th>Description</th></tr></thead><tbody><tr><td>0</td><td>ENA_DISCON. When set, this bit indicates that disconnect and reselection are enabled on this connection.</td></tr><tr><td>1</td><td>DIS_RETRY. When set, this bit indicates that command retry is disabled on this connection.</td></tr></tbody></table>	Bit	Description	0	ENA_DISCON. When set, this bit indicates that disconnect and reselection are enabled on this connection.	1	DIS_RETRY. When set, this bit indicates that command retry is disabled on this connection.
Bit	Description						
0	ENA_DISCON. When set, this bit indicates that disconnect and reselection are enabled on this connection.						
1	DIS_RETRY. When set, this bit indicates that command retry is disabled on this connection.						
3	Synchronous. When this longword contains 0, the connection supports asynchronous data transfers; when it contains a nonzero value, the connection supports synchronous data transfers.						
4	Transfer period. If the synchronous parameter is nonzero, this field contains the number of 4-nanosecond ticks between a REQ and an ACK. The default is 64 ₁₀ .						
5	REQ-ACK offset. If the synchronous parameter is nonzero, this field contains the maximum number of REQs outstanding before there must be an ACK.						
6	Busy retry count. Maximum number of retries allowed on this connection while waiting for the bus to become free.						
7	Select retry count. Maximum number of retries allowed on this connection while waiting for the port to be selected by the target device.						
8	Arbitration retry count. Maximum number of retries allowed on this connection while waiting for the port to win arbitration of the bus.						

VMS Macros Invoked by SCSI Class Drivers

SPI\$GET_CONNECTION_CHAR

Longword	Contents
9	Command retry count. Maximum number of retries allowed on this connection to successfully send a command to the target device.
10	Phase change timeout. Default timeout value (in seconds) for a target to change the SCSI bus phase or complete a data transfer. This value is also known as the DMA timeout. Upon sending the last command byte, the port driver waits this many seconds for the target to change the bus phase lines and assert REQ (indicating a new phase). Or, if the target enters the DATA IN or DATA OUT phase, the transfer must be completed within this interval. If this value is not specified, the default value is 4 seconds.
11	Disconnect timeout. Default timeout value (in seconds) for a target to reselect the initiator to proceed with a disconnected I/O transfer. If this value is not specified, the default value is 4 seconds.

Inputs to the SPI\$GET_CONNECTION_CHAR macro include the following:

Location	Contents
R2	Address of the connection characteristics buffer.
R4	Address of the SPDT.
R5	Address of the SCDRP.
SCDRP\$L_CDT	Address of the SCDT.

The port driver returns the following values to the class driver, preserving R3, R4, and R5:

Location	Contents				
R0	Port status. The port driver returns one of the following values: <table style="margin-left: 20px; border: none;"> <tr> <td>SS\$_NORMAL</td> <td>Normal, successful completion</td> </tr> <tr> <td>SS\$_NOSUCHID</td> <td>No connection for this SCSI connection ID</td> </tr> </table>	SS\$_NORMAL	Normal, successful completion	SS\$_NOSUCHID	No connection for this SCSI connection ID
SS\$_NORMAL	Normal, successful completion				
SS\$_NOSUCHID	No connection for this SCSI connection ID				
R2	Address of the connection characteristics buffer in which device characteristics are returned.				

VMS Macros Invoked by SCSI Class Drivers

SPI\$MAP_BUFFER

SPI\$MAP_BUFFER

Makes the process buffer involved in a data transfer available to the port driver.

FORMAT

SPI\$MAP_BUFFER

DESCRIPTION

The SPI\$MAP_BUFFER macro makes the process buffer involved in a data transfer accessible to the port driver. Typically, the I/O buffer is specified in the \$QIO call, is in process space (P0 space), and is mapped by process page-table entries. Because a port driver executes in system context, it cannot access a process's page table.

The means by which the SPI\$MAP_BUFFER macro makes a process buffer available to the port driver depends upon the port hardware. For certain implementations, it allocates a segment of the port's DMA buffer and a set of system page-table entries that double-map the process buffer. In others, it obtains a set of port map registers and loads them with the page-frame numbers of the process buffer pages.

VMS Macros Invoked by SCSI Class Drivers

SPI\$MAP_BUFFER

Inputs to the SPI\$MAP_BUFFER macro include the following:

Location	Contents										
R4	Address of the SPDT.										
R5	Address of the SCDRP. The class driver must provide values in the following fields:										
	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%; vertical-align: top;">SCDRP\$L_BCNT</td> <td style="vertical-align: top;">Size in bytes of the buffer to be mapped. The largest single transfer that can be mapped is determined by the port driver in the call to SPI\$CONNECT. The SPI\$CONNECT macro returns this value to the class driver in R1. If the class driver must accomplish transfers larger than this value, it must segment them.</td> </tr> <tr> <td style="vertical-align: top;">SCDRP\$W_BOFF</td> <td style="vertical-align: top;">Byte offset into the first page of the buffer.</td> </tr> <tr> <td style="vertical-align: top;">SCDRP\$L_SVAPTE</td> <td style="vertical-align: top;">System virtual address of the page-table entry that maps the first byte of the user buffer.</td> </tr> <tr> <td style="vertical-align: top;">SCDRP\$L SCSI_FLAGS</td> <td style="vertical-align: top;">SCSI mapping flags. If SCDRP\$V_S0BUF is set, SPI\$MAP_BUFFER does not double-map the buffer into system space.</td> </tr> <tr> <td style="vertical-align: top;">SCDRP\$W_STS</td> <td style="vertical-align: top;">Transfer direction flags. IRP\$V_FUNC must be set for read I/O functions and clear for write I/O functions.</td> </tr> </table>	SCDRP\$L_BCNT	Size in bytes of the buffer to be mapped. The largest single transfer that can be mapped is determined by the port driver in the call to SPI\$CONNECT. The SPI\$CONNECT macro returns this value to the class driver in R1. If the class driver must accomplish transfers larger than this value, it must segment them.	SCDRP\$W_BOFF	Byte offset into the first page of the buffer.	SCDRP\$L_SVAPTE	System virtual address of the page-table entry that maps the first byte of the user buffer.	SCDRP\$L SCSI_FLAGS	SCSI mapping flags. If SCDRP\$V_S0BUF is set, SPI\$MAP_BUFFER does not double-map the buffer into system space.	SCDRP\$W_STS	Transfer direction flags. IRP\$V_FUNC must be set for read I/O functions and clear for write I/O functions.
SCDRP\$L_BCNT	Size in bytes of the buffer to be mapped. The largest single transfer that can be mapped is determined by the port driver in the call to SPI\$CONNECT. The SPI\$CONNECT macro returns this value to the class driver in R1. If the class driver must accomplish transfers larger than this value, it must segment them.										
SCDRP\$W_BOFF	Byte offset into the first page of the buffer.										
SCDRP\$L_SVAPTE	System virtual address of the page-table entry that maps the first byte of the user buffer.										
SCDRP\$L SCSI_FLAGS	SCSI mapping flags. If SCDRP\$V_S0BUF is set, SPI\$MAP_BUFFER does not double-map the buffer into system space.										
SCDRP\$W_STS	Transfer direction flags. IRP\$V_FUNC must be set for read I/O functions and clear for write I/O functions.										

The port driver returns the following values to the class driver, preserving R3, R4, and R5:

Location	Contents				
R0	Port status. The port driver returns one of the following values:				
	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%; vertical-align: top;">SS\$_NORMAL</td> <td style="vertical-align: top;">Normal, successful completion</td> </tr> <tr> <td style="vertical-align: top;">SS\$_BADPARAM</td> <td style="vertical-align: top;">Bad parameter provided by class driver</td> </tr> </table>	SS\$_NORMAL	Normal, successful completion	SS\$_BADPARAM	Bad parameter provided by class driver
SS\$_NORMAL	Normal, successful completion				
SS\$_BADPARAM	Bad parameter provided by class driver				

VMS Macros Invoked by SCSI Class Drivers

SPI\$MAP_BUFFER

Location	Contents								
R5	Address of the SCDRP. The port driver initializes the following fields:								
	<table> <tr> <td>SCDRP\$L_SVA_USER</td> <td>System virtual address of the process buffer as mapped in system space (S0 space)</td> </tr> <tr> <td>SCDRP\$L_SVA_SPTE</td> <td>System virtual address of the system page-table entry that maps the first page of the process buffer in S0 space</td> </tr> <tr> <td>SCDRP\$W_NUMREG</td> <td>Number of port DMA buffer pages allocated</td> </tr> <tr> <td>SCDRP\$W_MAPREG</td> <td>Page number of the first port DMA buffer page allocated</td> </tr> </table>	SCDRP\$L_SVA_USER	System virtual address of the process buffer as mapped in system space (S0 space)	SCDRP\$L_SVA_SPTE	System virtual address of the system page-table entry that maps the first page of the process buffer in S0 space	SCDRP\$W_NUMREG	Number of port DMA buffer pages allocated	SCDRP\$W_MAPREG	Page number of the first port DMA buffer page allocated
SCDRP\$L_SVA_USER	System virtual address of the process buffer as mapped in system space (S0 space)								
SCDRP\$L_SVA_SPTE	System virtual address of the system page-table entry that maps the first page of the process buffer in S0 space								
SCDRP\$W_NUMREG	Number of port DMA buffer pages allocated								
SCDRP\$W_MAPREG	Page number of the first port DMA buffer page allocated								

SPI\$RESET

Resets the SCSI bus and SCSI port hardware.

FORMAT SPI\$RESET

DESCRIPTION

The SPI\$RESET macro first resets the SCSI bus and then resets the port hardware. A SCSI class driver should rarely invoke this macro; those class drivers that do use it should be aware of the impact of a reset operation on other devices on the same bus. The VMS SCSI port driver logs an error when a class driver invokes the SPI\$RESET macro.

Inputs to the SPI\$RESET macro include the following:

Location	Contents
R0	Reset bit mask. The only supported value is RESET\$M_BUS.
R4	Address of the SPDT.
R5	Address of the SCDRP.
SCDRP\$L_CDT	Address of the SCDT.

The port driver returns the following value to the class driver, preserving R3, R4, and R5:

Location	Contents
R0	Port status. The port driver returns one of the following values: SS\$_NORMAL Normal, successful completion. SS\$_ABORT Reset aborted before completion.

VMS Macros Invoked by SCSI Class Drivers

SPI\$SEND_COMMAND

SPI\$SEND_COMMAND

Sends a command to a SCSI device.

FORMAT	SPI\$SEND_COMMAND
---------------	--------------------------

DESCRIPTION	<p>The SPI\$SEND_COMMAND macro sends a command to a SCSI device. A class driver invokes this macro, after calling SPI\$ALLOCATE_COMMAND_BUFFER to allocate a port command buffer and formatting a SCSI command descriptor block in it.</p>
--------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The port driver responds to the SPI\$SEND_COMMAND macro call by arbitrating for ownership of the SCSI bus, selecting the target device, sending the SCSI command descriptor block to the target, and waiting for a response. Prior to returning to the class driver, the port driver sends data to or receives data from the target device, obtains command status, processes SCSI message bytes, and transfers the data. When it returns from the SPI\$SEND_COMMAND call, the port driver returns port status and SCSI status to the class driver.

VMS Macros Invoked by SCSI Class Drivers

SPI\$SEND_COMMAND

Inputs to the SPI\$SEND_COMMAND macro include the following:

Location	Contents										
R4	Address of the SPDT.										
R5	Address of the SCDRP. The class driver must provide values in the following fields:										
	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%; vertical-align: top;">SCDRP\$L_CMD_PTR</td> <td style="vertical-align: top;">Address of the port command buffer. The first longword of the port command buffer contains the number of bytes in the buffer (not including the count longword). Subsequent bytes contain the SCSI command descriptor block.</td> </tr> <tr> <td style="vertical-align: top;">SCDRP\$L_BCNT</td> <td style="vertical-align: top;">Size in bytes of the mapped process buffer.</td> </tr> <tr> <td style="vertical-align: top;">SCDRP\$L_SVA_USER</td> <td style="vertical-align: top;">System virtual address of the process buffer as mapped in system space (S0 space).</td> </tr> <tr> <td style="vertical-align: top;">SCDRP\$L_STS_PTR</td> <td style="vertical-align: top;">Address of the status longword. The port driver copies the SCSI status byte it receives in the bus STATUS phase into the low-order byte of this buffer.</td> </tr> <tr> <td style="vertical-align: top;">SCDRP\$W_FUNC</td> <td style="vertical-align: top;">Read or write operation.</td> </tr> </table>	SCDRP\$L_CMD_PTR	Address of the port command buffer. The first longword of the port command buffer contains the number of bytes in the buffer (not including the count longword). Subsequent bytes contain the SCSI command descriptor block.	SCDRP\$L_BCNT	Size in bytes of the mapped process buffer.	SCDRP\$L_SVA_USER	System virtual address of the process buffer as mapped in system space (S0 space).	SCDRP\$L_STS_PTR	Address of the status longword. The port driver copies the SCSI status byte it receives in the bus STATUS phase into the low-order byte of this buffer.	SCDRP\$W_FUNC	Read or write operation.
SCDRP\$L_CMD_PTR	Address of the port command buffer. The first longword of the port command buffer contains the number of bytes in the buffer (not including the count longword). Subsequent bytes contain the SCSI command descriptor block.										
SCDRP\$L_BCNT	Size in bytes of the mapped process buffer.										
SCDRP\$L_SVA_USER	System virtual address of the process buffer as mapped in system space (S0 space).										
SCDRP\$L_STS_PTR	Address of the status longword. The port driver copies the SCSI status byte it receives in the bus STATUS phase into the low-order byte of this buffer.										
SCDRP\$W_FUNC	Read or write operation.										
SCDRP\$L_CDT	Address of the SCDT.										

The port driver returns the following values to the class driver, preserving R3, R4, and R5:

Location	Contents												
R0	Port status. The port driver returns one of the following status values:												
	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%; vertical-align: top;">SS\$_BADPARAM</td> <td style="vertical-align: top;">Bad parameter specified by the class driver.</td> </tr> <tr> <td style="vertical-align: top;">SS\$_CTRLERR</td> <td style="vertical-align: top;">Controller error or port hardware failure.</td> </tr> <tr> <td style="vertical-align: top;">SS\$_DEVACTION</td> <td style="vertical-align: top;">Command outstanding on this connection.</td> </tr> <tr> <td style="vertical-align: top;">SS\$_LINKABORT</td> <td style="vertical-align: top;">Connection no longer exists.</td> </tr> <tr> <td style="vertical-align: top;">SS\$_NORMAL</td> <td style="vertical-align: top;">Normal, successful completion.</td> </tr> <tr> <td style="vertical-align: top;">SS\$_TIMEOUT</td> <td style="vertical-align: top;">Failed during selection or arbitration.</td> </tr> </table>	SS\$_BADPARAM	Bad parameter specified by the class driver.	SS\$_CTRLERR	Controller error or port hardware failure.	SS\$_DEVACTION	Command outstanding on this connection.	SS\$_LINKABORT	Connection no longer exists.	SS\$_NORMAL	Normal, successful completion.	SS\$_TIMEOUT	Failed during selection or arbitration.
SS\$_BADPARAM	Bad parameter specified by the class driver.												
SS\$_CTRLERR	Controller error or port hardware failure.												
SS\$_DEVACTION	Command outstanding on this connection.												
SS\$_LINKABORT	Connection no longer exists.												
SS\$_NORMAL	Normal, successful completion.												
SS\$_TIMEOUT	Failed during selection or arbitration.												

VMS Macros Invoked by SCSI Class Drivers

SPI\$SEND_COMMAND

Location	Contents
R5	Address of the SCDRP. The port driver provides information in the following fields: SCDRP\$L_STS_PTR Address of the status longword. The port driver copies the SCSI status byte it receives in the bus STATUS phase into the low-order byte of this buffer. SCDRP\$L_TRANS_CNT Actual number of bytes sent or received by the port driver during the Data phase.

SPI\$SET_CONNECTION_CHAR

Sets characteristics of an existing connection.

FORMAT SPI\$SET_CONNECTION_CHAR

DESCRIPTION The SPI\$SET_CONNECTION_CHAR macro sets characteristics of an existing SCSI connection. Prior to altering the characteristics of a connection, a SCSI class driver should read and examine the current connection characteristics using the SPI\$GET_CONNECTION_CHAR macro.

The class driver specifies the characteristics to be set for the connection in a connection characteristics buffer. The buffer has the following format:

Longword	Contents						
1	Number of longwords in the buffer, <i>not</i> including this longword. The value of this field must be 10.						
2	Connection flags. Bits in this longword are defined as follows:						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Bit</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td>ENA_DISCON. When set, this bit enables disconnect and reselection on the connection.</td> </tr> <tr> <td style="text-align: center;">1</td> <td>DIS_RETRY. When set, this bit disables command retry on the connection.</td> </tr> </tbody> </table>	Bit	Description	0	ENA_DISCON. When set, this bit enables disconnect and reselection on the connection.	1	DIS_RETRY. When set, this bit disables command retry on the connection.
Bit	Description						
0	ENA_DISCON. When set, this bit enables disconnect and reselection on the connection.						
1	DIS_RETRY. When set, this bit disables command retry on the connection.						
3	Synchronous. When this longword contains 0, the connection uses asynchronous data transfer mode; when it contains a nonzero value, the connection uses synchronous data transfer mode.						
4	Transfer period. If the synchronous parameter is nonzero, this field controls the number of 4-nanosecond ticks between a REQ and an ACK. The default is 64 ₁₀ .						
5	REQ-ACK offset. If the synchronous parameter is nonzero, this field controls the maximum number of REQs outstanding before there must be an ACK.						
6	Busy retry count. Maximum number of retries allowed on this connection while waiting for the port to become free.						
7	Select retry count. Maximum number of retries allowed on this connection while waiting for the port to be selected by the target device.						

VMS Macros Invoked by SCSI Class Drivers

SPI\$SET_CONNECTION_CHAR

Longword	Contents
8	Arbitration retry count. Maximum number of retries allowed on this connection while waiting for the port to win arbitration of the bus.
9	Command retry count. Maximum number of retries allowed on this connection to successfully send a command to the target device.
10	Phase change timeout. Default timeout value (in seconds) for a target to change the SCSI bus phase or complete a data transfer. This value is also known as the DMA timeout. Upon sending the last command byte, the port driver waits this many seconds for the target to change the bus phase lines and assert REQ (indicating a new phase). Or, if the target enters the DATA IN or DATA OUT phase, the transfer must be completed within this interval. If this value is not specified, the default value is 4 seconds.
11	Disconnect timeout. Default timeout value (in seconds) for a target to reselect the initiator to proceed with a disconnected I/O transfer. If this value is not specified, the default value is 4 seconds.

Inputs to the SPI\$SET_CONNECTION_CHAR macro include the following:

Location	Contents
R2	Address of the connection characteristics buffer.
R4	Address of the SPDT.
R5	Address of the SCDRP.
SCDRP\$L_CDT	Address of the SCDT.

The port driver returns the following values to the class driver, preserving R3, R4, and R5:

Location	Contents
R0	Port status. The port driver returns one of the following values: SS\$_NORMAL Normal, successful completion SS\$_NOSUCHID No connection for this SCSI connection ID

SPI\$UNMAP_BUFFER

Releases port mapping resources and deallocates port DMA buffer space, as required to unmap a process buffer.

FORMAT SPI\$UNMAP_BUFFER

DESCRIPTION The SPI\$UNMAP_BUFFER macro releases mapping resources and deallocates port DMA buffer space, as required to unmap a process buffer.

Inputs to the SPI\$UNMAP_BUFFER macro include the following:

Location	Contents
R4	Address of the SPDT.
R5	Address of the SCDRP. The class driver must provide values in the following fields: SCDRP\$W_NUMREG Number of port DMA buffer pages allocated SCDRP\$W_MAPREG Page number of the first port DMA buffer page

The port driver returns the following values to the class driver, preserving R3, R4, and R5:

Location	Contents
R0	SS\$_NORMAL.
R5	Address of the SCDRP. The port driver clears SCDRP\$W_NUMREG and SCDRP\$W_MAPREG.

VMS Macros Invoked by SCSI Class Drivers

B.2 SCSI Port Interface Extension Macros for Asynchronous Event Notification

B.2 SCSI Port Interface Extension Macros for Asynchronous Event Notification

This section describes SPI extensions that support asynchronous event notification.

SPI\$FINISH_COMMAND

Completes an I/O operation initiated with asynchronous event notification.

FORMAT SPI\$FINISH_COMMAND

DESCRIPTION

The SPI\$FINISH_COMMAND macro allows the host acting as a target to send a status byte, return the COMMAND COMPLETE message, and drive the SCSI bus to BUS FREE. The class driver's callback routine should invoke SPI\$FINISH_COMMAND or SPI\$RELEASE_BUS, but not both, before exiting.

The SPI\$FINISH_COMMAND function is a higher-level function that class drivers can use to finish an I/O operation that is executing with asynchronous event notification.

Inputs to the SPI\$FINISH_COMMAND macro include the following:

Location	Contents
R1	Address of the system buffer containing the SCSI status byte
R4	Address of the SPDT
R5	Address of the SCDRP

The port driver returns SS\$_NORMAL status in R0, destroys R2, and preserves all other registers.

VMS Macros Invoked by SCSI Class Drivers

SPI\$RECEIVE_BYTES

SPI\$RECEIVE_BYTES

Receives command, message, and data bytes from a device acting as an initiator on the SCSI bus.

FORMAT SPI\$RECEIVE_BYTES

DESCRIPTION The SPI\$RECEIVE_BYTES macro allows the host to receive information from the device acting as an initiator. A class driver uses SPI\$RECEIVE_BYTES to receive command, message, and data bytes. This macro uses DMA operations for the transfer of large segments of data where appropriate.

Inputs to the SPI\$RECEIVE_BYTES macro include the following:

Location	Contents
R0	Size of the system buffer into which the target returns the requested bytes
R1	Address of the system buffer into which the target device returns the requested bytes
R4	Address of the SPDT

The port driver returns the following values to the class driver, destroying R2, and preserving all other registers:

Location	Contents
R0	Port status. The port driver returns one of the following values: SS\$_NORMAL Normal, successful completion. SS\$_CTRLERR Timeout occurred during the operation.
R1	Actual number of bytes received.

SPI\$RELEASE_BUS

Releases the SCSI bus.

FORMAT SPI\$RELEASE_BUS

DESCRIPTION

The SPI\$RELEASE_BUS macro allows the host acting as a target to release the SCSI bus. The class driver's callback routine should invoke either SPI\$RELEASE_BUS or SPI\$FINISH_COMMAND, but not both, before exiting.

The class driver should use SPI\$RELEASE_BUS instead of SPI\$FINISH_COMMAND if it must explicitly send the SCSI status byte and COMMAND COMPLETE message using SPI\$SEND_BYTES, or if it simply wants to drop off the bus and terminate the thread in certain error conditions.

Inputs to the SPI\$RELEASE_BUS macro include the following:

Location	Contents
R4	Address of the SPDT
R5	Address of the SCDRP

The port driver returns SS\$_NORMAL status in R0, destroys R2, and preserves all other registers.

VMS Macros Invoked by SCSI Class Drivers

SPI\$SEND_BYTES

SPI\$SEND_BYTES

Sends command, message, and data bytes to a device acting as an initiator on the SCSI bus.

FORMAT SPI\$SEND_BYTES

DESCRIPTION

The SPI\$SEND_BYTES macro allows the host to send information to the device acting as an initiator. A class driver uses SPI\$SEND_BYTES to send command, message, and data bytes. This macro uses DMA operations for the transfer of large segments of data where appropriate.

Inputs to the SPI\$SEND_BYTES macro include the following:

Location	Contents
R0	Size of the system buffer that contains the bytes to be sent
R1	Address of the system buffer that contains the bytes to be sent
R4	Address of the SPDT

The port driver returns the following values to the class driver, destroying R2, and preserving all other registers:

Location	Contents
R0	Port status. The port driver returns one of the following values: SS\$_NORMAL Normal, successful completion. SS\$_CTRLERR Timeout occurred during the operation.
R1	Actual number of bytes sent.

VMS Macros Invoked by SCSI Class Drivers
SPI\$SENSE_PHASE

SPI\$SENSE_PHASE

Returns the current phase of the SCSI bus.

FORMAT SPI\$SENSE_PHASE

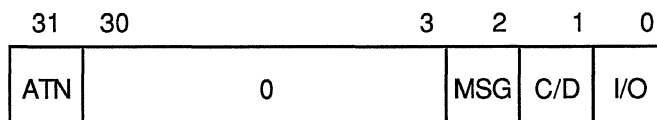
DESCRIPTION The SPI\$SENSE_PHASE macro allows the host to read the current SCSI bus phase, and the state of the ATN signal, while using the asynchronous event notification feature.

A class driver must supply the address of the SPDT in R4 as input to the SPI\$SENSE_PHASE macro.

The port driver returns the following values to the class driver, destroying R2, and preserving all other registers:

Location	Contents
R0	SS\$_NORMAL.
R1	SCSI bus phase (and ATN signal). This SCSI-defined longword has the format illustrated in Figure B-1.

Figure B-1 SCSI Bus Phase Longword Returned to SPI\$SENSE_PHASE



ZK-1377A-GE

VMS Macros Invoked by SCSI Class Drivers

SPI\$SET_PHASE

SPI\$SET_PHASE

Sets the bus to a new phase.

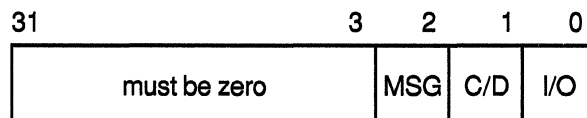
FORMAT SPI\$SET_PHASE

DESCRIPTION The SPI\$SET_PHASE macro allows the host to set the SCSI bus to a new phase. A class driver uses this macro to drive the phase transitions of the SCSI bus while using the asynchronous event notification feature.

Inputs to the SPI\$SET_PHASE macro include the following:

Location	Contents
R0	New SCSI bus phase. This SCSI-defined longword has the format shown in Figure B-2.
R4	Address of the SPDT.

Figure B-2 SCSI Bus Phase Longword Supplied to SPI\$SET_PHASE



ZK-1376A-GE

The port driver returns SS\$_NORMAL status in R0, destroys R2, and preserves all other registers.

C

VMS Template SCSI Class Driver

This appendix lists the contents of the VMS SCSI template class driver. The code in this template can serve as the starting point for a new third-party SCSI class driver. You can obtain a machine-readable copy of this driver from SYS\$EXAMPLES:SKDRIVER.MAR.

```
.TITLE SKDRIVER - VAX/VMS Sample SCSI Class Driver
.IDENT 'X-3'
; .LIST MEB
;*****
;*
;* COPYRIGHT (c) 1989 BY
;* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
;* ALL RIGHTS RESERVED.
;*
;* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
;* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
;* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
;* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
;* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
;* TRANSFERRED.
;*
;* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
;* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
;* CORPORATION.
;*
;* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
;* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
;*
;*
;*****
;
;
;
; FACILITY:
;
; VAX/VMS Sample SCSI Class Driver
;
; ABSTRACT:
;
; This module contains a sample SCSI class driver. This template
; supports two modes of operation: either the SCSI command
; packets are formatted in the application program (passthru mode) or
; the SCSI command packets are formatted within the driver. In the
; latter case, command processing and error recovery are implemented
; within a third-party SCSI class driver derived from this driver.
;
; Passthru mode is the method of access used by the generic SCSI
; class driver (GKDRIVER). GKDRIVER provides access to a SCSI device
; from an application program. The QIO interface of the GKDRIVER
; is fixed. However, third-party SCSI class drivers can have device
; specific QIO interface. Third-party class drivers can have device
; specific error recovery, log device errors and implement asynchronous
; event notification (AEN). Third-party class drivers have direct access
; to the SCSI Port Interface (SPI) routines, while using the passthru
```

VMS Template SCSI Class Driver

```
;      function provides access to SCSI without writing a driver.
;
;      The code to perform the IO$_DIAGNOSE function is included in this
;      driver for informational purposes only. Typical third-party SCSI
;      class drivers do not require this function. If the IO$_DIAGNOSE
;      function is required, you should use the VMS-supported SCSI
;      generic class driver (GKDRIVER).
;
;      SKDRIVER supports three I/O functions:
;
;          IO$_AVAILABLE - Inquiry and Test Unit Ready sequence,
;          IO$_DIAGNOSE  - Passthru function
;          IO$_READLBLK  - Return Inquiry data to user
;
;
;      .SBTTL +
;      .SBTTL + SYMBOL DEFINITIONS
;      .SBTTL +
;      .SBTTL External symbol definitions
;+
; External symbols
;-
;
;      $CRBDEF          ; Channel request block
;      $DDBDEF          ; Device data block
;      $DEVDEF          ; Device characteristics
;      $EMBDEF          ; Error log message buffer
;      $DYNDEF          ; Data structure types
;      $FKBDEF          ; Define fork block symbols
;      $IODEF           ; I/O function codes
;      $IPLDEF          ; Hardware IPL definitions
;      $IRPDEF          ; I/O request packet
;      $PCBDEF          ; Process control block
;      $PRVDEF          ; Privilege mask
;      $SCDRPDEF        ; SCSI SCDRP symbols
;      $SSDEF           ; System status codes
;      $UCBDEF          ; Unit control block
;      $VECDEF          ; Interrupt vector block
;
;      .SBTTL Miscellaneous local symbols
;+
; Local symbols
;-
;+
; Argument list (AP) offsets for device-dependent QIO parameters
;-
P1      = 0             ; First QIO parameter
P2      = 4             ; Second QIO parameter
P3      = 8             ; Third QIO parameter
P4      = 12            ; Fourth QIO parameter
P5      = 16            ; Fifth QIO parameter
P6      = 20            ; Sixth QIO parameter
```

VMS Template SCSI Class Driver

```
SCDRPS_PER_UNIT      = 2           ; Number of SCDRPs to allocate per unit
UCB_STACK_SIZE       = 10          ; Size of internal stack in UCB
MAX_BCNT              = ^XFFFF     ; Maximum byte count
.IIF NDF DT$_GENERIC SCSI, DT$_GENERIC SCSI = 5 ; GENERIC SCSI DEVICE
ASSEMBLE_PASSTHRU    = 0           ; If 0 don't assemble DIAG code, if 1 do.
SCSI$M_STS            = ^XC1       ; Used to extract vendor unique STS bits.
DIAG_BUF_LEN         = 60          ; Length in bytes of DIAGNOSE input buffer.
MAX_CMD_LEN          = 248         ; Maximum size in bytes of a SCSI CMD.
INQ_DATA_LEN         = 36          ; Exact number of INQUIRY bytes required.
NUM_ARGS             = 10          ; Number of SET/GET CONNECTION CHAR arguments.
```

```
.SBTTL SCSI Peripheral Device Types
```

```
;++
; Define SCSI Peripheral Device Types
;--
SCSI_C_DA             = 0           ; Direct Access
SCSI_C_SA             = 1           ; Sequential Access
SCSI_C_PT             = 2           ; Printer
SCSI_C_PR             = 3           ; Processor
SCSI_C_WR             = 4           ; Write-once Read-multiple
SCSI_C_RO             = 5           ; Read-only direct access
```

```
.SBTTL Sense key codes
```

```
;++
; Define SCSI sense key codes.
;--
SCSI_C_NO_SENSE       = 0           ; No sense data
SCSI_C_RECOVERED_ERROR = 1         ; Recovered error (treated as success)
SCSI_C_NOT_READY      = 2           ; Device not ready
SCSI_C_MEDIUM_ERROR   = 3           ; Medium (parity) error
SCSI_C_HARDWARE_ERROR = 4           ; Hardware error
SCSI_C_ILLEGAL_REQUEST = 5         ; Illegal request
SCSI_C_UNIT_ATTENTION = 6           ; Unit attention (media change, reset)
SCSI_C_DATA_PROTECT   = 7           ; Data protection (write lock error)
SCSI_C_BLANK_CHECK    = 8           ; Blank check (advance past end of data)
SCSI_C_VENDOR_UNIQUE  = 9           ; Vendor unique key
SCSI_C_COPY_ABORTED   = 10          ; Copy operation aborted
SCSI_C_ABORTED_COMMAND = 11        ; Command aborted
SCSI_C_EQUAL          = 12          ; Compare operation, data match
SCSI_C_VOLUME_OVERFLOW = 13        ; Write beyond physical end of tape
SCSI_C_MISCOMPARE     = 14         ; Compare operation, data mismatch
```

```
;++
; Define offsets in various SCSI command packets.
;--
```

VMS Template SCSI Class Driver

```
;+
; REQUEST SENSE data offsets.
;-
    SCSI_XS_B_ERR_CODE      = 0      ; Extended sense error code
    SCSI_XS_B_KEY           = 2      ; Extended sense KEY field
    SCSI_XS_V_KEY          = 0      ; Extended sense KEY bit number
    SCSI_XS_S_KEY          = 4      ; Extended sense KEY length
    SCSI_XS_B_ADDNL_INFO   = 3      ; Extended sense additional code
    SCSI_XS_B_ADDNL_CODE   = 12     ; Extended sense additional code
    SCSI_XS_B_ADDNL_CODE30 = 8      ; " (TZ30)
    SCSI_XS_B_ADDNL_CODE50 = 8      ; " (TZK50)
    SCSI_XS_M_EOF          = ^X80   ; Extended sense end of file
    SCSI_XS_M_EOM          = ^X40   ; Extended sense end of medium
    SCSI_XS_M_ILI          = ^X20   ; Extended sense illegal length indicator
    SCSI_XS_V_ADDNL_VALID  = 7      ; Extended sense additional data valid

;+
; INQUIRY data offsets.
;-
    SCSI_INQ_B_DEVTYPE     = 0      ; Inquiry device type
    SCSI_INQ_B_DEVQUAL     = 1      ; Inquiry device qualifier field
    SCSI_INQ_V_DEVQUAL     = 0      ; Inquiry device qualifier starting bit
    SCSI_INQ_S_DEVQUAL     = 7      ; Inquiry device qualifier length

    SCSI_INQ_V_REMOVABLE   = 7      ; Inquiry removable bit

    SCSI_SKIP_B_CNT       = 2      ; Skip record count

;+
; MODE SELECT/SENSE data offsets.
;-
    SCSI_MSNS_B_WP        = 2      ; Mode sense write protect field
    SCSI_MSNS_V_WP        = 7      ; Mode sense write protect bit

    SCSI_MSEL_W_RSVD0     = 0      ; Mode select reserved
    SCSI_MSEL_B_SPEED     = 2      ; Mode select speed field
    SCSI_MSEL_B_MODE      = 2      ; Mode select buffered mode
    SCSI_MSEL_B_DSCLLEN   = 3      ; Mode select record descriptor length
    SCSI_MSEL_C_DSCLLEN   = 8      ; Mode select record descriptor length
    SCSI_MSEL_B_DENS      = 4      ; Mode select density
    SCSI_MSEL_B_BLOCKS    = 5      ; Mode select number of blocks
    SCSI_MSEL_B_RSVD1     = 8      ; Mode select reserved
    SCSI_MSEL_B_BLKLEN    = 9      ; Mode select block length
    SCSI_MSEL_B_VULEN     = 12     ; Mode select vendor unique length
    SCSI_MSEL_B_VU        = 13     ; Mode select vendor unique field
    SCSI_MSEL_M_BUF       = ^X10   ; Mode select buffered mode
    SCSI_MSEL_M_NOF       = 7      ; Number of fillers for generic device
    SCSI_MSEL_M_NOF50     = 7      ; Number of fillers for TZK50
    SCSI_MSEL_M_NOF30     = ^X0F   ; Number of fillers for TZ30
    SCSI_MSEL_M_RESEL     = ^X40   ; Reselection timeout flag

;+
; SPI interface, Get/set connect characteristics symbols.
;-
    SET_CON_L_LEN         = 0      ; Length field
    SET_CON_L_CON_FLAGS   = 4      ; Flags field
    SET_CON_M_DISC        = 1      ; Enable disconnect flag
    SET_CON_M_NORETRY     = 2      ; Disable command retry flag
    SET_CON_L_SYN_FLAG    = 8      ; Synchronous flag field
    SET_CON_M_SYN         = 1      ; Synchronous flag
```

VMS Template SCSI Class Driver

```

.SBTTL Template class driver extensions to the UCB
;+
; Template class driver extensions to the UCB.
;-
$DEFINI UCB                                ; Start of UCB definitions
      . = UCB$K_LCL_DISK_LENGTH            ; Position at end of UCB
$DEF  UCB_L_STACK_PTR .BLKL 1              ; Internal stack pointer
$DEF  UCB_L_STACK     .BLKL UCB_STACK_SIZE ; Internal stack
$DEF  UCB_L_SCDRP     .BLKL 1              ; Address of active SCDRP
$DEF  UCB_L_SCDT      .BLKL 1              ; SCDT address
$DEF  UCB_L_SK_FLAGS  .BLKL 1              ; Class driver flags
      _VIELD UCB,0,<-                      ;
          <DISCONNECT,,M>,-                ; Device supports disconnect
          <DISABL_ERRLOG,,M>,-             ; Disable error logging
          <SYNCHRONOUS,,M>>               ; Device supports synchronous operation
$DEF  UCB_W_PHASE_TMO .BLKW 1              ; Phase change timeout
$DEF  UCB_W_DISC_TMO  .BLKW 1              ; Disconnect timeout
$DEF  UCB_L_SCDRPQ_FL .BLKL 1              ; Queue of free SCDRPs used to
$DEF  UCB_L_SCDRPQ_BL .BLKL 1              ; send SCSI commands
$DEF  UCB_L_SAVER6    .BLKL 1              ; Safe place for R6.
$DEF  UCB_L_SAVER7    .BLKL 1              ; Safe place for R7.
$DEF  UCB_L_SCDRP_SAV1 .BLKL 1             ; Safe place for SCDRP address.
$DEF  UCB_B_LUN       .BLKB 1              ; Save device LUN
$DEF  UCB_K_SK_UCBLEN ; Length of extended UCB
$DEFEND UCB                                ; End of UCB definitions

.SBTTL Error log packet formats
;+
; The following are the definitions for class driver error log packets.
; The VMS error log formatter formats third-party SCSI class driver
; error log packets. The ERF utility formats a standard error
; log packet for third-party class drivers. The standard packet is defined
; below. If a user would like to dump additional data to the error log, simply
; increase the size of the error log packet defined. The additional data
; will be dumped as untranslated longwords in the error log.
;-
$DEFINI ERROR_PACKETS
      . = EMB$L_DV_REGSAV                  ; Start of area to dump error info
$DEF  ERR_LW_CNT      .BLKL 1              ; Count of number of longwords that follow
$DEF  ERR_REVISION    .BLKB 1              ; Revision level
$DEF  ERR_HW_REV      .BLKL 1              ; Hardware revision
$DEF  ERR_TYPE        .BLKB 1              ; Error type
$DEF  ERR_SCSI_ID     .BLKB 1              ; SCSI ID
$DEF  ERR_SCSI_LUN    .BLKB 1              ; SCSI logical unit
$DEF  ERR_SCSI_SUBLUN .BLKB 1              ; SCSI sublogical unit
$DEF  ERR_PORT_STATUS .BLKL 1              ; Port status code
$DEF  ERR_CMD_LEN     .BLKB 1              ; SCSI command length field
$DEF  ERR_CMD_BYTES   .BLKB 12             ; Maximum possible command bytes
$DEF  ERR_SCSI_STS    .BLKB 1              ; SCSI status byte
$DEF  ERR_TXT_LEN     .BLKB 1              ; Error message text size
$DEF  ERR_TXT_BYTES   .BLKB 60            ; Maximum possible text bytes
      . = .+4                              ; Reserve one longword after end of defined
      ; packet.
$DEF  ERR_K_COMMAND_LENGTH ; Length of packet containing SCSI command
$DEFEND ERROR_PACKETS

```

VMS Template SCSI Class Driver

```
.SBTTL SCSI Class driver error log types.
;+
; SCSI class driver error log types. Each error that is logged by the
; class driver should have a unique error type.
;-
CLS_DRV_ERROR_01 = 1           ; Class driver specific error type.
CLS_DRV_ERROR_02 = 2           ; Class driver specific error type.
CLS_DRV_ERROR_03 = 3           ; Class driver specific error type.
CLS_DRV_ERROR_04 = 4           ; Class driver specific error type.
CLS_DRV_ERROR_05 = 5           ; Class driver specific error type.
CLS_DRV_ERROR_06 = 6           ; Class driver specific error type.

.SBTTL +
.SBTTL + MACRO DEFINITIONS
.SBTTL +
.SBTTL SCSI_CMD             - Define a SCSI command packet

;+
; SCSI_CMD
;
; This macro defines the contents of a SCSI command packet. Each SCSI command
; can have associated with it a DMA buffer used during the DATAIN/DATAOUT bus
; phases. A DMA length of zero indicates there is no DATA(IN/OUT) phase
; associated with this command (except in the case of a read/write SCSI command,
; which is handled specially.)
; Class drivers can specify on a command by command basis the DMA Timeout and
; Disconnect Timeout values. The disconnect timeout is the maximum number
; of seconds that an I/O can be disconnected from the bus. A timeout of -1
; allows an infinite timeout. The DMA timeout is the maximum timeout for
; a DMA transfer to complete or a phase change on the SCSI bus to occur;
; this timeout is also in units of seconds.
; The SETUP_CMD routine uses this information in preparing to send a SCSI
; command. The macro generates a label and the SCSI command information as
; follows:
;
;
; +-----+
; | SCSI cmd length | 1 byte
; +-----+
; | SCSI cmd bytes  | n bytes
; +-----+
; | DMA buffer length | 2 bytes
; +-----+
; | DMA direction   | 1 byte
; +-----+
; | DMA Timeout     | 1 longword
; +-----+
; | Disconnect Timeout | 1 longword
; +-----+
;
;
; DMA direction is defined as: 0=write, 1=read.
;-
.MACRO SCSI_CMD, NAME, CMD_BYTES, DMA_LEN=0, DMA_DIR=READ,-
DMA_TMO=0, DISCON_TMO=0
'NAME' _CMD:
$$$BYTE_COUNT=0
.IRP CMD_BYTE, <CMD_BYTES>
$$$BYTE_COUNT = $$$BYTE_COUNT + 1
.IIF EQ $$$BYTE_COUNT-1, SCSI_C_'NAME' = CMD_BYTE           ; Define opcode
.ENDR
.BYTE $$$BYTE_COUNT
.IRP CMD_BYTE, <CMD_BYTES>
.BYTE CMD_BYTE
```

VMS Template SCSI Class Driver

```

.ENDR
.WORD DMA_LEN
$$$DIRECTION = 0
.IIF IDN DMA_DIR, READ, $$$DIRECTION = 1
.BYTE $$$DIRECTION
.LONG DMA_TMO
.LONG DISCON_TMO
.ENDM SCSI_CMD

.SBTTL LOG_ERROR - Log a SCSI class driver error
;+
; LOG_ERROR
;
; This macro logs a SCSI class driver error. The error type and VMS status
; code are placed in R7 and R8 respectively, and the LOG_ERROR routine is
; called.
;-

.MACRO LOG_ERROR,TYPE,VMS_STATUS,UCB=R3,MESSAGE='',?LABEL_1
.SHOW EXPANSIONS
PUSHR #^M<R5,R7,R8,R11> ; Save registers
.IF DIF UCB,R5
MOVL UCB,R5 ; Get UCB address
.ENDC
MOVL #'TYPE',R7 ; Get error code
MOVL VMS_STATUS,R8 ; And VMS status code
.IF LESS_THAN 60-%LENGTH(MESSAGE) ; Maximum size message is 60
.ERROR ; Message text is greater than 60 characters
.ENDC
.SAVE_PSECT LOCAL_BLOCK
.PSECT $$$111_TEXT
LABEL_1:
.ASCIC //'MESSAGE'/
.RESTORE_PSECT
MOVAL LABEL_1,R11
BSBW LOG_ERROR ; Write an error log entry
POPR #^M<R5,R7,R8,R11> ; Restore registers
.NOSHOW EXPANSIONS

.ENDM LOG_ERROR

.SBTTL WORD_BRANCHES - Define word displacement branches
;+
; WORD_BRANCHES
;
; This macro defines for each Bxxx (conditional branch) instruction an equivalent
; macro named BxxxW with a word displacement. The macro takes as an argument
; a list of tuples, each tuple containing 3 items: 1) a conditional branch
; opcode; 2) the opcode with the opposite polarity; and 3) the number of
; arguments required by the opcode.
;-

.MACRO WORD_BRANCHES LIST
.MACRO WORD_BRANCHES2, OPCODE1, OPCODE2, ARGCNT
.IF EQ ARGCNT-0
.MACRO OPCODE1, DST, ?L
OPCODE2 L
BRW DST
L: .ENDM OPCODE1
.ENDC

```


VMS Template SCSI Class Driver

```

        .IF EQ ARGCNT-1
        .MACRO OPCODE1, FIELD, DST, ?L
        OPCODE2 FIELD,L
        BRW     DST
L:      .ENDM  OPCODE1
        .ENDC

        .IF EQ ARGCNT-2
        .MACRO OPCODE1, BIT, FIELD, DST, ?L

        OPCODE2 BIT,FIELD,L
        BRW     DST
L:      .ENDM  OPCODE1
        .ENDC

        .ENDM  WORD_BRANCHES2

        .MACRO WORD_BRANCHES1, OPCODE1, OPCODE2, ARGCNT

        WORD_BRANCHES2 'OPCODE1'W, OPCODE2, ARGCNT
        WORD_BRANCHES2 'OPCODE2'W, OPCODE1, ARGCNT

        .ENDM  WORD_BRANCHES1

        .IRP   ENTRY, <LIST>
        WORD_BRANCHES1 ENTRY
        .ENDR

        .ENDM  WORD_BRANCHES

        WORD_BRANCHES <-
                <BBC,   BBS,   2>,-
                <BBCC,  BBSC,  2>,-
                <BBCS,  BBSS,  2>,-
                <BCC,   BCS,   0>,-
                <BEQL,  BNEQ,  0>,-
                <BEQLU, BNEQU, 0>,-
                <BGEQ,  BLSS,  0>,-
                <BGEQU, BLSSU, 0>,-
                <BGTR,  BLEQ,  0>,-
                <BGTRU, BLEQU, 0>,-
                <BLBC,  BLBS,  1>,-
                <BVC,   BVS,   0>>

        .SBTTL  INIT_UCB_STACK  - Initialize the internal UCB stack
        .SBTTL  SUBPUSH         - Push an item on the UCB stack
        .SBTTL  SUBPOP          - Pop an item from the UCB stack
        .SBTTL  SUBSAVE         - Save a return address on the UCB stack
        .SBTTL  SUBRETURN      - Return to the address saved on the UCB stack
;+
; INIT_UCB_STACK
; SUBPUSH
; SUBPOP
; SUBSAVE
; SUBRETURN
;
; These macros manipulate the UCB internal stack, which is used to save
; routine return address and temporary variables.
;-
        .MACRO  INIT_UCB_STACK,UCB=R5,?L1

        MOVAL   UCB_L_STACK-4(UCB),-
                UCB_L_STACK_PTR(UCB)

        .ENDM   INIT_UCB_STACK

        .MACRO  SUBPUSH,ARG,UCB=R3,?L1,?L2

```

VMS Template SCSI Class Driver

```

        ADDL    #4,UCB_L_STACK_PTR(UCB)
        MOVL   ARG,@UCB_L_STACK_PTR(UCB)

        .ENDM   SUBPUSH

        .MACRO  SUBPOP,ARG,UCB=R3,?L1,?L2
        MOVL   @UCB_L_STACK_PTR(UCB),ARG
        SUBL   #4,UCB_L_STACK_PTR(UCB)
        .ENDM   SUBPOP

        .MACRO  SUBSAVE,UCB=R3,?L1,?L2
        SUBPUSH (SP)+,UCB
        .ENDM   SUBSAVE

        .MACRO  SUBRETURN,UCB=R3,?L1,?L2
        SUBPOP -(SP),UCB
        RSB
        .ENDM   SUBRETURN

        .SBTTL  SK_WAIT          - Stall a thread for a specific number of seconds
;+
; SK_WAIT
;
; This macro uses the device timeout mechanism to stall a thread for a specified
; number of seconds. The UCB address and stall time are required as inputs.
;-
        .MACRO  SK_WAIT,SECONDS,UCB=R5,SCRATCH=R0,?L
        .IF DIF UCB,R5
        MOVL   R5,SCRATCH
        MOVL   UCB,R5
        MOVL   SCRATCH,UCB
        .ENDC
        DSBINT ENVIRON=UNIPROCESSOR
        PUSHL  SECONDS
        BSBW   SK_WAIT
        .WORD  L-.
L:      IOFORK
        BICW   #UCB$M_TIMEOUT,-
            UCB$W_STS(R5)
        .IF DIF UCB,R5
        MOVL   UCB,SCRATCH
        MOVL   R5,UCB
        MOVL   SCRATCH,R5
        .ENDC
        .ENDM   SK_WAIT

        .SBTTL  +
        .SBTTL  + DRIVER TABLES
        .SBTTL  +
        .SBTTL  Driver prologue table
;+
; Driver prologue table
;
; This table provides various information about the driver, such as its name
; and length, and causes initialization of various fields in the I/O database
; when the driver is loaded.
;-
        .IIF NDF DPT$M_NO_IDB_DISPATCH, DPT$M_NO_IDB_DISPATCH = ^X1000

```

VMS Template SCSI Class Driver

```

DPTAB      -                               ; DPT-creation macro
          END=SK_END,-                       ; End of driver label
          ADAPTER=NULL,-                     ; Adapter type
          UCBSIZE=<UCB_K_SK_UCBLEN>,-       ; Length of UCB
          NAME=SKDRIVER,-                   ; Driver name
          FLAGS=<DPT$M_SMPMOD!-             ; Driver runs in SMP environment
          DPT$M_NO_IDB_DISPATCH>           ; Don't fill in IDB$L_UCBLST
DPT_STORE INIT                               ; Start of load
          ; initialization table
DPT_STORE UCB,UCB$L_MAXBCNT,L,MAX_BCNT      ; Maximum byte count
DPT_STORE UCB,UCB$B_FLCK,B,SPL$C_IOLOCK8    ; Device FORK LOCK
DPT_STORE UCB,UCB$B_DIPL,B,22               ; Device interrupt IPL
DPT_STORE UCB,UCB$L_DEVCHAR,L,<-           ; Device characteristics
          DEV$M_AVL!-                         ; Available
          DEV$M_IDV!-                         ; Input device
          DEV$M_ODV!-                         ; Output device
          DEV$M_SHR!-                         ; Shareable Device
          DEV$M_ELG!-                         ; Error logging enabled
          DEV$M_RND>                          ; Random Access Device
DPT_STORE UCB,UCB$L_DEVCHAR2,L,<-           ; Device characteristics
          DEV$M_NNM>                          ; Prefix name with "node$"
DPT_STORE UCB,UCB$B_DEVTYPE,B,DT$_GENERIC_SCSI ; Generic SCSI device
DPT_STORE UCB,UCB$B_DEVCLASS,B,DC$_MISC     ; Sample device class
DPT_STORE UCB,UCB$W_DEVSTS,W,-             ; Set no logical to physical
          UCB$M_NOCNVRT                       ; block number conversion
DPT_STORE REINIT                             ; Start of reload
          ; initialization table
DPT_STORE DDB,DDB$L_DDT,D,SK$DDT           ; Address of DDT
DPT_STORE CRB,-                             ; Address of controller
          CRB$L_INTD+VEC$L_INITIAL,-         ; initialization routine
          D,SK_CTRL_INIT
DPT_STORE CRB,-                             ; Address of device
          CRB$L_INTD+VEC$L_UNITINIT,-       ; unit initialization
          D,SK_UNIT_INIT                     ; routine
DPT_STORE CRB,CRB$B_FLCK,B,IPL$_IOLOCK8    ; Initialize fork lock field
DPT_STORE END                               ; End of initialization
          ; tables

.SBTTL Driver dispatch table
;+
; Driver dispatch table
;
; This table defines the entry points into the driver.
;-

DDTAB      -                               ; DDT-creation macro
          DEVNAM=SK,-                         ; Name of device
          START=SK_STARTIO,-                 ; Start I/O routine
          FUNCTB=SK_FUNCNTABLE,-            ; FDT address
          REGDMP=SK_REG_DUMP                 ; Register dump routine
.SBTTL Function decision table
;+
; Function decision table
;
; This table lists the $QIO function codes implemented by the driver and the
; preprocessing routines used by each function.
;-

```

VMS Template SCSI Class Driver

```
SK_FUNC_TABLE:                                ; FDT for driver
    FUNCTAB,-                                  ; Valid I/O functions
        <AVAILABLE,-                          ; Inquiry and Test Unit Ready
        READLBLE,-                            ; Perform a "read" function
        READVBLE,-                            ; Perform a "read" function
        DIAGNOSE>                             ; Special pass-through function

    FUNCTAB,<>                                  ; Buffered I/O functions

    FUNCTAB SK_READ,<READLBLE,READVBLE>        ; Issue SCSI INQUIRY command.
    FUNCTAB +EXE$ZEROPARM,<AVAILABLE>         ; Issue SCSI INQUIRY command.
    FUNCTAB SK_DIAGNOSE,<DIAGNOSE>           ; Special pass-through function

.SBTTL SCSI Command Packet Definition Table

SK_CMD_DEFS:
    SCSI_CMD -
        NAME = TEST_UNIT_READY,-
        CMD_BYTES = <0, 0, 0, 0, 0, 0>

    SCSI_CMD NAME = INQUIRY,-
        CMD_BYTES = <18 , 0, 0, 0, 36, 0>,-
        DMA_LEN = 36,-
        DMA_DIR = READ,-
        DMA_TMO = 0,-                          ; Use default
        DISCON_TMO = 0                          ; Use default

    SCSI_CMD NAME = REQUEST_SENSE,-
        CMD_BYTES = <3, 0, 0, 0, 18, 0>,-
        DMA_LEN = 18,-
        DMA_DIR = READ,-
        DMA_TMO = 0,-                          ; Use default
        DISCON_TMO = 0                          ; Use default

    SCSI_CMD NAME = MODE_SELECT,-
        CMD_BYTES = <21, 0, 0, 0, 4, 0>,-
        DMA_LEN = 4,-
        DMA_DIR = WRITE,-
        DMA_TMO = 0,-                          ; Use default
        DISCON_TMO = 0                          ; Use default

    SCSI_CMD NAME = QIO_INQUIRY,-              ; Normally this would be
        CMD_BYTES = <18 , 0, 0, 0, 0, 0>,-; a read/write command.
        DMA_LEN = -1,-                          ; If data goes to user buffer,
        DMA_DIR = READ,-                        ; then use -1 here.
        DMA_TMO = 0,-                          ; Use default
        DISCON_TMO = 0                          ; Use default

SK_CMD_DEFS_END = .

.SBTTL +
.SBTTL + DRIVER ENTRY POINTS
.SBTTL +
```

VMS Template SCSI Class Driver

```
.SBTTL SK_CTRL_INIT      - Controller initialization routine
; ++
; SK_CTRL_INIT
;
; This routine is called to perform controller-specific initialization and
; is called by the operating system in three places:
;
;   - at system startup
;   - during driver loading and reloading
;   - during recovery from a power failure
;
; Currently this routine is a NOP.
;
; INPUTS:
;
;   R4      - address of the CSR (controller status register)
;   R5      - address of the IDB (interrupt data block)
;   R6      - address of the DDB (device data block)
;   R8      - address of the CRB (channel request block)
;
; OUTPUTS:
;
;   All registers preserved
; --
SK_CTRL_INIT:
    MOVZWL  #SS$_NORMAL,R0      ; Set success status
    RSB                    ; Return to caller

.SBTTL SK_UNIT_INIT      - Unit initialization routine
; ++
; SK_UNIT_INIT
;
; This routine allocates a set of SCDRPs and places them on a queue in the
; UCB, forms a connection to the port driver by calling SPI$CONNECT, and
; sets the unit online.
;
; INPUTS:
;
;   R5      - UCB address
;
; OUTPUTS:
;
;   R0-R3   - Destroyed
;   All other registers preserved
; --
SK_UNIT_INIT:
                                ; Initialize unit
    BBC     #UCB$_POWER,-        ; Branch if we're not here due to a
                                ; powerfail
    RSB                    ; Otherwise, exit immediately

; +
; Fork twice for now to allow the port driver's unit init routine to execute
; before ours.
; -
2$:   FORK                    ; Fork to drop IPL to SYNCH
      FORK                    ; 2nd Fork synchronizes with port driver.
      INIT_UCB_STACK          ; Initialize the internal stack in the UCB
```

VMS Template SCSI Class Driver

```

        MOVAL   UCB_L_SCDRPQ_FL(R5),R0   ; Initialize the SCDRP queue header
        MOVL    R0,(R0)                  ; in the UCB
        MOVL    R0,4(R0)                 ;
        MOVL    #SCDRPS_PER_UNIT,R4     ; Number of SCDRPs allocated per unit
10$:   MOVL    #<SCDRP$C_LENGTH>,R1     ; Length of SCDRP
        MOVL    R5,R3                   ; Copy UCB address
        BSBW    ALLOC_POOL              ; Go allocate an SCDRP
        MOVW    R1,SCDRP$W_SCDRPSIZE(R2); Save length of SCDRP
        INSQUE  SCDRP$L_FQFL(R2),-      ; Place SCDRP in UCB queue
        UCB_L_SCDRPQ_FL(R5)            ;
        SOBGTR  R4,10$                  ; Repeat for all SCDRPs

;+
; All SCSI device unit numbers should be of the form "n0m" where n is the SCSI
; ID between 0 and 7 and m is the LUN between 0 and 7. Extract the ID from the
; LUN by dividing the unit number by 100. The quotient is then used as the ID
; while the remainder is the LUN. Note that the unit number contains three
; digits because early versions of SCSI provided for sublogical unit numbers.
; This feature has since been removed and the second digit in the unit number
; is not used.
;-
        MOVZWL  #SS$_BADPARAM,R0        ; Assume bad LUN or SUBLUN specified
        MOVZWL  UCB$W_UNIT(R5),R1       ; Get device unit number
        CLRL    R2                      ; Prepare for extended divide
        EDIV    #100,R1,R1,R2           ; Extract SCSI bus ID from LUN
        CMLPL  R1,#7                    ; Valid SCSI ID (0 <= n <= 7)?
        BGTRUW  20$                     ; Branch if not
        CMLPL  R2,#7                    ; Valid LUN (0 <= n <= 7)?
        BGTRUW  20$                     ; Branch if not
        MULB3   #<1@5>,R2,UCB_B_LUN(R5); Save LUN (shifted left 5 bits for use
        ; later in SETUP_CMD)
        ASHL    #16,R1,R1               ; Place SCSI ID in high-order word of R1
        ASHL    #16,R2,R2               ; Place LUN in high-order word of R2
        MOVL    UCB$L_DDB(R5),R0        ; Get DDB address
        SUBB3   #^A'A',-                ; Translate controller letter to
        DDB$_T_NAME+3(R0),R1           ; SCSI bus ID
        SPI$CONNECT                       ; Connect to the port driver
        BLBC    R0,20$                  ; Branch if connect attempt failed
        CMLPL  R1,UCB$L_MAXBCNT(R5)     ; For MAXBCNT, use minimum supported
        BGEQ    15$                     ; value of port and class drivers
        MOVL    R1,UCB$L_MAXBCNT(R5)    ; Save maximum byte count in UCB
15$:   MOVL    R2,UCB_L_SCDT(R5)        ; Save SCDT address
        MOVL    R4,UCB$L_PDT(R5)        ; Save PDT address

        BISW    #UCB$_M_ONLINE,-        ; Set unit online
        UCB$W_STS(R5)                   ;
20$:   RSB                               ; Return to caller

        .SBTTL  +
        .SBTTL  + QIO FDT INTERFACE ROUTINES
        .SBTTL  +

        .SBTTL  SK_READ                 - FDT preprocessing for sending SCSI Inquiry command.
;+
; SK_READ
;
; This routine performs FDT preprocessing including:

```

VMS Template SCSI Class Driver

```
;
;   - Validating access to, and locking, the read/write buffer
;
; INPUTS:
;
;   R0      - Address of FDT routine
;   R3      - IRP address
;   R4      - PCB address
;   R5      - UCB address
;   R6      - CCB address
;   R7      - Bit number of user-specified I/O function code
;   R8      - Address of current entry in FDT
;   AP      - Address of first function-dependent argument (P1)
;
; OUTPUTS:
;
;--
SK_READ:
;+
; Use system routines to execute I/O preprocessing.
;-
    TSTL     P2(AP)                ; There must be bytes to receive.
    BEQL     BADPARAM              ; Bad input parameters.
    JMP      G^EXE$MODIFY          ; Lock down pages, set up IRP,
                                ; JUMP to EXE$QIODRVPKT, etc...

BADPARAM:
    MOVZWL   #SS$_BADPARAM,R0      ; Set bad parameter status
    JMP      G^EXE$ABORTIO         ; Abort the I/O with status in R0

    .SBTTL   SK_DIAGNOSE          - FDT preprocessing for special pass-through function
; ++
; SK_DIAGNOSE
;
; This routine performs FDT preprocessing including:
;
;   - Validating access to the descriptor buffer
;   - Validating access to, and locking, the read/write buffer
;   - Copying the SCSI command to a buffer in nonpaged pool
;
; INPUTS:
;
;   R0      - Address of FDT routine
;   R3      - IRP address
;   R4      - PCB address
;   R5      - UCB address
;   R6      - CCB address
;   R7      - Bit number of user-specified I/O function code
;   R8      - Address of current entry in FDT
;   AP      - Address of first function-dependent argument (P1)
;
; OUTPUTS:
;
;--

    DSC_OPCODE = 0
    DSC_FLAGS = 4
    DSC_CMDADR = 8
```

VMS Template SCSI Class Driver

```

DSC_CMDLEN = 12
DSC_DATADR = 16
DSC_DATLEN = 20
DSC_PADCNT = 24
DSC_PHSTMO = 28
DSC_DSCTMO = 32

SK_DIAGNOSE:
    .IF NOT_EQUAL ASSEMBLE_PASSTHRU ; Flag to control assembly of IO$_DIAGNOSE
    IFPRIV  DIAGNOSE,10$             ; Branch if process has DIAGNOSE priv
    MOVZWL  #SS$_NOPRIV,R0          ; Set no privilege status
    BRW     50$                     ; Branch to abort the I/O
;+
; First, check that we have read access to the user's descriptor.
;-
10$:  MOVQ   (AP),R0                 ; Get user descriptor address, length
      MOVL  R0,R9                   ; Save a copy of descriptor address
      CMPL  R1,#DIAG_BUF_LEN        ; Valid descriptor length
      BLSSW 40$                     ; Branch if not
      JSB   G^EXE$WRITECHK          ; Check for read access to the descriptor
                                           ; buffer (don't return if no access)

      CMPL  DSC_OPCODE(R9),#1       ; Valid opcode?
      BNEQW 40$                     ; Branch if not

      CMPL  DSC_DATLEN(R9),-        ; Reasonable read/write data buffer
           UCB$L_MAXBCNT(R5)        ; length?
      BGTRUW 40$                     ; Branch if not
      CMPL  DSC_PADCNT(R9),#511     ; Reasonable pad count?
      BGTRU  40$                     ; Branch if not

      MOVQ  DSC_CMDADR(R9),R0        ; Get SCSI command buffer address, length
      CMPL  R1,#MAX_CMD_LEN         ; Valid command length?
      BGTRU 40$                     ; Branch if not
      JSB   G^EXE$WRITECHK          ; Check for read access to the command
                                           ; buffer (don't return if no access)
      ADDL  #8,R1                   ; Reserve space for command buffer overhead
      JSB   G^EXE$ALONONPAGED       ; Allocate a buffer in which to copy
                                           ; the SCSI command
      BLBC  R0,50$                  ; Branch on error
      MOVL  R1,(R2)+                ; Save length of buffer
      MOVL  R2,IRP$L_MEDIA(R3)       ; Save the command buffer address
      MOVL  DSC_CMDLEN(R9),R0        ; Get length of the SCSI command
      MOVL  R0,(R2)+                ; Save it in the command buffer
      PUSHR #^M<R2,R3,R4,R5>        ; Save registers
      MOV3  R0,@DSC_CMDADR(R9),(R2) ; Copy the SCSI command from the user's
                                           ; buffer to the buffer in pool
      POPR  #^M<R2,R3,R4,R5>        ; Restore registers
      CLRL  IRP$L_BCNT(R3)           ; Assume no user read/write data
      MOVL  DSC_DATADR(R9),R0        ; Get address of user data buffer
      BEQL  30$                     ; Branch if no user read/write data
      MOVL  DSC_DATLEN(R9),R1        ; Get length of user data buffer
      BEQL  30$                     ; Branch if no user read/write data
      MOVAL G^EXE$READLOCKR,R2       ; Assume user is performing a read
      BLBS  DSC_FLAGS(R9),20$        ; Branch if this is a read operation
      MOVAL G^EXE$WRITE LOCKR,R2    ; Other check for read access
20$:  JSB   (R2)                     ; Check access to and lock down buffer
      BLBC  R0,60$                  ; Branch on error
30$:  MOVAL IRP$C_CDRP(R3),R0        ; Get address of SCDRP within IRP
      MOVL  DSC_FLAGS(R9),(R0)+      ; Save flags field in IRP/SCDRP

```


VMS Template SCSI Class Driver

```

        MOVAL   DSC_PADCNT(R9),R1          ; Get address of pad count field
        .REPT   3
        MOVL    (R1)+, (R0)+              ; Save pad count, timeout values
        .ENDR
        JMP     G^EXE$QIODRVPKT           ; Queue the packet to the driver
40$:   MOVZWL   #SS$_BADPARAM,R0          ; Set bad parameter status
50$:   JMP     G^EXE$ABORTIO             ; Abort the I/O with status in R0

;+
; We arrive here if the last FDT operation - checking access to and locking
; down the user's read/write buffer - fails. EXE$READLOCKR or EXE$WRITE LOCKR
; returns to us through a coroutine call to allow us to give up any resources
; which we have allocated during FDT processing. Deallocate the buffer
; containing a copy of the SCSI command, then return from the coroutine call.
; R0 and R1 must be preserved.
;-
60$:   PUSHQ   R0                        ; Save registers
        MOVL   IRP$_L_MEDIA(R3),R0      ; Get address of nonpaged pool buffer
                                           ; containing SCSI command
        MOVL   -(R0),R1                  ; Get length of buffer
        JSB    G^EXE$DEANONPGDSIZ      ; Deallocate the packet
        POPQ   R0                        ; Restore registers
        RSB    ; Return from coroutine call
        .ENDC                             ; IF ASSEMBLE_PASTHRU

        .IF EQUAL ASSEMBLE_PASSTHRU    ; IF IO$_DIAGNOSE not assembled, do this..
        MOVZBL #SS$_ILLIOFUNC,R0       ; Specify the error type
        JMP    G^EXE$ABORTIO           ; Abort the I/O with status in R0
        .ENDC

        .SBTTL +
        .SBTTL + STARTIO SCSI COMMAND EXECUTION ROUTINES
        .SBTTL +

        .SBTTL SK_STARTIO             - Driver STARTIO entry point

;++;
; SK_STARTIO
;
; This routine is the STARTIO entry point into the driver. Its main function
; is to dispatch to the function-code-specific routine that starts a specific
; I/O function.
;
; INPUTS:
;
;      R3      - IRP address
;      R5      - UCB address
;
; OUTPUTS:
;
;      R0      - 1st longword of I/O status: contains status code and
;                number of bytes transferred
;      R1      - 2nd longword of I/O status: low-order word contains high-order
;                word of number of bytes transferred
;      R4      - Destroyed
;      All other registers preserved
;--
SK_STARTIO:
        .ENABLE LSB                      ; SK_STARTIO
        INIT_UCB_STACK                   ; Initialize the internal stack in the UCB

```

VMS Template SCSI Class Driver

```

        MOVL    UCB$L_PDT(R5),R4      ; Get PDT address
        MOVL    R3,R2                 ; Copy IRP address
        MOVL    R5,R3                 ; Copy UCB address
        BSBW    ALLOC_SCDRP           ; Allocate an SCDRP
        MOVL    R2,SCDRP$L_IRP(R5)    ; Save IRP address in SCDRP

        EXTZV   #IRP$V_FCODE,-        ; Extract I/O function code
                #IRP$$_FCODE,-        ;
                IRP$W_FUNC(R2),R1     ;
        ASSUME  IRP$$_FCODE LE 7      ; Allow byte mode dispatch
        DISPATCH R1,TYPE=B,<-        ; Dispatch according to function
                <IO$_DIAGNOSE, IO_DIAGNOSE>,-
                <IO$_READPBLK, IO_READ>,-
                <IO$_AVAILABLE, IO_INQUIRY>>

;+
; Bogus I/O function code will fall through. Set illegal function code
; status and complete the I/O.
;-
IO_BOGUS:
        MOVZBL  #SS$_ILLIOFUNC,R0     ; Specify the error type
        BRB     COMPLETE_IO           ; Fall through to exit path for
;                                     ; if other error then uncomment.

COMPLETE_IO:
        ESBW    DEALLOC_SCDRP         ; Deallocate the SCDRP
        MOVL    R3,R5                 ; Copy UCB address
        REQCOM  ; Complete the I/O
        .DISABLE LSB                  ; SK_STARTIO

        .SBTTL  IO_INQUIRY            - Send SCSI INQUIRY command.

;+
; IO_INQUIRY
;
; This routine is intended as an example of how to write a STARTIO
; routine for a SCSI class driver.
;
; This routine sends an inquiry command to the target. If
; errors occur during the execution of this operation no retries
; occur. However, this class driver issues a REQUEST SENSE to
; determine the nature of the event. If the event is fatal, the
; error is logged and the I/O fails. If the event is
; benign, then the I/O completes with a REQCOM.
;
; IO_INQUIRY calls the port driver to allocate command buffer areas,
; maps the system or user buffer such that the port driver has access
; to these areas, and then calls the port driver's SEND_CMD entry point
; to send the SCSI command to a target.
; When the port driver returns from this call, the INQUIRY data has been
; moved, the command status is in the status-in buffer and the SCSI
; bus is free. The class driver checks the transfer count, releases
; its resources and completes the I/O with a call to REQCOM.
;
; INPUTS:
;
;       R3      - UCB address
;       R4      - PDT address

```

VMS Template SCSI Class Driver

```

;      R5      - SCDRP address
;
; OUPUTS:
;
;      R0      - Status
;
;              SSS_NORMAL - I/O completed successfully.
;              SSS_ILLSEQOP - I/O failed, bad sense key.
;              SSS_IVSTSFLG - Invalid SCSI status returned.
;              SSS_OPINCMPL - I/O failed, insufficient data returned.
;
;--
IO_INQUIRY:
    .ENABLE LSB ; IO_INQUIRY
    MOVAL INQUIRY_CMD,R2 ; Address of INQUIRY command
    BSBW SETUP_CMD ; Perform setup for SCSI command
    BLBC R0,35$ ;
    BSBW SEND_COMMAND ; Send the SCSI command
;+
; Determine by sending the INQUIRY command, what target is at this ID.
;
; After a call to the port driver, when the port status (R0) and SCSI
; command status have been checked, the class driver must verify that
; the number of bytes that were to be received or sent have been delivered
; by the port driver. SCDRP$L_TRANS_CNT contains the actual number of bytes
; of data transferred by the port driver.
;-
    BLBC R0,35$ ; Branch on error
    CMPL SCDRP$L_TRANS_CNT(R5),- ; Sufficient inquiry data returned?
        #INQ_DATA_LEN ;
    BLSSUW 34$ ; Branch if not
    MOVL SCDRP$L_SVA_USER(R5),R1 ; Get address of inquiry data
    CMPB #SCSI_C_DA,- ; Is this a SCSI disk device?
        SCSI_INQ_B_DEVTYPE(R1) ; Check INQUIRY data
;*** BNEQ SOMEWHERE ; If it's not the target you want.
30$: BSBW CLEANUP_CMD ; Clean up from the SCSI command
;+
; Now that the class driver knows what target is out there, determine if
; the target is ready by sending a TEST UNIT READY command.
;-
    MOVAL TEST_UNIT_READY_CMD,R2 ; Test Unit Ready command
    BSBW SETUP_CMD ; Perform setup for SCSI command
    BLBC R0,35$ ; Branch on error
    BSBW SEND_COMMAND ; Send the SCSI command
    BLBC R0,35$ ; Branch on error
    BSBW CLEANUP_CMD ; Clean up from the SCSI command
    CLRL R1 ; Clean up R1
    BRW COMPLETE_IO ; Complete the user's I/O.
;+
; Any error the class driver encounters is logged.
; R0 contains the VMS status.
;-
34$: MOVZWL #SS$_OPINCOMPL,R0
35$: LOG_ERROR - ; Log an invalid inquiry data error
        TYPE=CLS_DRV_ERROR_01,- ;
        VMS_STATUS=R0,- ; I/O operation failed
        UCB=R3,- ;
        MESSAGE=<ERROR DURING INQUIRY_TEST UNIT RDY SEQUENCE>
    BSBW CLEANUP_CMD ; Clean up from the SCSI command
    CLRL R1 ; Clean up R1
    BRW COMPLETE_IO ; Complete the user's I/O.
    .DISABLE LSB ; IO_INQUIRY

```

VMS Template SCSI Class Driver

```
.SBTTL IO_READ - Send SCSI INQUIRY command and return data.
;+
; IO_READ
;
; This routine is intended as an example of how to write a STARTIO
; routine that reads data from a target device and returns the data
; to a user buffer. Normally, some form of read command would be used
; to retrieve data from a target; however the format of read commands
; varies depending on the SCSI device class. Therefore, this
; example uses the INQUIRY command to get data from the target; the
; INQUIRY command is one of the few commands that is common among
; all device types.
;
; Third-party class drivers traditionally do NOT return the INQUIRY
; data to the application. Rather, the class driver uses this
; information to establish the characteristics of the SCSI target
; and the class driver's connection to this target.
;
; IO_READ calls the port driver to allocate command buffer areas,
; maps user read buffer such that the port driver has access to these
; areas and then calls the port driver's SEND_CMD entry point
; to send the SCSI command to a target. When the port driver returns from
; this call, the INQUIRY data has been moved to the user's buffer,
; the command status is in the status-in buffer and the SCSI bus is free.
; The class driver checks the transfer count, releases its resources and
; complete the I/O with a call to REQCOM.
;
; INPUTS:
;
;     R3      - UCB address
;     R4      - PDT address
;     R5      - SCDRP address
;
; OUTPUTS:
;
;     R0      - Status
;
;             SS$_NORMAL   - I/O completed successfully.
;             SS$_ILLSEQOP - I/O failed, bad sense key.
;             SS$_IVSTSFLG - Invalid SCSI status returned.
;             SS$_OPINCMPL - I/O failed, insufficient data returned.
;
;--
IO_READ:
    .ENABLE LSB                ; IO_READ

;+
; WARNING: If the user provides the wrong byte count the SCSI bus may hang.
; SCSI port drivers can recover from this error; however, the recovery mechanism
; may be severe and this I/O request will fail.
;-
    MOVL    #SCDRP$M_BUFFER_MAPPED,-; Set buffer mapped flag to prevent
    SCDRP$L_SCSE_FLAGS(R5)          ; allocation of S0 buffer for data
    MOVAL   QIO_INQUIRY_CMD,R2     ; Address of INQUIRY command for user data
```

VMS Template SCSI Class Driver

```

        BSBW     SETUP_CMD                ; Perform setup for SCSI command
        BLBC     R0,300$                  ; Setup failed
        SPI$MAP_BUFFER                ; Map the user buffer
        BSBW     SEND_COMMAND            ; Send the SCSI command
;+
; The port driver has been called to send the command and now returns
; with the data moved to the user's buffer, the port status in R0, and SCSI
; status in the STATUSIN buffer. The class driver checks the port driver
; and SCSI command status and then verifies that the number of bytes that were
; received equals the BCNT. SCDRP$L_TRANS_CNT contains the actual number
; of bytes of data transferred by the port driver.
;-
        BLBC     R0,35$                  ; Branch on error
        CMPL     SCDRP$L_TRANS_CNT(R5),- ; Sufficient inquiry data returned?
                SCDRP$L_BCNT(R5)
        BNEQUW   34$                      ; Branch if not
30$:  MOVL     SCDRP$L_TRANS_CNT(R5),R1; Return transaction count in IOSB
        BSBW     CLEANUP_CMD            ; Clean up from the SCSI command
        BRW      COMPLETE_IO            ; Complete the user's I/O
;+
; Errors the class driver encounters are logged.
; R0 contains the VMS status.
;-
34$:  MOVZWL   #SS$_OPINCOMPL,R0
35$:  LOG_ERROR -                        ; Log an invalid inquiry data error
        TYPE=CLS_DRV_ERROR_04,- ;
        VMS_STATUS=R0,-             ; I/O operation failed
        UCB=R3,-                     ;
        MESSAGE=<ERROR DURING READ QIO FUNCTION>
        BSBW     CLEANUP_CMD            ; Clean up from the SCSI command
        CLRL     R1                     ; Clean up R1
        BRW      COMPLETE_IO            ; Complete the user's I/O
;+
; The template driver does not support segmented I/O. This exercise
; is left to the user.
;-
300$: BICL     #SCDRP$L_BUFFER_MAPPED,-; No buffer mapped, so don't unmap.
        SCDRP$L_SCSI_FLAGS(R5) ;
        LOG_ERROR -                    ; Log an invalid inquiry data error
        TYPE=CLS_DRV_ERROR_05,- ;
        VMS_STATUS=R0,-             ; I/O operation failed.
        UCB=R3,-                     ;
        MESSAGE=<ERROR I_O OPERATION NOT PROPERLY SEGMENTED>
        CLRL     R1                     ; Clean up R1
        BRW      COMPLETE_IO            ; Complete the user's I/O
        .DISABLE LSB                    ; IO_READ

        .SBTTL IO_DIAGNOSE             - Special pass-through function
;+
; IO_DIAGNOSE
;
; STARTIO routine for the passthru function of the template SCSI
; class driver. This routine assumes that the user has provided
; a buffer that contains the SCSI command packet and that the
; FDT routines in the driver have made the appropriate checks
; during I/O preprocessing to allow access to the user data areas
; during STARTIO.

```

VMS Template SCSI Class Driver

```

;
; IO_DIAGNOSE makes calls into the port driver to allocate command
; buffer areas, maps the user buffer such that the port driver
; can access user areas, and then calls the port driver's SEND_CMD
; entry point to send the SCSI command to a target. When the port driver
; returns from this call, the user's data has been moved, the
; command status is in the status-in buffer and the SCSI bus
; is free. The class driver releases its resources and
; completes the I/O with a call to REQCOM.
;
; INPUTS:
;
;      R2      - IRP address
;      R3      - UCB address
;      R4      - PDT address
;      R5      - SCDRP address
;
; OUTPUTS:
;
;      R0      - Status
;      R1,R2   - Destroyed
;      All other registers preserved
;--
IO_DIAGNOSE:
    .ENABLE LSB ; IO_DIAGNOSE
    .IF NOT_EQUAL ASSEMBLE_PASSTHRU ; IF assemble IO$_DIAGNOSE if ASSM_PASS.
    MOVL IRP$L_MEDIA (R2),- ; Copy command buffer from IRP to
        SCDRP$L_MEDIA (R5) ; SCDRP
    MOVL IRP$L_SVAPTE (R2),- ; and SVAPTE,
        SCDRP$L_SVAPTE (R5) ;
    MOVL IRP$L_BCNT (R2),- ; BCNT,
        SCDRP$L_BCNT (R5) ;
    MOVW IRP$W_BOFF (R2),- ; and BOFF
        SCDRP$W_BOFF (R5) ;
    MOVW IRP$W_STS (R2),- ; and STS
        SCDRP$W_STS (R5) ;
    MOVAL IRP$C_CDRP (R2),R0 ; Get address of SCDRP portion of IRP
    EXTZV #1,#1,(R0),R1 ; Get disconnect flag
    INSV R1,#UCB_V_DISCONNECT,- ; Fill in disconnect flag in UCB
        #1,UCB_L_SK_FLAGS (R3) ;
    EXTZV #2,#1,(R0),R1 ; Get synchronous flag
    INSV R1,#UCB_V_SYNCHRONOUS,- ; Fill in synchronous flag in UCB
        #1,UCB_L_SK_FLAGS (R3) ;
    ADDL #4,R0 ; Advance to pad count field
    MOVL (R0)+,- ; Fill in the pad count in the SCDRP
        SCDRP$L_PAD_BCNT (R5) ;
    MOVL (R0)+,- ; Fill in the phase change (DMA) timeout
        SCDRP$L_DMA_TIMEOUT (R5) ; in the SCDRP
    MOVL (R0)+,- ; Fill in the disconnect timeout in the
        SCDRP$L_DISCON_TIMEOUT (R5) ; SCDRP
    BSEW SET_CONN_CHAR ; Set up the connect characteristics
    MOVL SCDRP$L_MEDIA (R5),R1 ; Get address of SCSI command in pool
    MOVL (R1)+,R1 ; Get length of SCSI command
    ADDL #8,R1 ; Account for overhead
    SPI$ALLOCATE_COMMAND_BUFFER ; Allocate a command buffer
    MOVL R2,SCDRP$L_CMD_BUF (R5) ; Save address of command buffer
    CLRL (R2)+ ; Reserve a longword for status
    MOVB #^XFF,-1 (R2) ; Initialize status field

```

VMS Template SCSI Class Driver

```

        MOVAL    -1(R2),-                ; Address to save status byte
                SCDRP$L_STS_PTR(R5)      ;
        MOVL     R2,SCDRP$L_CMD_PTR(R5)  ; Address of SCSI command in cmd buffer
        MOVL     SCDRP$L_MEDIA(R5),R0    ; Get SCSI command in pool again
        MOVL     (R0),(R2)+              ; Copy SCSI command length
        PUSHR    #^M<R0,R2,R3,R4,R5>    ; Save registers
        MOVCS    (R0),4(R0),(R2)        ; Copy SCSI command to command buffer
        POPR     #^M<R0,R2,R3,R4,R5>    ; Restore registers
        MOVL     -(R0),R1                ; Get length of command buffer in pool
        JSB      G^EXE$DEANONPGDSIZ     ; Deallocate the buffer
        TSTL     SCDRP$L_BCNT(R5)       ; Any user data buffer?
        BEQL     10$                    ; Branch if not
        SPI$MAP_BUFFER                   ; Map the user's data buffer
10$:    SPI$SEND_COMMAND                 ; Send the SCSI command
        PUSHL    R0                     ; Save returned port status
        TSTL     SCDRP$L_BCNT(R5)       ; User buffer mapped?
        BEQL     20$                    ; Branch if not
        SPI$UNMAP_BUFFER                 ; Unmap the user's data buffer
20$:    MOVL     SCDRP$L_CMD_BUF(R5),R0  ; Get the command buffer address
        PUSHL    (R0)                   ; Save the SCSI status byte
        SPI$DEALLOCATE_COMMAND_BUFFER   ; Deallocate the command buffer
        POPL     R1                     ; Restore the SCSI status byte
        POPL     R0                     ; Restore the port status
        INSV     SCDRP$L_TRANS_CNT(R5),- ; Copy the transfer count to the
                #16,#16,R0             ; high-order word of R0
        .ENDC                             ; If ASS_DIAG FALSE don't assemble
        BRW      COMPLETE_IO            ; Complete the QIO
        .DISABLE LSB                     ; IO_DIAGNOSE

        .SBTTL +
        .SBTTL + UTILITY ROUTINES
        .SBTTL +

        .SBTTL SEND_COMMAND             - Send a SCSI command
; ++
; SEND_COMMAND
;
; This routines sends a command to the SCSI device. It returns any failing
; port status to the caller. If the port status is success, it checks the
; SCSI status byte. If a check condition status is returned, a request
; sense command is sent to the target and the sense key is translated into a
; VMS status code, which is returned as status.
;
; INPUTS:
;
;     R3      - UCB address
;     R4      - PDT address
;     R5      - SCDRP address
;
; OUTPUTS:
;
;     R0      - Status
;
;             SS$_IVSTSFLG - Invalid SCSI status returned.
;             SS$_ILLSEQOP - I/O operation failed.
;     R1,R2   - Destroyed
;     All other registers preserved
; --
SEND_COMMAND:

```

VMS Template SCSI Class Driver

```

        .ENABLE LSB                ; SEND_COMMAND
        SUBSAVE                    ; Save return address
        SPI$SEND_COMMAND           ; Send the SCSI command
        BLBC R0,10$                ; If port failed, return
        MOVZBL @SCDRP$L_STS_PTR(R5),R1 ; Get SCSI status byte
        BICB #SCSI$M_STS,R1       ; Clear reserved, vendor-unique bits
        BNEQ 20$                  ; Branch if bad status
10$:    SUBRETURN                  ; Return to caller

;+
; A bad SCSI status code was returned. If the code is a check condition, then
; send a request sense command to the device. Otherwise, the status code is
; something unexpected. Log an error and return SS$_MEDOFL status.
;-
20$:    CMPB R1,#2                ; Check condition status?
        BNEQ 90$                  ; Branch if not

;+
; A check condition status code was returned. Save the original SCDRP address,
; allocate a second one and send a request sense command. If the request
; sense succeeds, translate the sense key to a VMS status code and return that
; as the status code for the original command.
;-
45$:    MOVL R5,UCB_L_SCDRP_SAV1(R3) ; Save original SCDRP address
        BSBW ALLOC_SCDRP          ; Allocate an additional SCDRP
        BSBW REQUEST_SENSE       ; Send a request sense command
        BLBC R0,50$              ; Branch on error
                                        ; a VMS status code in R0

;+
; Look at the results of the request sense to determine the exact nature
; of the event.
;-
        MOVL SCDRP$L_SVA_USER(R5),R1 ; Get address of REQUEST SENSE DATA.
        BICB3 #^XF0,SCSI_XS_B_ERR_CODE(R1),- ; First check ERROR CODE.
        R0                                ; In this case zero is good, but this
        BNEQ 50$                          ; is really device specific.
        BICB3 #^XF0,SCSI_XS_B_KEY(R1),- ; Mask off SENSE KEY.
        R0

;+
; Depending on the value of the sense key, dispatch to the appropriate
; error recovery.
;-
        DISPATCH R0,TYPE=B,<-                ; Dispatch according to SENSE KEY.
        <SCSI_C_NO_SENSE,SK_OK>,-           ; No sense data
        <SCSI_C_RECOVERED_ERROR,SK_OK>,- ; Recovered error
        <SCSI_C_NOT_READY,SK_BAD>,-        ; Device not ready
        <SCSI_C_MEDIUM_ERROR,SK_BAD>,-    ; Medium (parity) error
        <SCSI_C_HARDWARE_ERROR,SK_BAD>,-  ; Hardware error
        <SCSI_C_ILLEGAL_REQUEST,SK_BAD>,- ; Illegal request
        <SCSI_C_UNIT_ATTENTION,SK_BAD>,- ; Unit attention ( reset... )
        <SCSI_C_DATA_PROTECT,SK_BAD>,-    ; Data protection (write lock)
        <SCSI_C_BLANK_CHECK,SK_BAD>,-     ; Blank check
        <SCSI_C_VENDOR_UNIQUE,SK_BAD>,-   ; Vendor unique key
        <SCSI_C_COPY_ABORTED,SK_BAD>,-    ; Copy operation aborted
        <SCSI_C_ABORTED_COMMAND,SK_BAD>,- ; Command aborted
        <SCSI_C_EQUAL,SK_BAD>,-          ; Data match
        <SCSI_C_VOLUME_OVERFLOW,SK_BAD>,- ; Write past physical end
        <SCSI_C_MISCOMPARE,SK_BAD>>      ; Data mismatch

;+
; Either the sense key was bad or the key was invalid. In either case

```


VMS Template SCSI Class Driver

```
; indicate that the command failed. Some class drivers will want
; to translate each bad sense key to a unique class driver SS$_XXXXX
; status code. Here we will always return SS$_ILLSEQOP.
;-
SK_BAD:
    MOVL    #SS$_ILLSEQOP,R0        ; I/O operation failed
    BRB     50$                     ; cleanup and return error
;+
; If the sense key indicated that the operation completed successfully,
; then return success.
;-
SK_OK:
    MOVL    #SS$_NORMAL,R0         ; I/O operation succeeded
    BRB     50$                     ; Clean up and return error
50$:
    BSBW    CLEANUP_CMD             ; Clean up the request sense command
    BSEW    DEALLOC_SCDRP          ; Deallocate the request sense SCDRP
    MOVL    UCB_L_SCDRP_SAV1(R3),R5 ; Restore original SCDRP address
    MOVL    R5,UCB_L_SCDRP(R3)     ; Copy it to the UCB
    BRW     10$                     ; Return to caller
;+
; If the status returned for the last command was anything other than
; check condition, log an error and return a status of SS$_IVSTSFLG to
; indicate that command failed and that there is no request sense data.
;-
90$:
    MOVL    #SS$_IVSTSFLG,R0       ; Return a generic status code
    LOG_ERROR -                     ; Log a send command error
        TYPE=CLS_DRV_ERROR_02,-   ; Generic user class driver error
        VMS_STATUS=R0,-           ;
        UCB=R3,-                  ;
        MESSAGE=<ERROR BAD SCSI COMMAND STATUS>
    BRW     10$
    .DISABLE LSB                     ; SEND_COMMAND

    .SBTTL REQUEST_SENSE - Send a request sense command
;+
; REQUEST_SENSE
;
; This routine is called by SEND_COMMAND when a command fails with check
; condition status. A request sense command is sent to the target.
;
; INPUTS:
;
;     R3     - UCB address
;     R4     - PDT address
;     R5     - SCDRP address
;
; OUTPUTS:
;
;     R0     - Status
;                SS$_IVSTSFLG - Bad SCSI status returned during
;                REQUEST SENSE.
;
;     R1,R2  - Destroyed
;
;     All other registers preserved
;--
REQUEST_SENSE:
    .ENABLE LSB                       ; REQUEST_SENSE
```

VMS Template SCSI Class Driver

```

        SUBSAVE                ; Save return address
        MOVAL  REQUEST_SENSE_CMD,R2 ; Address of REQUEST_SENSE command
        BSBW  SETUP_CMD        ; Perform setup for SCSI command
        BLBC  R0,10$           ; Branch on error
        SPI$SEND_COMMAND       ; Send the SCSI command
        BLBC  R0,10$           ; Return on error
        MOVZBL @SCDRP$L_STS_PTR(R5),R1 ; Get SCSI status byte
        BICB  #SCSI$M_STS,R1   ; Clear reserved, vendor unique bits
        BNEQ  20$              ; Branch if bad status
10$:    SUBRETURN              ; Return to caller

20$:    MOVZWL #SS$_IVSTSFLG,R0 ; Return bad SCSI status to caller.
        BRB   10$
        .DISABLE LSB          ; REQUEST_SENSE

        .SBTTL  SET_CONN_CHAR - Modify connection characteristics
; ++
; SET_CONN_CHAR
;
; This routine is called to initialize the connection characteristics, which
; specify such things as whether the device supports disconnect and
; synchronous operation, and the bus busy, arbitration, selection, and
; command retry counters.
;
; This routine first does a SPI$GET_CONNECTION_CHAR to get the current
; values of the connection characteristics, modifies the values of interest,
; then does a SPI$SET_CONNECTION_CHAR to set up the new values. This allows
; the class driver to change a subset of the characteristics and leave the
; rest unmodified.
;
; INPUTS:
;
;     R3      - UCB address
;     R4      - SPDT address
;     R5      - SCDRP address
;
; OUTPUTS:
;
;     R0-R2   - Destroyed
;     All other registers preserved
; --
SET_CONN_CHAR:
        .ENABLE LSB          ; SET_CONN_CHAR

        SUBSAVE                ; Save return address
        MOVL  #<<NUM_ARGS+1>*4>,R1 ; Size of get/set connection char buffer
        BSBW  ALLOC_POOL       ; Allocate the buffer
        SUBPUSH R2             ; Save address of buffer
        MOVL  #NUM_ARGS,(R2)   ; Set argument count in buffer
        SPI$GET_CONNECTION_CHAR ; Get current connection characteristics
        BLBC  R0,10$           ; Branch on error
; +
; Some devices won't select if selected with attention.
;
; NOTE: It is strongly suggested that targets and devices
; support the disconnect/reselection sequence. All
; Digital-supplied devices support this feature to
; ensure consistent bus performance.
;

```

VMS Template SCSI Class Driver

```

;      EXTZV  #UCB_V_DISCONNECT,#1,- ; Fill in disconnect flag
;      UCB_L_SK_FLAGS(R3),4(R2);
;-
      EXTZV  #UCB_V_SYNCHRONOUS,#1,- ; Fill in synchronous flag
      UCB_L_SK_FLAGS(R3),8(R2);
10$:  SPI$SET_CONNECTION_CHAR      ; Set the connection characteristics
      PUSHL  R0                    ; Save return status
      SUBPOP R0                    ; Get address of characteristics buffer
      BSBW  DEALLOC_POOL          ; Deallocate the buffer
      POPL  R0                    ; Restore return status
      BLBS  R0,20$                ; Branch if success status
      MOVL  #SS$_CTRLERR,R0       ; Otherwise, return a reasonable status
20$:  SUBRETURN                    ; Return to caller
      .DISABLE LSB                ; SET_CONN_CHAR

      .SBTTL  SK_WAIT              - Stall for the specified number of seconds
; ++
; SK_WAIT
;
; This routine is used by the SK_WAIT macro to stall a thread for a specified
; number of seconds. It sets the timeout bit in the UCB and relies on the
; device timeout mechanism to resume the stalled thread.
;
; INPUTS:
;
;      IPL      - 31
;      R5       - UCB address
;      (SP)     - Return address
;      4(SP)    - Wait time in seconds
;      8(SP)    - Saved IPL
;      12(SP)   - Address of caller's caller
;
; OUPUTS:
;
;      Stack    - Return address, wait time, IPL removed
;      Control  returns to caller's caller
;      All registers preserved
;
;      NOTE: The use of the SK_WAIT macro destroys R0-R3
; --
SK_WAIT:
      MOVQ   R3,UCB$L_FR3(R5)      ; Save R3 and R4 in fork block
      ADDL3  #2,(SP)+,UCB$L_FPC(R5) ; Save return address in fork block
      BISW   #UCB$M_TIM,UCB$W_STS(R5); Set timer expected bit
      ADDL3  (SP)+,G^EXE$GL_ABSTIM,- ; Set up timeout time in UCB
      UCB$L_DUETIM(R5)            ;
      BICW   #UCB$M_TIMEOUT,-     ; Clear timer expired bit
      UCB$W_STS(R5)              ;
      ENBINT                    ; Reenable interrupts
      RSB                        ; Return to caller's caller

      .SBTTL  ALLOC_SCDRP          - Allocate an SCDRP
; ++
; ALLOC_SCDRP
;
; This routine allocates an SCDRP by attempting to remove one from the queue
; in the UCB. If the queue is empty (which should never happen), then bugcheck.
; The entire SCDRP is zeroed and various fields are initialized.

```

VMS Template SCSI Class Driver

```

;
; INPUTS:
;
;     R3         - UCB address
;
;     UCB_L_SCDRPQ_FL - Queue of SCDRPs
;
; OUTPUTS:
;
;     R5         - SCDRP address
;     All other registers preserved
;
;     SCDRP$L_UCB      - UCB address
;     SCDRP$L_IRP     - IRP address
;     SCDRP$L_CDT     - SCDT address
;     SCDRP$L_SCSI_FLAGS - Initialized
;     SCDRP$L_CL_SSK_PTR - Initialized
;--

ALLOC_SCDRP:
    .ENABLE LSB                ; ALLOC_SCDRP
    REMQUE @UCB_L_SCDRPQ_FL(R3),R5 ; Remove an SCDRP from the queue
    PUSHR #^M<R0,R1,R2,R3,R4,R5> ; Save registers
    MOVC5 #0,..,#0,-          ; Initialize the SCDRP
                                #SCDRP$C_LENGTH-12,-
                                12(R5) ;
    POPR #^M<R0,R1,R2,R3,R4,R5> ; Restore registers
    MOVL R5,UCB_L_SCDRP(R3)    ; Save SCDRP address in UCB
    MOVL R3,SCDRP$L_UCB(R5)    ; Save UCB address in SCDRP
    MOVB UCB$B_FLCK(R3),-      ; Copy the fork lock field from the
                                SCDRP$B_FLCK(R5) ; UCB to the SCDRP
    MOVL UCB_L_SCDT(R3),-      ; Save SCDT address in SCDRP
                                SCDRP$L_CDT(R5) ;
    MOVAL SCDRP$L_SCSI_STK-4(R5),- ; Initialize the SCDRP stack pointer
                                SCDRP$L_SCSI_STK_PTR(R5);
    RSB

    .DISABLE LSB                ; ALLOC_SCDRP

    .SBTTL DEALLOC_SCDRP - Deallocate an SCDRP
;++
; DEALLOC_SCDRP
;
; This routine deallocates an SCDRP by returning it to the queue in the
; UCB. A sanity check is made to ensure that any map registers for this
; command have been deallocated.
;
; INPUTS:
;
;     R3         - UCB address
;     R5         - SCDRP address
;
; OUTPUTS:
;
;     R3         - UCB address
;     R5         - UCB address (for _R5 entry point)
;     All other registers preserved
;
;     UCB_L_SCDRP - Cleared to indicate no active SCDRP
;--

DEALLOC_SCDRP:

```

VMS Template SCSI Class Driver

```

        .ENABLE LSB                ; DEALLOC_SCDRP
INSQUE  SCDRP$L_FQFL(R5),-        ; Insert SCDRP in UCB queue
        UCB_L_SCDRPQ_FL(R3)      ;
CLRL    UCB_L_SCDRP(R3)          ; No active SCDRP for this UCB
RSB

        .DISABLE LSB              ; DEALLOC_SCDRP

        .SBTTL  ALLOC_POOL        - Allocate a block of nonpaged pool
; ++
; ALLOC_POOL
;
; This routine allocates a block of nonpaged pool no smaller than the
; size of a fork block (allowing COM$DRVDEALMEM to fork on this block
; during deallocation.) An extra quadword at the top of the block is reserved
; to save the size field, relieving the caller of this responsibility.
; The caller is presented with the address just beyond the reserved quadword.
; Although a word would be sufficient for this field, a quadword is used for
; alignment purposes (some blocks are used as IRPs, which are placed on
; self-relative queues and require quadword alignment.)
;
; If an allocation failure occurs, the thread is stalled and wakes up once a
; second to retry the allocation.
;
; INPUTS:
;
;      R1      - Size of block to allocate
;      R3      - UCB address
;
; OUTPUTS:
;
;      R0      - Destroyed
;      R1      - Size of block allocated
;      R2      - Address of allocated block
;      -8(R2)  - Length of allocated block (used by DEALLOC_POOL)
;      All other registers preserved
; --
ALLOC_POOL:
        .ENABLE LSB                ; ALLOC_POOL
ADDL    #8,R1                      ; Reserve a quadword to save size
CMLPL   R1,#FKB$C_LENGTH           ; Requested size smaller than fork block?
BGEQ    10$                        ; Branch if not
MOVL    #FKB$C_LENGTH,R1          ; Use fork block size as minimum
10$:    PUSHL   R1                  ; Save allocation length
        PUSHL   R3                  ; Save UCB address
        JSB    G^EXE$ALONONPAGED   ; Allocate a block
        POPL   R3                  ; Restore register
        BLBC  R0,20$               ; Branch if error
        ADDL  #4,SP                 ; Remove allocation length from stack
        PUSHR  #^M<R0,R1,R2,R3,R4,R5> ; Save registers
        MOVCS  #0,.,#0,R1,(R2)     ; Initialize the packet
        POPR   #^M<R0,R1,R2,R3,R4,R5> ; Restore registers
        MOVL   R1,(R2)+            ; Save size of block
        ADDL   #4,R2                ; Skip a longword
        RSB                          ; Return to caller

```

VMS Template SCSI Class Driver

```

;+
; A pool allocation failure occurred. Come back once a second and retry the
; operation until successful.
;-
20$:   SUBPUSH (SP)+           ; Save allocation length (PUSHL R1 above)
       SUBSAVE              ; Save return address
       SK_WAIT #1,UCB=R3    ; Wait a second
       SUBPOP -(SP)         ; Restore return address
       SUBPOP R1            ; Restore allocation length
       BRW 10$              ; Try again
       .DISABLE LSB        ; ALLOC_POOL

       .SBTTL DEALLOC_POOL  - Deallocate a block of nonpaged pool
;++;
; DEALLOC_POOL
;
; This routine deallocates a block of nonpaged pool. The size of the block
; is stored in the reserved quadword at a negative offset from the beginning
; of the block.
;
; INPUTS:
;
;   R0      - Address of block to deallocate
;   -8(R0)  - Length of block to deallocate
;
; OUTPUTS:
;
;   R0      - Destroyed
;   All other registers preserved
;--
DEALLOC_POOL:
       .ENABLE LSB         ; DEALLOC_POOL
       PUSHQ R1            ; Save R1,R2
       SUBL #4,R0          ; Skip a longword
       MOVL -(R0),IRP$W_SIZE(R0) ; Copy size field
       CLRB IRP$B_TYPE(R0) ; Clear type field (prevents block from
                           ; being interpreted as shared memory
                           ; during deallocation)
       JSB G^EXE$DEANONPAGED ; Deallocate the block
       POPQ R1             ; Restore R1,R2
       RSB
       .DISABLE LSB       ; DEALLOC_POOL

       .SBTTL SETUP_CMD   - Common setup for all SCSI commands
;++;
; SETUP_CMD
;
; This routine performs common setup prior to the sending of a SCSI command.
; Setup includes allocating a command buffer, filling in the pointers in the
; SCDRP to the command and status fields, copying the SCSI command to the
; command buffer, allocating an S0 "user" buffer if the command requires
; transferring data to or from the class driver, filling in the SCDRP fields
; used to map this buffer, and mapping the buffer.
;
; Since this routine calls SPI$ALLOCATE_COMMAND_BUFFER, which can suspend
; the thread, the return PC must be saved in the SCDRP.
;

```

VMS Template SCSI Class Driver

```

; INPUTS:
;
;       R2      - Pointer to entry in SCSI_CMD table
;       R4      - PDT address
;       R5      - SCDRP address
;
; OUTPUTS:
;
;       R0      - Status
;       R1,R2   - Destroyed
;
;       SCDRP$L_CMD_BUF - Address of SCSI command buffer
;       SCDRP$L_CMD_PTR - Address of SCSI command
;       SCDRP$L_STS_PTR - Address to save SCSI status byte
;       SCDRP$L_SVA_USER- Address of S0 "user" buffer
;       SCDRP$L_BCNT   - Length of S0 "user" buffer
;       SCDRP$W_BOFF   - Byte offset of S0 "user" buffer
;       SCDRP$L_SVAPTE - SVAPTE of S0 "user" buffer
;       IRP$V_FUNC     - SET/CLEAR to indicate READ/WRITE from S0 "user" buffer
;       SCDRP$L_DMA_TIMEOUT - Time in seconds for a DMA timeout.
;       SCDRP$L_DISCON_TIMEOUT - Time in seconds for a disconnect to time out.
;
;--
        .ENABLE LSB                      ; SETUP_CMD
SETUP_CMD:
SCSI_CMD_BUF_OVHD = 4 + 4                ; 4 bytes to save status byte +
                                          ; 4 bytes for SCSI command length
        SUBSAVE                          ; Save return address
        MOVZBL (R2),R1                    ; Get size SCSI command
        ADDL   #SCSI_CMD_BUF_OVHD,R1     ; Add in command buffer overhead
        SUBPUSH R2                        ; Save R2
        SPI$ALLOCATE_COMMAND_BUFFER      ; Allocate a command buffer
        MOVL   R2,R1                      ; Copy command buffer address
        SUBPOP R2                         ; Restore R2
        MOVB   #^XFF,(R1)                ; Initialize status field
        MOVAL  (R1)+,-                    ; Address to put SCSI status byte
                                          SCDRP$L_STS_PTR(R5) ;
        MOVL   R1,SCDRP$L_CMD_PTR(R5)    ; Save address of SCSI command
        MOVZBL (R2)+,R0                    ; Get SCSI command length
        MOVL   R0,(R1)+                    ; Save length in command buffer
        ASHL   #-1,R0,R0                  ; Change byte count to word count
10$:      MOVW   (R2)+,(R1)+                ; Copy a byte of SCSI command
        SOBGTR R0,10$                    ; Repeat for entire SCSI command
;+
; There is a dependency here that the format of the SCSI_COMMAND record
; does not change.
; Copy the per command timeout values from the SCSI_CMD block to the
; SCSI Class Driver Request Packet.
;
; R2 points at the direction field in the SCSI_CMD block.
;--
        MOVL   3(R2),-                    ; Time in seconds for a DMA timeout.
                SCDRP$L_DMA_TIMEOUT(R5)
        MOVL   7(R2),-                    ; Disconnect timeout in seconds.
                SCDRP$L_DISCON_TIMEOUT(R5)
;+
; Determine if a buffer has already been mapped. If no buffer has been mapped,
; then allocate a system buffer and map it to receive the data from the
; target device.

```

VMS Template SCSI Class Driver

```

;-
        BBC      #SCDRP$V_BUFFER_MAPPED,-; If buffer is mapped then do special
                SCDRP$L SCSI_FLAGS(R5),-; setup for this command.
                20$
;+
; During the STARTIO operation in the class driver, the user's QIO parameters
; must be copied from the IRP to SCDRP (SCSI Class Driver Request Packet).
; The user data is then mapped, the SCSI CMD packet is allocated, and the
; command is sent to a target, over the connection established during UNIT INIT.
;-
        MOVL     UCB$L_IRP(R3),R2          ; Get current I/O's IRP address
        CLRL     SCDRP$L_ABCNT(R5)        ; Initialize accumulated byte count
        MOVW     IRP$W_FUNC(R2),-        ; Copy function code and modifiers,
                SCDRP$W_FUNC(R5)        ; MEDIA, SVAPTE, and BOFF fields,
        MOVW     IRP$W_STS(R2),-        ; and STS
                SCDRP$W_STS(R5)        ;
        MOVL     IRP$L_MEDIA(R2),-        ; from the IRP to the SCDRP
                SCDRP$L_MEDIA(R5)        ;
        MOVL     IRP$L_SVAPTE(R2),-      ;
                SCDRP$L_SVAPTE(R5)        ;
        MOVW     IRP$W_BOFF(R2),-        ;
                SCDRP$W_BOFF(R5)        ;
        MOVL     IRP$L_BCNT(R2),-        ; Copy user's BCNT from IRP to SCDRP.
                SCDRP$L_BCNT(R5)        ;
        Cmpl     SCDRP$L_BCNT(R5),-      ; Transfer length greater than maximum
                UCB$L_MAXBCNT(R3)        ; supported?
        BGTR     300$                    ; GTR, therefore I/O must be segmented
        CLRL     SCDRP$L_PAD_BCNT(R5)    ; No padding of last page required
        ADDL3    #<4+4>,-                ; Address of transfer length field in
                SCDRP$L_CMD_PTR(R5),R1   ; SCSI command
        MOVW     SCDRP$L_BCNT(R5),(R1)   ; Copy user-supplied byte count to command.
        BRW      50$                    ; Setup finished.

20$:     CVTWL   (R2),R1                 ; Get length of send data buffer
        BLSS    50$                    ; Branch if negative, no system buffer
                                                ; involved, leave SCDRP$L_BCNT unchanged
        BEQL    30$                    ; Branch if zero length, zero SCDRP$L_BCNT
        SUBPUSH R2                      ; Save R2
        BSBW    ALLOC_POOL              ; Allocate a buffer to receive response
        MOVL    R2,R1                  ; Copy buffer address
        SUBPOP  R2                      ; Restore R2
        MOVL    R1,SCDRP$L_SVA_USER(R5) ; Save address of allocated buffer
        MOVZWL  (R2)+,SCDRP$L_BCNT(R5) ; Save length of transfer
        CLRL    SCDRP$L_PAD_BCNT(R5)    ; No padding required
        BICW3   #^C^X1FF>,R1,-        ; And byte offset within page
                SCDRP$W_BOFF(R5)        ;
        INSV    (R2),#IRP$V_FUNC,#1,-  ; Set/clear FUNC bit to indicate READ/
                SCDRP$W_STS(R5)        ; WRITE function
        PUSHL   R3                      ; Save R3
        MOVL    SCDRP$L_SVA_USER(R5),R2 ; Get user buffer address
        JSB     G^MMG$SVAPTECHK        ; Get SVAPTE of allocated system buffer
        MOVL    R3,SCDRP$L_SVAPTE(R5)  ; Save SVAPTE in SCDRP
        POPL   R3                      ; Restore R3
        BISB    #SCDRP$M_S0BUF!-      ; This buffer is an S0 "user" buffer
                SCDRP$M_BUFFER_MAPPED,-; and it has been mapped
                SCDRP$L SCSI_FLAGS(R5) ;
        SPI$MAP_BUFFER                  ; Map the "user" buffer for read access

50$:     MOVZWL  #SS$ _NORMAL,R0        ; Set success status
52$:     SUBRETURN

30$:     CLRL    SCDRP$L_BCNT(R5)        ; No data being transferred
        BRB     50$                    ; Use common exit

```


VMS Template SCSI Class Driver

```

300$: MOVZWL #SS$_IVBUFLN,R0      ; Bad byte count
      BRB   52$
      .DISABLE      LSB          ; SETUP_CMD

      .SBTTL  CLEANUP_CMD      - Common cleanup for all SCSI commands
; ++
; CLEANUP_CMD
;
; This routine performs common cleanup after the sending of a SCSI command,
; including unmapping the user buffer and deallocating the command buffer.
;
; INPUTS:
;
;      R4      - PDT address
;      R5      - SCDRP address
;
; OUTPUTS:
;
;      R2      - Destroyed
;      All other registers preserved
; --

CLEANUP_CMD:
      .ENABLE LSB                ; CLEANUP_CMD
      PUSHR  #^M<R0,R1,R3>      ; Save registers
      BBCC  #SCDRP$V_BUFFER_MAPPED,-; Branch if no buffer has been mapped
           SCDRP$L_SCSI_FLAGS(R5),-;
           10$
      SPI$UNMAP_BUFFER          ; Unmap the mapped buffer
10$:  BBCC  #SCDRP$V_S0BUF,-      ; Branch if this is not an S0 "user"
           SCDRP$L_SCSI_FLAGS(R5),-; buffer
           20$
      MOVL  SCDRP$L_SVA_USER(R5),R0 ; Get address of S0 user buffer
      CLRL  SCDRP$L_SVA_USER(R5)   ; Buffer no longer owned
      BSBW  DEALLOC_POOL          ; Deallocate the buffer
20$:  MOVL  SCDRP$L_CMD_BUF(R5),R0 ; Get address of command buffer
      SPI$DEALLOCATE_COMMAND_BUFFER ; Deallocate the command buffer
30$:  POPR  #^M<R0,R1,R3>        ; Restore registers
      RSB
      .DISABLE LSB              ; CLEANUP_CMD

      .SBTTL  LOG_ERROR        - Write an entry to the error log file
; ++
; LOG_ERROR
;
; This routine writes an entry to the error log file. If the device is offline,
; no error is logged. This prevents the error log file from being filled up while
; the class driver does its periodic polling of devices that have been set
; offline. The assumption is that the initial error that caused the device to
; be placed offline has been logged and that subsequent error log entries would
; be redundant.
;
; INPUTS:
;

```

VMS Template SCSI Class Driver

```
;      R5      - UCB address
;      R7      - Error type
;      R8      - VMS status code
;
; OUTPUTS:
;
;      All registers preserved
;--
LOG_ERROR:
    .ENABLE LSB                      ; LOG_ERROR
    PUSHR   #^M<R0,R2,R9,R10>       ; Save registers
    MOVB    UCB$B_DEVTYPE(R5),R9     ; Save SCSI device type
10$:      MOVB    UCB$B_DEVCLASS(R5),R10 ; Save DEVCLASS field
    MOVL    UCB$L_DDT(R5),R0         ; Get DDT address
    MOVW    #ERR_K_COMMAND_LENGTH,- ; Length of packet containing SCSI command
          DDT$W_ERRORBUF(R0)        ; in the DDT
    JSB     G^ERL$DEVICERR           ; Log a device error
    BBCC    #UCB$V_ERLOGIP,-        ; Clear error log in progress
          UCB$W_STS(R5),30$         ;
    MOVL    UCB$L_EMB(R5),R2         ; Get address of error message buffer
    BEQL    30$                     ; Branch if none available
    JSB     G^ERL$RELEASEMB         ; Release the error log buffer
30$:      POPR   #^M<R0,R2,R9,R10>   ; Restore registers
40$:      RSB                      ; Return to caller
    .DISABLE LSB                    ; LOG_ERROR

    .SBTTL  SK_REG_DUMP             - Device register dump routine
```

VMS Template SCSI Class Driver

```
;++
; SK_REG_DUMP
;
; This routine dumps device-specific information into an error log packet.
; The format of this information is as follows:
;
;      +-----+
;      |      Longword count      | 4 bytes
;      +-----+
;      |      Revision            | 1 byte
;      +-----+
;      |      HW revision         | 4 bytes
;      +-----+
;      |      Error Type          | 1 byte
;      +-----+
;      |      SCSI ID             | 1 byte
;      +-----+
;      |      SCSI LUN            | 1 byte
;      +-----+
;      |      SCSI SUBLUN        | 1 byte
;      +-----+
;      |      Port status         | 4 bytes
;      +-----+
;      |      SCSI CMD LENGTH     | 1 byte
;      +-----+
;      |      SCSI CMD BYTES      | Up to 12 bytes
;      +-----+
;      |      SCSI STS            | 1 byte
;      +-----+
;      |      Error Text Count    | 1 byte
;      +-----+
;      |      Error Text          | Up to 60 bytes
;      +-----+
;
; Inputs:
;
;      R0      - Output buffer address
;      R5      - UCB address
;
; Outputs:
;
;      R1-R3   - Destroyed
;      All other registers preserved
;--
```

VMS Template SCSI Class Driver

```

SK_REG_DUMP:
    .ENABLE LSB ; SK_REG_DUMP
    MOVL #<<ERR_K_COMMAND_LENGTH/4>+1>,-
        (R0)+ ; Length of error log packet in words
    MOVB #0,(R0)+ ; Save revision level
    CLRL (R0)+ ; Save hardware revision level
    MOVB R7,(R0)+ ; Save error type
    MOVZWL UCBSW_UNIT(R5),R1 ; Get unit number
    CLRL R2 ; Prepare for extended divide
    EDIV #100,R1,R1,R2 ; Extract SCSI bus ID from unit number
    MOVB R1,(R0)+ ; Save SCSI bus ID
    MOVL R2,R1 ; Copy LUN, SUBLUN
    CLRL R2 ; Prepare for extended divide
    EDIV #10,R1,R1,R2 ; Extract LUN and SUBLUN
    MOVB R1,(R0)+ ; Save LUN field
    MOVB R2,(R0)+ ; Save SUBLUN field
    MOVL R8,(R0)+ ; Save port status code
    MOVL UCB_L_SCDRP(R5),R1 ; Get active SCDRP address
    BEQL 50$ ; Branch if none active
    MOVL SCDRP$L_CMD_PTR(R1),R2 ; Get address of SCSI command
    BEQL 50$ ; Branch if none active
    MOVL (R2)+,R3 ; Get number of SCSI command bytes
    MOVB R3,(R0)+ ; Save command length
10$: MOVB (R2)+,(R0)+ ; Save a command byte
    SOBGTR R3,10$ ; Continue for entire SCSI command
    MOVL SCDRP$L_STS_PTR(R1),R2 ; Get address of status byte
    MOVB (R2),(R0)+ ; Save SCSI status byte
    MOVB (R11)+,R3 ; Get count of number of text bytes.
    BEQL 50$ ; If no text finished
    MOVB R3,(R0)+ ; Save text length
20$: MOVB (R11)+,(R0)+ ; Save a text byte in error packet
    SOBGTR R3,20$ ; Continue for entire text string command
50$: RSB ; Return
    .DISABLE LSB ; SK_REG_DUMP
SK_PATCH:
    .BLKB 200 ; Patch space
SK_END: ; Last location in driver
    .END

```

D Interpreting SCSI Driver Error Log Entries

As dictated by the VMS SCSI class/port driver architecture, a VMS SCSI port driver logs port-specific events in a defined form. Port driver error log entries can provide clues that are helpful in resolving problems that may occur during the development of a third-party SCSI class driver.

The VMS SCSI class/port driver architecture also specifies a form for class driver error log entries. Because of the value of the error log in debugging, Digital highly recommends that a third-party SCSI class driver incorporate an error logging routine that records events significant to the device. (See Section 3.5.2 for a discussion of the procedures by which class drivers interpret status, format events, and register error log entries.)

You can use the VMS Error Log Utility, as described in the *VMS Error Log Utility Manual*, to list and format SCSI port and class driver error log entries.

D.1 SCSI Port Driver Error Log Entries

The SCSI port driver is responsible for all low-level activity associated with sending commands to a target SCSI device. The standard format of an error log entry generated by a port driver has two parts: a port-common section and a port-specific section. All VMS port drivers provide the same type of information in the port-common section of the entry. The information a port driver supplies in the port-specific section depends upon the SCSI port hardware that is in use.

Table D-1 describes the contents of a formatted port driver error log entry. A reference number in the table associates each table item with an entry in the representative error logs presented in Examples D-1 and D-2.

When inspecting a SCSI port driver error log entry, you should first examine the *error type* and *error subtype*. These fields indicate the nature of the event that occurred. You should also check the *SCSI ID* field to determine the device for which the event has been reported. Although the SCSI ID may not always identify the device responsible for the event, it may help you interpret the significance of the information in this and other error log entries.

Next, you should examine the *SCSI CMD* field to determine which SCSI command was current at the time of the logged event. The *phase queue* entry lists those SCSI bus phases that have been successfully completed during execution of this command. You can derive the current phase of the SCSI bus by referring to the description of the phase signals defined for the command in the ANSI SCSI specification. In addition, the port-specific section of the error log entry of certain VMS port drivers lists the currently asserted bus lines.

Interpreting SCSI Driver Error Log Entries

D.1 SCSI Port Driver Error Log Entries

Finally, the sets of counters that appear in a port driver error log entry can help you discern patterns of activity on the SCSI bus. For instance, a large number of parity errors are a symptom of a bus termination problem or other hardware problem.

Table D–1 Key to Port Driver Error Log Entries

Field ¹	Description		
General Event Information			
Error type ❶	Error type and subtype. The following types and subtypes are defined.		
Error subtype ❷	Error²	Definition	Description
	01	BUS_HUNG	SCSI bus was continuously busy during an arbitration attempt.
	02	ARB_FAIL	Arbitration of SCSI bus failed due to activity of higher priority devices.
	03	SEL_FAIL	Selection failed.
	04	TIMEOUT	Timeout occurred.
	05	PARITY_ERROR	Parity error detected.
	06	PHASE_ERROR	SCSI bus phase error. A phase error results from a missing SCSI bus phase, a phase that is entered more than once, or a bad phase sequence.
		Subtype²	Description
		01	Missing phase error
		02	Bad phase transition
		03	Timeout waiting for phase interrupt
		04	Unexpected phase change during DATA IN; error during REQ-ACK
		05	Unexpected phase change during DATA OUT; error during REQ-ACK
		06	Phase change timeout during DATA IN
		07	Phase change timeout during DATA OUT
		08	Timeout waiting for phase change
		09	Phase change timeout during COMMAND OUT
		10	Bus freed during command phase

¹Reference numbers refer to Examples D–1 and D–2.

²Error type and subtype values are rendered in hexadecimal format.

(continued on next page)

Interpreting SCSI Driver Error Log Entries

D.1 SCSI Port Driver Error Log Entries

Table D–1 (Cont.) Key to Port Driver Error Log Entries

Field ¹	Description		
General Event Information			
	07	BUS_RESET	Bus reset detected. Subtype² Description 01 Reset occurred while no I/O operation was active
	08	UNEXPECTED_INTERRUPT	Unexpected interrupt received.
	09	BUS_RESET_ISSUED	Bus reset initiated.
	10	RESEL_ERR	Error following a device disconnect. Subtype² Description 01 Bad parity during reselect 02 No target ID during reselect 03 Multiple target IDs during reconnect 04 No connection to this target 05 Failed while no reselect was pending 08 SEL failed to clear during reselect 09 REQ failed to set during reselect 10 Bad RESEL message
	11	CTL_ERR	Error detected by controller.
	12	BUS_ERR	Controller detected a SCSI bus protocol error.
	13	ILLEGAL_MSG	Illegal message received.
	14–19		Reserved.
SCSI ID ③			SCSI ID of the device to which the current command is being sent. Valid SCSI IDs range from 0 to 7. A value of FF ₁₆ in this entry indicates that the SCSI ID is unknown or not relevant (as in the case of a spurious bus reset).
SCSI CMD ④			Current SCSI command.
SCSI MSG ⑤			Current SCSI message.
SCSI STATUS ⑥			Current SCSI status. A status value of FF ₁₆ indicates that the SCSI bus has not yet returned status.

Port Error Counters³

Bus busy count ⑦	Number of times the port driver has attempted to arbitrate for the SCSI bus and has found the bus hung for an extended period of time. A value in this field indicates either that the bus is extremely busy or a device on the bus is hung.
------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

¹Reference numbers refer to Examples D–1 and D–2.

²Error type and subtype values are rendered in hexadecimal format.

³The port error counters record errors that cannot be attributed to a specific device on the SCSI bus.

(continued on next page)

Interpreting SCSI Driver Error Log Entries

D.1 SCSI Port Driver Error Log Entries

Table D-1 (Cont.) Key to Port Driver Error Log Entries

Field ¹	Description
Port Error Counters³	
Unsolicited reset count ⁸	Number of times the port driver has received a reset interrupt that is not due to its own pulling of the bus reset line. This could be due to noise on the reset line or to a device (or another initiator) pulling the bus reset line.
Unsolicited interrupt count ⁹	Number of times the port driver has received an unsolicited interrupt.
Connection Error Counters⁴	
Arbitration fail count ¹⁰	Number of times the port driver has attempted to arbitrate for the SCSI bus and has failed. Arbitration is attempted only when a bus free condition is detected. Thus, this counter reflects the number of times a low priority device loses arbitration to a higher priority device.
Selection fail count ¹¹	Number of times the port driver has attempted to select a target device and has failed. This could happen just after a target device has been reset, if it has been powered off or disconnected from the bus, or if it is hung in such a way that it is not also hanging the bus.
Parity error count ¹²	Number of times the port driver has detected a parity error while sending a command to a target SCSI device.
Phase error count ¹³	Number of times the port driver has detected a phase error while sending a command to a target SCSI device.
Bus reset count ¹⁴	Number of times the port driver has reset the bus because it was unable to send a command to a target SCSI device. The port driver resets the bus for a number of reasons: for instance when it detects a bus hang or a phase error.
Bus error count ¹⁵	Number of times the SCSI controller has detected an error on the SCSI bus. This field is not used by SCSI controllers on MicroVAX/VAXstation 3100 systems.
Controller error count ¹⁶	Number of times the SCSI controller has reported an internal error. This field is not used by SCSI controllers on MicroVAX/VAXstation 3100 systems.
Retry Counters	
Arbitration retry count ¹⁷	Number of arbitration retries attempted. A value of -1 indicates that the counter contains no valid data.
Selection retry counter ¹⁸	Number of selection retries attempted. A value of -1 indicates that the counter contains no valid data.
Bus busy retry counter ¹⁹	Number of bus busy retries attempted. A value of -1 indicates that the counter contains no valid data.
Phase Queue	

¹Reference numbers refer to Examples D-1 and D-2.

³The port error counters record errors that cannot be attributed to a specific device on the SCSI bus.

⁴The connection error counters record errors that can be attributed to a specific device on the SCSI bus. The SCSI ID field specifies the devices to which the command was being sent when the error occurred.

(continued on next page)

Interpreting SCSI Driver Error Log Entries

D.1 SCSI Port Driver Error Log Entries

Table D–1 (Cont.) Key to Port Driver Error Log Entries

Field ¹	Description
Phase Queue	
Element phase queue 20	Lists the SCSI phases that have been entered and completed during the execution of the current command. The digit preceding the list indicates the number of phases that have been completed.
Port dependent data 21	Contents of port controller registers. This section of the error log entry contains information specific to the SCSI port controller employed by the system.

¹Reference numbers refer to Examples D–1 and D–2.

D.2 SCSI Class Driver Error Log Entries

A SCSI class driver logs device-specific events in the manner described in Section 3.5.2. Although all class drivers use a common extension to the standard error message buffer when logging errors, the types of events detected and reported by class drivers are specific to the devices they control.

Table D–2 describes the contents of a formatted class driver error log entry. Each item in this table is likewise associated with a field in the error log contained in Example D–3.

Table D–2 Key to Class Driver Error Log Entries

Field ¹	Description																		
Hardware revision 22	Hardware revision information, returned by the SCSI INQUIRY command.																		
Error type 23	Type of error detected by the class driver. A SCSI class driver defines device-specific error types according to the nature of the device it services. The following error values are interpreted by the VMS Error Log Utility: <table border="1"><thead><tr><th>Error²</th><th>Name</th><th>Description</th></tr></thead><tbody><tr><td>01</td><td>CON_ERR</td><td>Attempt to connect to the port driver failed</td></tr><tr><td>02</td><td>MAP_ERR</td><td>Attempt to map a user buffer failed</td></tr><tr><td>03</td><td>SND_ERR</td><td>Attempt to send a SCSI command failed</td></tr><tr><td>04</td><td>INV_INQ</td><td>Invalid inquiry data was received</td></tr><tr><td>05</td><td>EXT_SNS_DAT</td><td>Extended sense data was returned from the SCSI device</td></tr></tbody></table>	Error ²	Name	Description	01	CON_ERR	Attempt to connect to the port driver failed	02	MAP_ERR	Attempt to map a user buffer failed	03	SND_ERR	Attempt to send a SCSI command failed	04	INV_INQ	Invalid inquiry data was received	05	EXT_SNS_DAT	Extended sense data was returned from the SCSI device
Error ²	Name	Description																	
01	CON_ERR	Attempt to connect to the port driver failed																	
02	MAP_ERR	Attempt to map a user buffer failed																	
03	SND_ERR	Attempt to send a SCSI command failed																	
04	INV_INQ	Invalid inquiry data was received																	
05	EXT_SNS_DAT	Extended sense data was returned from the SCSI device																	

¹These numbers refer to Example D–3.

²Error type values are rendered in hexadecimal format.

(continued on next page)

Interpreting SCSI Driver Error Log Entries

D.2 SCSI Class Driver Error Log Entries

Table D-2 (Cont.) Key to Class Driver Error Log Entries

Field ¹	Description
	06 INV_MOD_SNS Invalid mode sense data returned from the SCSI device
	07 REASSIGN_BLK Reassign block
	08 DIAG_DATA Invalid diagnostic data returned to the VMS SCSI tape class driver
SCSI ID ²⁴	SCSI ID of the device to which the current command was sent. The SCSI ID is an integer between 0 and 7.
SCSI LUN ²⁵	SCSI logical unit number of the device to which the current command was sent. The SCSI LUN is an integer between 0 and 7.
SCSI SUBLUN ²⁶	Not used. This field always contains 0.
Port status ²⁷	Current port status. A value of -1 indicates that there is no valid data in this field.
SCSI CMD ²⁸	Current SCSI command.
SCSI STS ²⁹	Current SCSI status. A value of -1 indicates that there is no valid data in this field.
Additional data ³⁰	<p>Additional data, preceded by a byte count of the data. A class driver defines what additional data would be meaningful in an error log entry based on the type of device it services. Additional data is displayed by the VMS Error Log Utility as untranslated longwords.</p> <p>Note that the VMS Error Log Utility can interpret extended sense data values when the extended sense data received error type is reported in the log and the driver's error logging routine places the sense data in this field of its error message buffer. Thus, the error log entry that appears in Example D-3 interprets the logged sense data as a unit attention message signifying a power on or reset condition.</p>

¹These numbers refer to Example D-3.

D.3 Resolving SCSI Class Driver Problems Using Error Logs

Taken as a unit, Examples D-1 through D-3 illustrate a standard event sequence that may occur during a SCSI bus transaction. This sequence involves the following actions:

- 1 The port detects an abnormal event, such as a timeout. (Example D-1)
- 2 The port driver resets the SCSI bus. (Example D-2)
- 3 A class driver receives extended sense data from the device informing it of the reset event. (Example D-3)

These events typically occur for one of the following reasons:

- The class driver has sent a SCSI command to a device that the device does not understand or does not support.
- The class driver has sent a misformatted SCSI command packet to a device.
- The class driver has failed to deallocate a port resource, such as a command buffer or port map registers.

Interpreting SCSI Driver Error Log Entries

D.3 Resolving SCSI Class Driver Problems Using Error Logs

- A hardware failure has occurred on the SCSI bus.

Example D-1 SCSI Bus Phase Error Port Driver Error Log Entry

```

V A X / V M S                SYSTEM ERROR REPORT          COMPILED 13-SEP-1989 15:05
                               PAGE 1.

***** ENTRY 208. *****
ERROR SEQUENCE 43028.          LOGGED ON:          SID 0A000005
DATE/TIME 13-SEP-1989 15:03:35.08          SYS_TYPE 04010102
SCS NODE:                               VAX/VMS X5.2-1C

DEVICE ATTENTION KA420 CPU REV# 6.

SCSI PORT SUB-SYSTEM, UNIT _PKA0:

  ERROR TYPE          06
  SCSI ID             02
  SCSI CMD            CA810208
                    0019
  SCSI MSG            00
  SCSI STATUS         FF
  PORT ERROR CNT      00000000
                    00000000
                    00000000
  CONN ERROR CNT      00000000
                    00000000
                    00000000
                    00000001
                    00000000
                    00000000
                    00000000
                    00000000
  SCSI RETRY CNT      00000000
                    0000
  PHASE QUEUE         0908

  PORT DEPENDENT DATA
  CNTLR INI CMD       02
  CNTLR MODE          20

```

SCSI BUS PHASE ERROR ①
SUB-ERROR TYPE = 02(X) ②
SCSI ID = 2. ③
READ ④
COMMAND COMPLETE ⑤
NO STATUS RECEIVED ⑥
BUS BUSY CNT = 0. ⑦
UNSOL RESET CNT = 0. ⑧
UNSOL INTRPT CNT = 0. ⑨
ARB FAIL CNT = 0. ⑩
SEL FAIL CNT = 0. ⑪
PARITY ERR CNT = 0. ⑫
PHASE ERR CNT = 1. ⑬
BUS RESET CNT = 0. ⑭
BUS ERROR CNT = 0. ⑮
CONTROLLER ERROR CNT = 0. ⑯
ARB RETRY CNT = 0. ⑰
SEL RETRY CNT = 0. ⑱
BUSY RETRY CNT = 0. ⑲
2. ELEMENT PHASE QUEUE ⑳
 _ARBITRATION
 _SELECTION

ATN ASSERTED
PARITY CHECK ENABLED

(continued on next page)

Interpreting SCSI Driver Error Log Entries

D.3 Resolving SCSI Class Driver Problems Using Error Logs

Example D-1 (Cont.) SCSI Bus Phase Error Port Driver Error Log Entry

```

CNTLR TAR CMD      00
CNTLR CURR STS    78

                                C/D ASSERTED
                                MSG ASSERTED
                                REQ ASSERTED
                                BUSY ASSERTED

CNTLR STATUS      02

                                ATN ASSERTED

DMA CNT           00000000
DMA ADDRESS       00004200
DMA DIR           01
  
```

Example D-2 SCSI Bus Reset Port Driver Error Log Entry

```

***** ENTRY 209. *****
READ OPERATION
ERROR SEQUENCE 43029.          LOGGED ON:      SID 0A000005
DATE/TIME 13-SEP-1989 15:03:35.08             SYS_TYPE 04010102
SCS NODE:                                     VAX/VMS X5.2-1C

DEVICE ATTENTION KA420 CPU REV# 6.

SCSI PORT SUB-SYSTEM, UNIT _PKA0:

  ERROR TYPE          09
                                BUS RESET INITIATED ①
                                SUB-ERROR TYPE = 00(X) ②

  SCSI ID             02
                                SCSI ID = 2. ③

  SCSI CMD            CA810208
                                READ ④
                                0019
                                COMMAND COMPLETE ⑤

  SCSI MSG            00
                                NO STATUS RECEIVED ⑥

  SCSI STATUS         FF

  PORT ERROR CNT     00000000
                                BUS BUSY CNT = 0. ⑦
                                00000000
                                UNSOL RESET CNT = 0. ⑧
                                00000001
                                UNSOL INTRPT CNT = 1. ⑨
                                00000000

  CONN ERROR CNT     00000000
                                ARB FAIL CNT = 0. ⑩
                                00000000
                                SEL FAIL CNT = 0. ⑪
                                00000000
                                PARITY ERR CNT = 0. ⑫
                                00000001
                                PHASE ERR CNT = 1. ⑬
                                00000001
                                BUS RESET CNT = 1. ⑭
                                00000000
                                BUS ERROR CNT = 0. ⑮
                                00000000
                                CONTROLLER ERROR CNT = 0. ⑯

  SCSI RETRY CNT     00000000
                                ARB RETRY CNT = 0. ⑰
                                0000
                                SEL RETRY CNT = 0. ⑱
                                BUSY RETRY CNT = 0. ⑲

  PHASE QUEUE        0908
                                2. ELEMENT PHASE QUEUE ⑳
  
```

(continued on next page)

Interpreting SCSI Driver Error Log Entries

D.3 Resolving SCSI Class Driver Problems Using Error Logs

Example D-2 (Cont.) SCSI Bus Reset Port Driver Error Log Entry

```

                                _ARBITRATION
                                _SELECTION

PORT DEPENDENT DATA21
    CNTLR INI CMD      00
    CNTLR MODE        00
    CNTLR TAR CMD     00
    CNTLR CURR STS   00
    CNTLR STATUS      08

                                PHASE MATCH

    DMA CNT           00000000
    DMA ADDRESS       00004200
    DMA DIR            01

                                READ OPERATION

```

Example D-3 SCSI Bus Reset Class Driver Error Log Entry

```

***** ENTRY      210. *****
ERROR SEQUENCE 43030.          LOGGED ON:      SID 0A000005
DATE/TIME 13-SEP-1989 15:03:35.49          SYS_TYPE 04010102
SCS NODE:                                VAX/VMS X5.2-1C

DEVICE ERROR KA420 CPU REV# 6.

RZ23 SUB-SYSTEM, UNIT _DKA200:
    HW REVISION      38313630
                                HW REVISION = 061822
    ERROR TYPE       05
                                EXTENDED SENSE DATA RECEIVED23
    SCSI ID          02
                                SCSI ID = 2.24
    SCSI LUN         00
                                SCSI LUN = 0.25
    SCSI SUBLUN     00
                                SCSI SUBLUN = 0.26
    PORT STATUS     00000001
                                %SYSTEM-S-NORMAL, NORMAL SUCCESSFUL
                                COMPLETION27
    SCSI CMD         CA810208
                                0019
                                READ28
    SCSI STATUS     02
                                CHECK CONDITION29

EXTENDED SENSE DATA30
    EXTENDED SENSE  00060070
                                0C000000
                                00000000
                                00000029
                                0000

                                UNIT ATTENTION
                                POWER ON OR RESET OCCURRED

```

(continued on next page)

Interpreting SCSI Driver Error Log Entries

D.3 Resolving SCSI Class Driver Problems Using Error Logs

Example D-3 (Cont.) SCSI Bus Reset Class Driver Error Log Entry

```

UCB$B_ERTCNT      00          0. RETRIES REMAINING
UCB$B_ERTMAX      00          0. RETRIES ALLOWABLE
ORB$L_OWNER       00010001    OWNER UIC [001,001]
UCB$L_CHAR        1C4D4008    DIRECTORY STRUCTURED
                                FILE ORIENTED
                                SHARABLE
                                AVAILABLE
                                MOUNTED
                                ERROR LOGGING
                                CAPABLE OF INPUT
                                CAPABLE OF OUTPUT
                                RANDOM ACCESS

UCB$W_STS         0010        ONLINE
UCB$L_OPCNT       0000104E    4174. QIO'S THIS UNIT

UCB$W_ERRCNT      0001        1. ERRORS THIS UNIT
IRP$W_BCNT        3200        TRANSFER SIZE 12800. BYTE(S)
IRP$W_BOFF        0000        TRANSFER PAGE ALIGNED
IRP$L_PID         0001000D    REQUESTOR "PID"
IRP$Q_IOSB        00000000    IOSB, 0. BYTE(S) TRANSFERRED
                    00000000

***** ENTRY      211. *****
ERROR SEQUENCE 43031.          LOGGED ON:      SID 0A000005
DATE/TIME 13-SEP-1989 15:04:51.53  SYS_TYPE 04010102
SCS NODE:                          VAX/VMS X5.2-1C

TIME STAMP KA420 CPU REV# 6.

```

E

VMS Requirements and Restrictions for Non-Digital-Supplied SCSI Devices

The VMS operating system offers a compliant implementation of the ANSI SCSI bus. Every effort has been made to support the widest possible variety of SCSI peripherals. However, the SCSI standard is so flexible that additional requirements and restrictions are necessary. Any SCSI device that does not conform to these requirements cannot be used on a MicroVAX/VAXstation system.

This appendix describes all the requirements and restrictions necessary for writing a third-party SCSI class driver. It does *not* enumerate the additional rules necessary for devices that use the VMS disk and tape class drivers, as that information is beyond the scope of this appendix.

Note that some of these restrictions may be removed in the future, when new SCSI features are supported in the SCSI port drivers. New restrictions may also be added as they are found necessary.

E.1 VMS Requirements

This section describes the requirements and restrictions that a SCSI device must meet in order to run properly on a MicroVAX/VAXstation system.

E.1.1 Conformance to Standards

The VMS SCSI implementation is currently based on the SCSI-1 standard. However, the SCSI-2 standard is compatible with SCSI-1, and clarifies many gray areas in the SCSI-1 standard. Therefore, all VMS requirements for non-Digital-supplied SCSI devices are based on the SCSI-2 standard.

The device must implement all of the mandatory features of the SCSI-2 standard as described in the specification. The device is permitted to implement any optional features, as long as they are implemented according to the SCSI-2 standard. The device may implement vendor-unique features, as long as they are implemented in areas clearly designated as such by the standard.

The SCSI-2 standard is the official guide to what must be implemented and how it must be implemented by the device. The remainder of this appendix is not intended to replace the SCSI-2 standard, but rather to specify which options of the SCSI-2 standard must be implemented, and to impose additional restrictions when necessary.

Requirements and Restrictions

E.1 VMS Requirements

E.1.2 Cabling

MicroVAX/VAXstation systems do not support the wide-SCSI option. Therefore, only the A Cable is used.

Only single-ended drivers and receivers can be used on MicroVAX/VAXstation systems.

The fast synchronous data transfer option is not supported, so the cables do not need to match the requirements for this option. (See Section 4.2.3 of the SCSI-2 standard.)

E.1.3 Connector Requirements

Alternative 2 (low-density) nonshielded A Cable connectors are used inside the system box. (See Section 4.3.1.2 of the SCSI-2 standard.)

Alternative 2 (low-density) shielded A Cable connectors are used between device enclosures. (See Section 4.3.2.2 of the SCSI-2 standard.)

The connectors should be shrouded and keyed in order to protect users from cabling errors.

E.1.4 SCSI Bus Termination

MicroVAX/VAXstation systems implement single-ended termination. Near-end termination is provided within the system enclosure. The far-end termination for the external bus should be provided by a terminating connector plugged into the last SCSI jack. There shall be no other terminators on the bus. The device should *not* terminate the bus internal to itself, even if it is the last device on the bus. If all devices follow this rule, the user can reconfigure his bus with a much higher probability of proper bus termination. Failure to terminate the bus properly leads to system crashes that are difficult to pinpoint and to corruption of data.

Vendors who want to install devices inside system enclosures must contact the appropriate Digital hardware engineering group for details.

E.1.5 Terminator Power

Each device enclosure (whether it contains initiators, targets, or both) must provide terminator power to the TERMPWR pin.

A 1-amp current limiter is required.

E.1.6 Dynamic Reconfiguration of Devices

Devices on the MicroVAX/VAXstation SCSI bus may not be added to the bus, removed from the bus, or recabled while the system is in operation. Failure to meet this requirement may cause loss of user data or system failure.

E.1.7 External Boxes

Devices residing outside the main system box should remain powered on at all times while the system is in operation, because

- Some powered-off SCSI devices fail to present high impedance to the SCSI bus, leaving the bus unusable.
- A powered-off device can reduce the terminator power on the bus to unacceptable levels, causing user data corruption or system failure.
- The device can spike various SCSI bus signals during power-on or power-off, leading to user data corruption or system failure.

Many users cycle the power on external boxes while the system is running despite warnings, because it often “seems to work.” For this reason, the device should be designed to minimize the possibility of these problems. However, the fact remains that powering off boxes can lead to serious consequences, and Digital strongly recommends that external devices remain powered on at all times during system operation.

E.1.8 Device Behavior Following Power-On

The device must meet the following requirements after it is powered on.

- Within 5 seconds of power-on, the device must be able to respond to a SCSI bus selection so that the VMS operating system is aware that there is an active device at this SCSI ID. However, the device is permitted to return any of the following errors until it is ready for normal operation:
 - BUSY status.
 - CHECK CONDITION status, followed by NOT READY sense key in response to the next REQUEST SENSE command.
 - CHECK CONDITION status, followed by UNIT ATTENTION sense key in response to the next REQUEST SENSE command. (Note that UNIT ATTENTION may only be returned once per power-up/bus-reset, as described in the SCSI-2 standard.)
- Within 15 seconds of power-on, the device must be able to successfully complete an INQUIRY command so the device can be identified.
- The VMS operating system does not specify a minimum value by which devices must respond to normal commands following power-on, because this value is determined by the third-party SCSI class driver.

E.1.9 Device Behavior Following Bus Reset

A SCSI bus reset is a serious event that may cause some devices such as tape drives to lose user data. Therefore, only the MicroVAX/VAXstation processor may set the SCSI RST signal, and even then only as a last resort. The device must never set the SCSI RST signal. The device must behave as follows after a SCSI bus reset, except when the reset occurs as the result of power-on (see Section E.1.8).

Requirements and Restrictions

E.1 VMS Requirements

- The device must be able to respond to a SCSI bus selection within 5 seconds of the bus reset. However, the device is permitted to return one of the following status values once it begins to accept bus selections:
 - BUSY status
 - CHECK CONDITION status followed by NOT READY sense key in response to the next REQUEST SENSE command.
 - CHECK CONDITION status, followed by UNIT ATTENTION sense key in response to the next REQUEST SENSE command. (Note that UNIT ATTENTION may only be returned once per power-up/bus-reset, as described in the SCSI-2 standard.)

The device may not return any other status values or sense keys, unless it is ready for normal operation.

- The device must be able to respond to normal commands within 10 seconds of the bus reset.

The device must be able to handle multiple bus resets in succession, with no ill effects. The device must recover within the time limits specified above, following the last bus reset.

E.1.10 Data Transfer

The following modes of data transfer are supported on MicroVAX/VAXstation systems:

System	Data Transfer Mode(s)
VAXstation 3100 MicroVAX/VAXserver 3100	Asynchronous
VAXstation 3520/3540	Asynchronous and synchronous

The fast synchronous transfer option is not supported on any systems. (See Section 4.8 of the SCSI-2 standard for a description of this option.)

The device must generate odd parity during all information transfer phases in which the device writes data to the SCSI bus, and check odd parity during all information transfer phases in which it reads data from the bus.

Current MicroVAX/VAXstation systems use an intermediate buffer for all incoming and outgoing SCSI data. When a system performs an I/O transfer involving a SCSI device, it allocates a portion of this buffer for the duration of the operation. Because the VMS operating system allows multiple concurrent I/O operations to the SCSI bus, the VMS operating system cannot allocate the full buffer to a single device. The following list shows the intermediate buffer size, the maximum data transfer size and the recommended data transfer size on supporting VAX systems:

- Intermediate buffer size — 128 KB on all systems supported today, which include the VAXstation 3100, MicroVAX/VAXserver 3100, and VAXstation 3520/3540.

Requirements and Restrictions

E.1 VMS Requirements

- Maximum data transfer size — 64 kilobytes on all systems supported today. If a device needs to transfer more than the maximum transfer size, the device must be modified to allow the data to be returned across multiple SCSI commands. A single SCSI command cannot transfer more than the maximum size.
- Recommended data transfer size — For the best overall system throughput, the device should only transfer 16 kilobytes per SCSI command.

E.1.11 Initiator/Target Operation

Normally, the MicroVAX/VAXstation processor is the only entity that may act as an initiator on the SCSI bus. However, the user's device is also permitted to act as an initiator, as long as it is for the purpose of selecting and sending a command to the processor. This feature is called "asynchronous event notification" in the SCSI-2 specification. The device may not select or issue a command to any other device on the bus.

E.1.12 SCSI IDs and Logical Unit Numbers

The SCSI ID is a value ranging from 0 to 7 that determines the logical position of the device controller on the SCSI bus. The device must have user-accessible switches or jumpers so that the SCSI ID can be changed easily in the field.

The SCSI ID assignments vary from system to system. However, the following SCSI ID assignments are used for most systems:

- SCSI ID 7 is used for the CPU on the VAXstation 3520/3540.
- SCSI ID 6 is used for the CPU on the MicroVAX/VAXstation 3100, MicroVAX 3100, and VAXserver 3100.
- SCSI ID 5 is used for the tape drive.
- SCSI IDs 2 to 4 are used for disks.
- SCSI IDs 0 and 1 are free.

A non-Digital-supplied device may use any SCSI ID that does not conflict with another SCSI ID on the same bus.

The device logical unit number (LUN) distinguishes between multiple units attached to a single SCSI controller. The VMS operating system supports logical unit numbers ranging from 0 to 7, and stores the logical unit number in the IDENTIFY message that is sent prior to every command. However, the VMS operating system does not support sub-LUNs, a feature described in the SCSI-1 standard but whose use is discouraged in the SCSI-2 standard.

Requirements and Restrictions

E.1 VMS Requirements

E.1.13 Bus Phases

The device must conform to the bus state transition table shown in Section 5.3 of the SCSI-2 standard, "Phase Sequences." In addition, the device must meet the following criteria:

- RESELECTION phase — When the host responds to a target reselection by asserting the BSY signal, the target must deassert the SEL signal within 500 microseconds of the BSY assertion. The target should also assert REQ for the first command or message byte within 500 microseconds of the BSY assertion.
- COMMAND phase — The device may only enter the command phase once per command. For instance, linked commands may not be issued to the device.
- ATTENTION response — The device must respond to an ATN condition at every phase transition, as long as the processor sets the ATN signal before it asserts the ACK signal for the last byte of the previous phase. During the data phase, the device should also respond to an ATN condition at least once per millisecond.
- STATUS phase — The device must enter the status phase once (no more, no less) per command. The only exception is during error cases when the device immediately enters the BUS FREE phase, as defined in the SCSI-2 standard. Storage devices should not enter the status phase following a WRITE BUFFER command until the data has been written to nonvolatile media, in order to avoid accidental loss of user data.
- BUS FREE — The target must not drive any SCSI bus lines after the deassertion of BSY, except during selection or reselection, as outlined in the SCSI-2 standard.

E.1.14 Disconnect and Reselection

The device must adhere to the following rules for disconnecting from the bus and reselecting the initiator.

- The device should proceed quickly through all phases of a SCSI command, and should not hold the bus for long periods without disconnecting.
- While a device is disconnected from the bus, it must respond to a selection with ATN by entering the MESSAGE OUT phase and accepting the message from the processor. The device must be prepared to properly handle either the ABORT or the BUS DEVICE RESET message at this time.
- If the initiator fails to respond to reselection, the device must time out the reselection attempt after 250 milliseconds as described in the Reselection Timeout Procedure, Section 5.1.4.2 of the SCSI-2 standard. The device must employ the second option described in this section of the SCSI-2 standard to release the bus; that is, the device may not assert the RST signal.

Requirements and Restrictions

E.1 VMS Requirements

- After a reselection timeout, the target should retry the host at least four times, but not more than 10 times. If the device retries more than 10 times, it can unnecessarily tie up the bus in an error condition. In order to prevent the retries from timing out other devices, the device should delay at least 20 microseconds between retry attempts.

The device should not hold the bus busy in a single state before disconnecting for longer than the following periods of time:

- 500 microseconds is recommended for best system performance.
- 3 milliseconds is required for performance reasons.

E.1.15 Messages

The VMS operating system accepts the following messages during the MESSAGE IN phase. The target device must not generate any messages except those stated in this list:

- COMMAND COMPLETE
- DISCONNECT
- IDENTIFY
- RESTORE POINTERS
- SAVE DATA POINTER
- SYNCHRONOUS DATA TRANSFER REQUEST (only for the VAXstation 3520/3540)

The VMS operating system generates the following messages, which the target must accept and process as described in the SCSI-2 standard.

- ABORT
- BUS DEVICE RESET
- IDENTIFY
- INITIATOR DETECTED ERROR
- MESSAGE PARITY ERROR
- MESSAGE REJECT
- NO OPERATION
- SYNCHRONOUS DATA TRANSFER REQUEST (only for devices that support synchronous data transfers)

When a device selects the host for the purpose of performing an asynchronous event notification, the host accepts the following messages in addition to those stated above. When one of these messages is received, the VMS operating system notifies the third-party SCSI class driver of the received message.

- ABORT
- BUS DEVICE RESET

Requirements and Restrictions

E.1 VMS Requirements

- INITIATOR DETECTED ERROR
- MESSAGE PARITY ERROR
- MESSAGE REJECT
- NO OPERATION

The following messages have additional requirements:

- **ABORT** — The ABORT message must be implemented as described in the SCSI-2 standard. When the target receives an ABORT message, it must terminate the current operation. The device must stop in a manner that does not cause side effects; for example, no parity errors should be written to the media, and devices such as tape drives, which store user data in nonvolatile memory, must write the data back to nonvolatile media before honoring the ABORT message. Following the abort operation, the device must use the recovery guidelines described in Section E.1.9.
- **COMMAND COMPLETE** — The device may not hold the BSY signal for more than 10 microseconds after sending the COMMAND COMPLETE message or the DISCONNECT message.
- **SAVE DATA POINTER** — This message is used to save the context of an operation while a device is disconnected from the bus. However, this message is not necessary or useful when the target disconnects before starting to transfer data. In this case, the target should not use this message as it can waste roughly 200 microseconds of VMS CPU time per I/O operation. Once the device transfers data, it must issue the SAVE DATA POINTER message before disconnecting.
- **RESTORE POINTERS** — the VMS operating system always performs an implicit restore pointers operation when the target reselects. The target should not use this message as it can waste roughly 200 microseconds of VMS CPU time per I/O operation. Note that the RESTORE POINTERS message cannot be used to retry the COMMAND or STATUS phases.

E.1.16 Commands

The VMS operating system allows commands up to 128 bytes in length. Otherwise, there are no restrictions on the size or type of commands that may be sent, as long as they conform to the SCSI-2 standard.

E.1.17 INQUIRY Command

The INQUIRY command must return one of the following values in the ANSI-Approved Version field:

- 01 The device complies with ANSI X3.131-1986 (SCSI-1).
- 02 The device complies with ANSI X3.131-198X (SCSI-2).

E.1.18 Status

The contents of the SCSI status byte must conform to rules of the SCSI-2 standard. Otherwise, there are no restrictions on the use of this byte.

E.1.19 Unsupported Features

In addition to the restrictions mentioned above, the following features are not supported in the VMS operating system:

- Wide SCSI is not supported. Only the 8-bit data path is supported by the VMS operating system.
- Linked commands are not supported.
- Queued commands are not supported.
- The soft reset alternative is not supported. The device must implement the hard reset option.

If a device implements a feature such as queued commands, users can operate the device on a MicroVAX/VAXstation system, but will find that they are unable to invoke queued commands on the device.



Glossary

AEN: See *Asynchronous event notification*.

Asynchronous event notification (AEN): SCSI protocol allowing a SCSI device that is usually a target to inform the processor (usually the initiator) that an event has occurred asynchronously with respect to the processor's current stream of execution.

Class driver: See *SCSI class driver*.

Command descriptor block: Structure created by a SCSI class driver (or an application using the VMS generic SCSI class driver) in order to initiate a request of a device on the SCSI bus.

Connection: Logical link between a SCSI class driver and a device on the SCSI bus, involving the binding of the class driver to the VMS SCSI port driver. The connection allows the driver to issue commands to the SCSI device.

The class driver invokes the SPI\$CONNECT macro to perform this linkage. A connection lasts throughout the runtime life of a system; a SCSI class driver should never need to break a connection.

Device ID: See *SCSI device ID*.

Initiator: A SCSI device (usually the host processor) that requests another SCSI device (the target) to perform an operation.

Logical unit number (LUN): Unique value, from 0 to 7, that identifies a physical or virtual device accessible by means of a SCSI device with respect to that device's SCSI device ID.

LUN: See *Logical unit number*.

Port: See *SCSI port*.

Port driver: See *SCSI port driver*.

Port ID: See *SCSI port ID*.

SCDRP: See *SCSI class driver request packet*.

SCDT: See *SCSI connection descriptor table*.

SCSI: Refers to the American National Standard for Information Systems Small Computer System Interface-1 (X3.131-1986) or the ANSI Small Computer System Interface-2 (X3.131-1989). This standard defines mechanical, electrical, and functional requirements for attaching small computers to each other and to intelligent peripheral devices.

Glossary

SCSI class driver: Component of the VMS SCSI class/port architecture that acts as an interface between the user and the SCSI port, translating I/O functions as specified in a user's \$QIO request to a SCSI command targeted to a device on the SCSI bus. Although the class driver knows about SCSI command descriptor buffers, status codes, and data, it has no knowledge of underlying bus protocols or hardware, command transmission, bus phases, timing, or messages. A single class driver can run on any given MicroVAX/VAXstation system, in conjunction with the SCSI port driver that supports that system.

SCSI class driver request packet (SCDRP): A VMS data structure that contains information specific to an I/O request that a SCSI class driver must deliver to the port driver, such as the address of the SCSI command descriptor buffer. The class driver allocates the SCDRP, and places in it data it originally received in the I/O request packet (IRP), such as the \$QIO system service parameters, I/O function, and the length and location of any user-specified buffer involved in a transfer.

SCSI connection descriptor table (SCDT): A VMS data structure that contains information describing a connection established between a SCSI class driver and the port, such as phase records, timeout values, and error counters. The SCSI port driver creates an SCDT each time a SCSI class driver, by invoking the SPI\$CONNECT macro, connects to a device on the SCSI bus. The class driver stores the address of the SCDT in the SCSI device's UCB.

SCSI device ID: Unique value, from 0 to 7, representing a device on a specific SCSI bus. A VMS SCSI device ID corresponds to the line on the SCSI data bus on which a given device asserts itself and thus is an analog for the term SCSI ID.

Typically, a MicroVAX/VAXstation 3100 system processor is assigned device ID 6 and asserts itself at DB(6); a VAXstation 3520/3540 system processor is assigned device ID 7, and asserts itself at DB(7).

SCSI ID: See *SCSI device ID*.

SCSI port: The SCSI controller channel that controls communications to and from a specific SCSI bus in the system.

SCSI port descriptor table (SPDT): A VMS data structure that contains information specific to a SCSI port, such as the port driver connection database. The SPDT also includes a set of vectors, corresponding to the SPI macros invoked by SCSI class drivers, that point to service routines within the port driver. The SCSI port driver's unit initialization routine creates an SPDT for each SCSI port defined for a specific MicroVAX/VAXstation system and initializes each SPI vector.

SCSI port driver: Component of the VMS SCSI class/port architecture that transmits and receives SCSI commands and data. It knows the details of transmitting data from the local processor's SCSI port hardware across the SCSI bus. Although it understands SCSI bus phases, protocol, and timing, the SCSI port driver has no knowledge of which SCSI commands a given device supports, what status messages it returns, or the format of the packets in which this information is delivered. Strictly speaking, the port driver is a communications path. When directed by a SCSI class driver, the port driver forwards commands and data from the class driver onto the SCSI bus to the device. On any given MicroVAX/VAXstation system, a single SCSI port driver

handles bus-level communications for all SCSI class drivers that may exist on the system.

SCSI port ID: A unique representation of a SCSI port (see *SCSI port*) identifying the SCSI bus it controls. Current legal port IDs are *A* and *B*, corresponding to a VMS controller ID.

Small Computer System Interface: See *SCSI*.

SPDT: See *SCSI port descriptor table*.

Target: A SCSI device that performs an operation requested by an initiator.

Index

A

- AEN
 - See Asynchronous event notification
- Asynchronous event notification • 1–8, 3–27 to 3–29, B–4, B–20 to B–26
 - example • 3–28 to 3–29
- Asynchronous SCSI data transfer mode
 - enabling • 2–4, 2–11, 3–12, B–17
- Autoconfiguration
 - of SCSI device • 1–8, 2–6, 3–29

C

- Cancel-I/O routine
 - of SCSI third-party class driver • 3–27
- Class driver • 1–2, 3–3
- Command
 - See SCSI command
- Connection • 3–4, 3–8
 - breaking • B–7
 - obtaining characteristics of • B–8 to B–9
 - requesting • 3–25, B–4 to B–5
 - setting characteristics of • B–17 to B–18
- Connection characteristics buffer • B–17

D

- Data transfer
 - buffering mechanisms • 3–14
 - incomplete • 3–18
 - mapping local buffer for • 3–26
 - mapping local buffer for SCSI port • 3–15 to 3–16, B–10 to B–12
 - maximum size of • 3–13, 3–18, E–4
 - performing • 3–12 to 3–18
 - unmapping local buffer • 3–16, 3–27, B–19
- Data transfer mode • E–4
 - as controlled by a third-party SCSI class driver • 3–12, B–17
 - as controlled by the generic SCSI class driver • 2–4, 2–11

- Data transfer mode (cont'd.)
 - asynchronous • 2–4, 2–11, 3–12, B–17
 - determining setting of • B–8
 - synchronous • 2–4, 2–11, 3–12, B–17
- DDT\$W_ERRORBUF • 3–20
- Device
 - See SCSI device
- Disconnect feature
 - determining setting of • B–8
 - enabling • 2–5, 2–11, 3–13, B–17
- Disk class driver
 - disabling the loading of • 2–7, 3–30
 - using for non-Digital-supplied device • 1–8
- DPT\$V_NO_IDB_DISPATCH • 3–24
- Driver prologue table (DPT)
 - of third-party SCSI class driver • 3–24

E

- Error log entry
 - examining the contents of • D–1 to D–10
- Error logging routine
 - in SCSI third-party class driver • 3–19 to 3–21
- Error message buffer
 - of third-party SCSI device driver • 3–19 to 3–20

G

- Generic SCSI class driver • 2–1 to 2–14
 - assigning a channel to • 2–8
 - compared to SCSI third-party class driver • 1–3
 - flow of • 2–2
 - I/O status block returned by • 2–9
 - loading • 2–7
 - obtaining device information from • 2–12
 - programming example • 2–13 to 2–14
 - \$QIO system service format for • 2–8 to 2–12
 - security considerations • 2–2
- Generic SCSI descriptor
 - format of • 2–10 to 2–12

Index

I

- I/O request
 - as serviced by SCSI class and port drivers • 3-21 to 3-23
- I/O status block
 - returned by generic SCSI class driver • 2-9
- Initiator • 1-7
 - completing an operation (in AEN mode) • B-21
 - enabling selection of • 3-27 to 3-29, B-4, B-20 to B-26
 - operation of • E-5
 - receiving data from target (in AEN mode) • B-22
 - sending bytes to target (in AEN mode) • B-24

L

- LUN (logical unit number) • 1-7, E-5

M

- MicroVAX/VAXstation 3100 systems
 - support for SCSI devices • 1-4

N

- NCR 5380 controller • 1-4
- Non-Digital-supplied SCSI class driver
 - See Third-party SCSI class driver

P

- Port • 1-6
 - DMA buffer • 1-8, 3-15, 3-26, B-10 to B-12
 - examining status of • 3-16 to 3-17
 - resetting • B-13
- Port capabilities longword • 3-12
- Port command buffer
 - allocating • 3-10, 3-26, B-3
 - deallocating • 3-10, 3-27, B-6
- Port driver • 1-1, 3-2

R

- Register dumping routine
 - of SCSI third-party class driver • 3-20, 3-27
- REQCOM macro • 3-27
- Request sense key • 3-17

S

- SCDRP\$L_ABCNT • 3-14
- SCDRP\$L_BCNT • 3-14, 3-18, B-11, B-15
- SCDRP\$L_CMD_PTR • 3-10, B-15
- SCDRP\$L_DISCON_TIMEOUT • 3-10, 3-11
- SCDRP\$L_DMA_TIMEOUT • 3-10, 3-11
- SCDRP\$L_IRP • 3-26
- SCDRP\$L_MEDIA • 3-14
- SCDRP\$L_PAD_COUNT • 3-14
- SCDRP\$L SCSI_FLAGS • 3-14, 3-15, 3-26, B-11
- SCDRP\$L_SPTESVAPTE • 3-15
- SCDRP\$L_STS_PTR • 3-10, 3-17, B-15, B-16
- SCDRP\$L_SVAPTE • 3-14, B-11
- SCDRP\$L_SVA_SPTESVAPTE • B-12
- SCDRP\$L_SVA_USER • 3-14, 3-15, B-12, B-15
- SCDRP\$L_TRANS_CNT • 3-18, B-16
- SCDRP\$V_BUFFER_MAPPED • 3-15, 3-26
- SCDRP\$V_S0BUF • 3-15, 3-26
- SCDRP\$W_BOFF • 3-14, B-11
- SCDRP\$W_FUNC • 3-14, B-15
- SCDRP\$W_MAPREG • 3-16, B-12
- SCDRP\$W_NUMREG • 3-15, B-12
- SCDRP\$W_STS • 3-14, 3-15, B-11
- SCDRP (SCSI class driver request packet) • 3-6,
A-1 to A-9
 - allocating • 3-26
 - deallocating • 3-27
 - defining fields of • 3-23
 - initializing • 3-14 to 3-15, 3-26
- \$SCDRPDEF macro • 3-23
- SCDT (SCSI connection descriptor table) • 3-6, A-9 to A-15
- SCSI (Small Computer System Interface)
 - definition • 1-1
- SCSI bus
 - phases of • E-6
 - releasing in AEN operation • B-23
 - resetting • B-13
 - sensing phase of • B-25
 - setting phase of • B-26

- SCSI bus (cont'd.)
 - termination requirements • E-2
- SCSI bus analyzer • 3-31
- SCSI class driver
 - See Class driver, Disk class driver, Generic SCSI class driver, Tape class driver, Template class driver, Third-party SCSI class driver
- SCSI class driver request packet
 - See SCDRP
- SCSI class/port architecture • 1-1, 3-1 to 3-4
 - summary of I/O request servicing • 3-21 to 3-23
- SCSI command
 - controlling the number of retries • 3-11
 - determining timeout setting for • B-9
 - disabling retry • 2-5, 3-11, B-8, B-17
 - enabling retry • 2-11, B-8
 - examining status of • 3-16 to 3-18, 3-26
 - padding, when required • 2-12
 - preparing to issue • 3-9 to 3-12
 - sending to SCSI device • 3-10, B-14 to B-16
 - setting disconnect timeout for • 2-6, 2-12, 3-10, 3-11, B-9, B-18
 - setting DMA timeout for • 2-6, 2-12, 3-10, 3-11, B-9, B-18
 - setting phase change timeout for • 2-6, 2-12, 3-10, 3-11, B-9, B-18
 - size of • 3-10
 - terminating • 3-27, B-2
- SCSI command byte
 - buffering • 3-10, 3-26, B-3
- SCSI command descriptor block
 - creating • 3-10
 - initializing pointer to • 3-10
- SCSI connection descriptor table
 - See SCDT
- SCSI controller
 - NCR 5380 • 1-4
 - SII • 1-5
- SCSI device
 - connecting to • 3-8
- SCSI device ID • 1-7
- SCSI device UCB • 3-7
 - extending • 3-24
- SCSI ID • 1-7, E-5
- SCSI messages
 - as implemented in VMS • E-7 to E-8, E-8
- SCSI port descriptor table
 - See SPDT
- SCSI port driver
 - See Port driver
- SCSI port ID • 1-6
- SCSI port interface
 - See SPI
- SCSI port UCB • 3-7
- SCSI status byte
 - examining • 3-17
 - initializing • 3-10
 - servicing CHECK CONDITION status • 3-17
- SII controller • 1-5
- Small Computer System Interface
 - See SCSI
- SPDT (SCSI port descriptor table) • 3-6, A-15 to A-21
 - creation of • 3-25
- SPI\$ABORT_COMMAND macro • 3-5, 3-27, B-2
- SPI\$ALLOCATE_COMMAND_BUFFER macro • 3-5, 3-10, 3-26, B-3
- SPI\$CONNECT macro • 3-5, 3-9, 3-25, 3-28, B-4 to B-5
- SPI\$DEALLOCATE_COMMAND_BUFFER macro • 3-5, 3-10, 3-27, B-6
- SPI\$DISCONNECT macro • 3-5, B-7
- SPI\$FINISH_COMMAND macro • 3-28, B-21
- SPI\$GET_CONNECTION_CHAR macro • 3-5, B-8 to B-9, B-17
- SPI\$MAP_BUFFER macro • 3-5, 3-15 to 3-16, 3-26, B-10 to B-12
- SPI\$RECEIVE_BYTES macro • 3-28, B-22
- SPI\$RELEASE_BUS macro • 3-28, B-23
- SPI\$RESET macro • 3-5
- SPI\$SEND_BYTES macro • 3-28, B-24
- SPI\$SEND_COMMAND macro • 3-5, 3-10, 3-16, 3-26, B-14 to B-16
- SPI\$SENSE_PHASE macro • 3-28, B-25
- SPI\$SET_CONNECTION_CHAR macro • 3-5, 3-11, 3-12, 3-13, 3-26, B-17 to B-18
- SPI\$SET_PHASE macro • 3-28, B-26
- SPI\$UNMAP_BUFFER macro • 3-5, 3-16, B-19
- SPI (SCSI port interface) • 3-4 to 3-5, B-1 to B-26
 - calling protocol for • 3-5, B-1
 - extensions to • 3-28 to 3-29, B-20 to B-26
- Start-I/O routine
 - of third-party SCSI class driver • 3-26 to 3-27
- Status
 - See SCSI command, Port, SCSI status byte
- Synchronous SCSI data transfer mode
 - determining REQ-ACK offset setting • B-8
 - determining transfer period setting • B-8
 - enabling • 2-4, 2-11, 3-12, B-17
 - setting REQ-ACK offset • 3-12, B-17
 - setting transfer period • 3-12, B-17

Index

SYS\$GETDVI

SCSI generic class driver • 2-12

SYS\$QIO

format for request to SCSI generic class driver • 2-8

System Generation Utility (SYSGEN)

configuring SCSI devices • 1-8, 2-6, 3-29

T

Tape class driver

disabling the loading of • 2-7, 3-30
using for non-Digital-supplied device • 1-8

Target • 1-7

enabling selection from • 3-27 to 3-29, B-4, B-20 to B-26
operation of • E-5

Target mode

See Asynchronous event notification

Template class driver • 3-8

listing of • C-1 to C-35

Third-party SCSI class driver

cancel-I/O routine of • 3-27
compared to SCSI generic class driver • 1-3
components • 3-23 to 3-27
data definitions • 3-23
debugging • 3-30 to 3-32
driver prologue table • 3-24
error logging • 3-19 to 3-21
loading • 3-29
maintaining local context of • 3-18 to 3-19
receiving notification of asynchronous events on target • 3-27 to 3-29, B-4, B-20 to B-26
register dumping routine of • 3-20, 3-27
start-I/O routine of • 3-26 to 3-27
unit initialization routine of • 3-25 to 3-26
writing • 3-1 to 3-32

Third-party SCSI device

bus-reset behavior • E-3
cabling requirements • E-2
connector requirements • E-2
dynamic reconfiguration of • E-2
external box restrictions • E-3
hardware requirements and restrictions • E-1 to E-9
power-on behavior • E-3
rules for disconnect and reselection • E-6
SCSI command requirements • E-8
SCSI message requirements • E-7 to E-8
SCSI status requirements • E-9

Third-party SCSI device (cont'd.)

summary of VMS support mechanisms • 1-3 to 1-4

terminator power requirement • E-2
using VMS disk or tape class driver • 1-8

Timeout

for SCSI device • 2-6, 2-12, 3-10, B-18

U

UCB\$B_DEVCLASS • 3-20, 3-24

UCB\$B_DEVTYPE • 3-20, 3-24

UCB\$L_MAXBCNT • 3-13, 3-25

UCB\$L_PDT • 3-25

UCB\$L_SCDT • 3-25

UCB\$V_POWER • 3-25

UCB\$W_STS • 3-25

UCB (unit control block)

See SCSI device UCB, SCSI port UCB

Unit control block (UCB)

See SCSI device UCB, SCSI port UCB

Unit initialization routine

of third-party SCSI class driver • 3-25 to 3-26

V

VAXstation 3520/3540 system

support for SCSI devices • 1-5

VM\$D2 system parameter • 2-7, 3-30

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	_____	Local Digital subsidiary or approved distributor
Internal ¹	_____	SSB Order Processing - WMO/E15 <i>or</i> Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

¹For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

VMS Version 5.3 Small
Computer System Interface
(SCSI) Device Support Manual
AA-PAJ2A-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Completeness (enough information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity (easy to understand)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization (structure of subject matter)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Figures (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Examples (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Index (ability to find topic)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Page layout (easy to find information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

I would like to see more/less _____

What I like best about this manual is _____

What I like least about this manual is _____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

I am using **Version** _____ of the software this manual describes.

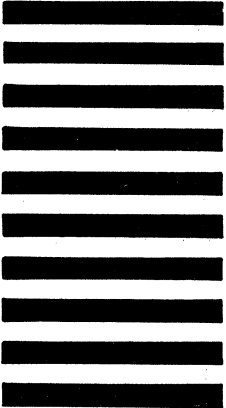
Name/Title _____ Dept. _____
Company _____ Date _____
Mailing Address _____
_____ Phone _____

--- Do Not Tear - Fold Here and Tape ---

digitalTM



No Postage
Necessary
if Mailed
in the
United States



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987



--- Do Not Tear - Fold Here ---

Cut Along Dotted Line

Reader's Comments

VMS Version 5.3 Small
Computer System Interface
(SCSI) Device Support Manual
AA-PAJ2A-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Completeness (enough information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity (easy to understand)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization (structure of subject matter)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Figures (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Examples (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Index (ability to find topic)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Page layout (easy to find information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

I would like to see more/less _____

What I like best about this manual is _____

What I like least about this manual is _____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

I am using **Version** _____ of the software this manual describes.

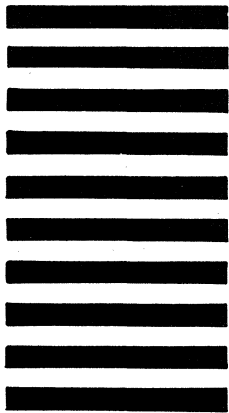
Name/Title _____ Dept. _____
Company _____ Date _____
Mailing Address _____
_____ Phone _____

-- Do Not Tear - Fold Here and Tape

digitalTM



No Postage
Necessary
if Mailed
in the
United States



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987



-- Do Not Tear - Fold Here

Cut Along Dotted Line

digital