1

OpenVMS V7.1 I/O New Features

September 6, 1996

1.1 High SCSI IDs on Wide Adapters (Alpha Only)

OpenVMS Alpha Version 7.1 device driver support of high SCSI IDs allows host adapters that support the wide SCSI bus to configure and use up to 16 devices per SCSI bus.

KZTSA, KZPSA, and the QLogic ISP1020 wide adapters currently support high SCSI IDs.

Note that although the maximum allowable number of devices per SCSI bus has changed, the SCSI bus length requirements have not. When configuring devices above the previous limit of 8 devices, failure to adhere to bus length requirements can produce unpredictable results.

Supported SCSI bus lengths are as follows:

- 3 meters (10 feet) on fast single-ended buses
- 6 meters (20 feet) on slow single-ended buses
- 25 meters on differential buses

Narrow devices do not talk to hosts above ID 7.

To configure wide IDs on a BA356 box, refer to the BA356 manual *StorageWorks Solutions BA356-SB 16-Bit Shelf User's Guide* (Order Number EK-BA356-UG).

When a narrow device is configured on a wide bus, the device reports itself in the lower 8 IDs appropriate to the slot in the box where it resides. For instance, in the following example of wide and narrow disks in a BA356 box with a starting address of 8, disks 800, 900, 1000 and 1200 are wide disks. Disk 300 is a narrow device located in Slot 3 of the BA350 box.

```
Slot 0 $1$DKA800:
Slot 1 $1$DKA900:
Slot 2 $1$DKA1000:
Slot 3 $1$DKA300:
Slot 4 $1$DKA1200:
```

SCSI IDs above 700 are displayed as `Invalid SCSI IDs` by the DIAGNOSE error logging utility.

1.2 OpenVMS SCSI Device Drivers: Enhanced for OpenVMS Version 7.1

OpenVMS SCSI device drivers have been enhanced significantly for OpenVMS Version 7.1. These enhancements allow OpenVMS SCSI device drivers to work with a wider range of non-Digital SCSI devices, while taking advantage of as much SCSI functionality as the devices offer. In most cases, the OpenVMS Version 7.1 SCSI driver enhancements eliminate the errors that occurred when some non-Digital devices were mounted after booting Version 6.2 of the OpenVMS operating system.

1.3 Device Support Documentation Changes

OpenVMS device support documentation has been enhanced and reorganized for OpenVMS Version 7.1. Two books about writing OpenVMS Alpha device drivers are now available, and several device support books from previous releases have been archived.

The main advantage of the Alpha device driver book changes for OpenVMS Version 7.1 is that two Alpha driver books are available, one for each type of driver-writer:

- *Writing OpenVMS Alpha Device Drivers in C*, for customers writing new Alpha drivers.
- *Creating an OpenVMS Alpha Device Driver from an OpenVMS VAX Device Driver*, for customers porting or maintaining drivers in Macro-32.

Table 1–1 summarizes the Alpha and VAX device support book changes for OpenVMS Version 7.1. It lists the device support books that have been archived, and indicates whether they have been replaced.

Table 1–1 OpenVMS Version 7.1 Device Support Books

Title	Replacement
Creating an OpenVMS AXP Step 2 Device Driver from a Step 1 Device Driver	None
Creating an OpenVMS AXP Step 2 Device Driver from an OpenVMS VAX Device Driver	*Creating an OpenVMS Alpha Device Driver from an OpenVMS VAX Device Driver* (This book now includes reference information.)
OpenVMS Alpha Device Support: Developer's Guide	*Writing OpenVMS Alpha Device Drivers in C*[1]
OpenVMS Alpha Device Support: Reference	C reference information is available in *Writing OpenVMS Alpha Device Drivers in C*. Macro-32 reference information is included in the *Creating an OpenVMS Alpha Device Driver from an OpenVMS VAX Device Driver* book.
OpenVMS VAX Device Support Manual	None
OpenVMS VAX Device Support Reference Manual	None

[1]This book is not available for OpenVMS 7.1–EFT2. It will be included in the hardcopy docset for the final version on OpenVMS 7.1.

2

Automatically Configuring Devices for OpenVMS Alpha

September 6, 1996

File-based autoconfiguration is a new method for configuring OpenVMS Alpha devices. This functionality provides a more consistent and less complex way to configure devices automatically than the methods available in previous versions of the OpenVMS Alpha operating system.

This chapter briefly explains device configuration and describes how to use file-based autoconfiguration.

2.1 Overview of Device Configuration

A device is configured on OpenVMS when system code is able to locate it on a bus, give it a name, and load a device driver for it. When a device is autoconfigured, all these steps happen without any user intervention.

Autoconfiguration is the process of discovering the hardware devices on a system and loading the appropriate device drivers. Devices on OpenVMS are discovered during the boot process in a bus-specific manner. Each device that is found has information about the device stored in a set of tables for each bus. Later, the hardware information is used to search a set of configuration tables of known devices. The configuration table provides the information necessary to determine the driver name, device name, and other parameters for loading and connecting the appropriate driver.

Prior to OpenVMS Alpha Version 7.1, configuration tables were built into the OpenVMS kernel and could not be modified without replacing a system execlet image. As of OpenVMS Alpha Version 7.1, the configuration tables are constructed from ASCII text files on the system disk. A system file (**SYS$SYSTEM:SYS$CONFIG.DAT**) is provided for all OpenVMS supported devices, and a user file (**SYS$SYSTEM:SYS$USER_CONFIG.DAT**) is provided for all third party, layered product, and user-written device drivers. These files are read by the system during system boot, combined, and used for subsequent autoconfiguration of hardware devices.

Section 2.2 describes how to use the SYS$SYSTEM:SYS$USER_CONFIG.DAT file to autoconfigure devices.

2.2 Using File-Based Autoconfiguration

File-based autoconfiguration reads two files during system boot to build the configuration tables of known devices:

SYS$SYSTEM:SYS$USER_CONFIG.DAT
SYS$SYSTEM:SYS$CONFIG.DAT

Both files use the same format, and the data from both files are combined to create the configuration tables for each bus on the system. The SYS$USER_CONFIG.DAT file is read first, and takes precidence over any duplicate device descriptions contained in both files. If multiple device descriptions exist in a single file, the first occurance of the description is used (the autoconfiguration logic does not search past the first occurance of a device in the configuration table).

Note

The SYS$CONFIG.DAT file is read-only and should NOT be modified by the user or third parties, it should only be modified by OpenVMS, and may be replaced by OpenVMS upgrades. Inappropriate edits to this file could result in the inability to boot the system.

The SYS$USER_CONFIG.DAT file is an ASCII text file that can be edited by any text editor, or appended to by DCL, or the APPEND command. The following section describes the syntax of the file.

The configuration files consist of of device definition blocks, each of which provides the information needed to configure the correct device driver for the device. Blank lines, multiple spaces, and tabs may be used for clarity, but they are ingnored by the system.

A device description consists of a series of commands starting with a DEVICE command, and ending with the END_DEVICE command. Between these two commands, additional commands define the hardware ID, the device name, the driver name, the bus type, and any other required or optional information.

Commands take the general form:

```
COMMAND = value
```

Where the value is a string, a quoted string, or a numeric value. The END_DEVICE command has no associated value.

Comments can occur anywhere in the file, and begin with the exclamation point (!). All text on the line following the exclamation point is discarded.

Each device definition in the new configuration file begins with the DEVICE command, includes the commands and parameters that describe the device, and ends with the END_DEVICE command. For example, a minimal description of a non-disk device might look like the following:

```
DEVICE          = "My device"
 NAME           = UU
 DRIVER         = USER$UUDRIVER
 ID             = 0x0005111
 ADAPTER        = PCI
 FLAGS          = NOVECTOR
END_DEVICE
```

Device-specific information is included within a DEVICE/END_DEVICE pair. In this example, it indicates that when a device with the hardware ID of 5111 (hex) is found on a PCI bus, the device named UU should be configured, and the USER$UUDRIVER device driver should be loaded. This example also specifies that the driver does not want to be connected to an interrupt vector. (If the bus information had a vector, it would be ignored.)

In addition to the values shown in the previous example, the following implied values can also be specified:

```
UNITS           = 1
NUM_VECTORS     = 1
ALIGN_VECTOR    = 0
```

These values usually default to the correct values for a single unit controller.

2.3 Configuration File Syntax

A minimal device description consists of a DEVICE command, a NAME command, a DRIVER command, an ADAPTER command, and a END_DEVICE command.

The following syntax rules apply.

- All strings are converted to uppercase unless within quotes.
- The quotation mark cannot be passed within a string (it will be stripped).
- All numbers are in the C language format: the default is decimal, octal may be specified by providing a leading "0", hex by providing a leading "0x".
- All tabs are compressed into spaces.
- The exclamation point may be specified to start a comment, and the rest of the line following the exclamation point is ignored.

The following commands are defined:

- **DEVICE** = string

  The DEVICE command starts a device definition. The device command takes a string argument which is an arbitrary description of the device being defined.

  Example: DEVICE = "NCR 53C810 SCSI port driver"

- **END_DEVICE**

  The END_DEVICE terminates the current device definition, and causes the parser to create a new table entry if all required keywords have been supplied. No parameters are required.

- **ID** = {number {, number} | string}

  The ID command specifies the hardware ID of the device. The hardware ID is a 64-bit quanity and can be specified by a pair of 32-bit values (low-order, high-order), or can be specified as a ASCII string of up to 8 characters. If only one numeric value is provided, the high-order 32-bits are set to zero.

  Example: ID = 0x00051000 ! HW ID of NCR810 in hex

  Example: ID = FLOPPY ! HW ID of Floppy in ASCII

  Example: ID = 0x31323334, 0x20202020 ! 64-bit ID => low-32, high-32

- **NAME** = string

  The NAME command specifies the mnemonic for the device. This is usually a two character name, but it cam be more.

  Example: NAME = PK

- **DRIVER** = string

  The DRIVER command specifies the filename of the device driver for the device.

Example: DRIVER = SYS$PKEDRIVER

- **ADAPTER** = string

  The ADAPTER command indicates the adapter type that the device is valid on. The current adapters are: *AT$_PCI, AT$_EISA, AT$_ISA, AT$_TC, and NONE.*

  Example: ADAPTER = AT$_PCI

- **UNITS** = number

  The UNITS command indicates the number of units for the controller. The default if not specified is 1.

  Example: UNITS = 1

- **FLAGS** = string

  The FLAGS command tells configuration software the type of driver, and optional information on how to load the driver. More than one flag may be specified, separated by the "," character.

  Example: FLAGS = NOVECTOR, CASE_BLIND

Table 2–1 Device Flags

Flag	Description
NOVECTOR	Device has no interrupt vector
CASE_BLIND	ID check is case blind ASCII
EXTENDED_ID	Use 64-bits of hardware ID to perform match with the configuration table entry and to fill out the upper part of the hardware ID.

- **PRIVATE_DATA** = CLASS=1, SUBCLASS = Oxff

  An arbitrary string intended for use by the type-specific configuration.

  All text starting on the line following the BEGIN_PRIVATE to the line preceding the END_PRIVATE will be copied to the private data buffer.

  No formatting will be done on any of the data. It will be copied as-is. Each line will be seperated by a <CR><LF> pair. Any 8-bit character can be used except for NULL. The text will be terminated by a single NULL.

  The consumer of the private data is responsible for all parsing of the text. The private data should be considered READ-ONLY by consumers.

- **NUM_VECTORS** = number

  The NUM_VECTORS command specifies the number of vectors that the device uses. The default if not specified is 1.

  Example: NUM_VECTORS = 4

- **ALIGN_VECTOR** = number

  The ALIGN_VECTOR command specifies the vector alignment. This is used only on the FUTUREBUS and XMI, where the value will typcially be 64 when the value is not zero.

  Example: ALIGN_VECTOR = 64

Table 2–2 indicates which must be included in the configuration file.

Table 2–2 Command Summary

Command	Required	Description
DEVICE	Yes	Begins a device definition
END_DEVICE	Yes	Ends a device definition
ID	Yes	Specifies the hardware ID.
NAME	Yes	Device Name
DRIVER	Yes	Driver name
ADAPTER	Yes	Adapter type
UNITS	No	Max Units *Default: 1*
FLAGS	No	Device Flags
NUM_VECTORS	No	Number of Vectors *Default: 1*
ALIGN_VECTOR	No	Vector Alignment (used on XMI and FBUS) *Default: 0*

2.4 Supported Buses for User Devices

File-based autoconfiguration is supported for user-written device drivers on PCI, ISA, EISA, and TURBOchannel buses. This section includes additional information apecific to ISA and EISA configurations.

2.4.1 ISA Configuration

ISA devices do not provide a standard device ID that can be discovered during bus probing. Thus the user must explicitly configure (indicate the presence of the device), and reserve any resources such as IRQ's needed by the device. ISA devices can be present on both ISA buses, as well as EISA buses, and each must be configured using the bus-specific configuration method provided by the console. Once the device is known to the console, OpenVMS can then autoconfigure it using file-based autoconfiguration.

2.4.1.1 Configuring ISA Devices on the ISA Bus

In previous versions of OpenVMS Alpha, ISA devices on a ISA bus required creating an entry in the file SYS$MANAGER:ISA_CONFIG.DAT to define the hardware, and the console command ISACFG to reserve system resources like IRQ's.

The device can be autoconfigured using the File-Based Autoconfiguration by creating a entry in the SYS$USER_CONFIG.DAT file using the HANDLE from the ISACFG command as the device ID. The HANDLE should be limited to 8 characters. Since ISACFG stores the string without upcasing, the ID should either be specified as a quoted string in the correct case, or the flag CASE_ BLIND should be provided (preferred) to allow the ID match to do a case blind comparison.

WARNING

As of OpenVMS Alpha Version 7.2, the ISA_CONFIG.DAT file will be removed and will no longer be supported. Refer to the *OpenVMS Version 7.1 Release Notes* for more information.

2.4.1.2 Configuring ISA devices on the EISA bus

ISA devices must be manually configured using the EISA Configuration Utility (ECU) which is run from the console. Devices must have a CFG file provided on a DOS floppy. The CFG file provides a string (up to 7 characters) as the device ID.

See your card manufacturer, or the EISA Bus Specification for details on CFG file format.

The ECU floppy (DOS format) contains example ISA CFG files that may be used as an example.

Note from Fred

It may be a good idea to provide a example here.

Once the ECU has been run, the device can be configured using file-based autoconfiguration.

Note

ISA devices cannot be easily autoconfigured on EISA bus systems on versions prior to OpenVMS Alpha V7.1-EFT2 because the ID is not copied from the ECU data into the OpenVMS bus structures.

2.4.2 EISA Configuration

EISA devices are probed during booting, and the device ID from the hardware ID register is stored by OpenVMS. This ID is a compressed ASCII string. OpenVMS does not decompress this string, and the ID that is encoded in the file-based autoconfiguration file should be the value that is read from the hardware ID register, and not an ASCII encoding of the value.

The ID can be obtained from the hardware documentation for the option.

ISA devices that do not have a EISA hardware ID must be manually configured using the ECU (see 5.5.2??) and will use the ASCII string specified in the CFG file.

In any case, the ECU must be run on EISA systems to properly configure the EISA bus resources.

3
Device Support on OpenVMS Systems

September 6, 1996

This chapter contains release notes pertaining to OpenVMS device support on Alpha and VAX systems. For each note, the heading indicates whether it contains Alpha-specific or VAX-specific information.

3.1 Recompiling and Relinking OpenVMS Device Drivers

The following sections contain release notes pertaining to recompiling and relinking OpenVMS device drivers.

3.1.1 OpenVMS Alpha SCSI Device Drivers Must Be Recompiled and Relinked for Version 7.1–EFT1

V7.1–EFT1

OpenVMS Alpha Version 7.1–EFT1 includes changes to the data structures used by OpenVMS Alpha SCSI device drivers.

As a result of these changes, all OpenVMS Alpha SCSI device drivers from OpenVMS Version 7.0 must be recompiled and relinked to run correctly on OpenVMS Alpha Version 7.1–EFT1. No source changes are required.

Version 7.1–EFT2 Note

OpenVMS Alpha device drivers that were recompiled and relinked for OpenVMS Alpha Version 7.1–EFT1 do not have to be recompiled and relinked for OpenVMS Alpha Version 7.1–EFT2.

3.1.2 OpenVMS VAX SCSI Device Drivers Must Be Recompiled and Relinked for Version 7.1

V7.1–EFT1

OpenVMS Version 7.1–EFT1 includes changes to the data structures used by OpenVMS VAX SCSI device drivers.

As a result of these changes, all OpenVMS VAX SCSI device drivers from previous releases of OpenVMS VAX must be recompiled and relinked to run correctly on OpenVMS Version 7.1–EFT1. No source changes are required.

Version 7.1–EFT2 Note

OpenVMS VAX device drivers that were recompiled and relinked for OpenVMS Alpha Version 7.1–EFT1 do not have to be recompiled and relinked for OpenVMS Alpha Version 7.1–EFT2.

3.1.3 OpenVMS Alpha Device Drivers Must Be Recompiled and Relinked for Version 7.0

V7.0

OpenVMS Alpha Version 7.0 includes significant changes to OpenVMS Alpha privileged interfaces and data structures. As a result of these changes, all device drivers from previous versions of OpenVMS Alpha must be recompiled and relinked to run correctly on OpenVMS Alpha Version 7.0.

For more details about OpenVMS Alpha Version 7.0 changes that may require source changes to customer-written drivers, see the *OpenVMS Alpha Guide to Upgrading Privileged-Code Applications*.

V7.1–EFT1

Version 7.1–EFT1 Note

OpenVMS Alpha device drivers that were recompiled and relinked for OpenVMS Alpha Version 7.0 do not have to be recompiled and relinked to run on OpenVMS Alpha Version 7.1–EFT1. However, OpenVMS Alpha SCSI device drivers from OpenVMS Version 7.0 must be recompiled and relinked as described in Section 3.1.1.

3.2 ICBMs Must be Recompiled and Relinked for OpenVMS Alpha Version 7.1–EFT1

V7.1–EFT1

Customer-written ICBMs (used to autoconfigure device drivers) from previous releases must be recompiled and relinked to run on OpenVMS Version 7.1–EFT1.

Recompiling and relinking ICBMs is necessary for this release because of additions to the bus array entry structure.

Version 7.1–EFT2 Note

OpenVMS Alpha ICBMs that were recompiled and relinked for OpenVMS Alpha Version 7.1–EFT1 do not have to be recompiled and relinked for OpenVMS Alpha Version 7.1–EFT2.

3.3 Disabling Tagged Command Queuing with the SCSI_MODE Utility

V7.1–EFT2

REVIEWERS: I am still working on this note.

3.4 SYS$MANAGER:ISA_CONFIG.DAT Unsupported in Future Releases of OpenVMS Alpha

V7.1–EFT2

Support for using the SYS$MANAGER:ISA_CONFIG.DAT file to configure ISA devices will be discontinued in a future release of OpenVMS Alpha. If you use this file, you should convert to using the ISACFG utility from the console, and the new file-based autoconfiguration method for loading device drivers (as described in the *OpenVMS Version 7.1–EFT2 New Features Manual*).

Table 3–1 contains a list of keywords from ISA_CONFIG.DAT and their equivalents in either file-based autoconfiguration or the ISACFG utility.

Table 3–1 ISA_CONFIG.DAT Keywords and Equivalents

ISA_ CONFIG.DAT	File-based Autoconfigure	ISACFG
NAME	NAME	
DRIVER	DRIVER	
IRQ		irq*x*
NODE		slot
DMA		dmachan*x*
PORT		iobase*x*
MEM		membase*x*
FLAGS	bit 1	unsupported
	bit 2	FLAG=NOVECTOR
USER_PARAM	PRIVATE_DATA	

Descriptions for each parameter in ISA_CONFIG.DAT are as follows:

NAME = xx
Use the same parameter for file-based autoconfiguration. Use the same value, where xx is the device code. (The device code is usually 2 letters.)

Example: NAME = ER

DRIVER = driver_name
Use the same parameter for file-based autoconfiguration. Use the same value for file based autoconfiguration. yy is the name of the driver in SYS$LOADABLE_ IMAGES.

Example: DRIVER = SYS$ERDRIVER

IRQ = i
Use irqx (italicize the x) from the ISACFG utility. You can express four irqs: -irq0 thru -irq3. Use the same value as you used for ISA_CONFIG.DAT. The IRQ is a value from 0 to F which specifies which ISA IRQ the device uses to report interrupts.

Examples:

This example assigns IRQ 6 to the device.

```
>>>isacfg -slot 3 -dev 0 -mk -handle MYDEV -enadev 1 -etyp 1 -handle MYDEV -irq0 6
```

This example assigns IRQs 6 and 12 (c) to the device.

```
>>>isacfg -slot 3 -dev 0 -mk -handle MYDEV -enadev 1 -etyp 1 -handle MYDEV -irq0 6 -irq1 c
```

NODE = n

Use -slot from the ISACFG utility. The slot number (n) does not represent the slot in which the device resides. It is a logical, not a physical number. However, the number does have to be between 1 and the maximum number of slots on the machine. Slot number 0 is not available to users.

Example:

```
>>>isacfg -slot 3 -dev 0 -mk -enadev 1 -etyp 1 -handle MYDEV -dmachan0 1 -irq0 6
```

This example assigned values to a device represented by slot 3. There must be at least 3 slots on the machine on which this command is executed. To see which logical slots are not being used, enter the following command:

```
>>>isacfg -all
```

DMA = (j,k, ...)

Use -dmachan*x* from the ISACFG utility. Use the same values as you used for ISA_CONFIG.DAT, but with different versions of -dmachan*x* (italic the x). j,k etc. are values zero through 7 which specify the channels of the DMA controller which the device is using to relay information. You can specify four DMA channels, using the keywords -dmachan0 thru -dmachan3.

Example:

```
>>>isacfg -slot 3 -dev 0 -mk -enadev 1 -etyp 1 -handle MYDEV -irq0 6 -dmachan0 1 -dmachan1 3
```

This example assigned two dma channels, 1 and 3, to the device.

PORT = (aa:b, cc:d, ...)

Use -iobase*x* from the ISACFG utility. You can specify six ports using the keywords -iobase0 through -iobase5. Each iobase*x* represents one byte. There is no equivalent for b, d etc.

Example:

```
>>>isacfg -slot 1 -dev 0 -mk -enadev 1 -etyp 1 -handle AAA321 -irq0 8 -iobase 2F8
```

This example assigned port 2F8 to the device.

MEM = (ee:f, gg:h, ...)

Use the ISACFG keywords -membase*x*, to specify the memory base, and -memlen*x*, to specify the memory length. Use the same values for ee, gg etc. and f, h etc. as you used for ISA_CONFIG.DAT. You can specify three memory regions using the keywords membase0 through membase2 and memlen0 through memlen2.

Example:

```
>>>isacfg -slot 3 -dev 0 -mk -enadev 1 -etyp 1 -handle MYDEV -irq0 6 -membase0 80000 -memlen0 20
```

FLAGS = n

The FLAGS field in ISA_CONFIG.DAT had 2 meaningful bits:

Bit 0 is the device being configured is a SCSI adapter.

Bit 1 indicates that no interrupt is required for the device.

Because it was never possible to use bit 0, it is not currently supported in file-based autoconfiguration. Bit 1 can be expressed with the file-base autoconfigure command FLAGS=NOVECTOR.

USER_PARAM = text
Use the PRIVATE_DATA command in file-based autoconfiguration to represent this value. [MARGIE - see me about how drivers get this value. We're working on this now.]

There is one other ISACFG command which must be specified correctly. The **handle** keyword specifies the device name, and is used as the hardware ID. It can contain any string the user wants to use to identify the device. However, the same string must be used in the autoconfiguration file with the keyword ID.

For example:

From the console:

```
>>>isacfg -slot 1 -dev 0 mod -handle MYDEVICE
```

In SYS$SYSTEM:SYS$USER_CONFIG.DAT:

```
ID = MYDEVICE
```

While using ISACFG you must also be familiar with the following commands:

To return the configuration to its initial state: `>>>isacfg -init`

To save your changes:

```
>>>init
```

To delete an entry:

```
>>>isacfg -slot 1 -dev 0 -rm
```

To see all the devices currently configured:

```
>>>isacfg -all
```

To modify a device, use -mod

```
>>>isacfg -slot 2 -dev 0 -mod  (etc.)
```

The following keywords do not have equivalents in ISA_CONFIG.DAT:

`-enadev a_number` This keyword takes the numbers 0 (disabled) and 1 (enabled). It allows you to disable a device so that it will not be used in resource allocation calculations.

`-etyp a_number` This keyword defines an entry type for this entry. VMS only supports the values 0 and 1. It should always be specified as a 1. It takes the following values:

```
0   -   Causes the entry to be deleted
1   -   Single option
2   -   Embedded multiport device
3   -   Multiport option device
```

3.5 Required Change for ISA_CONFIG.DAT on AlphaStation 200/400 Family Systems

V7.1–EFT2

Customers configuring ISA devices on AlphaStation 200/400 Family systems must change their ISA_CONFIG.DAT file, so that the node information for each device appears at the end of each device description block.

For upgrades from OpenVMS Version 6.2 or 7.0 systems, this change must be made before the upgrade.

For example, prior to OpenVMS Version 7.1—EFT2, the following device description block:

```
[AUA0]
 NAME=AU
 NODE=3
 DRIVER=SYS$MSBDRIVER
 IRQ=9
 DMA=(0,1)
 PORT=(388:4,530:8)
```

would be changed as follows:

```
[AUA0]
 NAME=AU
 DRIVER=SYS$MSBDRIVER
 IRQ=9
 DMA=(0,1)
 PORT=(388:4,530:8)
 NODE=3
```

3.6 Naming Serial Line Devices on Alpha Systems

V7.1–EFT1

OpenVMS has changed the way it handles the second serial port on the following Alpha systems:

AlphaStation 200, 400, 500 and 600 families
AlphaServer 1000, 2000, 2100, and 4100 families

If one of these systems is booted with the console environment variable set to graphics, the name of the serial line (COM1) will be different from that in previous releases. The COM1 device is called TTB0 instead of OPA1. In this case, the COM1 device is controlled by SYS$YSDRIVER instead of SYS$OPDRIVER.

If the console is set to serial, the COM1 device is called OPA0.

3.7 Memory Holes on AlphaServer 4100 Systems

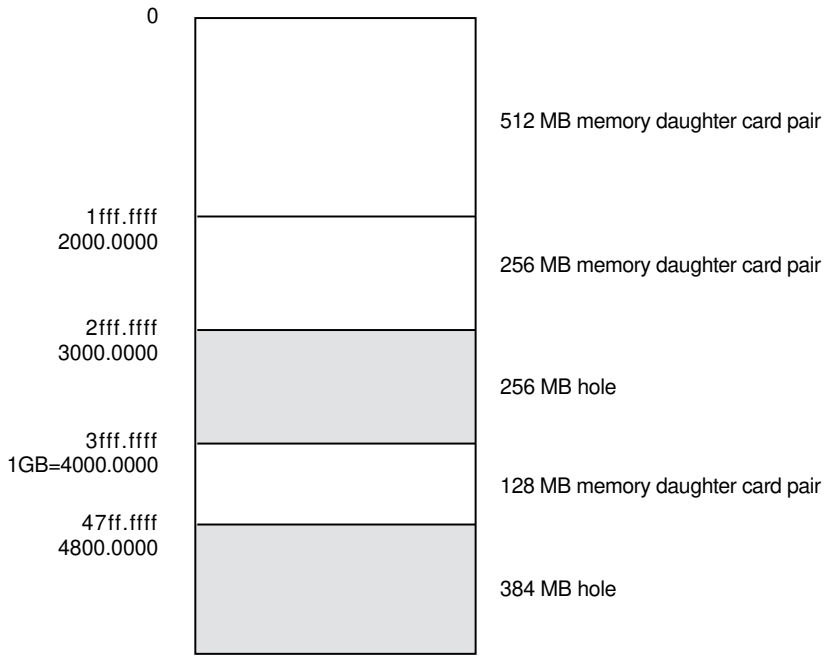
V7.1–EFT1

Physical memory holes might exist on AlphaServer 4100 systems. As illustrated in the example diagram in this section, there are three different sizes of memory daughter card pairs: 512 MB, 256 MB and 128 MB. In accordance with AlphaServer 4100 systems configuration rules, memory card pairs must be arranged in descending order of size.

The AlphaServer 4100 hardware reads the first set of memory daughter cards and assumes that any memory card pairs that follow are the same size. Memory holes occur because memory card pairs following the first set of cards read by the hardware might not be the same size. As shown in the following diagram, the hole at 3000.0000 must be dealt with by OpenVMS. The hole at 4800.0000 is at the top of the address space and can be ignored by OpenVMS.

Note

Previous versions of OpenVMS Alpha did not efficiently support systems with physical memory holes and ultimately led to an inefficient use of system memory. The memory management data structures in OpenVMS Alpha Version 7.1–EFT1 have been slightly modified to recognize the memory holes. As a result, inefficiencies in previous versions of the OpenVMS Alpha operating system have been eliminated.



Note that this configuration impacts the algorithm used to determine whether a driver needs to use map registers. In releases prior to OpenVMS Alpha Version 7.1–EFT1, device drivers do the following:

1. Call IOC$NODE_DATA with key IOC$K_DIRECT_DMA_SIZE to obtain the size of the direct DMA window (in megabytes). This is usually 1 GB.
2. Convert the size returned from IOC$NODE_DATA to pages and compare the size with **mmg$gl_memsize**, which contains the number of pages in physical memory.
3. If **mmg$gl_memsize** is greater than the size returned from IOC$NODE_ DATA, use map registers; otherwise, use the direct DMA window.

The **mmg$gl_memsize** global cell does not contain the memory hole. As a result, the system has only 7/8 GB of memory, but according to the algorithm in releases prior to OpenVMS Alpha Version 7.1–EFT1, it appears that the device can use the direct DMA window. Yet there is 128 MB of memory beyond the 1 GB border, which requires that the drivers use map registers. To eliminate this problem, drivers using the algorithm in releases prior to OpenVMS Alpha Version 7.1–EFT1 must substitute it with the following algorithm:

1. Call IOC$NODE_DATA with key IOC$K_DIRECT_DMA_SIZE to obtain the size of the direct DMA window (in megabytes). This is usually 1 GB.
2. Convert the size returned from IOC$NODE_DATA to pages by dividing the number of bytes by the contents of **mmg$gl_page_size**. For example:

```
int dma_size;
int pages;

status = IOC$NODE_DATA (crb, IOC$K_DIRECT_DMA_SIZE, &dma_size);
        /* dma_size contains the number of megabytes.
         * convert number of megabytes to bytes.
         */
dma_size = dma_size * (1024 * 1024);
        /* Convert number of bytes to number of pages by
         *  dividing by number of bytes per page.
         */
pages = dma_size / MMG$GL_PAGE_SIZE;
```

3. Compare the resulting number of pages with **mmg$gl_maxpfn + 1**.
4. If **mmg$gl_maxpfn + 1** is greater than the size returned from IOC$NODE_ DATA, use map registers; otherwise use the direct DMA window.

3.8 SYS$MSBDRIVER Removed from OpenVMS Alpha Distribution

V7.0

The driver for the Microsoft Windows Sound System ISA sound card (MSB), SYS$MSBDRIVER, has been removed from the OpenVMS Alpha distribution as of Version 7.0. The following files have been removed:

- SYS$LOADABLE_IMAGES:SYS$MSBDRIVER.EXE
- SYS$EXAMPLES:SOUND_SERVICES.C
- SYS$EXAMPLES:SOUND_SAMPLE.C
- SYS$EXAMPLES:SOUND_SAMPLE.SND
- SYS$LIBRARY:SYS$STARLET_C.TLB module MSB.H

An enhanced version of this driver, called MMOV$MSBDRIVER, is included in Multimedia Services Version 1.4 for OpenVMS Alpha. This layered product also includes support for video capture and playback, an enhanced version of DECsound, and other audio and video applications.

MMOV$MSBDRIVER provides the same $QIO programming interface as SYS$MSBDRIVER. Digital recommends that the WAVE Applications Programming Interface provided by Multimedia Services for OpenVMS be used instead because it is more flexible and is portable to other platforms. (Multimedia Services Version 1.4 for OpenVMS is described in SPD 64.24.00.)

3.9 OpenVMS Alpha PCI DMA Window Changes

V6.2

Reviewers:

This information is included in the *Writing OpenVMS Alpha Device Drivers in C* book, which will be included in the OpenVMS 7.1 docset. Therefore, this note will not be included in the SSB version of the V7.1 release notes.

This release note should be read by anyone writing a driver for a PCI, EISA, or ISA device.

PCI bus implementations on Alpha platforms define two ways for PCI devices to access main memory: scatter/gather memory access and physical memory access. In **scatter/gather memory access**, the PCI address generated by a PCI device is translated to a main memory address by a scatter/gather table. In **physical memory access**, the PCI address generated by a PCI device is translated to a main memory address by the addition of a constant. Scatter/gather memory access is called **scatter/gather DMA**. Physical memory access is called **physical (or direct) DMA**.

Each memory access technique has advantages and disadvantages. Scatter/gather DMA allows access to all system memory, but it is more complex to program. Physical DMA is easier to program, but it may limit the amount of main memory that can be addressed.

Alpha PCI platforms implement scatter/gather DMA and physical DMA through **DMA address windows** on the PCI bus. A PCI DMA address window is an address range on the PCI bus through which PCI devices (and EISA/ISA devices behind the PCI/EISA bridge or PCI/ISA bridge) access main memory. AlphaServer and AlphaStation systems provide a minimum of two DMA address windows on the PCI bus. A DMA address window can be a physical DMA window, where a PCI bus address is a linear function of a system memory address, or a scatter/gather DMA window, where a PCI bus address undergoes a page table translation before becoming a main memory address.

Alpha platforms that support PCI buses have both a physical DMA window and a scatter/gather DMA window. In OpenVMS Alpha Version 6.2, the scatter/gather DMA window is based at PCI address 0 and extends to a maximum address of 3FFFFFFF (the actual size of the scatter/gather DMA window is a function of the amount of physical memory in the system). In OpenVMS Alpha Version 6.2, the physical DMA window is typically based at a PCI address above the scatter/gather DMA window.

To allow drivers to find the base of the physical DMA window in a platform-independent manner, two new function codes have been added to the bus support routine IOC$NODE_DATA. The basic definition of the IOC$NODE_DATA routine is as follows:

```
int ioc$node_data (CRB  *crb, int  function_code, void *user_buffer)
```

Inputs:

crb	Address of CRB. IOC$NODE_DATA uses the crb$l_node field and the vec$ps_adp field to find the data structures associated with the I/O bus to which this device is connected.
function_code	From [lib.lis]iocdef.sdl. Specifies information to be returned by IOC$NODE_DATA.
user_buffer	Address of caller's buffer. On success, the requested information is returned in the caller's buffer.

Outputs:

SS$_NORMAL	Normal successful completion. Requested information is returned in the caller's buffer.
SS$_ILLIOFUN	Unrecognized function code.
SS$_BADPARAM	Bad parameter. Usually this means that crb$l_node contains an invalid slot number. Check that the driver has been loaded with the /node qualifier.

When called with function code IOC$K_DIRECT_DMA_BASE, the IOC$NODE_ DATA routine returns the base address of the physical DMA window. The base address is returned as a 64-bit value in anticipation of future 64-bit I/O buses. For this reason, the caller of the IOC$NODE_DATA routine should make sure the user_buffer argument points to a quadword cell when using the IOC$K_DIRECT_ DMA_BASE function code.

When called with function code IOC$K_DIRECT_DMA_SIZE, the IOC$NODE_ DATA routine returns the size of the physical DMA window, expressed in megabytes. The size of the direct DMA window is returned as a 32-bit value.

Using the physical DMA window for device DMA is straightforward. Once you have found the base of the physical DMA window using the IOC$NODE_DATA routine, you must adjust the DMA address that you assign to the device by adding the physical DMA window base to the main memory DMA buffer address. For example, on a typical system the physical DMA window is based at the PCI address 40000000 and extends to 7FFFFFFF. PCI addresses in the range from 40000000 to 7FFFFFFF are passed to main memory addresses 0 to 3FFFFFFF. This means that if you have a DMA buffer at main memory address 0, the PCI device would access this buffer at PCI address 40000000. The correspondence of main memory address and PCI DMA addresses is derived by the following formula:

```
PCI DMA address = (main memory buffer address) + (base of physical DMA window)
```

The main reason for locating the PCI DMA windows this way is to support bit-limited ISA devices. In OpenVMS Alpha Version 6.1, bit-limited ISA bus masters cannot use the scatter/gather DMA window because ISA bus masters do not have enough address bits to generate an address above 16 MB. For this reason, the scatter/gather DMA window was moved to PCI address 0. The scatter/gather table is now managed such that EISA and ISA devices that perform DMA in the scatter/gather window always get bus addresses below 16 MB.

Note that the size of the physical DMA window limits the amount of physical memory that can be addressed by a PCI, EISA, or ISA device. In the physical DMA window, it is not possible for an I/O device to address more than 1 GB of physical memory. For this reason, on large memory systems you may want to code your driver to perform DMA in the scatter/gather window.

3.10 Device IPL Setup for OpenVMS Alpha Drivers

V6.2

Alpha hardware platforms that support PCI, EISA, and ISA buses deliver I/O device interrupts at different IPLs, either 20 or 21. The IPL at which device interrupts are delivered can change if you move the device from one platform to another. This is a problem if the driver declares its device IPL to be 20, and then that driver is executed on a machine that delivers I/O device interrupts at IPL 21.

The simplest solution to this problem is for PCI, EISA, and ISA device drivers to use IPL 21. This works correctly on platforms that deliver I/O device interrupts at IPL 20 and on platforms that deliver I/O device interrupts at IPL 21.

A future release of OpenVMS Alpha may provide a platform-independent mechanism for drivers to determine the device IPL dynamically.

3.11 AlphaStation 205: PCI Configuration Restriction

V7.1–EFT1

The OpenVMS Alpha operating system does not support PCI option cards configured in PCI slot 0 on any AlphaStation 205 series systems.

PCI slot 0 is the lowest physical PCI option slot on AlphaStation 205 series systems. The interrupt signal for this slot is shared with the built-in Ethernet port. Because the OpenVMS Alpha operating system does not currently permit PCI devices to share an interrupt line, a PCI device installed in slot 0 will not function correctly or might cause errors to occur with the built-in Ethernet port. As a result of this restriction, AlphaStation 205 and 255 series systems support a maximum of two PCI option cards, configured in slot 1 and slot 2.

3.12 Unsupported Configurations for RZ25M and RZ26N Disk Drives

V7.1–EFT2

During the testing of Digital supported SCSI disk drives on configurations with long differential SCSI buses, two drives, RZ25M and RZ26N, were found to have bus phase problems. Therefore, these drives are not recommended for use in configurations were the differential bus length between the host controller and the target disk drive equals or exceeds 20 meters.

In multihost shared SCSI bus configurations (that is, SCSI clusters), the distance of the differential bus between each controller and the target drive must be less that 20 meters for the configuration to support RZ25M and RZ26N drives.

This restriction applies only to the RZ25M and RZ26N drives. All other Digital-supported SCSI disk drives, as listed in the OpenVMS SPD, may be used in configurations to the full bus lengths of the SCSI-2 specification.

3.13 SCSI Controller Restriction on AlphaServer 2100 Systems

The Adaptec 1740/1742 SCSI controller (PB2HA–SA) is not supported on AlphaServer 2100 systems having more than 1 gigabyte (GB) of memory. If the controller is connected to such a system, the following message appears on the operator's console:

```
%PKJDRVR-E- The direct DMA window does not map all of memory. Port is going OFF)
```

3.14 OpenVMS Alpha SCSI Firmware Support

The following sections relate to SCSI firmware support.

3.14.1 Updating Firmware in SCSI Configurations

V6.2—1H3

When you update the firmware for a SCSI device, that device must be connected directly to a SCSI adapter on the host system. If you apply firmware updates on SCSI devices that are accessed through an intelligent controller (such as an HSZ40 or KZPSC), the firmware update will fail.

3.14.2 Required Firmware Support for RZ26N and RZ28M Disks

V6.2—1H3

The minimum firmware revision level required for RZ26N and RZ28M disks is Revision 0568.

If the latest firmware revision level is not used with these disks, multiple problems may occur.

3.14.3 SCSI Firmware Update Instructions

V6.2

If your system contains RZ26L or RZ28 disks, the disks' firmware revision level must be at or above 442D for use in a SCSI OpenVMS Cluster system.

Important Note

Only certain RZ26L and RZ28 firmware revisions can be safely upgraded to firmware revision level 442D. Refer to Section 3.14.4 to determine if your disks are capable of being upgraded to firmware revision level 442D. If your disk is capable of supporting firmware revision level 442D, use the RZTOOLS Utility that is described in Section 3.14.5 to update the disk's firmware.

3.14.4 Firmware Revision Level 442D Requirements

V6.2

Only the following combinations of disk drives and firmware revision levels are capable of being upgraded safely to firmware revision level 442D. Performing the update procedure on any other combination can permanently damage the disk.

Table 3–2 Revision Level 442D Firmware Compatibility

Disk Drive	Firmware Revision	Disk File Name
RZ26L	440C	RZ26L_442D_DEC.FUP
RZ28	441C or D41C	RZ28_442D_DEC2104.FUP

3.14.5 442D Firmware Installation Procedure

V6.2

If you determine that your disk requires revision level 442D firmware, and it is capable of being upgraded safely, use the following procedure to update the firmware. (See Table 3–2 for the file name of the disk you are upgrading.)

```
$ MCR SYS$ETC:RZTOOLS_ALPHA DKB500 /LOAD=SYS$ETC:disk filename
  Read in 262144 bytes.
  Current FW version - X440C
  Upgrading to       - DEC0
  Loading code  ......
  New code has been sent to the drive.
```

3.15 OpenVMS Alpha SCSI Port and Class Drivers

V6.2

The following sections describe OpenVMS Alpha SCSI class and port device driver restrictions.

3.15.1 Add-On SCSI Adapters

V6.2

Version 6.2 and later of OpenVMS Alpha supports various add-on SCSI adapters. Digital's AlphaGeneration platforms typically support one or more integral SCSI adapters, with the option of installing additional add-on SCSI adapters. Due to differences in device-naming conventions used between the Alpha console and OpenVMS, the OpenVMS device name may not match the name displayed by the console.

For example, the console designation for a SCSI device on the integral SCSI adapter may be DKA100. However, when two additional add-on SCSI adapters are added, the ''A'' designation becomes ''C''; and DKA100 appears as DKC100 when OpenVMS is running.

Note that although the console and OpenVMS device names may be different, the unique specification of a device name from the console to the device name under OpenVMS will stay consistent, provided add-on SCSI adapters are not added or removed.

3.15.2 HSZ40 and Transportable SCSI Disk Shadow-Set Members

V6.2

An HSZ40 Raid-Array Controller provides the capability of an OpenVMS initialized SCSI disk (that is, one with a Files–11 ODS-2 format on it) to be moved between an OpenVMS controlled SCSI bus and an HSZ40 controlled SCSI bus without reinitializing the disk and losing data. Disks that contain this functionality are called **transportable** disks.

A SCSI disk initialized by the HSZ40 and then subsequently initialized by OpenVMS is called a **nontransportable** disk, and cannot be moved to an OpenVMS controlled SCSI bus without losing data.

OpenVMS Volume Shadowing requires that a SCSI disk support the SCSI commands READ_LONG/WRITE_LONG. These SCSI commands in conjunction with OpenVMS Volume Shadowing are used to handle certain classes of errors as seen under normal volume shadowing operations. SCSI disks that support the READ_LONG/WRITE_LONG capability while connected to an OpenVMS

controlled SCSI bus, lose this capability when the transportable disks are moved to the SCSI bus controlled by an HSZ40.

The lack of READ_LONG/WRITE_LONG capability is detected at shadow-set MOUNT time, by the following error:

```
MOUN$_DEVNOFE, device does not support FORCED ERROR handling
```

To correct this problem, specify the MOUNT qualifier /OVERRIDE=NO_FORCED_ERROR at shadow-set MOUNT time.

Note that specifying this MOUNT qualifier may cause shadow-set member SCSI disks to be removed from a shadow set if certain error conditions arise that cannot be corrected.

Digital recommends that HSZ40 nontransportable SCSI disks be used to contain shadow-set members that support READ_LONG/WRITE_LONG functionality, and offer benefits provided by the level of RAID chosen at initialization time.

3.16 OpenVMS VAX Device Support Documentation Corrections

This section describes corrections to OpenVMS VAX device support documentation.

3.16.1 *OpenVMS VAX Device Support Manual*

V6.1

The following sections describe corrections to the *OpenVMS VAX Device Support Manual*[1].

3.16.1.1 Linking a Device Driver

Chapter 12 of the *OpenVMS VAX Device Support Manual*, Version 6.0, describes how to assemble, link, and load a device driver. In step 3 of the procedure for preparing a driver for loading into the operating system, append the following text to the end of the procedure (following the paragraph that begins: ''The resulting image must . . . ''):

To produce an image with a symbol table compatible with the System Dump Analyzer (SDA), you must link again; this time, using the UNIVERSAL=* option statement (to include all global symbols and to ensure proper state of the REL bits in the object records). Relink as shown in the following example:

```
$ LINK /NOEXECUTABLE/NOTRACEBACK/NOSYSSHR -
_$     /SYMBOLS=MYDRIVER.EXE,-
_$     /SHARE=DUMMY_FILE_NAME,-
_$     /NOMAP,MYDRIVER1.OBJ,MYDRIVER2.OBJ,-
_$     SYS.STB/SELECTIVE,-
_$     SYS$INPUT/OPTION
_$     BASE=0
_$     UNIVERSAL=*
```

For more information about the Linker, see the *OpenVMS Linker Utility Manual*.

[1] This manual has been archived but is available in PostScript and DECW$BOOK (Bookreader) formats on the OpenVMS Documentation CD–ROM. A printed book can be ordered through DECdirect (800–354–4825).

3.16.1.2 Device-Register I/O Space: Usage Restrictions

Chapter 5 of the *OpenVMS VAX Device Support Manual*, Version 6.0, describes device driver coding and the restrictions on the use of device-register I/O space. The third sentence of the fifth bulleted paragraph in Section 5.2 states that the instructions that refer to UNIBUS adapter registers must use longword context. This is the wrong bus. The sentence should read:

> ''Instructions that refer to MASSBUS adapter registers must use the longword context.''

3.16.2 *OpenVMS VAX Device Support Reference Manual*

V6.1

The following sections describe corrections to the *OpenVMS VAX Device Support Reference Manual*[1].

3.16.2.1 COM$DRVDEALMEM Routine Synchronization

Chapter 3 of the *OpenVMS VAX Device Support Reference Manual*, Version 6.0, contains a section describing the COM$DRVDEALMEM routine.

At the end of the paragraph under *Synchronization*, add the following sentence:

''If called at IPL$_SYNCH or higher, the routine executes the fork process.''

3.16.2.2 CRB Data Structure

Chapter 1, Section 1.7, of the *OpenVMS VAX Device Support Reference Manual*, Version 6.0, contains Table 1-8 describing the CRB data structure fields. The description in the table for the CRB$L_INTD field is confusing and needs clarification. Replace the first two sentences in the CRB$L_INTD description as follows:

Field Name	Description
CRB$L_INTD	Portion of the interrupt transfer vector block that stores executable code, driver entry points, and I/O adapter information. This 10-longword area is overlaid with the contents of the interrupt transfer vector block that starts at VEC$L_INTD (offset 16) as described in Section 1.7.1. It contains pointers to the driver's . . .

3.16.2.3 SCDRP Data Structure SCSI Flags

Chapter 1 of the *OpenVMS VAX Device Support Reference Manual*, Version 6.0, contains a section describing the SCDRP data structure SCSI flags.

In the SCDRP$L_SCSI_FLAGS field description for bit SCDRP$V_LOCK, make the following correction:

Change: SCDRP$VLOCK

To: SCDRP$V_LOCK

[1] This manual has been archived but is available in PostScript and DECW$BOOK (Bookreader) formats on the OpenVMS Documentation CD–ROM. A printed book can be ordered through DECdirect (800–354–4825).

3.16.2.4 SPI$CONNECT Macro

Chapter 2 of the *OpenVMS VAX Device Support Reference Manual*, Version 6.0, contains a section describing the SPI$CONNECT macro.

In the table listing the required inputs, add:

R4 Address of the SPDT

In the values returned in R3, the SPDT$M_CMDQ bit was added to the port capability mask (SPDT$L_PORT_FLAGS). When set, SPDT$M_CMDQ indicates that the port driver supports command queuing I/O.

In the return values table listing R3 and the mask bits (after SPDT$M_LUNS), add:

SPDT$M_CMDQ Supports command queuing I/O

3.16.2.5 SPI$GET_CONNECTION_CHAR and SPI$SET_CONNECTION_CHAR Macros

Chapter 2 of the *OpenVMS VAX Device Support Reference Manual*, Version 6.0, contains sections describing the SPI$GET_CONNECTION_CHAR and SPI$SET_CONNECTION_CHAR macros. Appended to the macro characteristics buffer is longword #12 for SCSI-2 support.

At the end of the characteristics buffer table in these macro descriptions, add the longword #12 information as follows:

12 SCSI-2 device characteristic status bits. Bits of this longword are defined as follows:

Bit	Description
0	When set, (SCDT$V_SCSI_2) indicates the device connection is SCSI-2 conformant.
1	When set, (SCDT$V_CMDQ) indicates the device connection supports command queuing.

3.16.2.6 $EQULST Macro

Chapter 2 of the *OpenVMS VAX Device Support Reference Manual*, Version 6.0, contains a section describing the $EQULST macro.

In the parameter description for **symbol,value** insert the phrase *in decimal* as follows:

‘‘ . . . and value specifies in decimal the value of the symbol.’’

V6.0

The OpenVMS operating system ensures that when an I/O write operation returns a successful completion status, the data is available on the disk or tape media.

OpenVMS guarantees atomicity for single-block I/O write operations to DIGITAL Storage Architecture (DSA) drives.[1] However, if a system failure occurs while a multiple block I/O write operation is in progress, the OpenVMS operating system does not guarantee the successful completion of the I/O write. When a failure interrupts a write operation, the data can be left in any one of the following conditions:

- The new data is written completely to the disk blocks on the media, but a completion status was not returned before the failure.
- The new data is partially written to the media so that some of the disk blocks involved in the I/O contain the data from the in-progress I/O write and other blocks contain the data that was present prior to the I/O write operation. Note that blocks updated with new data can be interspersed with blocks containing old data.
- The new data was never written to any of the disk blocks on the media.

Individual blocks can contain either all new data or all old data; a block is not partially updated when dealing with DSA devices.

To guarantee that an I/O write operation either completes successfully, or (in the event of failure) is redone or rolled back as if it were never started, applications must be supplemented by additional techniques to ensure data correctness and recovery. For example, using RMS Journaling for OpenVMS or database or file journaling and recovery techniques allows applications to automatically recover from failures such as:

- Permanent loss of the path between a CPU data buffer containing the data being written and the disk being written to during a multiple block I/O operation. Communication path loss can occur due to node failure or a failure of node-to-node communications. Note that transient failures are automatically recovered from during mount verification without requiring any further recovery techniques.
- Failure of a CPU (such as a system crash, halt, power failure) during a multiple block I/O write operation.
- Mistaken deletion of a file.
- OpenVMS RMS incomplete bucket write operation to an indexed file.
- Cancellation of an in-progress multiple block write operation.

Note that Volume Shadowing for OpenVMS promotes availability by providing access to data in the presence of media deterioration, communication path failure, or controller or device failure. However, volume shadowing should not be mistaken for a journaling product that tracks changes to files for data integrity.

[1] Single block atomicity may not be guaranteed with SCSI devices. This includes the new HSZ and HSJ SCSI adapters that extend SCSI disk connectivity to the HSC family of devices. An OpenVMS operating system makes no distinction between the SCSI-adapted DSA devices and actual SCSI devices.

A

OpenVMS Alpha Version 6.2—1H3 Device Naming Notes

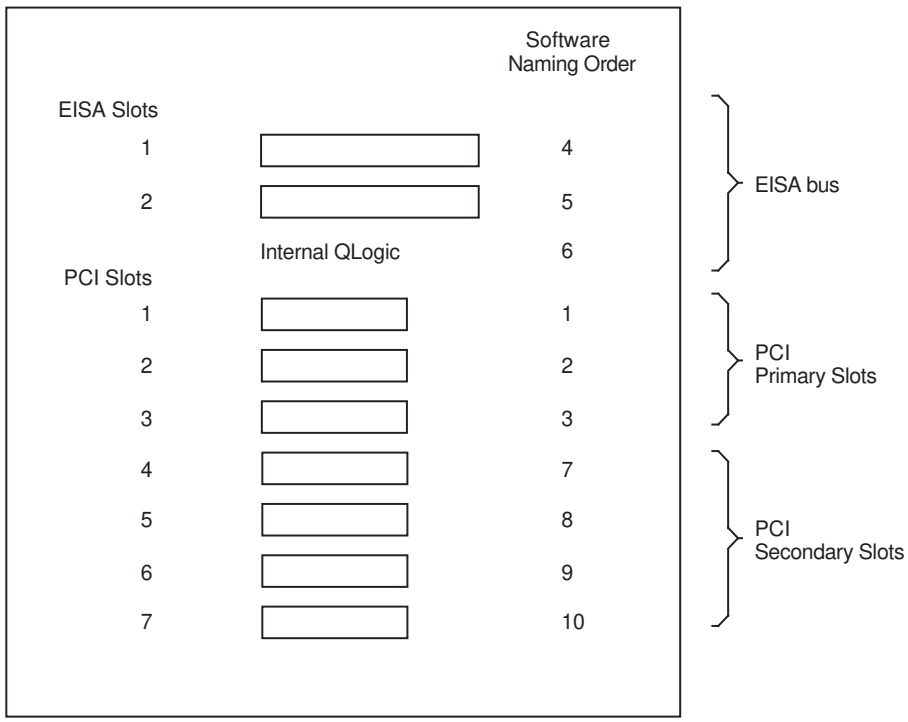
A.1 Device Naming on AlphaServer 1000A Systems

AlphaServer 1000A systems have seven PCI option slots and two EISA option slots, in addition to an integrated QLogic 1020A Fast Wide SCSI controller. The OpenVMS operating system and the console do not always assign the same names to devices in these option slots. In addition, the slot numbering convention used by the console and the OpenVMS operating system does not match the physical slot numbers on the backplane.

The EISA slots are connected using a PCI-to-EISA bridge on the primary PCI bus. The upper three PCI option slots (numbered 1 to 3) are also on the primary PCI bus, while the lower four PCI option slots (numbered 4 to 7) are connected using an integrated PCI-to-PCI bridge. The QLogic 1020A SCSI controller is in a fixed position relative to the option slots; this controller is always configured as the first device behind the integrated PCI-to-PCI bridge, preceding slots 4 to 7. (See Figure A–1.)

OpenVMS Alpha Version 6.2—1H3 Device Naming Notes
A.1 Device Naming on AlphaServer 1000A Systems

Figure A–1 AlphaServer 1000A Hardware Backplane



ZK–8661A–GE

The OpenVMS operating system assigns device names in the following order:

1. Devices on the primary PCI (slots 1 to 3)
2. EISA devices
3. Integrated SCSI controller
4. Devices on the secondary PCI (slots 4 to 7)
5. Devices on PCI-to-PCI bridges in secondary PCI slots
6. Devices on PCI-to-PCI bridges in primary PCI slots

Consider a system with only a KZPSA SCSI controller in slot 1, the first slot on the primary PCI bus. The KZPSA appears as PKA, and the integrated QLogic 1020A appears as PKB. If another KZPSA were added in slot 4, behind the integrated bridge, it would appear as PKC. If yet another KZPSA were added in slot 3, the third slot on the primary PCI bus, the device names would shift; the original KZPSA in slot 1 remains PKA, the new KZPSA in slot 3 becomes PKB, the QLogic becomes PKC, and the KZPSA in slot 4 becomes PKD.

In another example, consider a system with one KZPSA in slot 4. This KZPSA is behind the integrated bridge, behind the QLogic 1020A, and appears as PKB with the QLogic 1020A as PKA. A system with only bridge cards in its option slots, for instance, P1SE bridges with integrated SCSI controllers, would always see the QLogic 1020A as PKA and the SCSI controllers on the P1SE bridges as PKB and so forth.

A.1.1 Restriction for PCI Adapters

As shown in Figure A–1, PCI slots 4-7 on AlphaServer 1000A systems are connected by an integrated PCI-to-PCI bridge. Certain PCI adapters, such as the KZPSM, also contain an onboard PCI-to-PCI bridge. Because the KZPSM is not supported in PCI slots 4-7, you must place that adapter in PCI slots 1, 2, or 3.

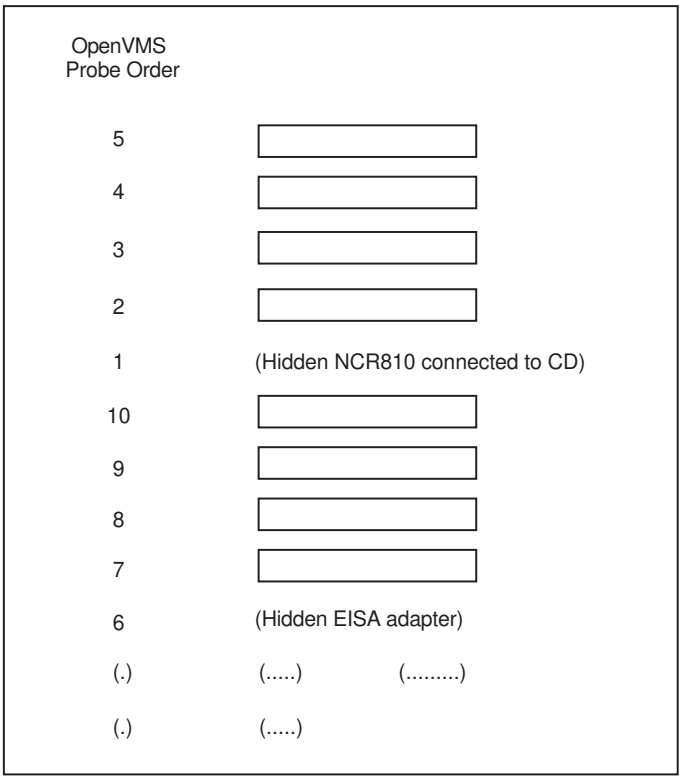
A.2 Device Naming on AlphaServer 4100 Systems

AlphaServer 4100 systems have eight PCI option slots, three of which are shared with EISA option slots. Because the OpenVMS operating system and the console each have their own device naming conventions, the console name of a device will not necessarily match the OpenVMS name of a device. In addition, the slot numbering convention used by the console and the OpenVMS operating system does not match the physical slot order on the backpanel.

AlphaServer 4100 systems also include an integrated NCR810 SCSI controller and an integrated PCI/EISA bridge. The integrated NCR810 SCSI controller is connected directly to the compact disc reader.

The order in which OpenVMS names devices is shown by the OpenVMS Probe Order column in Figure A–2.

Figure A–2 AlphaServer 4100 Hardware Backpanel



ZK–8684A–GE

The integrated NCR810 controller (connected to the compact disc reader) will always be assigned the same name by both the console and by OpenVMS because it is always named first. The OpenVMS operating system assigns device names in the following order (using the OpenVMS probe numbers shown in Figure A–2):

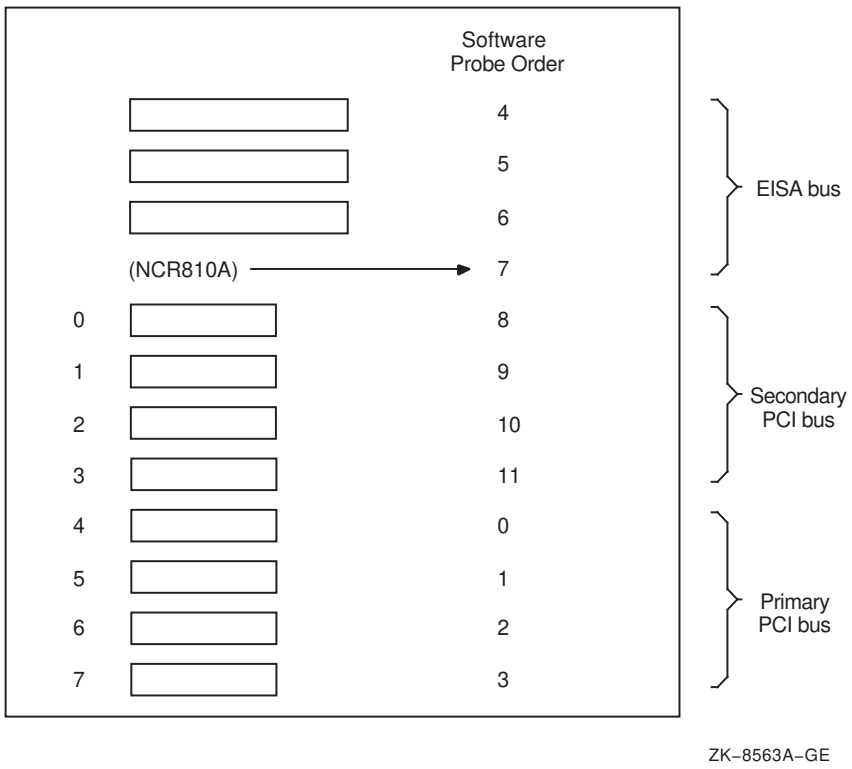
1. Integrated NCR 810 connected to the CD
2. PCI slots 2 through 5
3. Devices on PCI-to-PCI bridges on PCI slots 2 through 5
4. PCI slots 7 through 10
5. EISA slots 7 through 9
6. Devices on PCI-to-PCI bridges on PCI slots 7 through 10

A.3 Device Naming on AlphaServer 2100A Systems

AlphaServer 2100A systems have eight PCI option slots and three EISA option slots, in addition to an integrated NCR810A SCSI controller. The OpenVMS operating system and the console do not always assign the same names to devices in these option slots. In addition, the slot numbering convention used by the console and by the OpenVMS operating system does not match the physical slot numbers on the backplane.

The EISA slots are connected using a PCI-to-EISA bridge on the primary PCI bus. The bottom four PCI option slots (hardware numbers 4 to 7) are also on the primary PCI bus, while the remaining four PCI option slots (the top slots, hardware numbers 0 to 3) are connected using an integrated PCI-to-PCI bridge. The NCR810A SCSI controller is in a fixed position relative to the option slots; the NCR810A controller is always configured as the first device behind the integrated PCI-to-PCI bridge, preceding PCI slots 0 to 3. (See Figure A–3.)

Figure A–3 AlphaServer 2100A Hardware Backplane



The OpenVMS operating system assigns device names in the following order:

1. Devices on the primary PCI (slots 4 to 7)
2. EISA devices
3. Integrated NCR810A SCSI controller
4. Devices on the secondary PCI (slots 0 to 3)
5. Devices on PCI-to-PCI bridges in secondary PCI slots
6. Devices on PCI-to-PCI bridges in primary PCI slots

Consider a system with only a KZPSA SCSI controller in slot 4, the first slot on the primary PCI bus. The KZPSA appears as PKA, and the integrated NCR810A appears as PKB. If another KZPSA were added in slot 0, behind the integrated bridge, it would appear as PKC. If yet another KZPSA were added in slot 6, the third slot on the primary PCI bus, the device names would shift; the original KZPSA in slot 4 remains PKA, the new KZPSA in slot 6 becomes PKB, the 810A becomes PKC, and the KZPSA in slot 0 becomes PKD.

In another example, consider a system with one KZPSA in slot 0. This KZPSA is behind the integrated bridge, behind the 810A, and appears as PKB, with the 810A as PKA. A system with only bridge cards in its option slots, for instance, P1SE bridges with integrated SCSI controllers, would always see the 810A as PKA and the SCSI controllers on the P1SEs as PKB and so forth.