VAX/VMS Supplemental Information, Version 4.7

Order No. AA-KX21A-TE

digital™

software

VAX/VMS Supplemental Information, Version 4.7

Order Number: AA–KX21A–TE

December 1987

This document contains information that supplements the VAX/VMS documentation set.

Revision/Update Information: This is a new manual.

Operating System and Version: VAX/VMS Version 4.7

digital equipment corporation
maynard, massachusetts

December 1987

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem–10	PDP	VT
DECSYSTEM–20	PDT	
DECUS	RSTS	
DECwriter	RSX	digital ™

ZK4586

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

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

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

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

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

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.

™ PostScript is a trademark of Adobe Systems, Inc.

Contents

Preface

Intended Audience

This manual is for all users of the VAX/VMS operating system.

Document Structure

This manual contains the following four chapters and one appendix:

- Chapter 1 contains information for general users of the VAX/VMS operating system.
- Chapter 2 contains information for system managers.
- Chapter 3 contains information for application programmers.
- Chapter 4 contains information for system programmers.
- Appendix A contains information for users who need to write and load a device driver for a non-DIGITAL-supplied device attached to the VAXBI bus.

Conventions

Convention	Meaning
$\boxed{\text{RET}}$	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.)
CTRL/C	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
$ SHOW TIME 05-JUN-1988 11:55:22	In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red.
$ TYPE MYFILE.DAT . . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.

Convention	Meaning
input-file, . . .	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	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.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

1 Supplemental Information for General Users

This chapter contains supplemental information for general users of the VAX/VMS operating system.

1.1 *VAX/VMS DCL Concepts Manual* — Corrections

The following corrections apply to the *VAX/VMS DCL Concepts Manual*:

- Page 1–8. Disregard the note in Section 1.7.4. Applications are not affected by the SET TERMINAL/LINE_EDIT command.
- Page 2–4. The character "T" is no longer a valid abbreviation for the TYPE command. You must now enter "TY".
- Page 2–12, list item 4. Replace the /OUTPUT with the /OBJECT qualifier.
- Page 3–1. In Section 3.1, the syntax given for a full file specification is incorrect. It should appear as follows:

  node::device:[directory]filename.type;version
- Page 3–5. In Section 3.3.2, the list of special ANSI "a" characters that can be used in a magnetic tape file specification is incorrect. This list should contain the following characters:

  ```
  ! " % ' ( ) * + , - . / : ; > = < ?
  ```

  In order to use ANSI "a" characters in the file specification for a magnetic tape, the volume name must be enclosed in quotation marks.
- Page 4–14. In Section 4.7.1, replace the /PARENT qualifier with the /PARENT_TABLE qualifier.

1.2 *Guide to Using DCL and Command Procedures on VAX/VMS* — Corrections

The following corrections apply to the *Guide to Using DCL and Command Procedures on VAX/VMS*:

- Page 1–11. The last two lines in the example LOGICALS.COM file should read as follows:

  ```
  $ DEFINE JON DAISY::HARRIS
  $ DEFINE JANE DAISY::MOORE
  ```
- Page 4–16. In the example in Section 4.6.2, the statement

  ```
  $ WRITE "Result is",RES
  ```

  should be replaced as follows:

  ```
  $ WRITE SYS$OUTPUT "Result is ",RES
  ```

- Page 5–14. The following line of code is missing from the example at the top of the page:

  ```
  $     NUM = NUM + 1
  ```

  Insert this line under PROCESS_LOOP as follows:

  ```
  $ PROCESS_LOOP:
  $     FILE = F$ELEMENT(NUM,"/",FILE_LIST)
  $     IF FILE .EQS. "/" THEN GOTO DONE
  $     COPY 'FILE'.MEM MORRIS::DISK3:[DOCSET]*.*
  $     NUM = NUM + 1
  $     GOTO PROCESS_LOOP
  ```

- Page 5–15. The first statement in the example should have a hyphen (-) at the end of the line as follows:

  ```
  $ COMMAND_LIST = "DELETE/DIRECTORY/EXIT/" + -
  ```

- Page 6–7. In the example at the top of the page, the following statement is incorrect:

  ```
  $ INQUIRE RECORD "Enter name"
  ```

  Replace the preceding statement with the following statement:

  ```
  $ INQUIRE NAME "Enter name"
  ```

- Page 8–5. In the text following the third bullet, the qualifier /NOPRINT should be /NOPRINTER.

1.3 *VAX/VMS Mail Utility Reference Manual* — Addition

The description of the /SELF qualifier in the *VAX/VMS Mail Utility Reference Manual* should include the following information:

- There is a corresponding /NOSELF qualifier.
- If you send a message from the DCL level (that is, you do not receive the MAIL> prompt from within the Mail Utility), specifying /SELF or /NOSELF overrides any setting you have established by the SET COPY_SELF command within the Mail Utility.
- Specifying /SELF or /NOSELF on the DCL command line has no effect if you enter the Mail Utility and receive the MAIL> prompt.

Thus, for example, you could specify the following command to send MYFILE.DAT to user JONES and avoid receiving a copy of the file yourself even if you have previously entered the SET COPY_SELF command within the Mail Utility.

```
$ MAIL/NOSELF MYFILE.DAT JONES
```

1.4 *VAX/VMS User's Manual* — Correction

The positions of two figures in the *VAX/VMS User's Manual* were inadvertently reversed in the last release. The figure appearing under Appendix CHAR, CHAR.1 belongs under Appendix MAIL, MAIL.3.1. The figure that appears under Appendix MAIL, MAIL.3.1 belongs under Appendix CHAR, CHAR.1.

1.5 SET HOST/DTE/DIAL Command — Additional Modems Supported

The DCL command SET HOST/DTE/DIAL now supports the following modems:

- DF03
- DF112
- DMCL (any modem that uses DIGITAL Modem Control Language)

1.6 Support for VT300-Series Terminals

VAX/VMS supports the VT300-series terminals. The terminal device type is VT300_SERIES, and the terminal characteristic is DEC_CRT3. The DEC_CRT3 characteristic indicates the following:

- The ISO LATIN-1 character set is resident
- The ability to display a twenty-fifth line (the status line)
- The ability to report explicit state information about itself

When entering the SET TERMINAL command, you can specify VT300_SERIES as a terminal type for the /DEVICE_TYPE qualifier. You can also supply a value of 3 to the /DEC_CRT qualifier, which sets the DEC_CRT3 terminal characteristic.

1.7 Terminal Driver Line Editing — Clarification

The following information clarifies the documentation for CTRL/V in Table 1-2 of the *VAX/VMS DCL Concepts Manual* and in Table GEN-2 of the *VAX/VMS Mini-Reference.*

At DCL level, CTRL/V disables the command line editing features that were new with Version 4.0. For example, if you press CTRL/V and press CTRL/D, a CTRL/D is generated instead of the cursor moving left one character. Note, however, that CTRL/D is a terminator at the DCL level. Thus, pressing CTRL/V followed by pressing CTRL/D simulates a carriage return. DCL uses the default RMS terminator set, which is described in Chapter 8 of the *VAX/VMS I/O User's Reference Manual: Part I.*

Control characters that are not terminators at DCL level, such as CTRL/H and CTRL/J, have no effect when you use them with CTRL/V, because a backspace or a linefeed in the middle of a line results in an invalid command.

Certain control keys perform the same function with Version 4.0 as they did in previous versions of the VAX/VMS operating system. If you press one of these keys (including CTRL/U) after a CTRL/V, the key behaves as it did prior to Version 4.0.

1.8 New Hardware Configuration for VAX 8200/8300/8350 Systems

A new hardware configuration exists for VAX 8200/8300/8350 systems. Software documentation for the original VAX 8200/8300/8350 hardware configuration is still accurate, with the following exceptions:

- The main cabinet on the new configuration is wider than the original VAX 8200/8300/8350 cabinet.
- Diskette drives on the new configuration are oriented horizontally. CSA1 is the top diskette drive and CSA2 is the bottom diskette drive. Diskette drives on the original configuration are oriented vertically.
- The processor control panel on the new configuration is located to the right of the diskette drives. The processor control panel on the original configuration is located under the diskette drives.
- The new hardware configuration supports a 24-slot backplane; the original configuration supports a 12-slot backplane.

2 Supplemental Information for System Managers

This chapter contains supplemental information for system managers.

2.1 SET TIME Command — /[NO]CLUSTER Qualifier

The DCL command SET TIME has a new /[NO]CLUSTER qualifier. The default value for this qualifier is /NOCLUSTER.

Use the SET TIME/CLUSTER command to update the time on all nodes present in the VAXcluster. If you enter the SET TIME/CLUSTER command without a new time value, the system reads the time-of-year clock on the local node, then sets all nodes in the cluster to that time.

Because of communications and processing delays, the command cannot synchronize clocks exactly; the variation is typically less than a few hundredths of a second. If the command cannot verify that the time was set to within one half second of the specified time, you receive a warning message that specifies the name of the node that failed to respond quickly enough.

As a result of slight inaccuracies in each interval clock, times on the nodes of a cluster tend to drift apart. You can use the following procedure to keep the time on all cluster nodes reasonably synchronized:

```
$ SYNCH_CLOCKS:
$       SET TIME /CLUSTER
$       WAIT 6:00:00
$       GOTO SYNCH_CLOCKS
```

This procedure sets the time on all cluster nodes to the value obtained from the local time-of-year clock, waits, then resets the time for the cluster.

See the *VAX/VMS DCL Dictionary* for more information about the SET TIME command.

2.2 *VAX/VMS DECnet Test Sender/DECnet Test Receiver Utility Reference Manual* — Correction

The description of the /[NO]DISPLAY qualifier on page DTS-8 should be replaced as follows:

/[NO]DISPLAY=number
Instructs DTS to print the specified number of bytes (in hexadecimal) of data and interrupt messages to DTR. The default is /NODISPLAY.

2.3 *VAX/VMS Network Control Program Reference Manual* — Corrections

The following sections contain corrections to the *VAX/VMS Network Control Program Reference Manual.*

2.3.1 CONNECT NODE Command — Use VIA Parameter

The *VAX/VMS Network Control Program Reference Manual* contains an error on page NCP–40. SERVICE CIRCUIT is incorrectly listed as a command parameter that can be used with the CONNECT NODE command. Instead of the SERVICE CIRCUIT parameter, use the command parameter VIA as follows:

VIA circuit id — Specifies the circuit to be used to create the logical link between the host node and the target node. The circuit must be an Ethernet circuit.

In addition, replace the command example with the corrected command example that follows:

```
NCP> CONNECT NODE RTDEV SERVICE PASSWORD FEFEFEFEFEFEFEFE-
_ NCP> VIA UNA-0 PHYSICAL ADDRESS AA-00-04-00-38-00
```

2.3.2 X.25 Packet Level Events Numbered Incorrectly

In Section A.4.6 of the *VAX/VMS Network Control Program Reference Manual,* "X.25 Packet Level Events," the events numbered 7.3 through 7.14 should in fact be numbered 7.0 through 7.11.

2.3.3 DTE State Tables Corrected

The following two tables give corrected information for Tables NCP-6 and NCP-7 (pages NCP-178 and NCP-179) in the *VAX/VMS Network Control Program Reference Manual.*

Table 2–1 lists all VAX PSI management states and substates for DTEs. Table 2–2 provides a list of DTE state transitions.

Table 2–1 DTE States and Substates

State	Substate	Meaning
OFF	RUNNING	X.25 level 2 and level 3 software is operational, but the DTE is not available for use. Incoming calls are cleared.
	SYNCHRONIZING	X.25 level 2 software is operational, but level 3 software is not. The DTE is not available for use.
	UNSYNCHRONIZED	X.25 levels 2 and 3 are not operational, and the DTE is not available for use.
ON	RUNNING	The DTE is available for normal use.
	SYNCHRONIZING	X.25 level 2 software is operational, level 3 software is starting up, and the DTE will soon be available for use.
	UNSYNCHRONIZED	X.25 level 2 software is starting up, and the DTE will soon be available for use.
SHUT	RUNNING	X.25 levels 2 and 3 are operational, but the DTE is not to be used for any new activity; that is, all existing virtual circuits will be allowed to complete their operations. Incoming calls are cleared.
	SYNCHRONIZING	X.25 level 2 software is operational and level 3 software is starting up. When the DTE is available for use, no circuits can be established.
	UNSYNCHRONIZED	X.25 level 2 software is starting up. When the DTE is available for use, no circuits can be established.

Table 2–2 DTE State Transitions

Old State	New State	Cause of Change
OFF-RUNNING	ON-RUNNING	Operator command: SET MODULE X25-PROTOCOL DTE STATE ON.
	OFF-SYNCHRONIZING	X.25 level 3 software is resynchronizing.
	OFF-UNSYNCHRONIZED	X.25 level 2 software is resynchronizing.
OFF-UNSYNCHRONIZED	ON-UNSYNCHRONIZED	Operator command: SET MODULE X25-PROTOCOL DTE STATE ON.
	OFF-SYNCHRONIZING	X.25 level 2 startup has completed.
OFF-SYNCHRONIZING	ON-SYNCHRONIZING	Operator command: SET MODULE X25-PROTOCOL DTE STATE ON.

Table 2–2 (Cont.) DTE State Transitions

Old State	New State	Cause of Change
	OFF-RUNNING	X.25 level 3 startup has completed.
	OFF-UNSYNCHRONIZED	X.25 level 2 software is resynchronizing.
ON-RUNNING	OFF-RUNNING	Operator command: SET MODULE X25-PROTOCOL DTE STATE OFF.
	SHUT-RUNNING	Operator command: SET MODULE X25-PROTOCOL DTE STATE SHUT.
	ON-SYNCHRONIZING	X.25 level 3 software is resynchronizing.
	ON-UNSYNCHRONIZED	X.25 level 2 software is resynchronizing.
ON-UNSYNCHRONIZED	OFF-UNSYNCHRONIZED	Operator command: SET MODULE X25-PROTOCOL DTE STATE OFF.
	SHUT-UNSYNCHRONIZED	Operator command: SET MODULE X25-PROTOCOL DTE STATE OFF.
	ON-SYNCHRONIZING	X.25 level 2 startup has completed.
ON-SYNCHRONIZING	OFF-SYNCHRONIZING	Operator command: SET MODULE X25-PROTOCOL DTE STATE OFF.
	SHUT-SYNCHRONIZING	Operator command: SET MODULE X25-PROTOCOL DTE STATE SHUT.
	ON-RUNNING	X.25 level 3 startup has completed.
	ON-UNSYNCHRONIZED	X.25 level 2 software is resynchronizing.
SHUT-RUNNING	OFF-RUNNING	Operator command: SET MODULE X25-PROTOCOL DTE STATE OFF.
	ON-RUNNING	Operator command: SET MODULE X25-PROTOCOL DTE STATE ON.
	SHUT-SYNCHRONIZING	X.25 level 3 software is resynchronizing.
	SHUT-UNSYNCHRONIZED	X.25 level 2 software is resynchronizing.
SHUT-UNSYNCHRONIZED	OFF-UNSYNCHRONIZED	Operator command: SET MODULE X25-PROTOCOL DTE STATE OFF.

Table 2–2 (Cont.) DTE State Transitions

Old State	New State	Cause of Change
	ON-UNSYNCHRONIZED	Operator command: SET MODULE X25-PROTOCOL DTE STATE ON.
	SHUT-SYNCHRONIZING	X.25 level 2 startup has completed.
SHUT-SYNCHRONIZING	OFF-SYNCHRONIZING	Operator command: SET MODULE X25-PROTOCOL DTE STATE OFF.
	ON-SYNCHRONIZING	Operator command: SET MODULE X25-PROTOCOL DTE STATE ON.
	SHUT-RUNNING	X.25 level 3 startup has completed.
	SHUT-UNSYNCHRONIZED	X.25 level 2 software is resynchronizing.

2.3.4 SHOW CIRCUIT Command Changes

Prior to Version 4.6, the Network Control Program's SHOW CIRCUIT ... SUMMARY command displayed circuit information about all adjacent nodes (routing and nonrouting). For Version 4.6 and subsequent versions, the SUMMARY parameter displays circuit information about adjacent routing nodes only. However, the STATUS parameter continues to display circuit information about all adjacent nodes. For more information about the SHOW CIRCUIT command, refer to the *VAX/VMS Network Control Program Reference Manual.*

2.4 *VAX/VMS Networking Manual* — Corrections

Make the following corrections on page 3-46, in Section 3.5.7.4:

- In the first sentence, the default value for the RECALL TIMER parameter should be 100 seconds.
- In the third sentence, note that the circuit is placed in the ON-FAILED state if an attempt to make an outgoing call causes the system to exceed the MAXIMUM RECALLS parameter.

2.5 *Guide to VAX/VMS Software Installation* — Correction

In the *Guide to VAX/VMS Software Installation,* Section 7.5.4.1, the FILLM quota should be 60, not 20.

2.6 AUTOGEN Enhancements

The sections that follow describe enhancements to the AUTOGEN command procedure.

2.6.1 OLDSITE Parameter-Passing Mechanism Becoming Obsolete

The MODPARAMS parameter-passing mechanism supersedes the OLDSITE mechanism. The transition to the MODPARAMS mechanism involves two steps. Version 4.6 eliminated SYS$SYSTEM:OLDSITE1.DAT. The next major release of the VAX/VMS operating system will eliminate SYS$SYSTEM:OLDSITE2.DAT, OLDSITE3.DAT, and OLDSITE4.DAT.

The parameter values created by the files OLDSITEn.DAT can be found in the file SYS$SYSTEM:PARAMS.DAT. Comment lines in SYS$SYSTEM:PARAMS.DAT indicate which OLDFILEn.DAT the parameters come from. DIGITAL recommends that you review the parameters in the most recent version of SYS$SYSTEM:PARAMS.DAT.

When you review SYS$SYSTEM:PARAMS.DAT, you may find parameters necessary for your site that were transferred from OLDSITEn.DAT files. Include the records for these parameters from PARAMS.DAT in SYS$SYSTEM:MODPARAMS.DAT. If MODPARAMS.DAT does not exist, you can create it with a text editor.

After updating MODPARAMS.DAT to reflect the parameters transferred from the OLDSITEn.DAT files, invoke AUTOGEN as follows:

```
$ @SYS$UPDATE:AUTOGEN GETDATA REBOOT INITIAL
```

For more information about the MODPARAMS parameter-passing mechanism, refer to Section 11.4 of the *VAX/VMS System Manager's Reference Manual.*

2.6.2 New WSMAX Check

Since a value for the WSMAX parameter that is too high can disable pool expansion, AUTOGEN now displays a warning if it discovers a user-supplied value for the WSMAX parameter that appears to be too high.

2.6.3 Current Parameter Values Saved

AUTOGEN now saves current system parameters in SYS$SYSTEM:VAXVMSSYS.OLD before updating these parameters in SYS$SYSTEM:VAXVMSSYS.PAR.

2.6.4 Specifying an Alternate Startup Command Procedure

If you use a startup command procedure other than SYS$SYSTEM:STARTUP.COM, you can now assign the name of your procedure to the symbol STARTUP in MODPARAMS.DAT. After invoking AUTOGEN, your procedure becomes the default startup command procedure.

For example, to specify MY_STARTUP.COM as the new default startup command procedure, make the following entry in MODPARAMS.DAT:

```
STARTUP = "SYS$MANAGER:MY_STARTUP.COM"
```

2.6.5 QUORUM Value Calculated

AUTOGEN calculates a value for the QUORUM parameter by selecting the higher of the following two values: the initial quorum or the current cluster quorum.

2.6.6 Page and Swap File Handling — Improvements

AUTOGEN now understands and manipulates secondary page and swap files. Generally, if secondary page or swap files exist, AUTOGEN's file manipulation involves secondary files but excludes primary files; AUTOGEN assumes that primary files are on a cluster common system disk. AUTOGEN handles different types of input in the following ways:

1. If AUTOGEN does not receive user-supplied information from MODPARAMS.DAT, it performs default page and swap file size calculations. If no secondary files exist, AUTOGEN applies any changes to the primary files. If secondary files exist, AUTOGEN applies changes evenly across all secondary page or swap files, but does not modify primary files.
2. AUTOGEN can receive general user-supplied size information from MODPARAMS.DAT. This information consists of records of the form PAGEFILE = *n* or SWAPFILE = *n*. If *n* is zero, the corresponding section is skipped. If *n* is not zero and no secondary files exist, AUTOGEN applies the value to primary files. If *n* is not zero and secondary files exist, AUTOGEN applies any change evenly across all secondary files, but does not modify primary files.
3. You can now specify the individual sizes of all existing page and swap files (including secondary files), as well as the location and size of new files that you want AUTOGEN to create. To do this, define symbols in MODPARAMS.DAT using the following format:

   ```
   {PAGE/SWAP}FILEn_{NAME/SIZE}
   ```

   In this format, *n* is an integer that specifies the page or swap file. Refer to the primary page and swap files by specifying a value of 1 for *n*; refer to subsequent files by specifying increasingly higher integer values for *n*. For example, to refer to a secondary page or swap file, you could specify a value of 2 for *n*. Braces ({}) indicate that you must choose between the options delimited by a backslash (/). For example, specify PAGE or SWAP, NAME or SIZE.

For existing files, you typically define _SIZE symbols only; AUTOGEN already has the name and location. For example, to direct AUTOGEN to set the primary page file size to 10000 blocks, use the following symbol definition:

```
PAGEFILE1_SIZE = 10000
```

To direct AUTOGEN to create a new secondary swap file named PAGED$:[PAGESWAP]SWAPFILE.SYS that holds 30000 blocks, use the following symbol definitions:

```
SWAPFILE2_NAME = "PAGED$:[PAGESWAP]SWAPFILE.SYS"
SWAPFILE2_SIZE = 30000
```

Note that you must manually edit SYS$MANAGER:SYSTARTUP.COM to include a SYSGEN command that installs the newly created secondary file.

You cannot specify both general and explicit information as described in numbers 2 and 3 above. AUTOGEN issues a warning if conflicting symbol definitions exist in MODPARAMS.DAT.

If the creation or extension of a file would cause the target disk to become more that 95% full, AUTOGEN issues a warning and does not perform the operation.

For more information about AUTOGEN, see Section 11.1 of the *VAX/VMS System Manager's Reference Manual.*

2.7 Primary Page and Swap Files on Disks Other Than the System Disk

You can have primary page and swap files on disks other than the system disk.

Page and swap files in SYS$SYSTEM are installed automatically. If no page and swap files exist in SYS$SYSTEM, a message informs you that the files were not found. For example, you might receive the following message:

```
%SYSINIT-I-PAGEFILE.SYS not found - system initialization continuing...
```

To install page and swap files on any disk, create the file SYS$MANAGER:SYPAGSWPFILES.COM. In SYPAGSWPFILES.COM, specify the MOUNT and SYSGEN INSTALL commands that install page and swap files. The following example demonstrates some of the commands you might put in SYPAGSWPFILES.COM:

```
$ RUN SYS$SYSTEM:SYSGEN
INSTALL DISK_SYS2:[SYSTEM]PAGEFILE1.SYS /PAGEFILE
INSTALL DISK_SYS2:[SYSTEM]SWAPFILE1.SYS /SWAPFILE
```

Immediately before system overhead processes are created (for example, OPCOM and JOBCTL), STARTUP.COM searches for and executes SYPAGSWPFILES.COM.

After SYPAGSWPFILES.COM executes, control returns to STARTUP.COM. If no page files have been installed, STARTUP.COM returns the following error message:

```
%STARTUP-E-NOPAGFIL, no page files have been successfully installed.
```

Observe the following restrictions when you use SYPAGSWPFILES.COM:

- In order to use the primary page file for writing crash dumps, the primary page file must be located on the system disk.
- Disks mounted by SYPAGSWPFILES.COM must not be mounted by other processors during upgrades where SYSGEN parameter VAXCLUSTER is set to zero.

2.8 LOCKDIRWT SYSGEN Parameter

Version 4.6 and subsequent versions change the way the Lock Database is rebuilt when nodes are added or removed from a VAXcluster while SYSGEN parameter LOCKDIRWT is set to zero.

Prior to Version 4.6, setting LOCKDIRWT to zero on a VAXcluster node prevented that node from participating in the lock directory service unless all member nodes had LOCKDIRWT set to zero. Version 4.6 and subsequent versions retain the old behavior but include the following new feature: a node with LOCKDIRWT set to zero now gives other nodes an opportunity to become resource managers before reacquiring locks during a cluster state transition. This feature tends to move management of shared resources away from a node that has a zero value for LOCKDIRWT.

A zero value for LOCKDIRWT can be useful when the cluster contains a relatively low number of high-powered processors with a relatively high number of low-powered processors, where all processors access the same shared resources. Setting LOCKDIRWT to zero in this case tends to move resource management tasks to the larger processors and can reduce memory use on the smaller processors.

DIGITAL recommends you use the value for LOCKDIRWT that AUTOGEN assigns.

2.9 Monitor Utility Supports Monitoring of Additional Nodes

The Monitor Utility (MONITOR) now supports monitoring of the maximum number of nodes allowable in a cluster—currently 28.

However, insufficient process quotas of the following types can restrict the number of nodes you can monitor:

- ASTLM
- FILLM
- JTQUOTA

If necessary, use the Authorize Utility (AUTHORIZE) to increase the values for these quotas.

An insufficient value for maximum logical links can also restrict the number of nodes you can monitor. If necessary, use the Network Control Program (NCP) to increase the value for maximum links.

2.10 *VAX/VMS System Manager's Reference Manual* — Corrections

The sections that follow contain corrections to the *VAX/VMS System Manager's Reference Manual.*

2.10.1 Setting Up Queues for Spooled Line Printers

In Section 9.7.12 of the *VAX/VMS System Manager's Reference Manual,* "Guidelines for Setting Up Queues for Spooled Line Printers," Figures 9-6, 9-7, and 9-8 contain commands that do not conform to the procedure described in the preceding text.

The preferred method for setting up queues for spooled line printers is described in the text and not in the figures. This inconsistency will be eliminated in the next revision of the manual.

2.10.2 Bootstrapping a VAX 8200/8300/8350 from an HSC-Controlled Disk

This note provides information that was omitted from Sections 2.4 and 2.5 of the *VAX/VMS System Manager's Reference Manual.*

When you bootstrap your 8200/8300/8350 processor from a local (non-HSC) disk, no default bootstrap command procedure is required. When you bootstrap from an HSC-controlled disk, modify the CIBOO.CMD command procedure supplied on the console diskette and rename it DEFBOO.CMD. When the console diskette is in the console diskette drive and contains a file named DEFBOO.CMD, the processor uses DEFBOO.CMD to perform either of the following actions:[1]

- Reboot automatically
- Bootstrap the processor when you enter the BOOT command at the console mode prompt without specifying a device name

Use the following procedure to modify the CIBOO.CMD command procedure and rename it DEFBOO.CMD:

1 Be sure your console device is connected. If it is not, invoke SYSGEN and enter the following SYSGEN command to connect it:

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> CONNECT CONSOLE
SYSGEN> EXIT
```

2 Insert the console diskette into the console drive (CSA1:, the left-hand diskette drive).

3 Enter the following command to mount the console diskette:

```
$ MOUNT/FOREIGN CSA1:
```

[1] These actions occur assuming that the default boot descriptors in the processor's EEPROM are set to boot your system. An automatic reboot occurs only if the lower key switch on the system control panel is set to "Autostart."

4 Use the DXCOPY.COM command procedure to copy CIBOO.CMD to a disk directory. You cannot modify a file directly on the console diskette because of the way the diskette is formatted. Copy CIBOO.CMD to a disk directory as follows:

a. Enter the following command to invoke DXCOPY.COM command procedure:

```
$ @SYS$UPDATE:DXCOPY
```

b. Enter Y in response to the following prompt:

```
Is the system console storage medium mounted (Y/N)?: Y
```

c. Enter Y in response to the following prompt:

```
Copy from console medium (Y/N)?: Y
```

d. Enter CIBOO.CMD in response to the following prompt:

```
Name of file to be copied?: CIBOO.CMD
$
```

e. DXCOPY copies the file to your default directory and exits to DCL command level.

5 When the DCL prompt appears, use a text editor to edit CIBOO.CMD. Originally, this file contained the following text:

```
!CIBOO.CMD   :Boot command file to boot a VAX 8200/8300/8350 from an HSC disk.
!
!
!    Note "n", "p" (and "q"), "u", and "r" are single hexadecimal characters
!
D/G 0 20                ! CI Port Device Type Code
!D/G 1 n                ! n = CI adapter's VAXBI node number
!D/G 2 p                ! Use the HSC controller at CI node p
!D/G 2 0p0q             ! Use either the HSC controller at CI nodes p and q
!D/G 3 u                ! u = Disk drive unit number
D/G 4 0                 ! Boot Block LBN (not used)
!D/G 5 r0000000         ! r = system root [SYSR...], Software boot flags
D/G E 200               ! Address of Working Memory+^X200
LOAD VMB.EXE/START:200  ! Load Primary Bootstrap
START 200               ! Start Primary Bootstrap
```

Edit CIBOO.CMD as follows. All numbers you insert in this file are in hexadecimal radix.

a. Delete the comment character (!) that appears before the D/G 1 command and replace **n** with the VAXBI node number of the CI adapter.

b. If your processor is connected to one HSC controller, delete the comment character (!) that appears before the first D/G 2 command and replace **p** with the HSC controller number. If your processor is connected to two HSC controllers, delete the comment character that appears before the second D/G 2 command and replace **p** with the VAXBI node number of the first HSC and replace **q** with the VAXBI node number of the second HSC. Note that you can delete the comment character from only one of these commands.

c. Delete the comment character (!) that appears before the D/G 3 command and replace **u** with the unit number of the HSC disk from which you will bootstrap the VAX/VMS operating system.

   d. Delete the comment character (!) that appears before the D/G 5 command and replace **r** with the number of the system root from which you will bootstrap the VAX/VMS operating system. By default, the VAX/VMS operating system is stored in system root 0.

   e. Exit from the text editor.

6 Enter the following command to rename CIBOO.CMD to DEFBOO.CMD.

```
$ RENAME CIBOO.CMD DEFBOO.CMD
```

7 Use the DXCOPY.COM command procedure as follows to copy DEFBOO.CMD to the console diskette:

   a. Enter the following command to invoke the DXCOPY.COM command procedure:

   ```
   $ @SYS$UPDATE:DXCOPY
   ```

   b. Enter Y in response to the following prompt:

   ```
   Is the system console storage medium mounted (Y/N)?: Y
   ```

   c. Enter N in response to the following prompt:

   ```
   Copy from console medium (Y/N)?: N
   ```

   The negative response tells DXCOPY you want to copy the file from your default directory to the console storage medium.

   d. Enter DEFBOO.CMD in response to the following prompt:

   ```
   Name of file to be copied?: DEFBOO.CMD
   $
   ```

   DXCOPY copies DEFBOO.CMD to the console volume and exits to DCL command level.

8 Dismount and remount the console diskette using the following commands:

```
$ DISMOUNT CSA1:
$ MOUNT CSA1:
```

You have successfully created a default bootstrap command procedure, DEFBOO.CMD, on the console diskette.

2.10.3 Alternate, Nonstop, Bootstrap Procedure for a VAX 8200/8300/8350

The following text adds information to that given in Section 4.2.2 of the *VAX/VMS System Manager's Reference Manual*:

You normally use an alternate, nonstop, bootstrap command procedure when the default bootstrap procedure cannot be accessed because of problems with the device designated in the default procedure. To bootstrap the system using an alternate, nonstop procedure, follow these steps:

1 Halt the processor. You halt a VAX 8200/8300/8350 system by pressing CTRL/P.

2 Bootstrap the system, by entering the following command:

```
>>> B ddnu
```

The code *dd* is the device type, *n* is the VAXBI node number, and *u* is the unit number for the disk on which the alternate bootstrap procedure resides.

2.11 Creating a Command Procedure to Boot Standalone BACKUP from an Alternate System Root

Modify the text and table found in list item 5 on page 4-41 of the Version 4.4 *Guide to VAX/VMS Software Installation.*

On the line that begins with either the command DEPOSIT R5 or D/G 5, change the left-most digit of the number following this command to an E. For example, on a VAX–11/782, change the line that says DEPOSIT R5 4000nnnn, where the n's represent hexadecimal digits, to DEPOSIT R5 E000nnnn. On a VAX 8200, change the line that says D/G 5 0nnnnnnn to D/G 5 Ennnnnnn, where the n's represent hexadecimal digits. A different procedure, which is described in the *VAX 8800/8700/8550/8500 Console User's Guide,* is used for the VAX 8800/8700/8550/8530 processors.

2.12 Use VMSINSTAL to Install Optional Software Products

The VMSUPDATE command procedure, described in Appendix C of the Version 4.4 *Guide to VAX/VMS Software Installation,* should *not* be used to install optional software products. Instead, use the VMSINSTAL command procedure, described in Chapter 5 of the *Guide to VAX/VMS Software Installation.*

VMSUPDATE is not supported on tailored systems, cluster configurations that share a common system disk, or on the VAX 8600, VAX 8650, VAX 8800, VAX 8700, VAX 8550, and VAX 8530 processors.

2.13 Using Volume Shadowing on a VAX 8800/8700/8530/8550/Processor

Section 3.3.1 of the *VAX 8800/8700/8550/8500 Operations Guide* describes the procedure for modifying the template bootstrap command procedures BCI*aaa*.COM, BDA*aaa*.COM, and UDA*aaa*.COM (where *aaa* is BOO, GEN, or XDT) to bootstrap your processor. If volume shadowing is installed on your processor, modify the bootstrap command procedures to deposit the values listed in Table 2–3 in R3.

Table 2–3 Values to Deposit in R3 in the Bootstrap Command Procedures

Bit Position	Possible Values	Meaning
<31:24>	80_{16}	Shadow set indicator
<23:16>	*uu*	Shadow unit number (DUS*uu*)
<15:00>	*uu*	Unit number of booting member of shadow set

Modify registers R0 through R2 and R4 through R5 to contain the values listed in Table 3-2 of the *VAX 8800/8700/8550/8500 Operations Guide.* See the *VAX/VMS Volume Shadowing Manual* for more information about volume shadowing.

2.14 *VAX/VMS Authorize Utility Reference Manual* — Corrections and Additions

Make the following corrections to the *VAX/VMS Authorize Utility Reference Manual*. These corrections will be incorporated in the next revision of the manual.

- Page AUTH-2 — The summary of AUTHORIZE commands should include the following qualifiers:

  /ASTLM (for the ADD command)
  /GENERATE_PASSWORD (for the MODIFY command)

- Page AUTH-11 — In Table AUTH-2, the /FLAG=[NO]PWDEXPIRED function should read /FLAG=[NO]PWD_EXPIRED. Please include the underscore (_).

- Page AUTH-13 — The /PWDEXPIRED and /PWDLIFETIME qualifiers should appear as /[NO]PWDEXPIRED and /[NO]PWDLIFETIME, respectively.

- Page AUTH-14 — In the description of the /UIC qualifier, the documentation states that the value of the member number must be in the range of 0–1777776. The correct range is 0–177776.

- Pages AUTH-21, AUTH-37, AUTH-42 — The documentation states that the rights identifier values must be in the range 32,768 to 268,435. Note that user-defined identifiers must be in the range 65,536 to 268,435,455. Identifier values of less than 65,536 are reserved.

- Tables AUTH-2 and AUTH-4 — The recommended values for process resource limits should read as follows:

Limit	Value
ASTLM	24
BIOLM	18
BYTLM	8192
ENQLM	30
PGFLQUOTA	12800
WSDEFAULT	200
WSQUOTA	500
WSEXTENT	1000

2.14.1 Error Messages

The Authorize Utility has the following error messages that have not previously been documented:

BADNODFORM, improper node::remoteuser format

Facility: AUTHORIZE, Authorize Utility

Explanation: You specified the format for the remote node and user incorrectly. The correct format consists of a node name, a pair of colons, and the user name of the remote user. A node name may consist of 1–6 alphanumeric characters and must contain at least one alphabetic character. If you use a wildcard character for either the node or user, you must still include the colons.

User Action: Reenter your command with the correct format.

BADUSR, username does not exist

Facility: AUTHORIZE, Authorize Utility

Explanation: The user name you specified does not exist in the system user authorization file (SYSUAF.DAT).

User Action: Correct the user name and reenter your command. You can display the records in the user authorization file by using the AUTHORIZE command SHOW.

CLIWARNMSG, Warning: /CLITABLES field may need to reflect changes to /CLI field

Facility: AUTHORIZE, Authorize Utility

Explanation: If you modify the command language interpreter (CLI) field of a record in the system user authorization file, you may have to modify the CLITABLES field to reflect the change. If you have set the CLI field to DCL or MCR, however, the CLITABLES field defaults to the correct value.

User Action: If you have changed the CLI field to a value other than DCL or MCR, use the AUTHORIZE command MODIFY/CLITABLES to set the CLITABLES field to the corresponding tables. Refer to the description of the LOGIN Procedure in the *VAX/VMS DCL Dictionary* for further information about specifying CLI tables.

CMDTOOLONG, command line exceeds maximum length

Facility: AUTHORIZE, Authorize Utility

Explanation: The length of your command, after any symbols and logical names have been expanded, exceeds the maximum allowable length.

User Action: Reenter a shorter form of the command.

EXTRAPARM, superfluous parameter detected

Facility: AUTHORIZE, Authorize Utility

Explanation: You have specified too many parameters in the command line. The extra parameter is identified in the message.

User Action: Reenter your command without the excess parameter.

GRANTERR, unable to grant identifier 'id-name' to 'user name'

Facility: AUTHORIZE, Authorize Utility

Explanation: The specified identifier cannot be granted to the specified user. This message should be accompanied by a second message showing the specific reason why the identifier could not be granted.

User Action: Correct the condition identified by the second message and reenter your command.

GRANTMSG, identifier 'id-name' granted to 'user name'

Facility: AUTHORIZE, Authorize Utility

Explanation: The specified general identifier has been granted to the specified user. The user has access to all of the rights associated with the identifier.

User Action: None.

HELPERR, error finding or outputting HELP information

Facility: AUTHORIZE, Authorize Utility

Explanation: An error occurred trying to access the AUTHORIZE HELP file.

User Action: Check that the AUTHORIZE HELP file—by default named UAFHLP.HLB—is located in the proper directory and is not protected against read access.

IDOUTRNG, identifier value is not within legal range

Facility: AUTHORIZE, Authorize Utility

Explanation: The value you specified for an identifier is not within the permissible range. A general identifier may have an integer value between 32,768 and 268,435,455. A UIC identifier takes a value in standard UIC format.

User Action: Reenter your command with an identifier value that is within the permissible range.

INVCMD, invalid command

Facility: AUTHORIZE, Authorize Utility

Explanation: The command you have entered is not a valid AUTHORIZE command.

User Action: Refer to the *VAX/VMS Authorize Utility Reference Manual* for a description of the command you are trying to use and then reenter the command correctly.

INVUSERNAME, username syntax error

Facility: AUTHORIZE, Authorize Utility

Explanation: The user name you specified is invalid due to incorrect syntax. If you are adding a new user name to the system user authorization file with the AUTHORIZE command ADD, the new user name may be 1–12 alphanumeric characters, and it may include underscores.

User Action: Correct the user name and reenter your command.

INVUSERSPEC, error in user specification

Facility: AUTHORIZE, Authorize Utility

Explanation: Your command included an incorrect user specification. In a user specification, you can use a numeric UIC format (for example, [007,007]), an alphanumeric format (for example, [COMPOSERS,HAYDN]), or a user name (for example, HAYDN). You can use wildcards to specify multiple users. Refer to the *VAX/VMS Authorize Utility Reference Manual* for specific syntax rules for the command you are using.

User Action: Correct the user specification and reenter your command.

NAFADDERR, unable to add record to NETUAF.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The record you specified could not be added to the network user authorization file (NETUAF.DAT). This message should be accompanied by a VAX RMS message that identifies the specific reason for the error. For example, this error occurs if you try to add a record authorizing a remote user to access more than one local account. Each user at a remote node is allowed access to the files of only one user on the local node.

User Action: If possible, correct the condition identified by the RMS message and reenter your command. Otherwise, examine the network user authorization file to determine why the record could not be added. You can display the contents of the file by using the AUTHORIZE command SHOW/PROXY. You can write the contents of NETUAF.DAT to a listing file by using the AUTHORIZE command LIST/PROXY. If you want to delete a current record from NETUAF.DAT in order to add a new one, use the AUTHORIZE command REMOVE/PROXY.

NAFAEX, NETUAF.DAT already exists

Facility: AUTHORIZE, Authorize Utility

Explanation: A network user authorization file (NETUAF.DAT) already exists for the local node.

User Action: If you want to create a new network user authorization file, either delete or rename the current one (if you have sufficient privilege to do so). Once the current file has been deleted or renamed, reenter the AUTHORIZE command CREATE/PROXY. Note that you must have sufficient privilege to create a new file.

NAFCREERR, unable to create NETUAF.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: A network user authorization file (NETUAF.DAT) could not be created. This message should be accompanied by a VAX RMS message that identifies the specific reason why the file could not be created. For example, this error occurs if you do not have sufficient privilege to create the file.

User Action: Correct the condition identified by the RMS message and reenter your command.

NAFDNE, NETUAF.DAT does not exist

Facility: AUTHORIZE, Authorize Utility

Explanation: A network user authorization file (NETUAF.DAT) does not exist on the local node.

User Action: If you have sufficient privilege, use the AUTHORIZE command CREATE/PROXY to create a network user authorization file. Then you can add records to the file by using the AUTHORIZE command ADD/PROXY.

NAFDONEMSG, network authorization file modified

Facility: AUTHORIZE, Authorize Utility

Explanation: The network user authorization file (NETUAF.DAT) has been modified to reflect the change directed by your command.

User Action: None.

NAFNOMODS, no modifications made to network authorization file

Facility: AUTHORIZE, Authorize Utility

Explanation: No modifications have been made to the network user authorization file (NETUAF.DAT).

User Action: None.

NAFUAEERR, entry already exists in NETUAF.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The record you have tried to add to the network user authorization file is already in the file; it has not been duplicated.

User Action: None.

NAONAF, unable to open NETUAF.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The network user authorization file (NETUAF.DAT) could not be opened. This message should be accompanied by a VAX RMS message that identifies the specific reason for the error. Possible reasons are insufficient privilege, file protection violation, or location of the file in the wrong directory.

User Action: If you do not have sufficient privilege to open NETUAF.DAT, there is nothing you can do except to ask a privileged user, such as your system manager, to access the file for you. If you do have sufficient privilege, make sure the file is located in the proper directory and is not protected against read or write access. Then reenter your command.

NETLSTMSG, listing file NETUAF.LIS complete

Facility: AUTHORIZE, Authorize Utility

Explanation: The contents of the network user authorization file (NETUAF.DAT) have been written to the listing file named NETUAF.LIS.

User Action: None.

NOARG, missing argument for option

Facility: AUTHORIZE, Authorize Utility

Explanation: You specified a qualifier without a required argument.

User Action: Reenter your command and include the required argument.

NODTOOBIG, node name too long

Facility: AUTHORIZE, Authorize Utility

Explanation: VAX/VMS node names cannot exceed six characters. A node name may consist of 1–6 alphanumeric characters; at least one character must be alphabetic.

User Action: Check the node name and reenter your command with the correct name.

NOGRPWILD, wildcard group numbers not allowed

Facility: AUTHORIZE, Authorize Utility

Explanation: Wildcard characters are not allowed in the UIC group number field for the command you entered.

User Action: Reenter your command with a specific UIC group number instead of a wildcard character.

NOIDNAM, no ID name was specified

Facility: AUTHORIZE, Authorize Utility

Explanation: The command you entered requires that you include an identifier name.

User Action: Check the *VAX/VMS Authorize Utility Reference Manual* for the syntax rules regarding identifier names for the command you want to use. Then reenter the command including an identifier name.

NOTIDFMT, id name parameter does not translate to ID format

Facility: AUTHORIZE, Authorize Utility

Explanation: The identifier name that you specified does not translate to a corresponding value in general identifier format. Identifier name values translate to either general identifier format or UIC format. General identifier names can be 1 to 31 alphanumeric characters and are stored with an integer value in the range of 32,768 to 268,435,455. General identifiers are created by the AUTHORIZE command ADD/IDENTIFIER.

When you use the AUTHORIZE command GRANT/IDENTIFIER, the first identifier you specify must be in general identifier format. In other words, you cannot grant a UIC format identifier to another UIC format identifier.

User Action: Determine why the identifier name does not translate. You can display an identifier name and its corresponding value with the AUTHORIZE command SHOW/IDENTIFIER. To change the value of an identifier name, use the AUTHORIZE command MODIFY/IDENTIFIER.

NOTUICFMT, user id parameter does not translate to UIC format

Facility: AUTHORIZE, Authorize Utility

Explanation: The user specification in your command does not translate to a UIC identifier (an identifier in UIC format).

User Action: Determine why the user specification does not translate. You can display user names and their corresponding UIC values by using the AUTHORIZE command SHOW.

NOUSERNAME, missing username

Facility: AUTHORIZE, Authorize Utility

Explanation: The command you are using requires a user name. A user name is the member name from the alphanumeric form of a user's UIC (user identification code).

User Action: Reenter your command and include a user name.

NOUSERSPEC, missing user specification

Facility: AUTHORIZE, Authorize Utility

Explanation: The command you are using requires a user specification. A user specification can be a user name (for example, CAESAR), or a user identification code (for example, [100,44]).

User Action: Reenter your command and include a user specification.

PREMMSG, record removed from NETUAF.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The record specifed in the AUTHORIZE command REMOVE/PROXY has been removed from the network user authorization file.

User Action: None.

PWDNCH, password not changed

Facility: AUTHORIZE, Authorize Utility

Explanation: An error occurred using the random password generator to generate an account password.

User Action: None.

PWDNOL, password not on list; try again

Facility: AUTHORIZE, Authorize Utility

Explanation: The password you specified was not one of those listed.

User Action: Select another password and try again.

RDBADDERR, unable to add 'id-name' to RIGHTSLIST.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The identifier name you specified could not be added to the rights database file (RIGHTSLIST.DAT). This message should be accompanied by a VAX RMS message that identifies the specific reason for the error. Most likely, the identifier name already exists in the rights database file. Duplicate identifier names are not allowed in the rights database file.

User Action: Correct the condition identified by the RMS message and reenter your command. If you want to change the name of an identifier in the rights database file, use the AUTHORIZE command MODIFY/IDENTIFIER.

RDBADDERRU, unable to add 'id-name' value: '[UIC]' to RIGHTSLIST.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The specified identifier name and its corresponding user identification code (UIC) could not be added to the rights database file (RIGHTSLIST.DAT). This message should be accompanied by a VAX RMS message that identifies the specific reason for the error. Most likely, the identifier name already exists in the rights database file. Duplicate identifier names are not allowed in the rights database file.

This error also occurs if you copy a record in the system user authorization file (SYSUAF.DAT) without specifying a new UIC value for the copy. By default, an identifier name and corresponding UIC value for the new record are written to the rights database file (RIGHTSLIST.DAT); if the UIC has not been changed, it conflicts with the UIC of the original record, and a 'duplicate identifier' error results.

User Action: Correct the condition identified by the RMS message and reenter your command. If you want to change the UIC value of an identifier in the rights database file, use the /VALUE qualifier with the AUTHORIZE command MODIFY/IDENTIFIER. If you copy a record in the system user authorization file and you want an identifier for the new record to be added to the rights database file, use the /UIC qualifier with the AUTHORIZE command COPY.

RDBADDERRV, unable to add 'id-name' value: 'hex code' to RIGHTSLIST.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The specified identifier name and its corresponding integer value (expressed as an 8-bit hexadecimal code) could not be added to the rights database file (RIGHTSLIST.DAT). This message should be accompanied by a VAX RMS message that identifies the specific reason for the error. Most likely, the identifier name or value already exists in the rights database file. Duplicate identifier names or values are not allowed in the rights database file.

User Action: Correct the condition identified by the RMS message and reenter your command. If you want to change the value of an identifier in the rights database file, use the /VALUE qualifier with the AUTHORIZE command MODIFY/IDENTIFIER.

RDBADDMSG, identifier 'id-name' value: 'hex code' added to RIGHTSLIST.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: A general identifier with the specified name and value has been added to the rights database file (RIGHTSLIST.DAT).

User Action: None.

RDBADDMSGU, identifier 'id-name' value: '[UIC]' added to RIGHTSLIST.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: A UIC identifier with the specified name and value has been added to the rights database file (RIGHTSLIST.DAT).

User Action: None.

RDBCREERR, unable to create RIGHTSLIST.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The rights database file, named RIGHTSLIST.DAT, could not be created. This message should be accompanied by a VAX RMS message that identifies the specific reason for the error. For example, you cannot create another rights database file if one already exists, unless you first delete or rename the original file.

User Action: Correct the condition identified by the RMS message and reenter your command. If you want to create a new rights database file, either delete or rename the current one (if you have sufficient privilege to do so). Once the current file has been deleted or renamed, reenter your command.

RDBDONEMSG, rights database modified

Facility: AUTHORIZE, Authorize Utility

Explanation: The rights database file (RIGHTSLIST.DAT) has been modified.

User Action: None.

RDBMDFYERR, unable to modify identifier 'id-name'

Facility: AUTHORIZE, Authorize Utility

Explanation: The specified identifier could not be modified. This message should be accompanied by a VAX RMS message that identifies the specific reason for the error.

User Action: Correct the condition identified by the RMS message and reenter your command.

RDBMDFYERRU, unable to modify identifier '[UIC]'

Facility: AUTHORIZE, Authorize Utility

Explanation: The specified UIC identifier could not be modified. This message should be accompanied by a VAX RMS message that identifies the specific reason for the error.

User Action: Correct the condition identified by the RMS message and reenter your command.

RDBMDFYMSG, identifier 'id-name' modified

Facility: AUTHORIZE, Authorize Utility

Explanation: The record for the specified identifier in the rights database file has been modified according to the AUTHORIZE command MODIFY/IDENTIFIER.

User Action: None.

RDBNOMODS, no modifications made to rights database

Facility: AUTHORIZE, Authorize Utility

Explanation: The rights database file (RIGHTSLIST.DAT) was not modified.

User Action: None.

RDBREMERR, unable to remove 'id-name' from RIGHTSLIST.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The specified identifier could not be removed from the rights database file (RIGHTSLIST.DAT). This message should be accompanied by a VAX RMS message that identifies the specific reason for the error.

User Action: Correct the condition identified by the RMS message and reenter your command.

RDBREMMSG, identifier 'id-name' value: 'hex code' removed from RIGHTSLIST.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The general identifier with the specified name and hexadecimal value has been removed from the rights database file (RIGHTSLIST.DAT).

User Action: None.

RDBREMMSGU, identifier 'id-name' value: '[UIC]' removed from RIGHTSLIST.DAT

Facility: AUTHORIZE, Authorize Utility

Explanation: The UIC identifier with the specified name and user identification code has been removed from the rights database file (RIGHTSLIST.DAT).

User Action: None.

REVOKEERR, unable to revoke identifier 'id-name' from 'user name'

Facility: AUTHORIZE, Authorize Utility

Explanation: The specified identifier could not be revoked from the specified user.

User Action: Make sure that the user has been granted the identifier you are trying to revoke. Use the AUTHORIZE commands SHOW/IDENTIFIER /FULL or LIST/IDENTIFIER/FULL to display an identifier and the users who hold it.

REVOKEMSG, identifier 'id-name' revoked from 'user name'

Facility: AUTHORIZE, Authorize Utility

Explanation: The specified identifier has been revoked from the specified user. The user no longer has the rights associated with the identifier.

User Action: None.

RLSTMSG, listing file RIGHTSLIST.LIS complete

Facility: AUTHORIZE, Authorize Utility

Explanation: The contents of the rights database file (RIGHTSLIST.DAT) have been written to the listing file named RIGHTSLIST.LIS.

User Action: None.

SHOWERR, unable to complete show command

Facility: AUTHORIZE, Authorize Utility

Explanation: The AUTHORIZE command SHOW could not be completed. This message should be accompanied by a VAX RMS message that identifies the specific reason for the error.

User Action: Correct the condition identified by the RMS message and reenter your command.

SYSMSG2, Error code 'hex code' not found

Facility: AUTHORIZE, Authorize Utility

Explanation: The $GETMSG system service could not find a corresponding message for the specified error code, which probably indicates that the code is incorrect. Since an incorrect error code obviously should not be generated, this message probably indicates an internal software error.

User Action: Submit a Software Performance Report (SPR) that describes the conditions leading to the error.

WLDNOTALWD, wild card user specs not allowed

Facility: AUTHORIZE, Authorize Utility

Explanation: Wildcard characters are not allowed in the user specification for the command you are using.

User Action: Reenter your command without using wildcard characters.

ZZPRACREN, proxies to 'user name' renamed

Facility: AUTHORIZE, Authorize Utility

Explanation: Proxy access records for the specified user have been renamed to the new user name. When a user name in the system user authorization file (SYSUAF.DAT) is renamed, any records in the network authorization file (NETUAF.DAT) for the original user name are automatically renamed to the new user name.

User Action: None.

ZZSYSPWD, system password modified

Facility: AUTHORIZE, Authorize Utility

Explanation: The system password has been changed to the password directed by your command.

User Action: None.

2.14.2 New /ATTRIBUTES Keyword

The Authorize Utility has a new keyword for the /ATTRIBUTES qualifier. You can specify the [NO]DYNAMIC keyword with the following commands:

ADD/IDENTIFIER/ATTRIBUTES
GRANT/IDENTIFIER/ATTRIBUTES
MODIFY/IDENTIFIER/ATTRIBUTES

Specifying the [NO]DYNAMIC keyword indicates whether unprivileged holders of the identifiers may add or remove them from the process rights list. The default is NODYNAMIC.

This information will be added to the *VAX/VMS Authorize Utility Reference Manual* in a future release.

2.14.3 Enhanced /ACCESS Qualifier

The syntax string for the /ACCESS qualifier to the MODIFY command has been enhanced to allow more readable, flexible usage. The following commands produce identical results:

```
UAF> MODIFY SAM /ACCESS=(primary, 2-3, 5, secondary, 8-12)
UAF> MODIFY SAM /ACCESS="Primary: 2-3, 5; Secondary: 8-12"
UAF> MODIFY SAM /ACCESS=(p,2,s,8,p,3,s,9,p,5,s,10-12)
UAF> MODIFY SAM /ACCESS="2-3 SEC 8-12 PRIM 5"
```

2.14.4 /DEFPRIVILEGES and /PRIVILEGES Qualifiers

You can specify the keyword [NO]ALL for the /DEFPRIVILEGES and /PRIVILEGES qualifiers to disable/enable all user privileges.

2.14.5 Secondary Passwords — Change

Beginning with Version 4.2, users cannot initially give themselves secondary passwords. The initial setting of the secondary password must be done by the system manager using the Authorize Utility. The reason for this change is to protect careless users who leave their terminal sessions unattended.

In earlier versions of VAX/VMS, anyone could render an account useless by simply adding a secondary password that the account's owner did not know. If a user now tries to initiate a secondary password, the system will respond as follows:

```
$ SET PASSWORD/SECONDARY
%SET-F-PWD2NOTSET, system manager must initially set secondary passwords
```

2.14.6 New AUTOLOGIN Flag

A flag named AUTOLOGIN has been added to the flags field in the user authorization file (SYSUAF). The flag is set by specifying the qualifier /FLAGS=AUTOLOGIN to one of the following Authorize Utility commands: ADD, MODIFY, or COPY. When set, it makes the account available only by using the autologin mechanism. The following forms of access are disabled:

- Login by any terminal, LAT connection, or SET HOST involving presentation of username and password
- Access by DECnet task using explicit access control

The following forms of access remain permitted:

- Interactive login by the autologin mechanism
- Batch jobs
- Proxy access by DECnet task

2.15 *Guide to Multiprocessing on VAX/VMS* — Setting Up a VAX–11/782

This section supplements the *Guide to Multiprocessing on VAX/VMS* and provides instructions for building multiprocessing console diskettes on a VAX–11/782 system. The information in this section assumes the following:

- The VAX–11/782 hardware has already been installed and configured.
- The VAX/VMS operating system has been installed as described in the installation booklet packaged with the media (for example, *Installing VAX/VMS on a VAX-11/780 From Magnetic Tape*).

Note that you should keep a record of the following information regarding memory configuration:

- The number and type of memory controllers
- The transmit request (TR) levels at which the controllers are configured
- The amount of memory on each controller

Note: **The command procedure BOOTBLDR.COM does not recognize MS780-H memory. If your system configuration includes an MS780-H, contact your DIGITAL Customer Support Center or submit an SPR.**

2.15.1 Building Multiprocessing Console Diskettes

Each processor in the VAX–11/782 system must have its own console diskette. Multiprocessing console diskettes allow for the booting of the VAX–11/782 attached processor system by means of several boot command procedures. These bootstrap command procedures cause MA780 shared memory rather than local memory to be used as main memory, and they set the memory configuration registers to ensure that MA780 shared memory is configured at the low physical addresses (beginning at 0) and local memory at the higher addresses.

In addition, each multiprocessing console diskette contains the "reset memory" command procedure RMEM.COM, which is specific to the memory configuration of the processor. The RMEM.COM procedure reconfigures local memory to start at physical address 0 (zero) and MA780 shared memory to start at adjacent higher physical addresses. Thus, after executing RMEM.COM, you can boot the VAX–11/782 system as a single-processor VAX–11/780 system by using a standard VAX–11/780 console diskette (providing you have sufficient local memory on the system).

To build multiprocessing console diskettes, you execute the interactive command procedure SYS$UPDATE:BOOTBLDR.COM. This command procedure first creates a new console diskette for the primary processor and then one for the attached processor. The procedure executes interactively and prompts you for information about the memory configuration of the system.

The sections that follow describe how to obtain information about the memory configuration of the system and how to execute BOOTBLDR.COM.

2.15.1.1 Determining the Memory Configuration

In order to run BOOTBLDR.COM, you must determine certain information about the configuration of your system. For each memory controller on the system, you need to know the following:

- Its type (MS780-A, MS780-C, MS780-E, or MA780)
- The transmit request (TR) level at which it is configured
- The amount of memory it holds

In addition, you need to know the TR level of the first UNIBUS and MASSBUS adapter on the system (that is, the UBA and the MBA with the lowest TR number). You need the information regarding MS780-x memory and MA780 memory for both the primary and attached processors; information regarding UNIBUS and MASSBUS adapters is needed only for the primary processor. (Note that not all VAX 11/782 systems are configured with a MASSBUS adapter.)

You can obtain the necessary information from your DIGITAL Field Service Representative or by following the procedure in this section. If you already know the memory configuration of your system, proceed to Section 2.15.3.

This section describes how to obtain this information, first for the primary processor and then for the attached processor. The procedure is the same in both cases, with the following exceptions:

- For the primary processor, perform the procedure at the primary processor's console terminal; for the attached processor, at the attached processor's console terminal.
- You need to determine the TR level and memory amount of each MA780 controller only once (for the primary processor), since an MA780 memory controller must be configured at the same TR level on both processors. Note, however, that obtaining this information twice (for both processors) allows you to check whether MA780 memory has been configured correctly (is at the same TR level) on both processors.

You can determine the memory configuration of each processor by examining the configuration registers for TR levels 1 through 15. Memory controllers can be configured only at TR levels 1 through 6; UNIBUS and MASSBUS adapters can be configured at any TR level (1 through 15). TR level 0 is reserved for the CPU and is not of interest to BOOTBLDR.COM.

Table 2–4 shows the physical addresses for the configuration registers at each TR level. Table 2–5 shows the codes for each type of adapter.

Table 2–4 Configuration Register Physical Addresses

Transmit Request (TR) Level	Configuration Register (CR) Physical Address	MA780 Port Invalidation Configuration Register (PICR) Physical Address
1	20002000	2000200C
2	20004000	2000400C
3	20006000	2000600C
4	20008000	2000800C
5	2000A000	2000A00C
6	2000C000	2000C00C
7	2000E000	
8	20010000	
9	20012000	
10	20014000	
11	20016000	
12	20018000	
13	2001A000	
14	2001C000	
15	2001E000	

Table 2–5 Adapter Type Codes

Value	Adapter Type	Scale Factor
08,09	MS780-A	64KB
10,11	MS780-C	64KB
20	MBA (MASSBUS adapter)	n/a
28-2B	UBA (UNIBUS adapter)	n/a
40-43	MA780	256KB
68-6A	MS780-E	1024KB
70-74	MS780-H	n/a
Other	not of interest to BOOTBLDR.COM	

To determine the memory configuration on your system, perform the following steps:

1 Put the console terminal into console mode by pressing CTRL/P.

2 Enter the console command HALT at the console-mode prompt (> > >) to halt the processor.

3 Use the console command EXAMINE to read the appropriate configuration registers for each TR level (Table 2–4 shows the TR levels and their corresponding registers). For example, the following command reads the configuration register at TR level 1 (TR1):

```
>>> EXAMINE 20002000
P 20002000 00002610
```

If the specified TR level has no adapter at all, you will see a display like the following, where *n* represents a hexadecimal digit:

```
>>> EXAMINE 2000E000
? MIC-ERR ON FUNCTION
(nn)nnnnnnnn (nn)nnnnnnnn (nn)nnnnnnnn
(nn)nnnnnnnn (nn)nnnnnnnn (nn)nnnnnnnn
(nn)nnnnnnnn (nn)nnnnnnnn (nn)nnnnnnnn
```

The EXAMINE command in this example attempts to read the configuration register for TR level 7. Because there is no adapter at TR 7, the microcode returns an error after trying to read the nonexistent CR.

4 Using Table 2–5 for reference, interpret the displayed value to determine the type of adaptor connected to the designated TR level. To do this, extract the two rightmost digits from the configuration register (CR) display and match them with an entry listed under "Value" in the table. (Note that the table only shows the values of interest to BOOTBLDR.COM.)

For example, the CR value displayed by the EXAMINE command in step 3 is 00002610. The two rightmost digits are 10. According to Table 2–5, the value 10 designates an MS780-C memory controller containing 64-kilobyte memory boards.

5 Depending upon the type of adapter (MS780, MA780, UNIBUS, or MASSBUS), perform the following steps:

— MS780 memory (all types except MS780-H)

a. Extract the fifth and sixth (from the left) digits from the CR value. (In the example shown in step 3, these digits are 26.)

b. Convert this number from hexadecimal to decimal.

c. Divide the converted number by 2, discarding the remainder, and add 1 to the result.

d. Multiply the result by the scale factor (shown in Table 2–5) to determine the total amount of memory.

e. Record the TR number, memory type, and the amount of memory for later use.

Note that the BOOTBLDR.COM procedure does not recognize MS780-H memory. If your configuration includes this type of memory, contact your DIGITAL Customer Support Center or submit an SPR.

— MA780 memory

a. Examine the contents of the MA780 Port Invalidation Configuration Register (PICR).

b. Extract the fourth digit from the left and add 1 to that value. (There is no need to convert to decimal, as the value is never greater than 7.)

c. Multiply the result by the scale factor (shown in Table 2–5) to determine the total amount of memory.

d. Record the TR number, memory type, and the amount of memory for later use.

— UNIBUS or MASSBUS adapters (UBA or MBA)

Make note of the TR number of the first (lowest TR number) of each type.

A memory controller can be configured only at TR levels 1 through 6. However, a UBA or MBA can be configured at any TR level from 1 through 15. Therefore, you should examine the locations that correspond to TR levels 7 through 15 to determine whether a UBA or MBA is configured at any of them. If the two rightmost digits of the displayed value are in the range 28 through 2B, a UBA is configured at the TR level corresponding to the examined address. If the two rightmost digits of the displayed value are 20, an MBA is configured at the TR level corresponding to the examined address.

Note that, if the rightmost two digits of the value displayed by any EXAMINE command are not in the ranges shown in Table 2–5, the TR level corresponding to the examined address does not have a memory controller, UBA, or MBA. In this case, whatever is configured at that particular TR level is not of concern, and you need not further interpret the displayed value. Further, if a microcode error results when you examine any of the addresses, simply assume that a device is not configured at the TR level corresponding to the examined address.

6 Once you have completed the preceding steps, you have determined the memory configuration for the primary processor. To determine the memory configuration of the attached processor, repeat steps 1 through 4 at the attached processor's console terminal. That is, put the terminal in console mode, enter the EXAMINE commands for TR levels 1 through 6, and interpret the displayed values.

Again, you need not obtain information about MA780 memory a second time, since information about MA780 memory is identical for both the primary and attached processors. Thus, you need not examine one of the six addresses if an examination of that address at the primary processor's console terminal revealed an MA780 memory controller. However, it might be useful to examine all addresses in order to verify that your system is configured properly.

Once you have completed steps 1 through 5, you have determined the memory configuration of the system. You have all the information needed to execute BOOTBLDR.COM. Proceed to the next section.

2.15.1.2 Executing BOOTBLDR.COM

BOOTBLDR.COM is an interactive command procedure that builds console diskettes for the primary and attached processors. You must execute BOOTBLDR.COM when you initially set up your VAX–11/782 system and whenever you change its memory configuration. The multiprocessing console diskettes created by BOOTBLDR.COM can be used only in a system with the same memory configuration as the memory configuration of that system for which they were created.

BOOTBLDR.COM requests that you enter information and gives you instructions about what to do next. As it executes, it displays messages that indicate what is taking place.

This section discusses those parts of the command procedure over which you have control. That is, it discusses how to respond to requests for information. If you want more information, you can read the command procedure itself by entering the following command at the console terminal:

```
$ TYPE SYS$UPDATE:BOOTBLDR.COM
```

Note: The console diskettes you are creating initialize the starting physical addresses of all memory on the system. If you enter incorrect memory amounts when executing BOOTBLDR.COM, these starting physical addresses will be incorrect. A machine check results if VAX/VMS references an incorrect physical address.

The following steps describe how to use BOOTBLDR.COM:

1. Enter the following command to invoke the procedure:

   ```
   $ @SYS$UPDATE:BOOTBLDR
   ```

   The procedure prompts as follows:

   ```
   Enter memory type (MA780, MS780C, MS780E or <RETURN> to end):
   ```

2. Enter the name of the first memory controller configured on the primary processor. For example, enter MS780E if you have an MS780E memory controller configured on the primary processor. Do not abbreviate or add suffixes to your response; for example, do not abbreviate MS780E as MS.

   There are no defaults; do not press RETURN until you have entered all the necessary information (the procedure continues to prompt for memory type until you press RETURN).

   The procedure then prompts as follows:

   ```
   Enter TR level (1 through 6):
   ```

3. Enter the number of the TR level at which the memory controller (MS780 or MA780) you entered in response to the previous prompt is configured. Enter only a number. Note that, if you simply pressed the RETURN key in response to the previous question, this prompt does not appear.

   The procedure then prompts as follows:

   ```
   Enter amount of memory for this controller in .25 megabyte
   increments (for example, for 512 kilobytes, enter .5):
   ```

4 Enter the amount of memory configured at this TR level. Enter only a number; that is, do not enter a suffix such as Mbytes or megabytes. Note that memory must be present in increments of 0.25 megabytes.

The procedure repeats this sequence of requests until you press RETURN (and nothing else) in response to the first request. You should press RETURN after you have named all memory controllers connected to the primary processor.

After you respond to a prompt by pressing RETURN, the procedure displays the following message:

```
Would you like the bootstrap command files to boot the system using
local (MS780A, MS780C, or MS780E) memory as well as shared (MA780)
memory <YES or [NO]>:
```

5 Enter YES and press RETURN. The procedure responds with the following information and prompt:

```
The UNIBUS Adapter (UBA) is assumed to be at TR level x.
Enter the TR level of the UBA (Enter <RETURN> to default):
```

The letter *x* represents a number from 1 to 15. BOOTBLDR.COM derives the number represented by the letter *x* by adding 1 to the number of the highest TR level at which an MA780 memory controller is configured. BOOTBLDR.COM uses the TR level of the UBA in the creation of boot command procedures (such as DM0BOO.CMD) for UNIBUS devices (such as RK06 and RK07 disk drives).

If a UBA is configured at the TR level displayed in the preceding message, simply press RETURN; do not enter a number. On the other hand, if a UBA is not configured at the TR level displayed in the preceding message, enter the TR level at which the UBA is configured; enter only a number and then press RETURN.

Note that a system can have more than one UBA and that BOOTBLDR.COM can create boot command procedures for use on only one UBA. In a system with more than one UBA, you must select the UBA for which you want boot command procedures created. In this way, boot command procedures for devices on that UBA (but not for devices on the other UBA) are created.

6 The procedure continues with a similar prompt for the MASSBUS adapter:

```
The Massbus Adapter (MBA) is assumed to be at TR level x.
Enter the TR level of the MBA (Enter <RETURN> to default):
```

If an MBA is configured at the TR level displayed in the preceding message, press RETURN and do not enter a number. If an MBA is not configured at the TR level displayed in the preceding message, enter the TR level at which the MBA is configured; enter only a number and then press RETURN.

The procedure continues by prompting as follows:

```
Enter the name of the default boot command procedure (DEFBOO.CMD)
to be used when booting the system.  (Default is xxnBOO.CMD):
```

VAX/VMS supplies a number of default boot command procedures to enable you to boot the system from various devices. In general, the file name of the default boot command procedure you should choose has as its first three characters the device name of the device on which you expect the system disk to reside; the remaining characters in the file name are BOO.CMD. Respond to the request by entering the file name (for example, DB0BOO.CMD).

7 Next, the procedure asks for information about the memory configuration for the attached processor. Before prompting you for the information, BOOTBLDR.COM reminds you that MA780 memory must be identical on both processors. For this reason, the procedure does not prompt for the TR levels at which MA780 memory is configured on the attached processor. You have already provided the necessary information about MA780 memory.

The procedure then mentions that local (MS780A, MS780C, or MS780E) memory on the attached processor may be different from local memory on the primary processor. That is, MS780 memory on the attached processor may be configured at different TR levels; further, there may be more or less MS780 memory on the attached processor.

The procedure then prompts as follows:

```
Enter memory type (MA780, MS780C, MS780E or <RETURN> to end):
```

8 Enter the name of the first memory controller configured on the attached processor.

The procedure then prompts as follows:

```
Enter TR level (1 through 6):
```

9 Enter the number of the TR level at which the memory controller (MS780 or MA780) you entered in response to the previous prompt is configured.

The procedure then prompts as follows:

```
Enter amount of memory for this controller in .25 megabyte
increments (for example, for 512 kilobytes, enter .5):
```

10 Enter the amount of memory configured at this TR level.

The procedure repeats this sequence of requests until you press RETURN (and nothing else) in response to the first request. You should press RETURN only after you have named all memory controllers connected to the attached processor.

11 After you respond to a prompt by pressing RETURN, the procedure prompts you for the name of the diskette drive you want to use. Enter the drive name as in the following example. BOOTBLDR.COM then instructs you to insert the original diskette in the drive and asks whether you are ready to continue.

```
Enter the name of the floppy disk drive you want to use: CSA1
Insert original 11/780 console floppy in CSA1:.
Ready to continue? (YES or NO): YES
Copying console floppy to temporary directory.
Copying VMB.EXE from SYS$SYSTEM.
11/782 requires a V3 or later VMB in order to use MA780 memory.
```

12 Next, BOOTBLDR.COM instructs you to remove the original diskette and to place a scratch volume in the drive.

```
Please remove original floppy from CSA1:
Creating floppy for primary processor.
Place a scratch floppy in CSA1:.
WARNING -- CSA1: will be initialized.
```

After warning you that the volume will be initialized, the procedure asks whether you are ready to continue. Enter YES (assuming you have placed a scratch diskette in the drive as instructed). BOOTBLDR.COM proceeds as follows:

```
Ready to continue? (YES or NO): YES
Note: Console media must not contain any bad blocks.
Analyzing CSA1: for defective blocks, please stand by...
%MOUNT-I-MOUNTED ...
Copying unmodified files to CSA1:.
Creating multiprocessor bootstrap command procedures.
Primary processor console floppy completed.
Creating floppy for attached processor.
Place a scratch floppy in CSA1:.
WARNING -- CSA1: will be initialized.
Ready to continue? (YES or NO):
```

If BOOTBLDR.COM finds any bad blocks, it will not mount the volume. Instead, it asks you to place another scratch volume in the drive.

13 Remove the first scratch volume and place another scratch volume in the drive. Enter YES to continue; BOOTBLDR.COM proceeds as follows and completes the build.

```
Ready to continue? (YES or NO): YES
Note: Console media must not contain any bad blocks.
Analyzing CSA1: for defective blocks, please stand by...
%MOUNT-I-MOUNTED ...
Copying unmodified files to CSA1:.
Creating multiprocessor bootstrap command procedures.
Attached processor console floppy completed.
```

If BOOTBLDR.COM finds any bad blocks, it does not mount the volume. Instead, it asks you to place another scratch volume in the drive.

You now have multiprocessing console diskettes for the primary and attached processors. Be sure to label them correctly—ATTACHED for the attached processor's console diskette, PRIMARY for the primary processor's console diskette. They are not interchangeable.

You should also indicate on the label the machine for which the multiprocessing console diskettes are intended. These diskettes can be used only in a VAX–11/782 system whose memory configuration is identical to the memory configuration you described when you created the diskettes using BOOTBLDR.COM. These diskettes cannot be used in a single-processor VAX–11/780 system or in a VAX–11/782 system with a different memory configuration.

2.15.2 Shutting Down the System

After you have created console diskettes for the primary and attached processors, shut down the system by entering the following command:

```
$ @SYS$MANAGER:SHUTDOWN
```

This command invokes the system shutdown command procedure, which shuts down the system in an orderly fashion.

Ensure that both processors are halted and that the console-mode prompt (> > >) appears on both console terminals.

2.15.3 Booting the VAX–11/782 System

With the system shut down and both processors halted, boot the system in the following manner:

1. Insert the primary processor's console diskette in the primary processor's console diskette drive.
2. Insert the attached processor's console diskette in the attached processor's console diskette drive.
3. Enter the BOOT command on the primary processor's console terminal.
4. Log in using the SYSTEM account.
5. Enter the DCL command START/CPU on the primary processor's console terminal.
6. Enter the BOOT command on the attached processor's console terminal.

The VAX/VMS operating system is now running on the VAX–11/782 system. You should now follow the procedure described in Section 2.15.4 for editing SYSTARTUP.COM.

2.15.4 Editing SYSTARTUP.COM

When the VAX/VMS operating system is running on the VAX–11/782 system, you should edit the site-specific startup command procedure SYS$MANAGER:SYSTARTUP.COM to allow automatic restart of the attached processor following a system shutdown.

To accomplish this, edit the command procedure SYS$MANAGER:SYSTARTUP.COM to include the following commands:

```
$ START/CPU
$ WRITE SYS$OUTPUT "You can boot the attached processor now."
```

On a cold start, you can boot the attached processor when the message "You can boot the attached processor now" appears on the primary processor's console terminal. To boot the attached processor, you can enter the BOOT command at the attached processor's console terminal, or you can press the BOOT button on the attached processor's console panel.

2.16 *VAX/VMS Verify Utility Reference Manual* — Correction

On page VER-7, the example should read /READ_CHECK, not /[NO]READ_CHECK. This correction will be incorporated in the next revision of the manual.

2.17 *VAX/VMS Developer's Guide to VMSINSTAL* — Correction

The VMSINSTAL CHECK_NET_UTILIZATION callback documented in Section 5.2 of the *VAX/VMS Developer's Guide to VMSINSTAL* (a new optional manual) is described as follows:

> This callback determines whether the net number of free blocks on the VMI$ROOT device is sufficient to successfully complete the installation.

The description should state "peak number" rather than "net number" of free blocks. This correction will be incorporated into the manual in a future revision.

2.18 *VAX/VMS Install Utility Reference Manual* — Additions and Corrections

This section describes information not included in the Install Utility documentation.

2.18.1 New Method of Invoking INSTALL

On page INS–1, the format for invoking INSTALL is given as:

```
RUN SYS$SYSTEM:INSTALL
```

This command line format became obsolete with Version 4.0, when the foreign command format was implemented. To establish the INSTALL command as the default for your site, you must define the global symbol INSTALL in your SYLOGIN.COM file as follows:

```
$ INSTALL == "$INSTALL/COMMAND_MODE"
```

Once this symbol is defined, you can invoke the Install Utility by entering INSTALL as a DCL command.

In a future release, this format will become the default.

2.18.2 Enhanced LIST/GLOBAL/FULL Command

The LIST/GLOBAL/FULL command of the Install Utility now displays the following additional information on global sections:

- Owner and protection
- Access control entries (ACEs) if an access control list (ACL) exists

2.18.3 /SUMMARY Qualifier

Used with the INSTALL/GLOBAL command, the /SUMMARY qualifier displays a summary of global section and global page usage on the system for both local and shared memory global sections.

2.18.4 Corrections to Text

Make the following corrections to the *VAX/VMS Install Utility Reference Manual*. These corrections will be incorporated into the next revision of the manual.

- Page INS–2 — Footnote 2 under Example INS-2 should read "with the /OPEN qualifier", not "with the /SHARED qualifier".
- Page INS–6 — The privilege listed as SYSLCKL should read SYSLCK.
- Page INS–7 — The file name GRPCOMMEXE should read GRPCOMM.EXE.
- Page INS–15 — In the third paragraph, 00038E should read 0003E8.

2.19 *VAX/VMS Accounting Utility Reference Manual* — Corrections

Make the following corrections to the *VAX/VMS Accounting Utility Reference Manual*. These corrections will be incorporated in the next revision of the manual.

- Page ACC–4 — In the example, the keyword ELAPSES should read ELAPSED.
- Page ACC–49 — Figure ACC–7 is incorrect. There should be an empty, unused byte at offset 25. ACR$W_USERNAME should be at offset 26. Each item in the figure should be moved forward by 1 byte, starting with the USERNAME field.

2.20 *VAX/VMS Mount Utility Reference Manual* — Addition

The documentation for the jobwide MOUNT support was omitted from the documentation. It should read as follows:

> Any subprocess in the process tree can mount or dismount a volume for the job. When a subprocess mounts a volume (for the job) as a private volume, the master process of the job becomes the owner of this device. This provision is necessary because the subprocess may be deleted and the volume should remain privately mounted for this job.

2.21 Image Activation, Search Lists, and Known Images

One of the steps involved in image activation uses VAX Record Management Service (RMS) to open the specified image file. When the image to be activated is specified as a logical name, the file specification that is the translation of that logical name is accessed. RMS then opens the image by first attempting to locate the image on one of the known file lists. If the image is not known (that is, the lookup operation fails) then RMS has no other choice but to incur the overhead of locating and opening the image file on disk.

If the image specification includes a semicolon or a period to delimit the version number (whether or not an explicit version number is actually specified), the known file lookup by RMS is skipped. In that case, RMS always incurs the overhead of opening the image file on disk.

The precedence of the known file lookup over the normal file system access during image activation is extended when an image is being activated by way of a search list. For each element on the search list that does not include a file version delimiter, RMS executes a known file lookup. This continues until a lookup is successful or until the search list is exhausted. If the search list is exhausted, RMS then evaluates the entire search list from its beginning a second time in an effort to locate and open the image file on disk. Further information about locating files using search lists can be found in the *Guide to VAX/VMS File Applications.*

Because of this behavior, it is suggested that care be taken when defining a search list that contains specifications for images that are installed. Regardless of the order of the elements of the search list, the first image in that search list that is found to be installed is the image selected for activation. That occurs even if there are preceding images in the search list that are not installed.

2.22 *VAX/VMS System Generation Utility Reference Manual* — Corrections

The following notes document errors and omissions in the Version 4.2 manual:

- The SHARE command is incorrectly documented as SHARE/CONNECT.
- On pages SGN–19 and SGN–20, the examples shown for the CONNECT command are incorrect and should be as follows:

```
SYSGEN> CONNECT LPA0 /ADAPTER=3/CSR=%0777514 -
_ SYSGEN> /DRIVERNAME=LP2DRIVER/VECTOR=%0200
          .
          .
          .
SYSGEN> CONNECT NET /NOADAPTER/DRIVER=NETDRIVER
```

- On page SGN–58, the final sentence in the description of the ACP_SHARE parameter should be as follows: "This parameter should be set on when ACP_MULTIPLE is on."
- On page SGN–62, the parameters FREEGOAL and FREELIM are listed as dynamic. These parameters are not dynamic.
- On page SGN–66, the description of the LNMHASHTBL parameter should indicate that the values specified for this parameter are always rounded up to the nearest power of 2. The same is true for the LNMPHASHTBL parameter.
- On page SGN–73, the parameters listed as PQL_DJJQUOTA and PQL_MJJQUOTA are misspelled and should be PQL_DJTQUOTA and PQL_MJTQUOTA respectively.
- On pages SGN–77 and SGN–78, the descriptions of the RMS_DFMBC and RMS_DFNBC parameters should be as follows:

RMS_DFMBC (D)
RMS_DFMBC specifies the default disk block size used by RMS in accessing sequential files.

Normally the default value is adequate.

RMS_DFNBC (D)
RMS_DFNBC specifies a default block count for network access to remote, sequential, indexed sequential, and relative files.

The network block count value represents the number of blocks that RMS is prepared to allocate for the I/O buffers used to transmit and receive data. The buffer size used for remote file access, however, is the result of a negotiation between VAX RMS and the remote File Access Listener (FAL). The buffer size chosen is the smaller of the two sizes presented.

Thus, RMS_DFNBC places an upper limit on the network buffer size that can be used. It also places an upper limit on the largest record that can be transferred to or from a remote file. In other words, the largest record that can be transferred must be less than or equal to RMS_DFNBC multiplied by 512 bytes.

Normally the default value is adequate.

- On page SGN–79, the following information should be included in the description of the SCSNODE parameter:

  Specify the parameter value as an ASCII string enclosed in quotation marks ("). Note that the string may not include dollar sign ($) or underscore (_) characters.

- On page SGN–84, the description of the TTY_DIALTYPE parameter should be as follows:

  TTY_DIALTYPE
  TTY_DIALTYPE provides flag bits for dial-ups. Bit 0 is 1 for United Kingdom dial-ups and 0 for all others. Bit 1 controls the modem protocol used. Bit 2 controls whether modem lines hang up 30 seconds after seeing CARRIER if a channel is not assigned to the device. The remaining bits are reserved for future use. See the *VAX/VMS I/O User's Reference Manual: Part I* for more information on flag bits.

2.23 *Guide to VAX/VMS System Security* — Corrections

Make the following corrections to the *Guide to VAX/VMS System Security*. These corrections will be included in the next revision of the manual.

2.23.1 Defining Ownership Privileges

Section 4.4.2 defines the conditions needed to convey ownership privileges to a user. The numbered list should be replaced with the following:

1. Hold the resource attribute to the identifier that owns the file
2. Running with BYPASS or SYSPRV
3. Running with GRPPRV and in the same group as the file owner

2.23.2 Establishing and Changing File Ownership

Section 4.4.5 describes the steps VAX/VMS uses to determine the default owner of a file. These steps should be replaced with the following list:

1 An attempt is made to propagate the ownership from a previous version of the file. This succeeds only if the user is privileged (holds BYPASS, SYSPRV, or GRPPRV privilege) or has ownership rights to the owner of the previous version.

2 If the attempt to propagate from the previous version fails (either because there is no previous version, the creator lacks ownership rights to the previous version, or the creator is not privileged), then an attempt is made to propagate ownership from the parent directory. This succeeds only if the user is privileged or has ownership rights to the owner of the parent of the directory.

3 If the attempt to propagate from the parent directory fails, then the owner of the created file is the same as the creator of the file.

2.23.3 Default ACL Protection

The second sentence in Section 4.5.2.2 states the following:

> In addition, when you create a file whose owner identifier is not your UIC, an ACE is added to your ACL for the file that grants full access to your UIC.

Replace this sentence with the following corrected version:

> In addition, when you create a file whose owner identifier is not your UIC, an ACE is added to your ACL for the file that grants CONTROL access plus the access available to the owner of the file (the **Owner** field of the SOGW protection mask).

A similar change will also be made to Section 5.2.6.2 and the flowcharts in Figures 4–4 and 5–5. These changes will be incorporated in the next revision of the manual.

2.23.4 Example Change

In Figure 5–10, the line

```
$ READ /END...
```

should be placed following the line

```
$ DELETE /SYMBOL /LOCAL /ALL
```

This correction will be included in the next revision of the manual.

2.23.5 System Passwords Incompatible with LAT Terminal Servers

The following text, found on page 5-26 of the *Guide to VAX/VMS System Security,* is incorrect:

> Note that the use of a system password is incompatible with the use of the DIGITAL-supplied terminal concentrator known as the LAT–11. For more information about LAT–11, see the documentation provided with the LAT–11 software.

Replace this incorrect text with the following:

> Note that the use of a system password is incompatible with the use of the DIGITAL-supplied LAT terminal servers and LAT/VMS. For more information about terminal servers, see the documentation provided with the server. For more information about LAT/VMS, see the *LAT/VMS Management Guide.*

2.24 TMPJNL and PRMJNL Privileges Removed

The TMPJNL and PRMJNL privileges, which were never used by VAX/VMS, have been removed from VAX/VMS Version 4.4 and subsequent versions.

Any documentation that mentions these privileges will be updated to reflect this change in the next release.

3 Supplemental Information for Application Programmers

This chapter contains supplemental information for application programmers.

3.1 *VAX/VMS Linker Reference Manual* — Correction

The following corrections apply to the *VAX/VMS Linker Reference Manual.*

- On page LINK-1, the default for the command qualifier /[NO]USERLIBRARY should read /USERLIBRARY=ALL.
- The reference to Section 6.3.6.2 on page LINK-31 (third list item at the top of the page) is incorrect. The correct reference is to Section 5.3.6.2.
- The reference to Appendix A on page LINK–61 (fourth line, second paragraph from the bottom) is incorrect. The correct reference is to Chapter 6, which contains information on the VAX Object Language.
- On page LINK–113, replace the last paragraph of Section 6.7 with the following sentence:

  The linker produces a DST only if the /DEBUG or /TRACEBACK qualifier was specified at link time.
- On page LINK–128, replace the first sentence of the fourth paragraph of the Description section with the following sentence:

  If you specify /SHAREABLE, you cannot also specify /SYSTEM.
- Example 3 on page LINK–129 is incorrect. The example should read as follows:

  ```
  $ LINK LAMAR,SYS$INPUT/OPTION GRABLE/SHAREABLE
  ```

  Note, GRABLE is the name of a shareable image file and not an options file as previously documented. The above correction also applies to Example 2 on page LINK–142.

3.2 Debugger

The behavior of the LINK/SHARE command changed in Version 4.4. For Version 4.4 and subsequent versions, if you link your shareable images using the LINK/SHARE command, traceback information is passed to the shareable image.

When you debug your program and execution is suspended within that shareable image, the debugger sets the image automatically. This is called dynamic image setting.

This results in different symbolic information being made available. For example, the display for SHOW CALLS will look different. In contrast to module setting, the symbol information for only the currently set image is available at any one time. (See the description of the SET IMAGE, SET MODULE, and SET MODE [NO]DYNAMIC commands in the *VAX/VMS Debugger Reference Manual* for more information.)

If you prefer the old behavior, you can link your shareable image with the command LINK/SHARE/NOTRACE. Traceback information will not be present in the image and DEBUG will not set the image.

To take full advantage of the new shareable image support, you should link your shareable image with the command LINK/SHARE/DEBUG. Then full symbol table information will be available, the debugger will set the image, and you can perform symbolic debugging of that shareable image.

3.2.1 Predefined Breakpoints

If any portion of your program is written in VAX Ada, then the following two breakpoints are automatically established when you invoke the debugger (the output of a SHOW BREAK command is shown):

```
Breakpoint on ADA event "DEPENDENTS_EXCEPTION" for any value
Breakpoint on ADA event "EXCEPTION_TERMINATED" for any value
```

These breakpoints are equivalent to entering the following commands:

```
DBG> SET BREAK/EVENT=DEPENDENT_EXCEPTION
DBG> SET BREAK/EVENT=EXCEPTION_TERMINATED
```

Ada programmers find these breakpoints convenient for debugging tasking programs.

3.2.2 CALL from an Exception Breakpoint

Prior to Version 4.6, you could not enter the CALL command directly after an exception breakpoint was triggered. This restriction has been removed.

Some related restrictions still apply. If a routine is called with the CALL command just after an exception breakpoint was triggered, no breakpoints, tracepoints, or watchpoints set within that routine are triggered. However, they are triggered if the CALL command is given at another time.

3.2.3 STEP from an Exception Breakpoint

Prior to Version 4.6 you could not enter the STEP command directly after an exception breakpoint was triggered. This restriction has been removed.

Entering a STEP command at an exception breakpoint causes you to step to the start of whatever exception handler gets control. If you have not declared any exception handlers, the exception is resignalled and the debugger prompt is displayed — that is, the STEP command has no effect.

3.2.4 Nonstatic Watchpoints

You can set watchpoints on dynamically allocated variables, such as those on the stack or in registers. These are called nonstatic watchpoints.

You can set a watchpoint on a nonstatic variable only when its defining routine is active. If you try to set a watchpoint on a nonstatic variable when its defining routine is not active, the debugger issues a warning, as in the following example:

```
DBG> SET WATCH Y
%DEBUG-W-SYMNOTACT, nonstatic variable 'Y' is not active
```

To implement nonstatic watchpoints the debugger must trace every instruction, slowing down the execution of the program being debugged. When you set a nonstatic watchpoint, the debugger determines whether the watched location is statically or nonstatically allocated. If the location is nonstatically allocated, the debugger issues an informational message that you are setting a nonstatic watchpoint, so that you will be aware of the slower performance.

A nonstatic watchpoint is automatically canceled when execution returns from its defining routine, and an informational message is issued to that effect.

3.3 VAX PASCAL Run-Time Library — Changes

The sections that follow describe changes to the PASCAL Run-Time Library.

3.3.1 DEC and UDEC Built-in Routine

- The default number of significant digits for the DEC and UDEC built-in routines has been changed from eight to ten.

  For example, consider the following code segment:

  ```
  WRITELN('<',DEC(12345),'>');
  ```

  Prior to Version 4.6, this code generated the following output:

  ```
  <   00012345>
  ```

  This code now generates:

  ```
  < 0000012345>
  ```

  To duplicate previous behavior, specify the number of significant digits in calls to the DEC or UDEC built-in routines.

- The default length parameter has been changed from 12 to 11 characters.

For more information about changes to DEC and UDEC, see the release notes for VAX PASCAL Version 3.5.

3.3.2 KEY Attribute Enhanced

The VAX PASCAL Run-Time Library includes support for the enhanced VAX PASCAL KEY attribute. The KEY attribute accepts the following additional keywords:

[NO]CHANGES
[NO]DUPLICATES
ASCENDING
DESCENDING

3.3.3 New Default RECORD_LENGTH for TEXT Files

The default record length for TEXT files has increased from 133 to 255 characters. To duplicate previous behavior, you must specify the following in your OPEN statement:

```
RECORD_LENGTH := 133
```

3.3.4 Use of EXTEND, REWRITE, and TRUNCATE

Prior to Version 4.6, the VAX PASCAL Run-Time Library enforced solitary access to sequential files for the EXTEND, REWRITE, or TRUNCATE built-in routines. The PASCAL Run-Time Library now supports shared access to sequential files for these built-ins.

3.4 PL/I Run-Time Library Supports VAX PL/I Version 3.0

The PL/I Run-Time Library supports VAX PL/I Version 3.0. The Run-Time Library contains several minor changes that support the PL/I SUBSCRIPTRANGE, STRINGRANGE, and STORAGE conditions; these changes are incompatible with previous versions of VAX PL/I. Specifically, the PL/I Run-Time Library has the following new primary condition values:

- PLI$_SUBRG (replaces PLI$_SUBRANGEn)
- PLI$_STRRANGE (replaces PLI$_SUBSTRn)
- PLI$_STORAGE (replaces LIB$GET_VM)

For each new primary condition value, the built-in ONCODE function returns the old primary status value.

DIGITAL recommends that customers upgrade to VAX PL/I Version 3.0. If you choose to run a previous version VAX PL/I, you should recode where appropriate. For example, you might make the following code change:

```
ON VAXCONDITION(PLI$_SUBSTR2) BEGIN;
      .
      .
      .
     END;
```

Change to:

```
DCL PLI$_STRRANGE GLOBALREF FIXED BIN(31) VALUE;
ON VAXCONDITION(PLI$_STRRANGE) BEGIN;
    IF ONCODE() ^= PLI$_SUBSTR2
    THEN
        CALL RESIGNAL();
    ELSE
    .
    .
    .
    END;
```

3.5 VAX Ada Run-Time Library — Unhandled Exceptions

VAX/VMS has improved the way the VAX Ada Run-Time Library deals with unhandled VAX Ada exceptions and VAX conditions. Exceptions and conditions are considered to be unhandled if they propagate as far as they can go—to the level of a task or a main program—and a VAX/VMS or VAX Ada Run-Time Library catch-all handler gains control. Catch-all handlers are located in frames enclosing the main program and library packages, each task body, and each accept body.

Beginning with Version 4.6, new catch-all handler messages are produced, and changes to program execution behavior have been made, as follows:

- If an unhandled VAX condition with a severity of success, information, warning, or error (any severity except severe) reaches a VAX Ada Run-Time Library catch-all handler, the handler displays the message associated with the condition and continues program execution. This behavior is consistent with the behavior of VAX/VMS catch-all handlers.
- If a VAX Ada exception or a VAX condition with a severity of severe reaches a VAX Ada Run-Time Library catch-all handler, the handler displays the exception or condition message, and then the task, main program, or rendezvous exits. (Note, however, that when an exception or severe condition leaves an accept body, the message is not displayed because the exception or condition propagates to both of the tasks involved in the rendezvous.)
- The VAX Ada Run-Time Library catch-all handlers display a warning when an unhandled exception may have to wait for dependent tasks to terminate.

The new catch-all handler messages are directed to both SYS$OUTPUT and SYS$ERROR.

Also, since Version 4.6, the point in the VAX Ada exception-handling sequence at which waiting for dependent tasks takes place has changed. Prior to Version 4.6, waiting for dependent tasks took place during the search for an applicable exception handler; with Version 4.6, waiting has been deferred until an applicable handler has been found. A more detailed explanation of this change follows.

In VAX Ada, a general condition handler is automatically established for all stack frames that have exception handlers, and a run-time table of active exception parts is maintained for each frame. The general condition handler determines which VAX Ada exception handler in the frame eventually gains control (if any). Any subsequent VAX Ada exception propagation takes place in two phases. During the first phase, the general condition handler determines which VAX Ada exception handler should gain control; each frame on the stack is searched for this handler. When the applicable handler is found, the general condition handler requests a stack unwind, and the second phase begins. During the second phase, each frame is removed from the stack. Prior to VAX/VMS Version 4.6, waiting for the termination of tasks dependent on some VAX Ada frame took place during the first phase (search for a handler). Now, waiting for dependent tasks takes place during the second phase (unwind). After the unwind, the handler for the exception executes.

The exception-handling improvements have the following effects:

- Programs written entirely in VAX Ada are not visibly affected by the change in the point at which waiting for dependent tasks takes place. Such programs are affected only by the new catch-all handler error messages and by continuation of the main program in cases of nonsevere unhandled exceptions.
- Software (such as the CLI) that signals a condition in order to print a message, expecting continuation at the point of the signal, is now supported—provided that the program does not handle the condition (exception) before the condition gets to the VAX Ada Run-Time Library catch-all handlers.
- Software that signals a nonsevere condition value with a call to the VAX/VMS Run-Time Library routine LIB$SIGNAL, but does not want the continuation that LIB$SIGNAL usually leads to, must call the Run-Time Library routine LIB$STOP instead, or use a VAX Ada raise statement (if the signaling software is written in VAX Ada).
- A task no longer terminates silently because of an unhandled exception—the exception message is now displayed. In addition, the exception message appears before waiting begins for dependent tasks (because such waiting may cause a deadlock). This makes VAX Ada programs more robust because an unexpected exception in a production program now generates a message.

  If you do not want your software to produce task termination messages, you may want to have exception handlers in those task bodies to which you expect unhandled exceptions to propagate. For example, if you expect that the predefined exception END_ERROR will cause task termination messages in one of your tasks, you could have the following code, or its equivalent (the action need not be a null statement), in the exception part of the affected task body:

  ```
  when END_ERROR => null;
  ```

  The handler absorbs the unhandled exception and prevents it from propagating further. The use of a handler in this situation also allows you to see that the termination resulting from this exception is to be expected.
- The change in the point at which waiting for dependent tasks takes place might affect mixed-language programs. Prior to Version 4.6, a VAX Ada exception that propagated to non-VAX Ada code would cause execution to wait until all dependent VAX Ada tasks terminated; a handler in the non-VAX Ada code could not execute until the tasks terminated. With Version 4.6 and subsequent versions, the exception is propagated, and dependent tasks continue to execute; a handler in the non-VAX Ada code can execute concurrently with the dependent tasks.

If some software beyond your control is adversely affected by the messages resulting from unhandled exceptions, you can hide the messages by defining the logical names SYS$OUTPUT, SYS$ERROR, and ADA$OUTPUT. Define SYS$OUTPUT and SYS$ERROR to be where you want the messages to go, and define ADA$OUTPUT to be where you want VAX Ada output (from package TEXT_IO) to go.

Note: **You should redirect error-message output only as a temporary measure until you have modified your program as previously described. If you redirect SYS$OUTPUT, be careful to ensure that you do not miss other error messages that might occur; DIGITAL advises that you capture the output directed to SYS$OUTPUT and compare it with output containing the messages you would otherwise expect.**

For information about new features of the debugger that affect VAX Ada programs, see Section 3.2.1.

3.6 VAX C Run-Time Library — Changes

The sections that follow describe changes to the VAX C Run-Time Library.

3.6.1 Printf Function Restrictions Removed

Prior to Version 4.6, **printf** functions could not format more that 512 characters in a single call. Now, **printf** functions accept formatted output of unlimited length. However, an individual field in the resulting string cannot be longer than 512 characters.

3.6.2 File Sharing Now Supported

Prior to Version 4.6, the VAX C Run-Time Library did not support file sharing. The VAX C Run-Time Library now supports file sharing when you use record mode to access files; you must use the **ctx=rec** file attribute with all file open functions. Specify the **shr=xxx** file attributes as appropriate.

3.6.3 Stream I/O Facilities

Version 4.6 and subsequent versions improve stream I/O facilities in the VAX C Run-Time Library. You can now specify the **mbc=nnn** file attribute when opening stream files. The value for this attribute specifies the number of blocks to allocate for I/O buffer. Reads and writes are performed using this block size.

For more information about changes to the VAX C Run-Time Library, see documentation for VAX C Version 2.3.

3.7 *VAX/VMS Run-Time Library Routines Reference Manual* — Correction

The descriptions of the screen management routines SMG$ENABLE_UNSOLICITED_INPUT, SMG$SET_BROADCAST_TRAPPING, and SMG$SET_OUT_OF_BAND_ASTS should describe the parameter passing mechanism for *AST-routine* as "address of entry mask by value" and not as "entry mask by reference."

3.8 *VAX/VMS Command Definition Utility Reference Manual* — Example Correction

The following example is an excerpt from Example CDU–2.

To make this BASIC program execute as described in the documentation, change the following lines (comments describe the changes):

```
200 SUB EXIT_COMMAND !Same as documented.
    !exclude EXTERNAL INTEGER FUNCTION SYS$EXIT
    CALL SYS$EXIT(1% BY VALUE)     !Note addition.

290 SUBEND  !Same as documented.

1   EXTERNAL INTEGER FUNCTION CLI$DCL_PARSE,CLI$DISPATCH !Exclude LIB$GET_INPUT
    EXTERNAL INTEGER FUNCTION SEND_COMMAND,SEARCH_COMMAND,EXIT_COMMAND !Same
    EXTERNAL INTEGER TEST_TABLE,LIB$GET_INPUT      !Note addition.

2   IF NOT CLI$DCL_PARSE(,TEST_TABLE,LIB$GET_INPUT,LIB$GET_INPUT,'TEST> ')
    AND 1%
    THEN GOTO 2  !Note elimination of 0 above.
```

3.9 *VAX Text Processing Utility Reference Manual* — Additions

The sections that follow describe changes to the *VAX Text Processing Utility Reference Manual.*

3.9.1 GET_INFO — Restriction

The material in the *VAX Text Processing Utility Reference Manual* does not include a restriction on using the built-in procedure GET_INFO. The following material should be added to the manual's description of the built-in procedure GET_INFO.

Be careful when you write programs that attempt to search one of the lists maintained by VAXTPU. VAXTPU provides only one context for traversing each list. VAXTPU maintains lists of buffers, defined keys, key maps, key-map lists, processes, and windows. You can search a list by using "first," "next," "previous," "current," or "last" as the second parameter to the built-in procedure GET_INFO.

If you create nested loops that attempt to search the same list, the results are unpredictable. For example, a program attempting to search two key-map lists for common key maps may contain the built-in procedure GET_INFO (KEY_MAP, "next",...) in a loop within a loop containing GET_INFO (KEY_MAP, "previous",...). This creates an infinite loop.

3.9.2 VAX BLISS — VAXTPU Example

The VAX BLISS example TPU–1 in Section 12 of the *VAX/VMS Utility Routines Reference Manual* contains errors. Example 3–1 contains corrections to Example TPU–3.

Example 3–1 Sample VAX BLISS Template for Callable VAXTPU

```
! How to declare the VAXTPU routines

external routine
        tpu$FILEIO,
        tpu$HANDLER,
        tpu$INITIALIZE,
        tpu$EXECUTE_INIFILE,
        tpu$EXECUTE_COMMAND,
        tpu$CONTROL,
        tpu$CLEANUP;

! How to declare the VAXTPU literals

external literal
!
! File I/O operation codes
        tpu$k_close,
        tpu$k_close_delete,
        tpu$k_open,
        tpu$k_get,
        tpu$k_put,
!
! File access codes
        tpu$k_access,
        tpu$k_io,
        tpu$k_input,
        tpu$k_output,
!
! Item codes
        tpu$k_calluser,
        tpu$k_fileio,
        tpu$k_outputfile,
        tpu$k_sectionfile,
        tpu$k_commandfile,
        tpu$k_filename,
        tpu$k_journalfile,
        tpu$k_options,
!
! Mask for values in options
        tpu$m_recover,
        tpu$m_journal,
        tpu$m_read,
        tpu$m_command,
        tpu$m_create,
        tpu$m_section,
        tpu$m_display,
        tpu$m_output,
!
! Bit positions for values in options
        tpu$v_display,
        tpu$v_recover,
        tpu$v_journal,
```

Example 3–1 Cont'd. on next page

Example 3–1 (Cont.) Sample VAX BLISS Template for Callable VAXTPU

```
        tpu$v_read,
        tpu$v_create,
        tpu$v_command,
        tpu$v_section,
        tpu$v_output,
!
! VAXTPU status codes
        tpu$_nofileaccess,
        tpu$_openin,
        tpu$_inviocode,
        tpu$_failure,
        tpu$_closein,
        tpu$_closeout,
        tpu$_readerr,
        tpu$_writeerr,
        tpu$_success;

own
        OPTIONS:        bitvector [32];
! OPTIONS will be passed to VAXTPU

GLOBAL ROUTINE top_level =
BEGIN
!++
! Main entry point of your program
!--
! Your_initialization_routine must be declared as a BPV

local   BPV:    vector[2,long] initial (TPU_INIT,0);! Procedure block

! First establish the condition handler

LIB$ESTABLISH (tpu$handler);


! Call the intialization routine and pass it the address of the BPV
! which has the address of your initialization routine (VAXTPU
! calls this)

tpu$initialize (BPV);

! Use the following call if the options word passed to VAXTPU indicated that
! an initialization file needs to be executed and/or the TPU$INIT_PROCEDURE
! in the section file needs to be executed.

tpu$execute_inifile();


! Let VAXTPU take over.
tpu$control();

! To break out of VAXTPU, use call_user from within a VAXTPU program
! Upon return from tpu$control, the editing session is done

tpu$cleanup();

! Loop and start the sequence over or exit
return tpu$_success;

END;
```

Example 3–1 Cont'd. on next page

Example 3–1 (Cont.) Sample VAX BLISS Template for Callable VAXTPU

```
ROUTINE TPU_INIT =
BEGIN
!
!--
own     BPV:    vector[2,long] initial (TPU_IO,0);! Procedure block

Macro
        OUTFILE_NAME= 'OUTPUT.TPU'%,
        COMFILE_NAME= 'TPUINI.TPU'%,
        SECFILE_NAME= 'SYS$LIBRARY:EVESECINI.TPU$SECTION'%,
           FILE_NAME= 'FILE.TPU'%;

! Set VAXTPU options I want to enable

OPTIONS[tpu$v_display] = 1;
OPTIONS[tpu$v_section] = 1;
OPTIONS[tpu$v_create]  = 1;
OPTIONS[tpu$v_command] = 1;
OPTIONS[tpu$v_recover] = 0;
OPTIONS[tpu$v_journal] = 0;
OPTIONS[tpu$v_read] = 0;
OPTIONS[tpu$v_output] = 1;

begin           !Just for BIND
bind
! Set up item list to pass back to VAXTPU to tell it what to do
! VAXTPU calls me back later
ITEMLIST = uplit byte (

!buffer length,      item code,         buffer address,  return address

 word (4),   word (tpu$k_options),     long (OPTIONS),  long (0),
 word (4),   word (tpu$k_fileio),      long (BPV),      long (0),
 word (%charcount(outfile_name)),
             word (tpu$k_outputfile),  long (uplit(%ascii(outfile_name))),
                                                        long (0),
 word (%charcount(comfile_name)),
             word (tpu$k_commandfile), long (uplit(%ascii(commandfile_name))),
                                                        long (0),
 word (%charcount(file_name)),
             word (tpu$k_filename),    long (uplit(%ascii(file_name))),
                                                        long (0),
 word (%charcount(secfile_name)),
             word (tpu$k_sectionfile), long (uplit(%ascii(secfile_name))),
                                                        long (0),
 long (0) );

return ITEMLIST;
end;
END;                    ! End of routine TPU_INIT

GLOBAL ROUTINE TPU_IO (P_OPCODE, FILE_BLOCK, DATA: ref block [,byte]) =
BEGIN
!
!--
local
        item: ref block [3,long],       ! Item list entry
        status;
```

Example 3–1 Cont'd. on next page

Example 3–1 (Cont.) Sample VAX BLISS Template for Callable VAXTPU

```
! Look at the opcode (operation) that VAXTPU wants me to perform
! and if I don't want to do it, just call it back
! if (..P_OPCODE NEQ tpu$k_open)
!       then
!       return (tpu$fileio (.p_opcode, .file_block, .data));
!
! Else set what operation to do

selectone ..P_OPCODE of
set
[tpu$k_open]:
! Time to open a file
!
    begin
        item = .data;           ! Point to the FILENAME item list entry

        end;
        return tpu$_success;    ! End of tpu$k_open
    end;
[tpu$k_get]:                    ! If none exists, then no data
! Time to read a record
    begin
    end;

[tpu$k_put]:                    ! Time to write a record
    begin
        return tpu$_success;
    end;
[tpu$k_close]:                  ! Time to close a file
    begin

        return tpu$_success;
    end;

[tpu$k_close_delete]: lib$stop (..p_opcode);
[otherwise]: lib$stop (..p_opcode);
tes;

return tpu$_success;

END;                    ! End of routine TPU_IO
```

3.10 Error Log Utility — New Features and Changes

The new features and changes that have been added to the Error Log Utility are outlined in the following two sections.

3.10.1 Enhancements to the User Interface

The ANALYZE/ERROR_LOG command has been enhanced to support the following:

1 New device class keywords for /EXCLUDE and /INCLUDE:

WORKSTATION	Include or exclude workstation error log entries.
LINE_PRINTER	Include or exclude line printer error log entries.

2 The BUSES keyword that is supported by /INCLUDE and /EXCLUDE has been enhanced to include BI bus error log entries.

3 The DEVICE_ERRORS keyword that is supported by /INCLUDE and /EXCLUDE has been enhanced to include BI adapter error log entries.

The error log entries for workstations and line printers can also be specified by indicating the device name or the type of entry that is logged for the new hardware to /INCLUDE and /EXCLUDE.

3.10.2 /EXCLUDE Qualifier Added

By default, whenever an "unknown" device, CPU, or error log entry is encountered by ANALYZE/ERROR_LOG, it outputs the entry in a hexadecimal longword format. The /EXCLUDE=UNKNOWN qualifier excludes these entries from the report.

4 Supplemental Information for System Programmers

This chapter contains supplemental information of use to system programmers.

4.1 *VAX/VMS System Services Reference Manual* — Correction

The sections that follow describe changes to the *VAX/VMS System Services Reference Manual.*

4.1.1 New Nullarg Argument

The $CHANGE_ACL, $FORMAT_ACL, and $PARSE_ACL system services have a place-holding **nullarg** argument that appears as the last argument in the argument list. Following is the correct format of each of these system services:

FORMAT **SYS$CHANGE_ACL** *[chan] ,objtyp ,[objnam] ,itmlst ,[acmode] ,[nullarg] ,[contxt] ,[nullarg]*

FORMAT **SYS$FORMAT_ACL** *aclent ,[acllen] ,aclstr ,[width] ,[trmdsc] ,[indent] ,[accnam] ,[nullarg]*

FORMAT **SYS$PARSE_ACL** *aclstr ,aclent ,[errpos] ,[accnam] ,[nullarg]*

Add the following definition to the end of the argument definition list for the $CHANGE_ACL, $FORMAT_ACL, and $PARSE_ACL system services:

nullarg

VMS usage:	**null_arg**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

Place-holding argument. This argument is reserved to DIGITAL.

4.1.2 $GETSYI Service — New Item Codes

The following item codes have been added to $GETSYI:

SYI$_XCPU

When SYI$_XCPU is specified, $GETSYI returns the extended CPU processor type of the node. $GETSYI returns this information only for the local node.

The general processor type value should be obtained first by using the SYI$_CPU item code. For some of the general processor types, there is extended processor-type information provided by the item code, SYI$_XCPU. For other general processor types, the value returned by the SYI$_XCPU item code is currently undefined.

Since the processor type is a longword decimal number, the **buffer length** field in the item descriptor should specify 4 (bytes).

The $PRDEF macro defines the symbols for the extended processor types. The current extended processor types available and their symbols are as follows:

VAX Processor Type Symbol	Extended Processor Type	Extended Processor Symbol
PR$_SID_TYPUV	MicroVAX II VAXstation II	PR$_XSID_UV_UV2
	MicroVAX 2000 VAXstation 2000	PR$_XSID_UV_410
PR$_SID_TYP8NN	VAX 8530	PRS$_XSID_N8530
	VAX 8550	PRS$_XSID_N8550
	VAX 8700	PRS$_XSID_N8700
	VAX 8800	PRS$_XSID_N8800

SYI$_XSID

When SYI$_XSID is specified, $GETSYI returns processor-specific information. For the MicroVAX II, this information is the contents of the system-type register of the VAX node. The system-type register contains the full extended information used in determining the extended system type codes. For other processors, the data returned by SYI$_XSID is currently undefined.

Since the value of this register is a longword hexadecimal number, the **buffer length** field in the item descriptor should specify 4 (bytes).

4.1.3 $GETSYI Service — Changed Item Code

The SYI$_CPU item code description for $GETSYI has been revised as follows:

SYI$_CPU

When SYI$_CPU is specified, $GETSYI returns the general CPU processor type of the node. $GETSYI returns this information only for the local VAX node.

Since the processor type is a longword decimal number, the **buffer length** field in the item descriptor should specify 4 (bytes).

Symbols for the processor types are defined by the $PRDEF macro. The following chart gives the current processors and their symbols. For information about extended processor-type codes, see the description of the SYI$_XCPU item code in this section.

Processor	Symbol
VAX-11 780, 782, 785	PR$_SID_TYP780
VAX-11 750	PR$_SID_TYP750
VAX-11 730	PR$_SID_TYP730
MicroVAX I	PR$_SID_TYPUV1
MicroVAX II series	PR$_SID_TYPUV2
VAXstation 2000	PR$_SID_TYP410
VAX 8600, 8650	PR$_SID_TYP790
VAX 8200, 8300	PR$_SID_TYP8SS
VAX 8530, 8550, 8700, 8800	PR$_SID_TYP8NN

4.1.4 $QIO and $QIOW System Services

The documentation for the P1 through P6 arguments to the $QIO and $QIOW services is incorrect. These parameters can be passed either by reference or by value, depending on the I/O function they perform.

4.2 System Dump Analyzer — New Command

The System Dump Analyzer has a new command, SHOW CALL_FRAME. This section describes the new command in detail. The information in this section updates the *VAX/VMS System Dump Analyzer Reference Manual.*

SHOW CALL_FRAME

Displays the locations and contents of the longwords representing a procedure call frame.

FORMAT

SHOW CALL_FRAME *address*

COMMAND PARAMETER

address

An expression representing the starting address of the procedure call frame you want to display.

QUALIFIERS

/NEXT_FP

Displays the procedure call frame starting at the address stored in the FP longword of the last call frame displayed by this command.

DESCRIPTION

Whenever a procedure is called using CALLG or CALLS instructions, information is stored on the stack of the calling routine in the form of a procedure call frame. This call frame contains the following longwords:

- A condition handler address
- A longword containing the stack pointer alignment bits, the register save mask for registers R0 through R11, and the saved PSW of the caller
- The saved AP value of the calling routine
- The saved FP value of the calling routine
- The saved PC value (return address) of the calling routine
- One longword for each saved register (R0 through R11) of the caller, specified by the register save mask

The SHOW CALL_FRAME command displays the call frame information by interpreting a specified address expression as the beginning address of the call frame. If no address expression or options are specified, the default address expression for SHOW CALL_FRAME is the longword contained in the current process FP register.

The following example shows the display produced by the SHOW CALL_FRAME command. The display consists of the following sections:

Instruction Type	The display indicates what type of instruction, either a CALLG or CALLS instruction, generated the procedure call frame.
Call Frame Address	SDA lists all the virtual addresses that are part of the call frame. The call frame addresses are listed in a column that increases in increments of 4 bytes (one longword).
Call Frame Contents	SDA lists the contents of the call frame longwords in a column next to the call frame addresses.
Symbols	SDA attempts to display the contents of the longwords in the call frame with the exception of the "Mask-PSW" longword, which is not symbolized.
Longword Description	SDA provides a meaningful description of the contents of each longword in the context of a procedure call frame.
Stack Alignment	SDA provides a message describing the number of bytes by which the stack pointer was adjusted prior to storing the call frame information.
Argument List	For CALLS cases, the argument list is displayed by virtual address and contents in two columns below the stack alignment field.

All valid procedure call frames have a 0 in bit 28 of the second longword of the call frame. If the call frame specified has a 1 in bit 28 of the second longword of the call frame, the call frame is invalid and the SDA display shows:

```
Invalid Call Frame:   Bit 28 is Set in "Mask-PSW" Longword
```

All valid procedure call frames begin on a longword boundary. If the specified address expression does not begin on a longword boundary, the call frame is invalid and the SDA display shows:

```
Invalid Call Frame:   Start Address Not On Longword Boundary
```

EXAMPLE

```
SDA> SHOW CALL_FRAME 7FFE7D94
```

```
                              Call Frame Information
                              ----------------------
                                      Call Frame Generated by CALLS Instruction
                              Condition Handler     7FFE7D94  00000000
                              SP Align Bits = 00    7FFE7D98  20FC0000
                                 Saved  AP          7FFE7D9C  7FFED024
                                 Saved  FP          7FFE7DA0  7FFE7DE4    CLT$GL_KSTKBAS+005E4
                                 Return PC          7FFE7DA4  801A0CEE    SYSTEM_PRIMITIVES+005E4
                                      R2            7FFE7DA8  7FFE7DD0    CTL$GL_KSTKBAS+005D0
                                      R3            7FFE7DAC  7FFCCFF8
                                      R4            7FFE7DB0  80443D90
                                      R5            7FFE7DB4  7FFCD000
                                      R6            7FFE7DB8  7FFE6400    MMG$IMGHDRBUF
                                      R7            7FFE7DBC  00000003
                              Align Stack by 0 Bytes =>
                              Argument List         7FFE7DC0  00000003
                                                    7FFE7DC4  7FFE7DD0    CTL$GL_KSTKBAS+005D0
                                                    7FFE7DC8  00000000
                                                    7FFE7DC8  00000000
```

4.3 Ethernet/802 Device Drivers

The sections that follow describe changes to the Ethernet/802 device drivers.

4.3.1 IEEE 802 Response Packets

The Ethernet/802 device drivers now allow a response packet to be transmitted on channels that have the IEEE 802 packet format enabled. This is accomplished using the WRITE function code and the IO$M_RESPONSE modifier. Use of this modifier is validated for those IEEE 802 channels that have Class I service enabled; the control field value for channels with Class I service enabled must be either XID or TEST in order to send a response packet.

4.3.2 802 User-Supplied Services

Prior to Version 4.6, the Ethernet/802 device drivers responded to XID and TEST command packets. For Version 4.6, all XID and TEST packets (command or response) for channels with User Supplied service are not responded to by the Ethernet/802 device drivers, but are instead passed to the application through READ requests.

Ethernet/802 device drivers still respond to XID and TEST command packets for channels with Class I service enabled.

4.3.3 Protocol Type Validation

The protocol type (NMA$C_PCLI_PTY) parameter is now validated on the SETMODE QIO. The validation is implemented as follows: the Ethernet/802 device driver takes the low order word of the longword parameter and swaps the two bytes; this new word value may not be less than 1501 (O5DD hexadecimal). If the value is less than 1501, SS$_BADPARAM status is returned in the IOSB.

4.4 *Writing a Device Driver for VAX/VMS* — Corrections

The sections that follow contain corrections to the *Writing a Device Driver for VAX/VMS* manual.

4.4.1 IFNORD, IFNOWRT, IFRD, and IFWRT Macros

The descriptions of the IFNORD, IFNOWRT, IFRD, and IFWRT macros on pages B-16 through B-19 erroneously define the **dest** argument.

The published information is incorrect in both the text and parameter definition list. The following table supplies the correct definitions:

Macro	Definition of "dest"
IFNORD	Address to which IFNORD passes control if either of the specified bytes cannot be read in the specified access mode
IFNOWRT	Address to which IFNOWRT passes control if either of the specified bytes cannot be written in the specified access mode
IFRD	Address to which IFRD passes control if both bytes can be read in the specified access mode
IFWRT	Address to which IFWRT passes control if both bytes can be written in the specified access mode

4.4.2 Bootstrapping with XDELTA

In Section 15.1 of the *Writing a Device Driver for VAX/VMS* manual, the instructions for bootstrapping with XDELTA on the VAX 8200 and VAX 8800 systems apply as well to the VAX 8300, VAX 8350, VAX 8530, VAX 8550, and VAX 8700 systems.

4.4.3 EXE$QIODRVPKT Executive Routine

The executive routine EXE$QIODRVPKT was inadvertently omitted from Appendix C of the Version 4.4 *Writing a Device Driver for VAX/VMS* manual. (It is, however, described in Section 8 of that volume.)

The following is the routine description:

EXE$QIODRVPKT

MODULE: SYSQIOREQ

Driver FDT routines call EXE$QIODRVPKT to send an IRP to a driver start-I/O routine. This routine calls EXE$INSIOQ and then transfers control to EXE$QIORETURN.

Registers	Contents
R3	Address of IRP for the current I/O request
R4	Address of current PCB
R5	Address of UCB

Fields	Contents
UCB$B_FIPL	Driver fork IPL
UCB$V_BSY (in UCB$L_STS)	Unit busy flag
UCB$L_IOQFL	Address of unit I/O queue listhead

4.4.4 $DEF Macro

Page B-5 incorrectly describes the method for defining a second symbolic name for a single field. The following text should appear as the second paragraph in the description of the **alloc** argument:

> You can define a second symbolic name for a single field, using the $DEF macro a second time immediately following the first definition, leaving the **alloc** argument blank in the first definition. The following example does this, equating SYNONYM2 with LABEL2:

```
$DEFINI JLB                     ; Start structure definition
$DEF    LABEL1  .BLKL  1        ; First JLB field
$DEF    SYNONYM2                ; Synonym for LABEL2 field
$DEF    LABEL2  .BLKL  1        ; Second JLB field
$DEF    LABEL3  .BLKL  1        ; Third JLB field
$DEFEND JLB                     ; End of JLB structure
```

4.5 *VAX/VMS I/O User's Reference Manual: Part II* — DR32 Microcode Loader

The following information should be added in Section 4.4.4 of the *VAX/VMS I/O User's Reference Manual: Part II* immediately after item 2 ("If the opening procedure..."):

> By default, XFLOADER attempts to load microcode into all DR32s on a system. To limit microcode loading to a subset of DR32s, define the logical name XF$DEVNAM using the device names of the DR32s as the equivalence names. XFLOADER will search for the translation using the LNM$FILE_DEV search list. For example:

```
$ DEFINE/SYSTEM XF$DEVNAM XFAO,XFCO
```

> This definition tells XFLOADER to load microcode only in the first and third DR32s on the system.

The sentence that will immediately follow the above addition ("After loading microcode into all available DR32s,...") should be changed to read: "After loading microcode into all specified DR32s,...".

4.6 *VAX MACRO and Instruction Set Reference Manual* — Cyclic Redundancy Check (CRC)

The following step should be included in the Cyclic Redundancy Check (CRC) instruction on page 9–138 of the *VAX MACRO and Instruction Set Reference Manual*:

```
Upon completion of the CRC instruction,
registers  RO, R1, R2  and R3 are left as follows:

                RO = result of CRC

                R1 = 0

                R2 = 0

                R3 = address one byte beyond end of source string
```

A Generic VAXBI Device Support in VAX/VMS

This appendix provides information needed to write and load a device driver for a non-DIGITAL-supplied device attached to the VAXBI bus. VAX/VMS Version 4.5 provides special support for such devices in the system initialization routines for the VAX 8200, VAX 8300, VAX 8350, VAX 8530, VAX 8550, VAX 8700, and VAX 8800 systems. Because of the many and varied implementations of VAXBI devices, however, VAX/VMS support must of necessity be very general. Some devices may more fully utilize the VAXBI interface than others; a device might incorporate its interface initialization logic in microcode, whereas another might defer initialization to code in its driver.

The *VAXBI Options Handbook* includes a description and guidelines for possible VAXBI device implementations. Please refer to that manual for further discussion of all VAXBI topics discussed in brief in Section A.2 and elsewhere in this document.

A.1 Overview

A VAXBI device driver refers to the same data structures and contains the same routines as a traditional VAX/VMS driver. Since this document presents only information specific to writing a driver for a non-DIGITAL-supplied device, it presumes an understanding of the material discussed in the *Writing a Device Driver for VAX/VMS* manual.

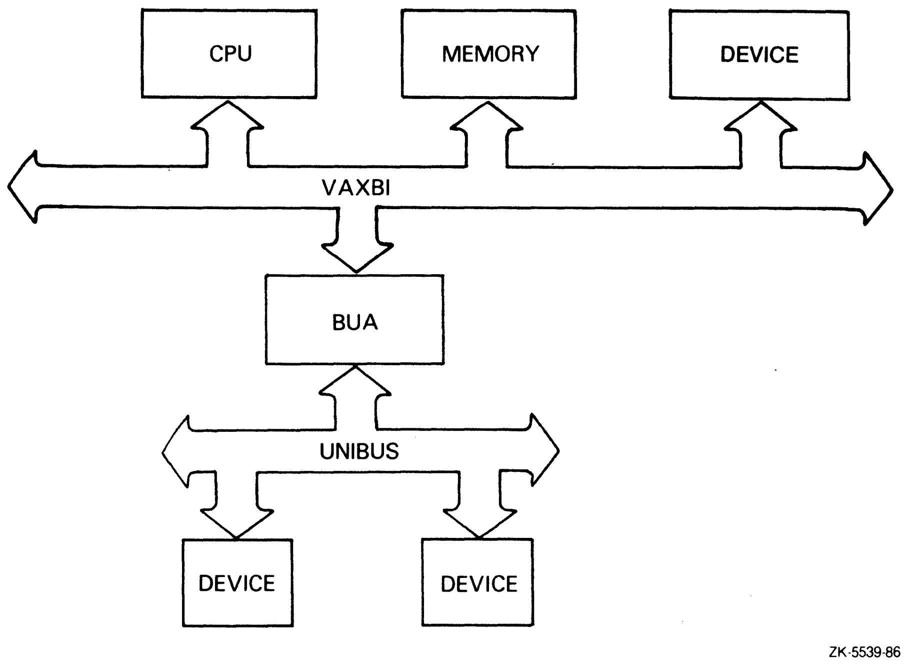
A VAXBI device driver deviates from the traditional VAX/VMS driver almost exclusively in the code that initializes the VAXBI interface or supports direct-memory-access (DMA) transfers for devices that address memory across the VAXBI bus. Section A.4 discusses tasks that drivers of various VAXBI devices can perform in their initialization routines to supplement both VAX/VMS initialization and that initialization performed by device microcode. Section A.5 contains a general discussion of how some VAXBI devices and their drivers manage DMA transactions.

Section A.3 describes those data structures the VAX/VMS adapter initialization routine creates and prepares for a generic VAXBI device, while Section A.7 discusses the method by which the driver for a generic VAXBI device can be loaded into the operating system. The final section of this document provides reference material and includes a description of the backplane interconnect interface chip (BIIC) registers and a summary of the IOC$ALLOSPT system routine.

A.2 VAXBI Concepts

The VAXBI serves as the I/O bus for the VAX 8200, VAX 8300, VAX 8530, VAX 8550, VAX 8700, and VAX 8800 systems (see Figure A–1).[1] Each of the VAX 8200 and VAX 8300 systems can have a single VAXBI; the VAX 8530, VAX 8550, VAX 8700, and VAX 8800 systems can have multiple VAXBI buses (see Figure A–2).

Figure A–1 VAX 8200, VAX 8300, and VAX 8350 Systems

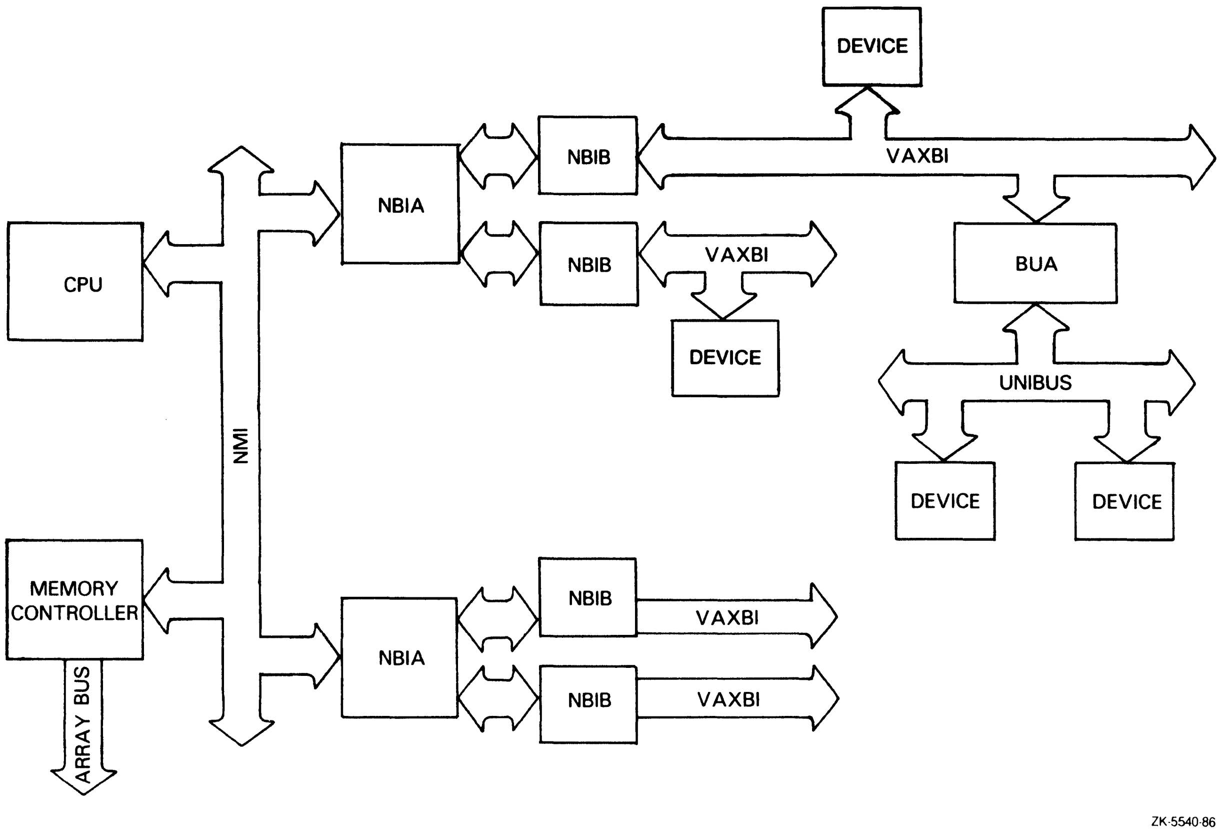


Each location on a VAXBI bus is called a *node*. A single VAXBI bus can service 16 nodes. In the case of the VAX 8200 and VAX 8300 systems, these nodes can be processors, memory, and adapters; the VAX 8530, VAX 8550, VAX 8700, and VAX 8800 systems permit only adapters to be attached to the VAXBI bus.[2] A node receives its *node ID*, a number from 0 to 15, from a plug on the VAXBI backplane slot into which the node module is inserted.

[1] The VAXBI is also the system bus for the VAX 8200 and VAX 8300 systems.

[2] For the VAX 8530, VAX 8550, VAX 8700, and VAX 8800, the NMI-to-BI adapter (NBIA/NBIB) (or, more specifically, the NBIB) resides at a node on a VAXBI bus, monitoring and controlling transactions to the memory interconnect (NMI) where the processors and memory reside.

Figure A–2 VAX 8530, VAX 8550, VAX 8700, and VAX 8800 Systems



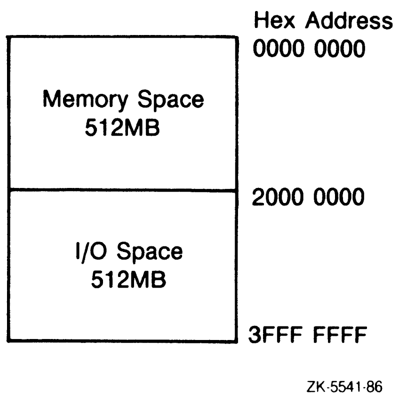
An *adapter* is a node that connects other buses, communication lines, and peripheral devices to the VAXBI bus. This document uses the term *device* to refer to a device or combination of devices serviced by a single adapter or controller.

A.2.1 VAXBI Address Space

Each VAXBI bus supports 30-bit addressing capability. This gigabyte of physical address space is split equally between memory and I/O space, as shown in Figure A–3.

All memory locations on a VAXBI bus are addressed using physical addresses in VAXBI memory space (from 00000000_{16} through $1FFFFFFF_{16}$). A VAXBI device, or its driver, that accesses memory directly, must perform virtual-to-physical translation before transmitting a memory address on the bus (see Section A.5).

Figure A–3 VAXBI Address Space



VAXBI I/O space (physical addresses 20000000_{16} through $3FFFFFFF_{16}$) is partitioned as illustrated in Figure A–4. Figure A–5 shows the structure of an I/O-space address.

As shown in Figures A–4 and A–5, VAXBI architecture grants each of the 16 nodes on a VAXBI bus two discrete sections in I/O space: node space and window space.

Node space	An 8KB block of addresses consisting of 256 bytes of *BIIC CSR space*, followed by *user interface CSR space*. A device can access the control and status registers (CSRs) of its backplane interconnect interface chip by using BIIC CSR space addresses. Device-specific registers reside in user interface CSR space. Because the VAX/VMS adapter initialization routine virtually maps node space for each VAXBI node on each VAXBI bus, a device driver can access both BIIC registers and device registers using virtual addresses. (See Sections A.4 and A.5 for a discussion of driver access to registers.)
Window space	A 256KB block used by a VAXBI adapter to map an I/O transfer to a target bus. Because VAX/VMS does not automatically map window space to virtual addresses, a driver that manipulates addresses in window space must itself allocate and fill sufficient system page-table entries for the range of its window space addresses. (See Section A.4.)

Figure A–4 Description of VAXBI I/O Space

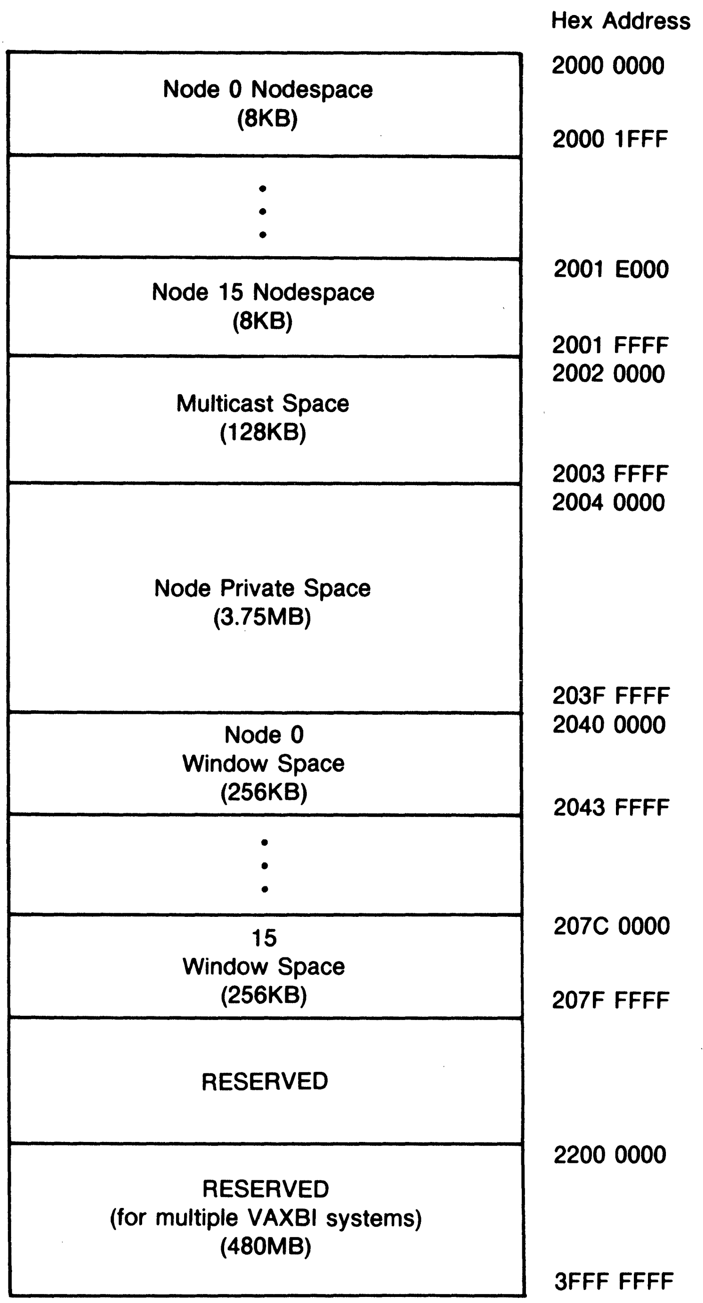
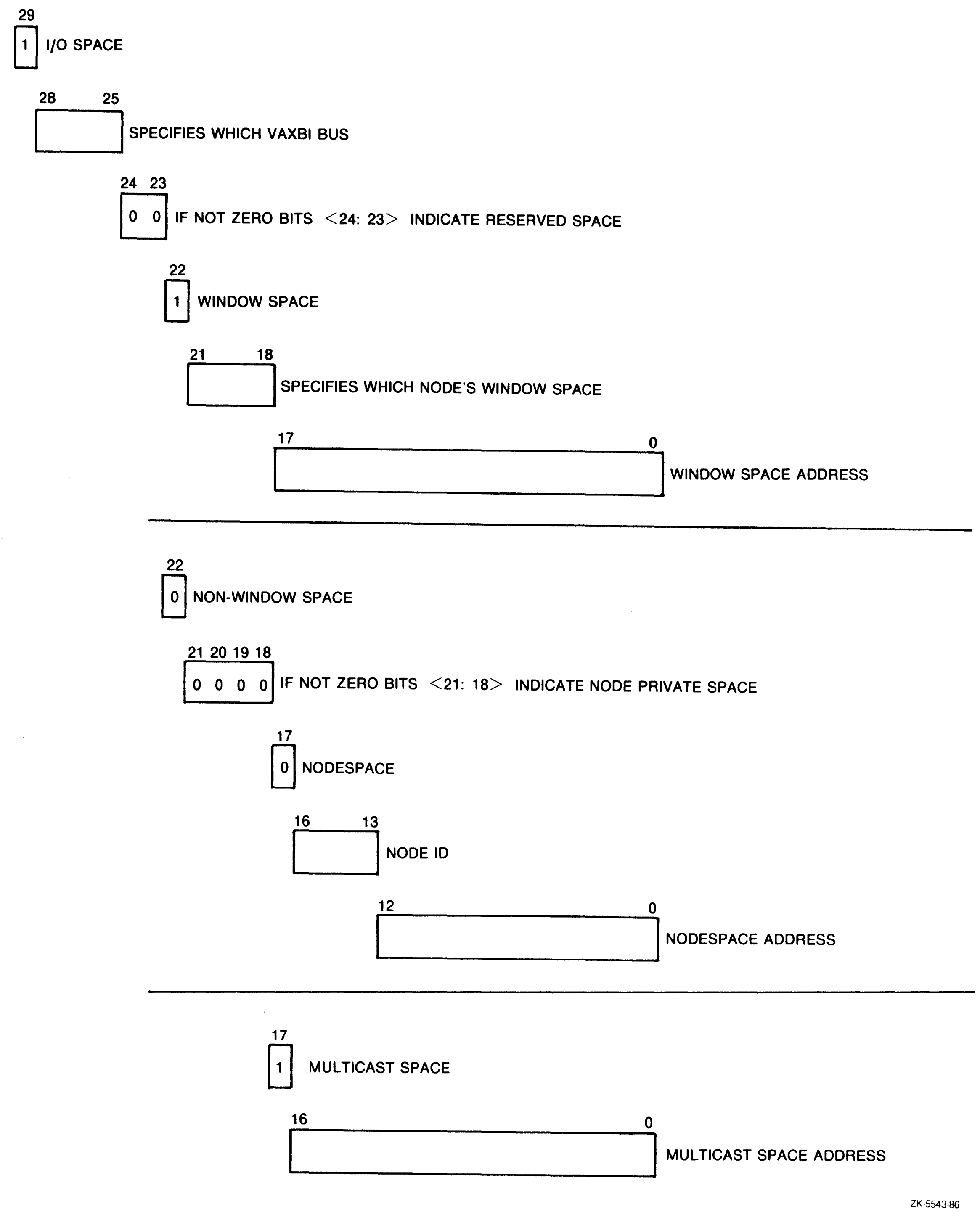


Figure A–5 Physical Addresses in VAXBI I/O Space



A.2.2 Backplane Interconnect Interface Chip (BIIC)

The *backplane interconnect interface chip* (BIIC) serves as the primary interface between the VAXBI bus and the user interface logic of a node. The BIIC supplies the logic necessary for a node to initiate and respond to transactions on the VAXBI bus, arbitrate bus ownership, send and receive interrupt requests, and monitor bus errors.

A node can enable, control, and monitor such activities by accessing the set of BIIC registers located in the first 256 bytes of its node space. Because the VAX/VMS adapter initialization routine virtually maps node space addresses, drivers for VAXBI devices can use virtual addresses to access BIIC registers. In addition, given the virtual address of the base of a device's node space, a driver can use the symbolic offsets, masks, and bit fields defined by the VAX/VMS macro $BIICDEF (in SYS$LIBRARY:LIB.MLB). Table A–1, in Section A.8.1, describes these symbols.

A.3 Initialization Performed by VAX/VMS

During the phase of system initialization known as adapter initialization (INIADP), VAX/VMS performs a set of processor-specific tasks to identify and configure each device it discovers at each of the 16 nodes on each VAXBI bus in the system configuration.

The INIADP module configures DIGITAL-supplied and non-DIGITAL-supplied devices alike, performing the following activities as part of its initialization cycle:

1. Tests for the presence of a device at the node by issuing a MOVL instruction, the target of which is a system virtual address temporarily mapped to the first longword of its node space. If this instruction is successful, it returns the contents of the BIIC Device-Type Register of the addressed node to the processor.[3]
2. Records the 32-bit contents of the BIIC Device-Type Register in the slot in the CONFREGL array that corresponds to the VAXBI bus and node at which it found the device[4] and compares this value against a table of recognized device types.
3. If it *recognizes* the device, maps the number of pages specified in the table for the device type and places the system virtual address of the base of the mapped node space in the slot in the SBICONF array that corresponds to the VAXBI bus and node at which it found the device.[5]

[3] If no device exists at a given VAXBI node address, the CPU becomes aware of this in a processor-dependent way. For example, the VAX 8200 and VAX 8300 processors experience a machine check, whereas the VAX 8530, VAX 8550, VAX 8700, and VAX 8800 processors determine that the node is vacant by reading an NXM (nonexistent memory) error from the BIIC Bus Error Register of the NBIB adapter on the VAXBI being examined.

[4] The CONFREGL array is a set of longwords in system pool pointed to by EXE$GL_CONFREGL. The CONFREGL array contains an entry for each possible VAXBI node. For VAX 8200 and VAX 8300 systems with one VAXBI, this array has 16 entries. For VAX 8530, VAX 8550, VAX 8700, and VAX 8800 systems, this array has 16 entries for each VAXBI bus on the system.

[5] The SBICONF array is a set of longwords, similar in structure to the CONFREGL array and pointed to by MMG$GL_SBICONF, that lists the system virtual addresses of the base of the node space for each node on a VAXBI bus.

If it does *not* recognize the device, maps the entire 8KB of the node's node space into VAX/VMS virtual address space by allocating 16 system page table entries (SPTEs) and associating them with the 16 page-frame numbers (PFNs) of the physical addresses assigned to this node's node space on this VAXBI bus. INIADP then saves the base system virtual address of the resulting 8KB range in the longword slot corresponding to this node in the SBICONF array.

4 Performs such additional tasks as allocating and filling in data structures in a device-specific manner. For a non-DIGITAL-supplied device attached to a VAXBI bus, INIADP creates generic versions of the channel request block, interrupt dispatch block, and adapter control block—as discussed in Section A.3.1—and fills in the appropriate vectors in the system control block.

For devices it *recognizes*, INIADP additionally calls a VAX/VMS-supplied subroutine, the address of which it obtains from the device-type table, that performs further device-specific initialization.

For devices it does *not* recognize, INIADP must defer device-specific initialization to the device driver's initialization routine.

A.3.1 Data Structures

The INIADP module creates and prepares a channel request block, interrupt dispatch block, and an adapter control block in the manner described below. For each data structure it creates, INIADP fills in the first three longwords with the standard VAX/VMS header information (that is, the structure type, size, and links).

Channel Request Block

For the newly created channel request block (CRB), INIADP performs the following tasks:

- Sets up the resource wait queue header (CRB$L_WQFL and CRB$L_WQBL).
- Initializes the single interrupt dispatcher (CRB$L_INTD) so that it has the effect of pushing general registers R0 through R5 onto the stack, and issuing a JSB instruction. The destination of the JSB instruction at initialization is a standard null interrupt handler that merely dismisses the interrupt. Later, when the specific device driver is loaded for the device (see Section A.7), the driver's interrupt-servicing routine address replaces this null interrupt handler in the dispatcher.

Interrupt Dispatch Block

INIADP initializes the interrupt dispatch block (IDB) in the following manner:

- Sets the number of device units controlled by this interrupt dispatch block (IDB$W_UNITS) to 1. As a result, the list of unit-control block (UCB) addresses in this IDB is one longword in size. The driver-loading procedure writes a UCB address into this longword whenever it creates a new UCB associated with the controller. Because there is only one slot in this array, drivers for non-DIGITAL-supplied multidevice controllers must use a different mechanism to locate the UCB of interest at the time of an interrupt.

- Copies the virtual address of the base of this device's node space to IDB$L_CSR from the corresponding slot in the SBICONF array.

Adapter-Control Block

INIADP creates a truncated adapter control block (ADP) for a non-DIGITAL-supplied VAXBI device (48 bytes as opposed to the traditional 600 bytes). The ADP it creates contains no fields reserved for the allocation and accounting of data paths or mapping registers. INIADP prepares this generic ADP in the following manner:

- Copies the virtual address of the base of this device's node space to ADP$L_CSR from IDB$L_CSR.
- Places the VAXBI node ID of this device in ADP$W_TR.
- Stores the value AT$_GENBI (signifying the *generic VAXBI* ADP type) in ADP$W_ADPTYPE. Symbol AT$_GENBI has the value 13_{10} in VAX/VMS Version 4.5.
- Inserts the address of the new channel request block in ADP$L_CRB.
- Calculates the address of the first of the four interrupt vectors for this node in the system control block (SCB) and places the address in ADP$L_AVECTOR. Each of these SCB vectors contains the address of CRB$L_INTD plus 1. A driver can determine the addresses of the other three SCB vectors by adding 64, 128, or 192, respectively, to the address of this first SCB vector.
- Saves the offset of this first SCB vector from the start of its SCB page in ADP$W_BI_VECTOR. (Refer to Section A.3.2 for a description of the SCB.)
- Places in ADP$L_BI_IDR a longword mask with a single bit set, as appropriate to the VAX processor, that specifies which VAXBI node should become the destination of interrupts from this node. On VAX 8200 and VAX 8300 systems, the VAXBI node of the primary processor becomes the destination for interrupts; on VAX 8530, VAX 8550, VAX 8700, and VAX 8800 systems, it is the VAXBI node at which the NBIB adapter for this VAXBI bus resides.
- Stores in ADP$L_MBASCB — and in each of the device's four SCB vectors — the address of the interrupt dispatcher. The actual stored value is CRB$L_INTD+1, the set low bit of the address indicating that the interrupt stack be used to service the interrupt. Certain powerfail recovery operations use the contents of ADP$L_MBASCB to refresh the SCB vectors.
- Saves in ADP$L_MBASPTE the contents of the first of the 16 SPTEs that map the device's node space. Certain recovery operations use the contents of ADP$L_MBASPTE to restore correct SPTE values and to remap node space following a power failure.

A.3.2 System Control Block

The system control block (SCB) consists of two or more pages of vectors. For all VAX processors, the first half page contains vectors used in exception dispatching. VAX/VMS uses the remainder of the first page, as well as subsequent pages, in a processor-dependent way.

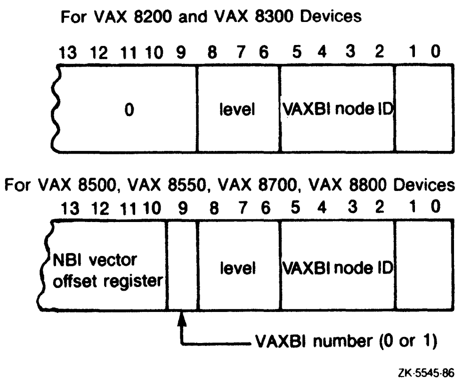
For VAX 8200 and VAX 8300 systems, VAX/VMS assigns the vectors from 100_{16} to $1FC_{16}$ to VAXBI devices in the order of their node IDs.

In contrast, VAX 8530/8550/8700/8800 system architecture relegates vectors 100_{16} to $1FC_{16}$ to NMI nexus vectors. Page 1 is reserved for the first "offsettable" device that exists in the system. (An "offsettable" device is an adapter, such as the BI-to-UNIBUS adapter [BUA], that passes interrupts from devices on another bus to the VAXBI and, from there, to the NMI and the processor.) If there is more than one "offsettable" device, an additional SCB page is needed for each.

Ultimately, the vectors for other devices attached to each of the four possible VAXBI buses of the system are contained in the four corresponding SCB pages from page 28 to page 31. Vectors for devices connected to VAXBI 0 and VAXBI 1 on NBI 0 are assigned to pages 28 and 29 of the SCB, respectively; vectors for devices connected to VAXBI 0 and VAXBI 1 on NBI 1 are likewise assigned to pages 30 and 31.

Generally, a VAX processor obtains a device vector from the BIIC registers of the node that has requested the interrupt (see Figure A–6). Information supplied in the device vector allows the processor to index to the corresponding interrupt-dispatching vector in the appropriate page of the SCB. For VAX 8200 and VAX 8300 systems, such information includes the interrupt level of the device and its VAXBI node ID. A similar vector for VAX 8530, VAX 8550, VAX 8700, and VAX 8800 devices further specifies the appropriate NBI vector offset and the number of the VAXBI bus.

Figure A–6 VAXBI Device Vectors



The specific SCB interrupt-dispatching vector, thus found, transfers control to the interrupt-dispatching code in the device's CRB. Upon an interrupt from this device, the SCB vector directs flow into the interrupt dispatcher in the CRB, which saves the register contents and dispatches to the interrupt handler established by the device driver.

A.4 Initialization Performed by the VAXBI Device Driver

All generic VAXBI device drivers must specify *GENBI* as the adapter type in the **adapter** argument to the DPTAB macro.

The device driver's initialization routines are expected to initialize the device-specific aspects of the VAXBI device. For non-DIGITAL-supplied devices, the initialization routines perform the sort of tasks that the INIADP module performs for the DIGITAL-supplied devices it discovers on a VAXBI bus. For single-unit devices, a separate unit-initialization routine may not be necessary.

The VAX/VMS System Generation Utility (SYSGEN) calls the controller-initialization routine at IPL 31, passing it the following values in the listed general registers:

- R4 pointing to the system virtual address of the device's node space
- R5 pointing to the IDB
- R6 pointing to the device data block
- R8 pointing to the CRB

After the controller-initialization routine has completed, SYSGEN calls the driver's unit-initialization routine at IPL 31 and passes it the following values in the listed general registers:

- R3 pointing to the system virtual address of the device's node space
- R5 pointing to the UCB

Hardware initialization might include such activities as writing values to BIIC and device-specific registers, examining the results of the BIIC self test, mapping a node's window space, building data structures to control the device, and linking these structures into chains of similar data structures.

This section provides some ideas and guidelines for code that may be necessary in an initialization routine. There is no requirement that driver code perform all the functions discussed here. The needs of various devices differ, and some devices make more demands on driver software than others.

Code examples in this section assume that R4 initially contains the virtual address of the base of the device's node space and R8 contains the virtual address of the device's CRB.

A.4.1 Examining BIIC Self-Test Status

According to the hardware specification for all devices attached to a VAXBI bus, a VAXBI node undergoes a self test on power failure recovery and at system boot time. The BIIC indicates the successful completion of the self test by setting BIIC$V_STS and by clearing BIIC$V_BROKE in BIIC$L_BICSR.

A driver unit initialization routine should test these bits before performing any transaction on the VAXBI bus. If BIIC$V_STS is clear, then self test is still under way. If BIIC$V_BROKE is set, then the driver action is implementation-specific. In any event, a driver should not set UCB$V_ONLINE in UCB$L_STS if the node is not usable.

The maximum duration of the BIIC self test is ten seconds. If a VAXBI node implements the maximum self-test time, then the driver unit initialization routine may have to spinwait for the setting of BIIC$V_STS (for instance, by embedding the testing instructions in an invocation of the TIMEDWAIT macro). Driver unit initialization routines should perform this spinwait only when UCB$W_POWER in UCB$L_STS is set. Otherwise, the driver is being loaded by SYSGEN, and a long spinwait at high IPL will have adverse effects on the rest of the VAX/VMS system.

Normally, only diagnostics initiate a self test by setting the SST bit in the BIIC. A VAXBI driver that sets this bit must take special precautions to avoid a machine-check and to avoid undetected corruption of VAXBI memory. These precautions include the following steps:

1. Begin a machine-check protection block (using the $PRTCTINI macro). Code within the block executes at IPL 31.
2. Disable arbitration on the VAXBI node being reset.
3. Set BIIC$V_SST and BIIC$V_STS simultaneously to initiate the self test.
4. Do not set SST in the same instruction that disables arbitration.
5. End the machine-check protection block (using the $PRTCTEND macro).
6. Do not access the BIIC registers for at least one microsecond. You cannot even check the state of the STS bit during this interval.
7. Do not access any other address on the VAXBI node until the self test has completed.

A.4.2 Clearing BIIC Errors, Setting Interrupts, and Enabling Interrupts

There is a set of tasks that a VAXBI driver should perform during initialization that ensures that interrupts are properly enabled and delivered to an appropriate VAXBI target node. These tasks include the following:

- Clearing any outstanding set bits in the Bus Error Register.
- Setting the target node for interrupts in the Interrupt Destination Register.
- Setting the device interrupt vector in the Error Interrupt Control Register.
- Setting the device interrupt vector in the User Interface Interrupt Control Register.
- Enabling hard and soft error interrupts as required by the device. Typically hard errors and enabled and soft errors are disabled.

- Enabling interrupts upon certain types of transactions to user interface CSR space.

It is important that the interrupt vectors and destination be set up *before* BIIC hard error and soft error interrupts are enabled. An error occurring while error interrupts are enabled but while the vector is uninitialized could lead to an invalid condition.

A.4.2.1 Clearing the Bus Error Register

The following example clears all set bits in the Bus Error Register (BIIC$L_BER) to prevent spurious or pending error interrupts at initialization:

```
MOVL   BIIC$L_BER(R4), BIIC$L_BER(R4)      ;Clear all set write-1-to-clear
                                           ; bits in BIIC$L_BER
```

A.4.2.2 Loading the Interrupt Destination Register

The Interrupt Destination Register (BIIC$L_IDR) specifies which VAXBI node should become the destination of interrupts from this node. On the VAX 8200 and VAX 8300 systems, the VAXBI node of the primary CPU becomes the destination for interrupts. On VAX 8530, VAX 8550, VAX 8700, and VAX 8800 systems, the VAXBI node of the NBIB on the particular VAXBI on which this device resides becomes the destination for such interrupts.

The VAX/VMS system initialization procedure described in Section A.3 creates a 32-bit mask with the appropriate bit set and stores it in ADP$L_BI_IDR. If a driver must set the Interrupt Destination Register, it can simply move this value to the BIIC register:

```
MOVL   CRB$L_INTD+VEC$L_ADP(R8),RO        ;Get ADP address
MOVL   ADP$L_BI_IDR(RO),BIIC$L_IDR(R4)    ;Write to IDR
```

A.4.2.3 Setting Interrupt Vectors

A VAXBI node uses the Error Interrupt Control Register (BIIC$L_EICR) to determine the SCB vector through which to interrupt when a BIIC at this node detects a bus error. The User Interface Interrupt Control Register (BIIC$L_UICR) similarly controls the operation of interrupts initiated by the device at this node.

Because the VAX/VMS system initialization procedure described in Section A.3 saves the offset of the node's first SCB vector from the start of its SCB page in ADP$W_BI_VECTOR, a driver can initialize both of these registers by using code similar to that in the following example:

```
MOVL   CRB$L_INTD+VEC$L_ADP(R8),RO        ;Get ADP address
MOVZWL ADP$W_BI_VECTOR(RO),R2             ;Get device vector
MOVL   BIIC$L_UICR(R4),BIIC$L_UICR(R4)    ;Clear user vector
MOVL   R2,BIIC$L_UICR(R4)                 ;Set user vector
BISL   #1@<BIIC$V_LEVEL+BIIC$S_LEVEL-1>,R2
                                          ;OR in interrupt level
                                          ;BR7 in this case
MOVL   BIIC$L_EICR(R4),BIIC$L_EICR(R4)    ;Clear error vector
MOVL   R2,BIIC$L_EICR(R4)                 ;Set error vector
```

Note that the driver clears both vectors before it actually sets them. Clearing BIIC$L_UICR and BIIC$L_EICR causes any pending interrupt to be cleared. Also note that the interrupt level must be set in BIIC$L_EICR, in this case BR7. If the level is not set, an error interrupt will never be generated.

A.4.2.4 Enabling Error Interrupts

Finally, to enable interrupts that report errors detected by the node's BIIC, the controller-initialization routine can set the soft error interrupt-enable or hard error interrupt-enable bits in the VAXBI Control and Status Register. The BIIC sets bits in the Bus Error Register (BIIC$L_BER) to reflect the type of bus error reported by the interrupt.

```
BISL    #<BIIC$M_SEIE!BIIC$M_HEIE>,-          ;Soft error interrupt enable
        BIIC$L_BICSR(R4)                      ;Hard error interrupt enable
```

A.4.2.5 Enabling BIIC Options

Device registers are in the area of node space called user interface CSR space and are located following the 256 bytes reserved for the BIIC-required registers. Use of user interface CSR space is implementation-dependent.

For the processor to be alerted to various transactions directed at user interface CSR space, the controller-initialization routine of devices that support such transactions should set appropriate bits in the BCI Control and Status Register (BIIC$L_BCICR). See Table A–1, in Section A.8.1, for definitions of these bits.

The following example enables a node to alert the node specified as the interrupt destination (in BIIC$L_IDR) when a retry timeout, STOP command, or read or write transaction is directed at its user interface CSR space:

```
BISL    #<BIIC$M_STOPEN!-                     ;Stop enable
          BIIC$M_RTOEVEN!-                    ;Retry timeout enable
          BIIC$M_UCSREN>,-                    ;User CSR enable
          BIIC$L_BCICR(R4)
```

A.4.3 Mapping Window Space

Each VAXBI, starting at address 20400000_{16}, provides 16 address blocks of 256K bytes apiece, called *window space*. VAXBI nodes can use window space if it is necessary to map VAXBI transactions to memory space on a target bus, although few nodes use this feature.

Whereas the VAX/VMS initialization routine maps each VAXBI node's node space to virtual addresses, it does not automatically map each node's window space. If a device needs to use its window space, it is up to the driver's unit initialization routine to map this space.

First of all, the driver must determine the starting physical address of the node's window space. Figure A–5 illustrates how VAXBI addresses are constructed. Drivers can use the following VAX/VMS-supplied macros (in SYS$LIBRARY:LIB.MLB) to access pertinent VAXBI addresses and values:

- $IO8SSDEF (for the VAX 8200 and VAX 8300 systems)
- $IO8NNDEF (for the VAX 8530, VAX 8550, VAX 8700, and VAX 8800 systems)

To determine the starting address of a node's window space, the driver should perform the following actions:

1 Extract the VAXBI node ID from BIIC$L_BICSR.

2 Multiply the node ID with the size of window space, as stored in IO8SS$AL_NDSPER for VAX 8200 and VAX 8300 systems and IO8NN$AL_NDSPER for VAX 8530, VAX 8550, VAX 8700, and VAX 8800 systems. VAXBI device drivers running on a VAX 8200 or VAX 8300 system can skip to step 4.

3 Perform steps necessary to account for the existence of multiple VAXBI buses on the system. These steps are only necessary for VAXBI device drivers running on a VAX 8530, VAX 8550, VAX 8700, or VAX 8800 system. They include the following:

   a. Determining which VAXBI bus the node is attached to by extracting the VAXBI bus number from bits <7:4> of ADP$W_TR.

   b. Multiplying the VAXBI bus number thus obtained by 2000000_{16}, the amount of physical address space allocated for each VAXBI bus.

   c. Adding the result to the product obtained in step 2.

4 Add to the accumulated calculations the base address of the start of window space for node 0 on a VAXBI bus. This address can be determined by adding the values contained in IO8SS$AL_NODESP and IO8SS$AL_IOBASE (for a VAX 8200 or VAX 8300 system) or IO8NN$AL_NODESP and IO8NN$AL_IOBASE (for a VAX 8530, VAX 8550, VAX 8700, or VAX 8800 system).

Secondly, each page of window space to be used must be associated with a system page-table entry (SPTE) that maps the page-frame number (PFN) of the physical page in window space to a system virtual address. VAX/VMS includes the routine IOC$ALLOSPT in module IOSUBNPAG that, given the number of SPTEs to be allocated in R1, returns in R2 the starting system virtual page number (SVPN) of the first allocated SPTE of the requested amount.

Because IOC$ALLOSPT expects to be called at IPL$_SYNCH, the unit-initialization routine must fork from IPL$_POWER to IPL$L_SYNCH before calling it. See Section A.4.4 for a discussion of forking in a driver initialization routine.

Finally, once the SPTEs have been allocated, the driver moves the physical page-frame numbers (PFNs) of the window space pages into the SPTEs.

A.4.4 Forking from a Driver Initialization Routine

If a driver initialization routine must fork to perform a thread of code that must synchronize with code or a structure synchronized at a lower IPL, it must take special care to avoid breaking that synchronization.

First of all, because the System Generation Utility under normal, circumstances, immediately calls a driver's unit-initialization routine at IPL$_POWER after its controller-initialization completes, the unit-initialization routine must be prepared for the instance of a controller-initialization routine that forks. Such a unit-initialization routine would complete before the fork thread of the controller-initialization routine resumed.

A fork thread in a unit-initialization routine (or a controller-initialization routine in a driver without a unit-initialization routine) must otherwise take the following precautions to avoid breaking synchronization:

- Allocate a separate fork block within the UCB. Do not attempt to allocate this block with EXE$ALONPAGEDYN. The separate fork block prevents a conflict with the use of the normal UCB fork block by the IOFORK routine.
- Use a semaphore bit to protect against multiple forking. Remember that the unit initialization routine can be called repeatedly in the case of power failures. If the semaphore shows that a fork is in progress, then exit without attempting to fork.
- Invoke EXE$FORK with R5 pointing to the new fork block. Restore the original value of R5 once the fork process is active.
- Remember to restore all registers on exit to the unit initialization routine. Since EXE$FORK removes the caller's address from the stack and returns to the caller's caller, the unit initialization routine must set up a dummy caller's caller routine to restore registers destroyed by EXE$FORK.

A.5 DMA Transfers

The method by which a device accomplishes direct-memory-access (DMA) transfers depends upon the characteristics of the device. As part of a VAXBI read or write transaction, such a device must place on the VAXBI bus a physical address, the target of which is a memory node or a node (such as an NBIB adapter) that transmits the request to memory across another bus.

For the DMA device to successfully access the memory pages of a buffer involved in an I/O transfer, it must be given sufficient information about the size and location of these buffer pages, the type of transaction that is requested, the offset into the first page of the buffer, and the length of the transaction. In addition, if the size of the transaction causes it to exceed the boundaries of a page, the device must have some means of accessing the remaining pages—even if they are, as is most likely, scattered throughout physical memory.

As a result, devices make use of several types of structures that map to the various pages of the buffer involved in the transfer to help generate a succession of contiguous physical addresses on the VAXBI bus. Some possible constructions of this kind include the following:

- A physically contiguous buffer in memory
- System page tables in system memory
- Process page tables locked in system memory
- Mapping registers in the device's VAXBI I/O address space

A separate but related issue results from the fact that the original buffer, as specified in the user Queue-I/O request, is in process space and is mapped by process page-table entries. Because the driver cannot rely on process context existing at the time the device is ready to service the I/O request, it must have some means of guaranteeing that it can access both the data involved in the transfer and the page-table entries that map the buffer.

VAX/VMS supplies two separate techniques applied by traditional VAX/VMS drivers and described in full in the *Writing a Device Driver for VAX/VMS* manual. These techniques are as follows:

- *Direct I/O,* the technique used most commonly by DMA drivers, locks the user buffer in memory as well as the page-table entries that map it. The function-decision table (FDT) of a direct I/0 DMA driver calls a VAX/VMS-supplied FDT routine that prepares the user buffer for direct I/O.
- *Buffered I/O* is the strategy whereby the driver FDT dispatches to an FDT routine in the driver that allocates a buffer from nonpaged pool. It is this intermediate buffer that is involved in the DMA transfer. Driver preprocessing routines copy the data from the user buffer to the system buffer for a write request; I/O postprocessing routines deliver data from the system buffer to the user buffer for a read request.

That DMA drivers may make use of either VAX/VMS direct I/O or buffered I/O is one way by which these drivers can supply specific information needed by the device to accomplish a DMA transfer. Those driver FDT routines that call a VAX/VMS direct-I/O FDT routine leave the following information in the device's unit-control block (UCB):

UCB$L_SVAPTE	Virtual address of the system page-table entry (PTE) for the first page used in the transfer
UCB$W_BOFF	Byte offset in the first page of the transfer buffer
UCB$W_BCNT	Size in bytes of the transfer

FDT routines that elect buffered-I/O call EXE$ALLOCBUF to obtain a nonpaged pool buffer and initialize the same UCB fields with the following information:

UCB$L_SVAPTE	Virtual address of system buffer used in the I/O transfer
UCB$W_BOFF	Number of bytes to be charged to the process for the transfer
UCB$W_BCNT	Size in bytes of the transfer

If a driver's fork process must manipulate the data in any way at fork level (that is, outside of the driver's FDT routines), then it needs a virtual address it can use to access the data. Typically this is done by using a nonpaged pool buffer. It can also be done by loading a system page-table entry with the correct PFN and computing the associated system virtual address. The drivers for the disks that have ECC correction do this when there is an ECC error detected. The controller can tell the driver that the error in the data in memory can be corrected by applying some pattern to a part of the data, but the fork process has to perform the correction, not the controller.

```
MOVL    IRP$L_SVAPTE(R3),R2               ;Get address of system buffer
SUBW3   #12,8(R2),UCB$W_BCNT(R5)          ;Calculate system buffer length
BICW3   #C^<VA$M_BYTE>,(R2),UCB$W_BOFF    ;Put offset in buffer
EXTZV   #VA$V_VPN,#VA$S_VPN(R2),R2        ;Get system virtual page number
MOVL    G^MMG$GL_SPTBASE,R1               ;Get address of system page table
MOVAL   (R1)[R2],UCB$L_SVAPTE(R5)         ;Get system virtual address of page
```

A.5.1 Example: DMB32 Asynchronous/Synchronous Multiplexer

The DMB32 asynchronous/synchronous multiplexer can use any of four different modes of address translation for DMA accesses. Under each of these modes, the DMB32 requires that its driver supply an address by which it can either directly or indirectly obtain the pages of the buffer that is involved in the transfer. The four different translation modes require such addresses in one of the following forms:

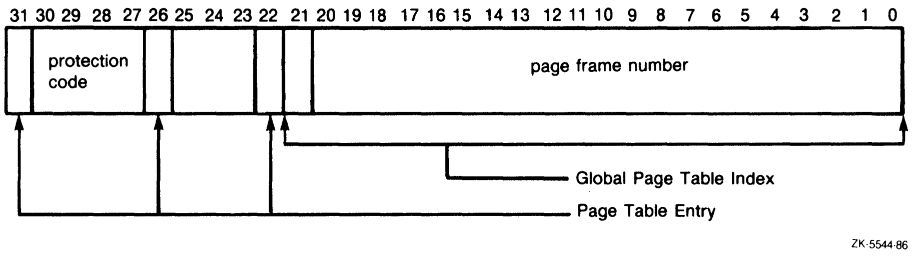
- System virtual address of a buffer
- System virtual address of a page-table entry
- Physical address of a page table
- Address of a physically-contiguous buffer

System Virtual Address of a Buffer and System Virtual Address of a Page-Table Entry

The DMB32 itself can perform the first two types of address translation because it can read entries in the VAX/VMS system page table (see Figure A–7, as well as the *VAX/VMS Internals and Data Structures* manual). The controller-initialization routine of a DMB32 device driver supplies the physical address and length of the VAX/VMS system page table, plus the virtual address and length of the VAX/VMS global page table. It also sets a page-table-valid bit in a device maintenance register.

As a result, a driver for a DMB32 device could use either direct I/O or buffered I/O and accordingly load a device register with the system virtual address of the page-table entry that maps the buffer or the system virtual address of the buffer itself. After the driver has loaded other device registers with a buffer offset value and a transfer size—and set the "start" bit in a DMB32 line-control register—the DMB32 performs the transfer without any additional mapping or other driver intervention.

Figure A–7 Page Table Entry



Physical Address of a Page Table

In this mode, the DMB32 can be given the physical address of a page table that maps the I/O transfer. The DMB32 architecture mandates that each page-table entry be four bytes long and that the page table be aligned on a longword boundary. Also, each page is 512 bytes long. However, the page table can be anywhere in memory, possibly at a range of VAXBI I/O-space addresses belonging to the node to which the DMB32 adapter is attached. To perform a DMA transfer under this addressing mode, the DMB32 adapter requires the offset of the first byte of the buffer, which is in the page described by the page-table entry. Each page-table entry contains bits <29:9> of the physical address of the page that is to be accessed.

In this case, the driver must extract the PFNs of the pages involved in the transfer and insert them into the page table of the device. The following is an example of a routine that translates a system virtual address to a physical address. It returns the physical address at the top of the stack.

```
VIRT_TO_PHYAD:
     PUSHL    (SP)                            ;Create slot at top of stack for return
                                              ; value
     PUSHR    #^M<R0,R1,R2,R3>                ;Save registers
     BICL3    #-512,R1,R0                     ;R0 = byte offset of address
     EXTZV    #VA$V_VPN,-                     ;Extract VPN
              #VA$S_VPN,R1,R2                 ; and put it in R2
     MOVL     G^MMG$GL_SPTBASE,R3             ;R3 => system page table
     MOVL     (R3)[R2],R3                     ;R3 => PTE
     EXTZV    #PTE$V_PFN,-                    ;Get page frame number of buffer
              #PTE$S_PFN,R3,R3                ; page into R3
     ASHL     #VA$V_VPN,R3,R3                 ;Shift into place for physical address
     BISL3    R0,R3,20(SP)                    ;Put result into stack slot
     POPR     #^M<R0,R1,R2,R3>                ;Restore registers
     RSB                                      ;Return to caller
```

Physical Address of a Buffer

If the device can neither read system page tables nor has its own scatter-gather map—and must perform a DMA transfer that spans physical pages—it must rely upon the actual contiguity of the physical pages involved in the transfer. Because there is no guarantee that this is the state of the user's buffer, the driver must allocate an intermediate buffer consisting of contiguous physical pages. The driver never deallocates this buffer unless the driver is being unloaded (by means of SYSGEN's RELOAD command). The best time to allocate such a buffer is during the device's initialization, when memory is most likely to be contiguous.

The VAX/VMS routine EXE$ALOPHYCNTG, described in the *Writing a Device Driver for VAX/VMS* manual, allocates such a buffer. The size of the buffer that should be allocated depends on the device's characteristics and the size of the transfers requested on the device. A buffer of four pages is likely to be large enough for most disk transfers, for example, but if you have enough memory on your system, you might want to make your buffer the size of a disk track in order to reduce disk latency. In any event, large transfers to the device can be segmented into transfers the size of your intermediate buffer.

The start-I/O routine of such a driver copies the data from the user's buffer into the intermediate, physically contiguous buffer by means of the routine IOC$MOVFRUSER.

The driver then sets up the device for the DMA transfer as follows:

1 Determines the physical address of the buffer from the system virtual address returned by EXE$ALOPHYCNTG
2 Moves the address to the device address register
3 Activates the device
4 If the transfer size exceeds the size of the buffer, returns to step 1

When a user requests a transfer from such a device, the driver moves the data from the device to the intermediate, physically contiguous buffer by means of a DMA transfer, then calls IOC$MOVTOUSER to copy the data into the user's buffer.

A.6 Register-Dumping Routine

In the event of a device error or a VAXBI bus error, a driver's register-dumping routine should contain code that makes certain interesting registers available for error logging. Apart from any device registers that should be saved, the following BIIC registers might contain information important in determining the cause of the error: the Device Register (BIIC$L_DTREG), the VAXBI Control and Status Register (BIIC$L_BICSR), the Bus Error Register (BIIC$L_BER), the Error Interrupt Control Register (BIIC$L_EICR), and the Interrupt Destination Register (BIIC$L_IDR).

The following is an example of part of a register-dumping routine that pushes the contents of these BIIC registers into an error buffer.

```
MOVL    BIIC$L_DTREG(R4),(RO)+          ;Device Type Register
MOVL    BIIC$L_BICSR(R4),(RO)+          ;BIIC CSR Register
MOVL    BIIC$L_BER(R4),(RO)+            ;Bus Error Register
MOVL    BIIC$L_EICR(R4),(RO)+           ;Error Interrupt Control Register
MOVL    BIIC$L_IDR(R4),(RO)+            ;Interrupt Destination Register
```

A.7 Loading a VAXBI Device Driver

The System Generation Utility (SYSGEN) loads the device driver into system virtual memory, creates additional data structures for the device unit, connects the device's interrupt vectors, and calls the device driver's controller-initialization routine and unit-initialization routine.

The *Writing a Device Driver for VAX/VMS* manual discusses the SYSGEN commands commonly used during driver loading. The following discussion pertains to those aspects of the loading process that specifically relate to the support of non-DIGITAL-supplied VAXBI devices.

Traditionally, SYSGEN is invoked near the end of system initialization, during the execution of the system startup command procedure (SYS$SYSTEM:STARTUP.COM). STARTUP.COM generally issues a SYSGEN AUTOCONFIGURE command with the result that SYSGEN scans various device tables to determine devices VAX/VMS expects to be connected to each VAXBI bus configured in the system. Ultimately, as the autoconfigure facility discovers the data structures associated with VAXBI devices recognized by VAX/VMS, it loads the associated device drivers and invokes their initialization routines.

Because the autoconfigure facility cannot recognize non-DIGITAL-supplied VAXBI devices, STARTUP.COM (or a later invocation of SYSGEN) must explicitly request that SYSGEN connect the device.[6] SYSGEN responds to such explicit requests by utilizing the data structures created by the INIADP module for the unknown VAXBI device to load the associated device driver and invoke its initialization routines.

For example, suppose that an unknown VAXBI device were located at node 3 on a given VAXBI bus and that the software device driver for this device were known as "ZZDRIVER". During INIADP processing, VAX/VMS would encounter an unknown type of VAXBI device at node 3 and would perform the following operations:

- Map the node space for node 3 into system virtual memory
- Construct various data structures to govern the future operation of this device

SYSGEN executes in response to the following commands:

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN>CONNECT ZZA0:/ADAPTER=3
```

SYSGEN performs the following activities:

1. Searches the list of ADPs in the system to find the ADP for this VAXBI node (node 3) and, in turn, locates the corresponding CRB and IDB by following pointers in the ADP.
2. Loads ZZDRIVER into system virtual memory. If the /DRIVER qualifier is specified, SYSGEN loads the specified driver instead.
3. Creates a UCB for device ZZA0: and places the address of the device's CRB in that UCB. SYSGEN also initializes other UCB fields at this time.
4. Sets the first entry in the IDB UCB array (IDB$L_UCBLST) to point to the new UCB.
5. Creates a DDB for the ZZA device/controller combination. This allows user programs to assign I/O channels to device ZZA0: later. This DDB, in turn, points to the location in memory where ZZDRIVER has been loaded and to the UCB for the ZZA0: device.
6. Calls the controller-initialization routine in ZZDRIVER at IPL 31.
7. Calls the unit-initialization routine in ZZDRIVER at IPL 31.

Note: **If you do not specify GENBI as the adapter type in the adapter argument to the DPTAB macro, the CONNECT command fails and issues the following error message:**

```
%SYSGEN-E-INVVEC, invalid or unspecified interrupt vector
```

A.8 Reference Material

The following sections include reference material concerning the contents of the BIIC register set and the routines discussed in this document.

[6] Because the autoconfigure facility is never called for a non-DIGITAL-supplied device, any unit-delivery routine that a VAXBI device driver may include is never called.

A.8.1 BIIC Register Definitions

Each VAXBI node is required to implement a minimum set of registers contained in specific locations within the node's node space. VAX/VMS automatically maps each node's node space at boot time and provides the macro $BIICDEF (in SYS$LIBRARY:LIB.MLB) to define offsets to the BIIC registers and their significant bit fields.

The contents of the BIIC registers are illustrated in Figure A–8 and described in Table A–1. See the *VAXBI Options Handbook* for a discussion of the BIIC and the rules for configuring its registers.

Note: **Fields marked "Reserved to DIGITAL" are reserved for future use by DIGITAL and should contain zeros.**

Figure A–8 Backplane Interconnect Interface Chip (BIIC) Registers

BIIC$L_DTREG
BIIC$L_BICSR
BIIC$L_BER
BIIC$L_EICR
BIIC$L_IDR
BIIC$L_IPIMR
BIIC$L_IPIDR
BIIC$L_IPISR
BIIC$L_SAR
BIIC$L_EAR
BIIC$L_BCICR
BIIC$L_WSR
BIIC$L_IPISTPF
unused
unused
unused
BIIC$L_UICR

(Continued on next page)

Figure A–8 (Cont.) Backplane Interconnect Interface Chip (BIIC) Registers

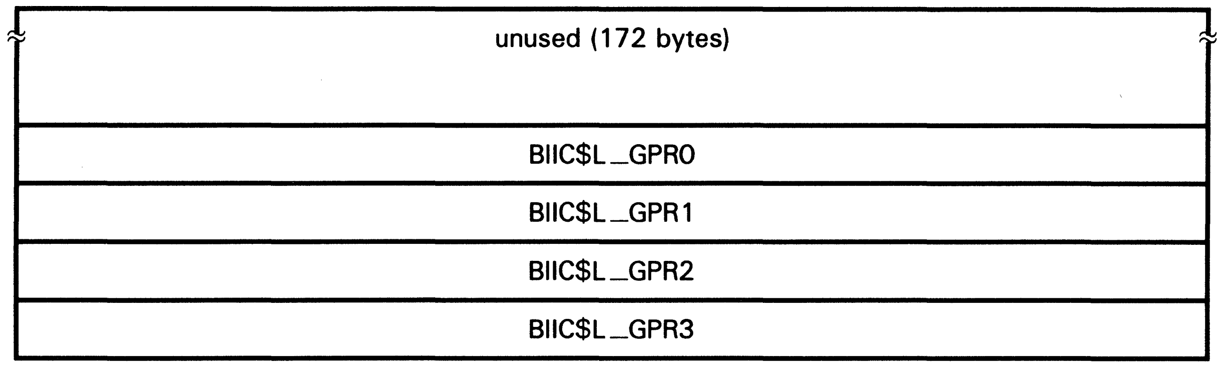


Table A–1 Contents of the BIIC Registers

Field Name	Contents	
BIIC$L_DTREG	Device Register.	
	BIIC$L_DTREG consists of the following two words:	
	BIIC$W_DEVTYPE	Device type. This field is written by device hardware and self-test microcode. It contains two bit fields:
		BIIC$V_MEMNODE (bits <14:8>) when clear, indicates a memory node.
		BIIC$V_NONDEC (bit 15), when clear indicates a DIGITAL-supplied device; it should be 1 otherwise.
	BIIC$W_REVCODE	Revision code.
BIIC$L_BICSR	VAXBI Control and Status Register.	
	The following fields are defined within BIIC$L_BICSR.	

Bit Field	Contents
BIIC$V_NODE_ID[1]	Node ID. This field is automatically loaded during the power-up sequence. Reserved to DIGITAL.
BIIC$V_ARBCNTL	Arbitration mode used by the node. Currently, all arbitration modes except dual round-robin arbitration are reserved to DIGITAL. Correspondingly, these two bits should be clear. When these two bits are set, arbitration is disabled, thus preventing a node from starting a VAXBI transaction.

[1]Read-only field.

Table A–1 (Cont.) Contents of the BIIC Registers

Field Name	Contents	
	BIIC$V_SEIE	Soft error interrupt enable. When set, this bit allows the node to generate an interrupt when the soft error summary bit (BIIC$V_SES) in this register is set.
	BIIC$V_HEIE	Hard error interrupt enable. When set, this bit allows the node to generate an interrupt when the hard error summary bit (BIIC$V_HES) in this register is set.
	BIIC$V_UWP	Unlock write pending. When set, this bit signals that the master port interface at this node has successfully completed an IRCI (Interlock Read with Cache Intent) transaction. The node clears this bit when it successfully completes a corresponding UWMCI (Unlock Write Mask with Cache Intent) instruction.
	<9> [1]	Reserved to DIGITAL. Must be zero.
	BIIC$V_SST	Node reset. This bit is normally used by diagnostics to initiate the BIIC internal self test. Prior to initiating a BIIC self test, a node should disable arbitration by setting both bits in BIIC$V_ARBCNTL. When BIIC$V_SST is set, the self-test status bit (BIIC$V_STS) in this register must also be set. Reads to BIIC$V_SST return a zero.
	BIIC$V_STS	Self-test status. When set, this bit indicates that the BIIC has passed its self test. The controller-initialization routine of a VAXBI device driver should inspect this bit and the BIIC$V_BROKE bit before proceeding with any VAXBI transactions. During the self-test sequence, BIIC$V_STS is automatically reset by the BIIC to allow the proper recording of the new self-test results at the end of self test.
	BIIC$V_BROKE[2]	Broke bit. When cleared by the device's self test, this bit indicates that device has passed its self test. The controller-initialization routine of a VAXBI device driver should inspect this bit and the BIIC$V_STS bit before proceeding with any VAXBI transactions.
	BIIC$V_INIT[2]	Initialization bit.
	BIIC$V_SES[1]	Soft error summary. When set, this bit indicates that one or more of the soft error bits in the Bus Error Register (BIIC$L_BER) is set.
	BIIC$V_HES[1]	Hard error summary. When set, indicates that one or more of the hard error bits in the Bus Error Register (BIIC$L_BER) is set.
	BIIC$V_BIICTYPE[1]	BIIC type. These bits <23:16> always contain 00000001.
	BIIC$V_BIICREVN[1]	BIIC revision number.

[1]Read-only field.

[2]Write-one-to-clear bit. Write-type transactions cannot set this bit.

Table A–1 (Cont.) Contents of the BIIC Registers

Field Name	Contents
BIIC$L_BER	Bus Error Register. The following bits are defined within BIIC$L_BER. Bits <30:16> are hard error bits and bits <2:0> are soft error bits.

Bit Field	Contents
BIIC$V_NPE[2]	Null bus parity error.
BIIC$V_CRD[2]	Corrected read data.
BIIC$V_IPE[2]	ID parity error.
BIIC$V_UPEN[1]	User parity enabled.
<14:4> [1]	Reserved to DIGITAL. Must be zero.
BIIC$V_ICE[2]	Illegal confirmation error.
BIIC$V_NEX[2]	Nonexistent address.
BIIC$V_BTO[2]	Bus timeout.
BIIC$V_STO[2]	Stall timeout.
BIIC$V_RTO[2]	Retry timeout.
BIIC$V_RDS[2]	Read data substitute.
BIIC$V_SPE[2]	Slave parity error.
BIIC$V_CPE[2]	Command parity error.
BIIC$V_IVE[2]	IDENT vector error.
BIIC$V_TDF[2]	Transmitter during fault.
BIIC$V_ISE[2]	Interlock sequence error.
BIIC$V_MPE[2]	Master parity error.
BIIC$V_CTE[2]	Control transmit error.
BIIC$V_MTCE[2]	Master transmit check error.
BIIC$V_NMR[2]	NO ACK to multiresponder command received.
<31> [1]	Reserved to DIGITAL. Must be zero.

Field Name	Contents
BIIC$L_EICR	Error Interrupt Control Register. This register supplies information the node uses to request and monitor the status of both BIIC-detected and forced-error interrupts; that is, those interrupts signaled by either the setting of a bit in the Bus Error Register (BIIC$L_BER) or the setting of the force bit (BIIC$V_EIFORCE) in this register, respectively. The node can initiate BIIC-detected error-interrupt requests only if the appropriate error-interrupt enables (BIIC$V_SEIE and/or BIIC$V_HEIE) are set in the VAXBI Control and Status Register (BIIC$L_BICSR).

[1]Read-only field.

[2]Write-one-to-clear bit. Write-type transactions cannot set this bit.

Table A–1 (Cont.) Contents of the BIIC Registers

Field Name	Contents
	The following fields are defined within BIIC$L_EICR.

Bit Field	Contents
<1:0> [1]	Reserved to DIGITAL. Must be zero.
BIIC$V_EIVECTOR	12-bit vector used in error interrupt sequences.
<15:14> [1]	Reserved to DIGITAL. Must be zero.
BIIC$V_LEVEL	These four bits (<19:16>) correspond to the four interrupt levels (INT <7:4>) of the VAXBI bus. A set bit causes the corresponding level to be used when INTR commands under control of this register are transmitted.
BIIC$V_EIFORCE	Force bit. When set, this bit posts an error interrupt request in the same way as a bit set in the Bus Error Register (BIIC$L_BER), except that the request is not qualified by the bits BIIC$V_HEIE and BIIC$V_SEIE in BIIC$L_BICSR.
BIIC$V_EISENT[2]	INTR sent.
<22> [1]	Reserved to DIGITAL. Must be zero.
BIIC$V_EIINTC[2]	INTR complete. When set, this bit indicates that the vector for an error interrupt has been successfully transmitted or an INTR command sent under the control of this register has been successfully aborted.
BIIC$V_EIINTAB[2]	INTR abort. When set, this bit indicates that an INTR command under the control of this register has been aborted (that is, a NO ACK or illegal confirmation code has been received). This bit is a status bit set by the BIIC and can be reset only by the user interface.
<31:25> [1]	Reserved to DIGITAL. Must be zero.

Field Name	Contents
BIIC$L_IDR	Interrupt Destination Register. The low-order word of this register indicates which nodes are to be selected by INTR commands.
BIIC$L_IPIMR	Interprocessor Interrupt Mask Register. The high-order word of this register indicates which nodes are permitted to send IPINTRs to this node.
BIIC$L_IPIDR	Force-bit IPINTR/STOP Destination Register. The low-order word of this register indicates which nodes are to be targeted by force-bit IPINTR or STOP commands sent by this node.
BIIC$L_IPISR	[2]IPINTR Source Register. The BIIC stores in the high-order word of this register the decoded ID of a node that sends an IPINTR command to this node.
BIIC$L_SAR	Starting Address Register. The Starting Address Register and Ending Address Register define storage blocks in either memory or I/O space. They must not be configured to include node space or multicast space. The low-order 18 bits of this register must be zero. This means that memories are multiples of 256K bytes. Software should set up the Starting Address Register before the Ending Address Register.

[1]Read-only field.

[2]Write-one-to-clear bit. Write-type transactions cannot set this bit.

Table A–1 (Cont.) Contents of the BIIC Registers

Field Name	Contents
BIIC$L_EAR	Ending Address Register. The low-order 18 bits of this register must be zero. This means that memories are multiples of 256K bytes. Software should set up the Starting Address Register before the Ending Address Register. See the description of the Starting Address Register (BIIC$L_SAR).
BIIC$L_BCICR	BCI Control Register. The following fields are defined within BIIC$L_BCICR.

Bit Field	Contents
<2:0> [1]	Reserved to DIGITAL. Must be zero.
BIIC$V_RTOEVEN	RTO EV enable.
BIIC$V_PNXTEN	Pipeline NXT enable.
BIIC$V_IPINTREN	IPINTR enable.
BIIC$V_INTREN	INTR enable.
BIIC$V_BICSREN	BIIC CSR Space enable.
BIIC$V_UCSREN	User Interface CSR Space enable.
BIIC$V_WINVALEN	WRITE Invalidate enable.
BIIC$V_INVALEN	INVAL enable.
BIIC$V_IDENT	IDENT enable.
BIIC$V_RESEN	RESERVED enable.
BIIC$V_STOPEN	STOP enable.
BIIC$V_BDCSTEN	BDCST enable.
BIIC$V_MSEN	Multicast Space enable.
BIIC$V_IPINTRF	IPINTR/STOP force.
BIIC$V_BURSTEN	Burst enable.
<31:18> [1]	Reserved to DIGITAL. Must be zero.

Field Name	Contents
BIIC$L_WSR	Write Status Register.

[1]Read-only field.

Table A–1 (Cont.) Contents of the BIIC Registers

Field Name	Contents
	The following fields are defined within BIIC$L_UICR.

Bit Field	Contents
<27:0> [1]	Reserved to DIGITAL. Must be zero.
BIIC$V_GPR0[2]	Indicates that a VAXBI transaction has written to General Purpose Register 0 (BIIC$L_GPR0).
BIIC$V_GPR1[2]	Indicates that a VAXBI transaction has written to General Purpose Register 1 (BIIC$L_GPR1).
BIIC$V_GPR2[2]	Indicates that a VAXBI transaction has written to General Purpose Register 2 (BIIC$L_GPR2).
BIIC$V_GPR3[2]	Indicates that a VAXBI transaction has written to General Purpose Register 3 (BIIC$L_GPR3).

Field Name	Contents
BIIC$L_IPISTPF	Force-Bit IPINTR/STOP Command Register. The following fields are defined within BIIC$L_IPISTPF.

Bit Field	Contents
<10:0> [1]	Reserved to DIGITAL. Must be zero.
BIIC$V_MIDEN	Master ID Enable.
BIIC$V_CMD	These four bits indicate the command code for either an IPINTR or STOP transaction that is initiated by setting the IPINTR/STOP force bit (BIIC$V_INTRF in BIIC$L_BCICR).
<31:16> [1]	Reserved to DIGITAL. Must be zero.

Field Name	Contents
BIIC$L_UICR	User Interface Interrupt Control Register. This register controls the operation of interrupts initiated by the device.

[1]Read-only field.

[2]Write-one-to-clear bit. Write-type transactions cannot set this bit.

Table A–1 (Cont.) Contents of the BIIC Registers

Field Name	Contents	
	The following fields are defined within BIIC$L_UICR.	
	Bit Field	**Contents**
	<1:0> [1]	Reserved to DIGITAL. Must be zero.
	BIIC$V_UIVECTOR	These 12 bits contain the vector used during user interface interrupt sequences (unless the external vector bit (BIIC$V_EXVECTOR in BIIC$L_UICR) is set). The vector is transmitted when this node wins an IDENT arbitration that matches the conditions given in BIIC$L_UICR.
	<14> [1]	Reserved to DIGITAL. Must be zero.
	BIIC$V_EXVECTOR	When set, the BIIC solicits the interrupt vector from the node rather than transmitting the vector contained in BIIC$L_UICR.
	BIIC$V_UIFORCE	These four bits correspond to the four interrupt levels (INT <7:4>). When a bit is set, the BIIC generates an interrupt at the indicated level.
	BIIC$V_UISENT[2]	These four bits correspond to the four interrupt levels (INT <7:4>). A set bit indicates that an INTR command for the corresponding level has been successfully transmitted.
	BIIC$V_UIINTC[2]	These four bits correspond to the four interrupt levels (INT <7:4>). A set bit indicates that the vector for an interrupt at the corresponding level has been successfully transmitted or that an INTR command sent under the control of this register has been successfully aborted.
	BIIC$V_UIINTAB[2]	These four bits correspond to the four interrupt levels (INT <7:4>). A set bit indicates that an INTR command at the corresponding level, sent under the control of this register, has been aborted (that is, a NO ACK or illegal confirmation code has been received).
BIIC$L_GPR0	General Purpose Register 0	
BIIC$L_GPR1	General Purpose Register 1	
BIIC$L_GPR2	General Purpose Register 2	
BIIC$L_GPR3	General Purpose Register 3	

[1]Read-only field.

[2]Write-one-to-clear bit. Write-type transactions cannot set this bit.

A.8.2 IOC$ALLOSPT

Drivers for non-DIGITAL-supplied VAXBI device drivers use the executive routine IOC$ALLOSPT when they need to map a portion of a device's node space to system virtual address space. See Section A.4.3 for a discussion of a driver's use of IOC$ALLOSPT to map a device's VAXBI window space. Tables A-2 and A-3 describe inputs to and output from the routine.

IOC$ALLOSPT

MODULE: IOSUBNPAG

Table A–2 Input

Register	Contents
R1	Number of system page table entries to be allocated.

Field	Contents
BOO$GL_SPTFREL	Lowest free virtual page number.
BOO$GL_SPTFREH	Highest free virtual page number.

IPL at execution: IPL$_SYNCH

Table A–3 Output

Register	Contents
R0	SS$_NORMAL or 0
R1	Number of allocated system page table entries.
R2	Starting system virtual page number (SVPN) allocated.
R3	Address of the base of the system page table (MMG$GL_SPTBASE).

IPL at exit: IPL$_SYNCH

Index

Reader's Comments

VAX/VMS Supplemental Information,Version 4.7
AA–KX21A–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)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

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 --

digital™

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 --

ᵢeader's Comments

VAX/VMS Supplemental Information,Version 4.7
AA–KX21A–TE

ᵢease use this postage-paid form to comment on this manual. If you require a written reply to a software ·oblem and are eligible to receive one under Software Performance Report (SPR) service, submit your ›mments on an SPR form.

hank you for your assistance.

rate this manual's:	Excellent	Good	Fair	Poor
ccuracy (software works as manual says)	☐	☐	☐	☐
ompleteness (enough information)	☐	☐	☐	☐
larity (easy to understand)	☐	☐	☐	☐
rganization (structure of subject matter)	☐	☐	☐	☐
gures (useful)	☐	☐	☐	☐
‹amples (useful)	☐	☐	☐	☐
ıdex (ability to find topic)	☐	☐	☐	☐
ıge layout (easy to find information)	☐	☐	☐	☐

would like to see more/less ______________________________

____

____

'hat I like best about this manual is ______________________________

____

____

'hat I like least about this manual is ______________________________

____

____

found the following errors in this manual:

ıge	Description
____	______________________________
____	______________________________
____	______________________________
____	______________________________
____	______________________________

dditional comments or suggestions to improve this manual:

____

____

____

____

am using **Version** ______ of the software this manual describes.

ame/Title ______________________________ Dept. __________

ompany ______________________________ Date __________

ᵢailing Address ______________________________

______________________________ Phone __________

------ Do Not Tear - Fold Here and Tape --

digital™

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 --