UPDATE NOTICE NO.1

VAX/VMS I/O User's Guide
AD-D028A-T1

February 1979

Insert this Update Notice page in the manual as a means of maintaining an up-to-date record of changes to the manual.

NEW AND CHANGED INFORMATION

This update reflects software changes and additions made in VAX/VMS Version 1.5, in particular the addition of a driver to support the LPA-11K Laboratory Peripheral Accelerator.

INSTRUCTIONS

Place the following pages in the VAX/VMS I/O User's Guide as replacements for, or additions to, current pages. The additions made on replacement pages are indicated in the outside margin by change bars (▌).

Old Page	New Page
Title Page/Copyright Page	Title Page/Copyright Page
iii/iv through xi/blank	iii/iv through xi/xii
-	xiii/blank
1-3/1-4	1-3/1-4
1-5/1-6	1-5/1-6
1-11/1-12	1-11/1-12
1-15/1-16 through 1-21/1-22	1-15/1-16 through 1-21/1-22
2-1/2-2	2-1/2-2
2-3/2-4	2-3/2-4
2-9/2-10	2-9/2-10
2-15/2-16	2-15/2-16
-	10-1/10-2 through 10-47/10-48
B-1/B-2	B-1/B-2
B-3/B-4	B-3/B-4
-	B-5/blank
Index-1/Index-2 through	Index-1/Index-2 through
Index-5/Index-6	Index-5/Index-6
-	Index-7/blank

Additional copies of this update to the VAX/VMS I/O User's Guide may be ordered from the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754. Order Code: AD-D028A-T1. The order code of the base manual is AA-D028A-TE.

February 1979

This document contains the information necessary to interface directly with the I/O device drivers supplied as part of the VAX/VMS operating system. Several examples of programming techniques are included. This document does not contain information on I/O operations using VAX-11 Record Management Services.

VAX/VMS
I/O User's Guide

Order No. AA-D028A-TE
and
Update Notice No. 1 (AD-D028A-T1)

SUPERSESSION/UPDATE INFORMATION:	This document includes Update Notice No. 1 (AD-D028A-T1).
OPERATING SYSTEM AND VERSION:	VAX/VMS V01.5
SOFTWARE VERSION:	VAX/VMS V01.5

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, August 1978
Updated, February 1979

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

CONTENTS

CONTENTS (Cont.)

FIGURES

Page

FIGURES (Cont.)

Page

TABLES

Page

TABLES (Cont.)

PREFACE

MANUAL OBJECTIVES

This manual provides users of the VAX/VMS operating system with the information necessary to interface directly with the I/O device drivers supplied as part of the operating system. It is not the objective of this manual to provide the reader with information on all aspects of VAX/VMS input/output (I/O) operations.

INTENDED AUDIENCE

This manual is intended for system programmers who want to take advantage of the time and/or space savings that result from direct use of the I/O devices. Readers are expected to have some experience with either VAX-11 FORTRAN IV-PLUS or VAX-11 MACRO assembly language. Users of VAX/VMS who do not require such detailed knowledge of I/O drivers can use the device-independent services described in the VAX-11 Record Management Services Reference Manual.

STRUCTURE OF THIS DOCUMENT

This manual is organized into ten chapters and two appendixes, as follows:

- Chapter 1 contains introductory information. It provides overviews of VAX/VMS I/O operations; I/O system services; and I/O quotas, privileges, and protection. This chapter also introduces I/O function encoding and how to make I/O requests, and describes how to obtain information on the different devices.

- Chapters 2 through 8 and 10 describe the use of all the I/O device drivers supported by VAX/VMS:

  - Chapter 2 deals with the terminal driver

  - Chapter 3 deals with disk drivers

  - Chapter 4 deals with magnetic tape drivers

  - Chapter 5 deals with the line printer driver

  - Chapter 6 deals with the card reader driver

- Chapter 7 deals with the mailbox driver
- Chapter 8 deals with the DMC11 driver
- Chapter 10 deals with the LPA11-K driver

- Chapter 9 describes the Queue I/O (QIO) interface to file system ancillary control processes (ACPs).
- Appendix A describes the QIO functions that are common to the disk and magnetic tape drivers and the QIO ACP interface.
- Appendix B summarizes the QIO function codes, arguments, and function modifiers used by the different device drivers.

ASSOCIATED DOCUMENTS

The following documents may also be useful:

- VAX/-11 Information Directory - contains a complete list of all VAX-11 documents
- VAX/VMS System Services Reference Manual
- VAX-11 Linker Reference Manual
- VAX-11 Software Handbook
- PDP-11 Peripherals Handbook
- VAX-11 FORTRAN IV-PLUS User's Guide
- VAX-11 MACRO User's Guide
- VAX-11 Record Management Services Reference Manual
- LPA11-K Laboratory Peripheral Accelerator User's Guide

CONVENTIONS USED IN THIS MANUAL

The following conventions are used in this manual:

- Brackets ([]) in QIO requests enclose optional arguments. For example:

```
IO$_CREATE P1,[P2],[P3],[P4],[P5]
```

- Horizontal ellipses (...) indicate that characters or QIO arguments that are not pertinent to the example have been omitted. For example:

```
(that is, 8, 16, 24,...).
```

- Vertical ellipses in coding examples indicate that lines of code not pertinent to the example are omitted. For example:

```
TTCHAN:  .BLKW 1
            .
            .
            .
         $ASSIGN_S DEVNAM=TTNAME,CHAN=TTCHAN
```

- Hyphens (-) in coding examples indicate that additional arguments to the QIO request are provided on the following line(s). For example:

```
$QIO_S  FUNC=#IO$_WRITEPBLK,-          ;FUNCTION IS
        -                              ;WRITE PHYSICAL
        CHAN=W^TTCHAN1,-               ;TO TTCHAN 1
        EFN=#1,-                       ;EVENT FLAG 1
        P1=W^ASTMSG,-                  ;P1 = BUFFER
        P2=#ASTMSGSIZE                 ;P2 = BUFFER SIZE
```

- Angle brackets (<>) enclose keys on the terminal keyboard. For example:

  ESC <0> <20-2F>...<40-7E>

- Unless otherwise noted, all numbers in the text are assumed to be decimal. In coding examples, the radix -- binary, octal, decimal, or hexadecimal -- will be explicitly indicated.

See the VAX/VMS System Services Reference Manual for detailed information on all these system services and examples of their use. The VAX/VMS System Services Reference Manual also contains information on physical and logical device-naming conventions.

1.4 QUOTAS, PRIVILEGES, AND PROTECTION

To preserve the integrity of the system, VAX/VMS I/O operations are performed under the constraints of quotas, privileges, and protection.

Quotas establish a limit on the number and type of I/O operations that a process can perform concurrently. They ensure that all users have an equitable share of system resources and usage.

Privileges are granted to a user to allow the performance of certain I/O-related operations; for example, create a mailbox and perform logical I/O to a file-structured device. Restrictions on user privilege protect the integrity and performance of both the operating system and the services provided other users.

Protection is used to control access to files and devices. Device protection is provided in much the same way as file protection: shareable and nonshareable file devices and shareable nonfile devices such as mailboxes, are protected by protection masks. Nonshareable, nonfile devices such as terminals, can be accessed if they are not allocated to another process.

The Set Resource Wait Mode ($SETRWM) system service allows a process to select either of two modes when an attempt to exceed a quota occurs. In the enabled (default) mode, the process waits until the required resource is available before continuing. In the disabled mode, the process is notified immediately by a system service status return that an attempt to exceed a quota has occurred. Waiting for resources is transparent to the process when resource wait mode is enabled; no explicit action is taken by the process when a wait is necessary.

The different types of I/O-related quotas, privileges, and protection are described in the following paragraphs.

1.4.1 Buffered I/O Quota

The buffered I/O quota specifies the maximum number of concurrent buffered I/O operations a process can have active. In a buffered I/O operation, the user's data is buffered in system dynamic memory. The driver deals with the system buffer and not the user buffer. Buffered I/O is used for terminal, line printer, card reader, and mailbox transfers. The user's buffer does not have to be locked in memory for a buffered I/O operation.

The buffered I/O quota value is established in the user authorization file by the system manager or by the process's creator. Resource wait mode is entered if enabled by the Set Resource Wait Mode system service and an attempt to exceed the buffered I/O quota is made.

1.4.2 Buffered I/O Byte Count Quota

The buffered I/O byte count quota specifies the maximum amount of buffer space that can be concurrently consumed from system dynamic memory for buffering I/O requests. All buffered I/O requests require system dynamic memory in which the actual I/O operation takes place.

The buffered I/O byte count quota is established in the user authorization file by the system manager or by the process's creator. Resource wait mode is entered if enabled by the Set Resource Wait Mode system service and an attempt to exceed the buffered I/O byte count quota is made.

1.4.3 Direct I/O Quota

The direct I/O quota specifies the maximum number of concurrent direct, that is, unbuffered, I/O operations that a process can have active. In a direct I/O operation, data is moved directly to or from the user buffer. Direct I/O is used for disk, magnetic tape, and DMC11 transfers. For direct I/O, the user's buffer must be locked in memory during the transfer.

The direct I/O quota value is established in the user authorization file by the system manager or by the process's creator. Resource wait mode is entered if enabled by the Set Resource Wait Mode system service and an attempt to exceed the direct I/O quota is made.

1.4.4 AST Quota

The AST quota specifies the maximum number of asynchronous system traps that a process can have outstanding. The quota value is established in the user authorization file by the system manager or by the process's creator. There is never an implied wait for this resource.

1.4.5 Physical I/O Privilege (PHY_IO)

Physical I/O privilege allows a process to perform physical I/O operations on a device. Physical I/O privilege also allows a process to perform logical I/O operations on a device. (Figures 1-1 and 1-2 show the use of physical I/O privilege in greater detail.)

1.4.6 Logical I/O Privilege (LOG_IO)

Logical I/O privilege allows a process to perform logical I/O operations on a device. A process can also perform physical operations on a device if the process has logical I/O privilege, the volume is mounted foreign, and the volume protection mask allows access to the device. (Figures 1-1 and 1-2 show the use of logical I/O privilege in greater detail.)

1.4.7 Mount Privilege

Mount privilege allows a process to use the IO$_MOUNT function to perform mount operations on disk and magnetic tape devices. IO$_MOUNT is used in ACP interface operations (see Chapter 9).

1.4.8 Volume Protection

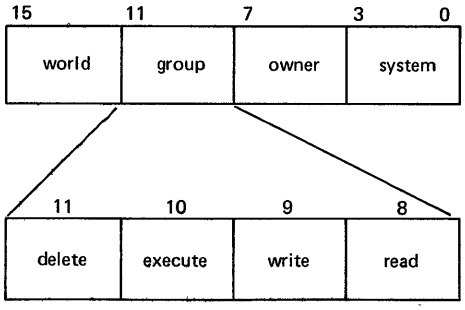
Volume protection protects the integrity of mailboxes and both foreign and Files-11 structured volumes. Volume protection for a foreign volume is established when the volume is mounted. Volume protection for a Files-11 structured volume is established when the volume is initialized. (The protection can be overridden when the volume is mounted if the process that is mounting the volume has the override volume protection privilege.)

Mailbox protection is established by the $CREMBX system service protection mask argument.

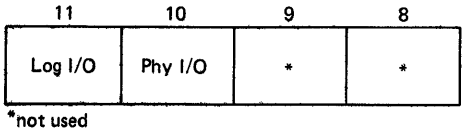
Protection for structured volumes and mailboxes is provided by a volume protection mask that contains four 4-bit fields. These fields correspond to the four classes of users that are permitted to access the volume. (User classes are based on the volume owner's user identification code (UIC).)

The 4-bit fields are interpreted differently for volumes that are mounted as structured (that is, volumes serviced by an Ancillary Control Process (ACP)) and volumes that are mounted as foreign.

The 4-bit fields have the following format for volumes mounted as structured:



The 4-bit fields have the following format for volumes mounted as foreign:



*not used

Usually, volume protection is meaningful only for read and write operations.

1.5 SUMMARY OF VAX/VMS QIO OPERATIONS

VAX/VMS provides QIO operations that perform three basic I/O functions: read, write, and set mode. The read function transfers data from a device to a user-specified buffer. The write function transfers data in the opposite direction - from a user-specified buffer to the device. For example, in a read QIO function to a terminal device, a user-specified buffer is filled with characters received from the terminal. In a write QIO function to the terminal, the data in a user-specified buffer is transferred to the terminal where it is displayed.

The set mode QIO function is used to control or describe the characteristics and operation of a device. For example, a set mode QIO function to a line printer can specify either uppercase or lowercase character format. Not all QIO functions are applicable to all types of devices. The line printer, for example, cannot perform a read QIO function.

1.6 PHYSICAL, LOGICAL, AND VIRTUAL I/O

I/O data transfers can occur in any one of three device addressing modes: physical, logical, or virtual. Any process with device access allowed by the volume protection mask can perform logical I/O on a device that is mounted foreign; physical I/O requires privilege. Virtual I/O does not require privilege; however, intervention by an ACP to control user access may be necessary if the device is under ACP control. (ACP functions are described in Chapter 9.)

1.6.1 Physical I/O Operations

In physical I/O operations, data is read from and written to the actual, physically addressable units accepted by the hardware; for example, sectors on a disk or binary characters on a terminal in the PASSALL mode. This mode allows direct access to all device-level I/O operations.

Physical I/O requires that one of the following conditions be met:

- The issuing process has physical I/O privilege (PHY_IO)
- The issuing process has logical I/O privilege (LOG_IO), the device is mounted foreign, and the volume protection mask allows physical access to the device

If neither of these conditions is met, the physical I/O operation is rejected by the QIO system service with a status return of SS$_NOPRIV (no privilege). Figure 1-1 illustrates the physical I/O access checks in greater detail.

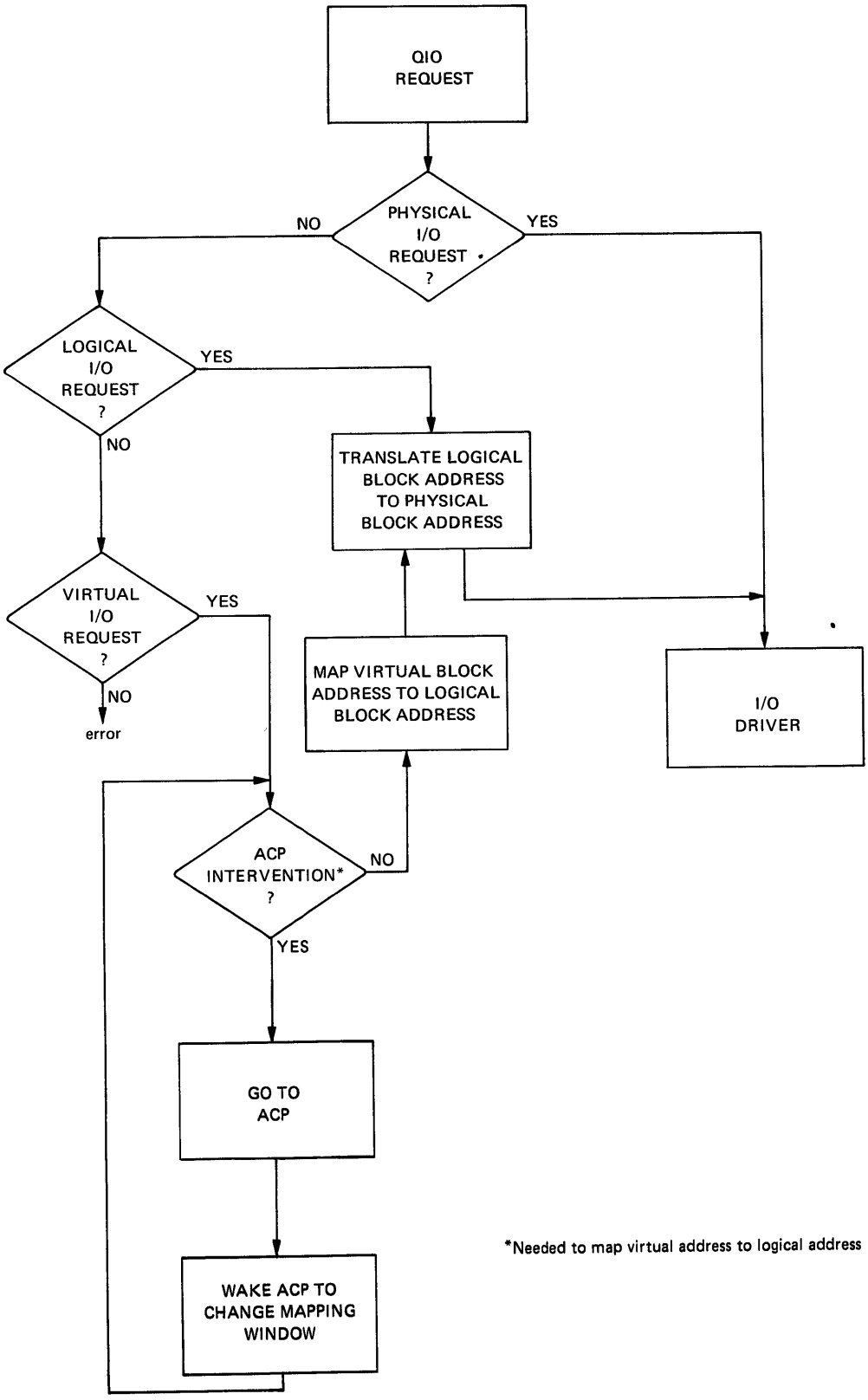


Figure 1-3 Physical, Logical, and Virtual I/O

1.7 I/O FUNCTION ENCODING

I/O functions fall into three groups that correspond to the three I/O device addressing modes (physical, logical, and virtual) described in Section 1.6. Depending on the device to which it is directed, an I/O function can be expressed in one, two, or all three modes.

I/O functions are described by 16-bit, symbolically-expressed values that specify the particular I/O operation to be performed and any optional function modifiers. Figure 1-4 shows the format of the 16-bit function value.

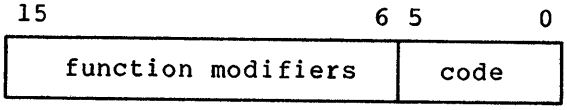


Figure 1-4 I/O Function Format

Symbolic names for I/O function codes are defined by the $IODEF macro, as described in the VAX/VMS System Services Reference Manual.

1.7.1 Function Codes

The low-order 6 bits of the function value are a code that specifies the particular operation to be performed. For example, the code for read logical block is expressed as IO$_READLBLK. Table 1-1 lists the symbolic values for read and write I/O functions in the three transfer modes.

Table 1-1
Read and Write I/O Functions

Physical I/O	Logical I/O	Virtual I/O
IO$_READPBLK IO$_WRITEPBLK	IO$_READLBLK IO$_WRITELBLK	IO$_READVBLK IO$_WRITEVBLK

The set mode I/O function has a symbolic value of IO$_SETMODE.

Function codes are defined for all supported devices. Although some of the function codes (for example, IO$_READVBLK and IO$_WRITEVBLK) are used with several types of devices, most are device dependent. That is, they perform functions specific to particular types of devices. For example, IO$_CREATE is a device-dependent function code; it is used only with file-structured devices such as disks and magnetic tapes. Chapters 2 through 8 and 10 provide complete descriptions of the functions and function codes.

1.7.2 Function Modifiers

The high-order 10 bits of the function value are function modifiers. These are individual bits that alter the basic operation to be performed. For example, the function modifier IO$M_NOECHO can be specified with the function IO$_READLBLK to a terminal. When used

The number of pending I/O requests, the amount of buffer space, and the number of outstanding ASTs that a process can have are controlled by quotas.

Each I/O request causes an I/O request packet to be allocated from system dynamic memory. Additional memory is allocated under the following circumstances:

- The I/O request function is an ACP function
- The target device is a buffered I/O device
- The target device is a network I/O device

After an I/O request is queued, the system does not require the issuing process to wait for the I/O operation to complete. If the process that issued the QIO request cannot proceed until the I/O completes, an event flag can be used to synchronize I/O completion (see Sections 1.8.6.1 and 1.9.1). In this case, the process should request the Wait for Single Event Flag ($WAITFR) system service at the point where synchronization must occur: that is, where I/O completion is required.

$WAITFR specifies an event flag for which the process is to wait. (The $WAITFR event flag must have the same number as the event flag used in the QIO request.) The process then waits while the I/O operation is performed. On I/O completion, the event flag is set and the process is allowed to resume operation.

Other ways to achieve this synchronization include the use of the $QIOW system service and ASTs, described in Sections 1.8.5 and 1.9.3, respectively. In addition, the I/O status block can be specified and checked if the user wants to determine whether the I/O operation completed without an error, regardless of whether or not the process waits for I/O completion (see Section 1.9.2.)

The QIO system service is accompanied by up to six device/ function independent and six device/function dependent arguments. Section 1.8.6 below describes device/function independent arguments. The device/function dependent arguments (P1 through P6) are potentially different for each device/function combination. However, similar functions that are performed by all devices have identical arguments. Furthermore, all functions performed by a particular class of device are identical. Device/function dependent arguments are described in more detail for the individual devices in Chapters 2 through 8 and 10.

1.8.4 $QIO Macro Format

The general format for the $QIO macro, using position-dependent arguments, is:

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

The first six arguments are device/function independent. If keyword arguments are used, they can be written in any order. Arguments P1 through P6 are device/function dependent. The chan and func arguments must be specified in each request; arguments enclosed in brackets ([]) are optional.

The following example illustrates a typical QIO request using keyword arguments:

```
$QIO_S      EFN=#1,-                    ;EVENT FLAG 1
            CHAN=TTCHAN1,-              ;CHANNEL
            FUNC=#IO$_WRITEVBLK,-       ;VIRTUAL WRITE
            P1=BUFADD,-                 ;BUFFER ADDRESS
            P2=#BUFSIZE                 ;BUFFER SIZE
```

1.8.5 $QIOW Macro Format

The Queue I/O Request and Wait For Event Flag ($QIOW) system service macro combines the $QIO and $WAITFR system services. It eliminates any need for explicit I/O synchronization by automatically waiting until the I/O operation is completed before returning control to the process. Thus, $QIOW provides a simpler way to synchronize the return to the originating process when the process cannot proceed until the I/O operation is completed.

The $QIOW macro has the same device/function independent and device/function dependent arguments as the $QIO macro:

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

1.8.6 $QIO and $QIOW Arguments

Table 1-2 lists the $QIO and $QIOW device/function independent arguments and their meanings. Additional information is provided in the paragraphs following the table and in the VAX/VMS System Services Reference Manual.

Table 1-2
Device/Function Independent Arguments

Argument	Meaning
efn (event flag number)	The number of the event flag that is to be cleared when the I/O function is queued and set when it is completed. This argument is optional in the macro form; if not specified, efn defaults to 0.
chan (channel number)	The number of the I/O channel to which the request is directed. The channel number is obtained from either the $ASSIGN or $CREMBX system service. This argument is mandatory in the macro form.
func (function value)	The 16-bit function code and modifier value that specifies the operation to be performed. This argument is mandatory in the macro form.
iosb (I/O status block)	The address of a quadword I/O status block to receive the final I/O status. This argument is optional in the macro form.

(Continued on next page)

Table 1-2 (Cont.)
Device/Function Independent Arguments

Argument	Meaning
astadr (AST address)	The entry point address of an AST routine to be asynchronously executed when the I/O completes. This argument is optional in the macro form.
astprm (AST parameter)	The 32-bit value to be passed to the AST routine as an argument when the I/O completes. It can be used to assist the routine in identifying the particular AST. This argument is optional in the macro form.

1.8.6.1 **Event Flag Number Argument** - The event flag number (efn) argument is the number of the event flag to be associated with the I/O operation. It is optional in a $QIO or $QIOW macro. The specified event flag is cleared when the request is issued and set when the I/O operation completes. The specified event flag is also set if the service terminates without queuing the I/O request.

If the process requested the $QIOW system service, execution is automatically suspended until the I/O completes. If the process requested the QIO system service (with no subsequent $WAITFR, $WFLOR, or $WFLAND macro), process execution proceeds in parallel with the I/O. As the process continues to execute, it can test the event flag at any point by using the Read Event Flags ($READEF) system service.

Event flag numbers must be in the range of 0 through 127 (however, event flags 24 through 31 are reserved for system use). If no specific event flag is desired, the efn argument can be omitted from the macro. In that case, efn defaults to 0.

1.8.6.2 **Channel Number Argument** - The channel number (chan) argument represents the channel number of the physical device to be accessed by the I/O request. It is required for all $QIO and $QIOW requests. The association between the physical device and the channel is specific to the process issuing the I/O request. The channel number is obtained from the $ASSIGN or $CREMBX system service (as described above in Section 1.8.1).

1.8.6.3 **Function Argument** - The function (func) argument defines the logical, virtual, or physical I/O operation to be performed when the $QIO or $QIOW system service is requested. It is required for all QIO and QIOW requests. The argument consists of a 16-bit function code and function modifier. Up to 64 function codes can be defined. Function codes are defined for all supported device types; most of the codes are device dependent. The function arguments for each I/O driver are described in more detail in Chapters 2 through 8 and 10.

1.8.6.4 **I/O Status Block Argument** - The I/O status block (iosb) argument specifies the address of the I/O status block to be associated with the I/O request. It is optional in the QIO and QIOW macros. If omitted, the iosb value is 0 which indicates no iosb address is supplied. This block is a quadword that receives the final completion status of the I/O request. Section 1.9.2 describes the I/O status block in more detail.

1.8.6.5 **AST Address Argument** - The AST address (astadr) argument specifies the entry point address of an AST routine to be executed when the I/O operation is complete. If omitted, the astadr value is 0 which indicates no astadr address is supplied. This argument is optional and can be used to interrupt a process to execute special code at I/O completion. When the I/O operation completes, the AST service routine is CALLed at the address specified in the astadr argument. The AST service routine is then executed in the access mode from which the QIO service was requested.

1.8.6.6 **AST Parameter Argument** - The AST parameter (astprm) argument is an optional, 32-bit arbitrary value that is passed to the AST service routine when I/O completes, to assist the routine in identifying the particular AST. A typical use of the astprm argument might be the address of a user control block. If omitted, the astprm value is 0.

1.8.6.7 **Device/Function Dependent Arguments** - Up to six device/function dependent arguments (P1 through P6) can be included in each QIO request. The arguments for terminal read function codes show a typical use of P1 through P6:

P1 = buffer address

P2 = buffer size

P3 = timeout count (for read with timeout)

P4 = read terminator descriptor block address

P5 = prompt string buffer address

P6 = prompt string buffer size

P1 is always treated as an address. Therefore, in the _S form of the macro, P1 always generates a PUSHAB instruction. P2 through P6 are always treated as values. In the _S form of the macro, these arguments always generate PUSHL instructions.

Inclusion of the device/function dependent arguments in a QIO request depends on the physical device unit and the function specified. A user who wants to specify only a channel, an I/O function code, and an address for AST routine might issue the following:

```
$QIO_S    CHAN=XYCHAN,FUNC=#IO$_READVBLK,-
          ASTADR=XYAST,P1=BUFADR,P2=#BUFLEN
```

In this example, XYCHAN is the address of the word containing the channel to which the request is directed; IO$_READVBLK is the function code; and XYAST is the AST entry point address. BUFADR and BUFLEN are the device/function dependent arguments for an input buffer.

1.8.7 $INPUT and $OUTPUT Macro Format and Arguments

The $INPUT and $OUTPUT macros simplify the use of the $QIOW macro. These macros generate code to perform virtual operations, using the IO$_READVBLK and IO$_WRITEVBLK function codes (the function code is automatically specified in the request), and wait for I/O completion. The macro formats and arguments are:

```
$INPUT   chan,length,buffer,[iosb],[efn]
$OUTPUT  chan,length,buffer,[iosb],[efn]
```

Table 1-3 lists the $INPUT and $OUTPUT arguments and their meanings.

Table 1-3
$INPUT and $OUTPUT Arguments

Argument	Meaning
chan	The channel on which the I/O operation is to be performed.
length	The length of the input or output buffer.
buffer	The address of the input or output buffer.
iosb	The address of the quadword that receives the completion status of the I/O operation. This argument is optional.
efn	The number of the event flag for which the process waits. This argument is optional; if not specified, efn defaults to 0.

Both the iosb and efn arguments are optional; all other arguments must be included in each macro. Note that the order of the length and buffer arguments is opposite that of the QIO and QIOW P1 and P2 arguments. Also note that $INPUT and $OUTPUT do not have the astadr and astprm arguments; neither of these operations can conclude in an AST.

1.8.8 Status Returns for System Services

On completion of a system service call, the completion status is returned as a longword value in register R0, shown in Figure 1-6. (System services save the data in all registers except R0 and R1.)

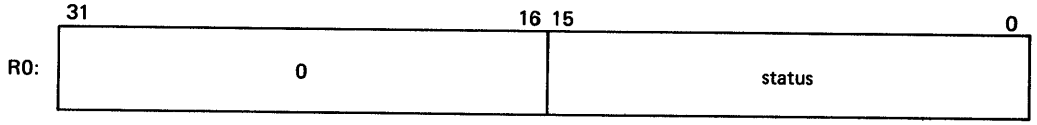


Figure 1-6 System Service Status Return

Completion status is indicated by a value in bits 0 through 15. The low-order 3 bits are encoded with the error severity level; all successful returns have an odd value:

0 = warning
1 = success
2 = error
3 = informational (nonstandard) success
4 = severe error
5-7 = reserved

Each numeric status code has a symbolic name in the form SS$_code. For example, the return might be SS$_NORMAL, which indicates successful completion of the system service. There are several error conditions that can be returned. For example, SS$_IVCHAN indicates that an invalid channel number was specified in an I/O request.

The VAX/VMS System Service Reference Manual describes the possible returns for each system service. Table 1-4 lists the valid status returns for the $QIO, $QIOW, $INPUT, and $OUTPUT system service requests.

Table 1-4
$QIO, $QIOW, $INPUT, and $OUTPUT System Services Status Returns

Status	Meaning
SS$_NORMAL	The $QIO, $QIOW, $INPUT, or $OUTPUT request was successfully completed; that is, an I/O request was placed in the appropriate device queue.
SS$ ACCVIO	The IOSB, the specified buffer, or the argument list cannot be accessed by the caller.
SS$_EXQUOTA	The buffer quota, buffered I/O quota, or direct I/O quota was exceeded and the process has disabled resource wait mode with the $SETRWM system service. (The $SETRWM system service is described in Section 1.4.) SS$_EXQUOTA is also set if the AST quota was exceeded.
SS$_ILLEFC	An illegal event flag number was specified.
SS$_INSFMEM	Insufficient dynamic memory is available to complete the service and the process has disabled resource wait mode with the $SETRWM system service. (The $SETRWM system service is described in Section 1.4.)

(Continued on next page)

Table 1-4 (Cont.)
$QIO, $QIOW, $INPUT, and $OUTPUT System Services Status Returns

Status	Meaning
SS$_IVCHAN	An invalid channel number was specified; that is, a channel number larger than the number of channels available.
SS$_NOPRIV	The specified channel was assigned from a more privileged access mode, the channel is not assigned, or the user does not have the proper privilege to access the device.
SS$_UNASEFC	A common event flag in an unassociated event flag cluster was specified.

Status returns for systems services are not the same as the I/O status returns described in Chapters 2 through 8 and 10 for the different I/O drivers (see Section 1.9). A system service status return is the status of the $QIO, $QIOW, $INPUT, $OUTPUT, or other system service call after completion of the service, that is, after the system returns control to the user. A system service status return does not reflect the completion (successful or unsuccessful) of the requested I/O operation. For example, a $QIO system service read request to a terminal might be successful (status return is SS$_NORMAL) but fail because of a device parity error (I/O status return is SS$_PARITY). System service error status return codes refer only to failures to invoke the service.

An I/O status return is the status at the completion of the I/O operation. It is returned in the quadword I/O status block (IOSB). Although some of the symbolic names (for example, SS$_NORMAL and SS$_ACCVIO) can be used in both types of status returns, they have different meanings.

1.9 I/O COMPLETION

Whether an I/O request completed successfully or unsuccessfully can be denoted by one or more return conditions. The selection of the return conditions depends on the arguments included in the QIO macro call. The three primary returns are:

- Event flag--an event flag is set on completion of an I/O operation.
- I/O status block--if the iosb argument was specified in the QIO macro call, a code identifying the type of success or failure is returned in bits 0 through 15 of a quadword I/O status block on completion of the I/O operation. The location of this block is indicated by the user-supplied iosb argument.
- Asynchronous system trap--if an AST address argument was specified in the I/O request, a call to the AST service routine occurs, at the address indicated, on completion of the I/O operation. (The I/O status block, if specified in the I/O request, is updated prior to the AST call.)

1.9.1 Event Flags

Event flags are status posting bits used by the $QIO, $QIOW, $INPUT, and $OUTPUT system services to indicate the completion or occurrence of an event. The system service clears the event flag when the operation is queued and sets it when the operation is completed. Event flag services allow users to set or clear certain flags, test the current status of flags, or place a program in a wait state pending the setting of a flag or group of flags.

See the VAX/VMS System Services Reference Manual for more information on event flags and their use.

1.9.2 I/O Status Block

The completion status of an I/O request is returned in the first word of the I/O status block (IOSB), as shown in Figure 1-7.

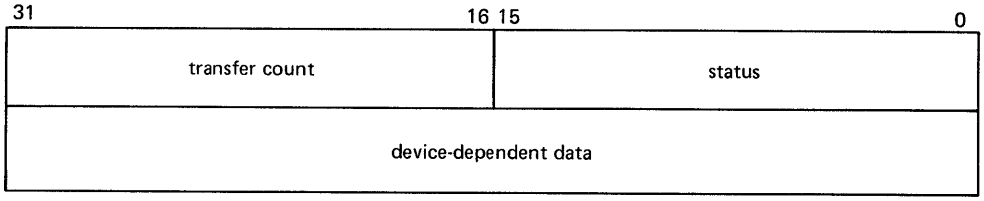


Figure 1-7 I/O Status Block Format

The IOSB indicates whether the operation was successfully completed, the amount of data transferred, and additional device-dependent information such as the number of lines printed. The status return code has the same format and bit significance (bit 0 set indicates success; bit 0 clear indicates error) as the system service status code (see Section 1.8.8). For example, if the process attempts to access a nonexistent disk, a status code of SS$_NONEXDRV is returned in the I/O status block. The status returns for the individual I/O drivers are listed in Chapters 2 through 8 and 10.

The upper half of the first IOSB longword contains the transfer count on completion of the I/O operation if the operation involved the transfer of data to or from a user buffer. For example, if a read operation is performed on a terminal, the number of bytes typed before a carriage return is indicated here. If a magnetic tape unit is the device and a read function is specified, the transfer count represents the number of bytes actually transferred. The second longword of the IOSB can contain certain device-dependent information. This information is supplied in more detail for each I/O driver in Chapters 2 through 8 and 10.

The status can be tested symbolically, by name. For example, the SS$_NORMAL status is returned if the operation was completed successfully. The following example illustrates the examination of the I/O status block XYIOSB to determine if an error occurred:

```
$QIO_S   CHAN=XYCHAN,FUNC=#IO$_WRITEVBLK,-
         IOSB=XYIOSB,P1=BUFADR,P2=#BUFLEN
BLBC     R0,REQERR                      ;CHECK SYSTEM SERVICE
                                        ;STATUS CODE
         .
         .
         .
CMPW     #SS$_NORMAL,XYIOSB             ;CHECK I/O STATUS
                                        ;CODE
BNEQ     ERROR
```

CHAPTER 2

TERMINAL DRIVER

This chapter describes the use of the VAX/VMS terminal driver. This driver supports the DZ-11 Asynchronous Serial Line Multiplexer and the console terminal.

2.1 SUPPORTED TERMINAL DEVICES

Each DZ-11 multiplexer interfaces 8 or 16 asynchronous serial communication lines for use with terminals. It supports programmable baud rates; however, input and output speeds must be the same. VAX/VMS supports the DZ-11 internal modem control.

The system console terminal is attached to the processor with a special purpose interface.

2.2 TERMINAL DRIVER FEATURES AND CAPABILITIES

The VAX/VMS terminal driver provides the following capabilities:

- Type-ahead
- Specifiable or default line terminators
- Special operating modes, such as NOECHO and PASSALL
- American National Standard escape sequence detection
- Terminal/mailbox interaction
- Terminal control characters and special keys
- Dial-up
- Optional parity specification

2.2.1 Type-ahead

Input (data received) from a VAX/VMS terminal is always independent of concurrent output (data sent) to a terminal. This capability is called type-ahead. Type-ahead is allowed on all terminals unless explicitly disabled by the Set Mode characteristic, inhibit type-ahead (TT$M_NOTYPEAHD; see Section 2.4.3).

Data typed at the terminal is retained in the type-ahead buffer until the user program issues an I/O request for a read operation. At that time, the data is transferred to the program buffer and echoed at the terminal where it was typed.

Deferring the echo until a read operation is active allows the user process to specify function code modifiers that modify the read operation. These modifiers can include, for example, noecho (IO$M_NOECHO) and convert lowercase characters to uppercase (IO$M_CVTLOW) (see Section 2.4.1.1).

If a read operation is already in progress when the data is typed at the terminal, the data transfer and echo are immediate.

The action of the driver when the type-ahead buffer fills depends on the Set Mode characteristic TT$M_HOSTSYNC (see Section 2.4.3). If TT$M_HOSTSYNC is not set, CTRL/G (BELL) is returned to inform the user that the type-ahead buffer is full. If TT$M_HOSTSYNC is set, the driver stops input by sending a CTRL/S and the terminal responds by sending no more characters. These warning operations are begun 8 characters before the type-ahead buffer fills. The driver sends a CTRL/Q to restart transmission when the type-ahead buffer empties completely.

The VAX/VMS System Manager's Guide describes the type-ahead buffer size.

2.2.2 Line Terminators

A line terminator is the control sequence that the user types at the terminal to indicate the end of an input line. Optionally, the user process can specify a particular line terminator or class of terminators for read operations.

Terminators are specified by an argument to the QIO request for a read operation. By default, they can be any ASCII control character except FF, VT, TAB, or BS. If included in the request, the argument is a user-selected group of characters (see Section 2.4.1.2).

All characters are 7-bit ASCII characters unless data is input on an 8-bit terminal (see Section 2.4.1). (The characteristic TT$M_EIGHTBIT determines whether the terminal uses the 7-bit or 8-bit character set; see Table 2-4.) All input characters are tested against the selected terminator(s). The input is terminated when a match occurs or the user's input buffer fills.

2.2.3 Special Operating Modes

The VAX/VMS terminal driver supports many special operating modes for terminal lines. Section 2.4.3 lists these modes. All special modes are enabled or disabled by the Set Mode QIO.

2.2.4 Escape Sequences

Escape sequences are strings of two or more characters, beginning with the escape character (decimal 27 or hexadecimal 1B), that indicate that control information follows. Many terminals send and respond to such escape sequences to request special character sets or to indicate the position of a cursor.

The Set Mode characteristic TT$M_ESCAPE (see Section 2.4.3) is used to specify that VAX/VMS terminal lines can generate valid escape sequences. If this characteristic is set, the terminal driver verifies the syntax of the escape sequences. The sequence is always considered a read function terminator and is returned in the read buffer, that is, a read buffer can contain other characters that are not part of an escape sequence, but an escape sequence always comprises the last characters in a buffer. The return information in the read buffer and I/O status block includes the position and size of the terminating escape sequence in the data record (see Section 2.5).

Any escape sequence received from the terminal is checked for correct syntax. If the syntax is not correct, SS$_BADESCAPE is returned as the status of the I/O. If the escape sequence does not fit in the user buffer, SS$_PARTESCAPE is returned. The remaining characters are transmitted on the next read. No syntax integrity is guaranteed across read operations. Escape sequences are never echoed. Valid escape sequences are any of the following forms (hexadecimal notation):

(ESC) <int>...<int> <fin>

where:

(ESC) is pressing the (ESC) key, a byte (character) of 1B

<int> is an "intermediate character" in the range of 20 to 2F. This range includes the character "space" and 15 punctuation marks. An escape sequence can contain any number of intermediate characters, or none.

<fin> is a "final character" in the range of 30 to 7E. This range includes uppercase and lowercase letters, numbers, and 13 punctuation marks.

There are four additional escape sequence forms:

(ESC) <;> <20-2F>...<30-7E>
(ESC) <?> <20-2F>...<30-7E>
(ESC) <O> <20-2F>...<40-7E>
(ESC) <Y> <20-7E>...<20-7E>

For example, when the IDENTIFY escape sequence, escape Z, is sent to a VT-55 terminal, the response from the terminal is (ESC) <C>. (Escape sequences are neither displayed nor echoed on the terminal.)

Section 2.2.6 describes control character functions during escape sequences.

2.2.5 Terminal/Mailbox Interaction

Mailboxes are virtual I/O devices used for communication between processes. The terminal driver can use a mailbox to communicate with a user process. Chapter 7 describes the mailbox driver.

A user program can use the $ASSIGN system service to associate a mailbox with one or more terminals. The terminal driver sends messages to this mailbox when terminal-related events occur that require the attention of the user image.

Mailboxes used in this way carry status messages, not terminal data, from the driver to the user program. For example, when data is received from a terminal for which no read request is outstanding (unsolicited data), a message is sent to the associated mailbox to indicate data availability. On receiving this message, the user program must read the channel assigned to the terminal to obtain the data. Messages are sent to mailboxes under the following conditions:

- Unsolicited data in the type-ahead buffer. The use of the associated mailbox can be enabled and disabled as a subfunction of the read and write QIO requests (see Sections 2.4.1 and 2.4.2). Thus, the user process can enter into a dialog with the terminal after an unsolicited data message arrives. Then, after the dialog is over, the user process can re-enable the unsolicited data message function on the last I/O exchange. The default for all terminals is enabled. Only one message is sent between read operations.

- Terminal hang-up. Hang-up occurs when a remote line loses the carrier signal; a message is sent to the mailbox. When hang-up occurs on lines that have the characteristic TT$M_REMOTE set, the line characteristics are returned to the system default characteristics (see the VAX/VMS System Generation Reference Manual).

Messages placed in the mailbox have the following content and format:

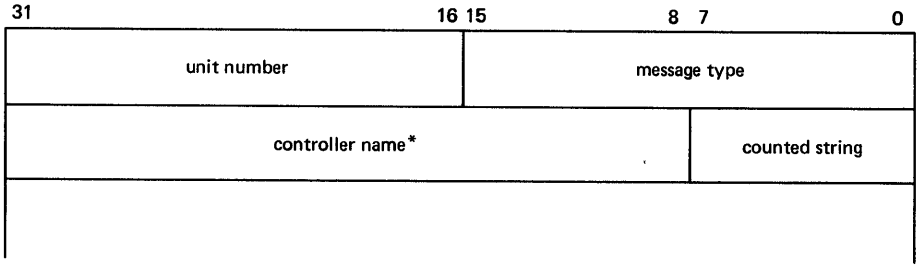
- Message type. The codes MSG$_TRMUNSOLIC (unsolicited data) and MSG$_TRMHANGUP (hang-up) identify the type of message. Message types are defined by the $MSGDEF macro.

- Device unit number to identify the terminal that sent the message.

- Counted string to specify the device name.

- Controller name

Figure 2-1 illustrates this format.



*does not include the colon (:) character

Figure 2-1 Terminal Mailbox Message Format

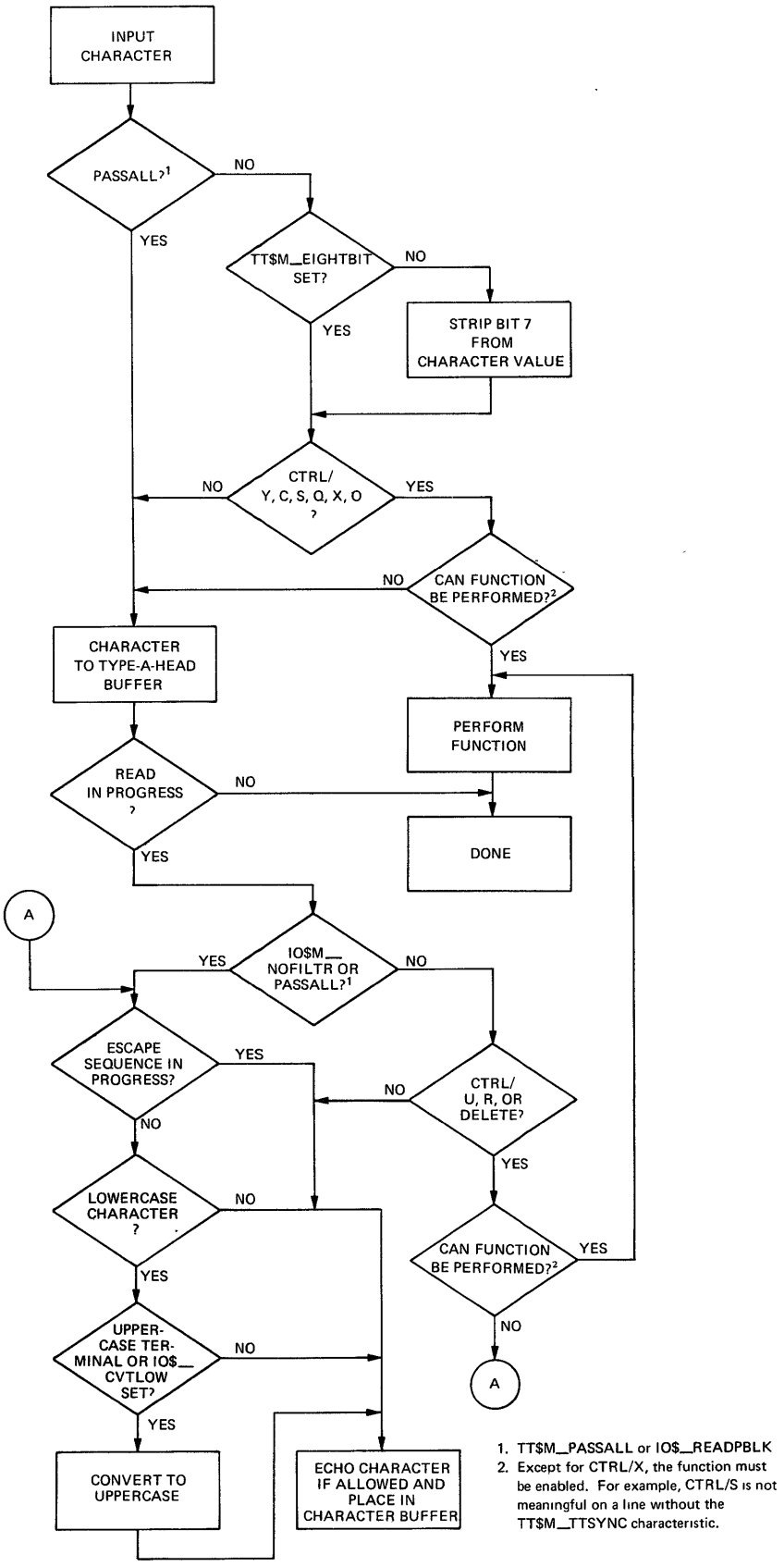


Figure 2-2 Character Interpretation

2.2.7 Dial-up

VAX/VMS supports the DZ-11 internal modem control (for example, Bell 103A, Bell 113, or equivalent) in autoanswer, full-duplex mode. The terminal driver does not support half-duplex operations on modems such as the Bell 202. The terminal characteristic TT$M_REMOTE designates the line as being remote to the local computer. The driver automatically sets TT$M_REMOTE if the carrier signal changes from off to on.

Dial-up lines are monitored periodically to detect a change in the modem carrier signal. The monitoring period is a system parameter. The VAX/VMS System Manager's Guide describes the dial-up monitoring period.

If a line's carrier signal is lost, the driver waits several monitor periods for the carrier signal to return. If the carrier signal is not detected during this time, the line is "hung-up." The hang-up action signals the owner of the line, through a mailbox message, that the line is no longer in use. (No dial-in message is sent; the unsolicited character message is sufficient when the first available data is received.) The line is not available for two monitor periods after the hang-up sequence begins. The hang-up sequence is not reversible. If the line hangs up, all enabled CTRL/Y ASTs are delivered; the CTRL/Y AST P2 argument is overwritten with SS$_HANGUP. The I/O operation in progress is cancelled and the status value SS$_ABORT is returned in the IOSB.

When a line with the TT$M_REMOTE characteristic is hung-up, the characteristics of the line are returned to the system default characteristics.

2.3 DEVICE INFORMATION

The user process can obtain terminal characteristics by using the $GETCHN and $GETDEV system services (see Section 1.10). The terminal-specific information is returned in the first three longwords of a user-specified buffer, as shown in Figure 2-3 (Figure 1-9 shows the entire buffer).

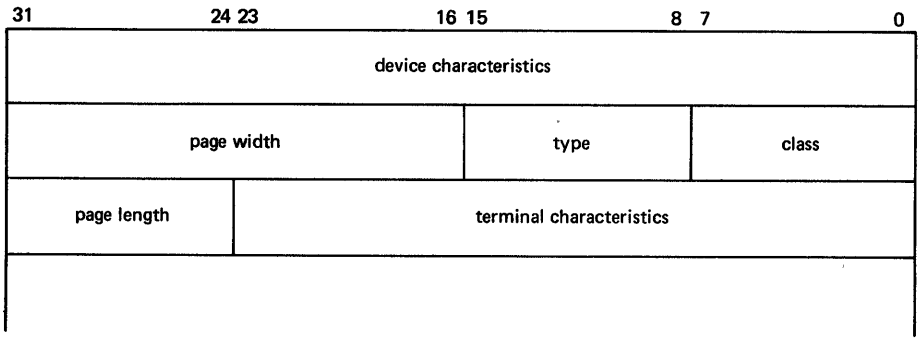


Figure 2-3 Terminal Information

The first longword contains device-independent data. The second and third longwords contain device-dependent data.

Table 2-3 lists the device-independent characteristics returned in the first longword.

In a read physical block operation, the data received from the associated terminal is placed in the user buffer as binary information without interpretation; the terminal line is in a temporary PASSALL mode. Since IO$_READPBLK is a physical I/O function, it can be specified only by a privileged user (see Section 1.6.1). IO$_READPBLK puts the terminal line in a PASSALL mode which is in effect only for the read physical block operation. This is in contrast with the more comprehensive PASSALL mode established by the Set Mode characteristic TT$M_PASSALL. All input and output data is in 8-bit binary format when TT$M_PASSALL is set (see Section 2.4.3).

Since IO$_READPBLK does not purge the type-ahead buffer (unless requested using the IO$M_PURGE function modifier) the characters in the type-ahead buffer may have been subjected to CTRL/Y/C/S/Q/O interpretation (Section 2.2.6.1). (Characters received while the IO$_READPBLK is in progress are not interpreted.)

2.4.1.1 **Function Modifier Codes for Read QIO Functions** - Seven function modifiers can be specified with IO$_READVBLK, IO$_READLBLK, IO$_READPROMPT, and IO$_READPBLK. Table 2-5 lists these function modifiers. IO$M_CVTLOW and IO$M_NOFILTR are not meaningful to IO$_READPBLK.

Table 2-5
Read QIO Function Modifiers

Code	Consequence
IO$M_NOECHO	Characters are not echoed (that is, displayed) as they are entered at the keyboard. The terminal line can also be set to a "no echo" mode by the Set Mode characteristic TT$M_NOECHO, which inhibits all read operation echoing.
IO$M_CVTLOW	Lowercase alphabetic characters (hexadecimal 61 to 7A) are converted to uppercase when transferred into the user buffer or echoed.
IO$M_NOFILTR	The terminal driver does not interpret CTRL/U, CTRL/R, or DEL. They are passed to the user.
IO$M_TIMED	The P3 argument specifies the maximum time (seconds) that can elapse between characters received; that is, the timeout value for the read operation. A value of 0 terminates the read operation, that is, an I/O timeout occurs, if no character is read within 1 second. In effect, data is read from the type-ahead buffer or an error is returned.
IO$M_PURGE	The type-ahead buffer is purged before the read operation begins.
IO$M_DSABLMBX	The mailbox is disabled for unsolicited data.
IO$M_TRMNOECHO	The termination character (if any) is not echoed. There is no formal terminator if the buffer is filled before the terminator is typed.

2.4.1.2 **Read Function Terminators** - The P4 argument to a read QIO function either specifies the terminator set for the read function or points to the location containing that terminator set. If P4 is 0, all ASCII characters with a code in the range 0 through 31 (hexadecimal 0 through 1F) except LF, VT, FF, TAB, and BS, are terminators. (This is the RMS-32 standard terminator set.)

If P4 does not equal 0, it contains the address of a quadword that either specifies a terminator character bit mask or points to a location containing that bit mask. The quadword has a short form and a long form, as shown in Figure 2-4. In the short form, the correspondence is between the bit number and the binary value of the character; the character is a terminator if the bit is set. For example, if bit 0 is set, NULL is a terminator; if bit 9 is set, TAB is a terminator. If a character is not specified, it is not a terminator. Since ASCII control characters are in the range of 0 through 31, the short form can be used in most cases.

The long form allows use of a more comprehensive set of terminator characters. Any mask size equal to or greater than 1 byte is acceptable. For example, a mask size of 16 bytes allows all 7-bit ASCII characters to be used as terminators; a mask size of 32 bytes allows all 8-bit characters to be used as terminators for 8-bit terminals. An unspecified mask is assumed to be all 0's.

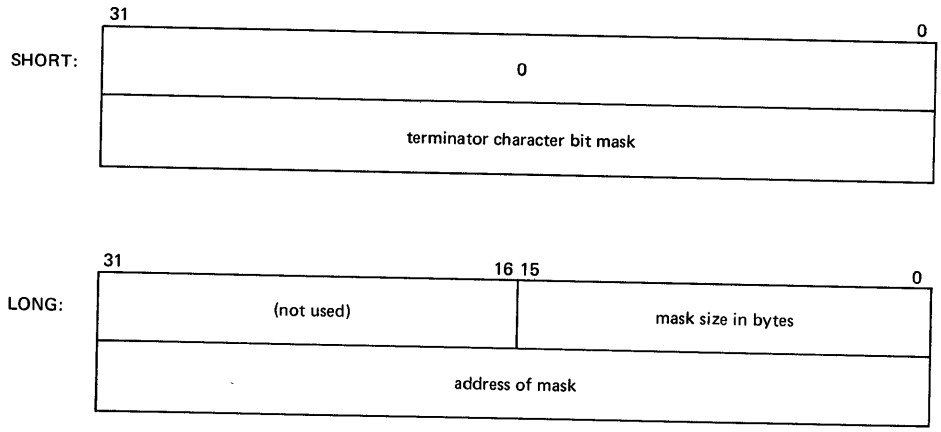


Figure 2-4 Short and Long Forms of Terminator Mask Quadwords

2.4.2 Write

Write operations display the contents of a user-specified buffer on the associated terminal. VAX/VMS defines three basic write I/O functions, which are listed with their function codes below:

- IO$_WRITEVBLK - write virtual block
- IO$_WRITELBLK - write logical block
- IO$_WRITEPBLK - write physical block

The write function codes can take the following device/function-dependent arguments:

- P1 = the starting virtual address of the buffer that is to be written to the terminal

CHAPTER 10

LABORATORY PERIPHERAL ACCELERATOR DRIVER

This chapter describes the use of the VAX/VMS Laboratory Peripheral Accelerator (LPA11-K) driver and the high level language procedure library that interfaces with the LPA11-K driver. The procedure library is implemented with callable assembly language routines that translate arguments into the format required by the LPA11-K driver and handle buffer chaining operations. Routines for microcode loading and device initialization are also described.

This chapter is written with the understanding that the reader has access to a copy of the LPA11-K Laboratory Peripheral Accelerator User's Guide.

10.1 SUPPORTED DEVICE

The LPA11-K is a peripheral device that controls analog to digital (A/D) and digital to analog (D/A) converters, digital I/O registers, and real-time clocks. It is connected to the VAX 11/780 through the UNIBUS Adapter (UBA).

The LPA11-K is a fast, flexible, and easy to use microprocessor subsystem that is designed for applications requiring concurrent data acquisition and data reduction at high rates. The LPA11-K allows aggregate analog input and output rates up to 150,000 samples per second. The maximum aggregate digital input and output rate is 15,000 samples per second.

Table 10-1 lists the useful minimum and maximum LPA11-K configurations supported by VAX/VMS.

10.1.1 LPA11-K Modes of Operation

The LPA11-K operates in two distinct modes: dedicated, and multirequest.

In dedicated mode, only one user, that is, one request, can be active at a time, and only analog I/O data transfers are supported. Up to two A/D converters can be controlled simultaneously. One D/A converter can be controlled at a time. Sampling is initiated either by an overflow of the real-time clock or by an externally supplied signal. Dedicated mode provides sampling rates of up to 150,000 samples per second.

Table 10-1
Minimum and Maximum Configurations per LPA11-K

Minimum	Maximum
1 - DD11-Cx or Dx Backplane	2 - DD11-Cx or Dx Backplanes
1 - KW11-K Real Time Clock	1 - KW11-K Real Time Clock
One of the following:	2 - AD11-K A/D Converters
AD11-K A/D Converter	2 - AM11-K Multiplexers for AD11-K Converters
AA11-K D/A Converter	1 - AA11-K D/A Converter
DR11-K Digital I/O Register	5 - DR11-K Digital I/O Registers

In multirequest mode, sampling from all the devices listed in Table 10-1 is supported. The LPA11-K operates like a multi-controller device; up to eight requests (from one to eight users) can be active simultaneously. The sampling rate for each user is a multiple of the common real-time clock rate. Independent rates can be maintained for each user. Both the sampling rate and the device type are specified as part of each data transfer request. Multirequest mode provides a maximum aggregate sampling rate of 15,000 samples per second.

10.1.2 Errors

The LPA11-K returns three classes of errors:

1. Errors associated with the issuance of a new LPA11-K command (SS$_DEVCMDERR).
2. Errors associated with an active data transfer request (SS$_DEVREQERR).
3. Fatal hardware errors which affect all LPA11-K activity (SS$_CTRLERR).

Appendix A of the LPA11-K Laboratory Peripheral Accelerator User's Guide lists these three classes of errors and the specific error codes for each class. The LPA11-K aborts all active requests if any of the following conditions occur:

- Power failure
- Device timeout
- Fatal error

Power failure is reported to any active users when power is recovered.

Device timeouts are monitored only when a new command is issued. For data transfers, the time between buffer full interrupts is not defined. Thus, no timeout errors are reported on a buffer to buffer basis.

If a required resource is not available to a process, an error message is returned immediately. The driver does not place the process in the resource wait mode.

10.2 SUPPORTING SOFTWARE

The LPA11-K is supported by a device driver, a high level language procedure library of support routines, and routines for microcode loading and device initialization. All data transfer algorithms for the laboratory data acquisition I/O devices are accomplished by the LPA11-K. The only purpose for the system software and support routines is to provide a control path for synchronizing the use of buffers, specifying requests, and starting and stopping requests.

The LPA11-K driver and the associated I/O interface have the following features:

- They permit multiple LPA11-K subsystems on a single UBA.
- They operate as an integral part of the VAX/VMS operating system.
- They can be loaded on an operating VAX/VMS system without relinking the executive.
- They handle I/O requests, function dispatching, UBA map allocation, interrupts, and error reporting for multiple LPA11-K subsystems.
- The LPA11-K functions as a multi-buffered device. Up to eight buffer areas can be defined per request. Up to eight requests can be handled simultaneously. Buffer areas can be reused after the data they contain is processed.
- Since the LPA11-K chains buffer areas automatically, a start data transfer request can transfer an infinite and continuous amount of data.
- Multiple ASTs are dynamically queued by the driver to indicate when a buffer has been filled (the data is available for processing), or emptied (the buffer is available for new data).

The high level language support routines have the following features:

- They translate arguments provided in the high level language calls into the format required for the Queue I/O interface.
- They provide a buffer chaining capability for a multibuffering environment by maintaining queues of used, in use, and available buffers.
- They adhere to all VAX/VMS conventions for calling sequences, use of sharable resources, and re-entrancy.
- They can be part of a resident global library, or be linked into a process image as needed.

The routines for microcode loading and device initialization have the following features:

- They execute, as separate processes, images which issue I/O requests. These I/O requests initiate microcode image loading, start the LPA11-K subsystem, and automatically configure the peripheral devices on the LPA11-K internal I/O bus.
- They can be executed by user or operator request.
- They can be executed at the request of other processes.
- They can be executed automatically when the system is initialized and on power recovery.

Figure 10-1 shows the relationship of the supporting software to the LPA11-K.

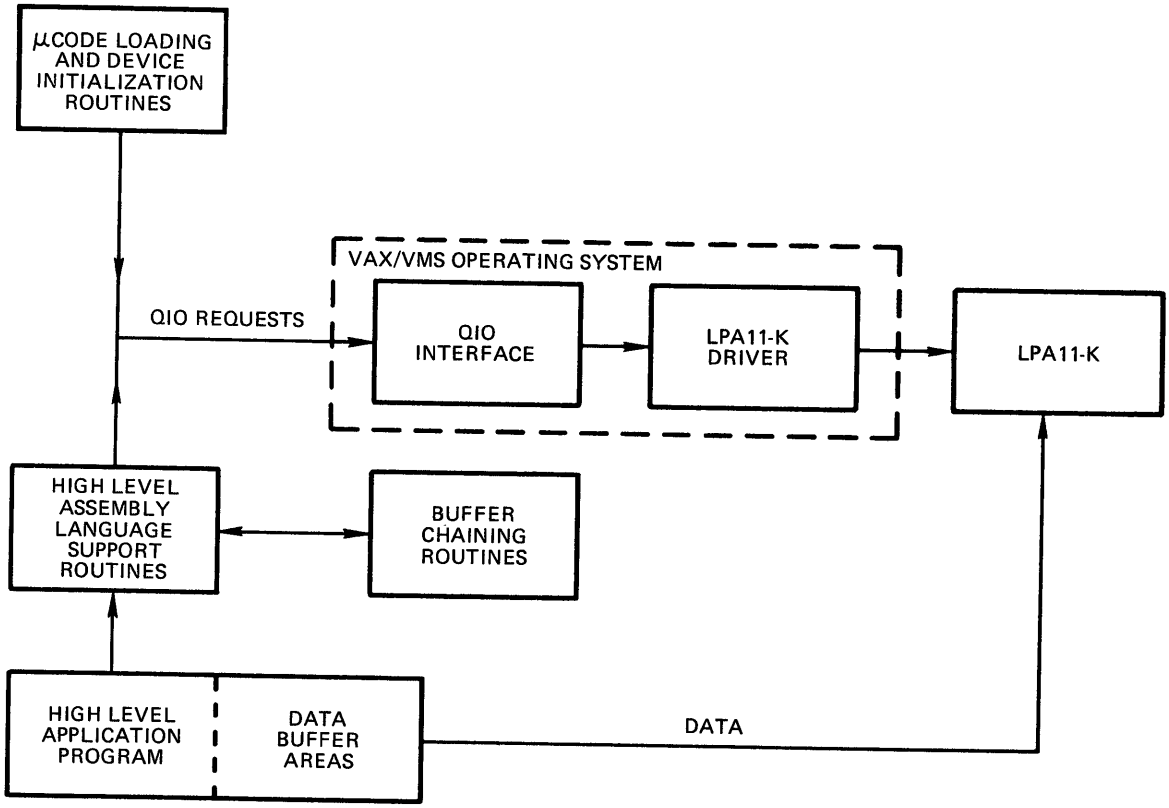


Figure 10-1 Relationship of Supporting Software to LPA11-K

10.3 DEVICE INFORMATION

Users can obtain information on all peripheral data acquisition devices on the LPA11-K internal I/O bus by using the $GETCHN and $GETDEV system services (see Section 1.10). The LPA11-K-specific information is returned in the first three longwords of a

user-specified buffer, as shown in Figure 10-2 (Figure 1-9 shows the entire buffer).

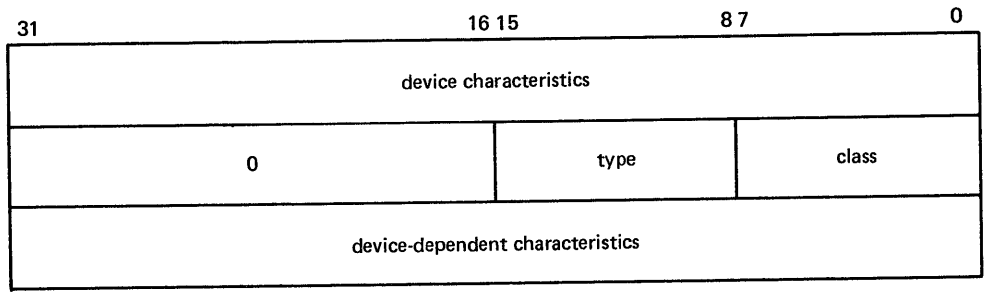


Figure 10-2 LPA11-K Information

The first longword contains device-independent information. The second and third longwords contain device-dependent data.

Table 10-2 lists the device-independent characteristics returned in the first longword.

Table 10-2
Device-independent Characteristics

Dynamic Bits[1] (Conditionally Set)	Meaning
DEV$M_AVL	Device is online and available
Static Bits[1] (Always Set)	
DEV$M_IDV	Input device
DEV$M_ODV	Output device
DEV$M_RTM	Real-time device
DEV$M_SHR	Device is sharable

[1]Defined by the $DEVDEF macro.

The second longword contains information on the device class and type. The device class for the LPA11-K is DC$_REALTIME and the device type is DT$_LPA11. The $LADEF macro defines these values. Buffer size is not applicable to the LPA11-K; this word is 0.

The third longword contains LPA11-K characteristics, that is, device-dependent data. LPA11-K characteristics are set by the set clock, initialize, and load microcode I/O functions to any one of, or a combination of, the values listed in Table 10-3.

Table 10-3
Device-Dependent Characteristics

Field[1]	Meaning
LA$M_MCVALID LA$S_MCVALID LA$V_MCVALID	The load microcode I/O function (IO$_LOADMCODE) was performed successfully. LA$M_MCVALID is set by IO$_LOADMCODE. Each microword is verified by reading it back and comparing it with the specified value. LA$M_MCVALID is cleared if there is no match.
LA$V_MCTYPE LA$S_MCTYPE	The microcode type, set by the load microcode I/O function (IO$_LOADMCODE), is one of the following values: LA$K_MRMCODE = microcode type is in multirequest mode LA$K_ADMCODE = microcode type is in dedicated A/D mode LA$K_DAMCODE = microcode type is in dedicated D/A mode
LA$V_CONFIG LA$S_CONFIG	The bit positions, set by the initialize I/O function (IO$_INITIALIZE), for the peripheral data acquisition devices on the LPA11-K internal I/O bus are one or more of the following: LA$V_CLOCKA = Clock A LA$M_CLOCKA LA$V_CLOCKB = Clock B LA$M_CLOCKB LA$V_AD1 = A/D device 1 LA$M_AD1 LA$V_AD2 = A/D device 2 LA$M_AD2 LA$V_DA = D/A device 1 LA$M_DA LA$V_DIO1 = Digital I/O Buffer 1 LA$M_DIO1 LA$V_DIO2 = Digital I/O Buffer 2 LA$M_DIO2 LA$V_DIO3 = Digital I/O Buffer 3 LA$M_DIO3 LA$V_DIO4 = Digital I/O Buffer 4 LA$M_DIO4 LA$V_DIO5 = Digital I/O Buffer 5 LA$M_DIO5

[1]Values defined by the $LADEF macro.

(continued on next page)

Table 10-3 (Cont.)
Device-Dependent Characteristics

Field[1]	Meaning
LA$V_RATE LA$S_RATE	The Clock A rate, set by the set clock function (IO$_SETCLOCK), is one of the following values: 0 = Stopped 1 = 1 MHz 2 = 100 kHz 3 = 10 kHz 4 = 1 kHz 5 = 100 Hz 6 = Schmidt trigger 7 = Line frequency
LA$V_PRESET LA$S_PRESET	The Clock A preset value set by the set clock function (IO$_SETCLOCK). (The value is in the range 0 through 65,535 - in two's complement form.) The clock rate divided by the clock preset value yields the clock overflow rate.

[1]Values defined by the $LADEF macro.

10.4 LPA11-K I/O FUNCTION CODES

The LPA11-K I/O functions are:

1. Load microcode into the LPA11-K.
2. Start the LPA11-K microprocessor.
3. Initialize the LPA11-K subsystem.
4. Set the LPA11-K real-time clock rate.
5. Start a data transfer request.

The Cancel I/O on Channel ($CANCEL) system service is used to abort data transfers.

10.4.1 Load Microcode

This I/O function resets the LPA11-K and loads an image of LPA11-K microcode. Physical I/O privilege is required. VAX/VMS defines a single function code:

IO$_LOADMCODE - load microcode

The load microcode function takes three device/function dependent arguments:

- P1 = the starting virtual address of the microcode image that is to be loaded into the LPA11-K
- P2 = the number of bytes (usually 2048) that are to be loaded
- P3 = the starting microprogram address (usually 0) in the LPA11-K that is to receive the microcode

If any data transfer requests are active at the time a load microcode request is issued, the load request is rejected and SS$_DEVACTIVE is returned in the I/O status block.

Each microword is verified by comparing it with the specified value in memory. If all words match, that is, the microcode was loaded successfully, the driver sets the microcode valid bit (LA$V_MCVALID) in the device-dependent characteristics longword (see Table 10-3). If there is no match, SS$_DATACHECK is returned in the I/O status block and LA$V_MCVALID is cleared to indicate that the microcode was not properly loaded. If the microcode was loaded successfully, the driver stores one of the microcode type values (LAK_MRCODE, LAK_ADCODE, or LA$K_DAMCODE) in the characteristics longword.

After a load microcode function is completed, the second word of the I/O status block contains the number of bytes loaded.

In addition to SS$_DATACHECK, IO$_LOADMCODE can return SS$_DEVACTIVE in the I/O status block.

10.4.2 Start Microprocessor

This I/O function resets the LPA11-K and starts (or restarts) the LPA11-K microprocessor. Physical I/O privilege is required. VAX/VMS defines a single function code:

- IO$_STARTMPROC - start microprocessor

This function code takes no device/function-dependent arguments.

The start microprocessor function can return five error codes in the I/O status block: SS$_DEVACTIVE, SS$_MCNOTVALID, SS$_CTRLERR, SS$_POWERFAIL, and SS$_TIMEOUT (see Section 10.6).

10.4.3 Initialize LPA11-K

This I/O function issues a subsystem initialize command to the LPA11-K. This command specifies LPA11-K laboratory I/O device addresses and other table information for the subsystem. It is issued only once after restarting the subsystem and before any other LPA11-K command is given. Physical I/O privilege is required. VAX/VMS defines a single function code:

- IO$_INITIALIZE - initialize LPA11-K

The initialize LPA11-K function takes two device/function-dependent arguments:

- P1 = the starting, word-aligned, virtual address of the Initialize Command Table in the user process. This table is read once by the LPA11-K during the execution of the initialize command. See the LPA11-K Laboratory Peripheral Accelerator User's Guide for additional information.

- P2 = length of the initialize command buffer (always 278 bytes)

If the initialize function is completed successfully, the appropriate device configuration values are set in the device-dependent characteristics longword (see Table 10-3).

The initialize function can return ten error codes in the I/O status block: SS$_IVMODE, SS$_INCLENGTH, SS$_BUFNOTALIGN, SS$_CTRLERR, SS$_DEVCMDERR, SS$_CANCEL, SS$_INSFMAPREG, SS$_MCNOTVALID, SS$_POWERFAIL, and SS$_TIMEOUT (see Section 10.6).

If a device specified in the Initialize Command Table is not in the LPA11-K configuration, an error condition (SS$_DEVCMDERR) occurs and the address of the first device not found is returned in the LPA11-K maintenance status register (see Section 10.6). A program can use this characteristic to poll the LPA11-K and determine the current device configuration.

10.4.4 Set Clock

This virtual function issues a clock control command to the LPA11-K. The clock control command specifies information necessary to start, stop, or change the sample rate at which the real-time clock runs on the LPA11-K subsystem.

If the LPA11-K has more than one user, caution should be exercised when the clock rate is changed. In multirequest mode, a change in the clock rate will affect all users.

VAX/VMS defines a single function code:

- IO$_SETCLOCK - set clock

The set clock function takes three device/function-dependent arguments:

- P2 = mode of operation. VAX/VMS defines the following clock start mode word (hexadecimal) values:

  1 = KW11-K Clock A
  11 = KW11-K Clock B

- P3 = clock control and status. VAX/VMS defines the following clock status word (hexadecimal) values:

  0 = stop clock
  143 = 1 MHz clock rate
  145 = 100 kHz clock rate
  147 = 10 kHz clock rate
  149 = 1 kHz clock rate
  14B = 100 Hz clock rate
  14D = clock rate is Schmidt trigger 1
  14F = clock rate is line frequency

- P4 = the two's complement of the real-time clock preset value. The range is 16 bits for the KW11-K Clock A and 8 bits for the KW11-K Clock B.

The LPA11-K Laboratory Peripheral Accelerator User's Guide describes the clock start mode word and the clock status word in greater detail.

If the set clock function is completed successfully for Clock A, the clock rate and preset values are stored in the device-dependent characteristics longword (see Table 10-3).

The set clock function can return six error codes in the I/O status block: SS$_CTRLERR, SS$_DEVCMDERR, SS$_CANCEL, SS$_MCNOTVALID, SS$_POWERFAIL, and SS$_TIMEOUT (see Section 10.6).

10.4.5 Start Data Transfer Request

This virtual I/O function issues a Data Transfer Start command that specifies the buffer addresses, sample mode, and sample parameters used by the LPA11-K. This information is passed to the Data Transfer Command Table. VAX/VMS defines a single function code:

- IO$_STARTDATA - start data transfer request

The start data transfer request function takes one function modifier:

- IO$M_SETEVF - set event flag

The start data transfer request function takes four device/function-dependent arguments:

- P1 = the starting virtual address of the Data Transfer Command Table in the user's process
- P2 = the length in bytes (always 40) of the Data Transfer Command Table
- P3 = the AST address of the normal buffer completion AST routine (optional)
- P4 = the AST address of the buffer overrun completion AST routine (optional). Only used when the buffer overrun bit (LA$M_BFROVRN) is set, that is, a buffer overrun condition is classified as a non-fatal error.

A buffer overrun condition is not the same as a data overrun condition. The LPA11-K fetches data from, or stores data in, memory. If data cannot be fetched quickly enough, for example, when there is too much UNIBUS activity, a data underrun condition occurs. If data cannot be stored quickly enough, a data overrun condition occurs. After each buffer has been filled or emptied, the LPA11-K obtains the index number of the next buffer to process from the User Status Word (USW). (See Section 2.5 of the LPA11-K Laboratory Peripheral Accelerator User's Guide). A buffer overrun condition occurs if the LPA11-K fills or empties buffers faster than the application program can supply new buffers. For example, buffer overrun can occur when the sampling rate is too high, the buffers are too small, or the system load is too heavy.

The LPA11-K driver accesses the ten-longword Data Transfer Command Table, shown in Figure 10-3, when the Data Transfer Start command is processed. After the command is accepted and data transfers have begun, the driver makes no further access to the table.

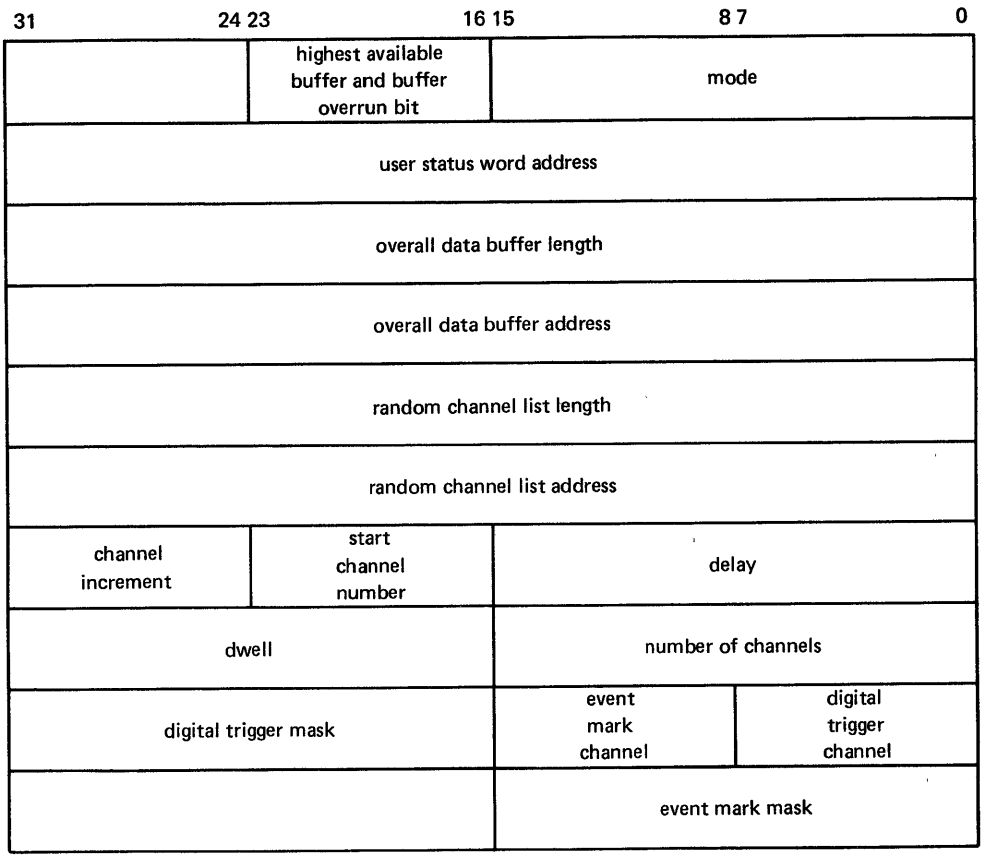


Figure 10-3 Data Transfer Command Table

In the first longword of the Data Transfer Command Table, the first two bytes contain the LPA11-K start data transfer request mode word. (Section 3.4.1 of the LPA11-K Laboratory Peripheral Accelerator User's Guide describes the functions of this word.)

The third byte contains the number (0-7) of the highest buffer available and the buffer overrun flag bit (bit 23; values: LA$M_BFROVRN and LA$V_BFROVRN). If this bit is set, a buffer overrun condition is a non-fatal error.

The second longword contains the User Status Word address (see Section 3.4.3 of the LPA11-K Laboratory Peripheral Accelerator User's Guide). This virtual address points to a two-byte area in the user process space, and must be word-aligned.

The third longword contains the size (in bytes) of the overall buffer area. The virtual address in the fourth longword is the beginning address of this area. This address must be longword-aligned. The overall buffer area contains a specified number of buffers (the number of the highest available buffer specified in the first longword plus one). Individual buffers are subject to length restrictions: in multirequest mode the length must be in multiples of two bytes; in dedicated mode the length must be in multiples of four bytes. All data buffers are virtually contiguous for each data transfer request.

The fifth and sixth longwords contain the Random Channel List (RCL) length and address, respectively. The RCL address must be word-aligned. The last word in the RCL must have bit 15 set. (See Section 3.4.6 of the LPA11-K Laboratory Peripheral Accelerator User's Guide for additional information on the RCL.)

The seventh through tenth longwords contain LPA11-K-specific sample parameters. The driver passes these parameters directly to the LPA11-K. (See Sections 3.4.7 through 3.4.12 of the LPA11-K Laboratory Peripheral Accelerator User's Guide for a detailed description of their functions.)

The start data transfer request function can return 15 error codes in the I/O status block: SS$_INCLENGTH, SS$_BUFNOTALIGN, SS$_DEVCMDERR, SS$_CTRLERR, SS$_DEVREQERR, SS$_ABORT, SS$_CANCEL, SS$_EXQUOTA, SS$_INSFBUFDP, SS$_INSFMAPREG, SS$_INSFMEM, SS$_MCNOTVALID, SS$_PARITY, SS$_POWERFAIL, and SS$_TIMEOUT (see Section 10.6).

Data buffers are chained and reused as the LPA11-K and the user process dispose of the data. As each buffer is filled or emptied, the LPA11-K driver notifies the application process by either setting the event flag specified by the QIO request efn argument or queueing an AST. Since buffer use is a continuing process, the event flag is set or the AST is queued a number of times. The user process must clear the event flag (or receive the AST), process the data, and specify the next buffer for the LPA11-K to use.

If the set event flag function modifier (IO$M_SETEVF) is specified, the event flag is set repeatedly: when the data transfer request is started, on each buffer completion, and when the request completes. If IO$M_SETEVF is not specified, the event flag is set only when the request completes.

ASTs are preferred over event flags for synchronizing a program with the LPA11-K because AST delivery is a queued process while setting of event flags is not. If only event flags are used, it is possible to lose buffer status.

Three AST addresses can be specified. For normal data buffer transactions the AST address specified in the P3 argument is used. If the buffer overrun bit in the Data Transfer Command Table is set and an overrun condition occurs, the AST address specified in the P4 argument is used. The AST address specified in the astadr argument of the QIO request is used when the entire data transfer request is completed. The astprm argument specified in the QIO request is passed to all three AST routines.

If insufficient dynamic memory is available to allocate an AST block, an error (SS$_INSFMEM) is returned. If the user does not have sufficient AST quota remaining to allocate an AST block, an error (SS$_EXQUOTA) is returned. In either case, the request is stopped. Normally, there are never more than three outstanding ASTs per LPA11-K request.

10.4.6 LPA11-K Data Transfer Stop Command

The Cancel I/O on Channel ($CANCEL) system service is used to abort data transfers for a particular process. When the LPA11-K driver receives a $CANCEL request, a Data Transfer Stop command is issued to the LPA11-K.

The normal way to stop a data transfer is to set bit 14 of the User Status Word. If this bit is set, the transfer stops at the end of the next buffer transaction (see Section 2.5 of the LPA11-K Laboratory Peripheral Accelerator User's Guide).

10.5 HIGH LEVEL LANGUAGE INTERFACE

VAX/VMS supports several program-callable procedures that provide access to the LPA11-K. The formats of these calls are documented here for VAX-11 FORTRAN IV-PLUS users. VAX-11 MACRO users must set up a standard VAX/VMS argument block and issue the standard procedure CALL. (Optionally, VAX-11 MACRO users can access the LPA11-K directly through the use of the device-specific Queue I/O functions described in Section 10.4.) Users of other high level languages must specify the proper subroutine or procedure invocation. If the subroutine or procedure call is not the standard linkage, some languages may require a MACRO interface.

10.5.1 High Level Language Support Routines

VAX/VMS provides 20 high level language procedures for the LPA11-K. These procedures are divided into four classes. Table 10-4 lists the VAX-11 procedures for the LPA11-K.

Table 10-4
VAX-11 Procedures for the LPA11-K

Class	Subroutine	Function
Sweep Control	LPA$ADSWP	Start A/D converter sweep
	LPA$DASWP	Start D/A converter sweep
	LPA$DISWP	Start digital input sweep
	LPA$DOSWP	Start digital output sweep
	LPA$LAMSKS	Specify LPA11-K controller and digital mask words
	LPA$SETADC	Specify channel select parameters
	LPA$SETIBF	Specify buffer parameters
	LPA$STPSWP	Stop sweep
Clock control	LPA$CLOCKA	Set Clock A rate
	LPA$CLOCKB	Set Clock B rate
	LPA$XRATE	Compute clock rate and preset value
Data Buffer Control	LPA$IBFSTS	Return buffer status
	LPA$IGTBUF	Return next available buffer
	LPA$INXTBF	Alter buffer order
	LPA$IWTBUF	Return next buffer or wait
	LPA$RLSBUF	Release buffer to LPA11-K
	LPA$RMVBUF	Remove buffer from device queue
Miscellaneous	LPA$CVADF	Convert A/D input to floating point
	LPA$FLT16	Convert unsigned integer to floating point
	LPA$LOADMC	Load microcode and initialize LPA11-K

10.5.1.1 **Buffer Queue Control** - This section is provided for informational purposes only. Normally, the user does not need to be concerned with the details of buffer queues.

Buffer queue control for data transfers by LPA11-K subroutines involves the use of three queues:

- Device queue (DVQ)
- User queue (USQ)
- In-use queue (IUQ)

Each data transfer request can specify from one to eight data buffer areas. The user specifies these buffers by address. During execution of the request, the LPA11-K assigns an index from 0 to 7 when a buffer is referenced.

The DVQ contains the indices of all the buffers that the user has released, that is, made available to be filled or emptied by the LPA11-K. For output functions (D/A and digital output), these buffers contain data to be output by the LPA11-K. For input functions (A/D and digital input), these buffers are empty and waiting to be filled by the LPA11-K.

The USQ contains the indices of all buffers that are waiting to be returned to the user. The LPA$IWTBUF and LPA$IGTBUF calls are used to return the index of the next buffer in the USQ. For output functions (D/A and digital output), these buffers are empty and waiting to be filled by the application program. For input functions (A/D and digital input), these buffers contain data to be processed by the application program.

The IUQ contains the indices of all buffers that are currently being processed by the LPA11-K. Normally, the IUQ contains the indices of two buffers:

- The buffer currently being filled or emptied by the LPA11-K
- The next buffer to be filled or emptied by the LPA11-K. This is the buffer specified by the next buffer index field in the User Status Word.

Because the LPA11-K driver requires that at least one buffer be ready when the input or output sweep is started, the user must call LPA$RLSBUF before the sweep is initiated.

Figure 10-4 shows the flow between the buffer queues.

10.5.1.2 **Subroutine Argument Usage** - Table 10-5 describes the general use of the subroutine arguments. The subroutine descriptions in the following sections contain additional information on argument usage. The IBUF, BUF, and ICHN (Random Channel List address) arguments must be aligned on specific boundaries. (The VAX-11 FORTRAN-IV-PLUS User's Guide describes the alignment of FORTRAN arguments.)

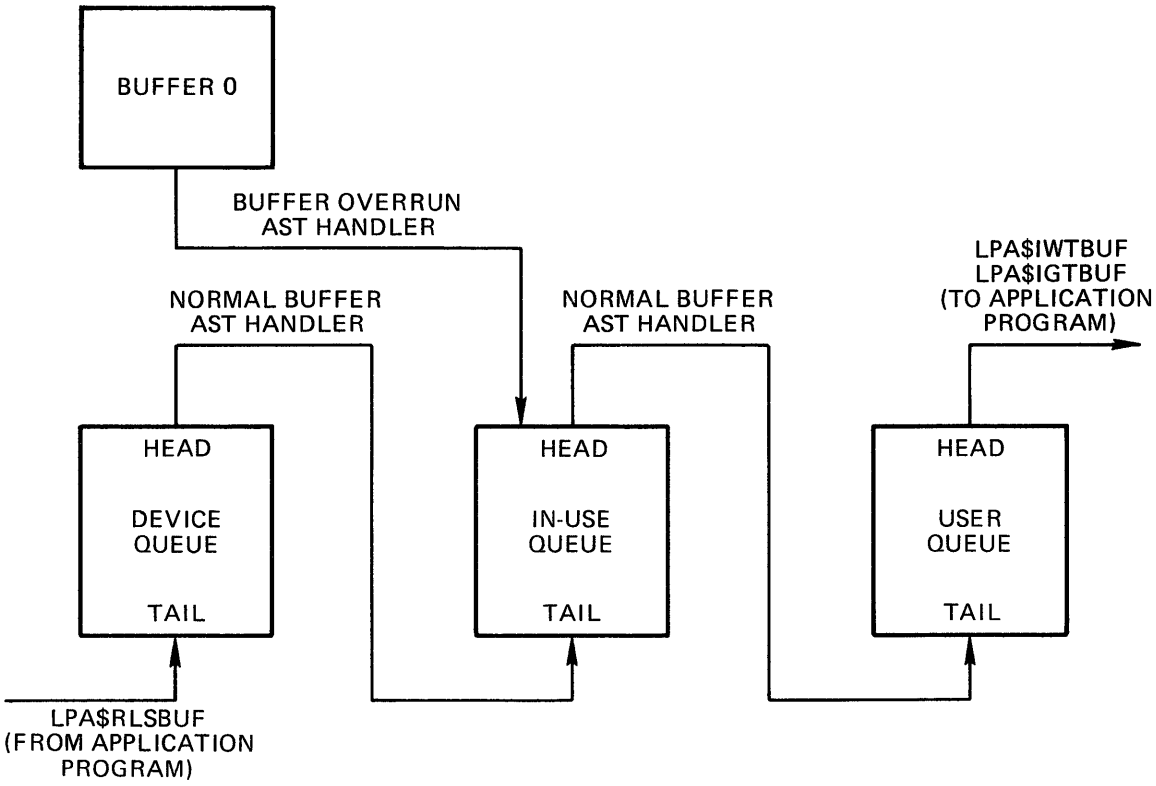


Figure 10-4 Buffer Queue Control

Table 10-5
Subroutine Argument Usage

Argument	Meaning
IBUF	A 50-longword array initialized by the LPA$SETIBF subroutine. IBUF is the impure area used by the buffer management subroutines. A unique IBUF array is required for each simultaneously active request. IBUF must be longword-aligned. The first quadword in the IBUF array is an I/O status block (IOSB) for high level language subroutines. The LPA$IGTBUF and LPA$IWTBUF subroutines fill this quadword with the current and completion status (see Section 10.6).

(continued on next page)

Table 10-5 (Cont.)
Subroutine Argument Usage

Argument	Meaning
LBUF	Specifies the size of each data buffer in words (must be even for dedicated mode sweeps). All buffers are the same size. The minimum value for LBUF is 1 for multirequest mode data transfers and 258 for dedicated mode data transfers. The aggregate size of the assigned buffers must be less that 32,768 words. Thus, the maximum size of each buffer (in words) is limited to 32,768 divided by the number of buffers. The LBUF argument length is one word.
NBUF	Specifies the number of times the buffers are to be filled during the life of the request. If 0 (default) is specified, sampling is indefinite and must be stopped with the LPA$STPSWP subroutine. The NBUF argument length is one longword.
MODE	Specifies sampling options. MODE bit values are listed in the appropriate subroutine descriptions. The default is 0. MODE values can be added to specify several options. No options are mutually exclusive although not all bits may be applicable at the same time. The MODE argument length is one word.
IRATE	Specifies the clock rate: 0 = Clock B overflow or no rate 1 = 1 MHz 2 = 100 kHz 3 = 10 kHz 4 = 1 kHz 5 = 100 Hz 6 = Schmidt trigger 7 = Line frequency The IRATE argument length is one longword.
IPRSET	Specifies the hardware clock preset value. This value is the two's complement of the desired number of clock ticks between clock interrupts. (The maximum value is the two's complement of 65,536.) IPRSET can be computed by the LPA$XRATE subroutine. The IPRSET argument length is one word.
DWELL	Specifies the number of hardware clock overflows between sample sequences in multirequest mode. For example, if DWELL is 20 and NCHN is 3, then after 20 clock overflows one channel is sampled on each of the next three successive overflows; no sampling occurs for the next 20 clock overflows. This allows different users to use different sample rates with the same hardware clock overflow rate. In dedicated mode, the hardware clock overflow rate controls sampling and DWELL is not accessed. Default for DWELL is 1. The DWELL argument length is one word.

(continued on next page)

Table 10-5 (Cont.)
Subroutine Argument Usage

Argument	Meaning
IEFN	Specifies the event flag number or completion routine address. The selected event flag is set at the end of each buffer transaction. If IEFN is 0 (default), event flag 22 is used. IEFN can also specify the address of a completion routine. This routine is called by the buffer management routine when a buffer is available and when the request is terminated, either successfully or with an error. The standard VAX/VMS calling and return sequences are used. The completion routine is called from an AST routine and is therefore at AST level. If IEFN specifies the address of a completion routine, the program must call LPA$IGTBUF to obtain the next buffer. If IEFN specifies an event flag, the program must call LPA$IWTBUF to obtain the next buffer and must use the %VAL operator: ,%VAL(3), (Event flag 3) ,BFRFULL, (Address of completion routine) The IEFN argument length is one longword. If multiple sweeps are initiated, they must use different event flags (the software does not enforce this policy). Event flag 23 is reserved for use by the LPA$CLOCKA and LPA$CLOCKB subroutines. If either of these subroutines is included in the user program, event flag 23 cannot be used. Also, if IEFN is defaulted, event flag 22 cannot be used in the user program.
LDELAY	Specifies the delay, in IRATE units, from the start event until the first sample is taken. The maximum value is 65,536; default is 1. The LDELAY argument length is one word. The LPA11-K supports the LDELAY argument in multirequest mode only.
ICHN	Specifies the number of the first I/O channel to be sampled. Default is channel 0. The ICHN argument length is one byte. The channel number is not the same as the channel assigned to the device by the $ASSIGN system service (see Section 1.8.1). The LPA11-K uses the channel number to specify the multiplexer address of an A/D, D/A, or digital I/O device on the LPA11-K internal I/O bus.

(continued on next page)

Table 10-5 (Cont.)
Subroutine Argument Usage

Argument	Meaning
NCHN	Specifies the number of I/O device channels to sample in a sample sequence. Default is 1. If the NCHN argument is 1, the single channel bit is set in the mode word of the start Request Descriptor Array (RDA) when the sweep is started. The RDA contains the information needed by the LPA11-K for each command (see the LPA11-K Laboratory Peripheral Accelerator User's Guide). The NCHN argument length is one word.
IND	Receives the VAX/VMS success or failure code of the call. The IND argument length is one longword.

10.5.2 LPA$ADSWP - Initiate Synchronous A/D Sampling Sweep

The LPA$ADSWP subroutine initiates A/D sampling through an AD11-K.

The format of the LPA$ADSWP call is as follows:

```
CALL LPA$ADSWP (IBUF,LBUF,[NBUF],[MODE],[DWELL],[IEFN],[LDELAY],
                [ICHN],[NCHN],[IND])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

MODE Specifies sampling options. VAX/VMS defines the following sampling option values:

Value	Meaning
32	Parallel A/D conversion sample algorithm is used if dual A/D converters are specified (value = 8192). Absence of this bit implies the serial A/D conversion sample algorithm.
64	Multirequest mode request. Absence of this bit implies a dedicated mode request.
512	External trigger (Schmidt trigger 1). Dedicated mode only. (The LPA11-K Laboratory Peripheral Accelerator User's Guide describes the use of an external trigger.)
1024	Time stamped sampling with Clock B. The double word consists of one data word followed by the value of the LPA11-K's internal 16-bit counter at the time of the sample (see Section 2.4.3 in the LPA11-K Laboratory Peripheral Accelerator User's Guide). Multirequest mode only.

Value	Meaning
2048	Event marking. Multirequest mode only. (The LPA11-K Laboratory Peripheral Accelerator User's Guide describes event marking.)
4096	Start method. If selected, digital input start. If not selected, immediate start. Multirequest mode only.
8192	Dual A/D converters are to be used. Dedicated mode only.
16384	Buffer overrun is a non-fatal error. The LPA11-K will automatically default to fill buffer 0 if a buffer overrun condition occurs.

If MODE is defaulted, A/D sampling starts immediately with absolute channel addressing in dedicated mode. The LPA11-K does not support delays in dedicated mode.

IND Returns the success or failure status:

0 = Error in call. Possible causes are: LPA$SETIBF was not previously called; LPA$RLSBUF was not previously called; size of data buffers disagrees with the size computed by the LPA$SETIBF call.

1 = successful sweep started

nnn = VAX/VMS status code

10.5.3 LPA$DASWP - Initiate Synchronous D/A Sweep

The LPA$DASWP subroutine initiates D/A output to an AA11-K.

The format for the LPA$DASWP call is as follows:

```
CALL LPA$DASWP (IBUF,LBUF,[NBUF],[MODE],[DWELL],[IEFN],[LDELAY],
     [ICHN],[NCHN],[IND])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

MODE Specifies the sampling options. VAX/VMS defines the following start criteria values:

Value	Meaning
0	Immediate start. This is the default value for MODE.
64	Multirequest mode. If not selected, this request is for dedicated mode.
4096	Start method. If selected, digital input start. If not selected, immediate start. Multirequest mode only.

Value	Meaning
16384	Buffer overrun is a non-fatal error. The LPA11-K will automatically default to empty buffer 0 if a buffer overrun condition occurs.

IND Returns the success or failure status:

0 = Error in call. Possible causes are: LPA$SETIBF was not previously called; LPA$RLSBUF was not previously called; size of data buffers disagrees with the size computed by the LPA$SETIBF call.

1 = successful sweep started

nnn = VAX/VMS status code

10.5.4 LPA$DISWP - Initiate Synchronous Digital Input Sweep

The LPA$DISWP subroutine initiates digital input through a DR11-K. LPA$DISWP is applicable in multirequest mode only.

The format of the LPA$DISWP call is as follows:

```
CALL LPA$DISWP (IBUF,LBUF,[NBUF],[MODE],[DWELL],[IEFN],[LDELAY],
          [ICHN],[NCHN],[IND])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

MODE Specifies sampling options. VAX/VMS defines the following sampling option values:

Value	Meaning
0	Immediate start. This is the default value for MODE.
512	External trigger for DR11-K. (The LPA11-K Laboratory Peripheral Accelerator User's Guide describes the use of an external trigger.)
1024	Time stamped sampling with Clock B. The double word consists of one data word followed by the value of the internal LPA11-K, 16-bit, counter at the time of the sample (see Section 2.4.3 in the LPA11-K Laboratory Peripheral Accelerator User's Guide).
2048	Event marking. (The LPA11-K Laboratory Peripheral Accelerator User's Guide describes event marking.)
4096	Start method. If selected, digital input start. If not selected, immediate start.
16384	Buffer overrun is a non-fatal error. The LPA11-K will automatically default to fill buffer 0 if a buffer overrun condition occurs.

IND Returns the success or failure status:

0 = Error in call. Possible causes are: LPA$SETIBF was not previously called; LPA$RLSBUF was not previously called; size of data buffers disagrees with the size computed by the LPA$SETIBF call.

1 = successful sweep started

nnn = VAX/VMS status code

10.5.5 LPA$DOSWP - Initiate Synchronous Digital Output Sweep

The LPA$DOSWP subroutine initiates digital output through a DR11-K. LPA$DOSWP is applicable in multirequest mode only.

The format of the LPA$DOSWP call is as follows:

```
CALL LPA$DOSWP (IBUF,LBUF,[NBUF],[MODE],[DWELL],[IEFN],[LDELAY],
          [ICHN],[NCHN],[IND])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

MODE Specifies the sampling options. VAX/VMS defines the following values:

Value	Meaning
0	Immediate start. This is the default value for MODE.
512	External trigger for DR11-K (The LPA11-K Laboratory Peripheral Accelerator User's Guide describes the use of an external trigger.)
4096	Start method. If selected, digital input start. If not selected, immediate start.
16384	Buffer overrun is a non-fatal error. The LPA11-K will automatically default to empty buffer 0 if a buffer overrun condition occurs.

IND Returns the success or failure status:

0 = Error in call. Possible causes are: LPA$SETIBF was not previously called; LPA$RLSBUF was not previously called; size of data buffers disagrees with the size computed by the LPA$SETIBF call.

1 = successful sweep started

nnn = VAX/VMS status code

10.5.6 LPA$LAMSKS - Set LPA11-K Masks and NUM Buffer

The LPA$LAMSKS subroutine initializes a user buffer which contains a number to append to the logical name LPA11$, a digital start word mask, an event mark mask, and channel numbers for the two masks.

LPA$LAMSKS must be called:

- By users who intend to use digital input starting or event marking
- By users who do not want to use the default of LAA0 assigned to LPA11$0
- If multiple LPA11-Ks are used

The format of the LPA$LAMSKS call is as follows:

```
CALL LPA$LAMSKS (LAMSKB,[NUM],[IUNIT],[IDSC],[IEMC],[IDSW],
          [IEMW],[IND])
```

Argument descriptions are as follows:

LAMSKB	Specifies a 4-word array
NUM	Specifies the number appended to LPA11$. The sweep is started on the LPA11-K assigned to LPA11$num.
IUNIT	Not used. This argument is present for compatibility only.
IDSC	Specifies the digital START word channel. Range is 0 to 4. The IDSC argument length is one byte.
IEMC	Specifies the event MARK word channel. Range is 0 to 4. The IEMC argument length is one byte.
IDSW	Specifies the digital START word mask. The IDSW argument length is one word.
IEMW	Specifies the event MARK word mask. The IEMW argument length is one word.
IND	Always equal to 1 (success). This argument is present for compatibility only.

10.5.7 LPA$SETADC - Set Channel Information For Sweeps

The LPA$SETADC subroutine establishes channel start and increment information for the sweep control subroutines (see Table 10-4). The LPA$SETIBF subroutine must be called to initialize IBUF before LPA$SETADC is called.

The two formats for the LPA$SETADC call are as follows:

```
CALL LPA$SETADC (IBUF,[IFLAG],[ICHN],[NCHN],[INC],[IND])
```

or,

```
IND=LPA$SETADC (IBUF,[IFLAG],[ICHN],[NCHN],[INC])
```

Argument descriptions are as follows:

IND	Returns the success or failure status: 0 = LPA$SETIBF was not called prior to the LPA$SETADC call 1 = LPA$SETADC call successful
IBUF	The IBUF array specified in the LPA$SETIBF call
IFLAG	Reserved. This argument is present for compatibility only.
ICHN	Specifies the first channel number. Range is 0 to 255; default is 0. The ICHN argument length is one longword. If INC = 0, ICHN is the address of a Random Channel List. This address must be word-aligned.
NCHN	Specifies the number of samples taken per sample sequence. Default is 1.
INC	Specifies the channel increment. Default is 1. If INC is 0, ICHN is the address of a Random Channel List. The INC argument length is one longword.

10.5.8 LPA$SETIBF - Set IBUF Array For Sweeps

The LPA$SETIBF subroutine initializes the IBUF array for use with the LPA$ADSWP, LPA$DISWP, LPA$DOSWP, LPA$DASWP, LPA$STPSWP, LPA$IWTBUF, LPA$IGTBUF, LPA$IBFSTS, LPA$RLSBUF, LPA$INXTBF, LPA$SETADC, and LPA$RMVBUF subroutines.

The format of the LPA$SETIBF call is as follows:

```
CALL LPA$SETIBF (IBUF,[IND],[LAMSKB],BUF0,[BUF1,...,BUF7])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

IBUF	Specifies a 50-longword array that is initialized by this subroutine. IBUF must be longword-aligned. (See Table 10-5 for additional information on IBUF.)
IND	Returns the success or failure status: 0 = Error in call. Possible causes are: incorrect number of arguments; IBUF array not longword-aligned; buffer addresses not equidistant. 1 = IBUF initialized successfully

LAMSKB — Specifies the name of a 4-word array. This array allows the use of multiple LPA11-Ks within the same program because the argument used to start the sweep is specified by the LPA$LAMSKS call. (See Section 10.5.6 for a description of the LPA$LAMSKS subroutine.)

BUF0, etc. — Specify the names of the buffers. A maximum of eight buffers can be specified. At least two buffers must be specified to provide continuous sampling. The LPA11-K driver requires that all buffers be contiguous. To ensure this, LPA$SETIBF verifies that all buffer addresses are equidistant. Buffers must be longword-aligned.

10.5.9 LPA$STPSWP - Stop In-progress Sweep

The LPA$STPSWP subroutine allows a user to stop a sweep that is in progress.

The format of the LPA$STPSWP call is as follows:

```
CALL LPA$STPSWP (IBUF,[IWHEN],[IND])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

IBUF — THE IBUF array specified in the LPA$ADSWP, LPA$DASWP, LPA$DISWP, or LPA$DOSWP call that initiated the sweep.

IWHEN — Specifies when to stop the sweep. VAX/VMS defines the following values:

0 = Abort sweep immediately. Uses the $CANCEL system service. This is the default sweep stop.

1 = Stop sweep when the current buffer transaction is completed. (This is the preferred way to stop requests.)

IND — Receives a success or failure code in the standard VAX/VMS format:

1 = Success

nnn = VAX/VMS error code issued by the $CANCEL system service

10.5.10 LPA$CLOCKA - Clock A Control

The LPA$CLOCKA subroutine sets the clock rate for Clock A.

The format of the LPA$CLOCKA call is as follows:

```
CALL LPA$CLOCKA (IRATE,IPRSET,[IND],[NUM])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

IRATE — Specifies the clock rate. One of the following values must be specified:

0 = Clock B overflow or no rate
1 = 1 MHz
2 = 100 kHz
3 = 10 kHZ
4 = 1 kHz
5 = 100 Hz
6 = Schmidt trigger 1
7 = Line frequency

IPRSET — Specifies the clock preset value. Maximum of 16 bits. The LPA$XRATE subroutine can be used to calculate this value. The clock rate divided by the clock preset value yields the clock overflow rate.

IND — Receives a success or failure code as follows:

1 = Clock A set successfully

nnn = VAX/VMS error code indicating an I/O error

NUM — Specifies the number to be appended to the logical name LPA11$. If defaulted, NUM is 0. This subroutine sets Clock A on the LPA11-K assigned to LPA11$num.

10.5.11 LPA$CLOCKB - Clock B Control

The LPA$CLOCKB subroutine provides the user with control of the KW11-K Clock B.

The format of the LPA$CLOCKB call is as follows:

```
CALL LPA$CLOCKB ([IRATE],IPRSET,MODE,[IND],[NUM])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

IRATE — Specifies the clock rate. One of the following values must be specified:

0 = Stops Clock B
1 = 1 MHz
2 = 100 kHz
3 = 10 kHz
4 = 1 kHz
5 = 100 Hz
6 = Schmidt trigger 3
7 = Line frequency

If IRATE is 0 (default), the clock is stopped and the IPRSET and MODE arguments are ignored.

IPRSET — Specifies the preset value by which the clock rate is divided to yield the overflow rate. Maximum of 8 bits. Overflow events can be used to drive Clock A. The LPA$XRATE subroutine can be used to calculate the IPRSET value.

MODE — Specifies options. VAX/VMS defines the following values:

1 = Clock B operates in non-interrupt mode.

2 = The feed B to A bit in the Clock B status register will be set (see Section 3.3 of the LPA11-K Laboratory Peripheral Accelerator User's Guide).

IND — Receives a success or failure code as follows:

1 = Clock B set successfully

nnn = VAX/VMS error code indicating an I/O error

NUM — Specifies the number to be appended to the logical name LPA11$. If defaulted, NUM is 0. This subroutine sets Clock B on the LPA11-K assigned to LPA11$num.

10.5.12 LPA$XRATE - Compute Clock Rate and Preset Value

The LPA$XRATE subroutine computes the clock rate and preset value for the LPA$CLOCKA and LPA$CLOCKB subroutines using the specified inter-sample interval (AINTRVL).

The two formats for the LPA$XRATE call are as follows:

```
CALL LPA$XRATE (AINTRVL,IRATE,IPRSET,IFLAG)
```

or,

```
ACTUAL=LPA$XRATE(AINTRVL,IRATE,IPRSET,IFLAG)
```

Arguments are as described in Section 10.5.1.2, with the following additions:

AINTRVL — Specifies the inter-sample time selected by the user. The time is expressed in decimal seconds. Data type is floating point.

IRATE — Receives the computed clock rate as a value from 1 to 5.

IPRSET — Receives the computed clock preset value.

IFLAG — If the computation is for Clock A, IFLAG is 0; if for Clock B, IFLAG is not 0 (the maximum preset value is 255). The IFLAG argument length is one byte.

ACTUAL	Receives the actual inter-sample time if called as a function. Data type is floating point. If there are truncation and roundoff errors, this time can be different from the specified inter-sample time. Note that when LPA$XRATE is called from VAX-11 FORTRAN IV-PLUS programs as a function, it must be explicitly declared a real function. Otherwise, LPA$XRATE defaults to an integer function.

If AINTRVL is too large or too small to be achieved, both IRATE and ACTUAL are returned to 0.

10.5.13 LPA$IBFSTS - Return Buffer Status

The LPA$IBFSTS subroutine returns information on the buffers used in a sweep.

The format of the LPA$IBFSTS call is as follows:

```
CALL LPA$IBFSTS (IBUF,ISTAT)
```

Argument descriptions are as follows:

IBUF	The IBUF array specified in the call that initiated the sweep.
ISTAT	Specifies a longword array with as many elements as there are buffers involved in the sweep (maximum of eight). LPA$IBFSTS fills each array element with the status of the corresponding buffer: +2 = Buffer in device queue. LPA$RLSBUF has been called for this buffer. +1 = Buffer in user queue. The LPA11-K has filled (data input) or emptied (data output) this buffer. 0 = Buffer is not in any queue. -1 = Buffer is in the in-use queue, that is, it is either being filled or emptied or is the next to be filled or emptied by the LPA11-K.

10.5.14 LPA$IGTBUF - Return Buffer Number

The LPA$IGTBUF subroutine returns the number of the next buffer to be processed by the application program, that is, the buffer at the head of the user queue (see Figure 10-4). LPA$IGTBUF should be called by a completion routine at AST level to determine the next buffer to process. If an event flag was specified in the start sweep call, LPA$IWTBUF, not LPA$IGTBUF, should be called.

The formats of the LPA$IGTBUF call are as follows:

```
CALL LPA$IGTBUF (IBUF,IBUFNO)
```

or,

```
IBUFNO=LPA$IGTBUF(IBUF)
```

Arguments are as described in Section 10.5.1.2, with the following additions:

IBUF	The IBUF array specified in the call that initiated the sweep.
IBUFNO	Returns the number of the next buffer to be filled or emptied by the application program.

Table 10-6 lists the possible combinations of IBUFNO and IOSB contents on the return from a call to LPA$IGTBUF. The first four words of the IBUF array contain the IOSB. If IBUFNO is -1, the IOSB must be checked to determine the reason.

Table 10-6
LPA$IGTBUF Call - IBUFNO and IOSB Contents

IBUFNO	IOSB(1)	IOSB(2)	IOSB(3),(4)	Meaning
n	0	(byte count)	0	Normal buffer complete.
-1	0	0	0	No buffers in queue. Request still active.
-1	1	0	0	No buffers in queue. Sweep terminated normally.
-1	VAX/VMS error code	0	LPA11-K ready-out and maint. registers (only if SS$_DEVREQERR, SS$_CTRLERR, or SS$_DEVCMDERR is returned)	No buffers in queue. Sweep terminated due to error condition. Section 10.6 describes the VAX/VMS error codes; Appendix A of the LPA11-K Laboratory Peripheral Accelerator User's Guide lists the LPA11-K error codes.

10.5.15 LPA$INXTBF - Set Next Buffer to Use

The LPA$INXTBF subroutine alters the normal buffer selection algorithm to allow the user to specify the next buffer to be filled or emptied. The specified buffer is reinserted at the head of the device queue.

The two formats of the LPA$INXTBF call are as follows:

```
CALL LPA$INXTBF (IBUF,IBUFNO,IND)
```

or,

```
IND=LPA$INXTBF(IBUF,IBUFNO)
```

Arguments are as described in Section 10.5.1.2, with the following additions:

IBUF	The IBUF array specified in the call that initiated the sweep.
IBUFNO	Specifies the number of the next buffer to be filled or emptied. The buffer must already be in the device queue.
IND	Returns the result of the call: 0 = Specified buffer was not in the device queue 1 = Next buffer was successfully set

10.5.16 LPA$IWTBUF - Return Next Buffer or Wait

The LPA$IWTBUF subroutine returns the next buffer to be processed by the application program, that is, the buffer at the head of the user queue. If the user queue is empty, LPA$IWTBUF waits until a buffer is available. If a completion routine was specified in the call that initiated the sweep, LPA$IGTBUF, not LPA$IWTBUF, should be called.

The two formats of the LPA$IWTBUF call are as follows:

```
CALL LPA$IWTBUF (IBUF,[IEFN],IBUFNO)
```

or,

```
IBUFNO=LPA$IWTBUF(IBUF,[IEFN])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

IBUF	The IBUF array specified in the call that initiated the sweep.
IEFN	Not used. This argument is present for compatibility only. (The event flag is the one specified in the start sweep call.)
IBUFNO	Returns the number of the next buffer to be filled or emptied by the application program.

Table 10-7 lists the possible combinations of IBUFNO and IOSB contents on the return from a call to LPA$IWTBUF. The first four words of the IBUF array contain the IOSB. If IBUFNO is -1, the IOSB must be checked to determine the reason.

Table 10-7
LPA$IWTBUF Call - IBUFNO and IOSB Contents

IBUFNO	IOSB(1)	IOSB(2)	IOSB(3),(4)	Meaning
n	0	(byte count)	0	Normal buffer complete.
-1	1	0	0	No buffers in queue. Sweep terminated normally.
-1	VAX/VMS error code	0	LPA11-K ready-out and maint. registers (only if SS$_DEVREQERR, SS$_CTRLERR, or SS$_DEVCMDERR is returned)	No buffers in queue. Sweep terminated due to error condition. Section 10.6 describes the VAX/VMS error codes; Appendix A of the LPA11-K Laboratory Peripheral Accelerator User's Guide lists the LPA11-K error codes.

10.5.17 LPA$RLSBUF - Release Data Buffer

The LPA$RLSBUF subroutine declares one or more buffers available to be filled or emptied by the LPA11-K. LPA$RLSBUF inserts the buffer at the tail of the device queue (see Figure 10-4).

The format of the LPA$RLSBUF call is as follows:

```
CALL LPA$RLSBUF (IBUF,[IND],INDEX0,INDEX1,...,INDEXN)
```

Arguments are as described in Section 10.5.1.2, with the following additions:

IBUF — The IBUF array specified in the call that initiated the sweep.

IND — Returns the success or failure status:

0 = Illegal buffer number or incorrect number of arguments specified, or a double buffer overrun occurred. A double buffer overrun can occur if buffer overrun was specified as a non-fatal error, a buffer overrun occurs, and buffer 0 was not released (probably on the user queue after a previous buffer overrun). LPA$_RLSBUF can return a double buffer overrun error only if buffer overrun was specified as a non-fatal error.

1 = Buffer(s) released successfully

INDEX0, etc. — Specify the indexes (0-7) of the buffers to be released. A maximum of eight indexes can be specified.

The LPA$RLSBUF subroutine must be called to release a buffer (or buffers) to the device queue before the sweep is initiated. (See Section 10.5.1.1 for a discussion on buffer management.) Note that LPA$RLSBUF does not verify whether or not the specified buffers are already in a queue. If a buffer is released when it is already in a queue, the queue pointers will be invalidated. This can cause unpredictable results.

10.5.18 LPA$RMVBUF - Remove Buffer from Device Queue

The LPA$RMVBUF subroutine removes a buffer from the device queue.

The format of the LPA$RMVBUF call is as follows:

```
CALL LPA$RMVBUF (IBUF,IBUFNO,[IND])
```

Arguments are as described in Section 10.5.1.2, with the following additions:

IBUF	The IBUF array specified in the call that initiated the sweep.
IBUFNO	Specifies the number of the buffer to remove from the device queue.
IND	Returns the success or failure status: 0 = Buffer not found in the device queue 1 = Buffer successfully removed from the device queue

10.5.19 LPA$CVADF - Convert A/D Input to Floating Point

The LPA$CVADF subroutine converts A/D input values to floating point numbers. LPA$CVADF is provided for compatibility reasons.

The formats of the LPA$CVADF call are as follows:

```
CALL LPA$CVADF (IVAL,VAL)
```

or,

```
VAL=LPA$CVADF(IVAL)
```

Argument descriptions are as follows:

IVAL	Contains the value (bits 11:0) read from the A/D input. Bits 15:12 are 0.
VAL	Receives the floating point value.

10.5.20 LPA$FLT16 - Convert Unsigned 16-bit Integer to Floating Point

The LPA$FLP16 subroutine converts unsigned 16-bit integers to floating point. LPA$FLT16 is provided for compatibility reasons.

The formats of the LPA$FLT16 call are as follows:

```
CALL LPA$FLT16 (IVAL,VAL)
```

or,

```
VAL=LPA$FLT16(IVAL)
```

Argument descriptions are as follows:

IVAL — An unsigned 16-bit integer.

VAL — Receives the converted value.

10.5.21 LPA$LOADMC - Load Microcode and Initialize LPA11-K

The LPA$LOADMC subroutine provides a program interface to the LPA11-K microcode loader. LPA$LOADMC sends a load request through a mailbox to the loader process to load microcode and initialize an LPA11-K (Section 10.7.1 describes the microcode loader process).

The format of the LPA$LOADMC call is as follows:

```
CALL LPA$LOADMC ([ITYPE][,NUM][,IND][,IERROR])
```

Argument descriptions are as follows:

ITYPE — The type of microcode to be loaded. VAX/VMS defines the following values:

Value	Meaning
1	Multirequest mode
2	Dedicated A/D mode
3	Dedicated D/A mode

If the ITYPE argument is defaulted, multirequest mode microcode is loaded.

NUM — The number to be appended to the logical name LPA11$. If defaulted, NUM is 0.

IND — Receives the completion status:

1 = Microcode loaded successfully.

nnn = VAX/VMS error code

IERROR — Provides additional error information. Receives the second longword of the IOSB if either SS$_CTRLERR, SS$_DEVCMDERR, or SS$_DEVREQERR is returned in IND. Otherwise, the contents of IERROR is undefined.

10.6 I/O STATUS BLOCK

The I/O status block format for the load microcode, start microprocessor, initialize LPA11-K, set clock, and start data transfer request QIO functions is shown in Figure 10-5.

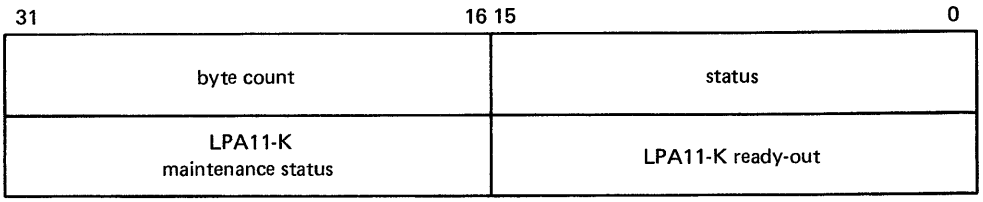


Figure 10-5 I/O Functions IOSB Content

VAX/VMS status values and the byte count are returned in the first longword. Status values are defined by the $SSDEF macro. The byte count is the number of bytes transferred by a IO$_LOADMCODE request. If SS$_CTRLERR, SS$_DEVCMDERR, or SS$_DEVREQERR is returned in the status word, the second longword contains the LPA11-K Ready-out Register and LPA11-K Maintenance Status Register values present at the completion of the request. The high byte of the Ready-out Register contains the specific LPA11-K error code (see Appendix A of the LPA11-K Laboratory Peripheral Accelerator User's Guide). Table 10-8 lists the status returns for LPA11-K I/O functions.

If high level language library procedures are used, the status returns listed in Table 10-8 can be returned from the resultant QIO functions. Since buffers are filled by these procedures asynchronously, two I/O status blocks are provided in the IBUF array: one for the high level language procedures and one for the LPA11-K driver. The first four words of the IBUF array contain the IOSB for the high level language procedures.

Table 10-8
LPA11-K Status Returns for I/O Functions

Status	Meaning
SS$_ABORT	Request aborted. A request in progress was cancelled by the $CANCEL system service. (Only for start data transfer request functions.)

(continued on next page)

Table 10-8 (Cont.)
LPA11-K Status Returns for I/O Functions

Status	Meaning
SS$_BUFNOTALIGN	Alignment error. If this error occurs for an initialize LPA11-K request, the initialize command table was not word-aligned. If this error occurs for a start data transfer request, there are several possible causes: • User status word (USW) not word-aligned • Buffer area not longword-aligned • Random Channel List (RCL) not word-aligned
SS$_CANCEL	Request cancelled by the $CANCEL system service before it started. (Only for the initialize LPA11-K, set clock, and start data transfer request functions.)
SS$_CTRLERR	Controller error. This is a fatal error that affects all LPA11-K activity. If this error occurs, the LPA11-K terminates all active requests. The third and fourth words of the IOSB contain the LPA11-K Ready-out Status Register and Maintenance Register contents. In particular, the high byte of the third word contains the specific LPA11-K error code (see Appendix A in the LPA11-K Laboratory Peripheral Accelerator User's Guide). (Only for the start microprocessor, initialize LPA11-K, set clock, and start data transfer request functions.)
SS$_DATACHECK	Data check error. A mismatch between the microcode in memory and the microcode loaded into the LPA11-K was detected. The second word of the IOSB contains the number of bytes successfully loaded. (Only for the load microcode function.)
SS$_DEVACTIVE	Device is active. The microcode cannot be loaded or the microprocessor cannot be started because there is an active data transfer request. (Only for the load microcode and start microprocessor functions.)
SS$_DEVCMDERR	LPA11-K command error. This error is associated with the issuance of a new LPA11-K command. The third and fourth words of the IOSB contain the LPA11-K Ready-out Status Register and Maintenance Register contents. In particular, the high byte of the third word contains the specific LPA11-K error code (see Appendix A in the LPA11-K Laboratory Peripheral Accelerator User's Guide). (Only for the initialize LPA11-K, set clock, and start data transfer request functions.)

(continued on next page)

Table 10-8 (Cont.)
LPA11-K Status Returns for I/O Functions

Status	Meaning
SS$_DEVREQERR	LPA11-K user request error. The third and fourth words of the IOSB contain the LPA11-K Ready-out Status Register and Maintenance Register contents. In particular, the high byte of the third word contains the specific LPA11-K error code (see Appendix A in the LPA11-K Laboratory Peripheral Accelerator User's Guide). (Only for start data transfer requests.)
SS$_EXQUOTA	AST quota exceeded. An AST cannot be queued for a buffer full/empty AST. Normally, a start data transfer request can require no more than three AST blocks at a time. (Only for start data transfer requests.)
SS$_INSFBUFDP	A UBA-buffered datapath was not available for allocation. (Only for start data transfer requests in dedicated mode.)
SS$_INSFMAPREG	Insufficient UBA map registers to map the command table or buffer areas. If the map registers were preallocated when the driver was loaded, the preallocation should be increased. (Only for the initialize LPA11-K and start data transfer request functions.)
SS$_INSFMEM	Insufficient dynamic memory to start request or allocate an AST block. (Only for start data transfer requests.)
SS$_IVBUFLEN	Incorrect length. If this error occurs for an initialize LPA11-K request, the initialize command table length is not the required 278 bytes. If this error occurs for a start data transfer request, there are several possible causes: • Command table length is not the required 40 bytes • Buffer area size is not evenly divisible by the number of buffers assigned • Individual buffer size is 0 • Individual buffer size is not a multiple of 2 for a multirequest mode request, or 4 for a dedicated mode request • Random Channel List length is 0 or not a multiple of 2 • Bit 15 in the last word of the Random Channel List is not set

(continued on next page)

Table 10-8 (Cont.)
LPA11-K Status Returns for I/O Functions

Status	Meaning
SS$_IVMODE	Incorrect mode. The first three bits (2:0) of the first word in the command table, that is, the mode word, are not 0. (Only for the initialize LPA11-K function.)
SS$_MCNOTVALID	Microcode has not been successfully loaded. (Only for the start microprocessor, initialize LPA11-K, set clock, and start data transfer request functions.)
SS$_PARITY	Parity error. A parity error occurred in a UBA-buffered datapath. (Only for start data transfer request in dedicated mode.)
SS$_POWERFAIL	A power failure occurred while a request was active. (Only for the start microprocessor, initialize LPA11-K, set clock, and start data transfer request functions.)
SS$_TIMEOUT	Device timeout. An interrupt was not received within one second after the request was issued. (Only for the start microprocessor, initialize LPA11-K, set clock, and start data transfer request functions.)

10.7 LOADING LPA11-K MICROCODE

The microcode loading and device initialization routines automatically load microcode on system initialization (if specified in the system manager's startup file) and on power recovery. These routines also allow a non-privileged user to load microcode and restart the system.

The LPA11-K loader and initialization routines consist of three parts:

- A microcode loader process which loads any of the three microcode versions, initializes the LPA11-K, and sets the clock rate. Loading is initiated by either a mailbox request or a power recovery AST. This process requires permanent mailbox (PRMMBX) and physical I/O privileges.

- An operator process which accepts operator commands or indirect file commands to load microcode and initialize an LPA11-K. This process uses a mailbox to send a load request to the loader process; temporary mailbox (TMPMBX) privilege is required.

- An LPA11-K procedure library routine that provides a program interface to the LPA11-K microcode loader. The procedure sends a load request through a mailbox to the loader process to load microcode and initialize an LPA11-K. Section 10.5.21 describes this routine in greater detail.

10.7.1 Microcode Loader Process

The microcode loader process loads microcode, initializes a specific LPA11-K, and sets the clock at the default rate (10 kHz interrupt rate). A bit set in a controller bitmap indicates that the specified controller was loaded. The process specifies a power recovery AST, creates a mailbox whose name (LPA$LOADER) is entered in the system logical name table, and then hibernates.

The correct device configuration is determined automatically. When LPA11-K initialization is performed, every possible device (see Table 10-1) is specified as present on the LPA11-K. If the LPA11-K returns a device not found error, the LPA11-K is reinitialized with that device omitted.

On receipt of a power recovery AST, the loader process examines the controller bitmap to determine which LPA11-Ks have been loaded. For each LPA11-K, the loader process performs the following functions:

- Obtains device characteristics
- Reloads the microcode previously loaded
- Reinitializes the LPA11-K
- Sets Clock A to the previous rate and preset value

10.7.2 Operator Process

The operator process loads microcode and initializes an LPA11-K through the use of either terminal or indirect file commands. The command input syntax is as follows:

```
devname/type
```

Devname is the device name of the LPA11-K to be loaded. A logical name can be specified. However, only one level of logical name translation is performed. If devname is omitted, LAA0 is the default name. /type specifies one of three types of microcode to load:

```
/MULTI_REQUEST = multirequest mode
/ANALOG_DIGITAL = dedicated A/D mode
/DIGITAL_ANOLOG = dedicated D/A mode
```

If /type is omitted, /MULTI_REQUEST is the default.

After receiving the command, the operator process formats a message and sends it to the loader process. Completion status is returned through a return mailbox.

10.8 RSX-11M VERSION 3.1 AND VAX/VMS DIFFERENCES

This section lists those areas where the VAX/VMS and RSX-11M Version 3.1 LPA11-K high level language support routines differ. The RSX-11M I/O Drivers Reference Manual provides a detailed description of the RSX-11M LPA11-K support routines. The exact differences between the VAX/VMS and RSX-11M routines can be determined by comparing the descriptions in the RSX-11M manual with the descriptions for the VAX/VMS routines in the preceding sections of this guide.

10.8.1 Alignment and Length

In VAX/VMS:

- Buffers must be contiguous.
- Buffers must be longword-aligned.
- The Random Channel List must be word-aligned.
- The IBUF array length is 50 longwords and must be longword-aligned.

10.8.2 Status Returns

In VAX/VMS:

- The I/O Status Block length is 8 bytes; numeric values of errors are different.
- Several routines return:

  1 - Success

  0 - Failure detected in support routine

  nnn - VAX/VMS status code. Failure detected in system service.

10.8.3 Sweep Routines

In VAX/VMS:

- If an event flag is specified, it must be within a %VAL( ) construction.
- A tenth argument, IND, has been added to return the success or failure status.

10.8.4 General

In VAX/VMS:

- The LUN argument is not used. Instead, the NUM argument specifies the number to be appended to the logical name LPA11$.
- All routine names have the prefix LPA$.
- In the LPA$SETIBF routine, buffer addresses are checked for contiguity.
- In the LPA$LAMSKS routine, the IUNIT argument is not used.

- In the LPA$IWTBUF routine, the IEFN argument is not used. The event flag specified in the sweep routine is used.
- The combinations of IBUFNO and I/O Status Block values returned by the LPA$IWTBUF and LPA$IGTBUF routines are different.
- If buffer overrun is specified as a non-fatal error, buffer 0 must be released. However, buffer 0 is set aside (placed on a queue) until a buffer overrun occurs. If a buffer overrun occurs and buffer 0 was not released, the LPA$RLSBUF routine returns an error the next time buffer 0 is released.

10.9 PROGRAMMING EXAMPLES

The following program examples use LPA11-K high level language procedures and LPA11-K Queue I/O functions.

10.9.1 LPA11-K High Level Language Program (Program A)

This program is an example of how the LPA11-K high level language procedures perform an A/D sweep using three buffers. The program uses default arguments whenever possible to illustrate the simplest possible calls. The program assumes that dedicated mode microcode has previously been loaded into the LPA11-K. Table 10-9 lists the variables used in this program.

Table 10-9
Program A Variables

Variable	Description
BUFFER	The data buffer array. BUFFER is a common area to guarantee longword-alignment.
IBUF	The LPA11-K high level language procedures use the IBUF array for local storage.
BUFNUM	BUFNUM contains the buffer number returned by LPA$IWTBUF. In this example, the possible values are 0, 1, and 2.
ISTAT	ISTAT contains the status return from the high level language calls.

```
C  ****************************************************************
C
C                              PROGRAM A
C
C  ****************************************************************


      INTEGER*2       BUFFER(1000,0:2),IOSB(4)
      INTEGER*4       IBUF(50),ISTAT,BUFNUM

      COMMON/AREA1/BUFFER

      EQUIVALENCE     (IOSB(1),IBUF(1))

C
C  SET CLOCK RATE TO 100 KHZ, CLOCK PRESET TO -10
C
      CALL LPA$CLOCKA(2,-10,ISTAT)
      IF (.NOT. ISTAT) GO TO 950
C
C  INITIALIZE IBUF ARRAY FOR SWEEP
C
      CALL LPA$SETIBF(IBUF,ISTAT,,BUFFER(1,0),BUFFER(1,1),BUFFER(1,2))
      IF (.NOT. ISTAT) GO TO 950
C
C  RELEASE ALL THE BUFFERS. NOTE USE OF BUFFER NUMBERS RATHER THAN
C  BUFFER NAMES.
C
      CALL LPA$RLSBUF(IBUF,ISTAT,0,1,2)
      IF (.NOT. ISTAT) GO TO 950
C
C  START A/D SWEEP
C
```

```
        CALL LPA$ADSWP(IBUF,1000,50,,,,,,ISTAT)
        IF (.NOT. ISTAT) GO TO 950
C
C  GET NEXT BUFFER FILLED WITH DATA.  IF BUFNUM IS NEGATIVE, THERE
C  ARE NO MORE BUFFERS AND THE SWEEP IS STOPPED.
C
100     BUFNUM = LPA$IWTBUF(IBUF)
        IF (BUFNUM .LT. 0) GO TO 800
C
C  PROCESS DATA IN BUFFER(1,BUFNUM) TO BUFFER (1000,BUFNUM)
        .
        .
        .
(Application dependent code is inserted at this point)
        .
        .
        .
C  RELEASE BUFFER TO BE FILLED AGAIN
C
200     CALL LPA$RLSBUF(IBUF,ISTAT,BUFNUM)
        IF (.NOT. ISTAT) GO TO 950
        GO TO 100
C
C  THERE ARE NO MORE BUFFERS TO PROCESS. CHECK TO ENSURE THAT THE
C  SWEEP ENDED SUCCESSFULLY. IOSB(1) CONTAINS EITHER 1 OR A
C  VAX/VMS STATUS CODE.
C
800     IF (.NOT. IOSB(1)) CALL LIB$STOP(%VAL(IOSB(1)))
        PRINT *,'SUCCESSFUL COMPLETION'
        GO TO 2000

C
C  ERROR RETURN FROM SUBROUTINE. ISTAT CONTAINS EITHER 0 OR
C  VAX/VMS ERROR CODE.
C
950     IF (ISTAT .NE. 0) CALL LIB$STOP(%VAL(ISTAT))
        PRINT   *,'ERROR IN LPA11-K SUBROUTINE CALL'
2000    STOP
        END

C  ***************************************************************
```

10.9.2 LPA11-K High Level Language Program (Program B)

This program is a more complex example of LPA11-K operations performed by the LPA11-K high level language procedures. The following operations are demonstrated:

- Program requested loading of LPA11-K microcode
- Setting the clock at a specified rate
- Use of non-default arguments whenever possible
- An A/D sweep that uses an event flag
- A D/A sweep that uses a completion routine
- Buffer overrun set (buffer overrun is a non-fatal error)
- Random Channel List addressing
- Sequential Channel addressing

Table 10-10 lists the variables used in this program.

Table 10-10
Program B Variables

Variable	Description
AD	An array of buffers for an A/D sweep (8 buffers of 500 words each)
DA	An array of buffers for a D/A sweep (2 buffers of 2000 words each)
IBUFAD	The IBUF array for an A/D sweep
IBUFDA	The IBUF array for a D/A sweep
RCL	The array containing the Random Channel List
ADIOSB	The array that contains the I/O status block for the A/D sweep. Equivalenced to the beginning of IBUFAD.
DAIOSB	The array that contains the I/O status block for the D/A sweep. Equivalenced to the beginning of IBUFDA.
ISTAT	Contains the status return from the high level language calls

```
C  ****************************************************************
C
C                                PROGRAM B
C
C  ****************************************************************

      EXTERNAL FILLBF
      REAL*4 LPA$XRATE

      INTEGER*2 AD(500,0:7),DA(2000,0:1),RCL(5),MODE,IPRSET
      INTEGER*2 ADIOSB(4),DAIOSB(4)

      INTEGER*4 IBUFAD(50),IBUFDA(50),LAMSKB(2)
      INTEGER*4 ISTAT,IERROR,IRATE,BUFNUM

      REAL*4 PERIOD

      COMMON  /SWEEP/AD,DA,IBUFAD,IBUFDA

      EQUIVALENCE (IBUFAD(1),ADIOSB(1)),(IBUFDA(1),DAIOSB(1))

      PARAMETER MULTI=1, HBIT='8000'X, LSTCHN=HBIT+7
C
C  SET UP RANDOM CHANNEL LIST.  NOTE THAT THE LAST WORD MUST HAVE BIT
C  15 SET.
C
      DATA RCL/2,6,3,4,LSTCHN/

C  ****************************************************************
C
C  LOAD MULTIREQUEST MODE MICROCODE AND SET THE CLOCK OVERFLOW RATE
C  TO 5 KHZ
C
C  ****************************************************************
C
C  LOAD MICROCODE ON LPA11-K ASSIGNED TO LPA11$3
C
      CALL LPA$LOADMC(MULTI,3,ISTAT,IERROR)
      IF (.NOT. ISTAT) GO TO 5000
C
C  COMPUTE CLOCK RATE AND PRESET.  SET CLOCK 'A' ON LPA11-K
C  ASSIGNED TO LPA11$3.
C
      PERIOD = LPA$XRATE(.0002,IRATE,IPRSET,0)
      IF (PERIOD .EQ. 0.0) GO TO 5500

      CALL LPA$CLOCKA(IRATE,IPRSET,ISTAT,3)
      IF (.NOT. ISTAT) GO TO 5000

C  ****************************************************************
C
C  SET UP FOR A/D SWEEP
C
C  ****************************************************************
C
C  INITIALIZE IBUF ARRAY. NOTE THE USE OF THE LAMSKB ARGUMENT BECAUSE
C  THE LPA11-K ASSIGNED TO LPA11$3 IS USED.
C
      CALL LPA$SETIBF(IBUFAD,ISTAT,LAMSKB,AD(1,0),AD(1,1),AD(1,2),
     1              AD(1,3),AD(1,4),AD(1,5),AD(1,6),AD(1,7))
      IF (.NOT. ISTAT) GO TO 5000

      CALL LPA$LAMSKS(LAMSKB,3)
C
```

```
C  SET UP RANDOM CHANNEL LIST SAMPLING (20 SAMPLES IN A SAMPLE
C  SEQUENCE)
C
      CALL LPA$SETADC(IBUFAD,,RCL,20,0,ISTAT)
      IF (.NOT. ISTAT) GO TO 5000
C
C  RELEASE BUFFERS FOR A/D SWEEP
C
      CALL LPA$RLSBUF(IBUFAD,ISTAT,0,1,2,3,4,5,6,7)
      IF (.NOT. ISTAT) GO TO 5000

C  ****************************************************************
C
C  SET UP FOR D/A SWEEP
C
C  ****************************************************************
C
C  NOTE THAT THE SAME LAMSKB ARRAY CAN BE USED BECAUSE THE LAMSKB
C  CONTENTS APPLY TO BOTH A/D AND D/A SWEEPS
C
      CALL LPA$SETIBF(IBUFDA,ISTAT,LAMSKB,DA(1,0),DA(1,1))
      IF (.NOT. ISTAT) GO TO 5000
C
C  SET UP SAMPLING PARAMETERS AS FOLLOWS:  INITIAL CHANNEL = 1.
C  NUMBER OF CHANNELS SAMPLED EACH SAMPLE SEQUENCE = 2, CHANNEL
C  INCREMENT = 2, THAT IS, SAMPLE CHANNELS 1 AND 3 EACH SAMPLE
C  SEQUENCE.
C
      CALL LPA$SETADC(IBUFDA,,1,2,2,ISTAT)
      IF (.NOT. ISTAT) GO TO 5000
C
C  FILL BUFFERS WITH DATA FOR OUTPUT TO D/A
C
      .
      .
      .
(Application dependent code is inserted here to fill buffers
DA(1,0) through DA(2000,0) and DA(1,1) through DA(2000,1) with data)
      .
      .
      .
C
C  RELEASE BUFFERS FOR D/A SWEEP
C
      CALL LPA$RLSBUF (IBUFDA,ISTAT,0,1)
      IF (.NOT. ISTAT) GO TO 5000

C  ****************************************************************
C
C  START BOTH SWEEPS
C
C  ****************************************************************
C
C  START A/D SWEEP. MODE BITS SPECIFY BUFFER OVERRUN IS NON-FATAL AND
C  MULTIREQUEST MODE. SWEEP ARGUMENTS SPECIFY 500 SAMPLES/BUFFER,
C  INDEFINITE SAMPLING, DWELL = 10 CLOCK OVERFLOWS, SYNCHRONIZE USING
C  EVENT FLAG 15, AND A DELAY OF 50 CLOCK OVERFLOWS.
C
      MODE = 16384 + 64
      CALL LPA$ADSWP(IBUFAD,500,0,MODE,10,%VAL(15),50,,,ISTAT)
      IF (.NOT. ISTAT) GO TO 5000
C
C  START D/A SWEEP.  MODE SPECIFIES MULTIREQUEST MODE.  OTHER
C  ARGUMENTS SPECIFY 2000 SAMPLES/BUFFER, FILL 15 BUFFERS, DWELL = 25
```

```
C  CLOCK OVERFLOWS, SYNCHRONIZE BY CALLING THE COMPLETION ROUTINE
C  'FILLBF', AND DELAY = 10 CLOCK OVERFLOWS.  (SEE THE FILLBF LISTING
C  AFTER THE PROGRAM B LISTING.)
C
        MODE = 64
        CALL LPA$DASWP(IBUFDA,2000,15,MODE,25,FILLBF,10,,,ISTAT)
        IF (.NOT. ISTAT) GO TO 5000

C  ****************************************************************
C
C  WAIT FOR AN A/D BUFFER AND THEN PROCESS THE DATA IT CONTAINS. D/A
C  BUFFERS ARE FILLED ASYNCHRONOUSLY BY THE COMPLETION ROUTINE FILLBF.
C
C  ****************************************************************
C
C  WAIT FOR A BUFFER TO BE FILLED BY A/D.  IF BUFNUM IS LESS THAN
C  ZERO, THE SWEEP HAS STOPPED (EITHER SUCCESSFULLY OR WITH AN ERROR).
C
100     BUFNUM = LPA$IWTBUF(IBUFAD)
        IF (BUFNUM .LT. 0) GO TO 1000
C
C  THERE IS A/D DATA IN AD(1,BUFNUM) THROUGH AD(500,BUFNUM)
C
        .
        .
        .
(Process the A/D data with the application dependent code inserted
here)
        .
        .
        .
C
C  ASSUME SWEEP SHOULD BE STOPPED WHEN THE LAST SAMPLE IN BUFFER
C  EQUALS 0. NOTE THAT THE SWEEP ACTUALLY STOPS WHEN THE BUFFER
C  CURRENTLY BEING FILLED IS FULL.  ALSO NOTE THAT LPA$IWTBUF
C  CONTINUES TO BE CALLED UNTIL THERE ARE NO MORE BUFFERS TO PROCESS.
C
        IF (AD(500,BUFNUM) .NE. 0) GO TO 200
        CALL LPA$STPSWP(IBUFAD,1,ISTAT)
        IF (.NOT. ISTAT) GO TO 5000

C
C  AFTER THE DATA HAS BEEN PROCESSED, THE BUFFER IS RELEASED TO BE
C  FILLED AGAIN.  THEN THE NEXT BUFFER IS OBTAINED FROM A/D.
C
200     CALL LPA$RLSBUF(IBUFAD,ISTAT,BUFNUM)
        IF (.NOT. ISTAT) GO TO 5000
        GO TO 100

C
C  ENTER HERE WHEN A/D SWEEP HAS ENDED.  CHECK FOR ERROR OR
C  SUCCESSFUL END. (NOTE: ASSUME THAT THE D/A SWEEP HAS ALREADY
C  ENDED - SEE COMPLETION ROUTINE FILLBF)
C
1000    IF(ADIOSB(1)) GO TO 6000
        CALL LIB$STOP(%VAL(ADIOSB(1)))

C
C  ENTER HERE IF THERE WAS AN ERROR RETURNED FROM ONE OF THE
C  LPA11-K HIGH LEVEL LANGUAGE CALLS. ISTAT CONTAINS EITHER 0
C  OR A VAX/VMS STATUS CODE.
C
5000    IF (ISTAT .NE. 0) CALL LIB$STOP (%VAL(ISTAT))
5500    PRINT *,'ERROR IN LPA11-K SUBROUTINE CALL'
```

```
        GO TO 7000

6000    PRINT *,'SUCCESSFUL COMPLETION'
7000    STOP
        END

C  ***************************************************************
C
C  SUBROUTINE FILLBF
C
C  ***************************************************************
C
C  THE FILLBF SUBROUTINE IS CALLED WHENEVER THE D/A HAS EMPTIED A
C  BUFFER, AND THAT BUFFER IS AVAILABLE TO BE REFILLED.  THIS
C  SUBROUTINE GETS THE BUFFER, FILLS IT, AND RELEASES IT BACK TO THE
C  LPA11-K.  NOTE THAT THE D/A SWEEP IS STOPPED AUTOMATICALLY AFTER
C  15 BUFFERS HAVE BEEN FILLED.  ALSO NOTE THAT FILLBF IS CALLED BY
C  AN AST HANDLER.  IT IS THEREFORE CALLED ASYNCHRONOUSLY FROM THE
C  MAIN PROGRAM AT AST LEVEL.  CARE SHOULD BE EXERCISED WHEN ACCESSING
C  VARIABLES THAT ARE COMMON TO BOTH LEVELS.
C
        INTEGER*2  AD(500,0:7),DA(2000,0:1),DAIOSB(4)
        INTEGER*4  IBUFAD(50),IBUFDA(50),BUFNUM,ISTAT
        EQUIVALENCE  (IBUFDA(1),DAIOSB(1))
        COMMON  /SWEEP/AD,DA,IBUFAD,IBUFDA
C
C  GET BUFFER NUMBER OF NEXT BUFFER TO FILL
C
        BUFNUM = LPA$IGTBUF(IBUFDA)
        IF (BUFNUM .LT. 0) GO TO 3000

C
C  FILL BUFFER WITH DATA FOR OUTPUT TO D/A
        .
        .
        .
(Application dependent code is inserted here to fill buffer
DA(1,BUFNUM) through DA(2000,BUFNUM) with data)
        .
        .
        .
C
C  RELEASE BUFFER
C
        CALL LPA$RLSBUF(IBUFDA,ISTAT,BUFNUM)
        GO TO 4000

C
C  CHECK FOR SUCCESSFUL END OF SWEEP
C
3000    IF(DAIOSB(1)) GO TO 4000

C
C  ERROR IN SWEEP
C
        CALL LIB$STOP(%VAL(DAIOSB(1)))

4000    RETURN
        END
C ***************************************************************
```

10.9.3 LPA11-K QIO Functions Program (Program C)

This sample program uses QIO functions to start an A/D data transfer from an LPA11-K. (The program assumes multirequest mode microcode has been loaded.) Sequential channel addressing is used. The data transfer is stopped after 100 buffers have been filled; no action is taken with the data as the buffers are filled. Note that this program starts the data transfer and then waits until the QIO operation completes.

```
; ********************************************************************
;
;                          PROGRAM C
;
; ********************************************************************

        .TITLE  LPA11-K EXAMPLE PROGRAM
        .IDENT  /V01/

        .PSECT  LADATA,LONG


IOSB:   .BLKQ   1                       ; I/O STATUS BLOCK
COUNT:  .LONG   0                       ; COUNT OF BUFFERS FILLED

CBUFF:                                  ; COMMAND BUFFER FOR START
                                        ; DATA QIO
        .WORD   ^X20A                   ; MODE = SEQUENTIAL CHANNEL
                                        ; ADDRESSING, A/D, MULTI-
                                        ; REQUEST MODE
        .WORD   3                       ; VALID BUFFER MASK (4
                                        ; BUFFERS)
        .LONG   USW                     ; USER STATUS WORD ADDRESS
        .LONG   4000                    ; AGGREGATE BUFFER LENGTH
        .LONG   DATA_BUFFER0            ; ADDRESS OF DATA BUFFERS
        .LONG   0                       ; NO RANDOM CHANNEL LIST
                                        ; LENGTH
        .LONG   0                       ; NO RANDOM CHANNEL LIST
                                        ; ADDRESS
        .WORD   10                      ; DELAY
        .BYTE   0                       ; START CHANNEL
        .BYTE   1                       ; CHANNEL INCREMENT
        .WORD   16                      ; NUMBER OF SAMPLES IN
                                        ; SAMPLE SEQUENCE
        .WORD   1                       ; DWELL
        .BYTE   0                       ; START WORD NUMBER
        .BYTE   0                       ; EVENT MARK WORD
        .WORD   0                       ; START WORD MASK
        .WORD   0                       ; EVENT MARK MASK
        .WORD   0                       ; FILLS OUT COMMAND BUFFER

USW:    .WORD   0                       ; USER STATUS WORD

        .ALIGN  LONG                    ;BUFFERS MUST BE
                                        ; LONGWORD ALIGNED

DATA_BUFFER0: .BLKW     500             ; DATA BUFFERS
DATA_BUFFER1: .BLKW     500
DATA_BUFFER2: .BLKW     500
DATA_BUFFER3: .BLKW     500

DEVNAME: .LONG  4,LANAME
```

```
CHANNEL: .BLKW   1                          ; CONTAINS CHANNEL NUMBER

LANAME:  .ASCII  /LAA0/

         .PSECT  LACODE,NOWRT

START:   .WORD   0
         $ASSIGN_S DEVNAME,CHANNEL          ; ASSIGN CHANNEL
         BLBS    R0,5$                      ; NO ERROR
         BRW     ERROR                      ; ERROR

5$:                                         ; SET CLOCK OVERFLOW RATE
                                            ; TO 2 KHZ. (1 MHZ RATE
                                            ; DIVIDED BY 500 PRESET)
         $QIOW_S ,CHANNEL,#IO$_SETCLOCK,-
                 IOSB,,,,#1,#^X143,#-500
         BLBC    R0,ERROR                   ; ERROR
         MOVZWL  IOSB,R0                    ; PICK UP I/O STATUS
         BLBC    R0,ERROR                   ; ERROR

                                            ; START DATA TRANSFER
         CLRW    USW                        ; CLEAR USW (START WITH
                                            ; BUFFER 0)
         MOVL    #100,COUNT                 ; FILL 100 BUFFERS
         $QIOW_S ,CHANNEL,#IO$_STARTDATA,-
                 IOSB,,,CBUFF,#40,#BFRAST
         BLBC    R0,ERROR                   ; ERROR

; NOTE THAT THE QIO WAITS UNTIL IT FINISHES. NORMALLY, THE DATA IS
; PROCESSED HERE AS THE BUFFERS ARE FILLED. CHECK FOR ERROR WHEN
; THE QIO COMPLETES.

         MOVZWL  IOSB,R0                    ; PICK UP I/O STATUS
         BLBC    R0,ERROR                   ; ERROR
         RET                                ; ALL DONE - EXIT

ERROR:                                      ; ENTER HERE IF ERROR.
                                            ; STATUS IN R0.
         PUSHL   R0                         ; PUSH ONTO STACK
         CALLS   #1,LIB$STOP                ; SIGNAL ERROR


BFRAST:                                     ; BUFFER AST ROUTINE.
                                            ; BFRAST IS CALLED WHENEVER
                                            ; A BUFFER IS FILLED.
         .WORD   0
         INCB    USW+1                      ; ADD 1 TO BUFFER NUMBER
         CMPZV   #0,#3,USW+1,#3             ; HANDLE WRAPAROUND

         BLEQ    10$
         CLRB    USW+1                      ; USE BUFFER 0

10$:     DECL    COUNT                      ; DECREMENT BUFFER COUNT
         BGTR    20$
         BISB    #^X40,USW+1                ; ENOUGH BUFFERS FILLED -
                                            ; SET STOP BIT
20$:     BICB    #^X80,USW+1                ; CLEAR DONE BIT
         RET

         .END    START

; ***************************************************************
```

APPENDIX B

I/O FUNCTION CODES

This appendix lists the function codes and function modifiers defined in the $IODEF macro. The arguments for these functions are also listed.

B.1 TERMINAL DRIVER

Function	Arguments	Modifier
IO$_READVBLK IO$_READLBLK IO$_READPBLK IO$_READPROMPT	P1 - buffer address P2 - buffer size P3 - timeout P4 - read terminator block address P5 - prompt string buffer address[1] P6 - prompt string buffer size[1]	IO$M_NOECHO IO$M_CVTLOW IO$M_NOFILTR IO$M_TIMED IO$M_PURGE IO$M_DSABLMBX IO$M_TRMNOECHO
IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1 - buffer address P2 - buffer size P3 - (ignored) P4 - carriage control specifier[2]	IO$M_CANCTRLO IO$M_ENABLMBX IO$M_NOFORMAT
IO$_SETMODE IO$_SETCHAR	P1 - characteristics buffer address P2 - Characteristics buffer size P3 - speed specifier P4 - fill specifier P5 - parity flags	
IO$_SETMODE!IO$M_HANGUP IO$_SETCHAR!IO$M_HANGUP	(none)	
IO$_SETMODE!IO$M_CTRLCAST IO$_SETMODE!IO$M_CTRLYAST IO$_SETCHAR!IO$M_CTRLCAST IO$_SETCHAR!IO$M_CTRLYAST	P1 - AST service routine address P2 - AST parameter P3 - access mode to deliver AST	
IO$_SENSEMODE IO$_SENSECHAR	P1 - Characteristics buffer address P2 - Characteristics buffer size	

[1] Only for IO$_READPROMPT

[2] Only for IO$_WRITELBLK and IO$_WRITEVBLK

B.2 DISK DRIVERS

Functions	Arguments	Modifiers
IO$_READVBLK IO$_READLBLK IO$_READPBLK IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1 - buffer address P2 - byte count P3 - disk address	IO$M_DATACHECK IO$M_INHRETRY IO$M_INHSEEK[1]
IO$_SETMODE IO$_SETCHAR	P1 - characteristic buffer address	IO$_INHRETRY
IO$_CREATE IO$_ACCESS IO$_DEACCESS IO$_MODIFY IO$_DELETE	P1 - FIB descriptor address P2 - file name string address P3 - result string length address P4 - result string descriptor address P5 - attribute list address	IO$M_CREATE[2] IO$M_ACCESS[2] IO$M_DELETE[3]

[1]Only for IO$_READPBLK and IO$_WRITEPBLK

[2]Only for IO$_CREATE and IO$_ACCESS

[3]Only for IO$_CREATE and IO$_DELETE

B.3 MAGNETIC TAPE DRIVERS

Functions	Arguments	Modifiers
IO$_READVBLK IO$_READLBLK IO$_READPBLK IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1 - buffer address P2 - byte count	IO$M_DATACHECK IO$M_INHRETRY IO$M_REVERSE[1] IO$M_INHEXTGAP[2]
IO$_SETMODE IO$_SETCHAR	P1 - characteristics buffer address	IO$M_INHRETRY IO$M_INHEXTGAP
IO$_CREATE IO$_ACCESS IO$_DEACCESS IO$_MODIFY IO$_ACPCONTROL	P1 - FIB descriptor address P2 - file name string address P3 - result string length address P4 - result string descriptor address P5 - attribute list address	IO$M_CREATE[3] IO$M_ACCESS[3] IO$M_DMOUNT[4]
IO$_SKIPFILE	P1 - skip n tape marks	IO$M_INHRETRY
IO$_SKIPRECORD	P1 - skip n records	IO$M_INHRETRY

[1]Only for read functions

[2]Only for write functions

[3]Only for IO$_CREATE and IO$_ACCESS

[4]Only for IO$_ACPCONTROL

Functions	Arguments	Modifiers
IO$_MOUNT	(none)	
IO$_REWIND IO$_REWINDOFF	(none)	IO$M_INHRETRY IO$M_NOWAIT
IO$_WRITEOF	(none)	IO$M_INHEXTGAP IO$M_INHRETRY
IO$_SENSEMODE	(none)	IO$M_INHRETRY

[1]Only for read functions
[2]Only for write functions
[3]Only for IO$_CREATE and IO$_ACCESS
[4]Only for IO$_ACPCONTROL

B.4 LINE PRINTER DRIVER

Functions	Arguments	Modifiers
IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1 - buffer address P2 - buffer size P3 - (ignored) P4 - carriage control specifier[1]	(none)
IO$_SETMODE IO$_SETCHAR	P1 - characteristics buffer address	(none)

[1]Only for IO$_WRITEVBLK and IO$_WRITELBLK

B.5 CARD READER DRIVER

Functions	Arguments	Modifiers
IO$_READLBLK IO$_READVBLK IO$_READPBLK	P1 - buffer address P2 - byte count	IO$M_BINARY IO$M_PACKED
IO$_SETMODE IO$_SETCHAR	P1 - characteristics buffer address	(none)
IO$_SENSEMODE	(none)	

B.6 MAILBOX DRIVER

Functions	Arguments	Modifiers
IO$_READVBLK IO$_READLBLK IO$_READPBLK IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1 - buffer address P2 - buffer size	IO$M_NOW
IO$_WRITEOF	(none)	
IO$_SETMODE!IO$M_READATTN	P1 - AST address	
IO$_SETMODE!IO$M_WRTATTN	P1 - AST parameter	

B.7 DMC11 DRIVER

Functions	Arguments	Modifiers
IO$_READLBLK IO$_READPBLK IO$_READVBLK IO$_WRITELBLK IO$_WRITEPBLK IO$_WRITEVBLK	P1 - buffer address P2 - message size P6 - diagnostic buffer[2]	IO$M_DSABLMBX[1] IO$M_NOW[1] IO$M_ENABLMBX[3]
IO$_SETMODE IO$_SETCHAR	P1 - characteristics buffer address	
IO$_SETMODE!IO$M_ATTNAST IO$_SETCHAR!IO$M_ATTNAST	P1 - AST service routine address P2 - (ignored) P3 - AST access mode	
IO$_SETMODE!IO$M_SHUTDOWN IO$_SETCHAR!IO$M_SHUTDOWN	P1 - characteristics block address	
IO$_SETMODE!IO$M_STARTUP IO$_SETCHAR!IO$M_STARTUP	P1 - characteristics block address P2 - (ignored) P3 - receive message blocks	

[1]Only for IO$_READPBLK and IO$_WRITEPBLK
[2]Only for IO$_READLBLK and IO$_READPBLK
[3]Only for IO$_WRITELBLK and IO$_WRITEPBLK

B.8 ACP INTERFACE DRIVER

Functions	Arguments	Modifiers
IO$_CREATE IO$_ACCESS IO$_DEACCESS IO$_MODIFY IO$_DELETE IO$_ACPCONTROL	P1 - FIB descriptor address P2 - file name string address P3 - result string length address P4 - result string descriptor address P5 - attribute list address	IO$M_CREATE[1] IO$M_ACCESS[1] IO$M_DELETE[2] IO$M_DMOUNT[3]
IO$_MOUNT	(none)	

[1]Only for IO$_CREATE and IO$_ACCESS
[2]Only for IO$_CREATE and IO$_DELETE
[3]Only for IO$_ACPCONTROL

B.9 LPA11-K DRIVER

QIO Functions	Arguments	Modifier
IO$_LOADCODE	P1 - starting address of microcode to be loaded P2 - load byte count P3 - starting microprogram address to receive microcode	(none)
IO$_STARTMPROC	(none)	(none)
IO$_INITIALIZE	P1 - address of Initialize Command Table P2 - initialize command buffer length	(none)
IO _SETCLOCK	P2 - mode of operation P3 - clock control and status P4 - real-time clock preset value (2's complement)	(none)
IO$_STARTDATA	P1 - Data Transfer Command Table address P2 - Data Transfer Command Table length P3 - normal completion AST address P4 - overrun completion AST address	IO$M_SETEVF

High Level Language Subroutines	Functions
LPA$ADSWP	Start A/D converter sweep
LPA$DASWP	Start D/A converter sweep
LPA$DISWP	Start digital input sweep
LPA$DOSWP	Start digital output sweep
LPA$LAMSKS	Specify LPA11-K controller and digital mask words
LPA$SETADC	Specify channel select parameters
LPA$SETIBF	Specify buffer parameters
LPA$STPSWP	Stop sweep
LPA$CLOCKA	Set Clock A rate
LPA$CLOCKB	Set Closk B rate
LPA$XRATE	Compute clock rate and present value
LPA$IBFSTS	Return buffer status
LPA$IGTBUF	Return next available buffer
LPA$INXTBF	Alter buffer order
LPA$IWTBUF	Return next buffer or wait
LPA$RLSBUF	Release buffer to LPA11-K
LPA$RMVBUF	Remove buffer from device queue
LPA$CVADF	Convert A/D input to floating point
LPA$FLT16	Convert unsigned integer to floating point
LPA$LOADMC	Load microcode and initialize LPA-11K

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W