RSX–11M/M–PLUS
I/O Drivers Reference Manual
Order No. AA-FD09A-TC

digital software

RSX–11M/M–PLUS
I/O Drivers Reference Manual
Order No. AA-FD09A-TC

RSX–11M Version 4.2
RSX–11M–PLUS Version 3.0

digital equipment corporation · maynard, massachusetts

First Printing, May 1979
Revised, December 1981
Revised, July 1985

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

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

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

Printed in Australia

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	PDT
DEC/CMS	EduSystem	RSTS
DEC/MMS	IAS	RSX
DECnet	MASSBUS	UNIBUS
DECsystem-10	MicroPDP-11	VAX
DECSYSTEM-20	Micro/RSTS	VMS
DECUS	Micro/RSX	VT
DECwriter	PDP	digital

ZK2645

HOW TO ORDER ADDITIONAL DOCUMENTATION

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

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

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6215 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

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

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
PSG Business Manager
c/o Digital's local subsidiary or approved distributor

CONTENTS

CONTENTS

CONTENTS

CONTENTS

FIGURES

TABLES

PREFACE

MANUAL OBJECTIVES

This manual provides all the information needed to interface directly with the I/O device drivers supplied as part of the Micro/RSX system.

INTENDED AUDIENCE

This manual is for experienced RSX-11M/M-PLUS programmers who want to take advantage of the time and/or space savings that result from direct use of the I/O drivers. Readers should be familiar with the information contained in the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual, have some experience using the Task Builder and either MACRO-11 or FORTRAN programs, and be familiar with the manuals describing their use.

STRUCTURE OF THE MANUAL

Chapter 1 provides an overview of RSX-11M/M-PLUS input/output operations. It introduces you to logical unit numbers, directive parameter blocks, event flags, macro calls, and so on; includes discussions of the standard I/O functions common to a variety of devices; and summarizes standard error and status conditions relating to completion of I/O requests.

Chapters 2 through 24 describe the use of all device drivers supported by RSX-11M/M-PLUS. Each of these chapters is structured in similar fashion and focuses on the following basic elements:

- The device, including information on physical characteristics such as speed, capacity, access, and usage
- The standard functions that the devices support and descriptions of device-specific functions
- The special characters, carriage control codes, and functional characteristics, if relevant
- The error and status conditions that the driver returns on acceptance or rejection of I/O requests
- Programming hints

Appendixes A through C provide quick reference material on I/O functions and status codes.

ASSOCIATED MANUALS

The following Micro/RSX manuals may be useful:

- RSX-11M/M-PLUS Information Directory and Master Index
- RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual
- RSX-11M/M-PLUS and Micro/RSX Task Builder Manual
- PDP-11 MACRO-11 Language Reference Manual
- RSX-11M/RSX-11S or RSX-11M-PLUS Release Notes

In addition, documentation for programming in any of the MicroPDP-11 languages may be helpful.

CONVENTIONS USED IN THIS MANUAL

The following conventions are observed in this manual.

Convention	Meaning
[ ]	Square brackets; enclose optional syntax
{ }	Braces; indicate that one of the enclosed items must be selected.
...	Horizontal ellipsis; indicates that parameters have been omitted. In QIO macro calls in this manual, indicates that standard QIO parameters have been omitted.
,,,	Consecutive commas; used in coding examples to indicate null arguments. You may omit commas that indicate null trailing optional arguments.

Note that while RSX-11M/M-PLUS systems require certain parameters, they ignore them. These parameters are necessary to maintain compatibility with RSX-11D.

Furthermore, except in MACRO-11 coding examples, all numbers are assumed to be decimal unless otherwise specified. In MACRO-11 coding examples, the reverse is true: all numbers are considered to be octal unless followed by a decimal point (which indicates a decimal number).

Finally, in FORTRAN subroutine models, parameters that begin with the letters i through n indicate integer variables. In general, where a call uses both i and n prefixes, the i form indicates the name of an array and the n form specifies the size of the array.

All integer arrays and variables are assumed to occupy one storage word per variable (that is, INTEGER*2) and all real arrays and variables are assumed to occupy two storage words per variable (that is, REAL*4).

SUMMARY OF TECHNICAL CHANGES

This revision of the **RSX-11M/M-PLUS I/O Drivers Reference Manual** reflects the following software technical changes and additions:

- A new I/O function, IO.EIO (Extended I/O), which contains new subfunctions, has been added to the full-duplex terminal driver.
- Support for the DZQ11 4-line terminal multiplexer, a DZV11 replacement, has been added to the full-duplex terminal driver.
- Support of the full-duplex terminal driver has been extended to allow its use as a Network Command Terminal.
- Support for the RC25 and RD52 disks and the KDA50 controller has been added to the disk drivers.
- Support for the TK25, TK50, and TU81 magnetic tape drives has been added to the tape drivers.
- Support for the KMC-11 line printer interface has been added to the line printer driver.
- Support for the RSX QIO DEUNA driver has been added.

In addition to these changes, Chapters 1 and 2 have been reorganized to make the information more easily accessible to the reader. Chapter 13, which describes the RSX QIO DEUNA driver, is completely new. Appendixes A and B have been updated to reflect the new I/O functions, subfunctions, and error codes that have been added.

CHAPTER 1

RSX-11M/M-PLUS INPUT/OUTPUT

1.1 OVERVIEW OF RSX-11M I/O

The RSX-11M/M-PLUS operating system supports a wide variety of input and output devices, including disks, DECtapes, magnetic tapes, tape cassettes, line printers, card readers, and such laboratory and industrial devices as analog-to-digital converters, universal digital controllers, and laboratory peripheral systems.

Digital Equipment Corporation supplies the drivers for these devices as part of the system software. This manual describes all the device drivers that the RSX operating system supports and the characteristics, functions, error conditions, and programming hints associated with each. You can add devices that this manual does not describe to basic RSX system configurations, but you must develop and maintain your own drivers for these devices. (See the RSX-11M Guide to Writing an I/O Driver, or the RSX-11M-PLUS Guide to Writing an I/O Driver, depending upon the system you are using.)

Input/output operations under RSX-11M are extremely flexible and are as device- and function-independent as possible. Programs issue I/O requests to logical units that you previously associated with particular physical device units. Each program or task can establish its own correspondence between physical device units and logical unit numbers (LUNs). The Executive queues I/O requests as your task issues them and subsequently processes them according to the relative priority of the tasks that issued them. Your tasks can issue I/O requests for appropriate devices through either the File Control Services or Record Management Services, or your tasks can interface directly to an I/O driver by the Queue I/O (QIO$) Executive directive macro.

Your task requests all of the I/O services that this manual describes by using QIO$ Executive directive macros. A function code that you include in the QIO$ macro indicates the particular input or output operation that the system and the driver is to perform. I/O functions can request such operations as:

- Attaching or detaching a physical device unit for a task's exclusive use
- Reading or writing a logical or virtual block of data
- Canceling a task's I/O requests

QIO macros can also specify a wide variety of device-specific input/output operations (for example, reading DECtape in reverse, rewinding cassette tape).

1.2 PHYSICAL, LOGICAL, AND VIRTUAL I/O

An I/O transfer can take place in three possible modes: physical, logical, and virtual.

Physical I/O takes place by reading and writing data in the actual physical units that the hardware accepts (for example, sectors on a disk). For most devices, physical I/O is identical to logical I/O. For example, the RK05 Cartridge Disk has sectors of 256 words, the same size as RSX-11M logical blocks for all disks. Thus, for the RK05, a logical block maps directly into a physical block. However, the mapping is not one to one for other devices. The RF11 Fixed Head Disk, for example, is word addressable, but no physical I/O may be done with the RF11. Data is always written in 256-word logical blocks. The system records data for the RX01 Flexible Disk in physical sectors of 64 words each. Therefore, logical blocks for the RX01 are made up of four physical sectors.

Logical I/O takes place by reading and writing data in blocks convenient for the operating system. For most devices logical blocks map directly into physical blocks. For block-structured devices (for example, disks), logical blocks are numbered, beginning with 0. For non-block-structured devices (for example, terminals), logical blocks are not addressable.

Virtual I/O takes place by reading and writing data to open files. In this case, the Executive maps virtual blocks into logical blocks. For file-structured devices (disks or DECtapes), virtual blocks are the same size as logical blocks, are numbered starting from one (1), and are relative to the file rather than to the device. For non-file-structured devices, the mapping from virtual block to logical block is direct.

1.3 LOGICAL UNITS

This section describes the construction of the logical unit table and the use of logical unit numbers.

1.3.1 Logical Unit Number

A logical unit number, or LUN, is a number that the system associates with a physical device unit during RSX-11M/M-PLUS I/O operations. For example, you might associate LUN 1 with one of the terminals in the system, LUNs 2, 3, 4, and 5 with DECtape drives, and LUNs 6, 7, and 8 with disk units. The association is a dynamic one; each task running in the system can establish its own correspondence between LUNs and physical device units, and can that association at almost any time. This dynamic, flexible association is a major factor in the device-independent programming of the system.

A logical unit number is simply a short name for the association between a logical unit and a physical device unit. Once the association has been made, the LUN provides a direct and efficient mapping to the physical device unit, thus eliminating the searching of device tables whenever the system encounters a reference to a physical device unit.

Remember that, although you or a task can change the association of a LUN to a physical device unit at any time, reassigning a LUN at run time causes pending I/O requests for the previous LUN assignment to be canceled. Therefore, you must verify that all outstanding I/O requests for a LUN have been serviced before you associate that LUN with another physical device unit.

1.3.2 Logical Unit Table

There is one Logical Unit Table (LUT) for each task running in an RSX-11M/M-PLUS operating system. The task header contains this table as a variable-length block. Each LUT contains enough 2-word entries for the number of logical units. You specify the number of logical units in the Task Builder by the "UNITS=" option when you build your task.

The first word of each 2-word entry contains a pointer to the Unit Control Block that represents the physical device unit currently associated with that LUN. This linkage may be indirect; that is, you may force redirection of references from one unit to another unit with the DCL command ASSIGN/REDIRECT. The second word of each 2-word entry contains a pointer to the window block of the task that has a file open and mounted. The window block contains pointers to areas on the file that are accessed by the task.

Each 2-word entry contains a pointer to the Unit Control Block that represents the physical device unit currently associated with that LUN. Whenever your task issues an I/O request, the system matches the appropriate physical device unit (by using the Unit and Device Control Blocks and other structures) to the LUN that the call specifies. The system does this by indexing into the LUT by the LUN number. Thus, if the call specifies 6 as the LUN, the system accesses the sixth 2-word entry in the LUT and associates the I/O request with the physical device unit to which the entry points. The number of LUN assignments valid for a task ranges from 0 to 255, but it cannot be greater than the number of LUNs specified at task-build time.

Figure 1-1 illustrates a typical Logical Unit Table.

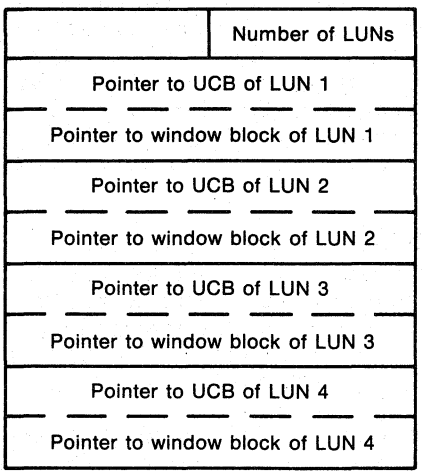


Figure 1-1 Logical Unit Table

1.3.3 Changing LUN Assignments

Logical unit numbers have no significance until you associate a LUN with a physical device unit by using one of the following methods:

- At the time you build the task that is to do the I/O operation, you can specify an ASG (Assign) keyword option to the Task Builder. This option associates a physical device unit with a logical unit number referenced by the task being built.
- You or the system operator can issue a REASSIGN command to MCR or a ASSIGN/REDIRECT command to DCL. This command reassigns a LUN to another physical device unit and thus changes the correspondence between the LUN and the physical device unit. Note that this reassignment has no effect on the in-core image of a task.
- At run time, a task can dynamically change a LUN assignment by issuing the Assign LUN Executive directive macro (ALUN$). This changes the association of a LUN with a physical device unit during task execution.

1.4 ISSUING AN I/O REQUEST

Your tasks perform I/O in the RSX-11M/M-PLUS system by submitting requests for I/O service as Queue I/O (QIO$) or Queue I/O and Wait (QIOW$) Executive directive macros. See the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for a complete description of system directives.

The RSX-11M/M-PLUS operating system has a set of system macros that make issuing QIO$ macros easier. You must make these macros available to the source program by placing the MACRO-11 Assembler directive .MCALL in the source program. The macros reside in the System Macro Library (LB:[1,1]RSXMAC.SML). Section 1.6.4 describes the function of .MCALL.

In RSX-11M/M-PLUS, as in most multiprogramming systems, tasks do not normally access physical device units directly. Instead, they use I/O services that the Executive provides, because it can effectively multiplex the use of physical device units over many tasks. The Executive routes I/O requests to the appropriate device driver and queues them by the priority of the requesting task. I/O operations proceed concurrently with other activities in an RSX-11M/M-PLUS system.

Before the Executive queues a QIO$ request to the driver, the QIO$ must pass a series of tests executed by the Executive. If the request fails, the Executive rejects it. The Executive signals this rejection by setting the C-bit. As good programming practice, you should check for directive rejection by following the QIO$ macro with a MACRO-11 BCS instruction or its equivalent.

After the Executive queues an I/O request, the system does not wait for the operation to complete. Perhaps the task that issued the QIO$ request cannot proceed until the I/O operation completes. In this case, the task should specify an event flag (see Section 1.4.1.4) in the QIO$ request and should issue a Wait For Single Event Flag (WTSE$) Executive macro specifying the same event flag at the point where synchronization must occur. Your task then waits for the I/O to complete by waiting for the Executive to set the specified event flag.

The QIO$ and Wait (QIOW$) macro is a more economical way to achieve this synchronization. QIOW$ waits until the system completes the I/O before returning control to the task. Thus, the additional WTSE$ macro is not necessary.

Each QIO$ or QIOW$ macro must supply sufficient information to identify and queue the I/O request. You may also want to include locations in your task to receive error or status codes, and to specify the address of an asynchronous system trap service routine. Certain types of I/O operations require the specification of device-dependent information as well. Typical QIO$ parameters are the following:

- I/O function to be performed
- Logical unit number associated with the physical device unit to be accessed
- Optional event flag number for synchronizing I/O completion processing (required for QIOW$)
- Optional address of the I/O status block to which the xp Status block, I/O Executive returns information indicating successful or unsuccessful completion
- Optional address of an asynchronous system trap service routine in your task to be entered upon completion of the I/O request
- Optional device- and function-dependent parameters specifying such items as the starting address of a data buffer, the size of the buffer, and a block number

Several of the first six parameters in the QIO$ macro are optional, but you must reserve space for these parameters. During expansion of a QIO$ macro, the Executive defaults to a value of 0 for all null (omitted) parameters. Inclusion of the device- and function-dependent parameters depends on the physical device unit and function that you specify. If you want to specify only an I/O function code, a LUN, and an address for an asynchronous system trap service routine, issue the following:

```
QIO$    IO.ATT,6,,,,ASTOX
```

where IO.ATT is the QIO$ function code and the following describes the meaning of the parameters:

Parameter	Meaning
IO.ATT	The I/O function code for attach.
6	The LUN or Logical Unit Number associated with the device unit.
,,,	Null arguments for the event flag number, the request priority, and the address of the I/O status block.
ASTOX	The AST address using the symbolic name ASTOX.

The system requires no additional device- or function-dependent parameters for an attach function. Section 1.7 describes the three legal forms of the macro.

For convenience, you may omit any comma if no parameters appear to the right of it. Therefore, you could issue the command above as follows, if you did not want the asynchronous system trap:

```
QIO$   IO.ATT,6
```

All extra commas have been dropped. However, if a parameter appears to the right of any place-holding comma, that comma must be retained.

1.4.1 QIO$ Macro Format

The arguments for a specific QIO$ macro call may be different for each I/O device your task accesses and for each I/O function it requests. However, the general format of the call is common to all devices. It appears as follows:

```
QIO$   fnc,lun,[efn],[pri],[isb],[ast],[<p1,p2,...,p6>]
```

1.4.1.1 Syntax Elements: Brackets [], Angle Brackets <>, Braces {} -

[ ] Brackets enclose optional parameters. You may use one or more of the optional parameters.

<> Angle brackets must enclose function-dependent parameters if the QIO$ requires the parameters <p1,...,p6>. The angle brackets are part of the syntax and must be used. The parameters may or may not be present in a given QIO$ macro and, if present, some may be optional.

{} Braces indicate that you must make a choice among the arguments enclosed within the braces.

The following paragraphs summarize the use of each QIO$ parameter. Section 1.7 explains different forms of the QIO$ macro itself.

1.4.1.2 FNC Parameter -

The **fnc** parameter is the symbolic name of the I/O function that you want to request. This name is usually of the form

```
IO.xxx
```

where **xxx** identifies the particular I/O operation.

For example, a QIO$ request to attach the physical device unit associated with a LUN specifies the function code IO.ATT with its complete QIO$ form appearing as

```
QIO$  IO.ATT,lun
```

where **lun** is the number assigned to the physical device unit.

A QIO$ request to cancel (or kill) all I/O requests for a LUN that you specified begins like this:

```
QIO$   IO.KIL,...
```

The system internally stores the fnc parameter, which you specify in the QIO$ request, as a function code in the high-order byte and as modifier bits in the low-order byte of a single word. The function code is in the range 0 through 31. (decimal) and is a binary value that the system supplies to match the symbolic name specified in the QIO$ request.

The system object module library defines the correspondence between global symbolic names and function codes. The Task Builder searches the library. You can obtain local symbolic definitions by the FILIO$ and SPCIO$ macros, which reside in the System Macro Library and are summarized in Appendix A.

Several similar functions may have identical function codes, and you may distinguish them only by their modifier bits. For example, the DECtape read logical forward and read logical reverse functions have the same function code. Although the function codes are the same, the system stores the modifier bits for these two operations.

1.4.1.3 **LUN Parameter** - The **lun** parameter represents the logical unit number (LUN) of the associated physical device unit that the I/O request is to access. The association between the physical device unit and the LUN is specific to the task that issues the I/O request, and the LUN reference is usually device independent. You begin an attach request to the physical device unit associated with LUN 14 like this:

```
QIO$   IO.ATT,14.,...
```

Because each task has its own LUT in which the correspondence between the LUN and the physical device unit is established, the legality of a LUN parameter is specific to the task that includes this parameter in a QIO$ request. In general, the LUN must be in the following range:

```
0  <LUN  <number of LUTs in table(if nonzero)/4
                                   ;each LUT is 2 words, 4 bytes
```

The number of LUNs specified in the LUT of a particular task cannot exceed 255.

1.4.1.4 **EFN Parameter** - The **efn** parameter is a number representing the event flag to be associated with the I/O operation. It is an optional parameter for inclusion in the QIO$ request. The specified event flag is cleared when the I/O request is queued and is set when the I/O operation has completed. If the task issued the QIOW$ macro, the Executive suspends task execution until the I/O completes. If the task issued the QIO$ macro (with no WTSE$ macro), task execution proceeds in parallel with the I/O. When the task continues to execute, it may test the event flag whenever it chooses by using the Read All Event Flags (RDAF$) Executive directive macro (if group-global event flags are not being used), the Read Extended Flags (RDXF$) Executive directive (for all event flags, including group-global event flags), or the Read Single Event Flag (RSEF$) Executive directive.

If you specify an event flag number, it must be in the range 1 through 96. If you do not want to specify an event flag, you can omit efn or supply it with a value of 0. Event flags 1 through 32 are local (specific to the issuing task); event flags 33 through 64 are global (shared by all tasks in the system). Event flags 65 through 96 are group-global event flags (shared by all tasks in the same user group). Flags 25 through 32 and 57 through 64 are reserved for use by system

software. Within these bounds, you can specify event flags as desired to synchronize I/O completion and task execution. Sections 1.4.2 and 1.4.3 provide a more detailed explanation of event flags and significant events.

NOTE

If an event flag is not specified, the Executive treats the directive as if it were a simple QIO$ request.

1.4.1.5 **PRI Parameter** - The optional **pri** parameter is supplied only to make RSX-11M/M-PLUS QIO$ requests compatible with RSX-11D. An RSX-11M I/O request assumes the priority of the requesting task. Thus, you should use a value of 0 (or a null) for this parameter.

1.4.1.6 **ISB Parameter** - The optional **isb** parameter identifies the address of the I/O status block associated with the I/O request. This block is a 2-word array in which a code is returned that represents the final status of the I/O request on completion of the operation. This code is a binary value corresponding to a symbolic name of the form IS.xxx (for successful returns) or IE.xxx (for error returns). The binary error code is returned to the low-order byte of the first word of the status block. It can be tested symbolically, by name. For example, the symbolic status IE.BAD is returned if a bad parameter is encountered. The following illustrates the examination of the I/O status block, IOSB, to determine whether a bad parameter has been detected:

```
QIO$    IO.ATT,14.,2,,IOSB
BCS     DIRERR
WTSE$C  2
        .
        .
        .
CMPB    #IS.SUC,IOSB
BNE     ERROR
```

The system object module library defines the correspondence between global symbolic names and I/O completion codes. The Task Builder searches this library. The IOERR$ macro, which resides in the System Macro Library, obtains local symbolic definitions (summarized in Appendix B).

On completion of the I/O operation, the system returns certain device-dependent information to the high-order byte of the first word of isb. If a read or write operation is successful, the second word is also significant. For example, in the case of a read function on a terminal, the system returns in the second word of isb the number of bytes that you typed preceding a carriage return. If a magnetic tape unit is the device and you specified a write function, this number represents the number of bytes actually written. The status block can be omitted from a QIO$ request if you do not intend to test for successful completion of the request.

1.4.1.7 **AST Parameter** - The optional **ast** parameter specifies the address of a service routine to be entered when an asynchronous system trap occurs. If you want to interrupt your task to execute special code on completion of an I/O request, you can specify an asynchronous system trap routine in the QIO$ request. When the specified I/O operation completes, control branches to this routine at the software priority of the requesting task. The system then executes the asynchronous code beginning at address ast, much like the way the system executes an interrupt service routine. If you do not want to perform asynchronous processing, you can omit the ast parameter or specify a value of 0 in the QIO$ macro call.

Section 1.4.5 discusses the use of asynchronous system traps, and the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual describes traps in detail.

1.4.1.8 **P1,P2,...,P6 Parameters** - The additional QIO$ parameters <p1,p2,...,p6> depend on the particular function and device specified in the I/O request. Typical parameters may include I/O buffer address, I/O buffer length, and so on. You can include between zero and six parameters depending on the particular I/O function. Subsequent chapters of this manual describe rules for including these parameters and legal values.

1.4.2 Significant Events

A significant event is a change in system status that causes the Executive to reevaluate the eligibility of all active tasks to run. (For some significant events, specifically those in which the current task becomes ineligible to run, only those tasks of lower priority are examined.) A significant event is usually caused (either directly or indirectly) by an Executive directive issued from within a task. This manual is concerned with the significant event caused by an I/O completion.

Significant events are normally set by Executive directives by completing a function that you specified. A task uses event flags to recognize the occurrence of specific events.

1.4.3 Event Flags

Event flags are a means by which tasks recognize specific events. (Tasks also use Asynchronous System Traps (ASTs) to recognize specific events.) In requesting a system operation (such as an I/O transfer), a task may associate en event flag with the completion of the operation. When the event occurs, the Executive sets the specified flag.

Ninety-six event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding unique event flag number (efn). Numbers 1 through 32 form a group of flags that are unique to each task and are set or cleared as a result of that task's operation. Numbers 33 through 64 form a second group of flags that are common to all tasks, hence their name "common flags." Common flags may be set or cleared as a result of any task's operation. The last eight flags in each group, local flags (25-32) and common flags (57-64), are reserved for use by the system. Numbers 65 through 96 form the third group of flags, known as "group global event flags." You can use these flags in any application where common event flags can be used; however, only tasks running under UICs containing the group code specified when the group-global event flags

were created can use them. Eight Executive directives provide the support for creating, setting, clearing, reading, and testing event flags. See the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for a description of these directives.

The following example illustrates the use of a common event flag to synchronize task execution.

A task issues a QIO$ macro with an efn parameter specified. A WTSE$ macro follows the QIO$ and specifies the same event flag number as an argument. The Executive clears the event flag when the Executive queues the I/O request. Then, the Executive blocks the task when the Executive executes the WTSE$ directive. The task remains blocked until a significant event is declared at the completion of the I/O request and the significant event sets the event flag. The task resumes when the appropriate event flag is set, and execution resumes at the instruction following the WTSE$ macro. Using these macros and an event flag in this way ensures that the task does not manipulate the data until all the I/O has completed.

Specifying an event flag does not mean that a WTSE$ macro must be issued. Event flag testing can be performed at any time. The purpose of a WTSE$ macro is to block the task execution until an indicated event occurs. Hence, it is not necessary to issue a WTSE$ macro immediately following a QIO$ macro, but a task that depends on a specific I/O operation to complete must issue it before continuing.

A task can issue a Stop For Single Event Flag (STSE$) macro instead of a WTSE$ macro. When this is done, an event flag condition not satisfied results in the task's being stopped instead of being blocked until the event flag is set. A blocked task still competes for memory resources at its running priority. A stopped task competes for memory resources at priority 0.

1.4.4 System Traps

System traps can interrupt task execution and cause a transfer of control to another memory location for special processing. The Executive handles system traps. The traps are relevant only to the task in which they occur. To use a system trap, a task must contain a trap service routine, which is automatically entered when the trap occurs.

There are two types of system traps: synchronous and asynchronous. You can use both to handle error or event conditions, but they differ in their relation to the task that is running when the traps are detected. The traps differ as follows:

- Synchronous system traps (SSTs) signal error conditions within the executing task. If the same instruction sequence were repeated, the same synchronous trap would occur at the same place in the task. Synchronous traps are fully described in the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual.

- Asynchronous system traps (ASTs) signal the completion of an external event such as an I/O operation. An asynchronous system trap (AST) usually occurs as the result of initiating or completing an external event rather than as a program condition.

Although not able to distinguish execution of an SST routine from task execution, the Executive is aware that a task is executing an AST routine. An AST routine can be interrupted by an SST routine, but not by another AST routine.

1.4.5 Asynchronous System Traps

The primary purpose of an AST is to inform the task that a certain event has occurred -- for example, the completion of an I/O operation. As soon as the task has serviced the event, it can return to the interrupted code.

Some directives can specify both an event flag and an AST; with these directives, you can use ASTs as an alternative to event flags or you can use the two together. Therefore, you can specify the same AST routine for several directives, each with a different event flag. Thus, when the Executive passes control to the AST routine, the event flag can determine the action required. However, it is standard programming practice to use the I/O Status Block (IOSB) rather than the event flags to determine which I/O operation is completed. Thus, when control is passed to an AST from a QIO$, the I/O Status Block (IOSB) is on top of the stack. Use this IOSB to determine which I/O has completed.

The Executive queues ASTs in a first-in-first-out queue for each task and monitors all asynchronous service routine operations. Because asynchronous traps may be the end result of I/O-related activity, the task cannot control the occurrence of the ASTs directly. An example of an asynchronous trap condition is the completion of an I/O request. The timing of such an operation clearly cannot be predicted by the requesting task. If the task does not specify an AST service routine in an I/O request, a trap does not occur and normal task execution continues.

However, the task may, under certain circumstances, block recognition of ASTs to prevent simultaneous access to a critical data region. When access to the critical data region has been completed, the queued ASTs may again be honored. The Disable AST Recognition (DSAR$S) and Enable AST Recognition (ENAR$S) Executive directives provide the mechanism for doing this.

Associating asynchronous system traps with I/O requests enables the requesting task to be truly event driven. The system executes the AST service routine contained in the initiating task as soon as possible, consistent with the task's priority. Using the AST routine to service I/O-related events provides a response time that is considerably better than a polling mechanism, and provides for better overlap processing than the simple QIO$ and WTSE$ macros. Asynchronous system traps also provide an ideal mechanism for use in multiple buffering of I/O operations.

The Executive inserts all ASTs in a first-in-first-out queue on a per task basis as they occur (that is, the event that they are to signal has expired). The Executive executes them one at a time whenever the task does not have ASTs disabled and is not already in the process of executing an AST service routine. Executing the AST includes storing certain information on the task's stack, including the task's WTSE$ mask word and address, the Directive Status Word (DSW), the program status (SP), the program counter (PC), and any trap-dependent parameters. The task's general-purpose registers R0-R5 are not saved, and thus AST service routines must save and restore all registers used. If the registers are not restored after an AST has occurred, the task's subsequent execution may be unpredictable.

After an AST is processed, the trap-dependent parameters (if any) must be removed from the task's stack and an AST Service Exit ASTX$S macro executed. The ASTX$S macro, described in Section 1.6.7 of this manual, issues the AST Service Exit directive. On AST service exit, control returns to another queued AST, to the executing task, or to another task waiting to run.

The RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual describes in detail the purpose of AST service routines and all Executive directives that handle them.

1.5 DIRECTIVE PARAMETER BLOCKS

A Directive Parameter Block (DPB) is a fixed-length area of contiguous memory that contains the arguments that you specify in an Executive directive macro call. The DPB for a QIO$ directive has a length of 12 words. The Executive generates it as the result of expanding a QIO$ macro call. The first two bytes of the DPB contain the following:

- The first byte of the DPB contains the directive identification code (DIC) -- always 1 for QIO$.
- The second byte contains the size of the DPB in words -- always 12 for RSX-11M/M-PLUS.

During the assembly of your task containing QIO$ requests, the MACRO-11 Assembler generates a DPB for each I/O request specified in a QIO$ macro call. At run time, the Executive uses the arguments stored in each DPB to create, for each request, an I/O packet in system dynamic storage. Figure 1-2 illustrates the layout of a sample DPB.

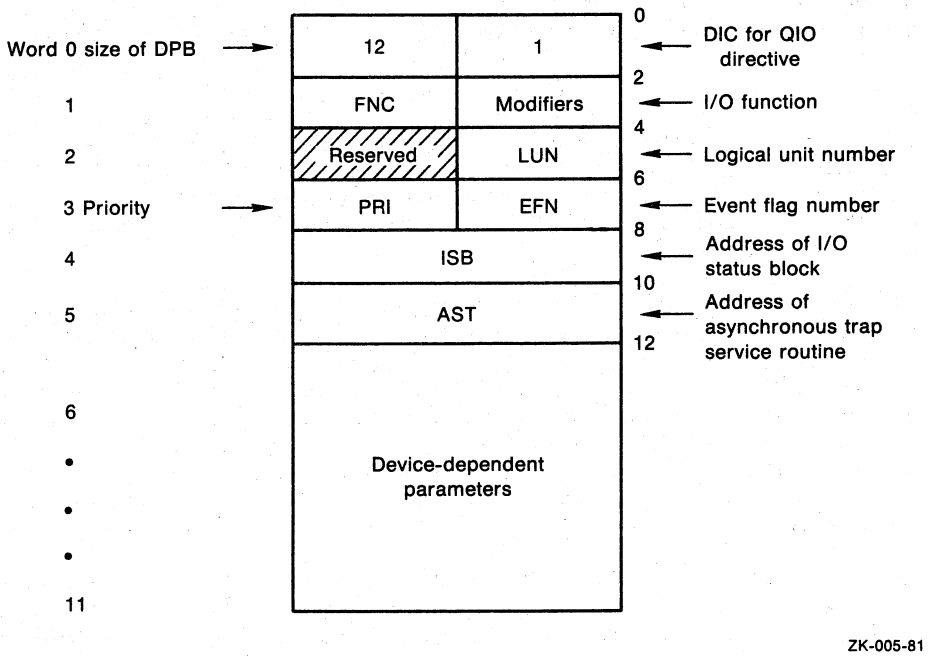


Figure 1-2 QIO$ Directive Parameter Block

1.5.1 I/O Packets

The Executive enters the I/O packet by priority into a queue of I/O requests for the specified physical device unit. The Executive creates and maintains this queue and orders it by the priority of the tasks that issued the requests. The I/O drivers examine their respective I/O packet queues for the I/O request with the highest priority capable of being executed. The driver removes this packet from the queue and performs the I/O operation. The process is then repeated until the queue is empty of all requests.

1.5.2 Significant Event Declaration

After the I/O request has been completed, the Executive declares a significant event and may do one or more of the following:

- Set an event flag.
- Cause a branch to an asynchronous system trap service routine.
- Return the I/O status.

Any of the above actions depend on the arguments specified in the original QIO$ macro call.

1.6 I/O RELATED MACROS

The RSX-11M/M-PLUS system supplies several system macros to issue and return information about I/O requests. These macros reside in the System Macro Library and must be made available during assembly by including the MACRO-11 Assembler directive .MCALL in the task's code.

The RSX-11M/M-PLUS system also supplies FORTRAN-callable subroutines that perform the same functions as the system macros. See the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for details.

Most of the Executive directive macros described in this section have three distinct forms. The following list summarizes the forms of QIO$, but the characteristics of each form also apply to QIOW$, ALUN$, GLUN$, and the other described Executive directive macros.

1. QIO$ (executed by using the DIR$ macro) generates a directive parameter block for the I/O request at assembly time, but does not provide the instructions necessary to execute the request. The QIO$ form is useful under the following conditions:
   - The task uses the DPB in several different places in the task.
   - The task modifies the DPB at run time.
   - The task references the DPB at run time.
2. QIO$S generates a directive parameter block for the I/O request on the stack, and also generates code to execute the request. This is a useful form for reentrant, shareable code because QIO$S generates the DPB dynamically at execution time.

3. QIO$C generates a directive parameter block for the I/O request at assembly time as well as generating code to execute the request. QIO$C generates the DPB in a separate program section called $DPB$$. QIO$C incurs little system overhead and it is useful when the task executes an I/O request from only one location. This manual uses the C form of the QIO$ macro in most of the examples in Chapter 1.

Parameters for both the QIO$ and QIO$C forms of the macro must be valid expressions for the MACRO-11 .WORD and .BYTE statements. Parameters for the QIO$S form must be valid source operand address expressions for Assembler instructions such as MOV and MOVB. The following example references the same parameters in the three distinct forms of the macro call.

```
QIO$     IO.RLB,6,2,,,AST01,<RDBUF,80.>

QIO$C    IO.RLB,6,2,,,AST01,<RDBUF,80.>

QIO$S    #IO.RLB,#6,#2,,,#AST01,<#RDBUF,#80.>
```

Only the QIO$S form of the macro produces the DPB dynamically. The other two forms generate the DPB at assembly time. The RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual describes the characteristics and use of these different forms.

The following section describes Executive directives and Assembler macros:

1. QIO$, which requests an I/O operation and supplies parameters for that request
2. QIOW$, which is equivalent to QIO$ followed by WTSE$
3. DIR$, which specifies the address of a directive parameter block as its argument, and generates code to execute the directive
4. .MCALL, which makes all macros referenced during task assembly available from the System Macro Library
5. ALUN$, which associates a logical unit number with a physical device unit at run time
6. GLUN$, which requests that the information about a physical device unit to LUN association be returned to a buffer that you specify
7. ASTX$S, which terminates execution of an asynchronous system trap (AST) service routine
8. WTSE$, which instructs the system to block execution of the issuing task until a specified event flag is set

1.6.1 The QIO$ Macro: Issuing an I/O Request

As previously described, you may use three general forms of the QIO$ macro. They are reviewed as follows:

- QIO$ generates only the DPB for the I/O request. This form of the macro call is used with DIR$ (see Section 1.6.3) to execute an I/O request.

- QIO$S generates a DPB for the I/O request on the stack as well as generating code to execute the request.
- QIO$C generates a DPB and code, but the DPB is generated in a separate program section.

1.6.2 The QIOW$ Macro: Issuing an I/O Request and Waiting for an Event Flag

The QIOW$ macro is equivalent to a QIO$ followed by a WTSE$. It [Opis more economical to issue a QIOW$ request than to use the two separate macros. An event flag (efn parameter) must be specified with QIOW$.

NOTE

Please note that tasks or applications that execute many I/O operations will run much more efficiently using QIOW$ rather than QIO$ followed by a WTSE$. The reason efficiency increases is that system overhead is reduced.

The QIOW$ macro has the following syntax:

```
QIOW$ function,lun,efn,[pri],[isb],[ast],[<p1,...,p6>]
```

See the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for a complete description of the QIOW$ macro.

1.6.3 The DIR$ Macro: Executing a Directive

The DIR$ (execute directive) macro allows a task to reference a previously defined DPB. Issue it in the form:

```
DIR$   [addr],[err]
```

The parameters have the following meanings:

Parameter	Meaning
addr	The address of a directive parameter block used in the directive. If addr is not included, the DPB itself or the address of the DPB is assumed to already be on the stack. This parameter must be a valid source operand for a MOV instruction generated by the DIR$ macro.
err	An optional argument which specifies the address of an error routine to which control branches if the directive is rejected. The branch occurs by means of a JSR PC, err if the C-bit is set, indicating rejection of the QIO$ directive.

In the following example, the DIR$ macro actually generates the code to execute the QIO$ directive. It provides no QIO$ parameters of its own, but references the QIO$ directive parameter block at address QIOREF by supplying this label as an argument.

```
QIOREF:  QIO$    IO.RLB,6,2,,,AST01,<BUFFER,80.> ;CREATE QIO$ DPB
         .
         .
         .
READ1:   DIR$    #QIOREF                  ; ISSUE I/O REQUEST
         .
         .
         .
READ2:   DIR$    #QIOREF                  ; ISSUE I/O REQUEST
```

1.6.4 The .MCALL Directive: Retrieving System Macros

.MCALL is a MACRO-11 Assembler directive that retrieves macros from the System Macro Library (LB:[1,1]RSXMAC.SML) for use during assembly. You must include it in every task that invokes system macros. .MCALL is usually placed at the beginning of your task source module and specifies, as arguments in the call, all system macros that must be made available to your task from the library.

The following example illustrates the use of this directive:

```
         .MCALL  QIO$,QIO$S,DIR$,WTSE$S     ; MAKE MACROS AVAILABLE
            .
            .
            .
ATTACH:  QIO$S   #IO.ATT,#6,,, IOSB,#AST02  ; ATTACH DEVICE
            .
            .
            .
QIOREF:  QIO$    IO.RLB,6,,,IOSB,AST01,...  ; CREATE ONLY QIO$ DPB
            .
            .
            .
READ1:   DIR$    #QIOREF,DIRERR             ; ISSUE I/O REQUEST
            .
            .
            .
```

You can include as many macro references as can fit on a line in a single .MCALL directive. You can specify any number of .MCALL directives.

1.6.5 The ALUN$ Macro: Assigning a LUN

The Assign LUN macro associates a logical unit number with a physical device unit at run time. All three forms of the macro call may be used. Assign LUN does not request I/O for the physical device unit, nor does it attach the unit for exclusive use by the issuing task. It only establishes a LUN-physical device unit relationship, so that when the task requests I/O for that particular LUN, the task can reference the associated physical device unit. Issue the macro from a MACRO-11 program in the following way:

```
ALUN$   lun,dev,unt
```

Parameter	Meaning
lun	The logical unit number to be associated with the specified physical device unit. See Sections 1.3 and 1.4.1.3.
dev	The device name of the physical device or a logical device name assigned to a physical device (see the MCR ASN command or the DCL ASSIGN command).
unt	The unit number of that device specified above.

For example, to associate LUN 10. with terminal unit 2, a task could issue the following macro call:

```
ALUN$C  10.,TT,2
```

A unit number of 0 represents unit 0 for multiunit devices such as disk, DECtape, or terminals; it indicates the single available unit for devices without multiple units, such as card readers and line printers.

Logical devices are system generation options on RSX-11M that allow you to assign logical names to physical devices with the MCR command ASN or the DCL command ASSIGN. Logical devices are included as part of RSX-11M-PLUS.

See the RSX-11M/M-PLUS MCR Operations Manual or the RSX-11M/M-PLUS Command Language Manual for a full description of the ASN command.

The following example illustrates the use of the three forms of the ALUN$ macro.

```
;
; DATA DEFINITIONS
;

ASSIGN: ALUN$   10.,TT,2        ; GENERATE DPB
        .
        .
        .

;
; EXECUTABLE SECTION
;

        DIR$    #ASSIGN         ; EXECUTE DIRECTIVE
        .
        .
        .
        ALUN$C  10.,TT,2        ; GENERATE DPB IN SEPARATE PROGRAM
        .                       ; SECTION, THEN GENERATE CODE TO
        .                       ; EXECUTE THE DIRECTIVE
        .
        ALUN$S  #10.,#"TT,#2    ; GENERATE DPB ON STACK, THEN
                                ; EXECUTE DIRECTIVE
```

1.6.5.1 **Physical Device Names** - The following list contains physical device names, listed alphabetically, that you may include as dev parameters:

Name	Device
AD	AD01-D Analog-to-Digital Converter (not supported in RSX-11M-PLUS systems)
AF	AFC11 Analog-to-Digital Converter (not supported in RSX-11M-PLUS systems)
AR	AR11 Laboratory Peripheral System (not supported in RSX-11M-PLUS systems)
BS	DT03/DT07 UNIBUS Switch (supported in RSX-11M-PLUS systems only)
CD	CD11 Card Reader
CP	Central Processor Unit (CPU) in a multiprocessor system (supported in RSX-11M-PLUS systems only)
CR	CR11/CM11 Card Reader
CT	TA11/TU60 Tape Cassette
DB	RP04, RP05, RP06 Pack Disk
DD	TU58 DECtape II
DF	RF11/RS11 Fixed-Head Disk
DK	RK11/RK05 Cartridge Disk
DL	RL11/RL01/RL02 Cartridge Disk
DM	RK611/RK06 and RK711/RK07 Cartridge Disk
DP	RP11/RP02/RP03 Pack Disk
DR	RM02/RM03/RM05 Pack Disk and RM80/RP07 Fixed-Media Disk
DS	RS03 and RS04 Fixed-Head Disks
DT	TC11/TU56 DECtape
DU	RA80/RA81 Fixed-Media Disk, RA60 Pack Disk, RC25 Disk Subsystem, RD51 Fixed-Media Disk, RD52 Fixed-Media Disk, RUX50 UNIBUS interface, and RX50 Flexible Disk
DX	RX11/RX01 Flexible Disk
DY	RX211/RX02 Flexible Disk
EM	ML-11 Fast Electronic Mass Storage Device
GR	VT11/VS60 Graphics Systems (not supported in RSX-11M-PLUS systems)
IC	ICS/ICR Industrial Control Local and Remote Subsystems (not supported in RSX-11M-PLUS systems)

Name	Device
IS	DSS/DRS Digital Input and Output Subsystems (not supported in RSX-11M-PLUS systems)
LA	LPA11-K Laboratory Peripheral Accelerator
LP	LA180/LP11/LS11/LV11 Line Printers and LN01/LN03 Laser Printer, KMC-11-A Auxiliary Processor
LR	PCL11-A/PCL11-B Receiver Port
LS	LPS11 Laboratory Peripheral System (not supported in RSX-11M-PLUS systems)
LT	PCL11-A/PCL11-B Transmitter Port
MM	TU16/TE16/TU45/TU77/TM02/TM03 Magnetic Tape
MS	TS11, TU80, TSV05, or TK25 Magnetic Tape
MT	TM11/TU10/TU11 or TS03 Magnetic Tape
MU	TK50/TU81 Cartridge Tape
NL	The Null Device
PP	PC11 Paper Tape Punch
PR	PC11 or PR11 Paper Tape Reader
TT	Terminals (regardless of interface) (not Network Command Terminals)
UD	UDC11 Universal Digital Controller (not supported in RSX-11M-PLUS systems)
XB	DA11-B Parallel Unibus Link (not supported in RSX-11M-PLUS systems)
XE	QIO DEUNA Driver
XL	DL11-E Asynchronous Communication Line Interface (not supported in RSX-11M-PLUS systems)
XM	DMC11 Synchronous Communication Line Interface
XP	DP11 Synchronous Communication Line Interface (not supported in RSX-11M-PLUS systems)
XQ	DQ11 Synchronous Communication Line Interface (not supported in RSX-11M-PLUS systems)
XU	DU11 Synchronous Communication Line Interface (not supported in RSX-11M-PLUS systems)
XW	DUP11 Synchronous Communication Line Interface
JA-JZ	Reserved for customer use (not used by DIGITAL)
QA-QZ	Reserved for customer use (not used by DIGITAL)
ZA-ZZ	Reserved for customer use (not used by DIGITAL)

1.6.5.2 **Pseudo-Device and Physical Device Names** - A pseudo-device name is a logical device name that must be directed to a physical device unit. A pseudo device name can be redirected, by the operator, to another physical device at any time without requiring changes in programs that reference the pseudo-device name. (The DV.PSE bit in the LUN information buffer is set to one if a pseudo name references a physical device.) Dynamic redirection of a physical device unit affects all tasks MCR REDIRECT command affects only one task.

Nonphysical device names are not associated with a physical device but with a driver that interfaces with data structures instead of a real physical device.

The following list indicates the pseudo devices supported by RSX-11M/M-PLUS:

Nonphysical Name	Physical Name	Driver	Unit
	CL (pseudo)		Console listing, normally the line printer.
	CO (pseudo)	CODRV	Console output, normally the main operator's console.
HT		HTDRV	Network remote terminal.
	LB (pseudo)		System library device, normally the device from which the system was bootstrapped. For example, tasks such as TKB and MAC access the LB: device for default library files.
NL		NLDRV	Null device.
NS			Network pseudo device for NSP.
NX			Network pseudo device for DLX.
RD		RDDRV	On-line reconfiguration pseudo device (RSX-11M-PLUS only).
RT		RTDRV	Network Command Terminals.
	SP (pseudo)		Spooling scratch disk device (RSX-11M-PLUS and Micro/RSX only).
	SY (pseudo)		Your system default device. On nonmultiuser systems, SY: is normally the disk from which the system was bootstrapped. On multiuser systems, SY: is normally the default login device.

Nonphysical Name	Physical Name	Driver	Unit
	TI (pseudo)		Pseudo input terminal; TI0: is the terminal from which a task was requested. The pseudo device TI cannot be redirected, because such redirection would have to be handled on a per-task rather than a systemwide basis (that is, you can change the TI device for one task without affecting the TI assignments for other tasks).
VT		VTDRV	Virtual terminal. Used by some RSX-11M-PLUS offspring tasks as TI: for command and data I/O (RSX-11M-PLUS and Micro/RSX only).

1.6.6 The GLUN$ Macro: Retrieving LUN Information

The Get LUN Information macro requests the return of information about association between a LUN and physical device unit in a 6-word buffer specified by the issuing task. Upon successful completion of a QIO$ directive, the buffer contains the information listed in Table 1-1, as appropriate for the specific device. All three forms of the macro call may be used. It is issued from a MACRO-11 program in the following way:

```
GLUN$   lun,buf
```

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number associated with the physical device unit for which information is requested. See Sections 1.3 and 1.4.1.3.
buf	The 6-word buffer to which information is returned.

For example, to request information on the disk unit associated with LUN 8, issue the following call:

```
GLUN$C   8.,IOBUF
```

Table 1-1
Get LUN Information

Numerical Offset			Symbolic Offset			Contents
Word	Byte	Bit	Word	Byte	Bit	
0			G.LUNA			Name of device associated with LUN (ASCII bytes)
1	0			G.LUNU		Unit number of associated device

(continued on next page)

Table 1-1 (Cont.)
Get LUN Information

Numerical Offset			Symbolic Offset			Contents
Word	Byte	Bit	Word	Byte	Bit	
	1			G.LUFB		Driver flag value. Returned as 128. (decimal) or 200 (octal) if the driver is resident, or as 0 if a loadable driver is not in the system
2			G.LUCW [1]			First device characteristics word:
		0	(U.CW1)		(DV.REC)	Unit record-oriented device (for example, card reader, line printer) (1 = yes)
		1			(DV.CCL)	Carriage-control device (for example, line printer, terminal) (1 = yes)
		2			(DV.TTY)	Terminal device (1 = yes)
		3			(DV.DIR)	Directory device (for example, DECtape, disk) (1 = yes)
		4			(DV.SDI)	Single directory device (for example, ANSI-standard magnetic tape) (1 = yes)
		5			(DV.SQD)	Sequential device (for example, ANSI-standard magnetic tape) (1 = yes)
		6			(DV.MSD)	Mass storage device (for example, disks and tapes) (1 = yes)
		7			(DV.UMD)	User-mode diagnostics supported (1 = yes)
		8			(DV.EXT)	Device supports 22-bit direct addressing

1. The following word and bit symbols shown in parentheses are used in defining and referencing corresponding items in the device UCB.

(continued on next page)

Table 1-1 (Cont.)
Get LUN Information

Numerical Offset			Symbolic Offset			Contents
Word	Byte	Bit	Word	Byte	Bit	
		9			(DV.SWL)	Unit software write-locked (1 = yes)
		10			(DV.ISP)	Input spooled device (1 = yes)
		11			(DV.OSP)	Output spooled device (1 = yes)
		12			(DV.PSE)	Pseudo device (1 = yes)
		13			(DV.COM)	Device mountable as a communications channel for Digital network support (for example, DP11, DU11) (1 = yes)
		14			(DV.F11)	Device mountable as a Files-11 device (for example, disk or DECtape) (1 = yes)
		15			(DV.MNT)	Device mountable (logical OR of bits 13 and 14) (1 = yes)
3			G.LUCW+02			Second device characteristics word:
			(U.CW2)	(U2.xxx)		Device-specific information
4			G.LUCW+04			Third device characteristics word:
			(U.CW3)	(U3.xxx)		Device-specific information [2]
5			G.LUCW+06			Fourth device characteristics word:
			(U.CW4)			Default buffer size (for example, for disks, and line length for terminals).

2. For mass storage devices, such as disks, DECtape, and DECtape II, this is the number of blocks (maximum logical block number plus one). For the proper use of the RX211/RX02 flexible disk, you must test G.LUCW+4 to determine the media density.

The following example illustrates the use of the three forms of the GLUN$ macro.

```
;
; DATA DEFINITIONS
;
GETLUN: GLUN$   6,DSKBUF        ; GENERATE DPB
        .
        .
        .
;
; EXECUTABLE SECTION
;
        DIR$    #GETLUN         ; EXECUTE DIRECTIVE
        .
        .
        .
        GLUN$C  6,DSKBUF        ; GENERATE DPB IN SEPARATE PROGRAM
        .                       ; SECTION, THEN GENERATE CODE TO
        .                       ; EXECUTE THE DIRECTIVE
        .
        GLUN$S  #6,#DSKBUF      ; GENERATE DPB ON STACK, THEN
                                ; EXECUTE DIRECTIVE
```

1.6.7 The ASTX$S Macro: Terminating AST Service

The ASTX$S macro terminates execution of an AST service routine. The Executive provides all forms of the macro. However, the S-form requires less space and executes at least as fast as the ASTX$ or ASTX$C form of the macro. Issue it as follows:

```
ASTX$S  [err]
```

The parameter has the following meaning:

Parameter	Meaning
err	An optional argument specifying the address of an error routine to which control branches if the directive is rejected.

After the Executive completes the operation specified in this macro call, the Executive executes the next AST immediately if another AST is queued and asynchronous system traps have not been disabled. Otherwise, the Executive restores the task's state existing before the AST was entered. (The AST service routine must save and restore the registers it uses.)

1.6.8 The WTSE$ Macro: Wait for Single Event Flag

The WTSE$ macro suspends execution of the issuing task until the Executive sets the event flag specified in the macro call. This macro is extremely useful in synchronizing other activity with the completion of I/O operations. You may use all three forms of the macro call. Issue it as follows:

```
WTSE$   efn
```

The parameter has the following meaning:

Parameter	Meaning
efn	The event flag number.

WTSE$ blocks the task from execution until the specified event flag is set. Frequently, you may include an efn parameter in a QIO$ macro call, and the Executive sets the event flag upon the completion of the I/O operation specified in that call. The following example illustrates task blocking until the specified event flag is set. This example also shows using three forms of the macro call.

```
;
        .MCALL   WTSE$, ALUN$S, QIO$C, DIR$
        .MCALL   QIO$S, WTSE$S, WTSE$C

; DATA DEFINITIONS
;

WAIT:   WTSE$    5              ; GENERATE DPB
IOSB:   .BLKW    2              ; I/O STATUS BLOCK
        .
        .
        .
;
; EXECUTABLE SECTION
;
        ALUN$S   #14.,#"MM      ; ASSIGN LUN 14 TO MAGNETIC
                                ; TAPE UNIT ZERO
        QIO$C    IO.ATT,14.,5   ; ATTACH DEVICE
        DIR$     #WAIT          ; EXECUTE WAIT FOR DIRECTIVE
        .
        .
        .
        QIO$S    #IO.RLB,#14.,#2,,#IOSB,,<#BUF,#80.>
        .                       ; READ RECORD, USE EFN2
        .
        .
        .
        WTSE$S   #2             ; WAIT FOR READ TO COMPLETE
        .
        .
        .
        QIO$C    IO.WLB,14.,3,,IOSB,,<BUF,80.>
        .                       ; WRITE RECORD, USE EFN3
        .
        .
        .
        WTSE$C   3              ; WAIT FOR WRITE TO COMPLETE
        .
        .
        .
        QIO$C    IO.DET,14.     ; DETACH DEVICE
        .
        .
        .
```

1.7 STANDARD I/O FUNCTIONS

You can specify a large number of input/output operations with the QIO$ macro. You can request a particular operation by including the appropriate function code as the first parameter of a QIO$ macro call. Certain functions are standard. These functions are almost totally device independent and thus you can request them for nearly every device described in this manual. Other I/O functions are device

dependent and are specific to the operation of only one or two I/O devices. This section summarizes the function codes and characteristics of the following standard device-independent I/O operations:

- Attaching to an I/O device
- Detaching from an I/O device
- Canceling I/O requests
- Reading a logical block
- Reading a virtual block
- Writing a logical block
- Writing a virtual block

For certain physical device units, a standard I/O function may be described as being a NOP. This means that no operation occurs as a result of specifying the function, and the Executive returns an I/O status code of IS.SUC in the I/O status block specified in the QIO$ macro call.

1.7.1 I/O Subfunction Bits

Most terminal QIO$ functions can be modified by using the symbolic name of a subfunction bit in a Logical OR with the QIO$ function. The symbolic names of subfunction bits take the form TF.xxx, where xxx is the acronym of the subfunction to be performed. A standard QIO$ function called IO.ATT (attach a device) in a Logical in a Logical OR with the TF.ESQ subfunction for terminals (recognize escape sequences) would look like the following:

```
QIO$C IO.ATT!TF.ESQ,lun,[efn],[pri],[isb],[ast]
```

A subfunction bit modifies and extends the operation indicated by the terminal QIO$ function. Note that the use of TF.ESQ with IO.ATT is a terminal-specific function. Often, you may want to use more than one subfunction bit when you use QIO$ requests to read or write to a terminal. In this case, you may use several subfunction bits together in a Logical OR. The standard QIO$ IO.ATT function may be extended to both recognize escape sequences and allow special processing in the task upon the occurrence of asynchronous system traps. To do this requires that you combine in a Logical combine in a Logical OR two subfunction bits with the IO.ATT function. If you do this, the QIO$ IO.ATT macro would look like the following:

```
QIO$C IO.ATT!TF.ESQ!TF.AST,lun,[efn],[pri],[isb],[ast]
```

Note that the use of TF.ESC or TF.AST with IO.ATT is a terminal-specific function.

If your task invokes a subfunction bit that is not supported on the system, the subfunction bit may be ignored or an error may be issued by the system and the QIO$ rejected.

The subfunction bits that apply to a specific QIO$ macro are described with that QIO$ macro in the Chapter 2.

1.7.2 QIO$C IO.ATT - Attaching to an I/O Device

Use the IO.ATT function code when your task requires exclusive use of an I/O device. The QIO$ IO.ATT macro has the following format:

```
QIO$C IO.ATT,lun,[efn],[pri],[isb],[ast]
```

Successful completion of an IO.ATT request exclusively dedicates the specified physical device unit to the task that issues the IO.ATT. This enables the task to process input or output in an unbroken stream and is especially useful on sequential, non-file-oriented devices such as terminals, card readers, and line printers. An attached physical device unit remains under control of the task until that task explicitly detaches it. To detach the device, the task issues the QIO$C IO.DET macro with the LUN previously assigned to the attached device.

While a task attaches a physical device unit, the I/O driver for that unit dequeues only I/O requests issued by the task that attaches the unit. However, a privileged task can issue a write breakthrough function (IO.WBT) to a terminal attached by another task. This is an exception for terminals only. Thus, except for the case of IO.WBT, the Executive does not process a request to attach a device unit already attached by another task until the attachment by the first task is broken and no higher-priority request exists for the attached unit.

A LUN that is associated with an attached physical device unit may not be reassigned by an Assign LUN (ALUN$) macro unless at least one LUN is still assigned to the attached device. If the task that issued an attach function exits or is aborted before it issues a corresponding detach, the Executive detaches the physical device unit.

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.
pri	Makes RSX-11M/M-PLUS QIO$ requests compatible with RSX-11D. An RSX-11M request assumes the priority of the requesting task. Thus, a value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Section 1.4.1.6.
ast	For IO.ATT, ast specifies the address of a service routine to be entered when the IO.ATT operation completes. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5. See the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for further details on ASTs.

1.7.3 QIO$C IO.DET - Detaching from an I/O Device

IO.DET detaches a physical device unit that has been previously attached by an IO.ATT request. Issue the QIO$C IO.DET macro as follows:

```
QIO$C   IO.DET,lun,[efn],[pri],[isb],[ast]
```

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.
pri	Makes RSX-11M/M-PLUS QIO$ requests compatible with RSX-11D. An RSX-11M request assumes the priority of the requesting task. Thus, a value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Section 1.4.1.6.
ast	Specifies the address of a service routine to be entered when an asynchronous system trap occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.

The LUN specifications of both IO.ATT and IO.DET must be the same, as in the following example, which also illustrates using S-forms of several macro calls.

```
        .MCALL  ALUN$S,QIO$S
        ALUN$S  #14.,#"LP       ; ASSOCIATE LINE PRINTER WITH LUN 14
        .
        .
        .
        QIO$S   #IO.ATT,#14.    ; ATTACH LINE PRINTER
        .
        .
        .
LOOP:   QIO$S   #IO.RLB,#14.,... ; PRINT
        .
        .
        .
        QIO$S   #IO.DET,#14.    ; DETACH LINE PRINTER
```

1.7.4 QIO$C IO.KIL - Canceling I/O Requests

IO.KIL cancels the issuing task's I/O requests for a particular physical device unit.

For I/O requests waiting for service (that is, in the I/O driver's queue), the Executive returns a status code of IE.ABO in the I/O status block. An event flag is set, if specified. But any AST service routine that you may have specified is not executed.

For I/O requests being processed by any I/O driver, except the disk or DECtape drivers, the Executive returns the IE.ABO status code. The Executive also returns other status information (byte count, and so on) in the I/O status block. An AST, if specified, is executed.

If your task issues an IO.KIL for disk, DECtape, or DECtape II I/O requests being processed, the IO.KIL acts as a NOP. The I/O request completes, except in the case in which a DECtape transfer is blocked by a select error. Because disk and DECtape operate quickly, IO.KIL causes the return of IS.SUC in the I/O status block.

IO.KIL is useful in such special cases as canceling an I/O request on a physical device unit from which a response is overdue (for example, a read on a paper tape reader).

The QIO$C IO.KIL macro has the following syntax:

```
QIO$C   IO.KIL,lun,[efn],[pri],[isb],[ast]
```

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.
pri	Makes RSX-11M/M-PLUS QIO$ requests compatible with RSX-11D. An RSX-11M request assumes the priority of the requesting task. Thus, a value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Section 1.4.1.6.
ast	Specifies the address of a service routine to be entered when an asynchronous system trap occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.

1.7.5 QIO$C IO.RLB - Reading a Logical Block

Issue IO.RLB to read a block of data from the specified physical device unit. The QIO$C IO.RLB macro has the following format:

```
QIO$C   IO.RLB,lun,[efn ],<stadd,size,pn>
                   , pri
                   , isb
                   ,[ast]
```

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.
pri	Makes RSX-11M/M-PLUS QIO$ requests compatible with RSX-11D. An RSX-11M request assumes the priority of the requesting task. Thus, a value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Section 1.4.1.6.
ast	Specifies the address of a service routine to be entered when an asynchronous system trap occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.
stadd	The starting address of the data buffer. The address must be word aligned for certain drivers; otherwise, stadd may be on a byte boundary.
size	The size of the stadd buffer in bytes. The buffer must be within the task's address space.
pn	One to four optional parameters that specify such additional information as block numbers for certain devices.

1.7.6 QIO$C IO.RVB - Reading a Virtual Block

IO.RVB reads a virtual block of data from the specified physical device unit. A "virtual" block indicates a relative block position within a file and is identical to a logical block for such sequential, record-oriented devices as terminals and card readers. For these sequential, record-oriented devices, the Executive converts IO.RVB to IO.RLB before it issues the QIO$.

NOTE

Any subfunction bits specified in the IO.RVB request are stripped off in this conversion.

All tasks should use virtual rather than logical reads. However, if a task issues a virtual read for a file-structured device (disk, DECtape, or DECtape II), you must ensure that a file is open on the specified physical device unit. Issue IO.RVB as follows:

```
QIO$C   IO.RVB,lun,[efn] ,<stadd,size,pn>
                   ,[pri]
                   ,[isb]
                   ,[ast]
```

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.
pri	Makes RSX-11M/M-PLUS QIO$ requests compatible with RSX-11D. An RSX-11M request assumes the priority of the requesting task. Thus, a value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Section 1.4.1.6.
ast	Specifies the address of a service routine to be entered when an asynchronous system trap occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.
stadd	The starting address of the data buffer. The address must be word aligned for certain drivers; otherwise, stadd may be on a byte boundary.
size	The size of the stadd buffer in bytes. The buffer must be within the task's address space.
pn	One to four optional parameters that specify such additional information as block numbers for certain devices.

1.7.7 QIO$C IO.WLB - Writing a Logical Block

IO.WLB writes a block of data to the specified physical device unit.

If the write goes to a terminal, the Executive converts the IO.WVB to an IO.WLB request.

Note that any subfunction bits specified in the IO.WVB request (for example, TF.CCO, TF.WAL, or TF.WBT) are stripped when the IO.WVB is converted to an IO.WLB.

The QIO$C IO.WLB macro has the following format:

```
QIO$C   IO.WLB,lun,[efn] ,<stadd,size,pn>
                   ,[pri]
                   ,[isb]
                   ,[ast]
```

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.
pri	Makes RSX-11M/M-PLUS QIO$ requests compatible with RSX-11D. An RSX-11M request assumes the priority of the requesting task. Thus, a value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Section 1.4.1.6.
ast	Specifies the address of a service routine to be entered when an asynchronous system trap occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.
stadd	The starting address of the data buffer. The address must be word aligned for certain drivers; otherwise, stadd may be on a byte boundary.
size	The size of the stadd buffer in bytes. The buffer must be within the task's address space.
pn	One to four optional parameters that specify such additional information as block numbers or format control characters for certain devices.

1.7.8 QIO$C IO.WVB - Writing a Virtual Block

IO.WVB writes a virtual block of data to a physical device unit. A virtual block indicates a block position relative to the start of a file. For sequential, record-oriented devices such as terminals and line printers, the Executive converts IO.WVB to IO.WLB.

NOTE

Any subfunction bits specified in the IO.WVB request (see Sections 2.3.1 and 3.3.1) are stripped off in this conversion.

All tasks should use IO.WVB rather than IO.WLB to file-structured devices. However, if you issue a virtual write for a file-structured device (disk or DECtape II), you must ensure that a file is open on the specified physical device unit. For record-oriented devices, you should use IO.WLB.

Note that any subfunction bits specified in the IO.WVB request (for example, TF.CCO, TF.WAL, or TF.WBT) are stripped when the IO.WVB is converted to an IO.WLB.

The QIO$C IO.WVB macro has the following format:

```
QIO$C IO.WVB,lun,[efn] ,<stadd,size,pn>
                 , pri
                 , isb
                 , ast
```

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.
pri	Makes RSX-11M/M-PLUS QIO$ requests compatible with RSX-11D. An RSX-11M request assumes the priority of the requesting task. Thus, a value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Section 1.4.1.6.
ast	Specifies the address of a service routine to be entered when an asynchronous system trap occurs. If you want to interrupt your task to execute special code upon completion of this I/O request you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.
stadd	The starting address of the data buffer. The address must be word aligned for certain drivers; otherwise, stadd may be on a byte boundary.
size	The size of the stadd buffer in bytes. The buffer must be within the task's address space.
pn	One to four optional parameters that specify such additional information as block numbers or format control characters for certain devices.

1.8 USER-MODE DIAGNOSTIC FUNCTIONS

The I/O function code subfunction bit, IQ.UMD, provides support for user-mode diagnostics. You can execute standard I/O functions such as Read Logical Block, Write Logical Block, Attach to Device, and Detach from Device as user-mode diagnostics. To perform a diagnostic function, you must specify in the QIO$ directive parameter block the Logical OR of IQ.UMD and the function you want to perform. For example, to perform a diagnostic Read Logical Block operation, specify QIO$C IO.RLB!IQ.UMD,lun,... as the QIO$ directive.

Support for user-mode diagnostics is always present for RSX-11M-PLUS, but not all drivers support user-mode diagnostic functions. Unpredictable device and driver behavior results when you set the IQ.UMD subfunction bit in QIO$s that are directed to the device if it does not support user-mode diagnostics. You can avoid problems if you issue a Get LUN (GLUN$) macro and check the user-mode diagnostics bit before emitting the user-mode diagnostic QIO$.

For a device to support user-mode diagnostics, the DV.UMD bit in the UCB must be set. DV.UMD is at offset U.CW1 in the UCB.

In addition to standard I/O functions, RSX-11M-PLUS provides the following device-dependent, user-mode diagnostic functions:

1. Disk diagnostic functions
   - IO.HMS Home seek or recalibrate
   - IO.BLS Block seek (explicit seek)
   - IO.OFF Offset position
   - IO.RDH Read disk header
   - IO.WDH Write disk header
   - IO.WCK Writecheck
2. DECtape diagnostic functions
   - IO.RNF Read block number forward
   - IO.RNR Read block number reverse
3. Magnetic tape diagnostic functions
   - IO.LPC Read longitudinal parity character
   - IO.ERS Erase tape

UMDIO$ is the macro that defines these functions.
To execute a user-mode diagnostic function, you must first attach a device for diagnostics by using I/O function code IO.ATT!IQ.UMD. Execute the diagnostic functions and then detach the device.

The parameter list in words 1 through 6 of the DPB should contain the following information:

- I/O buffer address.
- I/O buffer size.

- Offset factor for disks with offset recovery. To determine the offset factor, refer to the offset register in the hardware reference manual; this parameter is not used if the device does not have offset recovery.
- Double-precision logical block number.
- Your task's register buffer address (the I/O driver copies its hardware registers to this buffer in your program); see a hardware reference manual for the length of the address.

A typical DPB for a diagnostic function might look like this:

```
$DSKPB::
        .BYTE   3,12.               ; Size of the DPB, QIOW
                                    ;   directive code
        .WORD   IO.WDH!IQ.UMD       ; I/O function code
        .WORD   THELUN              ; Logical Unit Number
        .BYTE   THEEFN,0            ; Event flag number
        .WORD   $IOSTS              ; I/O status block address
        .WORD   0                   ; AST address
$IOBUF:: .WORD  0                   ; Buffer address
        .WORD   0                   ; Transfer size in bytes
        .WORD   0                   ; Device dependent
$LBH::  .WORD   0                   ; High-order logical block number
$LBL::  .WORD   0                   ; Low-order logical block number
        .WORD   $RGBUF              ; Register buffer address
```

The user-mode diagnostic functions return either Success (IS.SUC) or Device Not Ready (IE.DNR). No other error codes are returned. All error recovery is completely up to you. Any errors that occur are not logged in the error log.

A typical program fragment, using the user-mode diagnostic functions, might look like the following:

```
        .MCALL UMDIO$,ALUN$S,QIO$S
UMDIO$                              ; Define diagnostic functions
ALUN$S  #14.,#"DM,#0                ; Associate DM0 with lun 14
.
.
.
QIO$S   #IO.ATT!IQ.UMD,#14.         ; Attach DM for diagnostic I/O
.
.
QIO$S   #IO.RDH!IQ.UMD,#14.,,,,,<#$IOBUF,#512.,,#LBH,#LBL,#$RGBUF>
        ; Read disk header
.
QIO$S   #IO.RLB!IQ.UMD,#14.,,,,,<#$IOBUF,#512.,,#LBH,#LBL,#$RGBUF>
        ; Read logical block
.
.
QIO$S   #IO.DET!IQ.UMD,#14.         ; Detach DM
.
.
.
```

1.9 I/O COMPLETION

When the system completes an I/O request, either successfully or unsuccessfully, the Executive selects return conditions depending upon the parameters included in the QIO$ macro call. There are three major returns:

- The Executive declares a significant event when an I/O operation completes execution. If you included an efn parameter in the I/O request, the corresponding event flag is set.
- If you included an isb parameter in the QIO$ macro call, the Executive returns a code identifying the type of success or failure. The code is in the low-order byte of the first word of the I/O status block at the location represented by isb.

  This status return code is of the form IS.xxx (success) or IE.xxx (error). For example, if the device accessed by the I/O request is not ready, a status code of IE.DNR is returned in isb. The section named Return Codes summarizes general codes returned by most of the drivers described in this manual.

  If the isb parameter was omitted, the requesting task cannot determine whether the I/O request was successfully completed. A carry clear return from the directive itself simply means that the directive was accepted and the I/O request was queued, not that the actual input/output operation was successfully performed.
- If you specified an ast parameter in the QIO$ macro call, a branch to the AST service routine beginning at the location identified by ast occurs when the I/O operation completes execution.

1.9.1 Return Codes

The Executive recognizes and handles two kinds of status conditions when they occur in I/O requests:

- Directive conditions, which indicate the acceptance or rejection of the QIO$ directive itself
- I/O status conditions, which indicate the success or failure of the I/O operation

Directive conditions relevant to I/O operations may indicate any of the following:

- Directive acceptance
- Invalid buffer specification
- Invalid efn parameter
- Invalid lun parameter
- Invalid DIC number or DPB size
- Unassigned LUN
- Insufficient memory

The Executive returns a code indicating the acceptance or rejection of a directive to the Directive Status Word at symbolic location $DSW. You can test this location to determine the type of directive condition.

I/O conditions indicate the success or failure of the I/O operation that you specified in the QIO$ macro. I/O driver errors include such conditions as device not ready, privilege violation, file already open, or write-locked device. If you include an isb parameter in the QIO$ directive, identifying the address of a two-word I/O status block, the Executive returns an I/O status code in the low-order byte of the first word of this block when an I/O operation completes execution. This code is a binary value corresponding to a symbolic name of the form IS.xxx or IE.xxx. You can test the low-order byte of the word symbolically, by name, to determine the type of status return. The system object module library defines the correspondence between global symbolic names and directive and I/O completion status codes. You may also obtain local symbolic definitions by the DRERR$ and IOERR$ macros, which reside in the System Macro Library and are summarized in Appendix B.

Binary values of status codes always have the following meanings:

Code	Meaning
Positive (greater than 0)	Successful completion
0	Operation still pending
Negative	Unsuccessful completion

A pending operation means that the I/O request is still in the queue of requests for the respective driver, or the driver has not yet completely serviced the request.

1.9.2 Directive Conditions

Table 1-2 summarizes the directive conditions that your task may encounter by issuing QIO$ directives. The table lists acceptance condition first, followed by error codes indicating various reasons for rejection.

Table 1-2
Directive Conditions

Code	Reason
IS.SUC	**Directive accepted** The first six parameters of the QIO$ directive were valid, and sufficient dynamic memory was available to allocate an I/O packet.
IE.ADP	**Invalid address** The I/O status block or the QIO$ DPB was outside of the issuing task's address space or was not aligned on a word boundary.

(continued on next page)

Table 1-2 (Cont.)
Directive Conditions

Code	Reason
IE.IEF	**Invalid event flag number** The efn specification in a QIO$ directive was less than 0 or greater than 96.
IE.ILU	**Invalid logical unit number** The lun specification in a QIO$ directive was invalid for the issuing task. For example, there were only 5 logical unit numbers associated with the task, and the value specified for lun was greater than 5.
IE.SDP	**Invalid DIC number or DPB size** The directive identification code (DIC) or the size of the Directive Parameter Block (DPB) was incorrect; the legal range for a DIC is from 1 through 127, and all DIC values must be odd. Each individual directive requires a DPB of a certain size. If the size is not correct for the particular directive, this code is returned. The size of the QIO$ DPB is always 12 words.
IE.ULN	**Unassigned LUN** The logical unit number in the QIO$ directive was not associated with a physical device unit. Your task may recover from this error by issuing a valid Assign LUN (ALUN$) directive and then reissuing the rejected directive.
IE.UPN	**Insufficient dynamic memory** There was not enough dynamic memory to allocate an I/O packet for the I/O request. You can try again later by blocking the task with a WTSE$ macro. Note that WTSE$ is the only effective way for the issuing task to block its execution, because other directives usable for this purpose require dynamic memory for their execution (for example, Mark Time (MRKT$)).

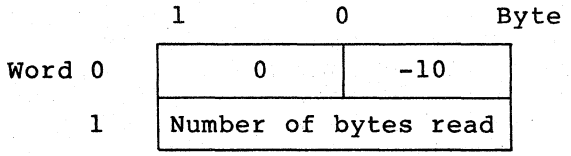
1.9.3 I/O Status Conditions

I/O status is returned in a 2-word I/O status block upon completion of the I/O operation. The status may show a successful completion or an error. The contents of the 2-word I/O status block is explained next:

- The low-order byte of the first word receives a status code of the form IS.xxx (success) or IE.xxx (error) when an I/O operation completes execution.
- The high-order byte of the first word is usually device dependent.
- The second word contains the number of bytes transferred or processed if the operation is successful and involves reading or writing.

If the isb parameter of the QIO$ directive is omitted, this information is not returned.

The following illustrates an example 2-word I/O status block on completion of a terminal read operation:



where -10 is the status code for IE.EOF (end of file). If this code is returned, it indicates that input was terminated by typing CTRL/Z, which is the end-of-file termination sequence on a terminal.

To test for a particular error condition, your task generally should compare the low-order byte of the first word of the I/O status block with a symbolic value, as in the following:

```
CMPB    #IE.DNR,IOSB
```

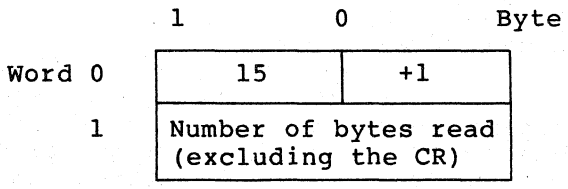
However, to test for certain types of successful completion of the I/O operation, the entire word value must be compared. For example, if a carriage return terminated a line of input from the terminal, a successful completion code of IS.CR is returned in the I/O status block. If an Escape (or Altmode) character was the terminator, a code of IS.ESC is returned. To check for these codes, your task should first test the low-order byte of the first word of the block for IS.SUC and then test the full word for IS.CC, IS.CR, IS.ESC, or IS.ESQ. (Other success codes that must be read in this manner are listed in Appendix B, Section B.1.2.)

Note that both of the following comparisons test as equal because the low-order byte in both cases is +1.

```
CMP     #IS.CR,IOSB

CMPB    #IS.SUC,IOSB
```

In the case of a successful completion where the carriage return is the terminal indicator (IS.CR), the following illustrates the status block:



where 15 is the octal code for carriage return and +1 is the status code for successful completion.

Table 1-3 summarizes status codes that may be returned in the I/O status block specified in the QIO$ directive on completion of the I/O request. The codes described in Table 1-3 are general status codes that apply to the majority of devices presented in subsequent chapters. Error codes specific to only one or two drivers are described only in relation to the devices for which they are returned. Table 1-3 describes successful and pending codes first, then error codes.

Table 1-3
I/O Status Conditions

Code	Reason
IS.SUC	**Successful completion** The I/O operation specified in the QIO$ directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	**I/O request pending** The I/O operation specified in the QIO$ directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The specified I/O operation was canceled with IO.KIL while in progress or while still in the I/O queue.
IE.ALN	**File already open** The task attempted to open a file on the physical device unit associated with the specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BAD	**Bad parameter** An invalid specification was supplied for one or more of the device-dependent QIO$ parameters (words 6 - 11). For example, a bad channel number or gain code was specified in an analog-to-digital converter I/O operation.
IE.BBE	**Bad block on device** One or more bad blocks were found. Data cannot be written on or read from bad blocks.
IE.BLK	**Illegal block number** An invalid block number was specified for a file-structured physical device unit. This code is returned, for example, if block 4800 is specified for an RK05 disk, on which legal block numbers extend from 0 through 4799.

(continued on next page)

Table 1-3 (Cont.)
I/O Status Conditions

Code	Reason
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a buffer, but only word (or double-word) alignment is legal for the physical device unit. For example, a disk function requiring word alignment was requested, but the buffer was aligned on a byte boundary. Alternatively, the length of a buffer was not an appropriate multiple of bytes. For example, all RP03 disk transfers must be an even multiple of four bytes.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached to the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached to the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	**Device not ready** The physical device unit specified in the QIO$ directive was not ready to perform the desired I/O operation. This code is often returned as the result of an interrupt time-out; that is, a reasonable amount of time has passed, and the physical device unit has not responded.
IE.EOF	**End-of-file encountered** An end-of-file mark, record, or control character was recognized on the input device.
IE.FHE	**Fatal hardware error** Controller is physically unable to reach the location where input/output is to be performed on the device. The operation cannot be completed.
IE.IFC	**Illegal function** A function code that was invalid for the specified physical device unit was specified in an I/O request. This code is returned if the task attempts to execute an invalid function or if, for example, a read function is requested on an output-only device, such as the line printer.

(continued on next page)

Table 1-3 (Cont.)
I/O Status Conditions

Code	Reason
IE.NLN	**File not open** The task tried to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	**Insufficient buffer space** Dynamic storage space has been depleted, and not enough buffer space was available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for such an operation.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO$ directive was not on line. When the system was bootstrapped, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	**Illegal read overlay request** A read overlay was requested and the physical device unit specified in the QIO$ directive was not the physical device unit from which the task was installed. The read overlay function can be executed only on the physical device unit from which the task image containing the overlays was installed.
IE.PRI	**Privilege violation** The task that issued a request was not privileged to execute that request. For example, for the UDC11 and LPS11 devices, a checkpointable task attempted to connect to interrupts or to execute a synchronous sampling function.
IE.SPC	**Illegal address space** The following conditions can cause this error: • The buffer that your task requested for a read or write operation was partially or totally outside the address space of your task. • You specified a byte count of 0. • You specified TF.XCC and AST2 in the same QIO$ request.

(continued on next page)

Table 1-3 (Cont.)
I/O Status Conditions

Code	Reason
IE.VER	**Unrecoverable error** After the system attempted its standard number of retries after an error occurred, the operation still could not be completed. This code is returned in the case of parity, CRC, or similar errors.
IE.WCK	**Write check error** An error was detected during the check (read) following a write operation.
IE.WLK	**Write-locked device** The task attempted to write on a write-locked physical device unit.

1.10 POWER-FAIL RECOVERY PROCEDURES FOR DISKS AND DECTAPE

Power-fail recovery recommendations for various devices are included in the following chapters. For disks and DECtape, power recovery ASTs should be used. Before returning for normal I/O operations, the AST service routine should provide a sufficient time delay, for the disk to attain normal operating speed before actually attempting read and write operations.

If QIO$s are being used for disk or DECtape I/O operations during power-fail recovery, an IE.DNR error status may be returned if the device is not up to operating speed when the request is issued. When this error is returned, your task should wait for the device to attain operating speed and attempt the I/O operation again prior to reporting an error. For example, an RK05 disk may require approximately 1 minute to attain operating speed after a power failure.

1.11 RSX-11M DEVICES

Both RSX-11M and RSX-11M-PLUS support the devices listed in Table 1-4 except as indicated. DEC supplies drivers for each of these devices. Table 1-4 lists the physical name, the driver, and the device description.

Table 1-4
Devices Supported by RSX-11M/M-PLUS

Physical Name	Driver	Description of Terminal
TT	TTDRV	ASR/KSR-33 and ASR/KSR-35 Teletypewriters
TT	TTDRV	All terminals supported by RSX-11M/M-PLUS, including the LA-, LQP-, VT05-, VT50-, VT61-, VT100-, VT200-, and RT02-series terminals. See the Software Product Description for your system.

(continued on next page)

Table 1-4 (Cont.)
Devices Supported by RSX-11M/M-PLUS

Physical Name	Driver	Description of Terminal Line Interface
TT	TTDRV	DH11 and DH11-DM11-BB Asynchronous Communication Line Interface Multiplexer
TT	TTDRV	DHV11 and DHU11 Asynchronous Communication Line Interface Multiplexer
TT	TTDRV	DL11-A, DL11-B, DL11-C, DL11-D, DL11-E and DL11-W Asynchronous Communication Line Interfaces
TT	TTDRV	DLV11-E, DLV11-F Asynchronous Communication Line Interfaces
TT	TTDRV	DZ11 and DZV11 Asynchronous Communication Line Interface Multiplexer
TT	TTDRV	DZQ11 Q-Bus 4-Line Terminal Multiplexer

Physical Name	Driver	Description of Disk Device
DB	DBDRV	RP04, RP05, RP06 Pack Disk
DF	DFDRV	RF11/RS11 Fixed-Head Disk
DK	DKDRV	RK11/RK05 or RK05F Cartridge Disk
DL	DLDRV	RLV12/RL01/RL02 Cartridge Disk
DM	DMDRV	RK611/RK06 or RK07 Cartridge Disk
DP	DPDRV	RP11/RP02 or RP03 Pack Disk
DR	DRDRV	RM02, RM03, RM05 Pack Disk
DR	DRDRV	RM80, RP07 Fixed-Media Disk
DS	DSDRV	RS03/RS04 Fixed-Head Disk
DU	DUDRV	KDA50/UDA50/RA80/RA81 Fixed-Media Disk
DU	DUDRV	KDA50/UDA50/RA60 Pack Disk
DU	DUDRV	RC25 Fixed-Media and Removable Cartridge Disk Subsystem
DU	DUDRV	RD51/RD52 Fixed-Media Disk
DU	DUDRV	RX50 Flexible Disk
DX	DXDRV	RX11/RX01 Flexible Disk
DY	DYDRV	RX211/RX02 Flexible Disk
EM	EMDRV	ML-11 Fast Electronic Mass Storage Device

(continued on next page)

Table 1-4 (Cont.)
Devices Supported by RSX-11M/M-PLUS

Physical Name	Driver	Description of Tape Device
DD	DDDRV	DL11/TU58 DECtape II
MS	MSDRV	TU80 Magnetic Tape Subsystem
MS	MSDRV	TSV05/TK25 Magnetic Tape Subsystem
MS	MSDRV	TS11 Magnetic Tape Subsystem
MT	MTDRV	TM11 Magnetic Tape Controller with TE10,TU10, or TS03 Drive (not supported in Micro/RSX)
MM	MMDRV	RH11/70 Controller with TM02/03 Formatter and TE16, TU16, or TU45 Drive (not supported in Micro/RSX)
MM	MMDRV	RH11/70 Controller with TM03 Formatter and TU77 Drive (not supported in Micro/RSX)
MU	MUDRV	TK50 Cartridge Tape Drive
MU	MUDRV	TU81 Tape Drive

Physical Name	Driver	Description of Cassette Device
CT	CTDRV	TA11 Tape Cassette
CT	CTDRV	TU60 Tape Cassette

Physical Name	Driver	Description of Line Printer
LP	LPDRV	LP11 Controller with LP14, LP01, LP02, LP04, LP05, LP06, LP07, LP26, LP27 Line Printers
LP	LPDRV	LPV11/LP25/LP26 Line Printers, LN01/LN03 Laser Printer
LP	LPDRV	LS11 Controller and Line Printer (not supported in Micro/RSX)
LP	LPDRV	LV11 Controller with LV01 Line Printer (not supported in Micro/RSX)
LP	LPDRV	LA180 Controller and Line Printer (not supported in Micro/RSX)

(continued on next page)

Table 1-4 (Cont.)
Devices Supported by RSX-11M/M-PLUS

Physical Name	Driver	Description of Card Reader
CR	CRDRV	CR11/CM11 Card Reader (not supported in Micro/RSX)

Physical Name	Driver	Description of Communication Line Interface
XB	XBDRV	DA11-B Asynchronous Communication Line Interface (RSX-11M support only)
XB	XBDRV	DA11-B Parallel Unibus Link (RSX-11M support only)
XL	XLDRV	DL11-E Asynchronous Communication Line Interface (not supported in Micro/RSX)
XL	XLDRV	DLV11-E Asynchronous Communication Line Interface (not supported in Micro/RSX)
XC	XMDRV	DMC11 Synchronous Communication Line Interface (not supported in Micro/RSX)
XE	XEDRV	RSX QIO DEUNA Driver
XP	XPDRV	DP11 Synchronous Communication Line Interface (RSX-11M support only)
XQ	XQDRV	DQ11 Synchronous Communication Line Interface (RSX-11M support only)
XU	XUDRV	DU11 Synchronous Communication Line Interface (RSX-11M support only)
XW	XWDRV	DUP11 Synchronous Communication Line Interface (no supported in Micro/RSX)

Physical Name	Driver	Description of Analog-to-Digital Converter
AF	AFDRV	AFC11 Analog-to-Digital Converter (RSX-11M support only)
AD	ADDRV	AD01-D Analog-to-Digital Converter (RSX-11M support only)

Physical Name	Driver	Description of Digital Controller
UD	UDDRV	UDC11 Universal Digital Controller (RSX-11M support only)

(continued on next page)

Table 1-4 (Cont.)
Devices Supported by RSX-11M/M-PLUS

Physical Name	Driver	Description of Laboratory Peripheral System or Device
AR	ARDRV	AR11 Laboratory Peripheral System (RSX-11M support only)
LS	LSDRV	LPS11 Laboratory Peripheral System (RSX-11M support only)
LA	LADRV	LPA11-K Laboratory Peripheral Accelerator (not supported on Micro/RSX)
Physical Name	**Driver**	**Description of Paper Tape Device**
PP	PPDRV	PC11 Paper Tape Reader/Punch (not supported on Micro/RSX)
PR	PRDRV	PR11 Paper Tape Reader (not supported on Micro/RSX)
Physical Name	**Driver**	**Description of Industrial Control Subsystem**
IC	ICDRV	ICS/ICR Local and Remote Subsystems (RSX-11M support only)
IS	ISDRV	DSS/DRS Digital Input and Output Subsystems (RSX-11M support only)
Physical Name	**Driver**	**Description of Null Device**
NL	NLDRV	Null device driver; a software construct to eliminate unwanted output
Physical Name	**Driver**	**Description of Graphic Subsystem**
GR	GRDRV	VT11 Graphics Display System (RSX-11M support only)
GR	GRDRV	VS60 Graphics Display System (RSX-11M support only)

(continued on next page)

Table 1-4 (Cont.)
Devices Supported by RSX-11M/M-PLUS

Physical Name	Driver	Description of K-Series Laboratory Peripheral (Not Micro/RSX)
		AA11-K Digital-to-Analog Converter and Display
		AD11-K Analog-to-Digital Converter
		AM11-K Multiple-Gain Multiplexer
		DR11-K Digital I/O Interface
		KW11-K Programmable Real-Time Clock

Physical Name	Driver	Description of Communications Device
LR/LT	LRDRV	PCL11 Parallel Communications Link (RSX-11M-PLUS support only)
LR/LT	LRDRV	PCL11-A/PCL11-B Receiver Port (RSX-11M-PLUS support only)

Physical Name	Driver	Description of Device
QA-QZ	Any	A physical name reserved for customer use
JA-JZ	Any	A physical name reserved for customer use

CHAPTER 2

FULL-DUPLEX TERMINAL DRIVER

2.1 INTRODUCTION

This chapter describes the use of the full-duplex terminal driver (TTDRV.TSK) supplied with the RSX-11M-PLUS system or available as a SYSGEN option for RSX-11M systems. This chapter contains descriptions of all the QIO$ functions that you can use to read from or write to a full-duplex terminal. Additionally, it contains a description of terminal subfunctions that are specific to terminal drivers and that modify the action of the QIO$ functions. You can combine the subfunctions in a Logical OR with the QIO$ function. Specific programming circumstances are combined with the description of the QIO$ function where they apply. A compact, half-duplex terminal driver is also available on RSX-11M systems only. It is described in Chapter 3.

Note that either terminal driver can be selected during RSX-11M system generation. RSX-11M-PLUS systems use the full-duplex terminal driver only.

Throughout the remainder of this chapter, references made to MCR can generally be applied to other command line interpreters (for example, DCL). In addition, the prompt displayed on a terminal in response to invoking a command line interpreter is appropriate for the specific command line interpreter in use. For example, when MCR is invoked, the MCR prompt is displayed as follows:

```
MCR>
```

2.1.1 Full-Duplex Terminal Driver

The full-duplex terminal driver described in this chapter works with a wide variety of terminals. It contains the following features:

- Full-duplex operation
- Type-ahead buffering
- Eight-bit characters
- Detection of hard receive errors
- Increased byte transfer length (8128 bytes)
- Additional terminal characteristics
- Additional terminal types
- Optional time-out on solicited input

- Device-independent cursor control
- Redisplay of prompt buffer when CTRL/R or CTRL/U is pressed
- Automatic XOFF character generation when a read is completed while in half-duplex mode, if requested
- Autobaud speed detection
- Added hardware support

2.1.2 Terminals Supported by the Full-Duplex Terminal Driver

The full-duplex terminal driver supports a variety of terminal devices, as listed in Table 2-1. Table 2-2 describes standard terminal interfaces. Subsequent sections describe each device in greater detail.

Table 2-1
Supported Terminal Devices

Model	Columns	Lines/ Screen [1]	Character Set	Baud Range	Uppercase Send	Lowercase Receive
ASR-33/35	72		64	110		
DTC01					9600	
KSR-33/35	72		64	110		
LA12	132		96	50-9600	yes	yes
LA100	132		96	110-9600	yes	yes
LA30-P	80		64	300		
LA30-S	80		64	110-300		
LA34	132		96	110-300	yes	yes
LA36	132		64-96	110-300	yes	yes [2]
LA38	132		96	110-300	yes	yes
LA120	132		96	50-9600	yes	yes
LA180S	132		96	300-9600		yes
LQP02	132/158			110-9600		
LA50	80/96/132			110-4800		
LN03			_ [4]	1200-19200		yes
RT02	64	1	64	110-1200		
RT02-C	64	1	64	110-1200		
VT05B	72	20	64	110-2400	yes	
VT50	80	12	64	110-9600		
VT50H	80	12	64	110-9600		
VT52	80	24	96	110-9600	yes	yes
VT55	80	24	96	110-9600	yes	yes
VT61	80	24	96	110-9600	yes	yes
VT100	80-132	24	96	50-9600	yes	yes
VT101	80-132	24	96	50-19200	yes	yes
VT102	80-132	24	96	50-9600	yes	yes
VT105	80-132	24	96	50-19200	yes	yes
VT125	80-132	24	96	50-9600	yes	yes

1. Applies only to video terminals.

2. Only for 96-character terminal. The terminal driver supports the terminal interfaces summarized in Table 2-2. These interfaces are described in greater detail in Section 2.17. Programming is identical for all interfaces.

4. Includes the DEC Multinational Character Set.

(continued on next page)

Table 2-1 (Cont.)
Supported Terminal Devices

Model	Columns	Lines/ Screen [1]	Character Set	Baud Range	Uppercase Send	Lowercase Receive
VT131	80-132	24	96	50-19200	yes	yes
VT132	80-132	24	96 [3]	50-19200	yes	yes
VT220	80-132	24	94 [3]	50-19200	yes	yes
VT240	80-132	24	94 [3]	50-19200	yes	yes
VT241	80-132	24	94 [3]	50-19200	yes	yes

1. Applies only to video terminals.

3. Five character sets of 94 characters each. Includes the DEC Multinational Character Set.

Table 2-2
Standard Terminal Interfaces

Model	Type
DH11	16-line multiplexer [1]
DHU11	Unibus 16-line asynchronous multiplexer
DHV11	8-line multiplexer [2]
DH11-DM11-BB	16-line multiplexer with modem control [3]
DJ11	16-line multiplexer
DL11-A/B/C/D/E/W	Single-line interfaces
DLV11-E/F	Single-line interfaces [4]
DZ11	8-line multiplexer with modem control [4]
DZQ11	Q-bus 4-line terminal multiplexer

1. Direct memory access (DMA) is supported in the full-duplex terminal driver only.

2. Full duplex terminal driver only.

3. Full-duplex control only. For example, in the United States, a Bell 103A-type modem provides full-duplex control only.

4. DLV11 support with modem control is provided in the full-duplex terminal driver only.

Terminal input lines can have a maximum length of 8128 (8K minus 64) bytes. Extra characters of an input line that exceed the maximum line length generally become an unsolicited input line if the terminal is not attached with the type-ahead buffering feature enabled. The full-duplex terminal driver discards all unsolicited input from an unattached, slave terminal.

2.1.2.1 **ASR-33/35 Teletypewriters** - The ASR-33 and ASR-35 Teletypewriters are asynchronous, hardcopy terminals. No paper-tape reader or punch capability is supported.

2.1.2.2 **KSR-33/35 Teletypewriters** - The KSR-33 and KSR-35 Teletypewriters are asynchronous, hardcopy terminals.

2.1.2.3 **LA12 Portable Terminal** - The LA12 is a personal, portable, hardcopy terminal.

2.1.2.4 **LA100 DECprinter** - The LA100 is a desk-top, matrix, hardcopy terminal.

2.1.2.5 **LA30 DECwriters** - The LA30 DECwriter is an asynchronous, hardcopy terminal that is capable of producing an original and one copy. The LA30-P is connected by a parallel line and the LA30-S is connected by a serial line.

2.1.2.6 **LA36 DECwriter** - The LA36 DECwriter is an asynchronous terminal that produces hard copy and operates in serial mode. It has an impact printer capable of generating multipart and special preprinted forms. The LA36 can receive and transmit both uppercase and lowercase characters.

2.1.2.7 **LA34/38 DECwriters** - The LA34 DECwriter is an asynchronous terminal that produces hard copy and uses a platen paper-feed mechanism.

The LA38 DECwriter includes a detachable tractor-feed mechanism for use with continuous forms.

2.1.2.8 **LA120 DECwriter** - The LA120 DECwriter is a hardcopy, uppercase and lowercase terminal. It can print multipart forms at speeds up to 180 characters per second. You can select serial communications speed from 14 baud rates ranging from 50 to 9600 bps; the terminal driver supports split transmit and receive baud rates. Hardware features allow bidirectional printing for maximum printing speed, and also allow you to select features, including font size, line spacing, tabs, margins, and forms control. Also, you can set up these functions if you issue appropriate ANSI-standard escape sequences.

2.1.2.9 **LA180S DECprinter** - The LA180S DECprinter is a serial version of the LA180. It is a print-only device (it has no keyboard) that can generate multipart forms. The LA180S can print uppercase and lowercase letters.

2.1.2.10 **LQP02 Letter-Quality Printer** - The LQP02 Letter-quality Printer is a formed-character, desktop printer incorporating daisywheel technology. This letter-quality printer offers over 100 character sets and handles regular office stationery up to a maximum of 15 inches (with a print capacity 13.5 inches). You can select lines per inch and characters per inch: 10 or 12 characters per inch and 2, 3, 4, 6, and 8 lines per inch. At 10 characters per inch you

get 132 columns, and at 12 characters per inch you get 158 columns. The buffer capacity is 256(decimal) characters.

2.1.2.11 **LA50 Personal Printer** - The LA50 Personal Printer is a desktop dot-matrix impact printer. It has two print modes: text mode and enhanced print mode. In text mode, it prints 100 characters per second. In enhanced print quality mode, it prints 50 characters per second and creates a crisper, more uniform character than an ordinary dot-matrix printer. You can choose 10, 12, or 16 characters per inch that print up to 80, 96, or 132 columns respectively. There can be 6, 8, or 12 lines per inch. The buffer capacity is 255(decimal) characters.

2.1.2.12 **RT02 Alphanumeric Display Terminal and RT02-C Badge Reader/Alphanumeric Display Terminal** - The RT02 is an alphanumeric display terminal for applications in which source data is primarily numeric. A shift key permits the entry of 30 discrete characters, including uppercase alphabetic characters. The RT02 can, however, receive and display 64 characters.

The RT02-C model also contains a badge reader. This option provides a reliable method of identifying and controlling access to the PDP-11 minicomputer or to a secure facility. Furthermore, data in a format corresponding to that of a badge (22-column fixed data) can be entered quickly.

2.1.2.13 **VT05B Alphanumeric Display Terminal** - The VT05B is an alphanumeric display terminal that consists of a CRT display and a self-contained keyboard. The VT05B offers direct cursor addressing.

2.1.2.14 **VT50 Alphanumeric Display Terminal** - The VT50 is an alphanumeric display terminal that consists of a CRT display and a keyboard. It is similar to the VT05B in operation, but does not offer direct cursor addressing.

2.1.2.15 **VT50H Alphanumeric Display Terminal** - The VT50H is an alphanumeric display terminal with CRT display, keyboard, and numeric pad. It offers direct cursor addressing, but its direct cursor addressing is not compatible with that of the VT05B.

2.1.2.16 **VT52 Alphanumeric Display Terminal** - The VT52 is an uppercase and lowercase alphanumeric terminal with CRT display. It also has a numeric pad and direct cursor addressing. The VT52's direct cursor addressing is compatible with that of the VT50H, but not with that of the VT05B. The VT52 can be configured with a built-in thermal printer.

2.1.2.17 **VT55 Graphics Display Terminal** - The VT55 is similar to the VT52 in its operation as an alphanumeric terminal. The VT55 offers graphics display features that are accessible by a task.

2.1.2.18 **VT61 Alphanumeric Display Terminal** - The VT61 is an uppercase and lowercase alphanumeric terminal with an integral microprocessor. It offers two 128-member character sets and numerous built-in functions for editing and forms preparation as well as a block-transfer mode.

2.1.2.19 **VT100 DECscope** - The VT100 DECscope is an uppercase and lowercase alphanumeric keyboard and video display terminal. It can display 24 lines of 80 to 132 characters per line. You can select serial communications speed from baud rates ranging from 50 to 9600 bps. Hardware features allow you to select display characteristics and functions including smooth scroll, reverse video, and so forth. The system also sets up these functions if you issue appropriate ANSI-standard escape sequences.

2.1.2.20 **VT101 DECscope** - The VT101 DECscope is functionally identical to the VT100. However, it does not support the advanced video features.

2.1.2.21 **VT102 DECscope** - The VT102 DECscope is functionally identical to the VT100. However, it does not have any expansion capability and does not support the advanced video features. It has enhanced modem control, and it includes a port for a printer.

2.1.2.22 **VT105 DECscope** - The VT105 DECscope is an alphanumeric and graphic display video terminal. The VT105 can display two graphs, two shaded graphs, or two strip charts. These graphs may have alphanumeric labels.

2.1.2.23 **VT131 DECscope** - The VT131 is the same as the VT102 with the addition of built-in editing features.

2.1.2.24 **VT220 Terminal** - The VT220 Terminal is a general-purpose video display terminal displaying 24 rows of 80 or 132 columns. It has ANSI compatible control functions; user-definable function keys; video reverse, bold, underline, blink, double height/double width line attributes; and can run in VT100, VT200 7-bit, VT200 8-bit, and VT52 mode. Setup state allows you to configure the terminal and examine its status.

2.1.2.25 **VT240 Terminal** - The VT240 Terminal is a general-purpose video display terminal displaying 24 rows of 80 or 132 columns. It has: ANSI compatible control functions; user definable function keys; video reverse, bold, underline, blink, double height/double width line attribute; and can run in VT100, VT200 7-bit, VT200 8-bit, VT52 mode, 4014 mode (Tektronic (c) 4010/4014), and ReGIS graphics mode. Set-up state allows you to configure the terminal and examine its status. The VT240 has graphics capability to draw points, vectors, circles, arcs, and curves.

2.1.2.26 **VT241 Terminal** - The VT241 Terminal is functionally identical to the VT240 terminal except that the VT241 has a color monitor.

2.2 GET LUN INFORMATION MACRO

The Get LUN information directive (GLUN$) instructs the system to fill a 6-word buffer with information about the physical device unit to which the LUN is assigned. For more information about this directive, refer to Get LUN in the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual. The following section describes the information that Get LUN makes available for terminals in word 2 of the buffer.

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the terminal information shown in Table 2-3. A setting of 1 indicates that the described characteristic is true for terminals.

Table 2-3
Word 2 of the Get LUN Macro Buffer

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	1	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device is mountable as a communications channel
14	0	Device is mountable as a FILES-11 volume
15	0	Device is mountable

Words 3 and 4 of the buffer are undefined. Word 5 indicates the default buffer size (the width of the terminal carriage or display screen).

2.3 QIO$ MACRO

Standard QIO$ functions may be used with any device, whereas device-specific QIO$ functions apply only to specific devices or uses.

2.3.1 Format of QIO$C for Standard Functions

The QIO$ macros for standard functions take the following forms:

```
QIO$C  {!IO.ATT} ,...,
       {!IO.DET}
       {!IO.KIL}

QIO$C  {!IO.RLB} ,...,<stadd,size,,[tmo]>
       {!IO.RVB}

QIO$C  {!IO.WLB} ,...,<stadd,size,vfc>
       {!IO.WVB}
```

2.3.2 Format of QIO$C for Device-Specific Functions

The QIO$ macro for device-specific functions take the following forms:

```
QIO$C IO.ATA,...,<ast,[parameter2],[ast2]>

QIO$C IO.CCO,...,<stadd,size,vfc>

QIO$C IO.EIO,...,<stadd,size>

QIO$C IO.HNG,...,

QIO$C  {!IO.RAL} ,...,<stadd,size,[tmo]>
       {!IO.RNE}

QIO$C IO.RPR,...,<stadd,size,[tmo],pradd,prsize,vfc>

QIO$C IO.RST,...,<stadd,size,[tmo]>

QIO$C IO.RTT,...,<stadd,size,[tmo],table>

QIO$C  {!IO.WAL} ,...,<stadd,size,vfc>
       {!IO.WBT}

QIO$C  {!SF.GMC} ,...,<stadd,size>
       {!IO.GTS}

QIO$C SF.SMC,...,<stadd,size>
```

Table 2-4 lists the standard and device-specific functions of the QIO macro that are valid for terminals. The standard functions are described in Chapter 1. Some device-specific functions are options that may be selected during system generation. Two device-specific functions, SF.SMC and SF.GMC, have nonstandard function names.

Table 2-4
Standard and Device-Specific QIO Functions for Terminals

Format	Function
STANDARD FUNCTIONS:	
READ FUNCTIONS	
QIO$C IO.RLB,...,<stadd,size,[tmo]>	Read logical block (read typed input into buffer).
QIO$C IO.RVB,...,<stadd,size,[tmo]>	Read virtual block (read typed input into buffer).
WRITE FUNCTIONS	
QIO$C IO.WLB,...,<stadd,size,vfc>	Write logical block (print buffer contents).
QIO$C IO.WVB,...,<stadd,size,vfc>	Write virtual block (print buffer contents).
ATTACH, DETACH, AND CANCEL FUNCTIONS	
QIO$C IO.ATT,...	Attach device.
QIO$C IO.DET,...	Detach device.
QIO$C IO.KIL,...	Cancel I/O requests. (continued on next page) Table 2-4 (Cont.) Standard QIO Functions for Terminals
DEVICE-SPECIFIC FUNCTIONS:	
READ FUNCTIONS	
QIO$C IO.RAL,...,<stadd,size,[tmo]>	Read logical block; pass all charcters.
QIO$C IO.RNE,...,<stadd,size,[tmo]>	Read logical block; do not echo.
QIO$C IO.RPR,...,<stadd,size,[tmo], pradd,prsize,vfc> [1]	Read logical block after prompt.
QIO$C IO.RST,...,<stadd,size,[tmo]>	Read logical block ended by special terminators.
QIO$C IO.RTT,...,<stadd,size,[tmo], table>	Read logical block ended by specified special terminators.

1. System generation options in RSX-11M.

(continued on next page)

Table 2-4 (Cont.)
Standard and Device-Specific QIO Functions for Terminals

Format	Function
WRITE FUNCTIONS	
QIO$C IO.WAL,...,<stadd,size,vfc>	Write logical block; pass all characters.
QIO$C IO.WBT,...,<stadd,size,vfc> [1]	Write logical block; break through any I/O conditions at terminal.
MISCELLANEOUS FUNCTIONS	
QIO$C IO.ATA,...,<ast,[parameter2],[ast2]> [1]	Attach device, specify unsolicited input-character AST.
QIO$C IO.CCO,...,<stadd,size,vfc>	Cancel CTRL/O (if in effect), then write logical block.
QIO$C IO.EIO {!TF.RLB / !TF.WLB} ,...,<stadd,size> [1]	Extended I/O.
QIO$C IO.GTS,...,<stadd,size> [1]	Get terminal support.
QIO$C IO.HNG,...	Hang up remote line.
QIO$C SF.GMC,...,<stadd,size> [1]	Get multiple characteristics.
QIO$C SF.SMC,...,<stadd,size> [1]	Set multiple characteristics.

1. System generation options in RSX-11M.

2.3.3 Parameters

The parameters for the various QIO$ macros have the following meanings:

Parameter	Meaning
ast	The entry point for an unsolicited input-character AST.
ast2	The entry point for a CTRL/C AST.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
parameter2	A number that you can specify in your task to identify this terminal as the input source when an unsolicited character AST routine is entered.
pradd	The starting address of the byte buffer where the prompt is stored.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
prsize	The size of the pradd prompt buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space. For IO.EIO, SF.GMC, IO.GTS, and SF.SMC functions, size must be an even value.
stadd	The starting address of the data buffer. The address must be word-aligned for IO.EIO, SF.GMC, IO.GTS, and SF.SMC; otherwise, stadd may be on a byte boundary.
table	The address of the 16-word user-defined terminator table.
tmo	An optional time-out count specified in 10-second intervals. (For IO.EIO, the interval is specified in seconds.) Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) intervals. If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer. If you need more than 255(decimal) intervals (or 255(decimal) seconds for IO.EIO), issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.
vfc	The vfc parameter normally specifies cursor position. If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

Parameter	Meaning
vfc (Cont.)	However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter. Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

2.3.4 Subfunction Bits

The terminal-specific functions described in this section are selected by using subfunction bits. A subfunction bit further modifies the action of an I/O function. A subfunction bit is specified by the name TF.xxx, and an I/O function is specified by the name IO.xxx, where xxx in each case is an acronym that represents the specific kind of function requested.

As an example, a QIO$ function to a terminal to request a read with no echo (IO.RNE) can be modified to read all characters. The "read all characters" subfunction bit is TF.RAL. To modify the function, you perform a logical OR of the subfunction bit with the QIO$ function in the QIO$ statement. To create the logical OR of the bit and the function, in this example, the QIO$ statement would look like this:

```
QIO$ IO.RNE!TF.RAL,...,...
```

See Section 2.4.2 for a listing of QIO$ functions and relative subfunction bits that can be issued.

Each subfunction bit is listed with its symbolic name and meaning as follows:

Subfunction	Meaning
TF.AST	Unsolicited-Input-Character AST - For IO.ATT or IO.ATT!TF.ESQ, ast in the QIO$ macro specifies the address of an AST service routine to be entered when an unsolicited input character is entered. Control passes to ast whenever an unsolicited character (other than CTRL/Q, CTRL/S, CTRL/X, or CTRL/O) is entered at the terminal.
TF.BIN	Binary Prompt (Send Prompt As Pass All) - The prompt is sent to the terminal without interpretation by the driver. This is similar, for the prompt, to a write-pass-all operation.
TF.CCO	Cancel CTRL/O - The driver writes a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. If CTRL/O is in effect, it is canceled before the write occurs.
TF.ESQ	Recognize Escape Sequences - Escape sequences from the terminal are returned to the task. Otherwise, ESC is a line terminator. The subfunction TF.ESQ is for use with IO.ATA or IO.ATT!TF.AST.

Subfunction	Meaning
TF.NOT	Notification Of Unsolicited Input - Unsolicited input causes an AST and entry into the AST service routine in the task. When the full-duplex terminal driver receives unsolicited terminal input (except CTRL/C) and you used the TF.NOT subfunction with IO.ATA, the resulting AST serves only as notification of unsolicited terminal input; the terminal driver does not pass the character to the task. Upon entry to the AST service routine, the high byte of the first word on the stack identifies the terminal causing the AST (parameter2 in the IO.ATA function). Using the TF.NOT subfunction allows a task to monitor more than one terminal for unsolicited input without continuously reading each terminal for possible unsolicited input. Note that the TF.NOT subfunction cannot be used with the CTRL/C AST (ast2 in IO.ATA); an unsolicited CTRL/C character flushes the type-ahead buffer.
TF.RAL	Read All Characters (Pass All) - This subfunction allows the passage of all characters to the requesting task. The driver does not intercept control characters. The characteristic TC.8BC, when set, allows the driver to pass 8 bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and are not interpreted by the driver.
TF.RCU	Restore Cursor Position - When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.
TF.RDI	Read With Default Input - The default input that you specified in the extended I/O item list is displayed as an input line at the start of the read on the terminal. You may change this line or use it as input to the system. This subfunction is for use with the extended I/O function (IO.EIO) only.
TF.RES	Read With Escape Sequence Processing Enabled - This subfunction enables escape sequence recognition for the read operation in extended I/O; it is effective for only one read.
TF.RLB	Read Logical Block - This subfunction causes the driver to read a logical block from the specified terminal; it is for use with the extended I/O (IO.EIO) function only.
TF.RLU	Read With Lowercase to Uppercase Conversion - The task that uses this subfunction gets input in the buffer in upper case; it is for use with the extended I/O (IO.EIO) function only.
TF.RNE	Read With No Echo - This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while Read With No Echo is in progress.

Subfunction	Meaning

TF.RNF Read With No Filter - This subfunction reads and passes through CTRL/U, CTRL/R, and DELETE characters as normal characters. It is for use with the extended I/O (IO.EIO) function only.

TF.RPR Read After Prompt - This subfunction is for use with the extended I/O only. The TF.RPR subfunction causes the driver to send a prompt to the terminal, and the driver immediately follows the prompt with a read function at the terminal. The TF.RPR acts as an IO.WLB (to write a prompt to the terminal) followed by IO.RLB. However, TF.RPR differs from the combination of those two functions as follows:

- System overhead is lower with the TF.RPR because only one QIO$ is processed.
- When using the TF.RPR function, there is no "window" during which a response to the prompt may be ignored. Such a window occurs if the task uses IO.WLB followed by an IO.RLB, because no read may be posted at the time the response is received.
- If the issuing task is checkpointable, it can be checkpointed during both the prompt and the read requested by the TF.RPR.
- A CTRL/O that may be in effect prior to issuing the TF.RPR is canceled before the prompt is written.

NOTE

If a TF.RPR function is in progress when the driver receives a CTRL/R or CTRL/U, the prompt is redisplayed.

TF.RPT Read In Pass-Through Mode - This subfunction passes all characters except XON/XOFF. It allows the passage of all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass eight bits instead of seven. The driver intercepts the control characters CTRL/S and CTRL/Q. Other control characters, for example, CTRL/C, CTRL/O, and CTRL/Z, are passed to the task and are not interpreted by the driver. This subfunction is for use with the extended I/O (IO.EIO) function only.

TF.RST Read With Special Terminators - Special characters in the ranges 0-037 and 175-177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT (177) does not erase, and a CTRL/C does not produce a CLI prompt, or, if CTRL/C abort is enabled, abort tasks. CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators.

TF.RST sets TF.TNE by default, which means that terminators are not echoed on the terminal screen.

If uppercase to lowercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

Subfunction	Meaning
TF.RTT	Read With Terminator Table - This subfunction is for use with the IO.EIO extended I/O function only. Control characters function normally with TF.RTT. Terminators echo by default. The additional use of subfunction TF.TNE prevents the echoing of terminators on the terminal screen. If you want to use special control characters as terminators, their normal function should be disabled with the subfunction TF.RNF or TF.RAL, or the characteristic TC.PTH. The terminator table (a bit mask table) length can be from 1 through 32(decimal) bytes, where bit 0 is a null character, bit 1 is a CTRL/A, and so forth. The terminator table address is in the item list of the IO.EIO function. To use ASCII characters 128(decimal) through 255(decimal), the characteristic TC.8BC must be set.
TF.TMO	Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. (For IO.EIO, the interval is specified in seconds.) Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) intervals. If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer. If you need more than 255(decimal) intervals (or 255(decimal) seconds for IO.EIO), issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.
TF.TNE	Read Terminators With No Echo - This subfunction allows reading terminator characters from the terminal without their being echoed on the terminal screen as they are entered. It is for use with the extended I/O function (IO.EIO) only.
TF.WAL	Write All Characters - During a write-pass-all operation (as in IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation. It does not intercept control characters, and it does not keep track of cursor position. Long lines are not wrapped around if input/output wraparound has been selected.

Subfunction	Meaning
TF.WBT	Break-through Write - This subfunction instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the break-through write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another break-through write that is in progress. The effect of this is that a CTRL/S can stop break-through write functions. Thus, it may be desirable for tasks to time out on break-through operations. If a read is currently posted, the break-through write proceeds, and an automatic CTRL/R redisplays any input that was received before the break-through write was effected (if the terminal is not in the full-duplex mode). CTRL/O, if in effect, is canceled. An escape sequence that was interrupted is rubbed out. Break-through write may be issued by a privileged task only. (The privileged MCR command BRO (broadcast) uses IO.WBT.)
TF.WIR	Write With Input Redisplayed - This subfunction performs a write to the terminal. If a read is in progress at the terminal and you have entered characters in the input line, the prompt and the characters are redisplayed at the end of the write.
TF.WLB	Write Logical Block To The Specified Device Unit - Write logical block to the specified terminal. This subfunction is used with the extended I/O (IO.EIO) function only.
TF.XCC	Exclude CTRL/C or Abort Active Tasks - For use with the IO.ATA function. When TF.XCC is included in the IO.ATA function, all characters (except CTRL/C) are handled in the manner previously described. CTRL/C marks the beginning of a command line interpreter (CLI) line that is processed by a CLI task, or, if CTRL/C abort is enabled, aborts tasks active at the terminal. None of the characters, including the CTRL/C, are sent to the task issuing the function. Note that you can use either ast2 or TF.XCC, but not both in the same QIO request. If both are specified in the request, an IE.SPC error is returned.
TF.XOF	Send XOFF - The driver sends an XOFF to the terminal after its read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

See Section 2.4.2 for a list of bits that can be combined in a logical OR with QIO$ functions. If a task invokes a subfunction bit that is not supported on the system, the subfunction bit is ignored, and the QIO$ request is not rejected. For example, if break-through write (TF.WBT) is not supported, an IO.WBT or IO.WLB!TF.WBT function is interpreted as an IO.WLB function.

In the following example, the QIO$ request uses more than one subfunction bit: a nonechoed read (TF.RNE), terminated by a special terminator character (TF.RST), and preceded by a prompt.

```
QIO$C IO.RPR!TF.RNE!TF.RST,...,<stadd,size,,pradd,prsize,vfc>
```

2.4 DEVICE-SPECIFIC QIO$ FUNCTIONS

The following sections describe the device-specific functions for the full-duplex terminal driver. Some full-duplex terminal driver functions and features are system generation options. These options are briefly described in the following section.

2.4.1 System Generation Options in the Full-Duplex Terminal Driver

Some device-specific functions described in this section are system generation options. These optional functions and other system generation options for the full-duplex terminal driver are listed as follows:

- **Unsolicited Input Timeout** - Discards unsolicited input when the time-out value that you specified during system generation expires.

- **Unsolicited Input Character AST** - Specifies an AST entry point for unsolicited input-character handling. This support is automatically included if your Executive supports ASTs.

- **Break-Through Write** - Instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the break-through write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another break-through write that is in progress. The effect of this is that a CTRL/S can stop break-through write functions. Thus, it may be desirable for tasks to time out on break-through operations.

  If a read is currently posted, the break-through write proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the break-through write was effected (if the terminal is not in the full-duplex mode).

  CTRL/O, if in effect, is canceled.

  An escape sequence that was interrupted is rubbed out.

  Break-through write may be issued by a privileged task only. (The privileged MCR command BRO (broadcast) uses IO.WBT.)

- **CTRL/R Retype** - Sends a carriage return and line feed to the terminal followed by the input buffer contents whenever you type a CTRL/R at the terminal.

- **Escape Sequence Handling** - Recognizes and treats escape sequences as line terminators for all solicited input except read-pass-all requests. See the QIO$ functions IO.RAL, IO.RST, and IO.RTT in the following sections, and see Section 2.7 for a description of escape sequences.

- **Extended I/O** - Allows the use of IO.EIO with TF.WLB or TF.RLB to increase the number of allowable I/O subfunctions.

- **Extended Network Command Terminal (NCT) Support** - Allows the use of a terminal as a network command terminal.

- **Get Multiple Characteristics** - A task can determine the characteristics of individual terminals. See Section 2.4.15 for information about the QIO$ SF.GMC function.

- **Set Multiple Characteristics** - A task can set the physical characteristics of a terminal. See Section 2.4.16 for information about the QIO$ SF.SMC function.

- **Get Terminal Support** - A task can determine which terminal driver options were selected during system generation. See Section 2.4.6 for information about the QIO$ IO.GTS function.

- **Read After Prompt** - Writes a prompt to the terminal and immediately follows it with a read. Reduces overhead and allows a task exclusive access to the terminal for the write and following read. See the QIO$ IO.RPR function in Section 2.4.10.

- **CRT Rubout** - Allows the DELETE (or RUBOUT) key to erase a character from the CRT screen by echoing the characters to be deleted as backspace-space-backspace. See Section 2.6.2.

- **Hard Receive Error Detection** - Known as Unrecoverable Input Error Notification in system generation. The driver flags framing errors, character parity errors, and data overruns and then passes the input characters to the requesting task with notification of an input error (including type). See Section 2.10.2.

- **Terminal-Independent Cursor Control** - The driver outputs a cursor-positioning command before it outputs the contents of the buffer if you specify the vfc parameter for an output buffer.

- **Modem Control** - The default answer speed for modems is set during system generation time but can be changed on line with the SET command.

2.4.2 Functions and Allowed Subfunctions

Any given function except SF.GMC, SF.SMC, IO.EIO, and IO.GTS can be issued by the logical OR of a particular subfunction bit with another QIO$ function. Table 2-5 lists the functions with their allowed subfunctions. The subfunction bits are specified in the following QIO$C function descriptions; subfunction bits are described in general in Section 2.3.4.

Table 2-5
Summary of Subfunction Bits

Function	Equivalent Subfunctions	Allowed Subfunctions
STANDARD FUNCTIONS		
IO.ATT	None	TF.AST, TF.ESQ
IO.DET	None	None
IO.KIL	None	None
IO.RLB	None	TF.RAL, TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.RVB [1]	None	TF.RAL, TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.WLB	None	TF.CCO, TF.RCU, TF.WBT, TF.WAL
IO.WVB [1]	None	TF.CCO, TF.RCU, TF.WAL, TF.WBT
DEVICE-SPECIFIC FUNCTIONS		
IO.ATA	IO.ATT!TF.AST	TF.ESQ, TF.NOT, TF.XCC
IO.CCO	IO.WLB!TF.CCO	TF.WAL, TF.WBT
IO.EIO [2]		TF.RLB, TF.WLB
SF.GMC		
IO.GTS		
IO.RAL	IO.RLB!TF.RAL	TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.RNE	IO.RLB!TF.RNE	TF.RAL, TF.RST, TF.TMO, TF.XOF
IO.RPR		TF.BIN, TF.RAL, TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.RST	IO.RLB!TF.RST	TF.RAL, TF.RNE, TF.TMO, TF.XOF
IO.RTT		TF.RAL, TF.RCU, TF.RNE, TF.TMO
SF.SMC		
IO.WAL	IO.WLB!TF.WAL	TF.CCO, TF.RCU, TF.WBT
IO.WBT	IO.WLB!TF.WBT	TF.CCO, TF.RCU, TF.WAL

1. Sufunctions are stripped off if they are specified with IO.RVB or IO.WVB.

2. You must use TF.RLB or TF.WLB with IO.EIO, but not both.

In addition to the device-specific QIO functions, the following sections also describe the use of subfunction bits.

2.4.3 QIO$C IO.ATA - Attach a Terminal with ASTs

The QIO$ IO.ATA macro attaches the terminal and identifies ast and ast2 as entry points for unsolicited input-character ASTs. With ast and ast2, IO.ATA specifies asynchronous system traps (ASTs) to process unsolicited input characters entered at the terminal. A minimum of one AST parameter (ast or ast2) is required.

IO.ATA is equivalent to the IO.ATT attach function executed in a logical OR with the subfunction bit TF.AST.

The use of IO.ATA is enhanced by the addition of the TF.NOT and TF.XCC subfunction bits, described later in this section. You may include any or all the subfunctions described in this section with the IO.ATA function.

Unless the TF.XCC subfunction is specified, CTRL/C is trapped by the task and does not reach the command line interpreter. Thus, any task that uses IO.ATA without the TF.XCC subfunction should recognize some input sequence as a request to terminate; otherwise, the command line interpreter cannot be invoked to abort the task in case of difficulty.

The format of the QIO$C IO.ATA macro is as follows:

```
                [!TF.ESQ]          [efn]
QIO$C IO.ATA    [!TF.NOT] ,lun,    [pri]  ,<[ast],[parameter2],[ast2]>
                [!TF.XCC]     ,    [isb]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	The entry point for an unsolicited input-character AST. Either ast or ast2 is required. Control passes to ast whenever an unsolicited character (other than CTRL/Q, CTRL/S, CTRL/X, or CTRL/O) is entered at the terminal. If ast2 is not specified, an unsolicited CTRL/C results in entering the AST specified in the ast parameter. If TF.NOT is specified, after the AST has been effected, the AST becomes "disarmed" until a read request is issued by the task. If multiple characters are received before the read request is issued, they are stored in the type-ahead buffer. Once the read request is received, the contents of the type-ahead buffer, including the character causing the AST,

Parameter	Meaning
ast (Cont.)	is returned to the task; the AST is then "armed" again for new unsolicited input characters. If TF.NOT is not specified, every usolicited character causes an AST. Upon entry to the AST routines, the unsolicited character and parameter2 are in the top word on the stack, as shown in ast2. That word must be removed from the stack before exiting the AST.
parameter2	Parameter2 is located in the high byte of SP+00. It is a value that you can specify to identify individual terminals in a multiterminal environment.
ast2	The entry point for an unsolicited CTRL/C AST. Either ast or ast2 is required. If you specify the ast2 parameter, an unsolicited CTRL/C character results in entering the AST specified in that parameter. If ast2 is not specified, an unsolicited CTRL/C results in entering the AST specified in the ast parameter. Upon entry to the AST routines, the unsolicited character and parameter2 are in the top word on the stack. That word must be removed from the stack before exiting the AST. The stack contents is shown next: SP+10 Event flag mask word SP+06 PS of task prior to AST SP+04 PC of task prior to AST SP+02 Task's directive status word SP+00 Unsolicited character in low byte After the AST has been effected, the AST becomes "disarmed" until a read request is issued by the task. If multiple characters are received before the read request is issued, they are stored in the type-ahead buffer. Once the read request is received, the contents of the type-ahead buffer, including the character causing the AST, is returned to the task; the AST is then "armed" again for new unsolicited input characters. Thus, using the TF.NOT subfunction allows a task to monitor more than one terminal for unsolicited input without the need to read each terminal continuously for possible unsolicited input. Note that the TF.NOT subfunction cannot be used with the CTRL/C AST; an unsolicited CTRL/C character flushes the type-ahead buffer. Either ast2 or TF.XCC can be used, but not both in the same QIO$ request. If you specify both in the request, an IE.SPC error is returned. See the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for further details on ASTs.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.ESQ	Recognize Escape Sequences - This subfunction issued with IO.ATT or IO.ATA attaches a terminal and notifies the driver that it recognizes escape sequences entered at that terminal. Escape sequences are recognized only for solicited input (if a read was issued to the terminal). (See Section 2.7 for a discussion of escape sequences.) If escape sequences are recognized, the sequence terminates input and a status code IS.ESC is returned. In addition, if uppercase to lowercase conversion is not enabled, the character ALTmode (codes 175 or 176, octal) is also treated as an escape character. If the terminal has not been declared capable of generating escape sequences, IO.ATA!TF.ESQ has no effect other than attaching the terminal. No escape sequences are returned to the task because any ESC sent by the terminal acts as a line terminator. The QIO$C SF.SMC function, the MCR SET /ESCSEQ command, or the DCL SET /[NO]ESCAPE command declare the terminal capable of generating escape sequences (see Table 2-7 in Section 2.4.15, and see also Section 2.7).
TF.NOT	Notification of Unsolicited Input - Unsolicited input causes an AST and entry into the AST service routine in the task. When the full-duplex terminal driver receives unsolicited terminal input (except CTRL/C) and you used the TF.NOT subfunction with IO.ATA, the resulting AST serves only as notification of unsolicited terminal input; the terminal driver does not pass the character to the task. Upon entry to the AST service routine, the high byte of the first word on the stack identifies the terminal causing the AST (parameter2 in the IO.ATA function). If TF.NOT is specified, after the AST has been affected, the AST becomes "disarmed" until a read request is issued by the task. If TF.NOT is not specified, every unsolicited character causes an AST. Using the TF.NOT subfunction allows a task to monitor more than one terminal for unsolicited input without the need to read each terminal continuously for possible unsolicited input. Note that the TF.NOT subfunction cannot be used with the CTRL/C AST (ast2 in IO.ATA); an unsolicited CTRL/C character flushes the type-ahead buffer.
TF.XCC	Exclude CTRL/C from AST Notification - TF.XCC is for use with the IO.ATA function. When TF.XCC is included in the IO.ATA function, all characters (except CTRL/C) are handled in the manner previously described. CTRL/C marks the beginning of a command line interpreter (CLI) line that is processed by a CLI task, or, if CTRL/C abort is enabled, aborts tasks active at the terminal. None of the characters of CLI input, including the CTRL/C, are sent to the task issuing the function. Note that you can use either ast2 or TF.XCC, but not both in the same QIO request. If both are specified in the request, an IE.SPC error is returned.

2.4.4 QIO$C IO.CCO - Cancel CTRL/O

The QIO$ IO.CCO macro directs the driver to write a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. If CTRL/O is in effect, it is canceled before the write occurs.

IO.CCO is equivalent to IO.WLB!TF.CCO.

The format of the QIO$ IO.CCO macro is as follows:

```
QIO$C IO.CCO [!TF.WAL] ,lun, [efn] ,<stadd,size,vfc>
             [!TF.WBT]     ,  [pri]
                           ,  [isb]
                           ,  [ast]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When the I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
vfc	The vfc parameter normally specifies cursor position. If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

Parameter	Meaning
vfc (Cont.)	However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter. Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.WAL	Write All Characters - During the write-pass-all operation specified by this subfunction (as in IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation. It does not intercept control characters, and it does not keep track of cursor position. Long lines are not wrapped around if input/output wraparound has been selected.
TF.WBT	Break-Through Write - This subfunction instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the break-through write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another break-through write that is in progress. The effect of this is that a CTRL/S can stop break-through write functions. Thus, it may be desirable for tasks to time out on break-through operations. If a read is currently posted, the break-through write proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the break-through write was effected (if the terminal is not in the full-duplex mode). CTRL/O, if in effect, is canceled. An escape sequence that was interrupted is rubbed out. Break-through write may be issued by a privileged task only. (The privileged MCR command BRO (broadcast) uses IO.WBT.)

2.4.5 QIO$C IO.EIO - Extended I/O Functions

The QIO$C IO.EIO macro allows the use of additional I/O subfunctions. The design of the QIO$ macro, as used with the other QIO$ functions, allows a limited number of I/O subfunctions to be implemented. With IO.EIO, the address of an item list buffer (stadd) is contained in the macro statement. The item list buffer contains IO.EIO modifiers (recognizable as subfunctions) and it allows the use of a maximum of two words of I/O subfunction bits. See Figure 2-1, which shows the structure of the Item List 1 buffer for use with TF.RLB, and Figure 2-2, which shows the structure of the Item List 2 buffer for use with TF.WLB.

The QIO$C IO.EIO reads from or writes to a terminal. The modifiers in the item list allow you to modify the nature or operation of that read or write. A read (TF.RLB) subfunction or write (TF.WLB) subfunction must be issued with the IO.EIO function. But both of these subfunctions cannot be executed as a Logical OR together.

NOTE

The IO.EIO function will not work if your terminal has been set as a remote terminal (RT:) to another system. That is, after entering

```
>SET HOST xxxxx
```

and logging into an RT:, the terminal driver will reject a QIO issuing an extended I/O request from the RT:.

The QIO$C IO.EIO macro has either one of the following formats:

```
QIO$C IO.EIO!TF.RLB,lun, [efn] ,<stadd,size>
                       , [pri]
                       , [isb]
                       , [ast]

QIO$C IO.EIO!TF.WLB,lun, [efn] ,<stadd,size>
                       , [pri]
                       , [isb]
                       , [ast]
```

The TF.WLB and TF.RLB subfunctions each allow specific modifiers, which are located in the item list, to be used with them. They are listed as follows:

Subfunction	Modifiers			
TF.RLB	TF.BIN,	TF.RAL,	TF.RDI,	TF.RES,
	TF.RLU,	TF.RNE,	TF.RNF,	TF.RPR,
	TF.RPT,	TF.RST [1],	TF.RTT [1],	TF.TMO,
	TF.TNE,	TF.XOF		
TF.WLB	TF.CCO,	TF.RCU,	TF.WAL,	TF.WBT,
	TF.WIR			

1. If both the TF.RST and TF.RTT modifiers are included, TF.RST supersedes the function of TF.RTT.

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When the I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the item list of the length specified in size. The address of the item list must be word-aligned and in the task's address space.
size	The size of the item list in bytes. The specified size for the IO.EIO!TF.RLB function must be 24 decimal bytes. The specified size for the IO.EIO!TF.WLB function must be 10 decimal bytes. The item list must be within the task's address space.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.BIN	Binary Prompt (send prompt as pass all) - The prompt is sent to the terminal without interpretation by the driver. This is similar, for the prompt, to a write-pass-all operation.
TF.CCO	Cancel CTRL/O - The driver writes a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. If the CTRL/O is in effect, it is canceled before the write occurs.
TF.RAL	Read All Characters (Pass All) - This subfunction allows the passage of all characters to the requesting task. The driver does not intercept control characters. The characteristic TC.8BC, when set, allows the driver to pass 8 bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and are not interpreted by the driver.

Subfunction	Meaning
TF.RCU	Restore Cursor Position - When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.
TF.RDI	Read With Default Input - The default input that you specified in the extended I/O item list is displayed as an input line at the start of the read on the terminal. You may change this line or use it as input to the system. This subfunction is for use with the extended I/O function (IO.EIO) only.
TF.RES	Read With Escape Sequence Processing Enabled - This subfunction enables escape sequence recognition for the read operation in extended I/O; it is effective for one read only.
TF.RLU	Read With Conversion From Lowercase To Uppercase - The task that uses this subfunction gets input in the buffer in uppercase. This subfunction is used with the extended I/O (IO.EIO) function only.
TF.RNE	Read With No Echo - This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while Read With No Echo is in progress.
TF.RNF	Read With No Filter - This subfunction reads and passes through CTRL/U, CTRL/R, and DELETE characters as normal characters. It is for use with the extended I/O (IO.EIO) function only.
TF.RPR	Read After Prompt - This subfunction is for use with the extended I/O (IO.EIO) function only. The TF.RPR subfunction causes a prompt to be sent to the terminal and immediately follows it with a read function at the terminal. The TF.RPR acts as an IO.WLB followed by IO.RLB. However, TF.RPR differs from the combination of those two functions as follows: • System overhead is lower with the TF.RPR because only one QIO$ is processed. • When using the TF.RPR function, there is no "window" during which a response to the prompt may be ignored. Such a window occurs if the task uses IO.WLB followed by an IO.RLB, because no read may be posted at the time the response is received. • If the issuing task is checkpointable, it can be checkpointed during both the prompt and the read requested by the TF.RPR. • A CTRL/O that may be in effect prior to issuing the TF.RPR is canceled before the prompt is written.

Subfunction	Meaning

NOTE

If a TF.RPR function is in progress when the driver receives a CTRL/R or CTRL/U, the prompt is redisplayed.

TF.RPT Read In Pass-Through Mode - This subfunction passes all characters except XON/XOFF. It allows the passage of all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass eight bits instead of seven. The driver intercepts the control characters CTRL/S and CTRL/Q. Other control characters, for example, CTRL/C, CTRL/O, and CTRL/Z, are passed to the task and are not interpreted by the driver. This subfunction modifier is for use with the IO.EIO!TF.RLB function only.

TF.RST Read With Special Terminators - Special characters in the ranges 0-037 and 175-177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT (177) does not erase, and a CTRL/C does not produce a CLI prompt, or abort tasks active at the terminal if CTRL/C abort is enabled. CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators.

TF.RST sets TF.TNE by default, which means that terminators are not echoed on the terminal screen.

If uppercase to lowercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

Exercise great care when using IO.RAL and TF.RST together. Obscure problems can result if you use them in this way.

TF.RTT Read With Specified Terminator Table - This subfunction is for use with the IO.EIO extended I/O function only. Control characters function normally with the TF.RTT subfunction. Terminators echo by default. The additional use of subfunction TF.TNE prevents the echoing of terminators on the terminal screen. If you want to use special control characters as terminators, their normal function should be disabled with the TF.RNF subfunction or the TC.PTH characteristic. The terminator table (a bit mask table) length can be from 1 through 32(decimal) bytes where bit 0 is a null character, bit 1 is a CTRL/A, and so forth. The terminator table address is in the item list of the IO.EIO function. To use ASCII characters 128(decimal) through 255(decimal), the characteristic TC.8BC must be set.

TF.TMO Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time.

Specify the time-out count in seconds. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) intervals.

Subfunction	Meaning
TF.TMO (Cont.)	If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer. If you need more than 255(decimal) seconds, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.
TF.TNE	Read Terminators With No Echo - This subfunction allows reading terminator characters from the terminal without their being echoed on the terminal screen as they are entered. It is for use with the extended I/O function IO.EIO only.
TF.WAL	Write All Characters - During the write-pass-all operation specified by this subfunction (as in IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation. It does not intercept control characters, and it does not keep track of cursor position. Long lines are not wrapped around if input/output wraparound has been selected.
TF.WBT	Break-Through Write - Instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the break-through write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another break-through write that is in progress. The effect of this is that a CTRL/S can stop break-through write functions. Thus, it may be desirable for tasks to time out on break-through operations. If a read is currently posted, the break-through write proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the break-through write was effected (if the terminal is not in the full-duplex mode). CTRL/O, if in effect, is canceled. An escape sequence that was interrupted is deleted. Break-through write may be issued by a privileged task only.
TF.WIR	Write With Input Redisplayed - This subfunction performs a write to the terminal. If a read is in progress at the terminal and you have entered characters in the input line, the prompt and the characters are redisplayed at the end of the write.
TF.XOF	Send XOFF - This subfunction causes the driver to send an XOFF to the terminal after its read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.5.1 Item List 1 for IO.EIO!TF.RLB - Figure 2-1 shows the structure of the Item List 1 buffer. You should use the Item List 1 buffer when you use the TF.RLB function with IO.EIO. Modifier word 2 is currently not used but must be 0. All the other fields in the item list must be present, but need not contain any specific information except what is pertinent to the function being performed. Thus, if a read with prompt (TF.RPR) is not being performed, words 10, 12, and 14 are not used.

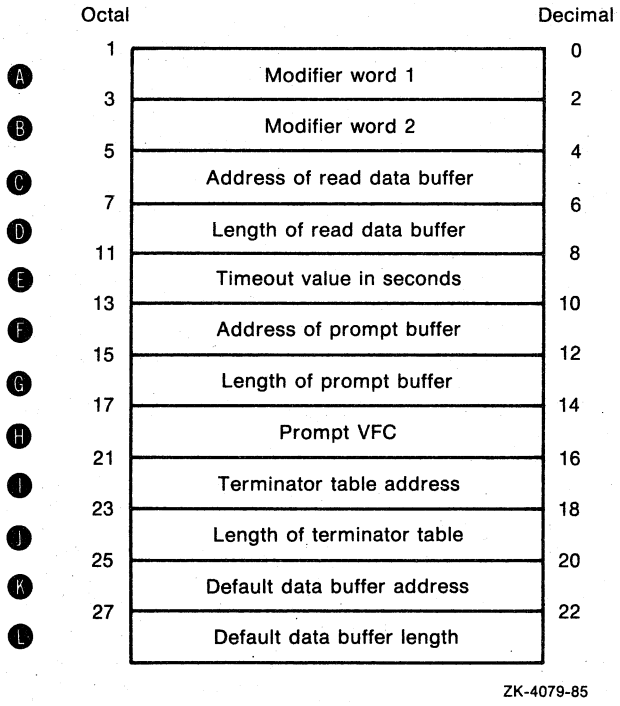


Figure 2-1 Structure of the Item List 1 Buffer

(A) Modifiers (subfunctions) of the group of additional modifiers allowed for any I/O read function.

(B) Currently must be 0.

(C) The starting address of the read data buffer. The read data buffer may be on a byte boundary.

(D) The size of the read data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

(E) For use with TF.TMO. TF.TMO must be in modifier word 1.

(F) For use with TF.RPR and contains the starting address of the prompt buffer. TF.RPR must be in modifier word 1. The prompt buffer may be on a byte boundary.

(G) For use with TF.RPR. The size of the prompt buffer in bytes. The buffer must be within the task's address space. The specified size must be greater than 0 and less than or equal to 8128 bytes.

(H) For use with TF.RPR. The vfc parameter normally specifies cursor position.

If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

(I) For use with TF.RTT. TF.RTT must be in modifier word 1. The table (1 to 32(decimal) bytes) starts at the address specified by the table address. The first word contains bits that represent the first 16 ASCII character codes (0-17); similarly, the second word contains bits that represent the next 16 character codes (20-37), and so forth, through the sixteenth word, bit 15, which represents character code 377. For example, to specify the % symbol (code 045) as a read terminator character, set bit 05 in the third word, because the third word of the table contains bits representing character codes 40-57.

The terminal must be set for read-pass-all operation (TC.BIN=1) or to read-pass 8-bits (TC.8BC) if you want to use any of the following characters as terminator characters:

- CTRL/S (023)
- CTRL/Q (021)
- Any characters whose codes are greater than 177

(J) Length of the terminator table specified in I.

(K) For use with TF.RDI. TF.RDI must be in modifier word 1. This buffer contains the default input that is to be displayed on the terminal.

(L) For use with TF.RDI. This word contains the length of the buffer at the address specified in K.

2.4.5.2 **Item List 2 for IO.EIO!TF.WLB** - You should use the Item List 2 buffer when you use the TF.WLB function with IO.EIO. Modifier word 2 is currently not used but must be 0. All the other fields in the item list must be present. Item list 2 is shown in Figure 2-2.

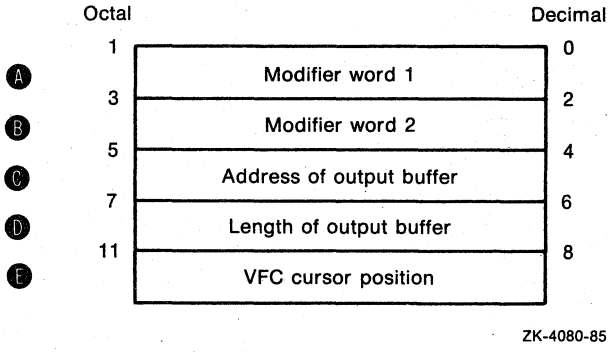


Figure 2-2 Structure of the Item List 2 Buffer

(A) Modifiers (subfunctions) of the group of modifiers allowed for I/O write functions.

(B) Currently must be 0.

(C) The starting address of the write data buffer. The address may be on a byte boundary.

(D) The size of the stadd buffer in an even number of bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

(E) The vfc parameter normally specifies cursor position.

If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

2.4.6 QIO$C IO.GTS - Get Terminal Support

The QIO$C IO.GTS macro returns information to a four-word buffer that specifies which system generation options are part of the terminal driver. Only two of these words are currently defined. Table 2-6 gives details for these words. The IO.GTS function is a system generation option. If IO.GTS is issued on a system without IO.GTS support, IE.IFC is returned in the I/O status block.

The format of the QIO$C IO.GTS macro is as follows:

```
QIO$C IO.GTS,lun,[efn],[pri],[isb],[ast],<stadd,size>
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the data buffer. The address must be word-aligned.
size	The size of the stadd data buffer in bytes. The specified size must be four bytes. The buffer must be within the task's address space. The size must be an even value.

The various symbols used by the IO.GTS, SF.GMC, and SF.SMC functions are defined in a system module, TTSYM. These symbols include F1.xxx and F2.xxx (Table 2-6); T.xxxx (Table 2-8); TC.xxx (Table 2-7); and the SE.xxx status returns described in Table 2-9, Section 2.5. These symbols may be defined locally within a code module by using:

```
.MCALL   TTSYM$
  .
  .
  .
TTSYM$
```

Symbols that are not defined locally are automatically defined by the Task Builder.

Octal values shown for the symbols are subject to change. Therefore, only the symbolic names should be used.

Table 2-6
Information Returned by Get Terminal Support (IO.GTS) QIO$

Bit	Octal Value	Mnemonic	Meaning When Set to 1
Word 0 of Buffer:			
0	1	F1.ACR	Automatic CR/LF on long lines
1	2	F1.BTW	Break-through write
2	4	F1.BUF	Checkpointing during terminal input
3	10	F1.UIA	Unsolicited input-character AST
4	20	F1.CCO	Cancel CTRL/O before writing
5	40	F1.ESQ	Recognize escape sequences in solicited input
6	100	F1.HLD	Hold-screen mode
7	200	F1.LWC	Lowercase to uppercase conversion
8	400	F1.RNE	Read with no echo
9	1000	F1.RPR	Read after prompting
10	2000	F1.RST	Read with special terminators
11	4000	F1.RUB	CRT rubout
12	10000	F1.SYN	CTRL/R terminal synchronization
13	20000	F1.TRW	Read all and write all
14	40000	F1.UTB	Input characters buffered in task's address space
15	100000	F1.VBF	Variable-length terminal buffers
Word 1 of Buffer:			
0	1	F2.SCH	Set characteristics QIO$ (SF.SMC)
1	2	F2.GCH	Get characteristics QIO$ (SF.GMC)
2	4	F2.DCH	Dump/restore characteristics
3	10	F2.DKL	Historical RSX-11D or IAS IO.KIL
4	20	F2.ALT	ALTmode is echoed
5	40	F2.SFF	Form feed can be simulated
6	100	F2.CUP	Cursor positioning
7	200	F2.FDX	Full-duplex terminal driver
8	400	F2.EIO	Extended I/O
9	1000	F2.NCT	Network command terminal support

2.4.7 QIO$C IO.HNG - Disconnect a Terminal

The QIO$C IO.HNG macro disconnects a terminal that is on a remote line or on a DECNET link. This function has no parameters.

A nonprivileged task can issue an IO.HNG request for its own terminal (TI:) only. A privileged task can issue IO.HNG to any terminal.

The format of the QIO$C IO.HNG macro is as follows:

```
QIO$C IO.HNG,lun,[efn],[pri],[isb],[ast]
```

Parameters:

The parameters have the following meaning:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.

2.4.8 QIO$C IO.RAL - Read All Characters Without Interpretation

The QIO$C IO.RAL macro causes the driver to pass all characters that were read to the requesting task. The driver does not intercept control characters. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the program and are not interpreted by the driver.

NOTE

IO.RAL echoes the characters that are read. To read all characters without echoing, use IO.RAL!TF.RNE.

IO.RAL is equivalent to IO.RLB used in a logical OR with the subfunction bit TF.RAL. The IO.RAL function can be terminated only by a full character count (input buffer full).

The format of QIO$C IO.RAL is as follows:

```
QIO$C IO.RAL [!TF.RNE ,lun, [efn  ,<stadd,size,[tmo]>
              !TF.RST     ,  pri
              !TF.TMO     ,  isb
              !TF.XOF     ,  ast]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
tmo	The optional time-out count for use with the TF.TMO subfunction.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.RNE	Read With No Echo - This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while Read With No Echo is in progress.
TF.RST	Read With Special Terminators - Special characters in the ranges 0-037 and 175-177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE (or RUBOUT) does not erase, and a CTRL/C does not produce a CLI prompt or abort tasks active at the terminal if CTRL/C abort is enabled. Also CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators. TF.RST sets TF.TNE by default, which means that terminators are not echoed on the terminal screen. If uppercase to lowercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (0a17, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner. Exercise great care when using IO.RAL and TF.RST together. Obscure problems can result if you use them in this way.
TF.TMO	Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) intervals. If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer. If you need more than 255(decimal) intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.
TF.XOF	Send XOFF - The driver sends an XOFF to the terminal after its read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.9 QIO$C IO.RNE - Read Input Without Echoing

The IO.RNE function reads terminal input characters without echoing the characters back to the terminal for display. You can use this feature when typing sensitive information (for example, a password or combination) or when reading a badge with the RT02-C terminal.

(Note that the no-echo mode can also be selected with the SF.SMC function; see Table 2-7 in Section 2.4.15, bit TC.NEC.)

CTRL/R is ignored while an IO.RNE is in progress.

The IO.RNE function is equivalent to IO.RLB in a logical OR with the subfunction bit TF.RNE.

The format of the QIO$C IO.RNE macro is as follows:

```
QIO$C IO.RNE [!TF.RAL] ,lun, [efn] ,<stadd,size,,[tmo]>
             [!TF.RST]     ,  [pri]
             [!TF.TMO]     ,  [isb]
             [!TF.XOF]     ,  [ast]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
tmo	The optional time-out count for use with the TF.TMO subfunction.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.RAL	Read All Characters (Pass All) - This subfunction allows the driver to pass all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass eight bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and are not interpreted by the driver. Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.
TF.RST	Read With Special Terminators - Special characters in the ranges 0-037 and 175-177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE (or RUBOUT) does not erase, and a CTRL/C does not produce a CLI prompt or abort tasks active at the terminal if CTRL/C abort is enabled. Also CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators. TF.RST sets TF.TNE by default, which means that terminators are not echoed on the terminal screen. If uppercase to lowercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner. Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.
TF.TMO	Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) intervals. If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer. If you need more than 255(decimal) intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.
TF.XOF	Send XOFF - The driver sends an XOFF to the terminal after its read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.10 QIO$C IO.RPR - Send Prompt, Then Issue Read

The QIO$C IO.RPR macro sends a prompt to the terminal and immediately follows it with a read function at the terminal. The IO.RPR functions as an IO.WLB (write a prompt to the terminal) followed by IO.RLB. However, IO.RPR differs from the combination of those two functions as follows:

- System overhead is lower with the IO.RPR because only one QIO$ is processed.
- When using the IO.RPR function, there is no "window" during which a response to the prompt may be ignored. Such a window occurs if the task uses IO.WAL/IO.RLB, because no read may be posted at the time the response is received.
- If the issuing task is checkpointable, it can be checkpointed during both the prompt and the read requested by the IO.RPR.
- A CTRL/O that may be in effect prior to issuing the IO.RPR is canceled before the prompt is written.

Subfunction bits may be excuted as a logical OR with IO.RPR to write the prompt as a "write all" (TF.BIN) and to send XOFF after the read (TF.XOF). In addition, your task can use TF.RAL, TF.RNE, and TF.RST with IO.RPR.

NOTE

If an IO.RPR function is in progress when the driver receives a CTRL/R or CTRL/U, the prompt is redisplayed.

The format of the QIO$C IO.RPR macro is as follows:

```
QIO$C IO.RPR [!TF.BIN  ],lun,[efn],<stadd,size,[tmo],pradd,prsize,vfc>
             [!TF.RAL  ]    ,[pri]
             [!TF.RNE  ]    ,[isb]
             [!TF.RST  ]    ,[ast]
             [!TF.TMO  ]
             [!TF.XOF  ]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.

Parameter	Meaning
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
tmo	The optional time-out count for use with the TF.TMO subfunction.
pradd	The starting address of the byte buffer where the prompt is stored.
prsize	The size of the pradd prompt buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
vfc	The vfc parameter normally specifies cursor position. If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor. However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter. Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.BIN	Binary Prompt (send prompt as pass all) - As used in IO.RPR, results in a "binary" prompt; that is, a prompt is sent to the terminal by the driver with no character interpretation (as if it were issued as an IO.WAL). The read follows the binary prompt.

Subfunction	Meaning
TF.RAL	Read All Characters (Pass All) - The driver passes all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass eight bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and not interpreted by the driver. Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.
TF.RNE	Read With No Echo - This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while Read With No Echo is in progress.
TF.RST	Read With Special Terminators - Special characters in the ranges 0-037 and 175-177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE (or RUBOUT) does not erase, and a CTRL/C does not produce a CLI prompt or abort tasks active at the terminal if CTRL/C abort is enabled. Also CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators.
TF.RST	TF.RST sets TF.TNE by default, which means that terminators are not echoed on the terminal screen. If uppercase to lowercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner. Exercise great care when using TF.RAL and TF.RST together. Obscure problems can result if you use them in this way.
TF.TMO	Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) intervals. If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer. If you need more than 255(decimal) intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.
TF.XOF	Send XOFF - The driver sends an XOFF to the terminal after its prompt-and-read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.11 QIO$C IO.RST - Read Logical Block With Special Terminators

A QIO$C IO.RST reads a block of data from the terminal. This function is equivalent to an IO.RLB!TF.RST. Certain special characters in the ranges 0-037 and 175-177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT (177) does not erase, and a CTRL/C does not produce a CLI prompt or abort tasks active at the terminal if CTRL/C abort is enabled. Also CTRL/U and CTRL/R do not perform their usual functions. All control characters are terminators.

IO.RST sets TF.TNE by default, which means that terminators are not echoed on the terminal screen.

If uppercase to lowercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

Upon successful completion of an IO.RST request that was not terminated by filling the input buffer, the first word of the I/O status block contains the terminating character in the high byte and the IS.SUC status code in the low byte. The second word contains the number of bytes contained in a buffer. The terminating character is not put in the buffer.

The format of QIO$C IO.RST is as follows:

```
QIO$C IO.RST [!TF.RAL ] ,lun,[pri],[isb],[ast]
             [!TF.RNE ]
             [!TF.TMO ]
             [!TF.XOF ]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.

Parameter	Meaning
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
tmo	The optional time-out count for use with the TF.TMO subfunction.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.RAL	Read All Characters (Pass All) - The driver passes all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass eight bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and not interpreted by the driver. Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.
TF.RNE	Read With No Echo - This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while Read With No Echo is in progress.
TF.TMO	Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) intervals. If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer. If you need more than 255(decimal) intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.
TF.XOF	Send XOFF - The driver sends an XOFF to the terminal after its read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.12 QIO$ IO.RTT - Read With Terminator Table

The QIO$C IO.RTT macro reads characters like the QIO$C IO.RLB macro, except that a character that you have previously specified terminates the read operation. The specified character's code can range from 0 through 377(octal). You can specify it by setting a bit in a 16-word table, which you specify, that corresponds to the desired character. Multiple characters can be specified by setting their corresponding value.

The 16-word table starts at the address specified by the table parameter. The first word contains bits that represent the first 16 ASCII character codes (0-17); similarly, the second word contains bits that represent the next 16 character codes (20-37), and so forth, through the sixteenth word, bit 15, which represents character code 377. For example, to specify the % symbol (code 045) as a read terminator character, set bit 05 in the third word, because the third word of the table contains bits representing character codes 40-57.

If you want to use the CTRL/S (023), CTRL/Q (021), or any characters greater than 177 as the terminator characters, the terminal must be set to allow a read-pass-all operation (TC.BIN=1), or read-pass eight bits (TC.8BC), as listed in Table 2-7 in Section 2.4.15.

The optional time-out count parameter may be included as desired.

The format of QIO$C IO.RTT is as follows:

```
QIO$C IO.RTT [!TF.RAL]  ,lun, [efn]  ,<stadd,size,[tmo],table>
             [!TF.RCU]       ,  [pri]
             [!TF.RNE]       ,  [isb]
             [!TF.TMO]       ,  [ast]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.

Parameter	Meaning
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
tmo	The optional time-out count for use with the TF.TMO subfunction.
table	The address of the 16-word user-specified terminator table that you create in your task.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.RAL	Read All Characters (Pass All) - The driver passes all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass eight bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and not interpreted by the driver. Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.
TF.RCU	Restore Cursor Position - When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.
TF.RNE	Read With No Echo - Reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while Read With No Echo is in progress.
TF.TMO	Read With Timeout - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) intervals. If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer. If you need more than 255(decimal) intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.

2.4.13 QIO$C IO.WAL - Write a Logical Block and Pass All Characters

The QIO$C IO.WAL macro causes the driver to pass all output from the buffer without interpretation. It does not intercept control characters. Long lines are not wrapped around if input/output wraparound has been selected.

IO.WAL is equivalent to the IO.WLB!TF.WAL function.

The format of the QIO$C IO.WAL macro is as follows:

```
QIO$C IO.WAL [!TF.CCO] ,lun, [efn] ,<stadd,size,vfc>
             [!TF.RCU]     ,  pri
             [!TF.WBT]     ,  isb
                           ,  ast
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
vfc	The vfc parameter normally specifies cursor position. If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor. However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can

Parameter	Meaning
vfc (Cont.)	use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.CCO	Cancel CTRL/O - The driver writes a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. The CTRL/O, if in effect, is canceled before the write occurs.

During a write-pass-all operation (IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation; it does not keep track of cursor position.

TF.RCU Restore Cursor Position - When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.

During a write-pass-all operation (IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation; it does not keep track of cursor position.

TF.WBT Break-Through Write - This subfunction instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the break-through write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another break-through write that is in progress. The effect of this is that a CTRL/S can stop break-through write functions. Thus, it may be desirable for tasks to time out on break-through write operations.

If a read is currently posted, the break-through write proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the break-through write was effected (if the terminal is not in the full-duplex mode).

CTRL/O, if in effect, is canceled.

An escape sequence that was interrupted is deleted.

Break-through write may be issued by a privileged task only.

During a write-pass-all operation, (IO.WAL or IO.WLB!TF.WAL) the terminal driver outputs characters without interpretation; it does not keep track of cursor position.

2.4.14 QIO$C IO.WBT - Break Through to Write a Logical Block

The QIO$C IO.WBT macro instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If an IO.WBT function is issued on a system that does not support IO.WBT, it is treated as an IO.WLB function.

- If another write function is currently in progress, it finishes the current request and the IO.WBT is the next write issued. The effect of this is that a CTRL/S can stop IO.WBT functions. Therefore, it may be desirable for tasks to time out on IO.WBT operations.
- If a read is currently posted, the IO.WBT proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the break-through write was effected (if the terminal is not in the full-duplex mode).
- If CTRL/O is in effect, it is canceled.
- An escape sequence that was interrupted is deleted.

An IO.WBT function cannot break through another IO.WBT that is in progress.

Break-through write may be issued by a privileged task only. The privileged MCR command BRO (broadcast) uses IO.WBT.

The format of the QIO$C IO.WBT macro is as follows:

```
QIO$C IO.WBT [!TF.CCO]  ,lun, [efn]  ,<stadd,size,vfc>
             [!TF.RCU]       ,  [pri]
             [!TF.WAL]       ,  [isb]
                             ,  [ast]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.

Parameter	Meaning
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.
vfc	The vfc parameter normally specifies cursor position. If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor. However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter. Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.CCO	Cancel CTRL/O - The driver writes a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. If the CTRL/O is in effect, it is canceled before the write occurs. The IO.WBT function implies the subfunction TF.CCO, therefore using IO.WBT!TF.CCO is redundant.
TF.RCU	Restore Cursor Position - When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.
TF.WAL	Write All Characters - During a write-pass-all operation (as in IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation. It does not intercept control characters, and it does not keep track of cursor position. Long lines are not wrapped around if input/output wraparound has been selected.

2.4.15 QIO$C SF.GMC - Get Multiple Characteristics

The QIO$ SF.GMC macro returns terminal information into a specified buffer. Table 2-7 in this section shows the terminal characteristics that can be obtained with the QIO$ SF.GMC macro and set with the QIO$ SF.SMC macro.

The format of the QIO$C SF.GMC macro is as follows:

```
QIO$C SF.GMC,lun,[efn],[pri],[isb],[ast],<stadd,size>
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1, but consider the following exception. For SF.GMC, the contents of the I/O Status Block (ISB) is different from that described in Chapter 1. The first word of the status block is the same as that described in Chapter 1. However, the second word is not the same. For SF.GMC or SF.SMC the second word contains the number of bytes in the specified user buffer that were sucessfully processed. For example, if you have a characteristic in the buffer that caused an error, ISB+2 (the second word) will contain the offset to the characteristic.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When the I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.
stadd	The starting address of a data buffer of length "size" bytes. Each word in the buffer has the form .BYTE characteristic-name .BYTE 0 **characteristic-name** One of the bit names that is given in Table 2-7. The value returned in the high byte of each byte-pair is 1 if the characteristic is true for the terminal and 0 if it is not true.
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space. For SF.GMC, size must be an even value.

For the TC.TTP characteristic (terminal type), one of the values shown in Table 2-7 is returned in the high byte.

Table 2-7
Terminal Characteristics
for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (if asserted)	MCR Command
TC.ABD	77	Auto-baud detection	SET /ABAUD=TTnn:
TC.ACD	---	Ancillary control driver. Value determined by system manager	--
TC.ACR	24	Wraparound mode	SET /WRAP=TTnn:
TC.ANI	122	ANSI CRT terminal	SET /ANSI=TTnn:
TC.ASP	76	Remote line answer speed. Initial speed over dial-up line.	SET /REMOTE=TTnn:speed
TC.AVO	123	VT100-family terminal display	SET /AVO=TTnn:
TC.BIN	65	Binary input mode (read-pass-all). No characters are interpreted as control characters.	SET /RPA=TTnn:
TC.BLK	42	Terminal is capable of block mode transfers	SET /BLKMOD=TTnn:
TC.CTS	72	Suspend output to terminal Task can cancel an input ^S or get the current state of the terminal with regard to ^S or ^Q. 0 = resume 1 = suspend	--
TC.DEC	124	Digital CRT terminal	SET/DEC=TTnn:
TC.DLU [1]	41	Dial-up line	SET /REMOTE=TTnn:
TC.EDT	125	Terminal performs editing functions	SET /EDIT=TTnn:

1. A program can enable the auto-call feature of the DF03 modem by setting TC.DLU to a value of two. Auto-call allows you to use the terminal to dial out of the computer. (This is in addition to receiving incoming calls.) While in this mode, read and write requests are serviced even when a line is not in use. Consequently, I/O requests do not fail when the line is hung-up, which is the case for remote lines (TC.DLU=1).

(continued on next page)

Table 2-7 (Cont.)
Terminal Characteristics
for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (if asserted)	MCR Command
TC.EPA	40	When TC.PAR is enabled: 0 = odd parity 1 = even parity	--
TC.ESQ	35	Input escape sequence recognition	SET /ESCSEQ=TTnn:
TC.FDX	64	Full-duplex mode	SET /FDX=TTnn:
TC.HFF	17	Hardware form-feed capability (If 0, form-feeds are simulated using TC.LPP.)	SET /FORMFEED=TTnn:
TC.HFL	13	Number of fill characters to insert after a carriage return (0-7=x)(Use a value of 7 for the LA30-S.)	SET /HFILL=TTnn:x
TC.HHT	21	Horizontal tab capability (if 0, horizontal tabs are simulated using spaces.)	SET /HHT=TTnn:
TC.HLD	44	Hold screen mode. Terminal has ability to hold screen. Not supported over net (NCT).	SET /HOLD=TTnn:
TC.HSY	137	Host to terminal synchronization. XOFF sent when resources are low. XON sent when resources are high. XOFF prevents terminal character input. 0 = No flow control 1 = Flow control exerted	SET /HSYNC=TTnn:
TC.ICS	141	Notify of change in type-ahead buffer (input count state)	
TC.ISL	6	Get MUX subline (=0-15) on interface to which user is connected (SF.GMC only).	--
TC.LPP	2	Page length (1-255.=x)	SET /LINES=TTnn:x
TC.MHU	145	Declare modem hangup AST. Specify address of AST activated by lost carrier.	--
TC.NBR	102	Broadcast disabled	SET /NOBRO=TTnn:

(continued on next page)

Table 2-7 (Cont.)
Terminal Characteristics
for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (if asserted)	MCR Command
TC.NEC	47	Echo suppressed	SET /NOECHO=TTnn:
TC.OOB	140	Specify out-of-band characters and whether they are included in the type-ahead buffer, and whether they are to clear the type-ahead buffer.	--
TC.PAR	37	Generate and check parity	SET /PARITY=TTnn:
TC.PPT	147	Terminal has printer port	SET /PRINTERPORT=TTnn:
TC.PRI	51	Terminal is privileged (SF.GMC only)	SET /PRIV=TTnn:
TC.PTH	146	Pass through enable Only CTRL/S and CTRL/Q are honored. 1 = pass through 0 = default; no pass through	SET /PASTHRU=TTnn:
TC.RAT	7	Type-ahead buffer: 0 = 1-character type-ahead 1 = 36-character type-ahead (RSX-11M only)	SET /TYPEAHEAD=TTnn:
TC.RGS	126	Terminal supports ReGIS instructions	SET /REGIS=TTnn.
TC.RSP	3	Receiver speed (bits-per-second)	SET /SPEED=TTnn:rcv:xmit
TC.SCP	12	Terminal is a scope (CRT)	SET /CRT=TTnn:
TC.SFC	131	Terminal supports soft character set	SET /SOFT=TTnn:
TC.SLV	50	No unsolicited input is accepted	SET /SLAVE=TTnn:
TC.SMR	25	Uppercase conversion disabled	SET /LOWER=TTnn:
TC.SSC	142	Specify terminal management switch characters. These cause a switch from normal mode to terminal management mode.	--
TC.TBF	71	Type-ahead buffer count obtained by SF.GMC. Cleared by SF.SMC.	--

(continued on next page)

Table 2-7 (Cont.)
Terminal Characteristics
for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (if asserted)	MCR Command
TC.TBM	101	Type-ahead buffer mode 0=task type-ahead 1=CLI type-ahead	SET /SERIAL=TTnn:
TC.TBS	100	Type-ahead buffer size (0-255=x) (RSX-11M-PLUS I/D systems only)	SET /TYPEAHEAD=TTnn:x
TC.TLC	130	CLI gets CTRL/C notification	SET /CTRLC=TTnn:
TC.TMM	143	In terminal management mode. Set when switch characters have been detected and terminal management mode is active. Cleared by QIO$ SF.SMC. 1 = In terminal management mode 0 = Exit terminal management mode	--
TC.TSY	144	Output flow control. Allows input XON or XOFF to function. XOFF prevents output from the terminal. 0 = XON/XOFF ignored 1 = default; process XON/XOFF	SET /TTSYNC=TTnn:
TC.TTP	10	Terminal type (=0-255.=x)	SET /X=TTnn: SET /TERM=TTnn:x
TC.VFL	14	Send four fill characters after line feed for vetical forms control.	SET /VFILL=TTnn:
TC.WID [2]	1	Page width (=1-255.=x)	SET /BUF=TTnn:x
TC.XSP	4	Transmitter speed (bits-per-second)	SET /SPEED=TTnn:rcv:xmit
TC.8BC	67	Pass eight bits on input, even if not binary input mode (TC.BIN).	SET /EBC=TTnn:

2. Unsolicited input that fills the buffer before a terminator is received is possibly invalid. When this happens, the driver discards the input by simulating a CTRL/U and echoing ^U.

In Table 2-8, the octal values 0-177 are reserved by DIGITAL. Values 200-377 are available for customer use to define non-DIGITAL terminals. The implicit characteristics shown are set by the driver. Values not shown are not automatically set by the driver. An "unknown" terminal type has no implicit characteristics.

Table 2-8
Bit TC.TTP (Terminal Type) Values
Set by SF.SMC and Returned by SF.GMC

Octal Value TC.SCP	Symbol	Terminal Type	Implicit Characteristics					
			TC.LPP	TC.WID	TC.HFF	TC.HHT	TC.HFL	TC.VFL
0	T.UNK0	Unknown						
1	T.AS33	ASR33	66	72		1		
2	T.KS33	KSR33	66	72		1		
3	T.AS35	ASR35	66	72		1		
4	T.L30S	LA30S	66	80		7		
5	T.L30P	LA30P	66	80				
6	T.LA36	LA36	66	132				
7	T.VT05	VT05	20	72	1		1	1
10	T.VT50	VT50	12	80	1			1
11	T.VT52	VT52	24	80	1			1
12	T.VT55	VT55	24	80	1			1
13	T.VT61	VT61	24	80	1			1
14	T.L180	LA180S	66	132	1			
15	T.V100	VT100	24	80	1			1
16	T.L120	LA120	66	132	1	1		
20	T.LA12	LA12	66	132	1	1		
21	T.L100	LA100	66	132	1	1		
22	T.LA34	LA34	66	132		1		
23	T.LA38	LA38	66	132		1		
24	T.V101	VT101	24	80		1		1
25	T.V102	VT102	24	80		1		1
26	T.V105	VT105	24	80		1		1
27	T.V125	VT125	24	80		1		1
30	T.V131	VT131	24	80		1		1
31	T.V132	VT132	24	80		1		1
32	T.LA50	LA50	66	80	1	1		
33	T.LQP1	LQP01	66	132	1	1		
34	T.LQP2	LQP02	66	132	1	1		
35	T.V2XX	VT2XX	24	80				
37	T.LN03	LN03	66	132	1	1		
40	T.DTC1	DTC01	66	132				

2.4.15.1 **Characteristic Bit Special Information** - The following bits have special, additional information:

- **TC.ASP, TC.HLD** - Effective for VT5x and VT61 only.
- **TC.RSP, TC.XSP, TC.ASP** - The MCR SET /SPEED command requires parameters for both receiver (rcv) and transmitter (xmit) baud rates. (The valid combinations for each are in the RSX-11M/M-PLUS MCR Operations Manual.) The MCR SET /SPEED command format is as follows:

  SET /SPEED=TTnn:rcv:xmit

The list of baud rates in bps and valid MCR SET /SPEED or SET /REMOTE values that may be set is as follows:

TC.ASP TC.RSP or TC.XSP Value	Baud Rate (in bps) and Valid MCR SET Values
S.0	(disabled)
S.50	50 (Baudot codes are not supported)
S.75	75
S.110	110
S.134	134
S.150	150
S.200	200
S.300	300
S.600	600
S.1200	1200
S.1800	1800
S.2000	2000
S.2400	2400
S.3600	3600
S.4800	4800
S.7200	7200
S.9600	9600
S.EXTA	(DH11 external speed A)
S.EXTB	(DH11 external speed B)
S.19.2	19200 (Not available on DZQ11 or DZV11)

Speed can be set only on DH11 and DZ11 controllers. DZV11 and DZQ11 transmitter and receiver speeds must be equal (no split baud rates permitted). Only one value may be specified for the remote answer speed. This value applies to both the transmitter and receiver.

- **TC.TTP** - When the terminal driver reads this bit, the driver sets implicit values for terminal characteristics TC.LPP, TC.WID, TC.HFF, TC.HFL, TC.HHT, TC.VFL, and TC.SCP, as shown in Table 2-7. You can change (override) these values by subsequent IO.SMC requests. In addition, the terminal driver uses TC.TTP to determine cursor positioning commands, as appropriate.

- **TC.CTS** - Returns the present suspend (CTRL/S), resume (CTRL/Q), or suppress (CTRL/O) state set via the SF.SMC function. Values returned are as follows:

Value Returned	State
0	Resume (CTRL/Q)
1	Suspend (CTRL/S)
2	Suppress (CTRL/O)
3	Both suppress and suspend

When a value of 0 is used with the SF.SMC function, the suspend state is cleared; a value of 1 selects the suspend state.

- **TC.TBF** - Returns the number of unprocessed characters in the type-ahead buffer for the specified terminal. This allows tasks to determine whether any characters were typed that did not require AST processing. In addition, you can use the value returned to read the exact number of characters typed, rather than a typical value of 80(decimal) or 132(decimal) characters for the terminal. Please note the following three items when attempting to use the number returned by TC.TBF:

  1. The task must attach the terminal to receive characters from the type-ahead buffer.

  2. The maximum capacity of the type-ahead buffer is 36(decimal) characters for RSX-11M systems and 255(decimal) characters for RSX-11M-PLUS systems.

  3. Using TC.TBF in an SF.SMC function flushes the type-ahead buffer.

2.4.16 QIO$C SF.SMC - Set Multiple Characteristics

The QIO$C SF.SMC macro enables a task to set and reset the characteristics of a terminal. SF.SMC is the inverse function of SF.GMC (Get Multiple Characteristics).

Table 2-7 in Section 2.4.15 notes the terminal characteristics for both the SF.SMC and the SF.GMC functions.

If the characteristic-name is TC.TTP (terminal type), the octal value that corresponds to the terminal type can have any one of the values listed in Table 2-8.

A nonprivileged task can issue an SF.SMC request for its own terminal (TI:) only. A privileged task can issue SF.SMC to any terminal.

Terminal output can be suspended or resumed (simulated CTRL/S and CTRL/Q, respectively) by specifying an appropriate value for TC.CTS. A value of 0 resumes output and a value of 1 suspends output. Specifying any value for TC.TBF flushes (clears) the type-ahead buffer (forces the type-ahead buffer count to 0).

For SF.SMC, the contents of the I/O Status Block (ISB) is different from that described in Chapter 1. The first word of the status block is the same as that in Chapter 1. However, the second word is not the same. For SF.GMC or SF.SMC the second word contains the number of bytes in the specified user buffer that were sucessfully processed. For example, if you have a characteristic in the buffer that caused an error, ISB+2 (the second word) will contain the offset to the characteristic.

The format of QIO$C SF.SMC is as follows:

```
QIO$C SF.SMC,lun,[efn],[pri],[isb],[ast],<stadd,size>
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	The number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.
pri	Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1; however, the I/O status block used for SF.SMC is different from that described in Chapter 1.
ast	If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.

Parameter	Meaning

stadd The starting address of a buffer of length "size" bytes. The address must be word aligned for SF.SMC. Except for the characteristics TC.MHU, TC.SSC, and TC.OOB, each word in the buffer has the form

```
.BYTE characteristic-name
.BYTE value
```

characteristic-name

One of the symbolic bit names given in Section 2.4.15 (Table 2-7).

value

Either 0 (to clear a given characteristic) or 1 (to set a characteristic).

size The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space. For SF.SMC, size must be an even value.

2.4.16.1 **Processing for TC.MHU, TC.SSC, and TC.OOB** - Three characteristics, TC.MHU, TC.SSC, and TC.OOB, require special processing and buffers. The buffers have the form

```
.BYTE characteristic name
.BYTE reserved
.WORD ...
```

TC.MHU This characteristic declares a modem hangup AST. The buffer required for TC.MHU is shown in Figure 2-3. The buffer must contain the address of an AST that is activated when the terminal driver detects that the carrier has been lost. A zero in word 2 (AST address) clears this characteristic.

The buffer has the format shown in Figure 2-3.
!..tp6.nfl.b.nf.nj

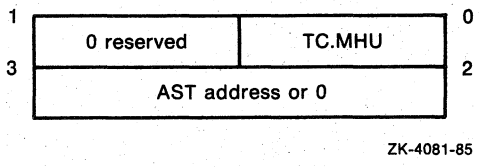


Figure 2-3 Buffer Required for TC.MHU

TC.SSC The characteristic TC.SSC defines or redefines terminal switch characters. The buffer required for TC.SSC is shown in Figure 2-4. The terminal must be attached (IO.ATT) before you set this characteristic. However, the terminal must not be attached for notification of unsolicited input ASTs (IO.ATA).

TC.SSC (Cont.) The buffer has the format shown in Figure 2-4.

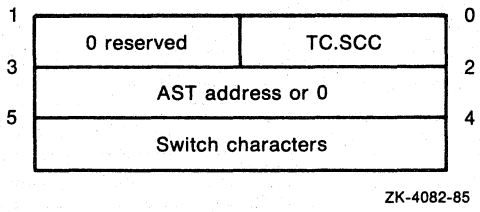


Figure 2-4 Buffer Required for TC.SSC

When the AST address is 0, the switch characters are disabled.

If the terminal is in terminal management mode, both CTRL/C and switch characters are treated as normal data. If the terminal is not currently in terminal management mode and switch characters have been enabled, the terminal driver compares the input characters against the specified switch characters. If there is a match, it cancels any pending read with a status of IS.TMM, flushes the type-ahead buffer, executes the specified AST, and sets the terminal in terminal management mode.

TC.OOB The characteristic TC.OOB defines the out-of-band (OOB) character set for the particular terminal. With this characteristic you can specify certain control characters as out-of-band. To use TC.OOB, the task must attach the terminal and set up the TC.OOB characteristic. After TC.OOB is set, and you enter a specified OOB character at the terminal, the character causes an AST and the typed-in character is on the stack. You specify the AST address when you set up the OOB characteristic.

Additionally, you can declare any of the OOB characters as a "clear OOB character." If the character is declared to be "clear," it clears the type-ahead buffer and terminates a pending read with a status of IS.OOB. Any character that is not a "clear" can be specified as an "include character." Such a character is included in the normal input stream. "Clear OOB" may not be declared as "include."

The buffer required for TC.OOB is shown in Figure 2-5. The terminal must be attached (IO.ATT) before you set this characteristic. However, the terminal must not be attached for notification of unsolicited input ASTs (IO.ATA).

Note the following items before using TC.OOB:

- Because all OOB are either HELLO or CLEAR, one set of bit masks may be used for both. A zero bit mask is a CLEAR. A one bit mask is a HELLO.
- Characters that are CLEAR OOB cannot also be used for INCLUDE OOB.
- To add a character to the OOB set all the characters must be defined, not just the one to be added.

TC.OOB (Cont.) The buffer has the format shown in Figure 2-5.

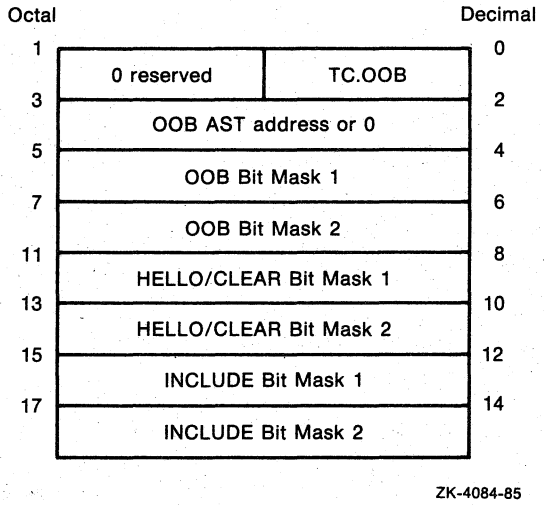


Figure 2-5 Buffer Required for TC.OOB

2.4.16.2 **Side Effects of Setting Characteristics** - Certain terminal characteristics that a task may set or that an operator may set using MCR or DCL commands can have undesirable side effects. In particular, the characteristics hold-screen (TC.HLD), disable lowercase-to-uppercase conversion (TC.SMR), and set switch characters (TC.SSC) can have some undesirable or unexpected side effects. Their effects are described as follows.

TC.HLD Unexpected behavior can result from a terminal in the hold-screen mode if its reception rate is much greater than its transmission rate. (The DHV11 supports split baud rates.) When it is in the hold-screen mode, the terminal automatically sends a CTRL/S when an output stream is received and the screen is nearly full. Output is resumed -- another screenfull -- when you type SHIFT/SCROLL (the terminal generates CTRL/Q). Thus, no output is lost as a result of scrolling off the screen before you can read it. However, if the terminal's transmission rate is far below its reception rate, some unread output may scroll out of sight before the CTRL/S can be transmitted.

Note that some terminals and interfaces are hardware-buffered. This can cause obscure timing problems for tasks that attempt to invoke the hold-screen mode.

TC.SMR If this characteristic is asserted (lowercase-to-uppercase conversion is disabled), octal characters 175 and 176 are interpreted as "right brace (})" and "tilde (~)," respectively. If TC.SMR is not asserted, these characters are interpreted as an ALTmode (that is, they function as line terminators that do not advance the cursor to a new line).

TC.SSC Setting switch characters disables the normal function of CTRL/C in that it becomes a normal data character. After typing switch characters and entering terminal management mode, switch characters are normal data characters until the terminal driver exits terminal management mode.

TC.SSC (Cont.) After you have entered the first switch character, the terminal driver must wait for the second one before entering terminal management mode. If the second character is not the second switch character, the terminal driver treats both entered characters as normal data characters. Any character or combination of characters entered after the two switch characters are considered data characters.

It is advisable to specify nonordinary characters as switch characters, for example, non-system-specific CTRL-key combinations.

2.5 STATUS RETURNS

Most RSX-11M/M-PLUS error and status codes that are returned are byte values in the status word. For example, the value for IS.SUC is 1, which is in the low byte in the first status word. However, IS.CC, IS.CR, IS.ESC, and IS.ESQ are values in the first word of the status block. When any of these codes are returned, the low byte indicates successful completion, and the high byte shows what type of completion occurred.

To test for one of these word-value return codes, first test the low byte of the first word of the I/O status block for the value IS.SUC. Then, test the full word for IS.CC, IS.CR, IS.ESC, or IS.ESQ. (If the full word is equal to IS.SUC, then its high byte is 0, indicating byte-count termination of the read.)

The "error" return IE.EOF may be considered a successful read, because the characters returned to the task's buffer can be terminated by a CTRL/Z character.

The SF.GMC and SF.SMC functions, as described in Sections 2.4.6 and 2.4.13, return the SE.xxx codes. When any of these codes are returned, the low byte in the first word in the I/O status block contains IE.ABO. The second word in the I/O status block word contains an offset (starting from 0) to the byte in error in the QIO's stadd buffer.

Table 2-9 lists error and status conditions that are returned by the terminal driver to the I/O status block.

Table 2-9
Terminal Status Returns

Code	Reason
IE.ABO	**Operation aborted** The specified I/O operation was canceled by IO.KIL while in progress or while in the I/O queue. The second word of the I/O status block indicates the number of bytes that were put in the buffer before the kill was effected.
IE.BAD	**Bad parameter** The size of the buffer exceeds 8128 bytes.

(continued on next page)

Table 2-9 (Cont.)
Terminal Status Returns

Code	Reason
IE.BCC	**Framing error** A framing error was hardware-detected and returned by the controller. All characters up to (but not including) the erroneous character are in the buffer. This condition can occur if you press the BREAK key on some terminals or if there are hardware problems.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task. IE.DAA indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task. The subfunction bits TF.AST or TF.ESQ have no effect if IO.ATT specified them.
IE.DAO	**Data overrun error** A data overrun error was hardware-detected and returned by the controller. All characters up to (but not including) the erroneous character are in the buffer. This error occurs when a hardware failure or incompatibility causes characters to be received by the controller faster than they can be processed (that is, when an incorrect serial I/O baud rate or format exists).
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	**Device not ready** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: • A time-out occurred on the physical device unit. (That is, an interrupt was lost.) • An attempt was made to perform a function on a remote DHV11 or DZV11 line without carrier present.

(continued on next page)

Table 2-9 (Cont.)
Terminal Status Returns

Code	Reason
IE.EOF	**Successful completion on a read with end-of-file** The line of input read from the terminal was terminated with the end-of-file character CTRL/Z. The second word of the I/O status block contains the number of bytes read before CTRL/Z was seen. The input buffer contains those bytes.
IE.IES	**Invalid escape sequence** An escape sequence was started, but escape-sequence syntax was violated before the sequence was completed (see Section 2.7). The character causing the violation is the last character in the buffer.
IE.IFC	**Illegal function** A function code specified in an I/O request was invalid for terminals, or the function code specified was a system generation option not selected for this system.
IE.NOD	**Buffer allocation failure** System dynamic storage has been depleted, resulting in insufficient space available to allocate an intermediate buffer for an input request or an AST block for an attach request.
IE.OFL	**Device off line** The physical device unit associated with the lun specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration. The physical device unit could have been configured off line.
IE.PES	**Partial escape sequence** An escape sequence was started, but read-buffer space was exhausted before the sequence was completed. See Section 2.7.
IE.PRI	**Privilege violation** A nonprivileged task issued an IO.WBT, directed an SF.SMC to a terminal other than TI:, or attempted to set its privilege bit.

(continued on next page)

Table 2-9 (Cont.)
Terminal Status Returns

Code	Reason
IE.SPC	**Illegal address space** One or more of the following conditions may have occurred: • The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. • You specified a byte count of 0. • You specified an odd or 0 AST address. • You specified TF.XCC and ast2 in the same QIO$ request.
IE.VER	**Character parity error** A parity error was hardware-detected and returned by the controller. All characters up to (but not including) the erroneous character are in the buffer.
IS.CC	**Successful completion on a read** The line of input read from the terminal was terminated by a CTRL/C. The input buffer contains the bytes read.
IS.CR	**Successful completion on a read** The line of input read from the terminal was terminated by a carriage return. The input buffer contains the bytes read.
IS.ESC	**Successful completion on a read** The line of input read from the terminal was terminated by an ALTmode character. The input buffer contains the bytes read.
IS.ESQ	**Successful completion on a read** The line of input read from the terminal was terminated by an escape sequence. The input buffer contains the bytes read and the escape sequence.
IS.PND	**I/O request pending** The operation specified in the QIO$ directive has not yet been executed. The I/O status block is filled with 0s.

(continued on next page)

Table 2-9 (Cont.)
Terminal Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO$ directive was completed successfully. If the operation involved reading or writing, you can examine the second word of the I/O status block to determine the number of bytes processed. The input buffer contains those bytes.
IS.TMO	**Successful completion on a read** The line of input read from the terminal was terminated by a time-out. (TF.TMO was set and the specified time interval was exceeded.) The input buffer contains the bytes read.
SE.ATA	The terminal is attached with AST notification enabled.
SE.BIN	An invalid value for a binary characteristic was used in SF.SMC.
SE.FIX	An attempt was made to change a fixed characteristic in a SF.SMC subfunction request. (For example, an attempt was made to change the unit number.)
SE.IAA	An invalid AST address was specified.
SE.NAT	The terminal is not attached.
SE.NIH	A terminal characteristic other than those listed in Table 2-7 (in Sect 2.4.15) was named in an SF.GMC or SF.SMC request, or a task attempted to assert TC.PRI.
SE.NSC	An attempt was made to change a nonsettable characteristic. This error can occur when an attempt is made to make a local-only line a remote line when the controller does not support remote lines.
SE.SPD	The new speed specified in an SF.SMC subfunction request was not valid for the controller associated with the specified terminal.
SE.UPN	There is not enough pool space for the terminal driver to allocate buffer space.
SE.VAL	The new value specified in an SF.SMC request for the TC.TTP terminal characteristic was not one of those listed in Table 2-7 (in Section 2.4.15).

2.6 CONTROL CHARACTERS AND SPECIAL KEYS

This section describes the meanings of special terminal control characters and keys for an RSX-11M/M-PLUS system. Note that the driver does not recognize control characters and special keys during a Read All request (IO.RAL) or a Read with Special Terminators (IO.RST).

2.6.1 Control Characters

A control character is input from a terminal by holding the control key (CTRL) down while typing one other key. Three of the control characters described in Table 2-10, CTRL/R, CTRL/U, and CTRL/Z, are echoed on the terminal as ^R, ^U, and ^Z, respectively.

Table 2-10
Terminal Control Characters

Character	Meaning
CTRL/C	Typing CTRL/C causes unsolicited input on that terminal to be directed to a command line interpreter, such as MCR. If CTRL/C abort is enabled, CTRL/C aborts tasks active at the terminal. (A command line interpreter is invoked and displays a prompt like MCR; therefore, for this text, the assumption is that MCR is the command line interpreter in use, although the terminal driver responds to other command line interpreters in a similar manner.) The "MCR>" prompt is echoed when the terminal driver is ready to accept an unsolicited MCR command line for input. When the unsolicited input is terminated, the command line is passed to MCR. If the last character typed on the terminal was a CTRL/S (suspend output), CTRL/C restarts suspended output and directs subsequent input to MCR. If the hold-screen mode system generation option has been selected and the terminal is a VT5x or VT61 in hold-screen mode, typing a CTRL/C removes the terminal from hold-screen mode. CTRL/C characters can also be directed to a task if the task has attached a terminal and has specified an unsolicited input character AST (see Section 2.4.3). CTRL/C characters are also passed to a task if you specify a TF.RPT, IO.RAL!TF.RPT or IO.RST function, or if the task has set switch characters for the terminal. NOTE If the terminal driver receives a CTRL/C character during a read operation (except during a read-pass-all operation, during a read with special terminators operation, or when the pass-through terminal characteristic (TC.PTH) has been set), the read operation is terminated, the type-ahead buffer is cleared, and an IS.CC status code is returned to the task.

(continued on next page)

Table 2-10 (Cont.)
Terminal Control Characters

Character	Meaning
CTRL/I	CTRL/I or TAB characters initiate a horizontal tab, and the terminal spaces to the next tab stop. Tabs at every eighth character position are simulated by the terminal driver. CTRL/I or TAB have no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT is enabled. That is, the TAB behaves as an ordinary character.
CTRL/J	CTRL/J is equivalent to a LINE FEED character. CTRL/J has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT is enabled. That is, it behaves as an ordinary character.
CTRL/K	CTRL/K initiates a vertical tab, and the terminal tabs to the next vertical tab stop. For a CRT terminal, four LINE FEEDs are output. CTRL/K has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT are enabled. That is, it behaves as an ordinary character.
CTRL/L	CTRL/L initiates a form feed. If the terminal has hardware form-feed support, the driver echoes ^L. Otherwise, the driver simulates the form feed by outputting enough line feed characters to advance the next character position to the top of the next page. If a CRT terminal is in use, four line feeds are output. CTRL/L has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT are enabled. That is, it behaves as an ordinary character.
CTRL/M	CTRL/M is equivalent to a carriage RETURN character (see Section 2.6.2). CTRL/M has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT are enabled. That is, it behaves as an ordinary character.
CTRL/O	CTRL/O suppresses terminal output except if IO.RAL or TF.RAL is enabled or the pass-through terminal characteristic (TC.PTH) has been set. For attached terminals, CTRL/O remains in effect (output is suppressed) until one of the following occurs: • The terminal is detached. • Another CTRL/O character is typed. • An IO.CCO or IO.WBT function is issued. • Input is entered. • IO.RPR is issued at the terminal. For unattached terminals, CTRL/O suppresses output for only the current output buffer (typically one line).

(continued on next page)

Table 2-10 (Cont.)
Terminal Control Characters

Character	Meaning
CTRL/Q	CTRL/Q resumes terminal output previously suspended by CTRL/S except if IO.RAL or TF.RAL is enabled. This applies only to terminals for which TC.TSY is enabled (XON/XOFF are processed). You can enable TTSYNC with the SET /TTSYNC=TTnn MCR command or by setting the TC.TSY terminal characteristic bit.
CTRL/R	CTRL/R response is a terminal driver feature that can be selected during RSX-11M system generation. CTRL/R functions as a normal character if TF.RNF, IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled. Otherwise, CTRL/R results in a carriage return and line feed being echoed, followed by the incomplete (unprocessed) input line. Any tabs that were input are expanded and the effect of anything deleted is shown. On hardcopy terminals, CTRL/R allows you to verify the effect of a tab or a delete, or both, in an input line. CTRL/R is also useful for CRT terminals when the CRT delete system generation option has been selected (see Section 2.6.2). For example, after deleting the leftmost character on the second displayed line of a wrapped input line, the cursor does not move to the right of the first displayed line. In this case, CTRL/R brings the input line and the cursor back together again.
CTRL/S	CTRL/S suspends terminal output except if IO.RAL or TF.RAL is enabled. (Output can be resumed by typing CTRL/Q or CTRL/C.) This applies only to terminals for which TTSYNC is enabled. You can enable TTSYNC with the SET /TTSYNC=TTnn MCR command or by setting the TC.TSY terminal characteristic bit.
CTRL/U	CTRL/U functions as a normal character if TF.RNF, IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled. Otherwise, typing CTRL/U before typing a line terminator deletes previously typed characters back to the beginning of the line. The system echoes this character as ^U followed by a carriage return and a line feed.
CTRL/X	CTRL/X is treated as a normal character if IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT is enabled. Otherwise, this character clears the type-ahead buffer.
CTRL/Z	CTRL/Z is treated as a normal character if IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled. Otherwise, CTRL/Z indicates an end-of-file for the current terminal input. It notifies MAC, PIP, TKB, and other system tasks that terminal input is complete, allowing the task to exit. The system echoes this character as ^Z, followed by a carriage return and a line feed.

2.6.2 Special Keys

The ESC, RETURN, and DELETE (or RUBOUT) keys have special significance for terminal input. A line can be terminated by the ESC (or ALT) key, RETURN key, or the CTRL/Z characters, or by completely filling the input buffer (that is, by exhausting the byte count before a line terminator is typed). The standard buffer size for a terminal can be determined for a task by issuing a Get LUN Information system directive and examining Word 5 of the buffer. An operator can obtain the same information with the MCR SET /BUF=TI: command.

Table 2-11 describes the special significance of the ESC, RETURN, and DELETE (or RUBOUT) keys.

Table 2-11
Special Terminal Keys

Key	Meaning
ESC	ESC (the escape key) functions as a normal character if IO.RAL, TF.RAL, or TF.RPT are enabled. Otherwise, if escape sequences are not recognized, typing ESC or ALT (the ALTmode key on some terminals) notifies the terminal driver that there is no further input on the current line. This line terminator allows further input on the same line, because the carriage or cursor is not returned to the first column position. If escape sequences are recognized, ESC signals the beginning of an escape sequence. (See Section 2.7.)
RETURN	RETURN functions as a normal character if IO.RAL, TF.RAL, or TF.RPT are enabled. Otherwise, typing RETURN terminates the current line and causes the carriage or cursor to return to the first column on the next line.
DELETE (or RUBOUT)	DELETE or RUBOUT functions as a normal character if TF.RNF (read no filter) is enabled. Otherwise, typing DELETE (or RUBOUT) deletes the last character typed on an input line. Only characters typed since the last line terminator may be deleted. Several characters can be deleted in sequence by typing successive DELETEs or RUBOUTs. For example, on a printing terminal, the first DELETE (or RUBOUT) echoes a backslash (\) followed by the character that has been deleted, even if the terminal is in the no-echo mode. Subsequent DELETEs (or RUBOUTs) cause only the deleted character to be echoed. The next character typed that is not a DELETE or RUBOUT causes another backslash to be printed, followed by the new character. The non-RUBOUT character is not echoed if the terminal is in the no-echo mode; however, a backslash is echoed in response to the first non-RUBOUT character. The following example illustrates rubbing out ABC and then typing CBA: ABC\CBA\CBA

(continued on next page)

Table 2-11 (Cont.)
Special Terminal Keys

Key	Meaning
DELETE (or RUBOUT)	The second backslash is not displayed if a line terminator is typed after rubbing out the characters on a line, as in the following example: ABC\CBA At system generation time, the "CRT rubout" feature can be selected. This feature applies to a terminal only after a SET MCR directive has been issued: SET /CRT=TI: If the CRT DELETE (or RUBOUT) feature was selected, DELETE (or RUBOUT) causes the last typed character (if any) to be removed from the incomplete input line and a backspace-space-backspace sequence of characters for that terminal is echoed. If the last typed character was a tab, enough backspaces are issued to move the cursor to the character position before the tab was typed. If a long input line was split, or "wrapped," by the automatic-carriage-return option, and a DELETE (or RUBOUT) erases the last character of a previous line, the cursor is not moved to the previous line. Your task must use CTRL/R to resynchronize the current display with the contents of the incomplete input line.

2.7 ESCAPE SEQUENCES

Escape sequences are strings of two or more characters beginning with an escape character. In RSX-11M systems, escape sequence support described in this section is an option during system generation. Some terminals generate an escape sequence when a special key is pressed (for example, the FCN key on the VT61). On any terminal, an escape sequence may be generated manually by typing ESC followed by the appropriate characters.

Escape sequences provide a way to pass input to a task without interpretation by the operating system. This could be done with a number of read-all functions, but escape sequences allow input to be read with IO.RLB requests.

2.7.1 Definition of Escape Sequence Format

The format of an escape sequence defined by American National Standard X 3.41 (1974) and used in the VT100 is:

ESC ... F

where:

ESC The introduced control character (033(octal)) that is named escape.

... The intermediate bit combinations that may or may not be present. These characters are bit combination 40(8) to 57(8) inclusive in both 7- and 8-bit environments.

F The final character. F characters are bit combinations 60(8) to 176(8) inclusive in escape sequences in both 7- and 8-bit environments.

The occurrence of a character in the inclusive ranges 0 to 37(octal) is technically an error condition. However, the recovery from the error occurs by immediately executing the function specified by the character and then continuing to execute the escape sequence. The exceptions to continuing the escape sequence execution are:

- The character ESC occurs, aborting the current escape sequence. A new one, starting with the ESC just received, begins.
- The character CTRL/X (30(octal)) or the character CTRL/Z (32(octal)) occurs, aborting the current escape sequence. This is the case with any control character.

There are five exceptions to this general syntax definition; these exceptions are discussed in Section 2.7.5.

2.7.2 Prerequisites

There are prerequisites that must be satisfied before escape sequences can be received by a task. First, the terminal must be declared capable of generating escape sequences. This may be done with the DCL SET command:

```
SET TERM/ESCAPE
```

After the preceding prerequisite is satisfied,, one of the following prerequisites must be met.

1. You must attach the terminal with IO.ATT!TF.ESQ.
2. You must use the TF.RES modifier with the IO.EIO!TF.RLB function.

NOTE

The second method will enable escape character recognition for only the duration of the read function.

If these prerequisites are not satisfied, the ESC character is treated as a line terminator. If these prerequisites are satisfied, your task may use CTRL/SHIFT/O (017 octal) as an ALTmode character. However, this character does not act as an ALTmode from a terminal that cannot generate escape sequences.

An ALTmode is a line terminator that does not cause the cursor to advance to a new line. On terminals that cannot generate escape sequences, the ESCape key acts as an ALTmode. Characters 175 and 176 also function as ALTmodes if the terminal has not been declared lowercase (DCL command SET TERM/LOWERCASE).

2.7.3 Characteristics

Escape sequences always act as line terminators. That is, an input buffer may contain other characters that are not part of an escape sequence, but an escape sequence always comprises the last characters in the buffer.

Escape sequences are not echoed. However, if a non-CRT delete sequence is in progress, it is closed with a backslash when an escape sequence is begun.

Escape sequences are not recognized in unsolicited input streams, or in a read all (subfunction bit TF.RAL).

2.7.4 Escape Sequence Syntax Violations

A violation of the syntax defined in Section 2.7.1 causes the driver to abandon the escape sequence and to return an error (IE.IES).

2.7.4.1 **DELETE or RUBOUT (177)** - The character DELETE or RUBOUT is not legal within an escape sequence. Typing it at any point within an escape sequence causes the entire sequence to be abandoned and deleted from the input buffer. Therefore, use DELETE or RUBOUT to abandon an escape sequence, if desired, once you have begun it.

2.7.4.2 **Control Characters (0-037)** - The reception of any except four characters in the range 0 to 037 (octal) is a syntax violation that terminates the read with an error (IE.IES).

The four control characters that are allowed are: CTRL/Q, CTRL/S, CTRL/X, and CTRL/O. These characters are handled normally by the operating system even when an escape sequence is in progress. For example, entering:

```
ESC CTRL/S A
```

gives:

IOSB

IS.ESQ
2

with the additional effect of turning off the output stream.

2.7.4.3 **Full Buffer** - When an escape sequence is terminated because there is no more buffer space rather than by typing a final character, the error IE.PES is returned. For example, after a task issues an IO.RLB with a buffer length of 2, and you type:

```
ESC ! A
```

the buffer contains "ESC !", and the I/O status block contains:

IOSB	IE.PES
	2

The "A" is treated as unsolicited input.

2.7.5 Exceptions to Escape Sequence Syntax

Five "final characters" that normally terminate an escape sequence are treated as special cases by the terminal driver for use with certain terminals:

```
ESC ?...
ESC O...
ESC P...
ESC Y...
ESC [...
```

Refer to documentation supplied with the specific terminal(s) in use for correct use of escape sequences.

2.8 VERTICAL FORMAT CONTROL

Table 2-12 is a summary of all characters that your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter in the IO.WLB, IO.WVB, IO.WBT, IO.CCO, or IO.RPR functions.

Table 2-12
Vertical Format Control Characters

Octal Value	Character	Meaning
040	blank	Single Space - Outputs one line feed, prints the contents of the buffer, and outputs a carriage return. Normally, printing immediately follows the previously printed line.
060	0	Double Space - Outputs two line feeds, prints the contents of the buffer, and outputs a carriage return. Normally, the buffer contents are printed two lines below the previously printed line.
061	1	Page eject - If the terminal supports FORM FEEDs, outputs a form feed, prints the contents of the buffer, and outputs a carriage return. If the terminal does not support FORM FEEDs, the driver simulates the form-feed character by either outputting four line feeds to a CRT terminal, or by outputting enough line feeds to advance the paper to the top of the next page on a printing terminal.

(continued on next page)

Table 2-12 (Cont.)
Vertical Format Control Characters

Octal Value	Character	Meaning
053	+	Overprint - Prints the contents of the buffer and outputs a carriage return, normally overprinting the previous line.
044	$	Prompting Output - Outputs one line feed and prints the contents of the buffer. This mode of output is used with a terminal on which a prompting message is output and input is then read on the same line.
000	null	Internal Vertical Format - Prints the buffer contents without addition of vertical format control characters. In this mode, more than one line of guaranteed contiguous output can be printed for each I/O request.

All other vertical format control characters are interpreted as blanks (040).

2.9 AUTOMATIC CARRIAGE RETURN

You can set individual terminals for wraparound, as desired, using the MCR SET command

```
>SET /WRAP=TTxx:
```

Once you select wraparound, you can select the column at which wraparound occurs by using the MCR SET command

```
>SET /BUF=TI:n
>
```

Your task can also use the SET /BUF command without an argument to display the current buffer width for a terminal:

```
>SET /BUF=TI:
BUF=TI:00072.
>
```

A task can determine the buffer width by issuing a Get LUN Information directive and examining word 5 returned in the buffer.

After the SET /BUF command has been entered, typing beyond the buffer width results in a carriage return and line feed being output before the next character is echoed. Although you may have typed only one line, it is displayed on two terminal lines.

You can lose track of where you are in the input buffer if wraparound is enabled for your terminal. For example, while deleting text on a wrapped line, the cursor does not back up to the previous line. To resynchronize the cursor with the contents of the incomplete input buffer, type CTRL/R (if this option was selected during system generation).

2.10 FEATURES AVAILABLE BY RSX-11M SYSTEM GENERATION OPTION

A number of terminal-driver features are available as options during system generation. (See the RSX-11M System Generation and Installation Guide.) Features previously discussed that are not repeated in this section include:

- Some device-specific QIO functions (see Section 2.3.2)
- Special keys: CTRL/R (see Section 2.6.1)

  DELETE (or RUBOUT) (see Section 2.6.2)
- Escape sequences (see Section 2.7)

The only remaining feature to be selected during system generation is hardware-unrecoverable input error notification (hard receive error detection), which is described in the following section, and terminal-independent cursor control, which is described in Section 2.16.

2.10.1 Hard Receive Error Detection

All terminal interfaces supported by the full-duplex terminal driver are capable of detecting and flagging hard receive errors. Hard receive errors include framing errors, enable character parity error, and data overrun error.

If the hard receive error detection option (T$$RED) is selected during system generation, the driver handles hard receive errors as follows:

1. If a read request is being processed and the character can be processed immediately, the read request is terminated with one of the following error codes returned in the status block:

Error Code	Hard Receive Error
IE.BCC	Framing error
IE.DAO	Data overrun
IE.VER	Character parity error

2. If a command line is being input for a command line interpreter task and the character can be processed immediately, a CTRL/U is simulated, ^U is echoed, and the input is terminated. No command line is sent to the task.
3. If the character would normally cause an AST if no error was detected, the character is ignored and no AST occurs.
4. If the character cannot be processed immediately, it is stored in the type-ahead buffer. A flag is set for the line, indicating that the last character in the type-ahead buffer has an error, disabling further storage in the type-ahead buffer. When the character is retrieved from the buffer, the appropriate action is taken, and the flag is cleared. Any characters received in the meantime are discarded, with a bell echoed for each character.

If the T$$RED option is not selected, hard receive errors are ignored.

2.11 TASK BUFFERING OF RECEIVED CHARACTERS

When task-buffering received characters, characters read from the terminal are sent directly to the task's buffer. Thus, there is no need to allocate a terminal driver buffer.

Task buffering of received characters does not necessarily reduce system overhead. For example, each character must be mapped to the task's buffer. However, if terminal driver buffering was used, the system does the mapping only once for all characters to be transferred.

With the full-duplex terminal driver, output buffering is always performed.

Task buffering is overridden during checkpointing. If a task is checkpointable, a driver buffer is allocated and the task is made eligible for checkpointing by any task, regardless of priority, while the read operation is in progress. (Checkpointing occurs in this situation only when there is another task that can be made active.) Because checkpointability is controlled by the task, you retain control over this operation.

2.12 TYPE-AHEAD BUFFERING

Characters received by the terminal driver are either processed immediately or stored in the type-ahead buffer. The type-ahead buffer allows characters to be temporarily stored and retrieved FIFO. The terminal driver uses the type-ahead buffer as follows:

1. **Store in buffer:**

   An input character is stored in the type-ahead buffer if one or more of the following conditions are true:

   - The driver is not ready to accept the character (fork process pending or in progress).
   - There is at least one character presently in the type-ahead buffer.
   - The character input requires echo, and the output line to the terminal is presently busy outputting a character.
   - No read request is in progress, no unsolicited input AST is specified, and the terminal is either attached or slaved and attached.

   NOTE

   Depending on the terminal mode and the presence of a read function, read subfunctions, and an unsolicited input AST, the CTRL/C, CTRL/O, CTRL/Q, CTRL/S, and CTRL/X characters may be processed immediately and not stored in the type-ahead buffer.

   A character is not echoed when it is stored in the buffer. Echoing a character is deferred until it is retrieved from the buffer, because the read mode (for example, read-without-echo) is not known by the driver until then.

2. **Retrieve from buffer:**

   When the driver becomes ready to process input, or when a task issues a read request, it attempts to retrieve a character from the buffer. If the attempt is successful, the character is processed and echoed, if required. The driver then loops, retrieving and processing characters until either the buffer is empty, the driver becomes unable to process another character, or a read request is finished with the terminal attached.

3. **Flush the buffer:**

   The buffer is flushed (cleared) when:

   - CTRL/C is received.
   - CTRL/X is received.
   - A clear out-of-band character is entered.
   - Switch characters are detected.
   - The terminal becomes detached.
   - TC.TBF is written by SF.SMC.
   - **Exceptions:** CTRL/C and CTRL/X do not flush the buffer if read-pass-all or read-with-special-terminators is in effect.

   If the buffer becomes full, each character that cannot be entered causes a BELL character to be echoed to the terminal.

   If a character is input and echo is required, but the transmitter section is busy with an output request, the input character is held in the type-ahead buffer until output (transmitter) completion occurs.

2.13 FULL-DUPLEX OPERATION

When a terminal line is in the full-duplex mode, the full-duplex driver attempts to service one read request and one write request simultaneously. The IO.ATA, IO.ATT, IO.DET, and SF.SMC functions are performed with the line in an idle state only (not executing a read or a write request).

2.14 PRIVATE BUFFER POOL

The driver has a private buffer pool for intermediate input and output buffers. Whenever the driver needs dynamic memory, it first attempts to allocate a buffer in the private pool. If this fails, it attempts to allocate a buffer in the system pool. If the allocation in the system pool fails during command line input, a CTRL/U is simulated and echoed.

Command line interpreter task buffers are handled in a special way. When unsolicited input begins, a buffer is allocated, as previously described, for the command line (a string of characters, followed by an appropriate terminator character). When the input is completed, the contents of the buffer is sent directly to the command line interpreter task if the buffer was allocated in the system pool. However, if the buffer was allocated in the driver's private pool, it must first be moved into a buffer in the system pool to provide access for the task.

2.15 INTERMEDIATE INPUT AND OUTPUT BUFFERING

Input buffering for checkpointable tasks with checkpointing enabled is provided in the private pool. As each buffer becomes full, a new buffer is automatically allocated and linked to the previous buffer. The Executive then transfers characters from these buffers to the task buffer, and the terminal driver deallocates the buffers once the transfer has been completed.

If the driver fails to allocate the first input buffer, the characters are transferred directly into the task buffer. If the first buffer is successfully allocated, but a subsequent buffer allocation fails, the input request terminates with the error code IE.NOD. In this case, the I/O status block contains the number of characters actually transferred to the task buffer. The task may then update the buffer pointer and byte count and reissue a read request to receive the rest of the data. The type-ahead buffer ensures that no input data is lost.

All terminal output is buffered. As many buffers as required are allocated by the terminal driver and linked to a list. If not enough buffers can be obtained for all output data, the transfer is done as a number of partial transfers, using available buffers for each partial transfer. This is transparent to the requesting task. If no buffers can be allocated, the request terminates with the error code IE.NOD.

The unconditional output buffering serves three purposes:

1. It reduces time spent at interrupt level.
2. It enables long DMA transfers for DH11 controllers.
3. It enables task checkpointing during the transfer to the terminal (if all output fits in one buffer list).

2.16 TERMINAL-INDEPENDENT CURSOR CONTROL

Terminal-independent cursor control capability is provided during system generation. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use. I/O functions associated with cursor positioning are described as follows.

Cursor position is specified in the vfc parameter of the IO.WLB or IO.RPR function. The parameter is interpreted simply as a vfc parameter if the high byte of the parameter is 0. However, if the parameter defines cursor position, the high byte must be nonzero, the low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1/1). Depending on terminal type, the driver sends to the terminal cursor-positioning commands appropriate for the terminal in use that move the cursor to

the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

When defining cursor position in an IO.WLB function, you can use the TF.RCU subfunction to save the current cursor position. When included in this manner, TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.

2.17 TERMINAL INTERFACES

This section summarizes the characteristics of the standard communication-line interfaces supported by RSX-11M. Refer to the Digital Terminals and Printers Handbook for additional details.

2.17.1 DH11 Asynchronous Serial Line Multiplexer

The DH11 multiplexer interfaces up to 16 asynchronous serial communications lines for terminal use. The DH11 supports programmable baud rates. Input and output baud rates may differ; the input rate may be set to 0 baud, thus effectively turning off the terminal. The DM11-BB option may be included to provide modem control for dial-up lines. These lines must be interfaced by a full duplex modem (for example, in the United States, a Bell 103A or equivalent modem).

2.17.2 DHV11 Asynchronous Serial Line Multiplexer

The DHV11 multiplexer interfaces up to eight asynchronous serial communications lines for terminal use. This multiplexer is the Q-BUS version of the DH11 UNIBUS multiplexer. The DHV11 supports programmable baud rates with the option of selecting split speed operation. (Split speed operation allows different transmit and receive speeds.) Also provided is modem control for full-duplex point-to-point operation.

2.17.3 DJ11 Asynchronous Serial Line Multiplexer

The DJ11 multiplexer interfaces as many as 16 asynchronous serial lines to the PDP-11 for local terminal communications. The DJ11 does not provide a dial-up capability. Baud rates are jumper selectable.

2.17.4 DL11 Asynchronous Serial Line Interface

The DL11 supports a single asynchronous serial line and handles communication between the PDP-11 and a terminal. A number of standard baud rates are available to DL11 users. However, because the DL11 does not have an input silo, baud rates greater than 1200 baud are not recommended. Higher baud rates may cause input characters to be lost.

For hardware design reasons, a DL11 is susceptible to losing receiver-interrupt-enable in its Receiver Status Register. The disabling of the receiver interrupt bit causes the terminal to print output requests but not to respond to input (for example, the terminal does not echo input characters). The terminal driver has no mechanism for recognizing the disabling. Therefore, it cannot recover. The bit must be reset with an MCR command OPEN, the console switch register, or a periodically rescheduled task.

2.17.5 DZ11 Asynchronous Serial Line Multiplexer

The DZ11 multiplexer interfaces up to eight asynchronous serial communication lines for use with terminals. It supports programmable baud rates; however, transmit and receive baud rates must be the same. The DZ11 can control a full-duplex modem in auto-answer mode.

2.18 PROGRAMMING HINTS

The following sections are supplied as additional general information to enhance your use of the full-duplex terminal driver.

2.18.1 Checkpointing During Terminal Input

If checkpointing during terminal input was selected as a system generation option, a checkpointable task is stopped (and therefore eligible to be checkpointed) when trying to read. Therefore, a stratagem such as issuing a read followed by a mark-time does not work. The intent might be to time out the read if input is not received in a reasonable length of time. But the mark-time is not issued until the read completes.

You can circumvent this behavior by disabling checkpointing for the read. This is not a desirable solution because it forces a task to remain in memory during the entire read. This defeats the purpose of selecting the checkpoint-during-terminal-input option.

2.18.2 RT02-C Control Function

Because the screen of an RT02C Badge Reader and Data Entry Terminal holds only one line of information, special care must be taken when sending a control character (for example, vertical tab) to the RT02-C. Use the IO.WAL (write all) function for this purpose.

It is recommended that your task use read without echoing when reading a badge with the RT02-C. Use IO.RAL or IO.RNE functions, followed by the IO.WAL function, to echo the information for display.

2.18.3 Remote DL11-E, DH11, and DZ11 Lines

Before a remote line is answered, the driver clears certain terminal characteristics (see Table 2-7) that may have been set by an MCR command SET, or by an SF.SMC function. The characteristics cleared are: TC.SCP, TC.ESQ, TC.HLD, TC.SMR, TC.NEC, TC.FDX, TC.HFF, TC.HHT, TC.VFL, TC.HFL, TC.TTP, TC.8BC, and TC.BIN. (Clearing TC.TTP means that a terminal type of "unknown" returns in an SF.GMC request.) The TC.ACR characteristic (automatic wraparound) is set. Buffer size is set to 72.

A DZ11 remote line must be declared to be remote before the terminal driver can handle the modem.

2.18.4 Modem Support

The terminal driver supports the following modem control operations:

- Local or remote operation
- Answer speed
- Autobaud speed detection

The characteristics bit that controls local or remote operation is TC.DLU. This bit can be set with the MCR command SET /REMOTE (or SET /NOREMOTE for local operation). The DCL command SET TERMINAL REMOTE (or SET TERMINAL LOCAL) can also be used.

When there is an incoming call on a remote line, the TC.ASP characteristic determines the baud rate for the answering modem.

Split baud rates (different transmit and receive speeds) are not supported for answer speed.

The default answer speed is set during system generation. However, the answer speed can be set on line using the MCR command SET /REMOTE=TTnn:speed. VMR can also be used to set the answer speed.

The terminal driver can determine the speed of the incoming call by sampling the first input character after dial-up for the following speeds:

```
110   1800
150   2400
300   4800
600   9600
1200
```

This is called autobaud speed detection. It is an option that you can select for each line by using the SET /AUTOBAUD command. When you set auto-baud speed detection for a given line, the terminal driver tries to sense the baud speed of the caller when the caller's line is set to remote and a call has been received. The terminal driver detects the baud rate as you press the RETURN key (enter carriage returns) several times when you first establish the remote connection from your terminal to the remote computer. Press the RETURN key until the default RSX prompt (>) is displayed.

The SET /ABAUD command sets the TC.ABD terminal characteristic.

CHAPTER 3

HALF-DUPLEX TERMINAL DRIVER

3.1 INTRODUCTION

The half-duplex terminal driver provides support for a variety of terminal devices under RSX-11M. (This terminal driver is not supported on RSX-11M-PLUS systems.) The half-duplex terminal driver generally is used in RSX-11M systems where small driver size is essential, and the additional functional capability provided by the larger full-duplex terminal driver (described in Chapter 2) is not required. Table 3-1 summarizes the terminals supported, and subsequent sections describe these devices in greater detail.

Table 3-1
Supported Terminal Devices

Model	Columns	Lines/ Screen [1]	Character Set	Baud Range	Upper- & Lowercase Send	Receive
ASR-33/35	72		64	110		
KSR-33/35	72		64	110		
LA12	132		96	50-9600	yes	yes
LA100	132		96	110-9600	yes	yes
LA30-P	80		64	300		
LA30-S	80		64	110-300		
LA34	132		96	110-300	yes	yes
LA36	80-132		64-96	110-300	yes	yes [2]
LA38	132		96	110-300	yes	yes
LA120	132		96	50-9600	yes	yes
LA180S	132		96	300-9600		yes
RT02	64	1	64	110-1200		
RT02-C	64	1	64	110-1200		
VT05B	72	20	64	110-2400	yes	
VT50	80	12	64	110-9600		
VT50H	80	12	64	110-9600		
VT52	80	24	96	110-9600	yes	yes
VT55	80	24	96	110-9600	yes	yes
VT61	80	24	96	110-9600	yes	yes
VT100	80-132	24	96	50-9600	yes	yes
VT101	80-132	24	96	50-19200	yes	yes
VT102	80-132	24	96	50-9600	yes	yes
VT105	80-132	24	96	50-9200	yes	yes
VT125	80-132	24	96	50-9600	yes	yes
VT131	80-132	24	96	50-19200	yes	yes
VT132	80-132	24	96	50-19200	yes	yes

1. Applies only to video terminals.

2. Only for 96-character terminal.

The terminal driver supports the terminal interfaces summarized in Table 3-2. These interfaces are described in greater detail in Section 3.9. Programming is identical for all.

Table 3-2
Standard Terminal Interfaces

Model	Type
DH11	16-line multiplexer [1]
DH11-DM11-BB	16-line multiplexer with modem control [2]
DJ11	16-line multiplexer
DL11-A/B/C/D/W	Single-line interfaces
DLV11-F	Single-line interface
DZ11	8-line multiplexer with modem control [2]

1. Direct memory access (DMA) not supported.

2. Full-duplex control only. For example, in the USA, a Bell 103A-type modem.

Terminal input lines can have a maximum length of 255 bytes (the maximum is set in the system generation dialog). The extra characters of an input line that exceeds the maximum length generally become an unsolicited input line.

3.1.1 ASR-33/35 Teletypewriters

The ASR-33 and ASR-35 teletypewriters are asynchronous, hard-copy terminals. No paper tape reader or punch capability is supported.

3.1.2 KSR-33/35 Teletypewriters

The KSR-33 and KSR-35 teletypewriters are asynchronous, hard-copy terminals.

3.1.3 LA30 DECwriters

The LA30 DECwriter is an asynchronous, hard-copy terminal that is capable of producing an original and one copy. The LA30-P is a parallel model and the LA30-S is a serial model.

3.1.4 LA36 DECwriter

The LA36 DECwriter is an asynchronous terminal that produces hard copy and operates in serial mode. It has an impact printer capable of generating multipart and special preprinted forms. The LA36 can receive and transmit both uppercase and lowercase characters.

3.1.5 LA120 DECwriter

The LA120 DECwriter is a hard-copy, uppercase and lowercase terminal capable of printing multipart forms at speeds up to 180 characters-per-second. Serial communications speed is selected from 14 baud rates ranging from 50 to 9600 bps. Hardware features allow bidirectional printing for maximum printing speed, and also allow user-selected features, including font size, line spacing, tabs, margins, and forms control. These functions can also be set up by your tasks that issue appropriate ANSI-standard escape sequences.

3.1.6 LA180S DECprinter

The LA180S DECprinter is a serial version of the LA180. It is a print-only device (it has no keyboard) that can generate multipart forms. The LA180S can print uppercase and lowercase letters.

3.1.7 RT02 Alphanumeric Display Terminal and RT02-C Badge Reader/Alphanumeric Display Terminal

The RT02 is a compact, alphanumeric display terminal for applications in which source data is primarily numeric. A shift key permits the entry of 30 discrete characters, including uppercase alphabetic characters. The RT02 can, however, receive and display 64 characters.

The RT02-C model also contains a badge reader. This option provides a reliable method of identifying and controlling access to the PDP-11 or to a secure facility. Furthermore, data in a format corresponding to that of a badge (22-column fixed data) can be entered quickly.

3.1.8 VT05B Alphanumeric Display Terminal

The VT05B is an alphanumeric display terminal that consists of a CRT display and a self-contained keyboard. From a programming point of view, it is equivalent to other terminals, except that the VT05B offers direct cursor addressing.

3.1.9 VT50 Alphanumeric Display Terminal

The VT50 is an alphanumeric display terminal that consists of a CRT display and a keyboard. It is similar to the VT05B in operation, but does not offer direct cursor addressing.

3.1.10 VT50H Alphanumeric Display Terminal

The VT50H is an alphanumeric display terminal with CRT display, keyboard, and numeric pad. It offers direct cursor addressing. (The VT50H's direct cursor addressing is not compatible with that of the VT05B.)

3.1.11 VT52 Alphanumeric Display Terminal

The VT52 is an uppercase and lowercase alphanumeric terminal with numeric pad and direct cursor addressing. (The VT52's direct cursor addressing is compatible with that of the VT50H, but not with that of the VT05B.) The VT52 can be configured with a built-in thermal printer.

3.1.12 VT55 Graphics Display Terminal

The VT55 is similar to the VT52 in its operation as an alphanumeric terminal. The VT55 offers graphics display features that are not supported by RSX-11M, although the system allows a knowledgeable task to access the explicitly special features of the VT55.

3.1.13 VT61 Alphanumeric Display Terminal

The VT61 is an "intelligent" uppercase and lowercase alphanumeric terminal with an integral microprocessor. It offers two 128-member character sets and numerous built-in functions for editing and preparing forms, as well as a block-transfer mode. (None of these special features is supported by RSX-11M.)

3.1.14 VT100 DECscope

The VT100 DECscope is an uppercase and lowercase alphanumeric keyboard/video display terminal. It is capable of displaying 24 lines of 80 characters (each line). Serial communications speed is selected from baud rates ranging from 50 to 9600 bps. Hardware features allow you to select display characteristics and functions including smooth scroll, reverse video, and so forth. These functions can also be set up by your tasks that issue appropriate ANSI-standard escape sequences.

3.2 GET LUN INFORMATION MACRO

Words 2 through 5 of the buffer filled by the Get LUN Information (GLUN$) system directive (with the first, second, third, and forth device characteristics words) contain the following information for terminals. For characteristic word 1, a setting of 1 indicates that the described characteristic is true for terminals. Words 3, 4, and 5 of the buffer are terminal specific.

Device Characteristics Word 1:

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	1	Terminal device
3	0	File structured device

Bit	Setting	Meaning
4	0	Single-directory device
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Terminal Dependent Characteristics Word 2:

Bit	Setting	Meaning
0	1	Lowercase to uppercase conversion (1=yes)
1	1	Unit is a VT05B terminal (1=yes)
2	1	Unit is a LA30S terminal (1=yes)
3	1	Unit is a privileged terminal (1=yes)
4	1	MCR command AT. being processed (1=yes)
5	1	Terminal is in hold screen mode (1=yes)
6	1	Unit is a DZ11 (1=yes)
7	1	Unit is a slave terminal (1=yes)
8	1	User logged on terminal (1=yes)
9	1	Unit generates escape sequences (1=yes)
10	1	Unit is a CRT (1=yes)
11	1	Don't echo solicited input (1=yes)
12	1	Unit handles hardware form feeds (1=yes)
13	1	Unit is remote (1=yes)
14	1	Unit is a DJ11 (1=yes)
15	1	Unit is a multiplexer (1=yes)

Terminal Dependent Characteristics Word 3:

Bit	Setting	Meaning
13	1	Uppercase output flag (1=yes)
14	1	Parity generation and checking (1=yes)
13	1	Parity sense (1=odd parity)

Terminal Dependent Characteristics Word 4:

Bit	Setting	Meaning
6	1	Look for carriage return (1=yes)

Terminal dependent characteristics Word 4 (word 5 of the buffer) indicates the default buffer size (the width of the terminal carriage or display screen).

Refer to the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for more information about the GLUN$ directive.

3.3 QIO$ MACRO

Table 3-3 lists the standard and device-specific functions of the QIO macro that are valid for terminals. All device-specific functions are options that may be selected at system generation.

Two device-specific functions, SF.SMC and SF.GMC, have nonstandard function names. These names are for compatibility with IAS.

Table 3-3
Standard and Device-Specific QIO Functions for Terminals

Format	Function
STANDARD FUNCTIONS:	
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	READ logical block (read typed input into buffer)
QIO$C IO.RVB,...,<stadd,size>	READ virtual block (read typed input into buffer)
QIO$C IO.WLB,...,<stadd,size,vfc>	WRITE logical block (print buffer contents)
QIO$C IO.WVB,...,<stadd,size,vfc>	WRITE virtual block

(continued on next page)

Table 3-3 (Cont.)
Standard and Device-Specific QIO Functions for Terminals

Format	Function
DEVICE-SPECIFIC FUNCTIONS (ALL SYSTEM GENERATION OPTIONS):	
QIO$C IO.ATA,...,<ast>	ATTACH device, specify unsolicited-input-character AST
QIO$C IO.CCO,...,<stadd,size,vfc>	CANCEL CTRL/O (if in effect), then write logical block
QIO$C SF.GMC,...,<stadd,size>	GET multiple characteristics
QIO$C IO.GTS,...,<stadd,size>	GET terminal support
QIO$C IO.RAL,...,<stadd,size>	READ logical block, pass all bits
QIO$C IO.RNE,...,<stadd,size>	READ logical block, do not echo
QIO$C IO.RPR,...,<stadd,size, [tmo],pradd,prsize,vfc>	READ logical block after prompt
QIO$C IO.RST,...,<stadd,size>	READ logical block ended by special terminators
QIO$C SF.SMC,...,<stadd,size>	SET multiple characteristics
QIO$C IO.WAL,...,<stadd,size>	WRITE logical block, pass all bits
QIO$C IO.WBT,...,<stadd,size,vfc>	WRITE logical block, break through most I/O conditions at terminal

ast

The entry point for an unsolicited-input-character AST.

pradd

The starting address of the byte buffer where the prompt is stored. The buffer must be within the task's address space.

prsize

The size of the pradd prompt buffer in bytes. If the system supports variable length reads, the buffer size must be greater than 0 and less than or equal to 255. If the system does not support variable length reads, the specified size must be greater than 0 and less than or equal to 80.

size

The size of the stadd data buffer in bytes (must be greater than 0). If the function is a read and the system supports variable-length reads, the size must be less than or equal to 255. Otherwise, the size must be less than or equal to 80. The buffer must be within the task's address space. For SF.GMC, IO.GTS, and SF.SMC, the size must be an even number less than 4065 (decimal). If the function is a write, size can be up to 32K bytes.

stadd

The starting address of the data buffer. The address must be word aligned for SF.GMC, IO.GTS, and SF.SMC; otherwise, stadd may be on a byte boundary.

tmo

An optional time-out count, included for IAS compatibility. If supplied, it is ignored.

vfc

A character for vertical format control from Table 3-11 (see Section 3.7).

3.3.1 Subfunction Bits

Most of the device-specific functions supported by the terminal driver are implemented by way of "subfunction bits." That is, these functions can be invoked by ORing a named bit with some other function. Table 3-4 shows the relationship of the 10 subfunction bits to the standard and device-specific functions.

The 10 subfunction bits, and their octal values, are:

TF.AST	Unsolicited-input-character AST	10
TF.BIN	Binary prompt	2
TF.CCO	Cancel CTRL/O	40
TF.ESQ	Recognize escape sequences	20
TF.RAL	Read all bits	10
TF.RNE	Read with no echo	20
TF.RST	Read with special terminators	1
TF.WAL	Write all bits	10
TF.WBT	Break-through write	100
TF.XOF	Send XOFF	100

The subfunction bits are defined in the system module TTSYM (discussed further in Section 3.3.2.5). The octal values of these entities are subject to change; therefore, it is recommended that you always use the symbolic names. As Table 3-4 shows, 7 of the 10 subfunction bits can be ORed with standard QIO functions to invoke device-specific functions. The remaining three subfunction bits (TF.BIN, TF.ESQ, and TF.XOF) can be ORed with Attach and Read After Prompt QIOs to provide added features, as described in Section 3.3.2.

Of the 10 subfunction bits, you can use 3 with Read QIO functions, 3 with Write functions, 2 with Attach functions, and 5 with Read After Prompt. The breakdown is:

Read	TF.RAL, TF.RNE, TF.RST
Write	TF.CCO, TF.WAL, TF.WBT
Attach	TF.AST, TF.ESQ
Read After Prompt	TF.BIN, TF.XOF, TF.RAL, TF.RNE, TF.RST

If a task invokes a subfunction bit that is not supported on the system, the subfunction bit is ignored, not rejected. For example, if Read with Special Terminators is not selected, either IO.RST or IO.RLB!TF.RST is interpreted as IO.RLB.

The following example shows a QIO request using more than one subfunction bit: a nonechoed read, which may be concluded by a special terminator, after a prompt.

```
QIO$C IO.RPR!TF.RNE!TF.RST,...,<stadd,size,,pradd,prsize,vfc>
```

3.3.2 Details on Device-Specific QIO Functions

All the device-specific functions described in this section are system generation options. All except SF.GMC, IO.RPR, SF.SMC, and IO.GTS can be issued by ORing a particular subfunction bit with another QIO function. These subfunction bits are specified in the text; subfunction bits are described in general in Section 3.3.1.

In addition to the 11 device-specific QIO functions, this section also gives details on the features provided by the 3 subfunction bits TF.ESQ, TF.BIN, and TF.XOF.

3.3.2.1 **IO.ATA** - IO.ATA is a variation of the IO.ATT function. It specifies an asynchronous system trap (AST) to process an unsolicited input character. When called as follows:

```
QIO$C   IO.ATA,...,<ast>
```

this function attaches the terminal and identifies "ast" as the entry point for an unsolicited-input-character AST. Control passes to this address whenever any unsolicited character (other than CTRL/Q, CTRL/S, or CTRL/O) is input. No checking is done on the specific AST address. A bad address is frequently detected only when the Executive tries to transfer control to it and the task crashes.

In particular, CTRL/C is trapped by the task and does not reach MCR. Thus, any task that uses IO.ATA should recognize some input sequence as a request to terminate, because MCR can not be invoked to abort the task in case of difficulty.

Note that this mechanism gets a single character into the system -- not a series of characters. Because the driver must become a fork process to declare an AST, a second character can arrive before the driver can queue an AST for the first character. The buffer for unsolicited input characters, however, is one byte long. Therefore, the terminal driver ignores the second character. This circumstance can occur because of fast input on a busy system or because output is in progress when the characters are received. Thus, neither type-ahead nor full-duplex operations can be simulated perfectly using unsolicited character ASTs.

Table 3-4
Subfunction Bits

FUNCTION	EQUIVALENT FUNCTION	ALLOWED SUBFUNCTION BITS				
Standard		Functions				
IO.ATT		TF.AST	TF.ESQ			
IO.DET						
IO.KIL						
IO.RLB [1]		TF.RAL	TF.RNE	TF.RST		
IO.RVB [2]		TF.RAL	TF.RNE	TF.RST		
IO.WLB		TF.CCO	TF.WAL	TF.WBT		
IO.WVB [2]		TF.CCO	TF.WAL	TF.WBT		
Device-Specific Functions						
IO.ATA	IO.ATT!TF.AST	TF.ESQ				
IO.CCO	IO.WLB!TF.CCO	TF.WAL	TF.WBT			
SF.GMC						
IO.GTS						
IO.RAL [2]	IO.RLB!TF.RAL	TF.RNE	TF.RST			
IO.RNE [2]	IO.RLB!TF.RNE	TF.RAL	TF.RST			
IO.RPR [2]		TF.BIN	TF.RAL	TF.RNE	TF.RST	TF.XOF
IO.RST [2]	IO.RLB!TF.RST	TF.RAL	TF.RNE			
SF.SMC						
IO.WAL	IO.WLB!TF.WAL	TF.CCO	TF.WBT			
IO.WBT	IO.WLB!TF.WBT	TF.CCO	TF.WAL			

1. Exercise great care when using Read All (.RAL) and Read with Special Terminators (.RST) together. Otherwise, obscure problems can result.

2. These subfunction bits are allowed but are not effective. They are stripped off when the read or write virtual is converted to a read or write logical.

At entry, the unsolicited character is the low-order byte of the top word on the stack. Before exiting the AST, be sure to pop that word off the stack; otherwise, the task crashes. In all other respects the AST environment is standard:

SP+10	Event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	Task's directive status word
SP+00	Unsolicited character in low byte

See the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for further details on ASTs. See Section 3.10.10 for hints on ASTs in a multiterminal environment.

IO.ATA is equivalent to IO.ATT ORed with the subfunction bit TF.AST.

3.3.2.2 **IO.ATT!TF.ESQ** - IO.ATT causes the issuing task to attach a terminal and notify the driver that it recognizes escape sequences input from that terminal. Escape sequences are recognized only for solicited input. See Section 3.6 for a discussion of escape sequences.

If the terminal has not been declared capable of generating escape sequences, IO.ATT!TF.ESQ has no effect beyond attaching the terminal. No escape sequences are returned to the task, because any ESC sent by the terminal acts as a line terminator. Use the SF.SMC QIO or the MCR SET /ESCSEQ command to declare the terminal capable of generating escape sequences (see Table 3-5 and Section 3.3.2.12).

3.3.2.3 **IO.CCO** - This write function directs the driver to write to the terminal regardless of a CTRL/O condition that may be in effect. If CTRL/O is in effect, it is canceled before the write is done.

IO.CCO is equivalent to IO.WLB!TF.CCO.

3.3.2.4 **SF.GMC** - SF.GMC returns information on terminal characteristics. Use Get Multiple Characteristics in the following way:

```
QIO$C SF.GMC,...,<stadd,size>
```

stadd

The starting address of a data buffer of length "size" bytes. Each word in the buffer has the form

```
.BYTE  characteristic-name
.BYTE  0
```

characteristic-name

One of the eight bit names given in Table 3-5.

The QIO function returns a value in the high-order byte of each byte-pair: 1 if the characteristic is true for the terminal, 0 if not true.

For the TC.TTP characteristic (terminal type), one of three values is returned in the high-order byte, as shown in Table 3-6.

NOTE

The half-duplex terminal driver treats the terminal type as a required characteristic for the type of terminal specified. The terminal type (TC.TTP) does not set any implicit terminal characteristics other than those noted in Table 3-6.

Table 3-5
Terminal Characteristics for SF.GMC and SF.SMC Requests

Bit Name	Octal Value	Meaning (If Asserted)	Corresponding MCR Command
TC.ASP [3]	76	Remote line answer speed	SET /REMOTE=TI:speed
TC.ESQ [1]	35	Can generate escape sequences	SET /ESCSEQ=TI:
TC.HLD [1]	44	Is in hold-screen mode	SET /HOLD=TI:
TC.NEC [2]	47	Is in no-echo mode	SET /NOECHO=TI:
TC.PRI [2]	51	Is privileged	SET /PRIV=TTnn:
TC.SCP	12	Is a scope (CRT)	SET /CRT=TI:
TC.SLV	50	Is slaved	SET /SLAVE=TTnn:
TC.SMR	25	Uppercase conversion disabled on input	SET /LOWER=TI:
TC.TTP	10	Terminal type	SET /LA30S=TI: SET /VT05B=TI:
TC.HFF [3]	17	Handle hardware form feeds	SET /FORMFEED=TI:
TC.RSP [3]	3	Receiver speed	SET /SPEED=TI:rcv:xmit
TC.XSP [3]	4	Transmitter speed	(As above)

1. Effective for VT5x and VT61 only.

2. Cannot be changed by a task; must use MCR command.

3. Recognized only by the SF.SMC function.

Table 3-6
Bit TC.TTP (Terminal Type): Values Set by SF.SMC and Returned by SF.GMC

Octal Value	Symbolic	Meaning
0	T.UNK0	Terminal type is unknown (resets all other types)
1	T.AS33	Terminal is an ASR (sets uppercase conversion on output)
4	T.L30S	Terminal is an LA30 (sets horizontal fill after carriage return)
7	T.VT05	Terminal is a VT05B (sets a vertical fill count of 4)

3.3.2.5 **IO.GTS** - IO.GTS returns a 4-word buffer of information specifying which optional system generation features are part of the terminal driver. Of these four words, two are currently defined. Table 3-7 gives details on these two words. The IO.GTS QIO is itself a system generation option. If IO.GTS is issued on a minimum system (one with no terminal-driver system generation options), IE.IFC is returned in the I/O status block.

Table 3-7
Information Returned by Get Terminal Support (IO.GTS) QIO

Bit	Value	Mnemonic	Meaning When Set to 1
Word 0 of Buffer:			
0	'1	F1.ACR	Automatic CR/LF on long lines
1	2	F1.BTW	Break-through write
2	4	F1.BUF	Checkpointing during terminal input
3	10	F1.UIA	Unsolicited-input-character AST
4	20	F1.CCO	Cancel CTRL/O before writing
5	40	F1.ESQ	Recognize escape sequences in solicited input
6	100	F1.HLD	Hold-screen mode
7	200	F1.LWC	Lowercase to uppercase conversion
8	400	F1.RNE	Read with no echo
9	1000	F1.RPR	Read after prompting
10	2000	F1.RST	Read with special terminators
11	4000	F1.RUB	CRT rubout
12	10000	F1.SYN	CTRL/R terminal synchronization
13	20000	F1.TRW	Read all and write all
14	40000	F1.UTB	Input characters buffered in task's address space
15	100000	F1.VBF	Variable-length terminal buffers
Word 1 of Buffer:			
0	1	F2.SCH	Set characteristics QIO (SF.SMC)
1	2	F2.GCH	Get characteristics QIO (SF.GMC)

The system module, TTSYM, defines the various symbols used by the IO.GTS, SF.GMC, and SF.SMC QIOs. These symbols include: F1.xxx and F2.xxx (Table 3-7); T.xxxx (Table 3-6); TC.xxx (Table 3-5); and the SE.xxx error returns described in Table 3-8, Section 3.4. You may define these symbols locally within a code module by using:

```
.MCALL   TTSYM$
 .
 .
 .
TTSYM$
```

If the symbols are not defined locally, they are automatically defined by the Task Builder.

The octal values of these symbols are subject to change. Therefore, it is recommended that you always use the symbolic names.

3.3.2.6 **IO.RAL** - IO.RAL causes the driver to pass all bits to the requesting task. The driver does not intercept control characters or mask out the "parity" (high-order) bit. This means, for example, that CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the program and are not interpreted by the driver.

NOTE

IO.RAL echoes the characters that are read. To read all bits without echoing, use IO.RAL!TF.RNE.

IO.RAL is equivalent to IO.RLB ORed with the subfunction bit TF.RAL. The only way to terminate an IO.RAL function is by a character count (that is, filling the input buffer).

3.3.2.7 **IO.RNE** - IO.RNE causes the driver to read a line from the terminal without echoing the characters that are input. This feature is useful when typing sensitive information: for example, a password or combination. You can also use IO.RNE to read a badge with the RT02-C.

(Another way to suppress echoing of input is to set the terminal to no-echo mode with the SF.SMC QIO or the MCR SET /NOECHO command. See Table 3-5, bit TC.NEC.)

Note that the TC.NEC subfunction only suppresses echoing of solicited input. Unsolicited input is still echoed.

CTRL/R, if selected as a system generation option, is ignored while an IO.RNE is in progress.

IO.RNE is equivalent to IO.RLB ORed with the subfunction bit TF.RNE.

3.3.2.8 **IO.RPR** - IO.RPR (Read After Prompt) has the same effect as IO.WLB (to write a prompt to the terminal) followed by IO.RLB. However, IO.RPR differs in four ways from this combination of QIOs. With IO.RPR:

- System overhead is lower because only one QIO is processed.
- There is no "window" during which a response to the prompt may be ignored. Such a window occurs if you use IO.WAL/IO.RLB, because no read may be posted at the time the response is received.
- If the issuing task is checkpointable, it is checkpointed during both the prompt and the read.
- A CTRL/O that may be in effect is canceled before the prompt is written.

The third argument that you can specify to IO.RPR, tmo, is required for compatibility with IAS. If supplied, it is ignored.

Subfunction bits may be ORed with IO.RPR to write the prompt as a Write All (TF.BIN) and to send XOFF after the read (TF.XOF). See the next two sections. In addition, you can use the three Read subfunction bits (TF.RAL, TF.RNE, TF.RST) with IO.RPR.

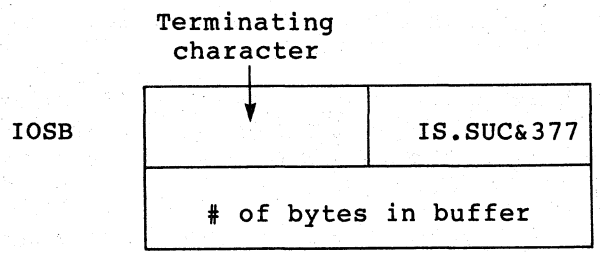
3.3.2.9 **IO.RPR!TF.BIN** - IO.RPR!TF.BIN results in a read after a "binary" prompt, that is, a prompt that is written by the driver to the terminal with no character interpretation (as if it were issued as an IO.WAL).

3.3.2.10 **IO.RPR!TF.XOF** - IO.RPR!TF.XOF causes the driver to send an XOFF to the terminal after its prompt-and-read. The XOFF, or CTRL/S, may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose.

3.3.2.11 **IO.RST** - IO.RST acts like IO.RLB, except that certain special characters terminate the read. These characters are in the ranges 0-37(octal) and 175-177(octal). The driver does not interpret the terminating character, with certain exceptions. For example, a horizontal TAB (11 octal) is not expanded, a RUBOUT (or DEL, 177 octal) does not erase, and a CTRL/C does not get MCR's attention.

If uppercase and lowercase conversion is disabled (see remarks in Section 3.10.9), the character 175(octal) echoes as right-brace and 176(octal) as tilde, and these characters do not act as terminators. The three characters CTRL/O, CTRL/Q, and CTRL/S (17, 21, and 23(octal), respectively) are not special terminators. The driver interprets them as output effectors.

Upon successful completion of an IO.RST request that was not terminated by filling the input buffer, the I/O status block looks like the following:



The terminating character is not in the buffer.

IO.RST is equivalent to IO.RLB!TF.RST.

3.3.2.12 **SF.SMC** - SF.SMC allows a task to set and reset the characteristics of a terminal. Set Multiple Characteristics is the inverse of SF.GMC. Like SF.GMC, it is called in the following way:

```
QIO$C SF.SMC,...,<stadd,size>
```

stadd

The starting address of a buffer of length "size" bytes.

Each word in the buffer has the form

```
.BYTE characteristic-name
.BYTE value
```

characteristic-name

One of the symbolic bit names given in Table 3-5.

value

Either 0 (to clear a given characteristic) or 1 (to set a characteristic). Table 3-5 notes the restrictions that apply to these characteristics.

If characteristic-name is TC.TTP (terminal type), then value can have any of the values listed in Table 3-6.

A nonprivileged task can only issue an SF.SMC request to affect its own terminal, TI0:. A privileged task can issue SF.SMC to any terminal.

3.3.2.13 **IO.WAL** - IO.WAL causes the driver to pass all output from the buffer without interpretation. It does not intercept control characters. Lines are neither wrapped around (if input/output wrap-around has been selected) nor truncated (if wrap-around is not selected).

IO.WAL is equivalent to IO.WLB!TF.WAL.

3.3.2.14 **IO.WBT** - IO.WBT instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If an IO.WBT is issued on a system that does not support IO.WBT, it is treated as an IO.WLB.

- If another write is in progress, it finishes and the IO.WBT is the next write issued. The effect of this is that IO.WBTs can be stopped by a CTRL/S. Therefore, tasks may still want to time out on IO.WBT.
- If a read is posted, the IO.WBT proceeds anyway, and an automatic CTRL/R is performed to redisplay any input that was received before the break-through write.
- CTRL/S or CTRL/O, or both, are canceled if they are in effect.
- Characters input during a break-through write are ignored.

An IO.WBT cannot break through another IO.WBT that is in progress or if a prompt is being written by IO.RPR. In either case, the low-order byte of the first word of the I/O status block contains IE.RSU&377. The task receiving this error need only reissue the write.

Break-through write may only be issued by a privileged task. However, the task does not have to be mapped to the Executive (Task Builder options /PR:4 or /PR:5). A task can use IO.WBT if it is built with the /PR:0 switch specified. The privileged MCR command BRO (broadcast) uses IO.WBT.

Break-through write cannot break through a multiecho. Instead, it returns error code IE.RSU. When this occurs, the task should reissue the write request.

3.4 STATUS RETURNS

Table 3-8 lists error and status conditions that are returned by the terminal driver.

Upon successful completion of a read, the I/O status block contains data of this sort:

		1	0	Byte
Word	0	ret	+1	
	1	Number of bytes read		

ret = 0 means read terminated by buffer full (byte count satisfied);

ret = 15 means IS.CR: read terminated by carriage return.

ret = 33 means IS.ESC: read terminated by an Altmode.

ret = 233 means IS.ESQ: read terminated by an escape sequence.

+1 is IS.SUC: the return code for successful completion.

Most RSX-11M return codes are byte values: for example, IS.SUC = 1 is a byte value. By contrast, the three return codes IS.CR, IS.ESC, and IS.ESQ are word values. The low-order byte indicates successful completion, and the high-order byte is required to show what type of completion occurred.

To test for one of these word-value return codes, first test the low-order byte of the first word of the IOSB for the value IS.SUC. Then test the full word for IS.CR, IS.ESC, or IS.ESQ. (If the full word tests equal to IS.SUC, then its high-order byte is 0, indicating byte-count termination of the read.)

The "error" return IE.EOF may be considered to indicate a successful read, because characters can be returned to the task's buffer.

The three errors in Table 3-8 with SE.xxx codes are returned by the SF.GMC and SF.SMC QIOs. They are characterized by IE.ABO&377 in the low-order byte of the first IOSB word. The high-order byte contains the error code. The second IOSB word contains an offset (starting from 0) to the byte in error in the QIOs stadd buffer.

3.5 CONTROL CHARACTERS AND SPECIAL KEYS

This section describes the meanings of special terminal control characters and keys for RSX-11M. Note that the driver does not recognize control characters and special keys during a Read All request (IO.RAL), and recognizes only some of them during a Read with Special Terminators (IO.RST).

Table 3-8
Terminal Status Returns

Code	Reason
IE.EOF	**Successful completion on a read with end-of-file** The line of input read from the terminal was terminated with the end-of-file character CTRL/Z. The second word of the I/O status block contains the number of bytes read before CTRL/Z was seen. The input buffer contains those bytes.
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. If the operation involved reading or writing, you can examine the second word of the I/O status block to determine the number of bytes processed. The input buffer contains those bytes.
IS.CR	**Successful completion on a read** The line of input read from the terminal was terminated by a carriage return. The input buffer contains the bytes read.
IS.ESC	**Successful completion on a read** The line of input read from the terminal was terminated by an Altmode character. The input buffer contains the bytes read.
IS.ESQ	**Successful completion on a read** The line of input read from the terminal was terminated by an escape sequence. The input buffer contains the bytes read and the escape sequence.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The specified I/O operation was canceled by IO.KIL while in progress or while in the I/O queue. The second word of the IOSB shows how many bytes were processed before the kill took effect. Note that the SE.xxx error codes are characterized by IE.ABO& 377 in the low-order byte of the first word of the IOSB.
IE.BAD	**Bad parameter** The size of the prompt in a read-after-prompt QIO is too big (that is, greater than 255 bytes on systems supporting variable-length buffers or greater than 80 on systems that do not).

(continued on next page)

Table 3-8 (Cont.)
Terminal Status Returns

Code	Reason
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task. If the attach specified TF.AST or TF.ESQ, these subfunction bits have no effect.
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	**Device not ready** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: • A time out occurred on the physical device unit (that is, an interrupt was lost). • An attempt was made to perform a function on a remote DH11 or DZ11 line without carrier present. (The line is hung up.)
IE.IES	**Invalid escape sequence** An escape sequence was started but escape-sequence syntax was violated before the sequence was completed. See Section 3.6.4.
IE.IFC	**Illegal function** A function code specified in an I/O request was illegal for terminals; or, the function code specified was a system generation option not selected for this system.
IE.NOD	**Buffer allocation failure** System dynamic storage has been depleted, and there was insufficient space available to allocate an intermediate buffer for an input request.

(continued on next page)

Table 3-8 (Cont.)
Terminal Status Returns

Code	Reason
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.PES	**Partial escape sequence** An escape sequence was started, but read-buffer space was exhausted before the sequence was completed. See Section 3.6.4.3.
IE.PRI	**Privilege violation** In a multiuser system, a nonprivileged task either issued an IO.WBT or directed an SF.SMC to a terminal other than its own TI0:.
IE.RSU	**Resource in use** The prompt of an IO.RPR, or a break-through write, was in progress when an IO.WBT was issued. Reissue the IO.WBT later.
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of 0 was specified.
SE.BIN	The new value specified for a terminal characteristic in an SF.SMC request was not 0 or 1. (Characteristics other than TC.TTP -- see Table 3-5.)
SE.NIH	A terminal characteristic other than those in Table 3-5 was named in an SF.GMC or SF.SMC request; or, a task attempted to assert TC.PRI.
SE.VAL	The new value specified in an SF.SMC request for the TC.TTP terminal characteristic was not one of those listed in Table 3-6, or the baud rate (speed) specified is not valid.

3.5.1 Control Characters

You enter a control character from a terminal by holding the control key (CTRL) down while typing one other key. Two of the control characters described in Table 3-9, CTRL/U and CTRL/Z, are echoed on the terminal as ^U and ^Z, respectively. Other control characters are recognized by the terminal driver, but are not printing characters and therefore are not echoed.

Table 3-9
Terminal Control Characters

Character	Meaning
CTRL/C	Typing CTRL/C repeatedly is the way to get a terminal's attention. Normally, typing CTRL/C causes unsolicited input on that terminal to be directed to the Monitor Control Routine (MCR). "MCR>" echoes when the terminal is ready to accept unsolicited input. When the unsolicited input completes, it is passed to MCR. If the last item typed on the terminal was CTRL/S (suspend output), then CTRL/C restarts suspended output and directs subsequent input to MCR. If the hold-screen mode option has been selected at system generation time, and if the terminal is a VT5x or VT61 in hold-screen mode, then typing a string of CTRL/Cs eventually removes the terminal from hold-screen mode. Not all CTRL/Cs act to get MCR's attention. CTRL/Cs are directed to a task if the task has attached a terminal and has specified an unsolicited-input-character AST. See the discussion on unsolicited-input-character ASTs, Section 3.3.2.1. CTRL/Cs also go to a task if an IO.RAL (Read All) or IO.RST (Read with Special Terminators) is posted.
CTRL/I	Typing CTRL/I or TAB initiates a horizontal tab, and the terminal spaces to the next tab stop. Tabs at every eighth character position are simulated by the terminal driver.
CTRL/J	Typing CTRL/J is equivalent to typing the LINE FEED key on the terminal.
CTRL/K	Typing CTRL/K initiates a vertical tab, and the terminal performs four line feeds.
CTRL/L	Typing CTRL/L initiates a form feed, and the terminal performs eight line feeds. Paging is not performed.
CTRL/M	Typing CTRL/M is equivalent to typing the carriage RETURN key on the terminal (see Section 3.5.2).

(continued on next page)

Table 3-9 (Cont.)
Terminal Control Characters

Character	Meaning
CTRL/O	Typing CTRL/O suppresses output being sent to a terminal by the current I/O request. For attached terminals, CTRL/O remains in effect, and output continues to be suppressed until any of the following occurs: 1. The terminal is detached. 2. Input is entered. 3. Another CTRL/O character is typed. 4. An IO.CCO, IO.WBT, or IO.RPR is processed. For unattached terminals, CTRL/O suppresses output for only the current output buffer (generally one line).
CTRL/Q	(system generation option.) Typing CTRL/Q resumes terminal output previously suspended by means of CTRL/S.
CTRL/R	(system generation option.) Typing CTRL/R on a terminal results in the echo of CR/LF followed by the incomplete (unprocessed) input line. Any tabs that were input are expanded and the effect of any rubouts is shown. On hard-copy terminals, CTRL/R allows you to verify the effect of a tab or rubout, or both, in an input line. CTRL/R is also useful for CRT terminals when the automatic-cariage-return and CRT rubout system generation options have been selected (see Section 3.8). For example, after rubbing out the leftmost character on the second displayed line of a wrapped input line, you then find that the cursor does not move to the right of the first displayed line. In this case, CTRL/R brings the input line and the cursor back together again.
CTRL/S	(system generation option.) Typing CTRL/S causes terminal output to be suspended. Output is resumed by typing CTRL/Q or CTRL/C.
CTRL/U	Typing CTRL/U before typing a line terminator causes previously typed characters to be deleted back to the beginning of the line. The system echoes this character as ^U followed by a carriage return and a line feed. This allows you to retype the line.
CTRL/Z	Typing CTRL/Z indicates an end-of-file for the current terminal input. It signals MAC, PIP, TKB, and other system tasks that terminal input is complete and the task should exit. The system echoes this character as ^Z followed by a carriage return and a line feed.

3.5.2 Special Keys

The ESCape, carriage RETURN, and RUBOUT keys have special significance for terminal input, as described in Table 3-10. A line can be terminated by an ESCape (or Altmode) character, by a carriage RETURN, by CTRL/Z, or by completely filling the input buffer (that is, by exhausting the byte count before a line terminator is typed). The standard buffer size for a terminal can be determined by issuing a GET LUN INFORMATION system directive and examining Word 5 of the information buffer. Another way is to type the MCR command "SET /BUF=TI:".

3.6 ESCAPE SEQUENCES

Escape sequences are strings of two or more characters beginning with 33 octal. Some terminals generate an escape sequence when a special key is pressed (for example, the PF1 key on the VT100). On any terminal, an escape sequence may be generated manually by typing ESCape and the appropriate following characters.

Escape sequences provide a way to pass input to a task without interpretation by the operating system. This could be done with a number of 1-character Read Alls, but escape sequences allow a neater way to accomplish it (they can be read with ordinary IO.RLBs).

Most DIGITAL software currently does not employ escape sequences. The specifics provided here are for your benefit if you want to take advantage of escape sequences in your tasks.

3.6.1 Definition

An escape sequence is defined as follows:

```
ESC [int] ... [int] fin
```

ESC

The result of pressing the ESCape key, a byte (character) of 33(octal).

int

An "intermediate character" in the range 40(octal) to 57(octal). This range includes the character "space" and 15 punctuation marks. An escape sequence may contain any number of intermediate characters, or none.

fin

A "final character" in the range 60(octal) to 176(octal). This range includes uppercase and lowercase letters, numbers, and 13 punctuation marks.

There are four exceptions to this general definition discussed in Section 3.6.5.

Table 3-10
Special Terminal Keys

Key	Meaning
ESCape	If escape sequences are not recognized, typing ESCape or Altmode signals the terminal driver that there is no further input on the current line. This line terminator allows further input on the same line, because the carriage or cursor is not returned to the first column position. If escape sequences are recognized, ESCape signals the beginning of an escape sequence. See Section 3.6.
RETURN	Typing RETURN terminates the current line and causes the carriage or cursor to return to the first column on the line.
RUBOUT	Typing RUBOUT deletes the last character typed on an input line. Only characters typed since the last line terminator may be deleted. Several characters can be deleted in sequence by typing successive RUBOUTs. The first RUBOUT echoes as a backslash (\), followed by the character that has been deleted. Subsequent RUBOUTs cause only the deleted character to be echoed. The next character typed that is not a RUBOUT causes another backslash, followed by the new character, to be echoed. The following example illustrates rubbing out ABC and then typing CBA: ABC\CBA\CBA The second backslash is not displayed if a line terminator is typed after rubbing out the characters on a line, as in the following: ABC\CBA At system generation time you may elect to support a "CRT rubout" feature. This feature applies to a terminal only after a SET MCR directive has been issued: SET /CRT=TI: (Note: See Section 3.3.2.12 for another way this SET can be accomplished, with the SF.SMC QIO function.) When a RUBOUT is struck, the last typed character (if any) is removed from the incomplete input line and backspace-space-backspace is echoed. If the last typed character was a tab, enough backspaces are issued to move the cursor to the character position before the tab was typed. If a long input line was split, or "wrapped," by the automatic-carriage-return option, and a RUBOUT erases the last character of a previous line, the cursor is not moved to the previous line. You must use CTRL/R to resynchronize the display with the contents of the incomplete input line.

3.6.2 Prerequisites

Two prerequisites must be satisfied before escape sequences can be received by a task.

First, the task must "ask" for them by issuing an IO.ATT and invoking the subfunction bit TF.ESQ.

Second, the terminal must be declared capable of generating escape sequences. This may be done with an MCR SET command:

```
SET /ESCSEQ=TI:
```

An alternative way to tell the driver that the terminal can generate escape sequences is by issuing the Set Multiple Characteristics QIO. See Section 3.3.2.13.

If either of these prerequisites is not satisfied, the ESC character is treated as a line terminator. If both prerequisites are satisfied, then an additional feature results. You may use CTRL/SHIFT/O (37(octal)) as an Altmode. This character does not act as an Altmode from a terminal that cannot generate escape sequences.

An Altmode is a line terminator that does not cause the cursor to advance to a new line. On terminals that cannot generate escape sequences, the ESCape key acts as an Altmode. So do the characters 175(octal) and 176(octal), if the terminal has not been declared lowercase (MCR command SET /LOWER). If the terminal is lowercase, then these characters represent right-brace and tilde, respectively.

3.6.3 Characteristics

Escape sequences always act as line terminators. That is, an input buffer may contain other characters that are not part of an escape sequence, but an escape sequence always comprises the last characters in the buffer.

Escape sequences are not echoed. However, if a non-CRT rubout sequence is in progress, it is closed with a backslash when an escape sequence is begun.

Escape sequences are not recognized in unsolicited input streams. Neither are they recognized in a Read with Special Terminators (subfunction bit TF.RST) nor in a Read All (subfunction bit TF.RAL).

3.6.4 Escape Sequence Syntax Violations

A violation of the syntax defined in Section 3.6.1 causes the driver to abandon the escape sequence and to return an error (IE.IES).

3.6.4.1 **DEL or RUBOUT (177(octal))** - The character DEL or RUBOUT is not legal within an escape sequence. Typing it at any point within an escape sequence causes the entire sequence to be abandoned and deleted from the input buffer. Thus, use DEL or RUBOUT to abandon an escape sequence, if desired, once you have begun it. For example, if you enter:

```
AB ESC " DEL CR
```

the buffer contains "AB" and the I/O status block looks like the following:

IOSB

IS.CR
2

3.6.4.2 **Control Characters (0-37(octal))** - The reception of any character in the range 0 to 37(octal) (with four exceptions -- see Note) is a syntax violation that terminates the read with an error (IE.IES). For example, entering:

```
ESC !  CTRL/SHIFT/O
```

results in a buffer that contains these three characters and an I/O status block that is similar to the following:

IOSB

IE.IES
3

NOTE

Four control characters are allowed: CTRL/Q, CTRL/S, CTRL/C, and CTRL/O. These characters are handled normally by the operating system even when an escape sequence is in progress. For example, entering:

```
ESC CTRL/S A
```

gives:

IOSB

IS.ESQ
2

with the side effect of turning off the output stream.

3.6.4.3 **Full Buffer** - A syntax error results when an escape sequence is terminated by running out of read-buffer space, rather than by reception of a final character. The error IE.PES is returned. For example, after a task issues an IO.RLB QIO with a buffer length of 2, and you type:

```
ESC !  A
```

the buffer contains "ESC !", and the I/O status block contains:

IOSB

IE.PES
2

The "A" is treated as unsolicited input.

3.6.5 Exceptions to Escape Sequence Syntax

Four "final characters" that normally would terminate an escape sequence are treated as special cases by the terminal driver. These special cases exist for historical compatibility reasons. Three of these characters are: ; (73(octal)), ? (77(octal)), and O (117(octal)). The syntax for escape sequences that contain these four characters as intermediates is:

```
ESC ; [int] ... [int] fin

ESC ? [int] ... [int] fin

ESC O [int] ... [int] finl
```

```
int    =    40-57   (octal).
fin    =    60-176  (octal).
finl   =   100-176  (octal).
```

The fourth exception to the general syntax given in Section 3.6.1 involves the "final character" Y (131(octal)). Historically (for example, in the VT52), the use of ESC Y has been to signal the cursor position. It is followed by two numbers signifying column and row positions:

```
ESC Y colpos rowpos
```

where colpos and rowpos are both characters in the range 40-176(octal). They represent bias-40 numbers: colpos = 40 corresponds to column 0, and so forth.

3.7 VERTICAL FORMAT CONTROL

Table 3-11 summarizes the meanings of all characters used for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter in the functions IO.WLB, IO.WVB, IO.WBT, IO.CCO, or IO.RPR.

Table 3-11
Vertical Format Control Characters

Octal Value	Character	Meaning
40	blank	SINGLE SPACE - Output a line feed, print the contents of the buffer, and output a carriage return. Normally, printing immediately follows the previously printed line.
60	0	DOUBLE SPACE - Output two line feeds, print the contents of the buffer, and output a carriage return. Normally, the buffer contents are printed two lines below the previously printed line.
61	1	PAGE EJECT - Output eight line feeds (or, if the terminal is an LA180S, output a form feed), print the contents of the buffer, and output a carriage return.

(continued on next page)

Table 3-11 (Cont.)
Vertical Format Control Characters

Octal Value	Character	Meaning
53	+	OVERPRINT - Print the contents of the buffer and output a carriage return, normally overprinting the previous line.
44	$	PROMPTING OUTPUT - Output a line feed and print the contents of the buffer. This mode of output is for a terminal on which a prompting message is output, and input is then read on the same line.
00	null	INTERNAL VERTICAL FORMAT - Print the buffer contents without addition of vertical format control characters. In this mode, more than one line of guaranteed contiguous output can be printed for each I/O request.

All other vertical format control characters are interpreted as blanks (40(octal)).

3.8 FEATURES AVAILABLE BY SYSTEM GENERATION OPTION

A number of terminal-driver features are available as options at the time you generate an RSX-11 system. See the following manuals as appropriate:

- RSX-11M System Generation and Installation Guide
- RSX-11S System Generation and Installation Guide
- RSX-11M-PLUS System Generation and Installation Guide

Some of the features that were mentioned previously in the text are:

- All the device-specific QIO functions
- Special keys
  - CTRL/S -- Suspend output
  - CTRL/Q -- Resume suspended output
  - CTRL/R -- Write incomplete input buffer
  - CRT rubout
- Escape sequences

Other features that you may select at system generation time are described in the following sections.

3.8.1 Automatic Carriage Return

By system generation, all terminals in a system may be set to "wrap around," on input and output, after a specified number of columns. If this option is selected, the number of characters per line is determined on a terminal-by-terminal basis. Use the MCR SET command to specify the wrap-around column, n:

```
>SET /BUF=TI:n
>
```

(Note that n is an octal number by default. Type an explicit decimal point to enter a decimal number.) After system generation and before this SET has been done for a given terminal, the default column width is 72 (decimal).

Using the SET /BUF command without an argument causes an inquiry that returns the current buffer width for a terminal:

```
>SET /BUF=TI:
BUF=TI0:00072.
>
```

A task can determine the buffer width by issuing a Get LUN Information directive and examining word 5.

After the SET has been done, typing the n+1st character results in a CR/LF being output before the n+1st character is echoed (at the leftmost character position of the next line). There is still only one input line, but it is displayed on two lines on the terminal.

Output also wraps around after column n. This is undesirable for some applications. To disable wrap-around, set the buffer to some number greater than the terminal's column width. Output -- and input too -- beyond the column width then overprints at the right margin. Wrap-around is also disabled when executing the IO.WAL function (see Section 3.10.11), because the driver does not keep track of the cursor's position.

It is possible to lose track of where you are in the input buffer if both the automatic carriage return and the CRT rubout features have been selected at system generation. If, while rubbing out text on a wrapped line, you rub out the first character on that line, the cursor does not back up to the previous line. To resynchronize the cursor with the contents of the incomplete input buffer, type CTRL/R (if this option has been selected).

It is also possible to cause wrap-around to malfunction. This can occur when more than 255(decimal) characters are output without an intervening carriage return. This condition is possible because the driver maintains a byte location with the current cursor position; thus, counts greater than 255(decimal) are truncated, and the cursor count is invalid until the next carriage return is received.

3.8.2 Variable-Length Buffering

If this user-transparent system generation option is selected, up to 255(decimal) characters may be read from a terminal. The terminal driver allocates an Executive buffer the same size as the read request.

If the variable-length option is not chosen, any number of characters may be read from a terminal, but a maximum of 80(decimal) are transferred to the task issuing the read request. An Executive buffer of 80(decimal) characters is always allocated.

Note that, whether variable-length buffering is selected or not, a maximum of 80(decimal) characters may be directed to MCR as unsolicited input.

3.8.3 Task Buffering of Received Characters

This user-transparent system generation causes characters read from the terminal to be sent directly to the reading task's buffer. With this option, no Executive buffer need be allocated, and the completed input line need not be transferred to the task's buffer. This option, however, does not necessarily reduce system overhead. In a mapped system, each character must be mapped to the task's buffer. If the task uses Executive buffering, the mapping is done once and then all the characters are transferred. For the half-duplex terminal driver, the Executive buffers only input except for the prompt output on an IO.RPR request.

Task buffering may be overridden by checkpointing. If a task is checkpointable, an Executive buffer is allocated in the normal way and the task is made eligible for checkpointing by any task, regardless of priority, while the read proceeds. (Checkpointing only occurs when there is another task that can be made active.) Because checkpointability is a dynamic quality controlled by the task, you retain control over the resource trade-off.

3.8.4 LA30-P Support

This option provides a 1-byte software buffer for terminal input from an LA30-P. Because LA30-Ps communicate with RSX-11M by a single-buffered hardware interface, the echoing of an input character may block the reception of the next input character. This is because a character is normally discarded by the terminal driver if it is received before the echo of the previous character completes. The user-transparent system generation option for LA30-P support buffers the second character in the software.

This option should not be chosen at system generation if there are no LA30-Ps in the system.

3.9 TERMINAL INTERFACES

This section summarizes the characteristics of the four types of standard communication-line interfaces supported by RSX-11M. All four interfaces support parity, but RSX-11M does not.

3.9.1 DH11 Asynchronous Serial Line Multiplexer

The DH11 multiplexer interfaces up to 16 asynchronous serial communications lines for terminal use. The DH11 supports programmable baud rates. Input and output baud rates may differ; the input rate may be set to 0 baud, thus effectively turning off the terminal. The DM11-BB option may be included to provide modem control for dial-up lines. These lines must be interfaced by means of a full duplex modem (for example, in the United States, a Bell 103A or equivalent modem).

The direct memory access (DMA) capability of the DH11 is not supported by the RSX-11M terminal driver.

3.9.2 DJ11 Asynchronous Serial Line Multiplexer

The DJ11 multiplexer interfaces as many as 16 asynchronous serial lines to the PDP-11 for local terminal communications. The DJ11 does not provide a dial-up capability, but supports jumper-selectable baud rates.

3.9.3 DL11 Asynchronous Serial Line Interface

The DL11 supports a single asynchronous serial line and handles communication between the PDP-11 and a terminal. A number of standard baud rates are available to DL11 users. Four versions of the DL11 interface are supported by RSX-11M for terminal use: DL11-A, DL11-B, DL11-C, and DL11-D. The DL11-E is supported by the full-duplex terminal driver described in Chapter 2, and by the message-oriented communication drivers described in Chapter 11.

3.9.4 DZ11 Asynchronous Serial Line Multiplexer

The DZ11 multiplexer interfaces up to eight asynchronous serial communication lines for use with terminals. It supports programmable baud rates; however, input and output speeds must be the same. The DZ11 can control a full duplex modem in auto-answer mode.

3.10 PROGRAMMING HINTS

This section contains information relevant to you if your task uses the terminal driver.

3.10.1 Terminal Line Truncation

If automatic carriage return has not been selected at system generation, and if the number of characters to be printed exceeds the line length of the physical device unit, then the terminal driver discards the excess characters until it receives one that instructs it to return to horizontal position 1. You can determine when this happens by examining word 5 of the information buffer returned by the Get LUN Information system directive, or by typing "SET /BUF=TI:".

3.10.2 Escape Code Conversion

If escape sequences are not recognized, an ESCape or Altmode character code of 33, 175, or 176 is converted internally to 33 before it is returned to your task on input.

3.10.3 RT02-C Control Function

Because the screen of an RT02C Badge Reader and Data Entry Terminal holds only one line of information, special care must be taken when sending a control character (for example, vertical tab) to the RT02-C. Use IO.WAL (Write All).

It is advisable to read without echoing when reading a badge with the RT02-C. Use IO.RAL or IO.RNE, and then write the received information.

3.10.4 Checkpointing During Terminal Input

If checkpointing during terminal input was selected as a system generation option, a checkpointable task is stopped (and therefore eligible to be checkpointed) when trying to read. Therefore, a stratagem such as issuing a read followed by a mark-time does not work. The intent might be to time out the read if input is not received in a reasonable length of time. But the mark-time is not issued until the read completes.

You can circumvent this behavior by disabling checkpointing for the read. This is not a desirable solution because it forces a task to remain in memory during the entire read. This defeats the purpose of selecting the checkpoint-during-terminal-input option.

3.10.5 Time Required for IO.KIL

An IO.KIL request may take up to 1 second to succeed, because an internal mark-time mechanism generates a software interrupt to get into a clean state. The I/O may reach a state in which the kill can complete within this time (for instance, if a hardware interrupt is received). If not, the request is killed after 1 second.

3.10.6 Use of IO.WVB

We recommend that you routinely use IO.WVB, instead of IO.WLB, when writing to a terminal. If the write actually goes to a terminal, the Executive converts your IO.WVB into IO.WLB. However, if the LUN has been redirected to some inappropriate device -- a disk, for example -- using an IO.WVB is rejected because a file is not open on the LUN. This prevents privileged tasks from overwriting block zero of the disk (the boot block).

Note that any subfunction bits specified in the IO.WVB request (for example, TF.CCO, TF.WAL, or TF.WBT) are stripped off when the QIO is converted to an IO.WLB.

3.10.7 Remote DH11 and DZ11 Lines

All remote DH11 lines in a system are answered at the same baud rate. All remote DZ11 lines are also answered at the same rate, which may differ from the DH11 rate. These rates are specified at system generation.

Before a remote DH11 or DZ11 line is answered, the driver clears certain of the terminal characteristics (see Table 3-5) that may have been set by an MCR SET command or by an SF.SMC QIO. The characteristics cleared are: TC.SCP, TC.ESQ, TC.HLD, TC.SMR, and TC.TTP. (Clearing TC.TTP means that a terminal type of "unknown" is returned to an SF.GMC request.) Also, buffer size is set to 73.

A DZ11 remote line must be declared to be remote before the terminal driver can correctly handle the modem. You can do this with the MCR command SET /REMOTE=TI:.

NOTE

Because of the few modem signals that the DZ11 handles and the lack of interrupt support provided for those signals, the DZ11 may not adequately handle telephone exchange requirements in all countries.

3.10.8 High-Order Bit on Output

Setting the high-order bit of an output byte causes it to be transmitted but not interpreted by the driver.

3.10.9 Side Effects of Setting Characteristics

Some of the characteristics that a task may set, or that you may set from a terminal, have side effects that should be noted.

- **TC.HLD** -- Unexpected behavior can result from a terminal in hold-screen mode if its reception rate is much greater than its transmission rate. (The DH11 supports split baud rates.) In hold-screen mode the terminal sends a CTRL/S during reception of an output stream, when the screen is nearly full. Output is resumed -- another screen-full -- when you type SHIFT/SCROLL (the terminal generates CTRL/Q). Thus, no output is lost as a result of scrolling off the screen before you can read it. However, if the terminal's transmission rate is far below its reception rate, some unread output may scroll out of sight before the CTRL/S can be transmitted.

  A related point to note is that some terminals and interfaces are hardware buffered. This fact can cause obscure timing problems for tasks that try to implement hold-screen mode.

- **TC.SMR** -- If this characteristic is asserted (that is, if lowercase to uppercase conversion is disabled by, for example, SET /LOWER=TI:), the two characters 175(octal) and 1761(octal) are interpreted as [(right-brace) and ~(tilde), respectively. If TC.SMR is not asserted, these two characters act as Altmodes. That is, they act as line terminators that do not advance the cursor to a new line. Altmodes are not echoed.

3.10.10 Unsolicited-Input-Character ASTs for Tasks Attaching Several Terminals

For a task that attaches several terminals (for example, a reentrant language processor), the handling of unsolicited input requires special care. When the terminal driver passes an unsolicited input character to a task, it does not pass any information about which of several terminals generated the character. The task must ascertain this for itself.

One solution is for the task to name uniquely the AST entry points for each attached terminal. Each separate AST then identifies its terminal before branching to a common routine that processes the unsolicited character. For example:

```
ATT1:  QIO$C IO.ATA,...,<UIC1>
       BR CONT
ATT2:  QIO$C IO.ATA,...,<UIC2>
       BR CONT
        .
        .
        .
UIC1:  MOV #1,-(SP)
       BR UIC
UIC2:  MOV #2,-(SP)
       BR UIC
        .
        .
        .
UIC:   MOV (SP)+,INDEX
        .
        .
        .
```

3.10.11 Direct Cursor Control

The terminal driver generally examines the output stream to keep track of the cursor's horizontal position (so that output can be wrapped around or discarded). Therefore, tasks that want to use direct cursor control should use IO.WALs. This prevents the terminal driver from inserting CR/LFs (that the task considers spurious) into the output stream. FORTRAN WRITE statements become IO.WVBs, which are interpreted by the driver. To prevent this, a FORTRAN task can use the CALL QIO routine or can issue carriage returns at frequent intervals (to make the driver think the cursor is always well to the left of the rightmost column, and therefore no CR/LFs need be emitted to keep the cursor on the screen).

3.10.12 DL11 Receiver Interrupt Enable

For hardware reasons, a DL11 is susceptible to losing receiver interrupt enable in its Receiver Status Register. The disabling of the receiver interrupt bit causes the terminal to print output requests but not to respond to input (for example, the terminal does not echo input characters). The terminal driver has no mechanism for recognizing the disabling. Therefore, it cannot recover. The bit must be reset with an MCR OPEN command, the console switch register, or a periodically rescheduled task.

3.10.13 Loadable Driver Restrictions

Checkpointing during terminal input, variable-length terminal buffer support, and escape sequence support require the presence of conditionally assembled Executive support. If a loadable terminal driver supports one of these features and the Executive does not (or vice versa), the best that can happen is an undefined global when the terminal driver is built. At worst, the system is corrupted.

CHAPTER 4

VIRTUAL TERMINAL DRIVER

4.1 INTRODUCTION

The virtual terminal driver supports offspring task use of virtual terminals in RSX-11M-PLUS systems. Virtual terminals are not physical hardware devices; they are actually implemented in software through the use of data structures created by the RSX-11M-PLUS Executive. Virtual terminals are created by the Executive when requested by parent tasks with the Create Virtual Terminal directive. Virtual terminals are useful in batch processing and other processing environments in providing noninteractive terminal I/O support for offspring tasks, eliminating the need for operator intervention.

Offspring task(s) "spawned" by or "connected" to the parent task that created the virtual terminal can perform terminal I/O operations with the virtual terminal in the same manner as with physical terminals. Virtual terminals differ from physical terminals in that they receive input from or output to a program (the parent task), rather than from a keyboard or to a display (or printer), respectively.

4.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for virtual terminals. A setting of 1 indicates that the described characteristic is true for virtual terminals.

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	1	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Reserved
7	0	User-mode diagnostics supported
8	0	Massbus device
9	0	Unit software write-locked
10	0	Input spooled device

Bit	Setting	Meaning
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 are undefined. Word 5 specifies the maximum byte count (that is, maximum buffer size) to which offspring requests will be truncated; this value is specified by the parent task in the Create Virtual Terminal system directive, as described in the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual.

4.3 QIO$ MACRO

Table 4-1 lists the standard and device-specific functions of the QIO macro that are valid for virtual terminals.

Table 4-1
Standard and Device-Specific QIO Functions for Virtual Terminals

Format	Function
STANDARD FUNCTIONS:	
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O request
QIO$C IO.RLB,...,<stadd,size>	Read logical block
QIO$C IO.RVB,...,<stadd,size>	Read virtual block (effects IO.RLB)
QIO$C IO.WLB,...,<stadd,size,stat>	Write logical block
QIO$C IO.WVB,...,<stadd,size,stat>	Write virtual block (effects IO.WLB)

(continued on next page)

Table 4-1 (Cont.)
Standard and Device-Specific QIO Functions for Virtual Terminals

Format	Function
DEVICE-SPECIFIC FUNCTIONS:	
QIO$C IO.STC,...,<cb,sw2,sw1>	Set terminal characteristics (enable/disable intermediate I/O buffering, or return I/O completion status to offspring task)
QIO$C SF.GMC,...,<stadd,size>	Get multiple characteristics
QIO$C IO.GTS,...,<stadd,size>	Get terminal support
QIO$C IO.RPR,...,<stadd,size,[tmo], pradd,prsize,vfc>	Read logical block after prompt
QIO$C SF.SMC,...,<stadd,size>	Set multiple characteristics

size

The size of the data buffer in bytes (must be greater than 0). The buffer must be located within the addressing space of the parent or offspring task issuing the I/O request.

stadd

The starting address of the data buffer. The address must be word aligned for SF.GMC, IO.GTS, and SF.SMC; otherwise, it may be aligned on a byte boundary.

stat

The I/O completion status code, specified by the parent task, that is issued by the virtual terminal driver in response to an offspring task's read request upon successful completion.

cb

Characteristic bits to become set, selecting the following virtual terminal functions:

cb Value	Bits Set	Function
0	none	Enable intermediate buffering in the Executive pool
1	0	Return the specified virtual terminal I/O completion status to the requesting offspring task
2	1	Disable intermediate buffering
3	0 and 1	Return status for offspring write request

sw1

The I/O completion code for I/O completion status.

NOTE

The sw2 and sw1 parameters are valid in the IO.STC function only when cb=1 or cb=3.

tmo

An optional time-out count (see below).

vfc

A character for vertical format control. See Table 3-11.

pradd

The starting address of the prompt buffer.

prsize

The size of the prompt buffer in bytes. The buffer must be located within the address space of the offspring task issuing the I/O request.

4.3.1 Standard QIO Functions

4.3.1.1 **IO.ATT** - This I/O function can be issued by offspring task tasks to attach the virtual terminal. (It is illegal for parent tasks to issue IO.ATT). Attaching a virtual terminal prevents other offspring tasks from executing I/O operations with the virtual terminal. However, parent task I/O requests are always serviced when issued.

4.3.1.2 **IO.DET** - This I/O function can be issued by offspring tasks to detach the virtual terminal, making it available for use by other offspring tasks connected to the same parent task. (It is illegal for parent tasks to issue IO.DET.)

4.3.1.3 **IO.KIL** - Parent and offspring tasks can issue IO.KIL to cancel I/O requests. An offspring task issuing IO.KIL can result in IE.ABO being returned to the parent task.

4.3.1.4 **IO.RLB, IO.RVB, IO.WLB, IO.WVB** - These read and write functions execute requested I/O operations described in Chapter 2, except as follows:

1. The virtual terminal driver returns the tmo parameter of an offspring task's IO.RLB or IO.RVB request, or the vfc parameter of an offspring task's IO.WLB or IO.WVB request as a stack parameter on entry to the appropriate AST for the parent task.

2. The virtual terminal driver returns I/O completion status to the offspring task in response to successful completion of the offspring task's IO.RLB or IO.RVB request; however, the actual I/O completion status values returned are specified for data transfers in the third parameter word of the parent task's IO.WLB or IO.WVB response, or in the second and third parameters of the parent task's IO.STC function response when no data transfer is desired.

4.3.2 Device-Specific QIO Function (IO.STC)

The IO.STC function can be issued by parent tasks to enable/disable offspring task I/O buffering in secondary pool, or to force an appropriate I/O completion status for an offspring task read I/O request when no data transfer is desired. Both of these applications for the IO.STC function are described as follows.

Parent tasks can use IO.STC to enable (or disable) intermediate buffering in secondary pool. Intermediate buffering, when enabled, is performed on offspring task virtual terminal read and write requests when the offspring task is checkpointable.

Thus, offspring tasks can be stopped for virtual terminal I/O and checkpointed in a manner similar to that when you use physical terminals. Whenever the virtual terminal driver determines that it should not use intermediate buffering, offspring tasks that issue terminal requests become locked in memory until I/O completion; transfers occur directly between parent task and offspring task buffers without intermediate buffering in secondary pool.

In addition to the conditions that permit intermediate buffering (when specified), one condition can disable intermediate buffering of the parent task. If the buffer size specified in the Create Virtual Terminal directive exceeds the maximum size specified at system generation time (512(10) maximum), intermediate buffering is disabled.

The second application for IO.STC is to allow the virtual terminal driver to return an appropriate I/O completion status in response to an offspring task read request. I/O status returned in this manner allows successful completion of the offspring task's request when the parent task determines that no data transfer is desired; this condition can occur, for example, when no data is available for input to the offspring task by the virtual terminal driver. When you use the IO.STC function in this manner, you must include the three parameters, <cb,sw2,sw1>, as follows:

cb

A value of 1 is specified to indicate that the I/O completion status return to the offspring task is desired.

NOTE

If the virtual terminal is operating in full duplex mode, a cb value of 1 returns status for an offspring read request, and a cb value of 3 returns status for an offspring write request.

sw2

This parameter is the second word returned in the I/O completion status indicating the number of bytes read upon successful completion of an offspring task's read request. However, because no data transfer actually occurs, the value specified is 0; the byte count of 0 specified in this function is legal (and desired), whereas a byte count of 0 in write operations is illegal (and results in an error being returned to the parent task).

sw2

This parameter specifies the status code to be returned to the offspring task by the virtual terminal driver in the first word of the I/O completion status. This value is returned in the high byte and a value of +1 is returned in the low byte of the status word. Typical values and the status that each represent are listed as follows:

Code	Value	Completion Status Indicated
IS.SUC	+ 1	Successful completion
IS.CR	15	Read terminated by carriage return
IS.ESC	33	Read terminated by an Altmode
IS.ESQ	233	Read terminated by an escape sequence

4.3.3 SF.GMC

The Get Multiple Characteristics function returns information on terminal characteristics. This function can be issued by both the parent and the offspring tasks. The virtual terminal driver returns the characteristics that were set by the previous corresponding SF.SMC request. However, only the full duplex mode (TC.FDX) characteristic affects the operation of the virtual terminal driver. The SF.GMC function is provided only to maintain transparency to the offspring task.

Valid virtual terminal characteristics are listed in Table 4-2.

4.3.4 IO.GTS

The Get Terminal Support function returns a 4-word buffer of information specifying which features are a part of the virtual terminal driver. The virtual terminal driver provides the IO.GTS function only to maintain transparency to the offspring task. Table 2-7 lists the options returned by the full duplex terminal driver. Of those listed, the virtual terminal driver returns the following:

Word 1 -- F1.BUF, F1.RPR, F1.UTB, and F1.VBF

Word 2 -- F2.SCH and F2.GCH

4.3.5 IO.RPR

The Read After Prompt (IO.RPR) function can be issued only by the offspring task. When the offspring task issues this function, the function appears to the parent task as a separate write request followed by a read request. This function is described in Chapter 3.

4.3.6 SF.SMC

The SF.SMC function allows a task to set and reset the characteristics of a terminal. Both the parent and the offspring tasks may issue this function. The parent task may set virtual terminals to full duplex operation by using the SF.SMC function with the characteristics bit TC.FDX. When in full duplex mode, the virtual terminal driver attempts to process the offspring task's read and write requests simultaneously. To ensure that these operations are overlapped, the parent task should minimize the amount of time it spends in AST state.

The virtual terminal driver defaults to half duplex mode.

Table 4-2 lists the characteristics that either the parent or the offspring task may set.

Table 4-2
Virtual Terminal Characteristics

Bit Name	Octal Value	Meaning (If Asserted)	Default Value
TC.FDX	64	Full duplex mode	0
TC.SCP	12	Terminal is a scope	0
TC.SMR	25	Uppercase conversion disabled	0
TC.TTP	10	Terminal type	0

4.4 STATUS RETURNS

The error and status conditions listed in Tables 4-3 and 4-4 are returned by the virtual terminal driver described in this chapter. The SE.NIH error is returned by the SF.GMC and SF.SMC functions. For this error, the low byte of the first word in the I/O status block contains IE.ABO. The second word in the I/O status block contains an offset (starting at 0) pointing to the erroneous byte in the stadd buffer.

Table 4-3
Virtual Terminal Status Returns for Offspring Task Requests

Code	Reason
--	Successful completion of an offspring task read request results in an I/O completion status specified in a parent task QIO parameter being returned. Typically, the status information returned simulates a subset of I/O returns normally produced by the terminal drivers described in Chapter 2.
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block indicates the number of bytes transferred on a write operation.
IE.IFC	**Invalid function code** The offspring task attempted a read or a write function and the parent task did not specify an AST address in its response to the requested I/O function, or the offspring task issued an IO.STC or other invalid function.
IE.ABO	**Request terminated** The offspring task issued IO.KIL or the parent task eliminated the virtual terminal unit.
IE.SPC	**Illegal address space** Part or all of the buffer specified for a read or write request was outside of the task's address space, or a byte count of 0 was specified.
IE.UPN	**Insufficient dynamic storage** The driver could not allocate an AST block to notify the parent task of an offspring task request, or the driver could not allocate an intermediate buffer in the Executive pool.
SE.NIH	A terminal characteristic other than those in Table 4-2 was specified, or an offspring task attempted to assert TC.FDX.

Table 4-4
Virtual Terminal Status Returns for Parent Task Requests

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block indicates the number of bytes transferred on a read or write operation.
IE.EOF	**End of file encountered** The IO.STC function was completed successfully.
IE.BAD	**Bad parameters** The parent task specified a buffer size that exceeded the system maximum specified at system generation time.
IE.DUN	**Device not attachable** An IO.ATT or IO.DET function was issued by the parent task.
IE.IFC	**Invalid function code** A read, write, or IO.STC function was issued without a pending offspring task request. This status can occur if the offspring task cancels a pending read or write request. This function code is also returned when IO.STC is issued to enable intermediate buffering on a virtual terminal unit whose buffer size, specified in the Create Virtual Terminal directive, exceeds the system maximum specified at system generation time.
SE.NIH	A terminal characteristic other than those in Table 4-2 was specified in an SF.GMC or SF.SMC request.

CHAPTER 5

DISK DRIVERS

5.1 INTRODUCTION

The RSX-11M disk drivers support the disks summarized in Table 5-1. Subsequent sections describe these devices in greater detail.

All of the disks described in this chapter are accessed in essentially the same manner. Up to eight disks of each type (except RX01, RX02, RX50, RD51, RD52, RC25, RL01, RL02, RA60, RA80, or RA81) may be connected to their respective controllers. Disks and other file-structured media are divided logically into series of 256-word blocks.

5.1.1 RF11/RS11 Fixed-Head Disk

The RF11 controller/RS11 fixed-head disk provides random access bulk storage. It features fast track-switching time and a redundant set of timing tracks.

5.1.2 RS03 Fixed-Head Disk

The RS03 (RH11-RH70 controller/RS03 fixed-head disk) is a fixed-head disk that offers speed and efficiency. With 64 tracks per platter and recording on one surface, the RS03 has a capacity of 262,144 words.

5.1.3 RS04 Fixed-Head Disk

The RS04 (RH11-RH70 controller/RS04 fixed-head disk) is similar to the RS03 disk and interfaces to the same controller, but provides twice the number of words per track by recording on both surfaces of the platter, and thus has twice the capacity.

5.1.4 RP11/RP02 or RP03 Pack Disks

The RP11 controller/RP02 or RP03 pack disk consists of 20 data surfaces and a moving read/write head. The RP03 has twice as many cylinders, and thus double the capacity of the RP02. Only an even number of words can be transferred in a read/write operation.

Table 5-1
Standard Disk Devices

Drive	RPM	Secs	Trks	Cyls	Bytes/ Drive	Decimal Blocks
RS11	1800	--	1	128	524,288	1024
RS03	3600	64 [1]	1	64	524,288	1024
RS04	3600	64 [1]	1	64	1,048,576	2048
RPR02	2400	10	20	200	20,480,000	40,000
RP03	2400	10	20	400	40,960,000	80,000
RM02	2400	32	5	823	67,420,160	131,680
RM03	3600	32	5	823	67,420,160	131,680
RM05	3600	32	19	823	256,196,608	500,384
RP04,RP05	3600	22	19	411	87,960,576	171,798
RP06	3600	22	19	815	174,423,040	340,670
RP07	3600	50	32	630 [2]	516,096,000	1,008,000
RM80	3600	31	14	559 [2]	124,214,272	242,606
RK05	1500	12	2	200	2,457,600	4800
RL01	2400	40 [3]	2	256	5,242,880	10,240
RL02	2400	40 [3]	2	512	10,485,760	20,480
RK06	2400	22	3	411	13,888,512	27,126
RK07	2400	22	3	815	27,810,800	53,790
RX01	360	26 [4]	1	77	256,256	494
RX02	360	26 [4]	1	77	512,512	988
RA80	3600	31	14	546	121,325,568	236,964
RA81	3600	51	14	1248	456,228,864	891,072
RA60	3600	42	4	2382	204,890,112	400,176
RC25	2850	31	2	796	26,061,824	50,902
RD51	3600	16	4	306	10,027,008	19,584
RD52	Manufacturer dependent				30,9657,60	60,480
RX50	300	10	1	80	409,600	800

1. The RS03 has 64 words per sector; the RS04 has 128 words/sector.

2. The RP07 and the RM80 each have two additional CE cylinders.

3. The RL01 and RL02 each have 128 words per sector.

4. The RX01 has 64 words per sector; the RX02 has 128 words per sector.

5.1.5 RM02/RM03/RM05/RM80 Pack Disk

The RM02/RM03, RM05, and RM80 are MASSBUS disk drives and adapters that use the existing MASSBUS controller. With a single head per surface, they provide a 1.2 megabyte-per-second data transfer rate. PDP-11/70 systems use the RM03, RM05, and RM80 with the RH70 controller on PDP-11/70 systems. All other systems use the RM02 with the RH11 controller.

5.1.6 RP04, RP05, RP06 Pack Disks

The RP04 or RP05 (RH11-RH70 controller/RP04 or RP05 pack disk) pack disks consist of 19 data surfaces and a moving read/write head. Both offer large storage capacity with rapid access time. The RP06 pack disk has approximately twice the capacity of the RP04 or RP05. The RP07 fixed-media disk has approximately 3 times the capacity of the RP06.

5.1.7 RK11/RK05 or RK05F Cartridge Disks

The RK11 controller/RK05 DECpack cartridge disk is an economical storage system for medium-volume, random access storage. The removable disk cartridge offers the flexibility of large off-line capacity with rapid transfers of files between on- and off-line units without necessitating copying operations. The RK05F has twice the storage capacity of the RK05 and has a fixed (nonremovable) disk cartridge.

5.1.8 RL11/RL01 or RL02 Cartridge Disk

The RL01 is a low-cost, single-head per surface disk with a burst data transfer rate of 512 kilobytes per second. The storage capacity of the RL02 is twice that of the RL01.

5.1.9 RK611/RK06 or RK07 Cartridge Disk

The RK611 controller/RK06 cartridge disk is a removable, random access, bulk-storage system with three data surfaces. The storage capacity is 6,944,256 words per pack. The system, expandable to eight drives, is suitable for medium to large systems.

The RK611 controller/RK07 cartridge disk is generally similar to the RK611/RK06, except storage capacity is increased to approximately 13,905,400 words per pack. Both RK06 and RK07 disks can use the same RK611 controller; mixing RK06 and RK07 disks on the same controller is permitted.

5.1.10 RX11/RX01 Flexible Disk

The RX11 controller/RX01 flexible disk is an economical storage system for low-volume, random access storage. Data is stored in twenty-six 64-word sectors per track; there are 77 tracks per disk. Data may be accessed by physical sector or logical block. If logical or virtual block I/O is selected, the driver reads four physical sectors. These

sectors are interleaved to optimize data transfer. The next logical sector that falls on a new track is skewed by six sectors to allow for track-to-track switch time. Physical block I/O provides no interleaving or skewing and provides access to all 2002 sectors on the disk. Logical or virtual I/O starts on track 1 and provides access to 494 logical blocks.

5.1.11 RX211/RX02 Flexible Disk

The RX211 controller/RX02 flexible disk is an economical storage system for low-volume, random access storage. It is capable of operating in either an industry-standard, single-density mode (as stated for the RX11/RX01 flexible disk), or a double-density mode (not industry standard). In the single-density mode, each drive can store data exactly as stated in Section 5.1.10. In the double-density mode, data is stored in twenty-six 128-word sectors per track; there are 77 tracks per disk. The RX211/RX02 operating in the single-density mode can read disks written by an RX11/RX01 flexible disk system. In addition, disks written by the RX211/RX02 operating in the single-density mode can be read by the RX11/RX01 flexible disk system.

5.1.12 ML-11 Disk Emulator

The ML-11 is a fast, random access, block-mode MOS memory system. The RSX-11M and RSX-11M-PLUS operating systems treat the ML-11 as a disk. However, because it is not a disk, the statistics in Table 5-1 do not apply. Unlike a disk, the number of bytes per drive varies. One ML-11 provides from 512 blocks to 8192 blocks of storage.

5.1.13 KDA50,UDA50/RA60/RA80/RA81 Disks

The KDA50 or UDA50 controller is an intelligent disk controller that contains a high-speed microprogrammed processor capable of performing all disk functions, including data handling, error detection and correction, and optimization of disk drive activity and data transfers. The controller optimizes disk activity by reordering QIO$s. Therefore, QIO$ macros may not complete in the order in which they were issued. The types of drives that can be connected to the KDA50 or UDA50 controllers are the RA60 disk drive, which has a removable pack, and the RA80 and RA81, both of which are fixed media drives. (For data capacities and rates, see Table 5-1.) Up to four of these drives can be connected to a KDA/UDA, in any desired combination.

The KDA/UDA controller can perform an extensive self-test on power-up or initialization.

5.1.14 RC25 Disk Subsystem

The RC25 disk subsystem consists of a fixed-media drive and a removable-media drive, both of which revolve on the same spindle and share the same head mechanics. Each drive is a logical unit, so each RC25 disk subsystem consists of two logical units.

The RC25 Subsystem combines, in one package, a controller and a single disk drive that has a removable disk and a fixed disk. These disks reside in the drive as two separate logical units on a single spindle. Their size is the same. Both are single eight-inch disks with two surfaces, and both disks have the same data capacity. But mechanically they are different: One is a removable front-loading cartridge disk, while the other cannot be removed from the drive. The drive contains loadable Winchester heads.

RC25 subsystems are available in two types: a master drive that contains its own controller, and a slave drive, which must be connected to an RC25 master drive. Each RC25 master drive can support one RC25 slave drive. The added-on disk drive is a slave to the disk subsystem that has the controller. A master-slave configuration would contain four logical units.

5.1.15 RD51 Fixed 5.25 Disk/RX50 Flexible 5.25 Disk

This subsystem consists of a hard disk (RD51) and flexible disk (RX50) combination, and a RQDX1/RQDX2 controller. In combination, they are a mass storage medium for small systems. The basic configuration for this subsystem is an RD51 fixed disk drive and an RX50 flexible dual disk drive. In this configuration, the RD51 is the system device and the RX50 is a data or a backup device, or both. The RX50 dual disk is addressed as two separate units resulting in a basic configuration of three disk units. Also, you can add another RD51 to increase storage capacity. Some of the characteristics of the RD/RX drives are given in Table 5-1 and in the following paragraphs.

The RD51 disk drive is a 5.25 inch fixed disk with Winchester type heads. It has two disks with four data surfaces. The RD51 is soft sectored and field formattable. The headers for each sector contain the sector's cylinder number, head number, and sector number. The sector number is the logical sector number (0-15) that reflects the sector interleave of the disk.

The RX50 dual diskette drive is a compact mass storage drive with two access slots. Each slot can hold a single-sided 5.25 flexible disk. These diskettes are firm sectored and are not field formattable. Every track has sectors numbered from 1 to 10. The two diskettes share the same head transport mechanism.

RSX-11M-PLUS also supports the RUX50 Unibus interface for the RX50 dual diskette drive and the RX180 IBM-compatible diskette drive.

5.1.16 RD52 Fixed 5.25 Disk

The RD52 disk drive is a 5.25 inch fixed disk with Winchester type heads. The RD52 is soft sectored and field formattable. The maximum capacity of the RD52 is 30.97 megabytes.

5.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for disks. A bit setting of 1 indicates that the described characteristic is true for disks.

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	1	File structured device
4	0	Single-directory device
5	0	Sequential device
6	1	Mass storage device
7	X	User-mode diagnostics supported (device dependent)
8	X	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo-device
13	0	Device mountable as a communications channel
14	1	Device mountable as a FILES-11 volume
15	1	Device mountable

Words 3 and 4 of the buffer contain the maximum logical block number. Note that the high byte of U.CW2 is undefined. Your task should clear the high byte in the buffer before using the block number. For DU: type disks, these two words are undefined until the device has been mounted at least once. Word 5 indicates the default buffer size, which is 512 bytes for all disks.

5.3 QIO$ MACRO

This section summarizes the standard, and device-specific QIO functions for disk drivers.

5.3.1 Standard QIO$ Functions

Table 5-2 lists the standard functions of the QIO$ macro that are valid for disks.

Table 5-2
Standard QIO$ Functions for Disks

Format	Function
QIO$C IO.ATT,...	Attach device [1]
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Kill I/O [2]
QIO$C IO.RLB,...,<stadd,size,,blkh,blkl>	READ logical block
QIO$C IO.RVB,...,<stadd,size,,blkh,blkl>	READ virtual block
QIO$C IO.WLB,...,<stadd,size,,blkh,blkl>	WRITE logical block
QIO$C IO.WLC,...,<stadd,size,,blkh,blkl>	WRITE logical block followed by write check [3]
QIO$C IO.WVB,...,<stadd,size,,blkh,blkl>	WRITE virtual block

1. In RSX-11M systems, only unmounted volumes may be attached; in RSX-11M-PLUS systems, only volumes mounted foreign may be attached. Any other attempt to attach a mounted volume results in an IE.PRI status being returned in the I/O status doubleword.

2. In-progress disk operations are allowed to complete when IO.KIL is received, because they take such a short time. I/O requests that are queued when IO.KIL is received are killed immediately. An IE.ABO status is returned in the I/O status doubleword.

3. Not supported on RX01 or RX02 flexible disks.

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes (must be even, greater than 0, and, for the RP02 and RP03, also a multiple of four bytes).

blkh/blkl

Block high and block low, combining to form a double-precision number that indicates the actual logical/virtual block address on the disk where the transfer starts; blkh represents the high 8 bits of the address, and blkl the low 16 bits.

IO.RVB and IO.WVB are associated with file operations (see the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual). For these functions to be executed, a file must be open on the specified LUN if the volume associated with the LUN is mounted. Otherwise, the virtual I/O request is converted to a logical I/O request using the specified block numbers.

NOTE

When writing a new file using QIOs, the task must explicitly issue .EXTND File Control System library routine calls as necessary to reserve enough blocks for the file, or the file must be initially created with enough blocks allocated for the file. In addition, the task must put an appropriate value in the FDB for the end-of-file block number (F.EFBK) before closing the file. (Refer to the .EXTND routine description in the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual.)

Each disk driver supports the subfunction bit IQ.X: inhibit retry attempts for error recovery. You use this subfunction bit by using it in a Logical OR with the desired QIO; for example:

```
QIO$C IO.WLB!IQ.X,...,<stadd,size,,blkh,blkl>
```

The IQ.X subfunction permits you to specify retry algorithms for applications in which data reliability must be high.

The overlapped seek drivers for RSX-11M-PLUS support subfunction bit IQ.Q, which queues the request immediately without doing a seek (that is, uses implied seeks).

5.3.2 Device-Specific QIO$ Functions

The device-specific functions of the QIO$ macro are valid for the RX01/RX02/RL01/RL02 only; they are shown in Table 5-3.

Table 5-3
Device-Specific Functions for the
RX01,RX02, RL01, and RL02 Disk Drivers

Format	Function
QIO$C IO.RPB,...,<stadd,size,,,pbn>	Read physical block
QIO$C IO.SEC,...	Sense diskette characteristics (RX02 only)
QIO$C IO.SMD,...,<density,,>	SET media density (RX02 only)
QIO$C IO.WDD,...,<stadd,size,,,pbn>	Write physical block (with deleted data mark) (RX01 and RX02 only)
QIO$C IO.WPB,...,<stadd,size,,,pbn>	Write physical block

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes must be even and greater than 0).

pbn

The physical block number where the transfer starts (no validation will occur).

density

The media density as follows:

0 = single (RX01-compatible) density
2 = double density

5.3.3 Device-Specific QIO$ Function for the DUDRV

The DU: device driver supports the device-specific QIO$ function shown in Table 5-4.

Table 5-4
Device-Specific QIO$ Function for the DU: Device Driver

Format	Function
QIO$C IO.RLC,...,<stadd,size,blkh,blkl>	Read Logical with Read Check modifier

The IO.RLC function is a read logical block followed by a read check. The disk is read twice.

5.4 STATUS RETURNS

The error and status conditions listed in Table 5-5 are returned by the disk drivers described in this chapter.

Table 5-5
Disk Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO$ directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	**I/O request pending** The operation specified in the QIO$ directive has not yet been executed. The I/O status block is filled with 0s.

(continued on next page)

Table 5-5 (Cont.)
Disk Status Returns

Code	Reason
IS.RDD	**Deleted data mark read** A deleted record was encountered while executing an IO.RPB function. The second word of the I/O status block can be examined to determine the number of bytes processed (RX01 and RX02 only).
IE.ABO	**Request aborted** An I/O request was queued (not yet acted upon by the driver) when an IO.KIL was issued.
IE.ALN	**File already open** The task attempted to open a file on the physical device unit associated with specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BLK	**Illegal block number** An invalid logical block number was specified. This code would be returned, for example, if block 4800 were specified for an RK05 disk, on which legal block numbers extend from 0 through 4799. IE.BLK would also be returned if an attempt was made to write on the last track of an RK06 disk. (See Section 5.5.)
IE.BBE	**Bad block error** The disk sector (block) being read was marked as a bad block in the header word. Data cannot be written on or read from a bad block.
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a buffer, but only word alignment is legal for disk. Alternatively, the length of a buffer is not an appropriate number of bytes. For example, all RP03 and RP02 disk transfers must be multiples of four bytes.
IE.DNR	**Device not ready** The physical device unit specified in the QIO$ directive was not ready to perform the desired I/O operation.
IE.FHE	**Fatal hardware error** The controller is physically unable to reach the location where input/output operation is to be performed. The operation cannot be completed.

(continued on next page)

Table 5-5 (Cont.)
Disk Status Returns

Code	Reason
IE.IFC	**Illegal function** A function code was specified in an I/O request that is invalid for disks.
IE.NLN	**File not open** The task attempted to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	**Insufficient buffer space** Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for this operation.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line.
IE.OVR	**Illegal read overlay request** A read overlay was requested, and the physical device unit specified in the QIO$ directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.
IE.PRI	**Privilege violation** The task that issued the request was not privileged to execute that request. For disk, this code is returned if a nonprivileged task attempts to read or write a mounted volume directly (that is, using IO.RLB or IO.WLB). Also, this code is returned if any task attempts to attach a mounted volume.
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task, or a byte count of 0 was specified.
IE.VER	**Unrecoverable error** After the system's standard number of retries has been attempted upon encountering an error, the operation still could not be completed. For disk, unrecoverable errors are usually parity errors.

(continued on next page)

Table 5-5 (Cont.)
Disk Status Returns

Code	Reason
IE.WCK	**Write check error** An error was detected during the write check portion of an operation.
IE.WLK	**Write-locked device** The task attempted to write on a disk that was write-locked.

When a disk I/O error condition is detected, an error is usually not returned immediately. Instead, RSX-11M attempts to recover from most errors by retrying the function as many as eight times. Unrecoverable errors are generally parity, timing, or other errors caused by a hardware malfunction.

5.5 PROGRAMMING HINTS

5.5.1 UDA50 QIO$C IO.ATT Before GLUN$

The UDA50 dynamically updates the system data base to reflect the characteristics of the UDA50. Therefore, your task should issue a QIO$ IO.ATT function before requesting the device's characteristics with the GET LUN directive.

5.5.2 RX02 QIO$C IO.SEC Before GLUN$

The RX02 driver (DYDRV) dynamically updates the system data base to reflect the characteristics of the media in the RX02 drive. Therefore, your task should issue a QIO$C IO.SEC (sense characteristics) function before requesting the device's media characteristics with the GLUN$ directive.

5.5.3 Bad Sector Track on Disks

For the RK611 controller/RK06 or RK07 disk, the RL11 controller/RL01 or RL02 disk, RM02 disk, RM03 disk, RM05 disk, RM80 disk, and RP07 disk, the driver write-protects the last track of the cartridge. This track contains the factory-recorded bad-sector file.

5.5.4 Stalling Input and Output (I/O)

Because two RC25 disk units revolve on the same spindle and share the same head mechanics, you must spin down both units of a subsystem in order to spin down one unit. You cannot access either unit until the subsystem is spun up again. Because you must spin down the drive any time you want to insert or remove a disk from the removable-media unit, the device driver (DUDRV) allows you to spin down the subsystem and still retain context on the fixed-media unit, provided it is

mounted as a Files-11 or foreign volume. It does this by postponing input and output to the fixed-media unit until the subsystem is spun up again and the heads are reloaded. This is called stalled I/O.

When the driver receives an I/O request that it cannot process because the drive is spun down, it issues the following message to the console:

```
<ddnn:> - I/O stalled
```

When the drive is spun up again and I/O to the device is resumed, the driver issues the following message to the console:

```
<ddnn:> - I/O resumed
```

Note that because the only reason you would want to spin down the disk on a running system would be to replace the removable disk, and you would never specifically need to spin down the fixed-media unit, I/O is never stalled to the removable-media unit. The removable-media unit behaves like any other disk on an Micro/RSX system: if you spin it down, context is lost.

Stalling I/O to an RC25 subsystem affects the system's performance. If you initiate an operation requiring I/O to a stalled unit, you will not receive a timely response to the request. Although the I/O request is queued to the device driver, the driver ignores the request until the drive is loaded and the unit is ready. The driver then resumes processing requests. Note, however, that an operation can continue as long as it does not require access to the unit whose I/O is stalled.

Sometimes an operation that does not involve stalled-I/O units is delayed as well. For example, assume that your system disk is in the fixed-media unit and that you spin down a subsystem in order to change the disk pack in the removable-media unit. If a user then initiates an operation requiring a task to be loaded from the fixed unit, the loader issues a queued I/O request to the fixed unit. However, the device driver does not respond to this request immediately, since the subsystem is spun down. Also, because the loader cannot service additional tasks until it loads the current task from the disk, load operations to other disks on the system remain in the loader's work queue until the current load operation completes.

NOTE

Like the loader, the Files-11 Ancillary Control Processor (Files-11 ACP or F11ACP) is another single-threaded task that may delay response time when I/O is stalled to the RC25. To avoid this delay, you should always install a unique ACP for the RC25 fixed-media units (see the MOU command in the RSX-11M/M-PLUS MCR Operations Manual or the MOUNT command in the RSX-11M/M-PLUS Command Language Manual).

System users may find it difficult to distinguish between system crashes and system delays due to stalled I/O. Therefore, it is recommended that, before you spin down an RC25 subsystem, you inform all system users of your intentions.

5.5.5 Dismounting the RC25

You dismount a unit on the RC25 in the same way as for other disk devices, by using the DISMOUNT command. However, there are restrictions on using the /UNLOAD qualifier to spin down the disk. Since context may be lost on the removable disk if the subsystem is spun down, all spin down requests are ignored for the fixed unit of the RC25. For the removable disk unit, you must be privileged in order to spin down the device while dismounting it. The privileged status of DISMOUNT/UNLOAD is a safety measure to control who is able to spin down the system disk.

If you are a privileged user, DISMOUNT/UNLOAD issues the following message when the command executes properly:

```
Warning -- All units of multiunit drive will spin down <ddnn:>
```

If you are a nonprivileged user, DISMOUNT/UNLOAD refuses your request to spin down a unit and issues the following message:

```
Warning -- Volume will not spin down <ddnn:>
```

CHAPTER 6

DECTAPE DRIVER

6.1 INTRODUCTION

The RSX-11M DECtape driver supports the TC11-G dual DECtape controller with up to three additional dual DECtape transports. The TC11-G is a dual-unit, bidirectional, magnetic-tape transport system for auxiliary data storage. DECtape is formatted to store data at fixed positions on the tape, rather than at unknown or variable positions as on conventional magnetic tape. The system uses redundant recording of the mark, timing, and data tracks to increase reliability. Each reel contains 578 logical blocks. As with disk, each of these blocks can be accessed separately, and each contains 256 words.

6.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for DECtapes. A bit setting of 1 indicates that the described characteristic is true for DECtapes.

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	1	File structured device
4	0	Single-directory device
5	0	Sequential device
6	1	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit addressing
9	0	Unit software write-locked

Bit	Setting	Meaning
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	1	Device mountable as a FILES-11 volume
15	1	Device mountable

Words 3 and 4 of the buffer contain the maximum LBN. Word 5 indicates the default buffer size, 512 bytes, for DECtape.

6.3 QIO$ MACRO

This section summarizes standard and device-specific QIO$ functions for the DECtape driver.

6.3.1 Standard QIO$ Functions

Table 6-1 lists the standard functions of the QIO$ macro that are valid for DECtape.

Table 6-1
Standard QIO$ Functions for DECtape

Format	Function
QIO$C IO.ATT,...	Attach device [1]
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Kill I/O [2]
QIO$C IO.RLB,...,<stadd,size,,,lbn>	READ logical block (forward)
QIO$C IO.RVB,...,<stadd,size,,,lbn>	READ virtual block (forward)
QIO$C IO.WLB,...,<stadd,size,,,lbn>	WRITE logical block (forward)
QIO$C IO.WVB,...,<stadd,size,,,lbn>	WRITE virtual block (forward)

1. Only unmounted volumes may be attached. An attempt to attach a mounted volume results in an IE.PRI status being returned in the I/O status doubleword.

2. In-progress DECtape operations are allowed to complete when IO.KIL is received, unless the unit is not ready, because they take such a short time. I/O requests that are queued when IO.KIL is received are killed. An IE.ABO status is returned in the I/O status doubleword.

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes (must be even and greater than 0).

lbn

The logical block number on the DECtape where the transfer starts (must be in the range 0-577).

IO.RVB and IO.WVB are associated with file operations (see the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual). For these functions to be executed, a file must be open on the specified LUN if the volume associated with the LUN is mounted. Otherwise, the virtual I/O request is converted to a logical I/O request using the specified block numbers.

6.3.2 Device-Specific QIO$ Functions

The device-specific functions of the QIO$ macro that are valid for DECtape are shown in Table 6-2.

Table 6-2
Device-Specific Functions for DECtape

Format	Function
QIO$C IO.RLV,...,<stadd,size,,,lbn>	READ logical block (reverse)
QIO$C IO.WLV,...,<stadd,size,,,lbn>	WRITE logical block (reverse)

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes (must be even and greater than 0).

lbn

The transfer starts (must be in the range 0-577).

6.4 STATUS RETURNS

The error and status conditions listed in Table 6-3 are returned by the DECtape driver described in this chapter.

Table 6-3
DECtape Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO$ macro was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	**I/O request pending** The operation specified in the QIO$ macro has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Request aborted** An I/O request was queued (not yet acted upon by the driver) when an IO.KIL was issued.
IE.ALN	**File already open** The task attempted to open a file on the physical device unit associated with the specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BLK	**Illegal block number** An illegal logical block number was specified for DECtape. The number exceeds 577 (1101 (8)).
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a buffer, but only word alignment is legal for DECtape. Alternately, the length of the buffer is not an even number of bytes.
IE.DNR	**Device not ready** The physical device unit specified in the QIO$ macro was not ready to perform the desired I/O operation.
IE.IFC	**Illegal function** A function code was specified in an I/O request that is illegal for DECtape.

(continued on next page)

Table 6-3 (Cont.)
DECtape Status Returns

Code	Reason
IE.NLN	**File not open** The task attempted to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	**Insufficient buffer space** Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for this operation.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO$ macro was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	**Illegal read overlay request** A read overlay was requested and the physical device unit specified in the QIO$ macro was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.
IE.PRI	**Privilege violation** The task that issued the request was not privileged to execute that request. For DECtape, this code is returned when a nonprivileged task attempts to read or write a mounted volume directly (that is, IO.RLB, IO.RLV, IO.WLB, or IO.WLV). Also, this code is returned if any task attempts to attach a mounted volume.
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of 0 was specified.

(continued on next page)

Table 6-3 (Cont.)
DECtape Status Returns

Code	Reason
IE.VER	**Unrecoverable error** After the system's standard number of retries has been attempted upon encountering an error, the operation still could not be completed. For DECtape, this code is returned to indicate any of the following conditions. • A parity error was encountered. • The task attempted a forward multiblock transfer past block 577 (1101 (8)). • The task attempted a backward multiblock transfer past block 0.
IE.WLK	**Write-locked device** The task attempted to write on a DECtape unit that was physically write-locked.

6.4.1 DECtape Recovery Procedures

When a DECtape I/O error condition is detected, RSX-11M attempts to recover from the condition by retrying the function as many as five times. Unrecoverable errors are generally parity, mark track, or other errors caused by a faulty recording medium or a hardware malfunction. An unrecoverable error condition also occurs when the system permorms a read or write operation past the last block of the DECtape on a forward operation, or the first block of the DECtape on a reverse operation.

In addition to the standard error conditions, an unrecoverable error is reported when the "rock count" exceeds 8. The rock count is the number of times the DECtape driver reverses the direction of the tape while looking for a block number. Assume that the block numbers on a portion of DECtape are 99, 96, and 101, where one bit was dropped from block number 100, making it 96. If an I/O request is received for block 100 and the tape is positioned at block 99, the driver starts searching forward for block 100. The first block to be encountered is 96 and, because the driver is searching for block 100 in a forward direction and 96 is less than 100, the search continues forward. Block 101 is the next block and, because number 101 is greater than 100, the driver reverses the direction of the tape and starts to search backward. The next block number in this direction is 96, and the direction is reversed again because 100 is greater than 96. To prevent the DECtape from being hung in this position, continually rocking between block numbers 96 and 100, a maximum rock count of 8 has been established.

6.4.2 Select Recovery

If the DECtape unit is in an off-line condition when the driver performs an I/O function, the message shown below is output on the operator's console.

```
*** DTn:  -- SELECT ERROR
```

where n is the unit number of the drive that is currently off line. You should respond by placing the unit to REMOTE. The driver retries the function, from the beginning, once every second. It displays the message once every 15 seconds until the appropriate DECtape unit is selected. A select error may also occur when there are two drives with the same unit number or when no drive has the appropriate unit number.

6.5 PROGRAMMING HINTS

This section contains important information about programming the DECtape driver described in this chapter.

6.5.1 DECtape Transfers

If the transfer length on a write is less than 256 words, a partial block is transferred with zero fill for the rest of the physical block. If the transfer length on a read is less than 256 words, only the number of words specified is transferred. If the transfer length is greater than 256 words, more than one physical block is transferred.

6.5.2 Reverse Reading and Writing

The DECtape driver supports reverse reading and writing, because these functions speed up data transfers in some cases. A block should normally be read in the same direction in which it was written. If a block is read from a DECtape into memory in the opposite direction from that in which it was written, it is reversed in memory (for example, word 255 becomes word 0, and 254 becomes word 1). If this occurs, you must then reverse the data within memory.

6.5.3 Speed Considerations When Reversing Direction

It is possible to reverse direction at any time while reading or writing DECtape. However, you should understand that reversing direction substantially slows down the movement of the tape. Because DECtape must be moving at a certain minimum speed before reading or writing can be performed, a tape block cannot be accessed immediately after reversing direction. Two blocks must be bypassed before a read or write function can be executed, to give the tape unit time to build up to normal access speed. Furthermore, when a request is issued to read or write in a certain direction, the tape first begins to move in that direction, then starts detecting block numbers. The following examples illustrate these principles.

If a DECtape is positioned at block number 12 and the driver receives a request to read block 10 forward, the tape starts to move forward, in the direction requested. When block number 14 is encountered, the driver reverses the direction of the tape, because 14 is greater than 10. The search continues backward, and block numbers 11 and 10 are encountered. Because the direction must be reversed and the driver requires two blocks to build up sufficient speed for reading, block number 9 and 8 are also bypassed in the backward direction. Then the direction is reversed and the driver encounters blocks 8 and 9 forward before reaching block number 10 and executing the read request.

6.5.4 Aborting a Task

If a task is aborted while waiting for a unit to be selected, the DECtape driver recognizes this fact within 1 second.

CHAPTER 7

DECTAPE II DRIVER

7.1 INTRODUCTION

The DECTAPE II (TU58) driver supports TU58 system hardware, providing low-cost, block-replaceable mass storage.

7.1.1 TU58 Hardware

Each TU58 DECTAPE II system consists of one or two TU58 cartridge drives, one tape drive controller, and one DL11-type serial line interface. Each TU58 drive functions as a random access, block-formatted mass storage device. Each tape cartridge is capable of storing 512(10) blocks of 512(10) bytes each. Access time averages 10 seconds. All I/O transfers (commands and data) occur by means of the serial line interface at serial transmission rates of 9600 bps. All read and write check operations are performed by the controller hardware using a 16-bit checksum. The controller performs up to eight attempts to read a block, as necessary, before aborting the read operation and returning a hard error; however, whenever more than one read attempt is required for a successful read, the driver is notified so that it can report a soft error message to the error logger.

7.1.2 TU58 Driver

The TU58 driver communicates with the TU58 hardware by means of a serial line interface (DL11); no other interface is required. All data and command transfers between the PDP-11 system and the TU58 are done with programmed I/O and interrupt-driven routines; NPRs are not supported.

7.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for the TU58. A bit setting of 1 indicates that the described characteristic is true for this device.

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	1	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	1	Mass storage device
7	1	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	1	Device mountable as a FILES-11 volume
15	1	Device mountable

Words 3 and 4 of the buffer are a double-precision number specifying the total number of blocks on the device; this value is 512(10) blocks. Word 5 indicates the default buffer size, which is 512(10) bytes.

7.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for the TU58.

7.3.1 Standard QIO Functions

the QIO macro that are valid for the TU58.

Table 7-1
Standard QIO Functions for the TU58

Format		Function
QIO$C	IO.ATT,...	Attach device
QIO$C	IO.DET,...	Detach device
QIO$C	IO.KIL,...	Cancel I/O requests [1]
QIO$C	IO.RLB,...,<stadd,size,,,lbn>	READ logical block
QIO$C	IO.WLB,...,<stadd,size,,,lbn>	WRITE logical block

1. In-progress operations are allowed to complete when IO.KIL is received. I/O requests that are queued when IO.KIL is received are killed.

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes (must be even and greater than 0).

lbn

The logical block number on the cartridge tape where the data transfer starts (must be in the range of 0-777).

7.3.2 Device-Specific QIO Functions

The device-specific QIO system directive functions that are valid for the TU58 are shown in Table 7-2.

Table 7-2
Device-Specific QIO Functions for the TU58

Format	Function
QIO$C IO.WLC,...,<stadd,size,,,lbn>	WRITE logical block with check
QIO$C IO.RLC,...,<stadd,size,,,lbn>	READ logical block with check
QIO$C IO.BLS!IQ.UMD,...,<lbn>	POSITION tape
QIO$C IO.DGN!IQ.UMD,...	Run internal diagnostics

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes (must be even and greater than 0).

lbn

The logical block number on the cartridge tape where the data transfer starts (must be in the range of 0-777).

Additional details for device-specific QIO functions are provided in the following paragraphs.

7.3.2.1 **IO.WLC** - The IO.WLC function writes the specified data onto the tape cartridge. A checksum verification is then performed by reading the data just written; data is not returned to the task issuing the function. An appropriate status, based on the checksum verification, is returned to the issuing task.

7.3.2.2 **IO.RLC** - The IO.RLC function reads the tape with an increased threshold in the TU58's data recovery circuit. This is done as a check to insure data read reliability.

7.3.2.3 **IO.BLS** - You can use the IO.BLS function for diagnostic purposes to position the tape to the specified logical block number. If you specify IO.BLS, you must use the IQ.UMD subfunction (see Chapter 1).

7.3.2.4 **IO.DGN** - You can use the IO.DGN function for diagnostic purposes to execute the TU58's internal (firmware) diagnostics. Appropriate status information is returned to the issuing task by the I/O status block. If you specify IO.DGN, you must use the IQ.UMD subfunction (see Chapter 1).

7.4 STATUS RETURNS

Table 7-3 lists the error and status conditions that are returned by the TU58 driver.

Table 7-3
TU58 Driver Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IE.DNR	**Device not ready** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation.
IE.IFC	**Illegal function** A function code was specified in an I/O request that is illegal for the TU58.
IE.FHE	**Fatal hardware error**
IE.TMO	**Time-out error** The TU58 failed to respond to a function within the normal time specified by the driver.
IE.VER	**Unrecoverable error** After the system's standard number of retries (8) has been attempted upon encountering an error, the operation still could not be successfully completed.
IE.WLK	**Cartridge write-locked** The task attempted to write on a tape cartridge that is physically write-locked.

CHAPTER 8

MAGNETIC TAPE DRIVERS

8.1 INTRODUCTION

RSX-11M and RSX-11M-PLUS support a variety of magnetic tape devices. Table 8-1 summarizes these devices and subsequent sections describe them in greater detail.

Programming for magnetic tape is quite similar to programming for the magnetic tape cassette (see Chapter 9). Unlike cassette, however, magnetic tape can handle variable-length records.

8.1.1 TE10/TU10/TS03 Magnetic Tape

The TE10/TU10/TS03 consists of a TM11 controller with a TE10, TU10, or TS03 transport. It is a low-cost, high-performance system for serial storage of large volumes of data and programs in an industry-compatible format. All recording is NRZI format.

8.1.2 TE16/TU16/TU45/TU77 Magnetic Tape

The TE16/TU16/TU45/TU77 consists of an RH11/RH70 controller, a TM02 or TM03 formatter, and a TE16/TU16/TU45/TU77 transport. They are quite similar to the TE10/TU10 but are Massbus devices, with a common controller, a specialized formatter, and drives. Recording is either 800 bpi NRZI or 1600 bpi phase-encoded (PE).

8.1.3 TS11/TU80 Magnetic Tape

The TS11 and TU80 are integrated subsystems. Each has a drive, a controller, and a formatter. The hardware is microprocessor controlled for all operations, including I/O transfers, tape motion, and has comprehensive (internal) diagnostic test execution. Recording is 1600 bpi phase-encoded (PE).

The TS11 operates in conventional start and stop mode while the TU80 operates at either low speed (start and stop mode) or high speed (streaming mode). Tape speed is microprocessor controlled.

8.1.4 TSV05 Magnetic Tape

The TSV05 tape subsystem is a Q BUS device. It is an integrated subsystem with a drive, a controller, and a formatter. The hardware is microprocessor controlled for all operations, including I/O transfers, tape motion, and has comprehensive (internal) diagnostic test execution. Recording is 1600 bpi phase-encoded (PE). The TSV05 operates at 25 inches per second.

8.1.5 TK25 Magnetic Tape

The TK25 consists of a TKQ25 controller for the Q-bus and a TK25 streaming tape drive. The integrated subsystem consists of a tape drive and controller/formatter. The TK25 uses a DC600A 1/4 inch tape cartridge and stores data on serial data tracks in a serial serpentine recording method. The TK25 has storage capacity of 60 Mbytes for 8K-byte data records. Data recording is an 8000 bpi, modified GCR (group cyclical recording) method.

8.1.6 TK50 Magnetic Tape

The TK50 is an integrated subsystem that consists of a controller for the Q-bus and a TK50 streaming tape drive. The controller handles all error recovery and correction, and internally buffers multiple outstanding commands. The tape drive reads and writes data on a 1/2-inch tape cartridge that is recorded at 6667 bpi on serial data tracks in a serial serpentine recording (Modified Frequency Modulation) method. The tape speed is 75 inches per second in streaming mode and the storage capacity is approximately 100 Mbytes irrespective of record size.

8.1.7 TU81 Magnetic Tape

The TU81 is a nine track streaming tape drive that reads and writes data at either 6250 bpi (GCR) or 1600 bpi (PE) on half-inch tape. The TU81 internally buffers multiple outstanding commands. The tape trasnsport speed is 25 or 75 inches per second and is microprocessor controlled. At 6250 bpi density, the drive can store up to 140 Mbytes on a standard 2400-foot reel. The TU81 has its own UNIBUS controller (one drive per controller).

Table 8-1
Standard Magnetic Tape Devices

Device Driver	Channels	Recording Density (Frames/ Inch)	Tape Speed (Inches/ Second)	Maximum Data Rate	Recording Tansfer Method (Bytes/ Second)
TE10 TU10 MTDRV	9 7 or 9	7-channel: 200, 556, or 800 9-channel: 800	45	36,000	NRZI
TE16,TU16 MMDRV	9	800 or 1600	45	800 bpi: 36,000 1600 bpi: 72,000	NRZI or PE[5]
TU45 MMDRV	9	800/1600	75	800 bpi: 60,000 1600 bpi: 120,000	NRZI or PE[5]
TU77 MMDRV	9	800/1600	125	800 bpi: 100,000 1600 bpi: 200,000	NRZI or PE[5]
TS03 MTDRV	9	800	15	12,000	NRZI

5. Phase encoding

(continued on next page)

Table 8-1 (Cont.)
Standard Magnetic Tape Devices

Device Driver	Channels	Recording Density (Frames/ Inch)	Tape Speed (Inches/ Second)	Maximum Data Rate	Recording Tansfer Method (Bytes/ Second)
TS11 MSDRV	9	1600	45	72,000	PE [5]
TU80 MSDRV	9	1600	25 [1] 100 [2]	40,000(1) 160,000(2)	PE [5]
TU81 MUDRV	9	1600/6250	25 [1] 75 [2] 25 [1] 75 [2]	40,000 120,000 156,000 469,000	PE [5] PE [5] GCR GCR
TSV05 MSDRV	9	1600	25	40,000	PE [5]
TK25 MSDRV	s.s. [3]	8000	55	55,000 bit-serial data tracks recorded serial serpentine	Modified GCR
TK50 MUDRV	s.s. [3]	6667	75 [4]	45,000 bit-serial data tracks recorded serial serpentine	Modified FM

1. Low speed
2. High Speed
3. Serial serpentine
4. In streaming mode
5. Phase encoding

8.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for magnetic tapes. A bit setting of 1 indicates that the described characteristic is true for magnetic tapes.

Bit	Setting	Meaning
0	0 or 1	Record-oriented device (0 if the tape is mounted, 1 if it is not)
1	0	Carriage-control device
2	0	Terminal device
3	0	File-structured device
4	0 or 1	Single-directory device (0 if the tape is not mounted, 1 if it is)
5	1	Sequential device
6	1	Mass storage device
7	0 or 1	User-mode diagnostics supported(1)
8	0 or 1	Massbus device (set only for TE16, TU16, TU45, TU77 drives interfaced by means of an RH70 controller)[1]
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0 or 1	Device mountable as a FILES-11 volume[1]
15	0 or 1	Device mountable[1]

1. System generation and device-dependent characteristic.

Word 3 is used by Digital Equipment Corporation for tape density information. Word 4 of the buffer is undefined; word 5 indicates the default buffer size; for magnetic tapes it is 512 bytes.

8.3 QIO$ MACRO

This section summarizes standard and device-specific QIO$ functions for the magnetic tape drivers.

8.3.1 Standard QIO$ Functions

Table 8-2 lists the standard functions of the QIO$ macro that are valid for magnetic tape.

Table 8-2
Standard QIO$ Functions for Magnetic Tape

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	Read logical block (read tape into buffer)
QIO$C IO.RVB,...,<stadd,size>	Read virtual block (read tape into buffer)
QIO$C IO.WLB,...,<stadd,size>	Write logical block (write buffer contents to tape)
QIO$C IO.WVB,...,<stadd,size>	Write virtual block (write buffer contents to tape)

stadd

The starting address of the data buffer. It may be on a byte boundary for MSDRV devices. Ohterwise, it must be on a word boundary.

size

The data buffer size in bytes. Size must be even, greater than 0, and, for a write, must be at least 14 bytes. For MSDRV or MUDRV devices, the data transfer size may be an odd or even number of bytes.

8.3.1.1 **IO.KIL** - IO.KIL causes I/O termination upon the occurrence of:

- A select error (not applicable to TK50)
- Error recovery
- Interrupt servicing
- Driver timeout servicing

For the TK50/TU81, select errors do not occur but I/O in progress is canceled by IO.KIL.

8.3.2 Device-Specific QIO$ Functions

Table 8-3 lists the device-specific functions of the QIO$ macro that are valid for magnetic tape. Additional details on certain functions appear below.

8.3.2.1 **IO.RLV** - The data appears in the specified buffer in a fashion identical with IO.RLB or IO.RVB, as long as the data block has the same length as the buffer.

8.3.2.2 **IO.RWD** - Completion of IO.RWD means that the rewind has been initiated, but for MSDRV (MS:) devices that the rewind to BOT has been completed. Additional requests for operations on that controller may then be queued by the driver until load point (BOT) is reached.

8.3.2.3 **IO.RWU** - You normally use IO.RWU when operator intervention is required (for example, to load a new tape). The operator must turn the unit back on line manually before subsequent operations can proceed.

Table 8-3
Device-Specific QIO$ Functions for Magnetic Tape

Format	Function
QIO$C IO.DSE,...	Data Security Erase (TK50/TU81 only)
QIO$C IO.EOF,...	Write end-of-file mark (tape mark)
QIO$C IO.ERS,...	Erase (TE10 and TU10 not supported)
QIO$C IO.RLV,...,<stadd,size>	Read logical block reverse (TE10 and TU10 not supported.)
QIO$C IO.RWD,...	Rewind unit
QIO$C IO.RWU,...	Rewind and turn unit offline
QIO$C IO.SEC,...	Sense tape characteristics
QIO$C IO.SMO,...,<cb>	Mount tape and set tape characteristics (Unit must be ready with tape at load point.)
QIO$C IO.SPB,...,<nbs>	Space blocks
QIO$C IO.SPF,...,<nes>	Space files
QIO$C IO.STC,...,<cb>	Set tape characteristics

cb

The characteristic bits to set.

nbs

The number of blocks to space past (positive if forward, negative if reverse).

nes

The number of EOF marks to space past (positive if forward, negative if reverse).

size

The size of the stadd data buffer in bytes. The size must be an even number of bytes greater than 0, and it must be at least 14 bytes for a write. For MSDRV or MUDRV devices, data transfers may be an odd or even number of bytes.

stadd

The starting address of the data buffer. It may be on a byte boundary for MSDRV devices, but otherwise it must be on a word boundary.

8.3.2.4 **IO.ERS** - Causes an erase of 3 inches of (write blank) tape, effectively providing an extended interrecord gap. (Not supported on TU10 and TE10.)

8.3.2.5 **IO.DSE** - Causes the TK50 and TU81 to erase from the current position to end-of-tape and then rewind the tape to beginning-of-tape.

8.3.2.6 **IO.SEC** - Causes a return of the tape characteristics in the second I/O status word. The tape characteristic bits are defined as follows:

Bit	Meaning When Set	Can Be Set by IO.SMO and IO.STC
0	For TU10, 556 bpi density (7-channel). Reserved for TE16, TU16, TU45, TU77, TU81 and TS11, TK25, TK50.	X
1	For TU10, 200 bpi density (7-channel). For TS11, TU80, and TSV05, TSU05, TK25, swap byte mode (read/write). Data buffer size should be in even bytes. Reserved for TE16, TU16, TU45, TK50, TU77, TU81.	X

Bit	Meaning When Set	Can Be Set by IO.SMO and IO.STC
2	For TU10, core-dump mode (7-channel, see below). Reserved for TE16, TU16, TS11, TU45, TU77, TU80, TU81, TSV05, TSU05, TK25, and TK50.	X
3	For TU10, even parity (default is odd). For others, odd parity. (Not selectable for the TS11, TK50, TU80, TU81.)	X
4	Tape is past EOT.	
5	Last tape command encountered EOF in a forward tape direction.	
6	Writing is prohibited.	X
7	Writing with extended inter-record gap is prohibited (that is, no recovery is attempted after write error).	X
8	Select error on unit (not on TK50, TU81.)	
9	Unit is rewinding.	
10	Tape is physically write-locked.	
11	For TE10, TU10, TK50 and TS03, reserved.	X
	For the TU81, default 6250 bpi. If bit 11 is set, 1600 bpi.	
	For all other tapes, default is 800 bpi. If bit 1 is set, 1600 bpi density.	
12	For TU10, drive is 7-channel. For all other tapes, reserved.	
13	Tape is at load point (BOT).	
14	Tape is at end-of-volume (EOV).	
15	Tape is past EOV (reserved for driver; always 0 when read by your task).	

In core-dump mode (TU10 only, 800 bpi density, and 7-channel), each 8-bit byte is written on 2 tape frames, 4 bits per frame. In other modes on 7-channel tape, only 6 low-order bits per byte are written.

For the TS11/TU80/TSV05/TSU05 1600 bpi density is always selected (bit 11=1). Bit 11 cannot be modified by either the IO.SMO or IO.STC functions. For drives that use the TM03 controller, this bit can be either set or cleared; however, once the tape is moved from the load (beginning of tape) position (BOT), the device driver modifies this bit to reflect the actual density of the tape currently mounted. You cannot change bit 11 once the tape is moved beyond BOT.

8.3.2.7 **IO.SMO** - Use this function as a combination of the sense (IO.SEC) and set (IO.STC) tape characteristics functions. Unlike IO.STC, however, the IO.SMO function requires that the unit be ready and the tape be at load point (BOT). If either of these conditions is not met, the function returns an error status code of IE.FHE (refer to Table 8-4).

You should use the IO.SMO function to set the characteristics of a newly loaded tape. If the IE.FHE error code is returned, the tape drive is not on line and is not at BOT.

8.4 STATUS RETURNS

The error and status conditions listed in Table 8-4 are returned by the magnetic tape drivers described in this chapter.

Table 8-4
Magnetic Tape Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO$ directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing. This code is also returned if nbs equals 0 in an IO.SPB function or if nes equals 0 in an IO.SPF function.
IS.PND	**I/O request pending** The operation specified in the QIO$ directive has not yet been completed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The specified I/O operation was canceled by IO.KIL while in progress or while still in the I/O queue.
IE.BBE	**Bad block** A bad block was encountered while reading or writing and the error persists after nine retries. For TM11, IE.BBE may also indicate that a bad tape error (BTE) has been encountered. The status return IE.BBE does not apply to MSDRV or MUDRV devices.
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a buffer, while only word alignment is legal for the QIO. Alternatively, the length of a buffer is not an even number of bytes.

(continued on next page)

Table 8-4 (Cont.)
Magnetic Tape Status Returns

Code	Reason
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DAO	**Data overrun** On a read, a record exceeded the stated buffer size. The final portion of the buffer is checked for parity, but is transfered into memory.
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	**Device not ready** The physical device unit specified in the QIO$ directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: o A time-out occurred on the physical device unit (that is, an interrupt was lost). o A vacuum failure occurred on the magnetic tape drive. o While trying to read or space, the driver detected blank tape. o The LOAD switch on the physical drive was switched to the off position. o The unit failed internal diagnostic tests (TS04 only)
IE.EOF	**End-of-file encountered** An end-of-file (tapemark) was encountered.
IE.EOT	**End-of-tape encountered** The end-of-tape (physical end-of-volume) was encountered while the tape was moving in the forward direction for a write or a write tape mark operation. The IE.EOT code is returned continually in the I/O status block until the EOT marker is passed in the reverse direction. IE.EOT is not returned on a read operation.

(continued on next page)

Table 8-4 (Cont.)
Magnetic Tape Status Returns

Code	Reason
IE.EOT	A ten foot length of tape extends past the EOT marker, which is useful for writing data and markers, such as volume trailer labels. The physical end-of-tape for MUDRV (MU:) devices is defined as the end of usable recorded area, which is located in the tape trailer area. This area begins at the EOT marker and extends through a length that depends on the tape format. This length must be long enough to to store the aggregate of the following records: o Two device dependent "maximum recommended record length" records o Three 80-byte records o Three tape marks
IE.EOV	**End-of-volume encountered (unlabeled tape)** On a forward space function, the logical end-of-volume was encountered. An end-of-volume is two consecutive end-of-file marks (EOF), or a beginning-of-tape mark (BOT) followed by an EOF. The tape is normally left positioned between the two marks.
IE.FHE	**Fatal hardware error** Nonrecoverable hardware error; for example, the magnetic tape unit is not ready or the tape is not at load point, or both, when IO.SMO is issued.
IE.IFC	**Illegal function** An invalid function (or subfunction bit) was specified in a magnetic tape I/O request. Refer also to Section 8.4.3.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO$ directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. For magnetic tape, this code is also returned if a byte count of 0 was specified or if your task attempted to write a block that was less than 14 bytes long.

(continued on next page)

Table 8-4 (Cont.)
Magnetic Tape Status Returns

Code	Reason
IE.VER	**Unrecoverable error** After the system's standard number of retries has been attempted upon encountering an error, the operation still could not be completed. For magnetic tape, this code is returned in the case of CRC or checksum errors or when a tape block could not be read.
IE.WLK	**Write-locked device** The task attempted to write on a magnetic tape unit that was physically write-locked. Alternatively, tape characteristic bit 6 was set by the software to write-lock the unit logically.

After processing a QIO$ request, the magnetic tape driver returns two status words. The first word contains one of the I/O status codes listed in Table 8-4.

For successful QIO$ execution (IS.SUC) or read requests (IE.DAO), the second I/O status word may contain further information. The operations for which this is true, and the information returned, are shown in Table 8-5. For all other cases this word is undefined.

Table 8-5
Information Contained in the Second I/O Status Word

I/O Function Code	Information Returned IS.SUC	IE.DAO
IO.RLB	Number of bytes transferred	Number of bytes in tape record
IO.RLV	Number of bytes transferred	Number of bytes in tape record
IO.RVB	Number of bytes transferred	Number of bytes in tape record
IO.SEC	Tape characteristics word	
IO.SPB	Number of records spaced over	
IO.SPF	Number of files spaced over	
IO.WLB	Number of bytes transferred	
IO.WVB	Number of bytes transferred	

8.4.1 Select Recovery

If a request fails because the desired unit is off line, no drive has the desired unit number, or has its power off, the following message is output on the operator's console:

```
*** MTn: -- SELECT ERROR

    or

*** MSn: -- SELECT ERROR
```

n

The unit number of the specified drive.

The driver checks the unit for readiness and repeats the message every 15 seconds until the requesting task is aborted or the unit is made available. In the latter case, the driver then proceeds with the request.

MUDRV devices (TK50) do not issue select errors. If the drive is taken offline, the condition is treated as tape position lost.

8.4.2 Retry Procedures for Reads and Writes

If an error occurs during a read (for example, vertical parity error), the recovery procedure depends on the type of magnetic tape in use. Read errors for MT: or MM: device are retried by backspacing one record and then rereading the record in question. If the error persists after nine retries, IE.VER is returned.

Read errors for the MSDRV (MS:) devices are retried by rereading the block in error a predetermined number of times. For MS: devices, except for TK25, on every eighth reread the block is passed by the tape cleaner blade. If the error persists after a predetermined number of retries, IE.VER is returned.

For MUDRV devices (TK50/TU81) the controller handles error correction and recovery. Except for MU: devices, write recovery is the same for all devices. When a write operation fails, the driver attempts the following error recovery procedure:

1. Repositions the tape
2. Erases three inches of tape (resulting in an extended interrecord gap)
3. Retries the write operation

If the error persists after a predetermined number of retries, IE.VER is returned. The requesting task can use IO.STC to prohibit writing with an extended interrecord gap. In this case, the tape is backspaced and the write is retried.

8.4.3 Powerfail Recovery for Magnetic Tapes

If a power failure or loss of vacuum, or both, occurs on a magnetic tape drive, tape position is lost. (Note that an initial system boot simulates a recovery from a power failure.) Additionally, on auto-load drives, the tape is positioned at BOT when the unit is turned on line.

To prevent accidental destruction of data currently on tape, the driver maintains a power-fail status indicator. When this indicator is set, the driver disallows any data transfer or tape motion commands until a rewind (IO.RWD), rewind unload (IO.RWU), or mount and set characteristics (IO.SMO) function is issued. These functions clear the power-fail indicator and allow all tape functions to be issued. It is also possible to issue the set and sense characteristics functions (IO.STC and IO.SEC) while the power-fail indicator is set. These functions, however, do not clear the bit.

All functions other than those just described are considered invalid and cause the return of the IE.IFC (invalid function) error code to the requesting task. In situations where a tape is currently a mounted volume, the tape should be dismounted and then remounted before use. In doing this, the rewind command is issued, thereby clearing the power-fail indicator.

8.5 PROGRAMMING HINTS

This section contains important information about programming the magnetic tape drivers described in this chapter.

8.5.1 Issue Power-Fail QIOs for TM11 Before GLUN$

The TM11A/B device driver dynamically updates the system data base to reflect the density characteristics of the TE10/TU10. You should issue the QIO$ functions valid for powerfail before requesting the device's density characteristics with the GLUN$ directive.

8.5.2 Block Size

Each block must contain an even number of bytes at least 14 for a write and at most 65,534. However, tape usage is more efficient with a larger buffer.

8.5.3 Importance of Resetting Tape Characteristics

A task that uses magnetic tape should always set the tape characteristics to the proper value before beginning I/O operations. The task cannot be certain in what state a previous task left these characteristics. It is also possible that an operator might have changed the magnetic tape unit selection. If the selection switch is changed, the new physical device unit may not correspond to the characteristics of the unit described by the respective unit control block.

8.5.4 Aborting a Task

If you abort a task while it waits for a magnetic tape unit to be selected, the magnetic tape driver recognizes the abort request within 1 second.

If you abort a task while it waits for a magnetic tape unit to complete a space operation, the magnetic tape driver may allow spacing to the next tape mark.

For the TK50, if you abort a task while it waits for a magnetic tape unit to complete a space operation, the driver may have spaced some or all of the requested number of spaces.

8.5.5 Writing an Even-Parity Zero-NRZI

If an even-parity 0 were written normally, it would appear to the drive as blank tape. It is therefore converted to 20 (octal). If this conversion is undesirable, you must ensure that no even-parity 0s are output on the tape.

8.5.6 Density Selection

The TM03 controller imposes the following density selection restriction: You cannot mix recording densities on any volume associated with the controller.

Density for write operations is selected when the tape is at the load (BOT) position. Hardware selects the density for read operations during the first read (away from BOT); after the first read, you can determine (sense) tape density by using the IO.SEC function.

8.5.7 End-of-Volume Status (Unlabeled Tape)

The magnetic tape driver detects end-of-volume when it spaces over the second of two consecutive tape marks. The tape is left positioned between the two tape marks.

The magnetic tape driver returns the IE.EOV status code only on space operations. IE.EOV is never returned by read operations.

For the purpose of checking for end-of-volume, the driver treats beginning of tape (BOT) as a tape mark. Therefore, any forward space operation from BOT that immediately encounters a tape mark returns IE.EOV.

If a space operation stops between two tape marks but does not space over the second one, the driver returns end of file rather than end-of-volume. Any subsequent space operation from this point that immediately spaces over the second tape mark returns end-of-volume. During IO.SPF operations, the driver considers all tape marks to be files except for BOT and for the second tape mark spaced over at the end of volume.

Note that both IO.SPF and IO.SPB operations leave the tape positioned after the tape mark in the direction of travel.

If you want to treat two consecutive tape marks as end-of-volume on read operations, your application must keep track of the tape marks. The magnetic tape driver does not support two consecutive tape marks as end-of-volume on read operations.

8.5.8 Resetting Tape Transport Status or VCK

For an MS: device, if the tape transport status changes (goes on-line or off-line), further I/O operations are inhibited. A deliberate I/O sequencing must occur to reset the hardware volume check (VCK) indicator and allow physical I/O to proceed. This sequencing is done by a sucessful IO.RWD or IO.SMO QIO$ or including /RW or /REW switches to command requests (such as DMP).

Similarly, for a TK50 or TU81, if the tape transport status changes (goes on-line or off-line), further I/O operations are inhibited. A deliberate I/O sequencing must occur to allow physical I/O to proceed. This sequencing is done by a successful IO.RWD or IO.SMO QIO$ or including /RW or /REW switches to command requests (such as DMP).

8.5.9 Issuing QIO$s

Users issuing QIO$s directly to MSDRV/MUDRV must be aware of the following:

- Completion of an IO.RWD request occurs when the MS: device reaches BOT.
- Completion of an IO.RWD request occurs when the MU: device starts the rewind.
- When the MS: or MU: device changes status from off-line to on-line or vice versa, the MS: or MU: device inhibits further physical I/O operations. After such a change, the user must issue IO.RWD or IO.SMO requests that succeed before I/O can proceed.
- For the MS: or MU: device, read/write data transfer features are:
  - The data buffer starting address must be on a word boundary.
  - The data transfer size may be an odd or even byte count. The minimum must be 14 bytes.
  - For the MSDRV, you can swap odd and even data bytes by using the tape characteristic bit 1 of IO.SMO or IO.STC requests. When bit 1 is set to 0, no byte swap occurs; when it is set to 1, byte swap does occur. If you use byte swapping, it is recommended that the data buffer size be an even byte count.
- For MU: devices, issuing an IO.KIL terminate the in-progress I/O operations in reverse order.

- **CAUTION** -- The MU: device handles QIO$ requests in a different manner than other devices do. Multiple requests are queued in the controller itself and, therefore, the physical end-of-tape may be reached before all requests are processed. Thus, with multiple QIO$s it is possible to pull tape off the supply reel.

  It is recommended that QIOW$ be used, or that the total size of queued records to be written is not longer than the ANSI standard for the tape trailer size.

  The physical end-of-tape for MUDRV (MU:) devices is defined as the end of usable recorded area, which is located in the tape trailer area. This area begins at the EOT marker and extends through a length that depends on the tape format. This length must be long enough to to store the aggregate of the following records:

  - Two device dependent "maximum recommended record length" records
  - Three 80-byte records
  - Three tape marks

8.6 BLOCK SIZE ON TAPES MOUNTED /NOLABEL

Under certain conditions, if a file is written to a tape, its block size will be even and one more than the value specified in the MOUNT command. This conditions where this occurs are as follows:

- The tape is mounted /NOLABEL
- The MOUNT command specifies an odd record size
- The MOUNT command specifies an odd block size

FCS adds the padding character, an octal 136 (^) circumflex, to odd-sized blocks due to a hardware restriction; some tape drives will not allow an odd number of bytes to be transfered to or from tape. Therefore, blocks of data are padded with the circumflex character so that blocks of data can be written to tape on any tape drive.

CHAPTER 9

CASSETTE DRIVER

9.1 INTRODUCTION

RSX-11M supports the TA11 magnetic tape cassette (a TA11 controller with a TU60 dual transport). Programming for cassette is quite similar to programming for magnetic tape (see Chapter 8). The TA11 system is a dual-drive, reel-to-reel unit that replaces paper tape. Its two drives run nonsimultaneously, using DIGITAL Proprietary Philips-type cassettes.

The maximum capacity of a cassette, in bytes, is 92,000 (minus 300 per file gap and 46 per interrecord gap). It can transfer data at speeds of up to 562 bytes per second. Recording density ranges from 350 to 700 bits per inch, depending on tape position.

9.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for cassettes. A bit setting of 1 indicates that the described characteristic is true for cassettes.

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	File structured device
4	0	Single-directory device
5	1	Sequential device
6	1	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing

Bit	Setting	Meaning
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, for cassettes 128 bytes.

9.3 QIO$ MACRO

This section summarizes standard and device-specific QIO functions for the cassette driver.

9.3.1 Standard QIO Functions

Table 9-1 lists the standard functions of the QIO macro that are valid for cassette.

Table 9-1
Standard QIO Functions for Cassette

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	READ logical block (read tape into buffer)
QIO$C IO.RVB,...,<stadd,size>	READ virtual block (read tape into buffer)
QIO$C IO.WLB,...,<stadd,size>	WRITE logical block (write buffer contents to tape)
QIO$C IO.WVB,...,<stadd,size>	WRITE virtual block (write buffer contents to tape)

stadd

The starting address of the data buffer (may be on a byte boundary).

size

The data buffer size in bytes (must be greater than 0).

IO.KIL does not affect in-progress requests.

9.3.2 Device-Specific QIO Functions

Table 9-2 lists the device-specific functions of the QIO macro that are valid for cassette. The section on programming hints below provides more detailed information about certain functions.

Table 9-2
Device-Specific QIO Functions for Cassette

Format	Function
QIO$C IO.EOF,...	Write end-of-file gap
QIO$C IO.RWD,...	Rewind unit
QIO$C IO.SPB,...,<nbs>	SPACE blocks
QIO$C IO.SPF,...,<nes>	SPACE files

nbs

The number of blocks to space past (positive if forward, negative if reverse).

nes

The number of EOF gaps to space past (positive if forward, negative if reverse).

9.4 STATUS RETURNS

The error and status conditions listed in Table 9-3 are returned by the cassette driver described in this chapter.

Table 9-3
Cassette Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed if the operation involved reading or writing, or the number of blocks or files spaced if the operation involved spacing blocks or files.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The specified I/O operation was canceled by IO.KIL while still in the I/O queue.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DAO	**Data overrun** The driver was not able to sustain the data rate required by the TAll controller.
IE.DNA	**Device not attached** The physical device unit specified by an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	**Device not ready** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: • The cassette has not been physically inserted. • The unit is off line. • A time-out occurred on the physical device unit (that is, an interrupt was lost).

(continued on next page)

Table 9-3 (Cont.)
Cassette Status Returns

Code	Reason
IE.EOF	**End-of-file encountered** An end-of-file gap was recognized on the cassette tape. This code is returned if an EOF gap is encountered during a read, or if the cassette is physically removed during an I/O operation.
IE.EOT	**End-of-tape encountered** While reading or writing, clear trailer at end-of-tape (EOT) was encountered. Unlike magnetic tape, writing beyond EOT is not permitted on cassettes. This condition is always sensed on a write before it would be sensed on a read of the same section of tape. If IE.EOT is returned during a write, the cassette head has encountered EOT before finishing the writing of the last block. Your task must entirely rewrite the block on another cassette..
IE.IFC	**Illegal function** A function code was specified in an I/O request that is illegal for cassette.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternatively, a byte count of 0 was specified on a transfer.
IE.VER	**Nonrecoverable error** This code is returned when a block check error occurs (see Section 9.6.5). The cyclic redundancy check (CRC), a 2-byte value located at the end of each block, is a checksum that is tested during all read operations to ensure that data is read correctly. This is returned if a read request did not specify exactly the number of bytes of data in the record on tape. If a nonrecoverable error is returned, your task may attempt recovery by spacing backward one block and retrying the read operation.
IE.WLK	**Write-locked device** The task attempted to write on a cassette unit that was physically write-locked. This code may be returned after an IO.WLB, IO.WVB, or IO.EOF function.

9.4.1 Cassette Recovery Procedures

If an error occurs during a read or write operation, the operation should be retried several times. The recommended maximum number of retries is nine for a read and three for a write because each retry involves backspacing, which does not always position the tape in the same place. More than three retries of a write operation may destroy previously written data. For example, to retry a write, it is best to space two blocks in reverse, then space one block forward. This insures the tape is in the proper position to rewrite the block that encountered the error.

After read and write functions, the second I/O status word contains the number of bytes actually processed by the function. After spacing functions, it contains the number of blocks or files actually spaced.

9.5 STRUCTURE OF CASSETTE TAPE

Figure 9-1 illustrates a general structure for cassette tape. A different structure can be employed if you want.

Here the tape consists of blocks of data interspersed with sections of clear tape that serve as leader, trailer, interrecord gaps (IRGs), and end-of-file gaps.

The logical end-of-tape in this case consists of a sentinel label record, rather than the conventional group of end-of-file gaps. Each file must contain at least one block. The size of each block depends upon the number of bytes your task specifies when writing the block.

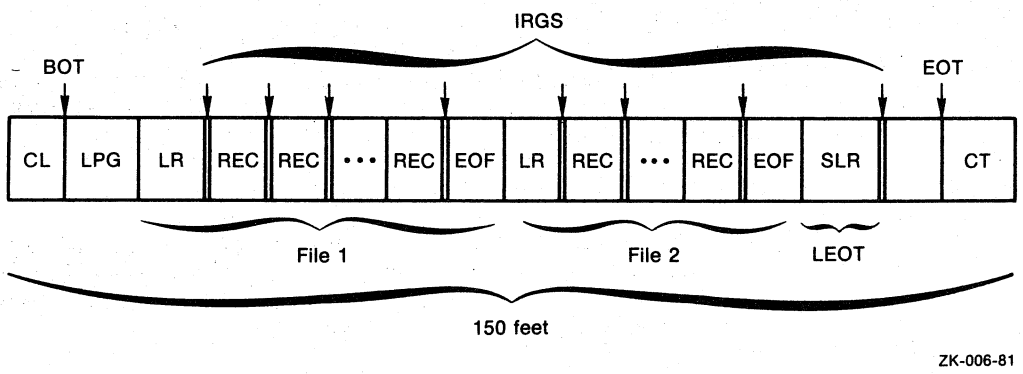


Figure 9-1 Structure of Cassette Tape

Abbreviation	Meaning
CL	Clear leader
BOT	Physical beginning-of-tape
LPG	Load point gap (blank tape written by driver before the first retrievable record)
LR	File label record
REC	Fixed-length record (data)

Abbreviation	Meaning
EOF	End-of-file gap
IRG	Interrecord gap
SLR	Sentinel label record
LEOT	Logical end-of-tape
EOT	Physical end-of-tape
CT	Clear trailer

9.6 PROGRAMMING HINTS

This section contains important information about programming the cassette driver described in this chapter.

9.6.1 Importance of Rewinding

The first cassette operation performed on a tape must always be a rewind to ensure that the tape is positioned to a known place. When it is positioned in clear tape, there is no way to determine whether it is in leader at the beginning-of-tape (BOT) or in trailer at the end-of-tape (EOT).

9.6.2 End-of-File and IO.SPF

The hardware senses end-of-file (EOF) as a time-out. When IO.SPF is issued in the forward direction (nes is positive), the tape is positioned two-thirds of the way from the beginning of the final file gap. In effect, this is all the way through the file gap. When IO.SPF is issued in the reverse direction (nes is negative), the tape is positioned one-third of the way from the beginning of the final file gap (that is, two-thirds of the way from the beginning of the last file spaced). Therefore, to correctly position the tape for a read or write after issuing IO.SPF in reverse, your task should issue IO.SPB forward for one block, followed by IO.SPB in reverse for one block.

9.6.3 The Space Functions IO.SPB and IO.SPF

IO.SPB always stops in an IRG gap, IO.SPF in an EOF gap. Neither space function actually takes effect until data is encountered. For example, suppose the tape is positioned in clear leader at BOT and your task requests that one block be spaced forward. The drive passes over the remaining leader until it reaches data, passes one block, and stops in the IRG. Similarly, if the same command is issued when the tape is at BOT on a blank tape or a tape containing only EOF gaps, the function does not terminate until EOT.

9.6.4 Verifying of Write Operations

Certain errors, such as cyclic redundancy check, are detected on read but not write operations. Therefore, to ensure reliability of recording, your task should perform a read to verify every write operation.

9.6.5 Block Length

You must specify the exact number of bytes per block when requesting read or write operations. An attempt to read a block with an incorrect byte count causes an unrecoverable error (see Section 9.4) to occur.

9.6.6 Logical End-of-Tape

The conventional method of signaling logical end-of-tape by multiple EOF gaps is inadequate for cassettes, because multiple EOF gaps are not distinguishable from each other. For example, two sequential EOF gaps would be read as three instead of two. Also spacing functions, because they are triggered by encountering data, can not recognize multiple EOF gaps. Consequently, the use of a sentinel or key record to signal logical end-of-tape is recommended.

CHAPTER 10

LINE PRINTER DRIVER

10.1 INTRODUCTION

The RSX-11M/M-PLUS line printer driver supports the line printers summarized in Table 10-1. Subsequent sections of this chapter describe these printers in greater detail.

Table 10-1
Standard Line Printer Devices

Controller	Printer	Column Width	Character Set	Lines per Minute
KMC-11-A Auxiliary Processor				
LP11-C	LP14-C	132	64	890
LP11-D	LP14-D	132	96	650
LP11-F	LP01-F	80	64	170-1110
LP11-H	LP01-H	80	96	170-1110
LP11-J	LP02-J	132	64	170-1110
LP11-K	LP02-K	132	96	170-1110
LP11-R	LP04-R	132	64	1110
LP11-S	LP04-S	132	96	1110
LP11-V	LP05-V	132	64	300
LP11-W	LP05-W	132	96	300
LP11-Y	LP06-Y	132	64	600
LP11-Z	LP06-Z	132	96	460
LP11-GA	LP07	132	96	1200
LP11-EA	LP26	132	64	600
LP11-EB	LP26	132	64/96	600/420
LP11-UA	LP27	132	64/96	1200/800

(continued on next page)

Table 10-1 (Cont.)
Standard Line Printer Devices

Controller	Printer	Column Width	Character Set	Lines per Minute
KMC-11-A Auxiliary Processor				
LS11	LS11	132	62	60-200
LV11	LV01	132	96	500
LA180	LA180	132	96	150
LN01	LN01	Variable	—[1]	600

1. Software selectable fonts not supported by RSX.

10.1.1 KMC-11 Auxiliary Processor

The KMC-11 controller is a microcode-controlled printer controller that supports up to 8 line printers. Multiple KMC-11 controllers are allowed. The KMC-11 provides higher performance printing than other controllers and, at the same time, uses fewer CPU resources. The use of the KMC-11 controller is a system generation option.

10.1.2 LP11 Line Printer

The LP11 is a high-speed line printer available in a variety of models. The LP11 model line consists of band line printers and drum line printers. The drum printers are impact printers, that use one hammer per column and a revolving drum with uppercase and optional lowercase characters. The LP11-R and LP11-S are fully buffered models that operate at a standard speed of 1110 lines per minute. The other LP11 drum models have 20-character print buffers. These printers are therefore able to print at full speed if the printed line is no longer than 20 characters. Lines that exceed this maximum are printed at a slower rate. You may use forms with up to six parts. The band line printers are impact printers that have a flat steel belt with raised metal characters on the face. The LP07, LP26, and LP27 offer speeds from 420 to 1200 lines per minute.

10.1.3 LS11 Line Printer

The LS11 is a medium-speed line printer. It has a 20-character print buffer, and lines of 20 characters or less are printed at a rate of 200 lines per minute. Longer lines are printed at a slower rate. RSX-11M does not support the LS11 expanded character set feature.

10.1.4 LV11 Line Printer

The LV11 is a fully-buffered, electrostatic printer-plotter that operates at a standard rate of 500 lines per minute. RSX-11M supports only the LV11 print capability, not the plotter mode.

10.1.5 LA180 DECprinter

The LA180 is a 180-character/sec, dot-matrix impact printer. It accepts multipart forms and pages of various lengths and widths.

10.1.6 LN01 Laser Printer

The LN01 is a non-impact page printer that uses laser imaging combined with xerographic printing. This technology provides letter quality printing at line printer speeds with no noise. Printing is done on standard 8 1/2 inch by 11 inch paper at 12 pages per minute, which equates to 600 lines per minute. Contributing to the high print quality is a printer resolution of 300 by 300 dots per inch. The LN01 offers the speed of a line printer with the advantages of a phototypeset device.

10.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for line printers. A bit setting of 1 indicates that the described characteristic is true for line printers.

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	0	Terminal device
3	0	File structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default size for the device, for line printers the width of the printer carriage (that is, 80 or 132).

10.3 QIO$ MACRO

Table 10-2 lists the standard functions of the QIO macro that are valid for line printers.

Table 10-2
Standard QIO Functions for Line Printers

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.WLB,...,<stadd,size,vfc>	WRITE logical block (print buffer contents)
QIO$C IO.WVB,...,<stadd,size,vfc>	WRITE virtual block (print buffer contents)

stadd

The starting address of the data buffer (may be on a byte boundary).

size

The data buffer size in bytes (must be greater than 0).

vfc

A vertical format control character from Table 10-4.

IO.KIL does not cancel an in-progress request unless the line printer is in an off-line condition because of a power failure or a paper jam, or because it is out of paper.

The line printer driver supports no device-specific functions.

10.4 STATUS RETURNS

Table 10-3 lists the error and status conditions that are returned by the line printer driver described in this chapter.

Table 10-3
Line Printer Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved writing.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The specified I/O operation was canceled while in progress or while in the I/O queue.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	**Device not attached** The physical device unit specified an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.IFC	**Illegal function** A function code was specified in an I/O request that is invalid for line printers.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The buffer specified for a write request was partially or totally outside the address space of the issuing task. Alternatively, a byte count of 0 was specified.

10.4.1 Ready Recovery

If any of the following conditions occur:

- Paper jam
- Printer out of paper
- Printer turned off line
- Power failure

the driver determines that the line printer is off line, and the following message is output on the operator's console:

***LPn: -- NOT READY

n
The unit number of the line printer that is not ready.

The driver retries the function that encountered the error condition from the beginning, once every second. It displays the message after m seconds. The value m is defined at system generation to be a value less than 256. The default is 15. The messages occur until you make the line printer ready. If a power failure occurs while printing a line, the entire line is reprinted from the beginning when power is restored.

10.5 VERTICAL FORMAT CONTROL

Table 10-4 summarizes the meaning of all characters that you can use for vertical format control on the line printer. Any one of these characters can be specified as the vfc parameter in an IO.WLB or IO.WVB function.

Table 10-4
Vertical Format Control Characters

Octal Value	Character	Meaning
040	Blank	SINGLE SPACE: Output a line feed, print the contents of the buffer, and output a carriage return. Normally, printing immediately follows the previously printed line.
060	Zero	DOUBLE SPACE: Output two line feeds, print the contents of the buffer, and output a carriage return. Normally, the buffer contents are printed two lines below the previously printed line.
061	One	PAGE EJECT: Output a form feed, print the contents of the buffer, and output a carriage return. Normally, the contents of the buffer are printed on the first line of the next page.

(continued on next page)

Table 10-4 (Cont.)
Vertical Format Control Characters

Octal Value	Character	Meaning
053	Plus	OVERPRINT: Print the contents of the buffer and perform a carriage return, normally overprinting the previous line.
044	Dollar sign	PROMPTING OUTPUT: Output a line feed and then print the contents of the buffer.
000	Null	INTERNAL VERTICAL FORMAT: The buffer contents are printed without addition of vertical format control characters. In this mode, more than one line of guaranteed contiguous output can be printed per I/O request.

All other vertical format control characters are interpreted as blanks (040(octal)).

10.6 PROGRAMMING HINTS

This section contains important information about programming the line printer driver described in this chapter.

10.6.1 RUBOUT Character

The line printer driver discards the ASCII character code 177 during output, because a RUBOUT on the LS11 printer causes a RUBOUT of the hardware print buffer.

10.6.2 Print Line Truncation

If the number of characters to be printed exceeds the width of the print carriage, the driver discards excess characters until it receives one that instructs it to empty the buffer and return to horizontal position 1. You can determine if truncation can occur by issuing a Get LUN Information system directive and examining word 5 of the information buffer. This word contains the width of the print carriage in bytes.

10.6.3 Aborting a Task

If a task is aborted while waiting for the line printer to be readied, the line printer driver recognizes this fact within 1 second.

CHAPTER 11

CARD READER DRIVER

11.1 INTRODUCTION

The RSX-11M card reader driver supports the CR11 card reader. This reader is a virtually jam-proof device that reads EIA standard 80-column punched cards at the rate of 300 per minute. The hopper can hold 600 cards. This device uses a vacuum picker that provides extreme tolerance to damaged cards and makes card wear insignificant. Cards are riffled in the hopper to prevent sticking. The reader uses a strong vacuum to deliver the bottom card. Because it has a very short card track, only one card is in motion at a time.

11.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for card readers. A bit setting of 1 indicates that the described characteristic is true for card readers.

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	File structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked

Bit	Setting	Meaning
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, which is 80 bytes for the card reader.

11.3 QIO$ MACRO

This section summarizes standard and device-specific QIO functions for the card reader driver.

11.3.1 Standard QIO Functions

Table 11-1 lists the standard functions of the QIO macro that are valid for the card reader.

Table 11-1
Standard QIO Functions for the Card Reader

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	READ logical block (alphanumeric)
QIO$C IO.RVB,...,<stadd,size>	READ virtual block (alphanumeric)

stadd

The starting address of the data buffer (may be on a byte boundary).

size

The data buffer size in bytes (must be greater than 0).

IO.KIL does not cancel an in-progress request unless the card reader is in an off-line condition because of a pick, read, stack, or hopper check, because of power failure, or because the RESET button has not been depressed.

11.3.2 Device-Specific QIO Functions

The device-specific functions of the QIO macro that are valid for the card reader are shown in Table 11-2.

Table 11-2
Device-Specific QIO Function for the Card Reader

Format	Function
QIO$C IO.ATA,...,<AST addr>	Attach for unsolicited card AST
QIO$C IO.RDB,...,<stadd,size>	Read logical block (binary)

stadd

The starting address of the data buffer (may be on a byte boundary).

size

The data buffer size in bytes (must be greater than 0).

11.4 STATUS RETURNS

A wide variety of error conditions and recovery procedures relate to the use of the card reader. This section describes the three major ways in which the system reports error conditions.

1. Lights and indicators on the card reader panel are turned on or off to indicate particular operational problems such as read, pick, stack, or hopper checks. Switches are available to turn the reader power on and off and to allow you to reset the error condition after correcting it.

2. A message is output on the operator's console if operational checks or power problems occur.

3. An I/O completion code is returned in the low-order byte of the first word of the I/O status block specified in the QIO macro to indicate success or failure on completion of an I/O function.

The following subsections describe each of these returns in detail.

11.4.1 Card Input Errors and Recovery

The table included below describes all external lights and switches on the reader that indicate to you that a hardware problem has occurred and must be corrected. There are two classes of hardware errors:

- Those requiring you to ready the reader and try the operation again

- Those requiring you to remove the last card from the output stacker, to replace it in the input hopper, and to try the operation again

In the first case, the card reader was unable to read the current card. In the second, the card was read incorrectly and must be physically removed from the output stacker. The card reader driver restarts a read operation within 1 second after the cards have been replaced in the input hopper.

Table 11-3 summarizes the functions of lights and indicators on the front panel of the card reader. It discusses common operational errors that might be encountered while reading cards and recovery procedures associated with these error conditions.

11.4.2 Ready and Card Reader Check Recovery

If any of the following conditions occur:

- Power failure
- Reset switch not pressed (reader off line)
- Timing error (Two columns were read before the card reader driver input the first column from the card reader.)

the driver determines that the card reader is not ready, and the following message is output on the operator's console:

```
*** CRn: -- NOT READY
```

When a timing error occurs, the operator can proceed with normal card reader operation by:

1. Placing the card reader off line by pressing the STOP switch
2. Removing the last card read and inserting it where it is read as the next card
3. Placing the card reader on line by pressing the RESET switch

If any of the following conditions occurs:

- Pick error (PICK CHECK)
- Read error (READ CHECK)
- Output stacker error (STACK CHECK)
- Input hopper out of cards (HOPPER CHECK)
- Output stacker full (HOPPER CHECK)

the driver determines that a card reader check has occurred, and the following message is output on the operator's console:

```
*** CRn: -- READ FAILURE. CHECK HARDWARE STATUS
```

where n is the unit number of the card reader that is not ready. The operator should correct the error and press RESET: The driver attempts the function from the beginning, once every second. It displays the message once every m seconds (m is defined at system generation as a value less than 256. The default is 15) until the card reader is readied. In all cases except pick error, the last card read should be reinserted in the input hopper, as described in Section 11.4.1.

Table 11-3
Card Reader Switches and Indicators

Indicator	Description	Action	Recovery
POWER Switch	Pushbutton indicator switch (alternate action: pressed for both ON and OFF)	Controls application of all power to the card reader. When indicator is off, depressing switch applies power to reader and causes associated indicator to light. When indicator is lit, depressing switch removes all power from reader and causes indicator to go out.	Card may have been read incorrectly; restore power if possible by depressing the POWER switch; insert the card again as the first card in the input hopper, and press the RESET switch; in some cases, it may be necessary to restart the program.
READ CHECK Indicator	White light	When lit, this light indicates that the card just read may be torn on the leading or trailing edges, or that the card may have punches in column positions 0 or 81. Because READ CHECK indicates an error condition, whenever this indicator is lit, it causes the card reader to stop operation and extinguishes the RESET indicator.	Card was read incorrectly; duplicate if necessary, insert the card again as the first card in the input hopper, and press the RESET switch.
PICK CHECK Indicator	White light	When lit, this light indicates that the card reader failed to move a card into the read station after it received a READ COMMAND from the controller. Stops card reader operation and extinguishes RESET indicator.	Card could not be read; press the RESET switch to try again or remove the cards from the input hopper, smooth the leading edges, replace, and then press the RESET switch.

(continued on next page)

Table 11-3 (Cont.)
Card Reader Switches and Indicators

Indicator	Description	Action	Recovery
STACK CHECK Indicator	White light	When lit, this light indicates that the previous card was not properly seated in the output stacker and therefore may be badly mutilated. Stops card reader operation and extinguishes RESET indicator.	Card may have been read incorrectly and is not positioned properly in the output stacker; duplicate the card if it is damaged; insert the card again as the first card in the input hopper and press the RESET switch.
HOPPER CHECK Indicator	White light	When lit, this light indicates that either the input hopper is empty or that the output stacker is full.	Card may have been read incorrectly; empty the stacker or fill the hopper; insert the card again as the fist card in the input hopper and press the RESET switch.
STOP Switch	Momentary pushbutton/ indicator switch (red light)	When depressed, immediately lights and drops the READY line, thereby extinguishing the RESET indicator. Card reader operation then stops as soon as the card currently in the read station has been read. This switch has no effect on the system power; it only stops the current operation.	
RESET Switch	Momentary pushbutton/ indicator switch (green light)	When depressed and released, clears all error flip-flops and initializes card reader logic. Associated RESET indicator lights to indicate that the READY signal is applied to the controller. The RESET indicator goes out whenever the STOP switch is depressed or whenever an error indicator lights (READ CHECK, PICK CHECK, STACK CHECK, or HOPPER CHECK).	

11.4.3 I/O Status Conditions

The error and status conditions listed in Table 11-4 are returned by the card reader driver described in this chapter.

Table 11-4
Card Reader Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The specified I/O operation was canceled while in progress or while still in the I/O queue.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task.
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.EOF	**End-of-file encountered** An end-of-file control card was recognized.
IE.IFC	**Illegal function** A function code was specified in an I/O request that is illegal for card readers.
IE.NOD	**Buffer allocation failure** Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a card buffer (that is, cards are read into a driver buffer, translated, and then moved to your task's buffer).

(continued on next page)

Table 11-4 (Cont.)
Card Reader Status Returns

Code	Reason
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The buffer specified for a read request was partially or totally outside the address space of the issuing task. Alternatively, a byte count of 0 was specified.

11.5 FUNCTIONAL CAPABILITIES

The card reader driver can perform the following functions:

1. Read cards in DEC026 format and translate to ASCII
2. Read cards in DEC029 format and translate to ASCII
3. Read cards in binary format

If the QIO macro specifies the IO.RLB or IO.RVB function, the driver interprets all data as alphanumeric (026 or 029 format). As explained below, control characters indicate whether 026 or 029 is desired. If the QIO macro specifies IO.RDB, the driver interprets all data, including 026 and 029 control characters, as binary.

11.5.1 Control Characters

Table 11-5 lists the multipunched cards that the card reader driver recognizes as control characters. They are never transferred to the buffer of your task or included in the count of transferred bytes in alphanumeric mode. In binary mode, the only control card recognized is binary EOF.

DEC026 is the default translation mode when the system is bootstrapped. This mode remains in effect until explicitly changed by a control card indicating that DEC029 cards follow. After encountering a DEC029 control card, the driver translates all cards in DEC029 format unless another DEC026 control card is encountered. This card overrides the 029 mode specification and indicates that subsequent cards are to be translated in 026 format. Control characters are addressed to the card reader itself, and remain in effect even when the reader is attached and subsequently detached.

The default condition can easily be changed from DEC026 to DEC029 by reading a 029 control card, and then saving the system with the MCR SAV command.

Table 11-5
Card Reader Control Characters

Punches	Columns	Meaning
12-11-0-1-6-7-8-9	1	End-of-file (alphanumeric)
12-11-0-1-6-7-8-9	(All 8 punches in the first 8 columns)	End-of-file (binary)
12-2-4-8	1	026-coded cards follow
12-0-2-4-6-8	1	029-coded cards follow

11.6 CARD READER DATA FORMATS

The card reader reads data in either alphanumeric or binary format.

11.6.1 Alphanumeric Format (026 and 0211)

Table 11-6 summarizes the translation from DEC026 or DEC029 card codes to ASCII.

11.6.2 Binary Format

In RSX-11M binary format, the data are not packed, but are transferred exactly as read, one card column per word. Because each word has 16 bits and each card column represents only 12, the data from the column are stored in the rightmost 12 bits of the word. The word's remaining four bits contain 0s.

11.7 PROGRAMMING HINTS

This section contains important information about programming the card reader driver described in this chapter. Section 11.4 contains information on operational error-recovery procedures that may be important for programming.

11.7.1 Input Card Limitation

Only one card can be read with a single QIO macro call. A request to read more than 80 bytes or columns, the length of a single card, does not result in a multiple card transfer. Only 80 columns are processed. It is possible to read fewer than 80 columns of card input with a QIO read function. For example, you can specify that only the first 10 columns of each card are to be read.

11.7.2 Aborting a Task

If a task waiting for the card reader to be readied is aborted, the card reader driver recognizes this fact within 1 second.

Table 11-6
Translation from DEC026 or DEC029 to ASCII

Character	Non-Parity ASCII	DEC029	DEC026	Character	Non-Parity ASCII	DEC029	DEC026
	173	12 0	12 0	'	054	0 8 3	0 8 3
	175	11 0	11 0	-	055	11	11
SPACE	040	none	none	.	056	12 8 3	12 8 3
!	041	12 8 7	12 8 7	/	057	0 1	0 1
"	042	8 7	0 8 5	0	060	0	0
#	043	8 3	0 8 6	1	061	1	1
$	044	11 8 3	11 8 3	2	062	2	2
%	045	0 8 4	0 8 7	3	063	3	3
AND	046	12	11 8 7	4	064	4	4
'	047	8 5	8 6	5	065	5	5
(	050	12 8 5	0 8 4	6	066	6	6
)	051	11 8 5	12 8 4	7	067	7	7
*	052	11 8 4	11 8 4	8	070	8	8
+	053	12 8 6	12	9	071	9	9
:	072	8 2	11 8 2	M	115	11 4	11 4
;	073	11 8 6	0 8 2	N	116	11 5	11 5
	074	12 8 4	12 8 6	O	117	11 6	11 6
=	075	8 6	8 3	P	120	11 7	11 7
>	076	0 8 6	11 8 6	Q	121	11 8	11 8
?	077	0 8 7	12 8 2	R	122	11 9	11 9
@	100	8 4	8 4	S	123	0 2	0 2
A	101	12 1	12 1	T	124	0 3	0 3
B	102	12 2	12 2	U	125	0 4	0 4
C	103	12 3	12 3	V	126	0 5	0 5
D	104	12 4	12 4	W	127	0 6	0 6
E	105	12 5	12 5	X	130	0 7	0 7
F	106	12 6	12 6	Y	131	0 8	0 8
G	107	12 7	12 7	Z	132	0 9	0 9
H	110	12 8	12 8	[	133	12 8 2	11 8 5
I	111	12 9	12 9	\	134	0 8 2	8 7
J	112	11 1	11 1	]	135	11 8 2	12 8 5
K	113	11 2	11 2	^ or	136	11 8 7	8 5
L	114	11 3	11 3	_ or	137	0 8 5	8 2

CHAPTER 12

MESSAGE-ORIENTED COMMUNICATION DRIVERS

12.1 INTRODUCTION

RSX-11M supports a variety of communication line interfaces: synchronous and asynchronous, single-line and multiplexers, character-oriented and message-oriented. These interfaces enable remote job entry, and terminal, multicomputer, laboratory and industrial control communications. Communications line interfaces can be roughly divided into two categories:

- Terminal (character-oriented) communications devices
- Multicomputer (message-oriented) communications devices

Chapters 1, 2, and 3 describe the character-oriented asynchronous communications line interfaces used primarily for terminal communications. The Terminals and Communications Handbook contains more detail on these devices. This chapter describes in some detail the RSX-11M message-oriented synchronous and asynchronous communication line interfaces. These are used most frequently in multicomputer communications.

Character-oriented communications devices include the DH11, DHV11, DHU11, DZV11, DZQ11, DJ11, DL11-A, DL11-B/C/D, and DZ11 interfaces. These are asynchronous multiplexers and single-line interfaces used almost exclusively for terminal communications. Transfers on all of these interfaces are performed one character at a time. None of the interfaces in this category has a driver of its own (that is, they are supported by the terminal driver), and none can be accessed directly as RSX-11M devices.

Message-oriented communications line interfaces usually link two separate but complementary computer systems. One system must serve as the transmitting device and the other as the receiving device. Devices in this category include the synchronous and asynchronous single-line interfaces summarized in Table 12-1.

The message-oriented communication line interfaces are for transferring large blocks of data.

Whereas the character-oriented interfaces can only be accessed indirectly through the terminal driver, the DA11-B, DL11-E, DMC11, DP11, DQ11, DU11, and DUP11 allow I/O requests to be queued directly for them. These devices have drivers of their own and can be accessed by means of the logical device names listed in Table 12-1. You can use these names in assigning LUNs with the Assign LUN system directive, at task build, or with the REASSIGN MCR command. The following subsections briefly discuss the message-oriented interfaces supported for RSX-11M.

Table 12-1
Message-Oriented Communication Interfaces

Model	Type	Rate (KBaud)	Duplex Half	Duplex Full	Data block (words)	Synchronous Character
DA11-B [1]	Parallel	500	x		32K	No
DL11-E [2]	Serial, asynchronous	0.05-9.6	x	x	32K	Programmable
DMC11	Serial, synchronous	19.2-1000	x	x	8K	No
DP11 [1]	Serial, synchronous	2-19.2	x	x	32K	Programmable
DQ11 [1]	Serial, synchronous	2.4-1000	x	x	32K	Programmable
DU11 [1]	Serial, synchronous	0.05-9.6	x	x	32K	Programmable
DUP11	Serial, synchronous	0.05-9.6	x	x	32K	Programmable

1. Support is not provided on RSX-11M-PLUS systems.

2. DL11-E support is provided on RSX-11M-PLUS systems using the full-duplex terminal driver only.

12.1.1 DA11-B Parallel Interface

The DA11-B provides a bit-parallel, direct memory access interface between two PDP-11 computer systems. Data transfers are performed a word at a time and are made directly between the memories of the two systems. The maximum transfer rate is 500,000 baud, and you can adjust it to match the system configuration requirements. Being a parallel device, the DA11-B does not use sync characters. The interface is half-duplex and transfers data in blocks of up to 32K words.

The DA11-B requires two cooperating computers to effect a data transfer. To control the physical link between the computers, the device driver contains its own simple line protocol. This protocol requires one system to issue a receive QIO$ and the other to issue a transmit QIO$ before any data is actually transferred.

12.1.2 DL11-E Asynchronous Line Interface

The DL11-E is an asynchronous, serial-bit, single-line interface. It is a block-transfer device for remote terminal and multicomputer communications. Baud rates are selectable between 50 and 9600, and full data-set control is supported. Software support for data-set control consists of interlocking RTS and CTS for data transmission, and the setting of DTR (data terminal ready) to enable auto-answer modems to answer incoming calls. DTR is set when an IO.INL QIO$ (initialize) is issued.

12.1.3 DMC11 Synchronous Line Interface

The DMC11 provides a direct memory access interface between two PDP-11 computer systems using the DDCMP line protocol, thus delivering high throughput and reliability while simplifying programming. The DMC11 supports Non-Processor Request (NPR) data transfers of up to 8K words at rates of 1,000,000 baud for local operation (over coaxial cable) and 19,200 baud for remote operation (using modems). Both full- and half-duplex modes are supported. The DMC11 also implements remote load detect, allowing it to reinitialize a halted computer system.

12.1.4 DP11 Synchronous Line Interface

The DP11 provides a program interrupt interface between a PDP-11 and a serial synchronous line. This interface facilitates the use of the PDP-11 in remote batch processing, remote data collection, and remote concentration applications. The modem control feature allows using the DP11 in switched or dedicated configurations.

On the DP11, baud rates are selectable between 2000 and 19,200. You can select a specific sync character to synchronize the transmitting and receiving systems.

12.1.5 DQ11 Synchronous Line Interface

The DQ11 provides a direct memory access interface between a PDP-11 and a serial synchronous line. The direct memory access characteristic of the DQ11 allows the device to operate at speeds higher than those of program interrupt devices, and with a lower interrupt overhead. Modem control of the DQ11 allows using the device in switched or dedicated configurations.

The DQ11 handles data rates from 2400 baud to 1,000,000 baud. The limiting rate is determined by the modem and data set interface level converters.

The DQ11 sync character is programmable in the same manner as the DP11 and the DU11. The maximum data block length transmitted is 65,536 characters.

12.1.6 DU11 Synchronous Line Interface

The DU11 synchronous line interface is a single-line communications device that provides a program-controlled interface between the PDP-11 and a serial synchronous line. The PDP-11 can be interfaced with a high-speed line to perform remote batch processing, remote data collection, and remote concentration applications. Modem control is a standard feature of the DU11 and allows using the device in switched or dedicated configurations. The DU11 transmits data at a maximum rate of 9600 baud; this rate is limited by modem and data set interface level converters.

The DU11 can be programmed to accept any sync character that you define. The use of the sync character is the same for the DU11 and the DP11.

12.1.7 DUP11 Synchronous Line Interface

The DUP11 is identical to the DU11, except that it incorporates hardware to perform cyclic redundancy checking.

12.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for message-oriented communication interfaces. A bit setting of 1 indicates that the described characteristic is true for the interfaces described in this chapter.

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	1	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	1	Device mountable

Words 3 and 4 are undefined, and word 5 has a special meaning for the DL11-E, DQ11, DP11, and the DU11 interfaces. Byte 0 of word 5 contains the number of sync characters to be transmitted before a synching message (for example, after line turn-around in half-duplex operation), and byte 1 is a sync counter.

12.3 QIO$ MACRO

This section summarizes the standard and device-specific functions of the QIO$ macro that are valid for the communication interfaces described in this chapter.

12.3.1 Standard QIO$ Functions

Table 12-2 lists the standard functions of the QIO$ macro that are valid for the communication devices.

Table 12-2
Standard QIO$ Functions for Communication Interfaces

Format	Function
QIO$C IO.ATT,...	Attach device [1]
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	READ logical block (stripping sync)
QIO$C IO.WLB,...,<stadd,size>	WRITE logical block (preceded by syncs)

1. Only unmounted channels may be attached. An attempt to attach a mounted channel results in an IE.PRI status return in the I/O status doubleword.

stadd

The starting address of the data buffer (may be on a byte boundary).

size

The data buffer size in bytes (must be greater than 0).

12.3.2 Device-Specific QIO$ Functions

The specific functions of the QIO$ macro that are valid for the communication line interfaces are shown in Table 12-3.

Table 12-3
Device-Specific QIO$ Functions for Communication Interfaces

Format	Function
QIO$C IO.FDX	Set device to full-duplex mode; Not applicable to DA11-B
QIO$C IO.HDX,...,<stat,mode>	SET device to half-duplex mode; Not applicable to DA11-B
QIO$C IO.INL,...	Initialize device and set device characteristics
QIO$C IO.RNS,...,<stadd,size>	READ logical block, without stripping sync characters (transparent mode); Not applicable to DQ11; for DA11-B and DMC11, treated like IO.RLB. Not supported on DU11 and DUP11.
QIO$C IO.SYN,...,<syn>	SPECIFY sync character; not applicable to DA11-B or DMC11
QIO$C IO.TRM,...	Terminate communication, disconnecting from physical channel
QIO$C IO.WNS,...,<stadd,size>	WRITE logical block without preceding sync characters (transparent mode); for DA11-B and DMC11, treated like IO.WLB

stadd

The starting address of the data buffer (may be on a byte boundary).

size

The data buffer size in bytes (must be greater than 0).

syn

The sync character, expressed as an octal value.

stat

The station assignment (primary or secondary).

mode

The transmission mode (normal or maintenance).

The device-specific functions listed in Table 12-3 are described in greater detail below.

12.3.2.1 **IO.FDX** - The IO.FDX QIO$ function sets the mode on a DL11-E, DP11, DQ11, DU11, DUP11, or DMC11 unit to full-duplex. The IO.FDX function code can be combined (ORed) with the IO.SYN function code, if desired, to set the operational characteristics of the physical device unit.

12.3.2.2 **IO.HDX** - The IO.HDX QIO$ function sets the mode on a DL11-E, DP11, DQ11, DU11, DUP11, or DMC11 unit to half-duplex. The IO.HDX function code can be combined (ORed together) with the IO.SYN function code, if desired, to set the operational characteristics of the physical device unit.

Setting half-duplex on the DMC11 also involves setting the station assignment (primary/secondary) and may include selecting maintenance mode (MOP) as opposed to normal mode. The station assignment is included in optional QIO$ parameter p1. A 0 indicates primary station and a nonzero indicates secondary station. The DMC11 works properly if both ends are primary stations or if there is one primary and one secondary station. It does not work if both ends are secondary stations. Optional QIO$ parameter p2 selects the mode. A 0 selects normal mode and a nonzero selects MOP mode. A DMC11 in MOP mode cannot communicate with a DMC11 in normal mode.

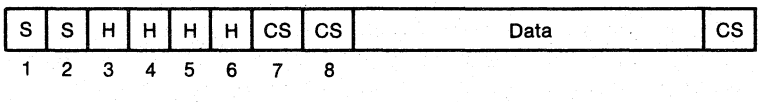
12.3.2.3 **IO.INL and IO.TRM** - These two QIO$ functions have the same function code but different modifier bits.

IO.INL initializes a physical device unit for use as a communications link. It turns the device on line, sets device characteristics, and ensures that the appropriate data terminal is ready.

IO.TRM disconnects the device. If the device has a dial-up interface, it also hangs up the line.

12.3.2.4 **IO.RNS** - The IO.RNS QIO$ function reads a logical block of data, without stripping the sync characters that may precede the data.

IO.RLB is a similar function, which is nontransparent, in that it causes the sync characters that precede the data message to be stripped. Use IO.RLB at the start of a segmented data request, in which the block might have the following layout:



ZK-007-81

S

A sync character.

H

A header character.

CS

A validity check character.

You must strip sync characters from the beginning of a data block in this way. Stripping only at the beginning of a read allows a later character that happens to have the same binary value as a sync character to be read without stripping. Use IO.RLB to read a logical block with leading sync characters stripped; use IO.RNS to read the block without stripping leading sync characters. Because the DA11-B is a parallel device and there are no sync characters, it treats the latter as if it were IO.RLB. Generally, you should use IO.RLB.

12.3.2.5 **IO.SYN** - This QIO$ function allows the programmer to specify the sync character to be recognized when an IO.RLB or IO.WLB function is performed. IO.SYN can be combined (ORed together) with IO.HDX or with IO.FDX to set the characteristics of the physical device unit.

12.3.2.6 **IO.WNS** - This QIO$ function causes a logical block to be written with no preceding sync characters. To ensure that the two systems involved in a communication are synchronized, two or more sync characters are transmitted by one system and received by the other before any other message can be sent.

Use IO.WLB to write a block of data, preceded by sync characters. Generally, you should use IO.WLB.

Use IO.WNS to perform a block transfer without sending sync characters first. Because the DA11-B is a parallel device and there are no sync characters, it treats the latter as if it were IO.WLB.

12.4 STATUS RETURNS

The error and status conditions listed in Table 12-4 are returned by the communication drivers described in this chapter.

Table 12-4
Communication Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO$ directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	**I/O request pending** The operation specified in the QIO$ directive has not yet been executed. The I/O status block is filled with 0s.

(continued on next page)

Table 12-4 (Cont.)
Communication Status Returns

Code	Reason
IE.BCC	**Block check error** When the Cyclic Redundancy Check (CRC) option is present on the DQ11, a check character is appended to each message transmitted. The receiver of the messages recalculates the check character and compares it with the one transmitted. This error code is returned when the two check characters do not match, and represents a transmission error.
IE.CNR	**Connection rejected** (DMC11 only.) The DMC11 has detected that the device on the other end of the line has restarted itself. Your task can recover by issuing IO.INL (initialize), and then reissuing the QIO$ in question.
IE.DAO	**Data overrun** Due to UNIBUS traffic or a modem problem, the DQ11 controller was unable to maintain the data rate required to prevent data loss (that is, the receipt of another byte before processing of a previous byte was completed).
IE.DNR	**Device not ready** The physical device unit specified in the QIO$ directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: • The physical device unit could not be initialized (that is, the circuit could not be completed). • The transmission of a character was not followed by an interrupt within the period of time selected as the device time-out period. This time-out occurs only when a transmission is in progress and the interrupt marking completion of a message does not occur. The appropriate response to this condition is to attempt to resynchronize the device by initializing and accepting the next request. A time-out does not occur on a read. If the receiving device is not ready, the transfer is not initiated by the transmitting device. Once the transfer is initiated, however, it completes its execution either by satisfying the requested byte count or by timing out.

(continued on next page)

Table 12-4 (Cont.)
Communication Status Returns

Code	Reason
IE.IFC	**Illegal function** A function code was specified in an I/O request that is illegal for message-oriented communication devices.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO$ directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternatively, a byte count of 0 was specified.
IE.VER	**Nonrecoverable error (DA11-B only)** The data transfer terminated before all of the data has been transmitted. The error code is returned on transmit when both systems attempt to transmit at the same time. This condition is detected by the device protocol. The error code is returned on receive when the transmit data count of the transmitting side does not equal the data count specified by the receive QIO$.
IE.ABO	**Operation Aborted** The specified I/O operation was canceled by IO.KIL while in progress or while still in the I/O queue.
IE.RSU	**Shareable Resources in use** The task attempted to allocate Unibus Mapping Registers. All UMRs were allocated to other tasks and were unable to complete the transfer.
IE.TMO	**Timeout Error** The physical device unit associated with the LUN specified in the QIO$ directive timed out. This occurs during a data transfer operation when the task does not receive an interrupt within a specified amount of time.

12.5 PROGRAMMING HINTS

This section contains important information about programming the message-oriented communication interfaces described in this chapter.

12.5.1 Transmission Validation

Because there is no way for the transmitting device to verify that the data block has successfully arrived at the receiving device unless the receiver responds, the transmitter assumes that any message that is clocked out on the line (without line or device outage) has been successfully transmitted. As soon as the receiver is able to satisfy a read request, it returns a successful status code (IS.SUC) in the I/O status block. Of course, only the task receiving the message can determine whether the message has actually been transmitted accurately.

The receiving device should be ready to receive data (with a read request) at the time the transmission is sent.

12.5.2 Redundancy Checking

By the nature of message-oriented communications, only the task that receives a communication can determine whether the message was received successfully. The transmitter simply transfers data, without validation of any kind. It is therefore the responsibility of the communicating tasks that use the device to check the accuracy of the transmission. A simple validity check is a checksum-type longitudinal redundancy check. A better approach to validating data is the use of a cyclic redundancy check (CRC). A CRC can be computed in software or with a hardware device, such as the KG-11 communications arithmetic option.

Both DQ11 and DUP11 incorporate hardware to compute a CRC. The DQ11 CRC hardware requires an extra system unit.

12.5.3 Half-Duplex and Full-Duplex Considerations

Because there is a single I/O request queue, only one QIO$ request can be performed at a time. It is therefore not possible, through QIO$s, for a device to send and receive data at the same time. Also, because timeouts are not set for receive functions, a receive QIO$ is terminated only by receiving a message from the remote system, or by issuing an IO.KIL QIO$ for the device. Therefore, if no message is transmitted by the remote system, a receive does not terminate, and no further I/O can be performed on that device until the receive is killed by issuing an IO.KIL QIO$.

You can use both half-duplex and full-duplex lines with the DL11-E, DMC11, DP11, DQ11, DU11, and DUP11. The mode is settable by using IO.FDX for full-duplex and IO.HDX for half-duplex. In half-duplex mode, the modem signal RTS (Request To Send) is cleared after each "transmit message." In full-duplex, this signal is always left on. Using full-duplex mode eliminates modem delays in transmission, but requires full-duplex hardware and communication links.

Only half-duplex mode is available with the DA11-B because of the nature of the hardware.

The DMC11 Driver maintains both transmits and receives separately in its own internal queues. Thus, it is a full-duplex driver. There is no limit on the number of outstanding I/O requests that can be active at any given time. The DMC11 hardware, however, allows a maximum of only seven transmits and seven receives to be active at any time. The driver gives the first seven transmits (or receives) directly to the DMC11 and queues the eighth and subsequent transmits (or receives) internally until the DMC11 acknowledges a successful I/O request. When running on an 11/70, the driver gives only two transmits (or receives) to the DMC11 because each request requires a UNIBUS mapping register. The DMC11 driver is assigned five UMRs: one for base table(s), two for active transmits, and two for active receives.

12.5.4 Low-Traffic Sync Character Considerations

If message traffic on a line is low, each message sent from a communications device should be preceded by a sync train. This enables the controller to resynchronize if a message is "broken" (that is, part or all of it is lost in transmission). Correspondingly, every message received by a communications device under low-traffic conditions, when messages are not contiguous (back-to-back), should be read with an IO.RLB (read, strip sync) function. This requires that the first character in the data message itself not have the binary value of the sync character.

12.5.5 Vertical Parity Support

Vertical parity is not supported by the DA11-B, DL11-E, DP11, DQ11, or DU11. Codes are assumed to be 8-bit only.

12.5.6 Powerfail with DMC11

The DMC11 currently cannot recover after a power failure because the RAM in its internal microprocessor is erased when power fails. Any I/O requests outstanding at the time of a power failure return the IE.ABO status. These requests must be reissued after initializing the DMC11 (IO.INL).

12.5.7 Importance of IO.INL

After the type of communication line has been determined, and after IO.SYN has specified the sync character, it is extremely important that IO.INL be issued before any transfers occur. This ensures that appropriate parameters are initialized and that the interface is properly conditioned. Note that IO.INL provides the only means of setting device characteristics, such as sync character. For this reason, you should always use IO.INL immediately prior to the first transfer over a newly activated link.

Tasks sending messages to the DMC11 should begin by terminating and reinitializing the device (IO.TRM,IO.INL). Note that this causes the error IE.CNR to be returned on any I/O outstanding on the other end of the line. IO.INL must be issued after each IO.KIL (which effectively kills the DMC11), after power-fail, and upon receipt of any error code.

12.6 PROGRAMMING EXAMPLE

The following example illustrates the initialization, setting of device parameters, and transmission of a block of data on a message-oriented communication device.

```
            .MCALL  ALUN$S,QIO$S
            .       .
            .       .
            .       .
            ALUN$S  #1,#"XP,#0                      ; USE LUN1 FOR DP11
            QIO$S   #IO.HDX!IO.SYN,<#1,,,,,#226>    ; SET DEVICE PARAMETERS
            QIO$S   #IO.INL,#1                      ; PUT DEVICE ON LINE
            QIO$S   #IO.WLB,#1,,,<#TXSTS,#TXAST,#TXBUF,#100>; SEND A BLOCK
            .       .
            .       .
            .       .
TXAST:      CMPB    #IS.SUC&377,@(SP)+              ; WAS DATA CLOCKED OUT
                                                    ; SUCCESSFULLY?
                                                    ; IF SO, SET UP FOR NEXT
            BEQ     10$                             ; BLOCK
```

CHAPTER 13

RSX QIO DEUNA DRIVER

13.1 INTRODUCTION

For systems without DECnet, the RSX QIO DEUNA driver allows messages to be sent by using the DEUNA device. The DEUNA driver provides direct control over a line, allowing you to send data over a line to another system. To use the DEUNA driver, you issue the QIO$ macro to the XE: device. The DEUNA driver is compatible with DECnet's Direct Line Access interface (DLX), which permits easy migration to a DECnet system.

Use of the DEUNA driver requires a thorough knowledge of MACRO-11 Assembler and experience in writing real-time application programs. You must write tasks that synchronize with each other before transferring data. If tasks are not synchronized, the data can be lost during task-to-task communication. You must provide your own error-handling routines. The DEUNA driver software informs your task of any errors, but your task must be written to process error recovery. In addition, you must provide your own flow control over incoming messages to avoid message loss. Furthermore, applications must be designed so that adjacent nodes contain like routines for handling communications. For example, the driver does not, by itself, handle communications with DECnet nodes.

You can use QIO$s to communicate between your program and a program on an adjacent computer using the Ethernet. In task-to-task communication between adjacent computers, the RSX QIO DEUNA driver is an efficient user of the CPU and communication lines. You can build your own protocol that best suits the application.

NOTE

All messages are transmitted from your task's buffer. However, the driver buffers messages that it receives in a limited number of driver receive buffers. Therefore, you should make sure that at least two or more receive requests are outstanding at any given time to prevent messages from being lost. Unwanted messages are discarded.

NOTE

A glossary of DEUNA terms is included at the end of this chapter.

13.1.1 Parameters That You Can Tailor

The parameters that you can tailor are as follows:

Parameter	Meaning
U$$NTS	Number of transmit ring entries (suggested 3). On systems with UNIBUS Mapping Registers (UMRs) this parameter controls the number of UMRs the driver may use. For each transmission, the driver uses one UMR during the transfer.
U$$NRS	Number of receive ring entries (suggested 8).
U$$NPC	Number of ports per controller (suggested 8).
U$$NCT	Number of controllers.

13.1.2 Requirements for Tasks Using the RSX QIO DEUNA Driver

To run programs that use the DEUNA driver, the following are required:

- The DEUNA driver must be loaded.
- The LUN must be assigned to the XE: device.

13.1.3 Special Considerations for Ethernet User Tasks

Externally, Ethernet devices appear to be single line point-to-point controllers (for example, UNA-0 and UNA-1). Internally, they are implemented as multipoint devices with each station representing an available port onto the Ethernet. Each driver supports eight ports. The limitation is due to the limited number of receive buffers available to the driver.

13.1.4 Messages on Ethernet

All messages on the Ethernet must include a destination address (48-bit) and a protocol type (16-bit). There are two modes that determine how messages are transmitted: physical address mode and multicast address mode.

Physical address mode defines a unique address for a single system on any Ethernet. Multicast address mode defines a multidestination address of one or more systems on a given Ethernet. With multicast addressing, any number of systems can be assigned a group address, so that they are all able to receive the same data in a single transmission.

Before transmitting and receiving messages, you must define a specific mode. You can do this by using the QIO$ IO.XSC macro, which sets characteristics. (See Section 13.3.2.)

13.1.5 Protocol and Address Pairs on Ethernet

Because the Ethernet allows multiple user tasks to access the physical link simultaneously, some way must be used to deliver received messages to the correct user task. To do this, each user task must enable unique protocol/address pairs to define which messages the task should receive. For example, user task 1 may enable protocol A to addresses 1 and 2, while user task 2 may enable protocol B to addresses 3 and 4. It is possible for two or more user tasks to enable the same protocol or addresses providing that the protocol/address pairs are unique.

13.1.6 Opening Ethernet for Transmit and Receive

The Ethernet may be opened in three different modes (defined in EPMDF$):

Protocol	Mode	Meaning
LF$EXC	Exclusive	Your task has exclusive use of the specific protocol LF$EXC and no other user may transmit or receive using this protocol. (DECnet routing uses this mode.)
LF$DEF	Default	Your task should receive messages on the protocol LF$DEF, which would otherwise be discarded because there was no protocol/address pair set up.
Specified	Normal	You must specify the protocol/address pairs that are used for communications.

13.1.7 Padding Messages on Ethernet

In addition, you may select padding for an Ethernet message (LF$PAD) that prefixes the message with a two-byte length field. The UNA pads the message out to the minimum Ethernet size on transmit. On receive, the length field indicates the amount of data present.

13.1.8 Hardware Errors on Ethernet

When a hardware error is detected on the Ethernet controller, all protocol/address pairings and multicast addresses are lost. After issuing the IO.XIN call to reinitialize the channel, you must reenable all protocol/address pairs and the multicast addresses.

13.2 DEUNA DRIVER QIO$S

Sections 13.2.1 through 13.2.4 describe some considerations for using QIO$ macros for the DEUNA driver.

13.2.1 Standards and Access to QIO$ Macros

The DEUNA driver conforms to normal RSX-11 QIO$ standards. Standards for logical unit numbers (LUNs), event flags, I/O status blocks, asynchronous system traps (ASTs), and argument and parameter lists are observed. According to RSX-11 standards, you may use any one of the three QIO$ formats. You may also use the QIO$ and Wait macro (QIOW$) to suspend further execution of the program until the call completes.

The macros are defined in the RSX macro library (EXEMC.MLB). This library is transferred to your system during system generation. The definitions and offsets that you use in the macros are contained in two definition macros, DLXDF$ and EPMDF$, in DEUNA.MLB.

You must issue .MCALL statements and explicitly invoke the macro in your MACRO-11 assembler program. An example follows:

```
.MCALL DLXDF$,EPMDF$
          .
          .
          .
       DLXDF$
       EPMDF$
```

Table 13-1 summarizes the QIO DEUNA driver function codes and their meaning. Sections 13.3.1 through 13.3.7 describe each call, with its arguments and completion status codes.

13.2.2 Programming Sequence

The following table is a list of the five steps required to transmit, receive, or read data on the Ethernet via the RSX QIO DEUNA driver.

Table 13-1
RSX QIO DEUNA Driver Function Codes and Their Meaning

Step	Code	Ethernet Operation
1	IO.XOP	Open the Ethernet device
2	IO.XSC	Set characteristics
3	IO.XTM	Transmit a message
4	IO.XRC	Receive a message
5	IO.XCL	Close the line
6	IO.XIN	Initialize the line after an unrecoverable hardware error.

NOTE

The IO.XTL control function loads DEUNA microcode. The driver support task, UML..., uses IO.XTL, a function you must not use.

13.2.3 Driver Installation

The system builds the driver at the time you perform a system generation.

To load the driver, enter the following command in MCR:

```
MCR>LOA XE:[/switches]
```

For RSX-11M-PLUS, you must perform the following additional steps to make the driver operational:

```
MCR>CON SET XEA VEC=vvv CSR=xxxxxx
MCR>CON ONLINE XEA
MCR>CON ONLINE XE0:
MCR>INS UML
```

NOTE

UML... is the microcode loader support task to the DEUNA driver (XEDRV). If you want the driver to bypass microcode loading, just remove the microcode support task (UML) from the system (RSX-11M-PLUS only). The microcode task must be present in RSX-11S and RSX-11M systems.

Make sure that the correct microcode file for the DEUNA is present on device LB: in account [1,1].

On RSX-11S systems the microcode ECO file must be installed in the UNAMC partition by VMR. The UML... task looks for the microcode in this partition.

13.2.4 RSX QIO DEUNA Status Returns

Table 13-2 lists the status returns from QIO$ macros issued to the DEUNA driver.

Table 13-2
RSX QIO DEUNA Driver Status Returns

Code	Value		Reason
	Decimal	Octal	
IS.SUC	1.		The line has been opened successfully.
IE.ALN	-34.	177736	The specified lun is already in use.
IE.IFC	-2.	177776	The specified lun is not assigned to XE:. for those characteristics blocks processed, return (XEDRV)
IE.NSF	-26.	177646	Either you have entered an invalid controller identification format or the specified controller is not in the system.

13.3 QIO$ MACROS

This section summarizes standard and device-specific QIO$ functions for the RSX QIO DEUNA Driver.

13.3.1 IO.XOP - Open a Line

You issue the QIO$ IO.XOP macro to open a line for direct line access, message transfer, and reception. The IO.XOP functions associates the specified lun with the specified line. The line is then used when you issue further QIO$s for transmitting or receiving. The lun must have been assigned to XE:. To open the Ethernet device from the DEUNA driver, you issue this call using a device-id string such as "UNA-0". The address of this string should be in p1. The driver scans its port database for an available port and assigns it to your task.

The QIO$ syntax is as follows:

```
QIO$ IO.XOP,lun,[efn],,[status],[ast],<p1,p2,p3>
```

The parameters in the QIO$ IO.XOP macro are:

Parameters	Meaning
lun	Logical unit number associated with the line that you are opening.
efn	Optional event flag number set when the call completes.
status	Address of an optional two-word status block that contains the completion status of the call in the low-order byte of the first word.
ast	Entry point into an optional AST routine, which you wrote, to be executed after this call completes.
p1	Address of an ASCII string that identifies the controller on which line is to be opened. The syntax of this string is: DEV-ctl where DEV (UNA) is the device mnemonic and ctl is the decimal value for the controller number.
p2	Length of the line identification field.
p3	Time-out value for the call. The time-out value is the amount of time that the receiver waits for a message to be transmitted. The low-order byte of the word designates the receive time-out value as follows: time-out = 0 for no receive timer. time-out = <n> where n is the timer value in seconds. The timer value n causes the timeout to have a range of n-1 to n.) The high-order byte of this word is ignored.

13.3.2 IO.XSC - Set Characteristics (Ethernet)

You use this Ethernet QIO$ to set up the protocol/address pairs and multicast addresses. This function supplies a single characteristics buffer in arguments p1 and p2. This buffer may contain multiple characteristics blocks of the general format given in Section 13.3.2.1.

The QIO$ syntax is as follows:

```
QIO$ IO.XSC,lun,[efn],,[status],[ast],<p1,p2>
```

The parameters of the QIO$ IO.XSC macro are:

Parameters	Meaning
lun	Logical unit number associated with the line that you are setting for a characteristics buffer.
efn	Optional event flag number set when the call completes.
status	Quantity processed on completion. The second word of the I/O status block indicates how much of the characteristics buffer has been processed.
ast	Entry point into an optional AST routine, which you wrote, to be executed after this call completes.
p1	Address of the characteristics buffer.
p2	Length of the characteristics buffer.

13.3.2.1 **The Set Characteristics Buffer; General Format** - The set characteristics buffer format may contain multiple characteristics blocks. Each characteristics block has the general format shown in Figure 13-1:

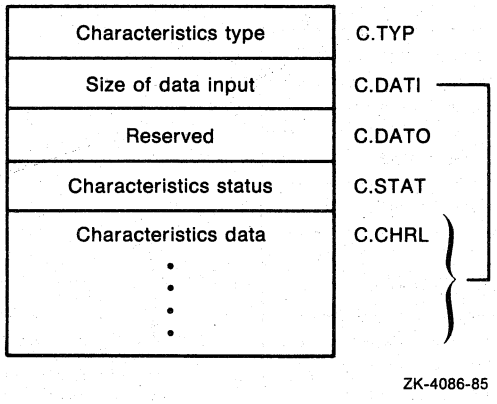


Figure 13-1 General Form of Characteristics Buffer

The fields in the general form of the characteristics block have the following meanings:

Field	Meaning
C.TYP	Indicates the type of characteristics being set.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set to indicate the success or failure of the characteristics function, for those characteristics blocks processed.

Protocol flags are defined in EPMDF$ (LF$xxx).

Common error codes that are returned in C.STAT are:

Error	Meaning
CE.UDF	Undefined function.
CE.RTS	Request too small (not enough data supplied).
CE.RTL	Request too large (too much data supplied).
CE.RES	Resource allocation failure.

NOTE

The address field(s) should not be present if LF$EXC or LF$DEF is specified in the flags.

13.3.2.2 **Set Characteristics -- Setting Up Protocol/Address Pairs** - Setting up protocol/address pairs allows transmission and reception of messages with the specified protocol to or from any of the addresses in the list. Figure 13-2 shows the characteristics buffer for this operation.

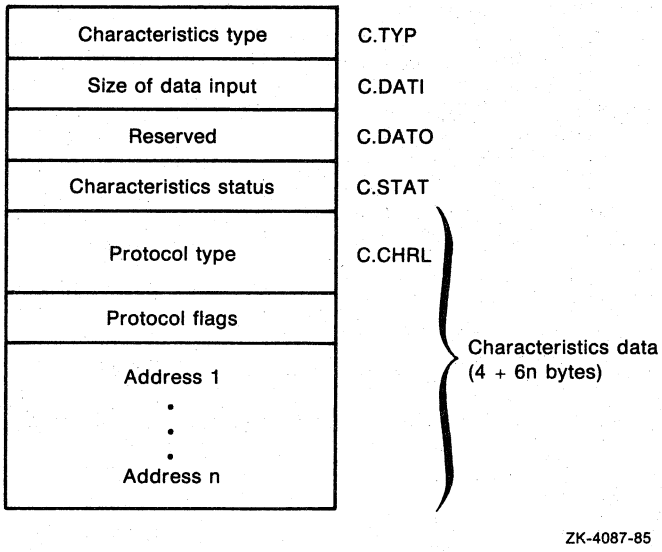


Figure 13-2 Buffer for Setting Up Protocol/Address Pairs

The fields in the characteristics buffer for setting up protocol/address pairs have the following meaning:

Field	Meaning
C.TYP	Contains the characteristics type to set up protocol/address pairs: CC.DST = 200.
C.DATI	Indicates the size of data input -- the number of bytes of characteristic data being supplied.
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed, to indicate the success or failure of the characteristics function.

Errors that are returned in C.STAT are shown in the following list:

Error	Meaning
CE.PCN	Protocol usage conflict: • Another user task has exclusive access to this protocol. • There is already a default task using this protocol, and this request is attempting to set up a new default user task. • The padding status of this protocol does not match what is requested.
CE.IUM	Invalid use of multicast address; one of the addresses specified is multicast.
CE.ACN	Address usage conflicts; the protocol/address pair is already in use.

13.3.2.3 **Set Characteristics -- Setting Up a Multicast Address** - Setting up a multicast address allows reception of messages that are sent to the specified multicast address. The buffer for setting up a multicast address is shown in Figure 13-3.

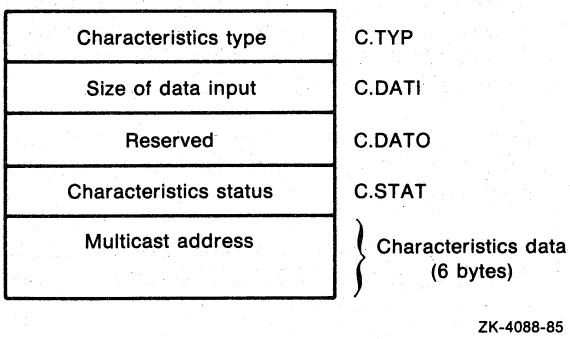


Figure 13-3 Buffer for Setting Up a Multicast Address

The fields in the characteristics buffer for setting up a multicast address have the following meaning:

Field	Meaning
C.TYP	Contains the characteristics type to set up a multicast address: CC.MCT = 201.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set for those characteristics blocks processed to indicate the success or failure of the characteristics function.

Errors returned in C.STAT are:

Error	Meaning
CE.NMA	Not a multicast address.
CE.MCE	Multicast address already enabled.

13.3.3 IO.XIN - Initialize the Line

You issue the QIO$ IO.XIN macro to reinitialize a line after a fatal device error has occurred. When you use this QIO$, you must reset the mode and timer values. The syntax of the QIO$ IO.XIN macro is as follows:

```
QIO$ IO.XIN,lun,[efn],,[status],[ast],<pl>
```

The parameters of the QIO$ IO.XIN macro are:

Parameters	Meaning
lun	Logical unit number associated with the line that you are initializing.
efn	Optional event flag number set when the call completes.
status	Address of an optional two-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 13.3.3.1).
ast	Entry point into an optional AST routine, which you wrote, to be executed after this call completes.
pl	Timer parameter. Use the same format as described for parameter p3 in IO.XOP. (See Section 13.3.1.)

13.3.3.1 **Completion Status Codes for IO.XIN** - IO.XIN returns completion status codes as follows:

Code	Value		Reason
	Decimal	Octal	
IS.SUC	1		The line has been successfully initialized.
IE.ABO	-15.	177761	The initialization attempt has been aborted. A hardware device error or an attempt to initialize a line that did not require it could cause this problem.
IE.IFC	-2.	177776	The specified lun is not assigned to XE:.
IE.NLN	-37.	177733	No line has been opened with the specified lun.

13.3.4 IO.XTM - Transmit a Message on the Line

When your task transmits a message on the Ethernet, it must specify the destination address or the multicast address of this message along with the protocol type. It does specify the address if you put the parameters for the optional auxiliary characteristics buffer in parameters p3 and p4 as shown in Figure 13-4. The syntax of the IO.XTM macro is as follows:

```
QIO$ IO.XTM,lun,[efn],,[status],[ast],<p1,p2,p3,p4,[p5,p6]>
```

The parameters of the QIO$ IO.XTM macro are:

Parameters	Meaning
lun	Logical unit number for the line on which you are transmitting data.
efn	Optional event flag number set when the call completes.
status	Address of an optional two-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 13.3.4.3).
ast	Entry point into an optional AST routine, which you wrote, to be executed after this call completes.
p1	Address of the buffer in your task that contains the message to be transmitted. Use the label specified in the DLXBUF macro call.
p2	Length of the message you are sending to the remote computer. Maximum buffer size is 1498.
p3	Address of the auxiliary characteristics buffer destination addresses.
p4	Length of the auxiliary characteristics buffer.

Parameters	Meaning
p5	Diagnostic buffer (see Section 13.4).
p6	Diagnostic buffer size (see Section 13.4).

13.3.4.1 **Auxiliary Buffer to Set the Destination Address** - To transmit on a line, you must first set up the auxiliary characteristics buffer with the Ethernet address. The auxiliary characteristics buffer has the same format as the set characteristics buffer described in Section 13.3.2.1. The buffer is shown in Figure 13-4.

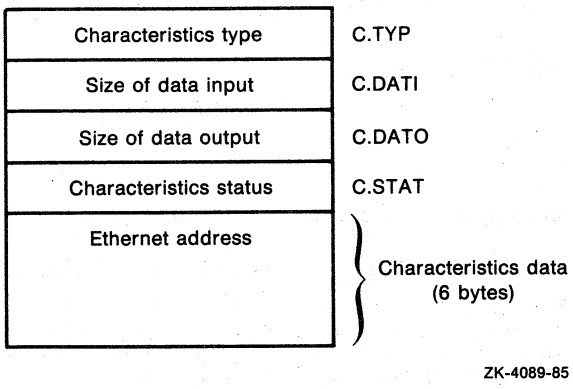


Figure 13-4 Buffer for Setting the Ethernet Address

The fields in the auxiliary characteristics buffer for setting the Ethernet address have the following meaning:

Field	Meaning
C.TYP	Contains the characteristics type to set the Ethernet address: CC.ADR = 100.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed to indicate the success or failure of the characteristics function.

13.3.4.2 **Auxiliary Buffer to Set the Protocol Type** - The protocol type must be transmitted along with the message. Use the auxiliary buffer shown in Figure 13-5 for this purpose.

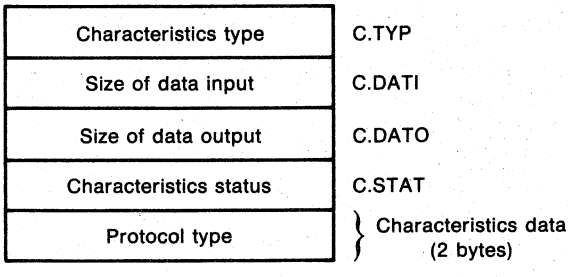


Figure 13-5 Buffer for Setting the Protocol Type

The fields in the auxiliary characteristics buffer for setting the protocol type are as follows:

Field	Meaning
C.TYP	Contains the characteristics type to set the protocol type: CC.PRO = 101.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed to indicate the success or failure of the characteristics function.

Transmit requests on Ethernet channels must include an auxiliary characteristics buffer including both the destination address and protocol type. Failure to do so causes the transmit message to be returned with an IE.BAD error.

13.3.4.3 **Completion Status Codes for IO.XTM** - QIO$ IO.XTM returns the following completion status codes:

Code	Value		Reason
	Decimal	Octal	
IS.SUC	1.		The message was transmitted to the remote system successfully.
IE.ABO	-15.	177761	The transmission was aborted because an unrecoverable error occurred in the hardware device. When a message transmission completes with an IE.ABO code, the line is hung up. You must either issue a QIO$ IO.XIN to initialize the line (see Section 13.3.3) or close and reopen the line (see Sections 13.3.6 and 13.3.1, respectively) before you can use it again.
IE.IFC	-2.	177776	The lun is not assigned to XE.
IE.NLN	-37.	177733	No line has been opened with the specified lun.
IE.SPC	-6.	177772	The transmit buffer is too large or too small.

13.3.5 IO.XRC - Receive a Message on the Line

The QIO$ IO.XRC function receives a message on the Ethernet. When you receive a message on the Ethernet, you must find out the source address for this message along with the protocol type. You can do this by having an optional auxiliary characteristics buffer for receive messages in parameters p3 and p4. The syntax of the QIO$ IO.XRC macro is as follows:

```
QIO$ IO.XRC,lun,[efn],,[status],[ast],<p1,p2,p3,p4,[p5,p6]>
```

The parameters of the QIO$ IO.XRC function are as follows:

Parameters	Meaning
lun	Logical unit number associated with the line on which you receive the message.
efn	Optional event flag number set when the call completes.
status	Address of an optional two-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 13.3.5.4).
ast	Entry point into an optional AST routine, which you wrote, to be executed after this call completes.
p1	Address of the buffer in your task that receives the message.
p2	Length in bytes that you are allocating for the receive buffer. Maximum buffer size is 1498.
p3	Address of the auxiliary characteristics buffer.
p4	Length of the auxiliary characteristics buffer.
p5	Diagnostic buffer (see Section 13.4).
p6	Diagnostic buffer size (see Section 13.4).

13.3.5.1 **Buffer For Reading the Ethernet Address** - The auxiliary characteristics buffer has the same format as the set characteristics buffer described in Section 13.3.2.1. The buffer to provide for reading the Ethernet characteristics is shown in Figure 13-6.

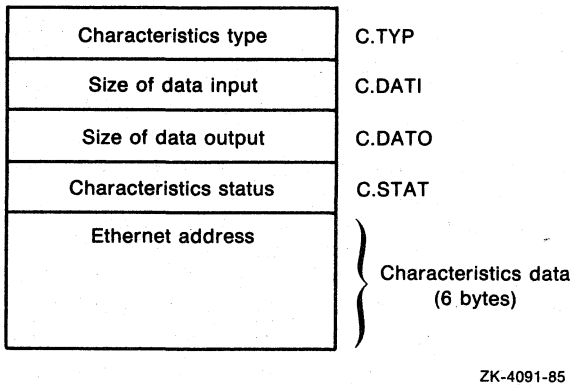


Figure 13-6 Buffer for Reading the Ethernet Address

The fields in the auxiliary characteristics buffer for reading the Ethernet address are as follows:

Field	Meaning
C.TYP	Contains the characteristics type to read the Ethernet address: CC.ADR = 100.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed, to indicate the success or failure of the characteristics function.

13.3.5.2 Buffer for Reading the Protocol Type - The buffer for reading the protocol type is shown in Figure 13-7 as follows:

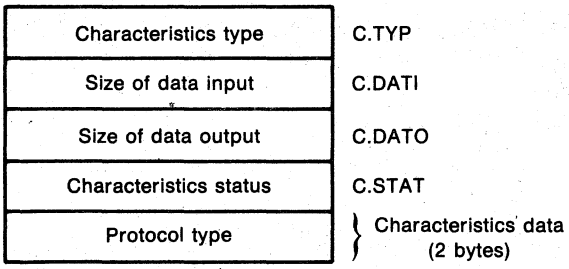


Figure 13-7 Buffer for Reading the Protocol Type

The fields in the auxiliary characteristics buffer for reading the protocol type are as follows:

Field	Meaning
C.TYP	Contains the characteristics type to read the protocol type: CC.PRO = 101.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed, to indicate the success or failure of the characteristics function.

13.3.5.3 **Buffer for Reading the Destination Ethernet Address** - The buffer for reading the destination Ethernet address is shown in Figure 13-8 as follows:

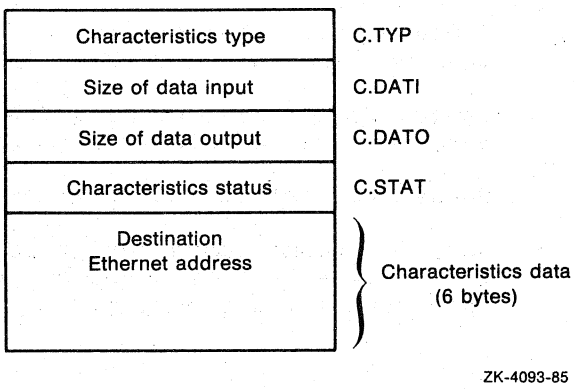


Figure 13-8 Buffer for Reading the Destination Ethernet Address

The fields in the auxiliary characteristics buffer for reading the destination Ethernet address are as follows:

Field	Meaning
C.TYP	Contains the characteristics type to read the destination Ethernet address: CC.ADR = 102.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set for those characteristics blocks processed to indicate the success or failure of the characteristics function.

13.3.5.4 **Completion Status Codes for IO.XRC** - QIO$ IO.XRC returns the following completion status codes:

Code	Value		Reason
	Decimal	Octal	
IS.SUC	1.		You successfully received a message from the remote system. The second word of the I/O status block contains the number of bytes you actually received.
IE.ABO	-15.	177761	The receive function was aborted because an unrecoverable error occurred in the hardware device. When a receive is aborted, the line is hung up. You must either issue QIO$ IO.XIN to initialize the line (see Section 13.3.3) or close and reopen the line (see Sections 13.3.6 and 13.3.1, respectively) before you can use it again.

Code	Value		Reason
	Decimal	Octal	
IE.DAO	-13.	177763	Either a message was received before a receive QIO$ was issued and the data is lost (this applies only to normal mode operations), or your task's buffer was too small to receive all the data. In the latter case, the message is truncated, and some data is lost. (The length of your task's buffer is contained in the second word of the I/O status block.)
IE.IFC	-2.	177776	The specified lun is not assigned to XE:.
IE.NLN	-37.	177733	No line has been opened with the specified lun.
IE.TMO	-74.	177666	A time-out condition has occurred. No message was received within the timer interval specified when you opened or initialized the line.
IE.SPC	-6.	177772	The transmit buffer is too large or too small.

13.3.6 IO.XCL - Close the Line

You issue the QIO$ IO.XCL macro to close an open line and stop the protocol. The syntax of the QIO$ IO.XCL macro is as follows:

```
QIO$ IO.XCL,lun,[efn],,[status],[ast]
```

Parameters	Meaning
lun	Logical unit number associated with the line that you are closing.
efn	Optional event flag number set when the call completes.
status	Address of an optional two-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 13.3.6.1).
ast	Entry point into an optional AST routine to be executed after this call completes.

13.3.6.1 **Completion Status Codes for IO.XCL** - QIO$ IO.XCL returns completion status codes as follows:

Code	Value Decimal	Octal	Reason
IS.SUC	1.		The line has been successfully closed.
IE.IFC	-2.	177776	The speicifed lun is not assigned to XE:.
IE.NLN	-37.	177733	No line has been opened with the specified lun.

13.3.7 IO.XTL - Control Function

The QIO$ IO.XTL macro loads the ECO microcode. IO.XTL is only valid when the driver is initializing the DEUNA controller. This function is a privileged function. Using it does not require a line to be open on the DEUNA. The syntax of the QIO$ IO.XTL macro is as follows:

```
QIO$ IO.XTL+subfunction,lun,[efn],,[status],[ast]
```

Subfunctions	Meaning
sub=0	Load microcode to DEUNA memory.
sub=1	End of load.
sub=2	Abort load.

Parameters	Meaning
lun	Logical unit number.
efn	Optional event flag number set when the call completes.
status	Address of an optional two-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 13.3.7.1). The second word is the count of the number of bytes loaded.
ast	Entry point into an optional AST routine, which you wrote, to be executed after this call completes.

13.3.7.1 **Completion Status Codes for IO.XTL** - IO.XTL returns the following completion status codes:

Code	Value Decimal	Octal	Reason
IS.SUC	1.		The load function was successful.
IE.IFC	-2.	177776	Invalid function.

Code	Value		Reason
	Decimal	Octal	
IE.ABO	-15.	177761	A hardware device error or an invalid buffer format could cause this error.
IE.SPC	-6.	177772	The microcode ECO buffer is too large.
IE.PRI	-16.	177760	Privilege violation.

13.4 DIAGNOSTIC FUNCTIONS FOR IO.XTM/IO.XRC

Your task may execute a number of the port control block functions of the DEUNA driver by using parameters p5 and p6. Parameter p5 is the address of the diagnostic buffer and parameter p6 is the size of the diagnostic buffer.

This buffer provides diagnostic hooks in the DEUNA driver. However, some of this buffer is needed for changing the DEUNA physical address, system ID, and so on.

Diagnostic function requests are passed to the driver in the same buffer format as the characteristics functions (see Section 13.3.2).

The diagnostic buffer format may contain multiple function request blocks. Each diagnostic request block is shown in Figure 13-9 as follows:

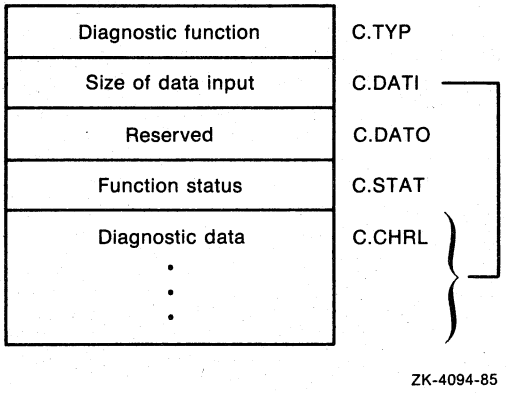


Figure 13-9 Diagnostic Request Block

NOTE

The status returned for the call does not reflect the status of the diagnostic functions in the diagnostic buffer. You must test C.STAT word of each function request specified in the optional diagnostic buffer.

The valid function codes are noted in Table 13-3 as follows:

Table 13-3
Diagnostic Functions for IO.XTM/IO.XRC

Function Code		Meaning	Octal Buffer Size in Words
0		NOP Function	0
2		Read Default Physical Address	3
4		Read Physical Address	3
5		Write Physical Address	3
6		Read Multicast List From UNA	36
12	1	Read Counters	100
13	1	Read and Clear Counters	100
14		Read UNA Mode	1
15		Write UNA Mode	1
16		Read Line Status	10
17	1	Read and Clear Line Status	10
22		Read System ID	144 Max
23	1	Write System ID	144 Max
24		Read Load Server Address	3
25		Write Load Server Address	3

1. These function codes must be issued by a privileged task.

NOTE

The buffer sizes specified are in addition to the four-word header, that is, the function, input size, output size, function status, and data buffer.

13.5 PROGRAMMING HINTS

This section contains information on important programming considerations for tasks using the DEUNA driver described in this chapter.

13.5.1 Information on the DEUNA Device

You should become familiar with the information contained in the DEUNA User's Guide (Order No. EK-DEUNA-UG-001), especially Chapter 4 entitled Programming.

13.5.2 DEUNA Read/Write Mode Function

To change the DEUNA mode, you should read the mode first and combine the change with the current mode by a logical OR function to prevent changing the other mode bits.

You cannot change the Transmit Message Pad Enable bit. The driver relies on the UNA to pad short messages. The driver reenables this bit each time it performs a Write Mode function.

13.5.3 DLX Incompatibility

The RSX DEUNA driver is not 100% compatible with DECnet's Direct Line Access (DLX). Under DECnet's DLX, the system ID, physical address, and mode are set via Network Management. Therefore, you must set these three using the diagnostic functions (see Section 13.4).

13.5.4 Asynchronous I/O

The order of request completion is not preserved by the driver, because the driver has no way of knowing when a receive can be expected. Also, if you use diagnostic functions for a transmit or receive, those without diagnostic functions may complete out of order. Therefore, you should use event flags to identify the request being completed.

13.5.5 Diagnostic Functions Without Data Transfer

To do diagnostic functions without data transfer, specify all the parameters correctly except for the size of the auxiliary characteristics buffer, which must be set to zero. This is an invalid buffer size and returns IE.SPC status for the call. However, the driver processes the diagnostic buffer, if present.

13.5.6 Maximum and Minimum Buffer Size

The maximum buffer size the DEUNA permits is 1500 decimal bytes. However, to provide the padding option described in Section 13.1.7, the maximum buffer size is two bytes less then the 1500 permitted by the DEUNA. The extra two bytes account for the byte count word in the transfer.

The minimum buffer size is 64 bytes. However, the driver does not check for a buffer size of less than 64; it assumes that the DEUNA always operates in padded mode. A small transmit buffer could result in transmitting 20 bytes and receiving 64 bytes, because the DEUNA pads the buffer with zeros out to 64 bytes. Thus, the first 20 bytes will be data and the rest will be null bytes.

13.5.7 Default MODE

The driver initializes the DEUNA with the following mode bits set:

- DEUNA pads short transmit messages.
- H4000 collision test is enabled.

NOTE

The "Enable Half Duplex" mode bit should be set where it is not desirable for the DEUNA to receive its own transmissions.

13.5.8 Example of Connecting to a Remote Task

The following is a list of steps for a task to take to establish connection with a remote task using the RSX QIO DEUNA driver:

- Open a line on the Ethernet device.

- Set characteristics as specified in Section 13.3.2. Setting characteristics establishes the protocol address pairs that the system uses when it communicates with the remote systems on the network. For example, if your task communicates with multicast address 101,252,38 and DEUNA address 304,404,100 using protocol 10000, the characteristics buffer would look like this:

```
.WORD   201       ; C.TYP   - Set multicast address (CC.MCT)
.WORD   6         ; C.DATI  - 6 bytes of address in buffer
.WORD   0         ; C.DATO  - Output data size 0 (none)
.WORD   0         ; C.STAT  - Characteristics status
.WORD   101       ; C.CHRL  - 1st word of multicast address
                              buffer
.WORD   252       ;         - 2nd word of multicast address
                              buffer
.WORD   38        ;         - 3rd word of multicast address
                              buffer
.WORD   200       ; C.TYP   - Set protocol address pair
.WORD   10.       ; C.DATI  - 10. bytes of characteristics data
.WORD   0         ;         - Output data size 0 (none)
.WORD   0         ; C.STAT  - Characteristics status
.WORD   10000     ; C.CHRL  - Protocol type
.WORD   0         ;         - Protocol flags (normal)
.BYTE   304,0     ;         - 1st word of address
.BYTE   1,1       ;         - 2nd word of address
.BYTE   100,0     ;         - 3rd word of address
```

- Once the protocol/address pairs are set, you should issue two or more receive QIO$s in anticipation of receiving a message on the Ethernet. In this way you can ensure that one request may be completing and still have another request outstanding to the driver. Upon completion of a receive request, your task must immediately issue another request before any other action to prevent received messages from being lost.

NOTE

The driver discards unsolicited messages.

- At this point, you may want to transmit a message to the participating systems, letting them know your presence on the network. The format of such a message exchange is application-dependent. Some sort of acknowledgment of the start-up message may complete the start-up sequence.

- Now, you are ready to transmit and receive messages. If you receive an abort notification (IE.ABO) for a request then, the line must be reinitialized via the QIO$ IO.XIN function before further activity can be resumed. Another way to reinitialize the line is to close it and reopen it. The auxiliary characteristics buffer for receives should have room for the address/protocol pair of the originating system:

```
.WORD   100       ; C.TYP   - Read Ethernet address (CC.ADR)
.WORD   6         ; C.DATI  - 6 bytes of address in buffer
.WORD   0         ; C.DATO  - Output data size (6)
```

```
.WORD   0       ; C.STAT  - Characteristics status
.BYTE   0,0     ; C.CHRL  - Driver returns a
.BYTE   0,0     ;           3-word address
.BYTE   0,0     ;           in these three words
.WORD   101     ; C.TYP   - Read protocol type (CC.PRO)
.WORD   2       ; C.DATI  - 2 bytes of protocol type
.WORD   0       ; C.DATO  - Output data size (2)
.WORD   0       ; C.STAT  - Characteristics status
.WORD   0       ; C.CHRL  - Contains protocol type
```

- The auxiliary characteristics buffer for transmits must contain the destination address/protocol pair:

```
.WORD   100     ; C.TYP   - Set Ethernet address (CC.ADR)
.WORD   6       ; C.DATI  - 6 bytes of address in buffer
.WORD   0       ; C.DATO  - Output data size 0
.WORD   0       ; C.STAT  - Characteristics status
.BYTE   304,0   ; C.CHRL  - Task must pass
.BYTE   1,1     ;           3-word address
.BYTE   100,0   ;           in these three words
.WORD   101     ; C.TYP   - Set protocol type (CC.PRO)
.WORD   2       ; C.DATI  - 2 bytes of protocol type
.WORD   0       ; C.DATO  - Output data size 0
.WORD   0       ; C.STAT  - Characteristics status
.WORD   10000   ; C.CHRL  - Must contain protocol type
```

- Upon completion of Ethernet I/O, issue a close (IO.XCL) to release the line.

13.6 GLOSSARY

CONTROLLER -- A single piece of peripheral equipment of the system bus that communicates with one or more external devices. A single DEUNA is a controller.

CSR -- The control and status registers for a controller. These are the ports through which the driver communicates with the device.

DEUNA -- DIGITAL Equipment UNIBUS Network Adapter.

DLX -- Direct Line Access Controller. Enables programs to have a direct, high-level interface to a physical line on systems with DECnet support.

DNA -- The DIGITAL Network Architecture. A network architecture of protocols, interfaces, and functions that enable DECnet network nodes to communicate.

LINE -- A communication path to another system. For example, a port on the NI is a line.

MULTI-ACCESS CHANNEL -- The Ethernet is unlike other data links supported by DIGITAL's communications software products in that more than one user task may use a single circuit simultaneously.

NI -- Network Interconnect is the group of DECnet products that implement the XEROX, INTEL, and DEC intercompany Ethernet Specifications.

PROTOCOL TYPE -- A unique 16-bit address that distinguishes each user task of the NI.

CHAPTER 14

PCL11 PARALLEL COMMUNICATIONS LINK DRIVERS

14.1 INTRODUCTION

PCL11 Parallel Communications Link hardware is supported on RSX-11M/M-PLUS systems by two drivers. One driver supports the transmitter function and the other driver supports the receiver function. The PCL11-B is a hardware interface that functions as a time division multiplexed (TDM) interface over which several PDP-11 computers can transfer data to each other. Each PCL11-B consists of a transmitter, receiver, and master section. The transmitter section can transfer parallel 16-bit words along the TDM bus to a receiver section of a separate PCL11-B on a different PDP-11 computer's UNIBUS. One of the PCL11-B units attached to the TDM bus must have its master section enabled to effect the data transfer.

14.1.1 PCL11-B Hardware

Each PCL11-B transmitter and receiver section has a unique TDM bus address (hardware-configured). When a master section is enabled, it places a transmitter address on the TDM bus for a period of time, called a timeslice. During the timeslice, the addressed transmitter can address the desired receiver section and transmit one word; the transmitter waits for the receiver to acknowledge the word or an indication that the word was not accepted. If the word is not accepted, it normally retransmits the word on the next available timeslice. Thus, a message up to 32k words long can be transmitted to a receiver one word at a time during the time in which other similar TDM transactions are multiplexed for other PCL11-B devices.

14.1.2 PCL11 Transmitter Driver

The PCL11 transmitter driver provides two basic functions. First, it must receive data sent by the attached task and store it in a silo buffer in the PCL11 hardware. Then, the driver passes proper receiver address and command information to the PCL11 transmitter hardware to effect the actual transfer over the TDM bus.

14.1.3 PCL11 Receiver Driver

The PCL11 receiver driver also performs two basic functions. First, it must remove data from the receiver silo and send it to the connected task. In addition, the receiver driver must acknowledge a transmitter when a data transmission is requested by that transmitter. Subsequent requests by other transmitters on the TDM bus are ignored until all message transactions with the current transmitter are completed.

14.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for the PCL11 transmitter and receiver drivers. A setting of 1 indicates that the described characteristics is true for PCL11 transmitter and receiver drivers.

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	1	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Word 3 contains device driver-specific information, as follows:

Transmitter driver:

The low byte of word 3 contains the number of transmit retries remaining after completing the current data transmit function if the current data transmit function attempt is not accepted by the addressed receiver. The high byte of word 3 is undefined.

Receiver driver:

The low byte of word 3 contains the index of the current state of the receiver driver. Use these states primarily for diagnostic purposes as they are defined next:

Index Value	Meaning
0	No task is connected.
+2	Task connected but not triggered.
+4	Task triggered and waiting for IO.RTF or IO.ATF function.

Index Value	Meaning
+6	Task triggered and timed out while waiting for IO.RTF or IO.ATF function.
-2	IO.ATF function is in progress.
-4	Task connected, not triggered, and has an IO.ATF function in progress.
-6	An IO.RTF function is in progress.

The high byte of word 3 is undefined. Word 4 is undefined. Word 5 is the default buffer size in bytes. For the PCL11, this value is 64 bytes.

14.3 QIO MACRO -- PCL11 TRANSMITTER DRIVER FUNCTIONS

14.3.1 Standard QIO Functions

Table 14-1 lists the standard functions of the QIO macro that are valid for the PCL11 transmitter driver.

Table 14-1
Standard QIO Functions for PCL11 Transmitters

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O request

14.3.2 Device-Specific QIO Functions

Table 14-2 lists the device-specific functions of the QIO macro that are valid for the PCL11 transmitter driver.

Table 14-2
Device-Specific QIO Functions for PCL11 Transmitters

Format	Function
QIO$C IO.ATX,...,<stadd,size, flagwd,id,retries,retadd>	Attempt message transmission
QIO$C IO.SEC,...,	Sense master section status
QIO$C IO.STC,...,<stadd,size, [state],[mode],,retadd>	Set master section characteristics

stadd

The starting address of a data buffer. (Its description and function is dependent upon the specific QIO function.)

size

The data buffer size in bytes. (Its description and function is dependent upon the specific QIO function.)

flagwd

The value of the flagword that is to precede the message being sent. The flags specify the desired receiver function as defined by your task's protocol.

id

The identifier of the CPU to which the message is to be sent. This identifier is the desired receiver's TDM bus address. It appears in the high byte of the first word of the master section I/O status block. The identifier number is an octal value contained in the high byte of the parameter word. For example, receiver number 1 is specified as 400, receiver number 2 is specified as 1000, and so forth.

retries

The number of retries that are attempted, following the first attempt, before returning error status to the calling task. Retries occur because of these conditions:

- The first attempt is unsuccessful
- Transmission errors occurred
- A master down condition occurred

retadd

The starting address of a 7-word buffer into which the contents of the six transmitter registers and the transmitter master/maintenance register are moved prior to returning to the calling task. Information describing the contents of these registers can be obtained by referring to the hardware documentation supplied with the PCL11 option.

state

The desired state setting for the transmitter, as follows:

Parameter Specified	State
SS.MAS	TDM bus master
SS.NEU	Neutral (default state)

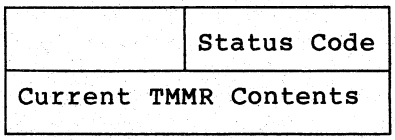
mode

The desired mode setting for allocating transmitter timeslices on the TDM bus, as follows:

Parameter Entered	Mode
MS.AUT	Auto addressing (default mode)
MS.ADS	Address silo

14.3.2.1 **IO.ATX** - This I/O function requests an attempt to transmit a message to a specified CPU. The message to be transmitted is contained in a data buffer starting at the address specified in the stadd parameter. This address must be on a word boundary. The data buffer size specified in the size parameter must be an even, positive value. The flagword parameter contains information, which you defined, that the receiving task uses to determine whether to accept or reject the message. The id parameter is the receiver TDM bus address. The task uses this address to direct a message to a specific CPU. Other parameters are as previously described.

14.3.2.2 **IO.SEC** - This I/O function senses the master section status. Upon successful completion of this function, the I/O status block contains a typical I/O status code (IS.SUC) return in the low byte of the first word, and current Transmitter Master/Maintenance Register (TMMR) contents in the second word, as follows:



NOTE

The optional isb parameter (see Section 1.5.1) must be included in this QIO request.

14.3.2.3 **IO.STC** - IO.STC sets the master section operational characteristics. IO.STC can only be issued by a privileged task. Correct use of the function depends upon the current (or specified) operating state of the master section and proper use of parameters. Use each parameter as described in the following paragraphs. Refer to all parameters in the sequence shown for a correct interpretation of parameter usage.

State -- The state parameter determines the overall function of this master section (and transmitter and receiver sections) in the PCL11 communications link as it relates to the TDM bus. The neutral state (SS.NEU) places the master section in an inactive state where the unit sends and receives messages in a normal manner, but the master section cannot control transmitter timeslice allocation on the TDM bus. The master state (SS.MAS) designates this unit as TDM bus master, enabling control of transmitter unit timeslice allotments on the TDM bus; only one master section on the TDM bus can be designated TDM bus master.

Mode -- The TDM bus master can allocate transmitter timeslices in one of two ways: auto address mode (MS.AUT) or address silo mode (MS.ADS). When operating in the auto address mode (MS.AUT), which is the default mode for the TDM bus master, equal timeslice allotments are given to each transmitter unit; transmitter unit addresses are sequentially put on the TDM bus in descending order, one address for each timeslice. When operating in the address silo mode, transmitter unit addresses are transmitted in a sequence, which you specified, allowing up to 50% of the timeslices to be allocated to one transmitter unit, if desired.

The actual sequence of transmitter timeslice allocations for the address silo mode is set up in your task data buffer referenced by the stadd parameter. Certain constraints must be observed when specifying this information, as follows:

- Each entry in the buffer is a byte containing a transmitter unit address.
- At least 20 entries, but not more than 50 entries, must be specified. If less than 20 entries are specified, the driver repeats the entire sequence, as specified, to attain the required minimum of 20 addresses. If more than 50 addresses are specified, no change in timeslice allocation is effected and an IE.VER error status is returned to the task.
- Identical transmitter addresses in either adjacent bytes or in first and last bytes should be avoided. When identical addresses appear in adjacent bytes in this manner, the driver inserts invalid "pad" transmitter addresses between identical addresses, effectively resulting in no-operation timeslices.
- Transmitter addresses are decimal values ranging from 1 to 32 (inclusive) that correspond to addresses implemented on the actual transmitter unit hardware.
- The size parameter must correctly specify the number of address bytes contained in the buffer referenced by the stadd parameter.

14.4 PCL11 TRANSMITTER DRIVER STATUS RETURNS

Table 14-3 lists PCL11 transmitter driver return status codes and probable reasons.

Table 14-3
PCL11 Transmitter Driver Status Returns

Code	Reason
IS.SUC	**Successful completion** The QIO function was successfully completed. If an IO.ATX function was completed, the second status word contains the number of bytes transferred; the message was not truncated. If an IO.SEC function was completed, the second status word contains the current contents of the master section's TMMR.
IS.TNC	**Successful transfer but message truncated** The IO.ATX function was completed, but the message was truncated by the receiver (the receiver buffer is too small). The transmitter unit cannot determine how many words were actually received by the receiver unit; the second word of the I/O status block contains the length of the requested transfer, rather than the actual count of words successfully received in the receiver's buffer.
IE.BAD	**Bad parameter specification** A bad parameter specification was included in the IO.ATX function, or an invalid state parameter or TDM bus timeslice allocation addressing mode was specified in the IO.STC function. This error status is also returned when an IO.STC function, issued to a TDM bus master operating in the address silo mode, refers to a data buffer containing an illegal series of transmitter addresses. An illegal series of addresses occurs when the number of entries specified for the timeslice allocation, plus the required number of pad addresses, either exceeds 50 or is less than 0.
IE.DNR	**Device not ready** This error status return occurs in response to an IO.ATX function when one of the following occurs: • Power failure in this CPU. • Device time-out (no response from the addressed receiver). • Receiver was too slow in accepting or rejecting the transfer request. • The master section is inoperative. This error status is returned only after the number of retries specified in the IO.ATX function have been attempted without success.

(continued on next page)

Table 14-3 (Cont.)
PCL11 Transmitter Driver Status Returns

Code	Reason
IE.VER	**Unrecoverable error** The IO.STC function state setting could not be achieved because the task is not privileged or another device is TDM bus master.
IE.SPC	**Illegal user task buffer** The buffer address specified in the IO.ATF function is outside of the issuing task's address space.
IE.REJ	**Transfer rejected** The data transfer request specified in the IO.ATX function was rejected by the addressed receiver--based on the source CPU identifier of the task issuing the request--and flagword.
IE.FLG	**Event flag already specified** An event flag was previously specified in an IO.STC function.
IE.BBE	**Transmission error** This error status is returned only after the number of retries specified in the IO.ATX function have been attempted without a successful transmission. (Cycle redundancy check errors or parity errors have been detected on each attempt.)
IE.ABO	**Request terminated** This status is returned when a pending I/O function has been aborted in response to an IO.KIL function being issued by the task.
IE.IFC	**Illegal function** A function code was specified in an I/O request that is illegal for PCL11 transmitters.

14.5 QIO MACRO -- PCL11 RECEIVER DRIVER FUNCTIONS

14.5.1 Standard QIO Functions

Table 14-4 lists the standard function of the QIO macro that is valid for the PCL11 receiver driver.

Table 14-4
Standard QIO Functions for PCL11 Receivers

Format	Function
QIO$C IO.KIL,...	Cancel I/O request

14.5.2 Device-Specific QIO Functions

Table 14-5 lists the device-specific functions of the QIO macro that are valid for the PCL11 receiver driver.

Table 14-5
Device-Specific QIO Functions for PCL11 Receivers

Format	Function
QIO$C IO.CRX,...,<tef,bufadd>	CONNECT for reception
QIO$C IO.RTF,...	Reject transfer
QIO$C IO.ATF,...,<stadd,size, retadd>	Accept transfer
QIO$C IO.DRX,...	Disconnect from reception

tef

The number of a "trigger" event flag that is set whenever a flagword is received over the TDM bus.

bufadd

The address of a 2-word buffer containing the transmitter id, trigger status, and the flagword.

stadd

The address of a data buffer to receive the message. This address must occur on a word boundary (even address).

size

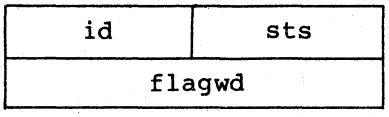
The data buffer size in bytes. The size specified must be an even, positive value.

retadd

The address of a 6-word buffer into which the contents of the six PCL11 receiver hardware registers are returned upon successful completion of the function. Information describing the contents of these registers can be obtained by referring to the hardware documentation supplied with the PCL11 option.

14.5.2.1 **IO.CRX** - This I/O function connects the issuing task to the receiver, if the receiver is not currently connected to another task. When connected, this task is the only task capable of receiving messages by means of the receiver on this CPU. The trigger event flag (a local, common, or group-global event flag) informs the task when a message is pending. It is set when a flagword is received over the TDM bus. When this happens, a significant event is declared and the connected task is considered "triggered." The flagword is the first word transmitted by a transmitter when attempting to send a message to the receiver unit.

The bufadd parameter must be included in this I/O function to specify the address of a 2-word block, as follows:



sts

The current trigger status.

id

The identification code of the transmitter attempting to send the message.

flagwd

The flagword transmitted to the connected receiver.

Based on the information contained in the flagword and the identification code of the transmitter unit, the task can accept or reject the transfer. (Two I/O functions are provided for this purpose; see Sections 14.5.2.2 and 14.5.2.3.) The receiver must respond to the transmitter's request within approximately 1.5 seconds; otherwise, an IE.DNR error status is returned to the task attempting the transmission.

14.5.2.2 **IO.RTF** - This function informs the transmitter device that the message is being rejected by the receiver. Any attempt to issue this I/O function when the trigger event flag is not set is ignored, and an IE.NTR error status is returned to the task.

14.5.2.3 **IO.ATF** - This function informs the transmitter device that the message is being accepted. Parameters specify both the data buffer into which the received data is transferred, and the 6-word buffer that receives the contents of the receiver section hardware registers upon successfully completing the function.

Unlike the IO.RTF function, the IO.ATF function can be issued before the task is triggered. When this is done, the IO.ATF function is queued for reception of any flagword. When the flagword is received, the receiver driver immediately executes the IO.ATF function; the connected task is not triggered and the flagword is not made available to the task. This approach is useful when it is not necessary to examine flagwords or to accept messages based on the source.

14.5.2.4 **IO.DRX** - This function is issued by a task to disconnect the receiver for use by other tasks.

14.6 PCL11 RECEIVER DRIVER STATUS RETURNS

Table 14-6 lists PCL11 receiver driver return status codes and probable reasons.

Table 14-6
PCL11 Receiver Driver Status Returns

Code	Reason
IS.SUC	**Successful completion** The I/O function or triggering of the task was successfully completed. When this status is returned upon completion of the IO.ATF function, the high-order byte of the first word in the I/O status block contains the identification code of the transmitter device that sent the flagword. The second word of the I/O status block contains the number of bytes transferred over the TDM bus. When this status is returned as a result of an IO.CRX function, and the task being triggered, the I/O status block contains information that enables the task to accept or reject the message (see Section 14.5.2.1).
IS.TNC	**Successful transfer but message truncated** This I/O status is returned when the message is terminated because the receiver task message buffer specified in the IO.ATF function is too small to contain the message being received. The second word of the I/O status word contains the number of bytes successfully transferred.
IE.BAD	**Bad parameter specification** A bad parameter specification was included in the requested function.
IE.DNR	**Device not ready** This error status return occurs in response to an IO.RTF or IO.ATF function when one of the following occurs: • Power failure in this CPU. • Device time-out (no response from addressed receiver). • Receiver was too slow in accepting or rejecting the transfer request. • The master section is inoperative.

(continued on next page)

Table 14-6 (Cont.)
PCL11 Receiver Driver Status Returns

Code	Reason
IE.SPC	**Illegal user task buffer** The buffer address specified in the IO.ATF function is outside of the issuing task's address space.
IE.DNA	**Task not connected for reception** The requested function cannot be executed because the task is not connected to the receiver.
IE.DAO	**Data overrun** This I/O status code is returned when the task is triggered, but the previous transfer request has neither been accepted nor rejected. When the task issues an IO.RTF or IO.ATF function, it applies to the new (most recent) flagword; the previous request is ignored.
IE.DAA	**Device already connected for reception** This I/O status code is returned in response to the IO.CRX function when the receiver is already connected to this task or any other task. No operation is performed.
IE.NTR	**Task not triggered** This I/O status code is returned when a task attempts to issue an IO.RTF function prior to the task being triggered.
IE.BBE	**Transmission error** This error status is returned when an IO.ATF function is in progress and a cycle redundancy check error or parity error has been detected.
IE.ABO	**Request terminated** This status is returned when a pending I/O function has been aborted in response to an IO.KIL function being issued by the task.
IE.FHE	**Fatal hardware error** The requested function cannot be executed because of a hardware failure.
IE.IFC	**Illegal function** A function code was specified in an I/O request that is illegal for PCL11 transmitters.

CHAPTER 15

ANALOG-TO-DIGITAL CONVERTER DRIVERS

15.1 INTRODUCTION

The AFC11 and AD01-D analog-to-digital (A/D) converters acquire industrial and laboratory analog data. (AFC11 and AD01-D driver support is not provided on RSX-11M-PLUS systems.) Although each has its own driver, programming for both is quite similar and both are multichannel, programmable gain devices. The AD01-D should not be confused with the ADU01, a UDC module, which is described in Chapter 16. Table 15-1 compares the AFC11 and the AD01-D briefly, and subsequent sections describe these devices in greater detail.

Table 15-1
Standard Analog-to-Digital Converters

	AFC11	AD01-D
Maximum Sampling Rate (Points per Second)	200 (20 per single) channel	Approximately 10,000
Number of Bits	13 or 14	10 or 11
Maximum Number of Analog Channels That Can Be Multiplexed	1024	64

15.1.1 AFC11 Analog-to-Digital Converter

The AFC11 is a differential analog input subsystem for industrial data-acquisition and control systems. It multiplexes signals, selects gain, and performs a 13- or 14-bit A/D conversion under program control. With the use of appropriate signal-conditioning modules, the system can intermix and accept low-level, high-level, and current inputs, with a high degree of noise immunity.

15.1.2 AD01-D Analog-to-Digital Converter

The AD01-D is an extremely fast analog data-acquisition system. It multiplexes signals, selects gain, and performs a 10- or 11-bit A/D conversion under program control. The AD01-D is normally unipolar, but an optional sign-bit facilitates bipolar operation.

15.2 GET LUN INFORMATION MACRO

If a Get LUN Information system directive is issued for a LUN associated with an analog-to-digital converter, word 2 (the first characteristics word) contains all 0s, words 3 and 4 are undefined, and word 5 is not significant, because there is no concept of a default buffer size for analog-to-digital converters.

15.3 QIO$ MACRO

This section summarizes standard and device-specific QIO functions for analog-to-digital converter drivers.

15.3.1 Standard QIO Function

The standard function that is valid for analog-to-digital converters is shown in Table 15-2.

Table 15-2
Standard QIO Function for the A/D Converters

Format	Function
QIO$C IO.KIL,...	Cancel I/O requests

Because all requests are processed within a small amount of time, no in-progress request is ever canceled. This function simply cancels all queued requests.

15.3.2 Device-Specific QIO Function

The device-specific function of the QIO macro that is valid for analog-to-digital converters is shown in Table 15-3.

Table 15-3
Device-Specific QIO Function for the A/D Converters

Format	Function
QIO$C IO.RBC,...,<stadd,size,stcnta>	INITIATE multiple A/D conversions

stadd

The starting address of the data buffer (must be on a word boundary).

size

The control buffer size in bytes (must be even and greater than 0); the data buffer is the same size.

stcnta

The starting address of the control buffer (must be on a word boundary); each control buffer word must be constructed as shown in Table 15-4.

Table 15-4
A/D Conversion Control Word

Bits	Meaning				AFC11	AD01-D
0-11	Channel number				Range: 0-1023	Range: 0-63
12-15	Gain value for this sample, expressed as a bit pattern as follows:				Gain:	Gain:
	15	14	13	12		
	0	0	0	0	1	1
	0	0	0	1	2	2
	0	0	1	0	illegal	4
	0	0	1	1	illegal	8
	0	1	0	0	10	illegal
	0	1	0	1	20	illegal
	0	1	1	0	illegal	illegal
	0	1	1	1	illegal	illegal
	1	0	0	0	50	illegal
	1	0	0	1	100	illegal
	1	0	1	0	illegal	illegal
	1	0	1	1	illegal	illegal
	1	1	0	0	200	illegal
	1	1	0	1	1000	illegal
	1	1	1	0	illegal	illegal
	1	1	1	1	illegal	illegal

15.4 FORTRAN INTERFACE

A collection of FORTRAN-callable subroutines provide FORTRAN programs access to the AFC11 and the AD01-D. These are described in this section. All are reentrant and may be placed in a resident library.

15.4.1 Synchronous and Asynchronous Process Control I/O

The ISA standard provides for synchronous and asynchronous I/O. Synchronous I/O is indicated by appending a "W" to the name of the subroutine (for example, AISQ/AISQW). The synchronous call suspends task execution until the I/O operation is complete. If you use the asynchronous form, execution continues and the calling program must periodically test the status word for completion.

15.4.2 The isb Status Array

The isb (I/O status block) parameter is a 2-word integer array that contains the status of the FORTRAN call, in accordance with ISA convention. This array serves two purposes:

1. It is the 2-word I/O status block to which the driver returns a status code on completion of an I/O operation.

2. The first word of isb receives a status code from the FORTRAN interface in ISA-compatible format, with the exception of the I/O pending condition, which is indicated by a status of 0. The ISA standard code for this condition is +2.

The meaning of the contents of isb varies, depending on the FORTRAN call that has been executed; but Table 15-5 lists certain general principles that apply. The section describing each subroutine provides further details.

Table 15-5
Contents of First Word of isb

Contents	Meaning
isb(1) = 0	Operation pending; I/O in progress
isb(1) = 1	Successful completion
isb(1) = 3	Interface subroutine unable to generate QIO directive, or number of samples is 0
3 < isb(1) < 300	QIO directive rejected and actual error code = -(isb(1) - 3)
isb(1) 300	Driver rejected request and actual error code = -(isb(1) - 300)

Unless otherwise specified, the value of isb(2) is the value returned by the driver to the second word of the I/O status block.

FORTRAN interface subroutines depend on asynchronous system traps to set their status. Thus, if the trap mechanism is disabled, proper status cannot be set.

15.4.3 FORTRAN Subroutine Summary

Table 15-6 lists the FORTRAN interface subroutines supported for the AFC11 and AD01-D under RSX-11M.

Table 15-6
FORTRAN Interface Subroutines for the AFC11 and AD01-D

Subroutine	Function
AIRD/AIRDW	Perform input of analog data in random sequence
AISQ/AISQW	Read a series of sequential analog input channels
ASADLN	Assign a LUN to the AD01-D
ASAFLN	Assign a LUN to the AFC11

The following subsections briefly describe the function and format of each FORTRAN subroutine call. Note the use of ASADLN and ASAFLN to assign a default logical unit number.

15.4.4 AIRD/AIRDW: Performing Input of Analog Data in Random Sequence

The ISA standard AIRD/AIRDW FORTRAN subroutines input analog data in random sequence. These calls are issued as follows:

CALL {AIRD / AIRDW} (inm,icont,idata,[isb],lun)

inm

The number of analog input channels.

icont

An integer array containing terminal connection data-channel number (right-justified in bits 0-11) and gain (bits 12-15), as shown in Table 15-4.

idata

An integer array to receive the converted values.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number, which is a required parameter.

The isb array has the standard meaning defined in Section 15.4.2. If inm = 0, then isb(1) = 3. The contents of idata are undefined if an error occurs.

15.4.5 AISQ/AISQW: Reading Sequential Analog Input Channels

The ISA standard AISQ/AISQW FORTRAN subroutines read a series of sequential analog input channels. These calls are issued as follows:

CALL {AISQ / AISQW} (inm,icont,idata,[isb],lun)

inm

The number of analog input channels.

icont

An integer array containing terminal connection data-channel number (right-justified in bits 0-11) and gain (bits 12-15), as shown in Table 15-4.

idata

An integer array to receive the converted values.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number, which is a required paramenter.

For sequential analog input, channel number is computed in steps of one, beginning with the value specified in the first element of icont. The channel number field is ignored in all other elements of the array.

The driver takes the gain it uses for each conversion from the respective element in icont. Thus, even though the channel number is ignored in all but the first element of icont, the gain must be specified for each conversion to be performed.

The isb array has the standard meaning defined in Section 15.4.2. If inm = 0, then isb(1) = 3. The contents of idata are undefined if an error occurs.

15.4.6 ASADLN: Assigning a LUN to the AD01-D

The ASADLN FORTRAN subroutine assigns the specified LUN to the AD01-D and defines it as the default logical unit number to use whenever you omit a LUN specification from an AIRD(W)/AISQ(W) subroutine call. Issue it as follows:

```
CALL ASADLN (lun,[isw],[iun])
```

lun

The logical unit number to be assigned to the AD01-D and defined as the default unit.

isw

An integer variable to which the result of the ASSIGN LUN system directive is returned.

iun

The unit number to be assigned. If unspecified, a value of 0 is assumed.

Only the LUN specified in the last call to ASADLN or ASAFLN is defined as the default unit.

15.4.7 ASAFLN: Assigning a LUN to the AFC11

The ASAFLN FORTRAN subroutine assigns the specified LUN to the AFC11 and uses it as the default logical unit number whenever you omit a LUN specification from an AIRD(W)/AISQ(W) subroutine call. Issue it as follows:

```
CALL ASAFLN (lun,[isw],[iun])
```

lun

The logical unit number to be assigned to AFC11 and defined as the default unit.

isw

An integer variable to which the status from the ASSIGN LUN system directive is returned.

iun

The unit number to be assigned. If unspecified, a value of 0 is assumed.

Only the LUN specified in the last call to ASAFLN or ASADLN is defined as the default unit.

15.5 STATUS RETURNS

The error and status conditions listed in Table 15-7 are returned by the analog-to-digital converter drivers described in this chapter.

Table 15-7
A/D Converter Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of A/D conversions performed.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The specified I/O operation was canceled with IO.KIL while still in the I/O queue.
IE.BAD	**Bad parameter** An illegal specification was supplied for one or more of the device-dependent QIO parameters (words 6-11). For the analog-to-digital converters, this code indicates that a bad channel number or gain code was specified in the control buffer.
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a data or control buffer, but only word alignment is legal for analog-to-digital converters. Alternatively, the length of the data and control buffer is not an even number of bytes.
IE.DNR	**Device not ready** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. For the AFC11, this code is returned if an interrupt time-out occurred or the power failed. In the case of the AD01-D, which is not operated in interrupt mode, this code indicates a software time-out occurred (that is, a conversion did not complete within 30 microseconds).

(continued on next page)

Table 15-7 (Cont.)
A/D Converter Status Returns

Code	Reason
IE.IFC	**Illegal function** A function code was specified in an I/O request that is illegal for analog-to-digital converters.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The data or control buffer specified for a conversion request was partially or totally outside the address space of the issuing task. Alternately, a byte count of 0 was specified.

FORTRAN interface values for these subroutines are presented in Section 15.5.1.

15.5.1 FORTRAN Interface Values

The values listed in Table 15-8 are returned in FORTRAN subroutine calls.

Table 15-8
FORTRAN Interface Values

Status Return	FORTRAN Value
IS.SUC	+01
IS.PND	+00
IE.ABO	+315
IE.ADP	+101
IE.BAD	+301
IE.BYT	+319
IE.DAO	+313
IE.DNR	+303
IE.IEF	+100
IE.IFC	+302
IE.ILU	+99
IE.NOD	+323
IE.ONP	+305
IE.PRI	+316
IE.RSU	+317
IE.SDP	+102
IE.SPC	+306
IE.ULN	+08
IE.UPN	+04

15.6 FUNCTIONAL CAPABILITIES

The AFC11 and AD01-D operate only in multisample mode, because you can simulate single-sample mode by simply specifying one sample. Multisample mode permits many channels to be sampled at approximately the same time without requiring your task to queue multiple I/O requests.

The maximum number of channels in the configuration is specified at system-generation time. This value is stored in the respective AFC11 and AD01-D unit control blocks.

15.6.1 Control and Data Buffers

Your task must define two buffers of equal size: the control buffer and the data buffer. The former contains the control words needed to perform one A/D conversion per channel specified. Each control word indicates the channel to be sampled and the gain to be applied (see Table 15-4).

The data buffer receives the results of the conversions. Each result is placed in the data buffer location that corresponds to the control word that specified it.

15.7 PROGRAMMING HINTS

This section contains important information about programming the analog-to-digital converter drivers described in this chapter.

15.7.1 Use of A/D Gain Ranges

Note that the A/D gain ranges overlap. The key to successful use of the A/D converters is to change to a higher gain whenever a full-scale reading is imminent, and to change to a lower gain whenever the last A/D value recorded was less than half of full scale. This method maintains maximum resolution while avoiding saturation.

15.7.2 Identical Channel Numbers on the AFC11

When requesting sampling of more than one channel, you should not specify multiple sampling of a single channel without 10 or more intervening samples on other channels. This ensures 50 milliseconds between samples on a single channel. If sampling occurs more often than this on a single channel, partial results are returned (see Section 15.7.3).

15.7.3 AFC11 Sampling Rate

Although the AFC11 can sample a maximum of 200 points per second, a single channel can only be sampled at 20 points per second. Because the channel capacitor needs 50 milliseconds to recharge after each conversion, more frequent sampling may result in partial readings. If this occurs, your task receives no indication that information is being lost. To ensure that information is not lost on any one channel, your task should sample approximately 10 other channels before returning to the first one.

15.7.4 Restricting the Number of AD01-D Conversions

The AD01-D is an extremely fast device, providing a 25-microsecond conversion rate, and is driven by program to minimize system overhead. However, an excessive number of conversions in a single request essentially locks out the rest of the system, because the driver does not return control to the system until it has finished all the specified conversions. No other task can run, although interrupts can still occur and are processed.

CHAPTER 16

UNIVERSAL DIGITAL CONTROLLER DRIVER

16.1 INTRODUCTION

The UDC11 is a digital input/output system for industrial and process control applications. It interrogates or drives, or both, up to 252 directly addressable digital sense modules, control modules, or combinations thereof. The UDC11 operates under program control as a high-level digital multiplexer, interrogating digital inputs and driving digital outputs. (UDC11 driver support is not provided on RSX-11M-PLUS systems.)

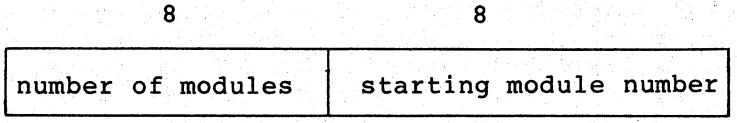
The UDC driver supports either the UDC11 or ICS11 subsystem. The ICS11 (Industrial Control Subsystem) operates as an input/output device that is functionally similar to the UDC11. A maximum of 16 I/O modules can be placed in one ICS11 subsystem; up to 12 ICS11s can be interfaced to one computer system. The ICS11 subsystem is also supported by the ICS/ICR-11 driver described in Chapter 19. The reader should consult that chapter for a comparison of driver features.

While performing analog-to-digital conversions, the UDC11 driver can handle other functions, such as contact or timer interrupts or latching output. These functions are performed immediately, without requiring any in-progress analog-to-digital conversions to first be completed.

Unlike other RSX-11M I/O device drivers, the UDC11 driver is neither a multicontroller nor a multiunit driver.

16.1.1 Creating the UDC11 Driver

Each installation must assemble the driver source module with a prefix file that defines the particular hardware configuration. The prefix file is created during system generation according to your response to questions relating to the UDC11. This file is named RSXMC.MAC and includes symbolic definitions of the UDC11 configuration. These definitions encode the relative module number and the number of modules for each generic type specified in the system generation dialog. The encoding has the following format:



One or more of the following symbols is generated:

Symbol	Module Type
U$$ADM	Analog input
U$$AOM	Analog output
U$$CIM	Contact interrupt
U$$CSM	Contact sense input
U$$LTM	Latching digital output
U$$SSM	Single-shot digital output
U$$TIM	Timer (I/O counter)

Note that all modules of a given type must be installed together in sequential slots.

16.1.2 Accessing UDC11 Modules

RSX-11M provides two methods of accessing the UDC11:

1. A QIO macro call issued to the driver
2. Restricted direct access by any task to I/O page registers dedicated to the UDC11

The first method, access through the driver, is required to service interrupting modules and to set and record the state of latching digital output modules.

The second method, direct access, is a high-speed, low-overhead way to service noninterrupting modules. The following functions may be performed in this manner:

- Analog output
- Contact sense input
- Single-shot digital output
- Read a contact interrupt module
- Read a timer module

16.1.2.1 **Driver Services** - The driver services the following types of modules:

- Contact interrupt
- Timer (I/O counter)
- Analog input
- Latching digital output

Contact and timer interrupts need not be serviced by a single task. One task may be connected to contact interrupts, and another to timer interrupts. A nonprivileged task can connect to either or both of these classes by providing a circular buffer to receive interrupt information and an event flag to allow triggering of the task whenever a buffer entry is made.

16.1.2.2 **Direct Access** - A global common block within the I/O page provides restricted direct access to the UDC11 device registers. In a mapped system, the length of the block is set to prevent access to other device registers. In an unmapped system, the use of the common block is optional, unless you use ISA FORTRAN calls. The ISA routines refer symbolically to the UCD11 registers, and thus require the use of global common. Section 16.4 explains direct access more fully.

16.2 GET LUN INFORMATION MACRO

If a Get Lun Information system directive is issued for a LUN associated with the UDC11, word 2 (the first characteristics word) contains all zeros, words 3 and 4 are undefined, and word 5 is not significant, because there is no concept of a default buffer size for universal digital controllers.

16.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for the UDC11 driver. In issuing them, note the numbering conventions described in 16.7.2.

16.3.1 Standard QIO Function

The standard function that is valid for the UDC11 is shown in Table 16-1.

Table 16-1
Standard QIO Function for the UDC11

Format	Function
QIO$C IO.KIL,...	Cancel I/O requests

IO.KIL cancels all queued requests and disconnects all interrupt connections, but does not stop any I/O that is currently in progress.

16.3.2 Device-Specific QIO Functions

Table 16-2 summarizes device-specific QIO functions that are supported for the UDC12.

Table 16-2
Device-Specific QIO Functions for the UDC11

Format	Function
QIO$C IO.CCI,...,<stadd,sizb,tevf>	CONNECT a buffer to contact interrupts
QIO$C IO.CTI,...,<stadd,sizb,tevf,arv>	CONNECT a buffer to timer interrupts
QIO$C IO.DCI,...	Disconnect a buffer from contact interrupts
QIO$C IO.DTI,...	Disconnect a buffer from timer interrupts
QIO$C IO.ITI,...,<mn,ic>	INITIALIZE a timer
QIO$C IO.MLO,...,<opn,pp,dp>	OPEN or close latching digital output points
QIO$C IO.RBC,...,<stadd,size,stcnta>	INITIATE multiple A/D conversions

stadd

The starting address of the data buffer (must be on a word boundary).

sizb

The data buffer size in bytes (must be even and large enough to include a 2-word buffer header plus one data entry; the buffer may cross a 4K boundary).

tevf

The trigger event flag number.

arv

The starting address of the table of initial/reset values (must be on a word boundary).

mn

The module number.

ic

The initial count.

opn

The first latching digital output point number, which must be on a module boundary (evenly divisible by 16).

pp

The 16-bit mask.

dp

The data pattern.

size

The control buffer size in bytes (must be even and greater than 0); the data buffer is the same size.

stcnta

The starting address of the control buffer (must be on a word boundary); each control buffer word must be constructed as shown in Table 16-3.

The following sections describe the functions listed in Table 16-2.

Table 16-3
A/D Conversion Control Word

Bits	Meaning	ADU01
0-11	Channel number	Range: 0-4095
12-15	Gain value for this sample, expressed as a bit pattern as follows:	Gain:

15	14	13	12	
0	0	0	0	1
0	0	0	1	2
0	0	1	0	Illegal
0	0	1	1	Illegal
0	1	0	0	10
0	1	0	1	20
0	1	1	0	Illegal
0	1	1	1	Illegal
1	0	0	0	50
1	0	0	1	100
1	0	1	0	Illegal
1	0	1	1	Illegal
1	1	0	0	200
1	1	0	1	1000
1	1	1	0	Illegal
1	1	1	1	Illegal

16.3.2.1 **Contact Interrupt Digital Input (W733 Modules)** - Digital input and change of state information from contact interrupt modules is reported in a requester-provided circular buffer. The buffer consists of a 2-word header, followed by a data area in the following format:

1	driver index
2	user task index
3	entry
4	entry
.	.
.	.
.	.

Whenever a change of state occurs in one or more contact points, an interrupt is generated. The UDC11 driver gains control, determines whether the change of state is of interest (that is, a contact closure and point closing (PCL) is set on the module), and then optionally makes an entry in the data area of the buffer, updates the index words, and sets the trigger event flag of the connected task.

Each entry consists of five words in the following format:

Word	Contents
0	Entry existence indicator
1	Change of state (COS) indicator
2	Module data (current point values)
3	Module number (interrupting module)
4	Generic code (interrupting module)

The driver enters data in the location currently indicated by the driver index. This pointer can be considered as a FORTRAN index into the buffer; that is, the first location of the buffer is associated with the index 1. The beginning of the data area is the location of the first entry (index 3). Entries are made in a circular fashion, starting at the beginning of the data area, filling in order of increasing memory address to the end of the data area, and then wrapping around from the end to the beginning of the data area.

The connected task must maintain its own pointer (the task's index) to the location in the buffer where it is next to retrieve contact interrupt data. When a task is triggered by the driver, it should process data in the buffer starting at the location indicated by its pointer and continuing in a circular fashion until the two pointers are equal or a zero entry existence indicator is encountered. Equality of pointers means that the connected task has retrieved all the contact interrupt information that the driver has entered into the buffer.

The entry existence indicator is set to nonzero when a buffer entry is made. When a requester has removed or processed an entry, he must clear the existence indicator to free the buffer entry position.

If data input occurs in a burst sufficient to overrun the buffer, data is discarded and a count of data overruns is incremented. The nonzero entry existence indicator also serves as an overrun indicator. A positive value (+1) indicates no overruns between entries; a negative value is the two's complement of the number of times data have been discarded between entries.

The module number indicates a module on which a change of state in the direction of interest has been recognized for one or more discrete points. The direction of the change may be from 0 to 1 or 1 to 0, depending on the PCL (point closing) and POP (point opening) module jumpers. The change of state (COS) indicator specifies which point or points of the module have changed state.

The bit position of an on-bit in the COS indicator provides the low-order bits (3-0) of a point number and the module number provides the high-order bits (15-4). The module data indicates the logical value (polarity) of each point in the module at the time of the interrupt.

Contact interrupt data can be reported to only one task. The functions IO.CCI and IO.DCI in Table 16-2 are provided to enable a task to connect and disconnect from contact interrupts. If the connection is successful, the second word of the I/O status block contains the number of words passed per interrupt in the low-order byte and the initial FORTRAN index to the beginning of the data area in the high-order byte.

NOTE

The size of the data area must be a multiple of the entry size.

16.3.2.2 **Timer (W734 I/O Counter Modules)** - A timer (I/O counter) module is a clock that is initialized (loaded), counts up or down, and then causes an interrupt. The UDC11 driver treats such modules in a way similar to that in which it handles contact interrupts. The requester provides a circular buffer similar to that for contact interrupts. Each entry consists of four words in the following format:

Word	Contents
0	Entry existence indicator
1	Module data (current value)
2	Module number (interrupting module)
3	Generic code (interrupting module)

The IO.CTI function in Table 16-2 enables a task to connect to timer interrupts. The driver uses the table of initial/reset values initially to load the timers and to reload them on interrupt (overflow). The table contains one word for each timer module. The driver uses the contents of the first word to load the first module, and so forth. If a timer has a nonzero value when it interrupts, the driver does not reload it so that self-clocking modules and modules that interrupt on half count can continue counting from the initial value.

The IO.DTI function in Table 16-2 disconnects a task from timer interrupts, and the IO.ITI function provides the capability of initializing a single timer. Requests to initialize a counter are valid only if the issuing task has connected a buffer for receiving counter interrupts.

NOTE

The size of the data area must be a multiple of the entry size.

16.3.2.3 **Latching Digital Output (M685, M803, and M805 Modules)** - Each module has 16 latching digital output points. The IO.MLO function in Table 16-2 opens or closes a set of up to 16 points. Bit n of the mask and data pattern corresponds to the point opn + n. If a bit in the mask is set, the corresponding point is opened or closed, depending on whether the corresponding bit in the data pattern is clear or set. If a bit in the mask is clear, the corresponding point remains unaltered.

16.3.2.4 **Analog-to-Digital Converter (ADU01 Module)** - Each ADU01 module has eight analog input channels. The IO.RBC function in Table 16-2 initiates A/D conversions on multiple ADU01 input channels. Restrictions on maximum sampling rates are the same as those defined for the AFC11 in Chapter 15.

The converted analog value is returned as 12 bits, left-justified, in a 16-bit word with the low-order 4 bits set to 0.

16.3.2.5 **ICS11 Analog-to-Digital Converter (IAD-IA Module)** - Each IAD-IA Module has eight analog input channels. The channel capacity may be expanded to 120 by the addition of IMX-IA multiplexers. Each multiplexer adds 16 input channels to the converter. Restrictions on maximum sampling rates are the same as those defined for the AFC11 in Chapter 15. The IAD-IA module preempts eight module slots regardless of the number of IMX-IA multiplexers installed.

For addressing purposes, each converter occupies a block of 120 channels. Thus, A/D converter 0 is addressed by referencing channels 0 through 119; A/D converter 1 is addressed by referencing channels 120 through 239, and so forth. When fewer than seven multiplexers are installed, not all addresses within the block are valid.

The converted analog value is returned as 12 bits, left-justified, in a 16-bit word with the low-order 4 bits set to 0.

16.4 DIRECT ACCESS

Section 16.1.2 describes UDC11 functions that may be performed by referencing a module through its physical address in the I/O page. Under RSX-11M such access is accomplished by one of the following methods:

1. A privileged task or any task running in an unmapped system has unrestricted access to the I/O page, and may therefore access each module by absolute address.

2. Using the Task Builder, a task may link to a global common area whose physical address limits span a set of locations in the I/O page. This method applies to either a mapped or unmapped system.

The latter method allows a task to be transported to any other system simply by relinking. Furthermore, in a mapped system the memory management hardware aborts all references to device registers outside the physical address limits of the common block.

The operations required to implement each method may be summarized as follows:

1. Unrestricted access to the I/O page

   a. Create an object module that defines the UDC11 configuration through a list of absolute global addresses and addressing limits for each module type.

   b. Include the object module in the system library file.

   c. Create a task containing the appropriate global references. Such references are resolved when the task builder searches the system library file.

Steps a and b are executed once, during system generation (see the RSX-11M System Generation and Installation Guide). Step c is performed each time a task is created that references the UDC12.

2. Access to the I/O page through a Global Common Block

   a. Create an object module that defines the UDC11 configuration through a list of relocatable global addresses and addressing limits for each module type.

   b. Link the object module using the Task Builder to create an image of the Global Common block on disk.

   c. Use the SET command to define a common block that resides on the I/O page.

   d. Use the INSTALL MCR command to make the Global Common Block resident in memory.

   e. Create a task containing the appropriate global references. Such references are resolved by directing the Task Builder to link the task to the common block.

Steps a through d are executed once, during system generation. Step e is performed each time you create a task that references the UDC11 common block. The following paragraphs describe each step in detail.

16.4.1 Defining the UDC11 Configuration

The source module, ODCOM.MAC, resides on the RK05 cartridge of the RSX-11M RK distribution kit labeled EXECUTIVE SOURCE. For RP distribution kits, it resides on the RP image. The file is located under UIC [11,10]. UDCOM.MAC, when assembled with the proper prefix file, provides global definitions for the following parameters:

- The starting address of each module type
- The highest point number within a given module type
- The highest module number within a given module type

The last two parameters are absolute quantities that you may use to prevent a task from referencing a module that is nonexistent or out of limits.

By means of conditional assembly, the list of addresses may be created as absolute symbols defining locations in the I/O page, or as symbols within a relocatable program section that you may use when building and linking to the UDC11 Global Common area.

16.4.1.1 **Assembly Procedure for UDCOM.MAC** -

UDCOM.MAC is assembled with the RSX-11M configuration parameters contained in the file RSXMC.MAC.

To create relocatable module addresses, either the parameter U$$DCM or M$$MGE must be defined. M$$MGE is included in RSXMC.MAC if memory management was specified when the system was generated. If not, you should edit the file to include the following definition:

```
U$$DCM=0
```

The file may then be assembled using the MCR command:

```
>MAC UDCOM,UDLST=[11,10]RSXMC,UDCOM
```

This command invokes the MACRO-11 assembler, which processes the input files RSXMC.MAC and UDCOM.MAC to create UDCOM.OBJ and UDLST.LST.

To create absolute module addresses, both U$$DCM and M$$MGE must be undefined. Edit RSXMC.MAC, if necessary, to remove definitions and then invoke the MACRO-11 assembler with the following MCR command:

```
>MAC UDCDF,UDLST=[11,10]RSXMC,UDCOM
```

In this sequence the files UDCDF.OBJ and UDLST.LST are created from the specified source modules. UDCDF.OBJ contains the module addresses in absolute form.

16.4.1.2 **Symbols Defined by UDCOM.MAC** -

This section lists the symbolic definitions created by UDCOM.MAC.

The following symbols define the absolute or relocatable address of the first module of a given type:

Symbol	Module Type
$.ADM	Analog input
$.AOM	Analog output
$.CIM	Contact interrupt
$.CSM	Contact sense input
$.LTM	Latching digital output
$.SSM	Single-shot digital output
$.TIM	Timer (I/O counter)

The addresses in relocatable form are defined in a program section named UDCOM having the attributes:

REL - relocatable
OVR - overlaid
D - data
GBL - global scope

Note that these attributes correspond to those attached to a named common block within a FORTRAN program.

In either the absolute or relocatable case, individual modules are referenced by the corresponding symbolic address plus a relative module index.

The following symbols define the highest digital point within a module type:

Symbol	Module Type
P$.CIM	Contact interrupt
P$.CSM	Contact sense input
P$.LTM	Latching digital output
P$.SSM	Single-shot digital output

The highest point number is defined relative to the first point on the first module of a specific type. For example, if two contact interrupt modules are installed, the symbol P$.CIM has octal value of 37.

The following symbols define the highest module number within a given module type.

Symbol	Module Type
M$.ADM	Analog input
M$.AOM	Analog output
M$.CIM	Contact interrupt
M$.CSM	Contact sense input
M$.LTM	Latching digital output
M$.SSM	Single-shot digital output
M$.TIM	Timer (I/O counter)

The highest module number is defined relative to the first module of a given type. Thus, based on the previous example, M$.CIM has a value of 1.

16.4.2 Including UDC11 Symbolic Definitions in SYSLIB.OLB

As described in Section 16.4, a task having unrestricted access to the I/O page may reference a UDC11 module by absolute address. The object module UDCDF contains symbolic definitions of absolute module addresses and may be included in the System Object Module Library:

```
SY:[1,1]SYSLIB.OLB
```

The Task Builder searches this file to resolve any undefined globals remaining after all input files have been processed.

The following example illustrates the procedure for including the file UDCDF.OBJ in the library:

```
>SET /UIC=[1,1]
>LBR SYSLIB/IN=[200,200]UDCDF
```

The SET MCR command is issued to establish the current UIC as [1,1]. Next, the RSX11M Librarian is invoked and instructed, through the use of the /IN switch, to include the object module UDCDF.OBJ in the file SYSLIB.OLB.

16.4.3 Referencing the UDC11 through a Global Common Block

The following sections define the procedure for creating a global common block in the I/O PAGE, making the block resident in memory, and creating a task that references UDC11 modules within the block. Examples are given for both mapped and unmapped systems.

16.4.3.1 **Creating a Global Common Block** - The following sequence illustrates the use of the object file UDCOM.OBJ to create a disk image of the global common area in a mapped system:

```
>SET /UIC=[1,1]
>TKB
TKB>UDCOM/MM,LP:,SY:UDCOM/PI/-HD=[200,200]UDCOM
TKB>/
ENTER OPTIONS:
TKB>PAR=UDCOM:0:1000
TKB>STACK=0
TKB>/
```

In the above example, a current UIC of [1,1] is established and the Task Builder is initiated. The initial input line to the Task Builder specifies the following files:

- A core image output file to be named UDCOM.TSK
- A memory map output to the line printer
- A symbol table file to be named UDCOM.STB

All files reside on SY: under UIC [1,1]. The single input file UDCOM.OBJ, containing the UDC11 address definitions as relocatable values, constitutes the input.

The switches specified for the output files convey the following information to the Task Builder:

/MM indicates that the core image of the common block resides on a system with Memory Management.

/PI indicates that the core image is position independent; that is, the virtual address of the common block may appear on any 4K boundary within a task's address space.

/-HD indicates that the core image does not contain a header. A header is only required for a core image file that is to be installed and executed as a task.

The names of the partition, task file, and symbol-table files must agree.

You must use the STACK option to eliminate the stack space.

The following sequence illustrates the corresponding procedure for an unmapped system:

```
>SET /UIC=[1,1]
>TKB
TKB>UDCOM/-MM,LP:,SY:UDCOM/PI/-HD=[200,200]UDCOM
TKB>/
ENTER OPTIONS:
TKB>STACK=0
TKB>PAR=UDCOM:171000:1000
TKB>/
```

Again the task builder is requested to produce a core image and symbol table file under the UIC [1,1], and a map file on the line printer from the input file UDCOM.OBJ. The output file switches convey the following information:

/-MM indicates that the core image of the common block resides on an unmapped system.

/PI Indicates that the core image is position independent. In an unmapped system, the core image is fixed in the same address space for all tasks; however, the global symbols defined in the symbol table file retain the relocatable attribute.

/-HD indicates that a core image without a header is to be created.

The PAR option specifies the base and length of the common area to coincide with the standard UDC11 addresses in the I/O page. All references to the common block by tasks are resolved within this region.

16.4.3.2 **Making the Common Block Resident** - The following SET command creates a UDC11 common block residing in the I/O page for a mapped system:

```
>SET /MAIN=UDCOM:7710:10:DEV
```

The corresponding command in an unmapped system is:

```
>SET /MAIN=UDCOM:1710:10:DEV
```

The preceding sequence specifies the allocation of a common block in the I/O page whose physical address limits correspond to the UDC11 standard locations. Note that the address bounds and length are defined in units of 32 words.

The command

```
>INS [1,1]UDCOM
```

declares the common block resident in the system.

16.4.3.3 **Linking a Task to the UDC11 Common Block** - A task may access UDC11 modules by linking to the common block as follows:

```
TKB>TASK,LP:=TASK.OBJ
TKB>/
ENTER OPTIONS:
TKB> COMMON=UDCOM:RW
TKB>/
```

The above sequence is valid for either a mapped or unmapped system. In both cases the Task Builder links the task to the common block by resolving references to the Global symbol definitions contained in UDCOM.STB. If memory management is present, the Executive maps the appropriate physical locations into the task's virtual addressing space when the task is made active.

16.5 FORTRAN INTERFACE

A collection of FORTRAN-callable subroutines provide FORTRAN programs access to the UDC12. These are described in this section. All are reentrant and may be placed in a resident library.

Instead of using the FORTRAN-callable subroutines described in this section, a FORTRAN program may use the global common feature described in Section 16.4 to reference UDC11 modules directly in the I/O page, as shown in the following example:

```
C
C       UDC11 GLOBAL COMMON
C
        COMMON /UDCOM/ ICSM(10),IAO(10)
C
C       READ CONTACT SENSE MODULE 1 DIRECTLY
C
        ICS=ICSM(1)
```

Note that the position of each module type must correspond to the sequence in which storage is allocated in the common statements.

16.5.1 Synchronous and Asynchronous Process Control I/O

The ISA standard provides for synchronous and asynchronous process I/O. Synchronous I/O is indicated by appending a "W" to the name of the subroutine (for example, AO/AOW). But due to the fact that nearly all UDC11 I/O operations are performed immediately, in most cases the "W" form of the call is retained only for compatibility and has no meaning under RSX-11M. In the case of A/D input, however, the "W" form is significant: The synchronous call suspends task execution until input is complete. You use the asynchronous form, execution continues and the calling program must periodically test the status word for completion.

16.5.2 The isb Status Array

The isb (I/O status block) parameter is a 2-word integer array that contains the status of the FORTRAN call, in accordance with ISA (Instrument Society of America) convention. This array serves two purposes:

1. It is the 2-word I/O status block to which the driver returns an I/O status code on completion of an I/O operation.

2. The first word of isb receives a status code from the FORTRAN interface in ISA-compatible format, with the exception of the I/O pending condition, which is indicated by a status of 0. The ISA standard code for this condition is +2.

The meaning of the contents of isb varies, depending on the FORTRAN call that has been executed; but Table 16-4 lists certain general principles that apply. The section describing each subroutine gives more details.

In some cases, the values or states of points being read, pulsed, or latched are returned to isb word 2.

FORTRAN interface subroutines for analog input depend on asynchronous system traps to set their status. Thus, if the trap mechanism is disabled, proper status cannot be set.

Table 16-4
Contents of First Word of isb

Contents	Meaning
isb(1) = 0	Operation pending; I/O in progress
isb(1) = 1	Successful completion
isb(1) = 3	Interface subroutine unable to generate QIO directive or number of points requested is zero
3 < isb(1) < 300	QIO directive rejected and actual error code = -(isb(1) - 3)
isb(1) > 300	Driver rejected request and actual error code = -(isb(1) - 300)

For direct access calls (indicated in Table 16-5), errors are detected and returned by the FORTRAN interface subroutine itself, rather than by the driver. Although the use of a 2-word status block is therefore unnecessary, these errors are returned in standard format to retain compatibility with functions called through QIO directives and handled by other drivers. Errors of this type that may be returned are:

isb(1) = 3	Number of points requested is 0
isb(1) = +321	Invalid UDC11 module

16.5.3 FORTRAN Subroutine Summary

Table 16-5 lists the FORTRAN interface subroutines supported for the UDC11 under RSX-11M. (D) indicates a direct access call, and the optional logical unit number for such a call may be specified to retain compatibility with RSX-11D; but this specification is ignored by RSX-11M.

The following subsections briefly describe the function and format of each FORTRAN subroutine call. Note the use of ASUDLN to specify a default logical unit number. Also consider the numbering conventions described in 16.7.2.

The following FORTRAN functions do not perform I/O directly, but facilitate conversions between BCD and binary.

Convert four BCD digits to a binary number:

```
IBIN = KBCD2B(IBCD)
```

Convert a binary number to four BCD digits:

```
IBCD = KB2BCD(IBIN)
```

Table 16-5
FORTRAN Interface Subroutines for the UDC11

Subroutine	Function
AIRD/AIRDW	Perform input of analog data in random sequence
AISQ/AISQW	Read a series of sequential analog input channels
AO/AOW	Perform analog output on several channels (D)
ASUDLN	Assign a LUN to the UDC11
CTDI	Connect a circular buffer to receive contact interrupt data

(continued on next page)

Table 16-5 (Cont.)
FORTRAN Interface Subroutines for the UDC11

Subroutine	Function
CTTI	Connect a circular buffer to receive timer interrupt data
DFDI	Disconnect a buffer from contact interrupts
DFTI	Disconnect a buffer from timer interrupts
DI/DIW	Read several 16-point contact sense fields (D)
DOL/DOLW	Latch or unlatch several 16-point fields
DOM/DOMW	Pulse several 16-point fields (D)
RCIPT	Read the state of a single contact interrupt point (D)
RDCS	Read the contents of a contact interrupt circular buffer, returning data on only those points that have changed state
RDDI	Read the contents of a contact interrupt circular buffer, one point for each call
RDTI	Read the contents of a timer interrupt circular buffer, one entry for each call
RDWD	Read the contents of a contact interrupt circular buffer, returning 16 bits of module data and change-of-state information
RSTI	Read a single timer module (D)
SCTI	Set a timer module to an initial value

16.5.4 AIRD/AIRDW: Performing Input of Analog Data in Random Sequence

The ISA standard AIRD/AIRDW FORTRAN subroutines input analog data in random sequence. These calls are issued as follows:

```
CALL {AIRD } (inm,icont,idata,[isb],[lun])
     {AIRDW}
```

inm

The number of analog input channels.

icont

An integer array containing terminal connection data-channel number (right-justified in bits 0-11) and gain (bits 12-15), as shown in Table 16-3.

idata

An integer array to receive the converted values.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The optional logical unit number.

The isb array has the standard meaning defined in Section 16.5.2. If inm = 0, then isb(1) = 3. The contents of idata are undefined if an error occurs.

16.5.5 AISQ/AISQW: Reading Sequential Analog Input Channels

The ISA standard AISQ/AISQW FORTRAN subroutines read a series of sequential analog input channels. These calls are issued as follows:

```
CALL {AISQ } (inm,icont,idata,[isb],[lun])
     {AISQW}
```

inm

The number of analog input channels.

icont

An integer array containing terminal connection data-channel number (right-justified in bits 0-11) and gain (bits 12-15), as shown in Table 16-3.

idata

An integer array to receive the converted values.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The optional logical unit number.

For sequential analog input, the routine computes the channel number in steps of one, beginning with the value that you specified in the first element of icont. The routine ignores the channel number field in all other elements of the array.

The routine takes the gain that it uses for each conversion from the respective element in icont. Thus, even though the channel number is ignored in all but the first element of icont, the gain must be specified for each conversion to be performed.

The isb array has the standard meaning defined in Section 16.5.2. If inm = 0, then isb(1) = 3. The contents of idata are undefined if an error occurs.

16.5.6 AO/AOW: Performing Analog Output

The ISA standard AO/AOW FORTRAN subroutines initiate analog output on several channels. These calls are issued as follows:

CALL {AO / AOW} (inm,icont,idata,[isb],[lun])

inm

The number of analog output channels.

icont

An integer array containing the channel numbers.

idata

An integer array containing the output voltage settings, in the range 0-1023.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number (ignored if present).

The isb array has the standard meaning defined in Section 16.5.2.

16.5.7 ASUDLN: Assigning a LUN to the UDC11

The ASUDLN FORTRAN subroutine assigns the specified LUN to the specified unit and uses it as the default logical unit number whenever a LUN specification is omitted from a UDC11 subroutine call. It is issued as follows:

```
CALL ASUDLN (lun,[isw],[iun])
```

lun

The logical unit number to be assigned to the specified unit, and defined as the default.

isw

An integer variable to which the result of the ASSIGN LUN system directive is returned.

iun

An integer defining the UDC11 unit number. If no number is specified, 0 is assumed.

16.5.8 CTDI: Connecting to Contact Interrupts

The CTDI FORTRAN subroutine connects a task to contact interrupts and specifies a circular buffer to receive contact interrupt data. The length of this buffer can be computed by considering the following:

- Rate at which contact module interrupts occur
- Number of modules that can interrupt simultaneously
- Rate at which the circular buffer is emptied

The UDC11 driver generates a 5-word entry for each contact interrupt and the interface subroutine itself requires 10 words of additional storage. Thus the isz parameter, described below, can be computed as follows:

```
isz = (10 + 5 * n)
```

where n is the number of entries in the buffer and isz is expressed in words.

The call is issued as follows:

```
CALL CTDI (ibuf,isz,iev,[isb],[lun])
```

ibuf

An integer array that is to receive contact interrupt data.

isz

The length of the array in words, with a minimum size of 15.

iev

The trigger event flag number. The specified event flag is set whenever the driver inserts an entry in the data buffer.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number.

The isb array has the standard meaning defined in Section 16.5.2.

16.5.9 CTTI: Connecting to Timer Interrupts

The CTTI FORTRAN subroutine connects a task to timer interrupts and specifies a circular buffer to receive timer interrupt data. The length of this buffer can be computed by considering the following:

- Rate at which timer module interrupts occur
- Number of modules that can interrupt simultaneously
- Rate at which the circular buffer is emptied

The UDC11 driver generates a 4-word entry for each timer and the interface subroutine itself requires 8 words of additional storage. Thus the isz parameter, described below, can be computed as follows:

```
isz = (8 + 4 * n)
```

where n is the number of entries in the buffer and isz is expressed in words.

When a timer module interrupt occurs, the driver resets the count to an initial value, normally that specified in iv. The initial value for a specific module can be modified by calling the SCTI subroutine (see Section 16.5.19).

The call is issued as follows:

```
CALL CTTI (ibuf,isz,iev,iv,[isb],[lun])
```

ibuf

An integer array that is to receive timer interrupt data.

isz

The length of the array in words, with a minimum size of 12.

iev

A trigger event flag number. The specified event flag is set whenever the driver inserts an entry in the data buffer.

iv

An integer array that contains the initial timer module values, with one entry for each timer module, where entry n corresponds to timer module number n-1.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number.

The isb array has the standard meaning defined in Section 16.5.2.

16.5.10 DFDI: Disconnecting from Contact Interrupts

The DFDI FORTRAN subroutine disconnects a task from contact interrupts. It is issued as follows:

```
CALL DFDI ([isb],[lun])
```

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number.

The isb array has the standard meaning defined in Section 16.5.2.

16.5.11 DFTI: Disconnecting from Timer Interrupts

The DFTI FORTRAN subroutine disconnects a task from timer interrupts. It is issued as follows:

```
CALL DFTI ([isb],[lun])
```

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number.

The isb array has the standard meaning defined in Section 16.5.2.

16.5.12 DI/DIW: Reading Several Contact Sense Fields

The ISA standard DI/DIW FORTRAN subroutines read several 16-point contact sense fields. These calls are issued as follows:

```
CALL {DI  } (inm,icont,idata,isb,[lun])
     {DIW }
```

inm

The number of fields to be read.

icont

An integer array containing the initial point number of each field to be read.

idata

An integer array that is to receive the input data, 16 bits of contact data for each field read.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number (ignored if present).

The isb array has the standard meaning defined in Section 16.5.2.

16.5.13 DOL/DOLW: Latching or Unlatching Several Fields

The ISA standard DOL/DOLW FORTRAN subroutines latch or unlatch one or more 16-point fields. These calls are issued as follows:

CALL {DOL / DOLW} (inm,icont,idata,imsk,[isb],[lun])

inm

The number of fields to be latched or unlatched.

icont

An integer array containing the initial point number of each 16-point field.

idata

An integer array that specifies the points to be latched or unlatched; bit n of idata corresponds to point number icont + n; if the corresponding bit in imsk is set, the bit is changed; a bit value of 1 indicates latching, and 0 unlatching; each entry in the array specifies a string of 16 points.

imsk

An integer array in which bits are set to indicate points whose states are to be changed in the corresponding idata bits; each entry in the array specifies a 16-bit mask word.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number.

The isb array has the standard meaning defined in Section 16.5.2.

16.5.14 DOM/DOMW: Pulsing Several Fields

The ISA standard DOM/DOMW FORTRAN subroutines pulse several 16-bit fields (1-shot digital output points). These calls are issued as follows:

CALL {DOM / DOMW} (inm,icont,idata,[idx],[isb],[lun])

inm

The number of fields to be pulsed.

icont

An integer array containing the initial point number of each 16-point field.

idata

An integer array that specifies the points to be pulsed; bit n of idata corresponds to point number icont + n.

idx

A dummy argument retained for compatibility with existing Instrument Society of America standard FORTRAN process control calls.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number (ignored if present).

The isb array has the standard meaning defined in Section 16.5.2.

16.5.15 RCIPT: Reading a Contact Interrupt Point

The RCIPT FORTRAN subroutine reads the state of a single contact interrupt point. It is issued as follows:

CALL RCIPT (ipt,isb,[lun])

ipt

The number of the point to be read; points are numbered sequentially from 0, the first point on the first contact interrupt module.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number (ignored if present).

The isb array has the same basic meaning defined in Section 16.5.2. However, isb word 2 is set to one of the following values, representing the state of the point:

Setting	Meaning
.FALSE. (0)	Point is open
.TRUE. (-1)	Point is closed

NOTE

To increase throughput, the subroutines RDCS, RDDI, RDTI, and RDWD described in the following four sections do not issue the clear Event Flag directive until a buffer-empty condition is detected. The calling task, therefore, must avoid issuing a Wait-For directive until a buffer-empty is reported.

16.5.16 RDCS: Read Contact Interrupt Change-of-State Data From Circular Buffer

The RDCS FORTRAN subroutine reads contact interrupt data from a circular buffer that was specified in a CTDI call (see Section 16.5.8). It does no actual input or output, but rather performs a point-by-point scan of an interrupt entry in the buffer, returning the state of each point that has changed state as a logical value. The trigger event flag that was specified in the CTDI call is cleared when the "buffer empty" condition is detected.

On the initial call to RDCS, the module number, module data, and change-of-state word of the next interrupt entry are read from the circular buffer and stored for subsequent reference. The subroutine then searches the entry change-of-state word until a nonzero point is encountered. The point number is computed and returned to the caller along with the state of the point. Scanning for points that have changed state resumes on the next call; all other points are bypassed. The next entry is automatically read when the caller has received all change-of-state information from the current entry. If a valid entry is not found, ipt is set negative and ict (if specified) is either assigned a value of 0 or an overrun count maintained by the UDC11 driver. If ict is 0, no further entries remain. A nonzero value indicates that the driver received more data than could be stored in the buffer, and ict represents the number of entries that were discarded.

The RDCS call is issued as follows:

```
CALL RDCS (ipt,ival,[ict])
```

ipt

A variable to which the digital input point number is returned; it may be set as follows:

- ipt 0 if no valid entry is found (that is, no interrupt data currently in buffer, or overrun detected). One of the following values is returned to indicate the condition detected:

  -1 = Buffer empty
  -2 = Overrun detected

- ipt => 0 if the value indicated is a point number that has changed state; the state is returned to ival.

ival

A variable to which the state of the point is returned; it may be set as follows:

- .FALSE. (0) if the point is open
- .TRUE. (-1) if the point is closed

ict

An integer variable for receiving the overrun count. A nonzero positive count indicates that the driver was unable to store the number of interrupts indicated.

16.5.17 RDDI: Reading Contact Interrupt Data from a Circular Buffer

The RDDI FORTRAN subroutine reads contact interrupt data from a circular buffer that was specified in a CTDI call (see Section 16.5.8). It does no actual input or output, but rather performs a point-by-point scan of an interrupt entry in the buffer, returning the state of each point as a logical value. The trigger event flag that was specified in the CTDI call is also cleared.

On the initial call to RDDI, the module number and data of the next interrupt entry are read from the circular buffer and stored for subsequent reference. The subroutine then sets the current data bit number n to 0, examines the state of data bit n, and converts bit n to a point number by the following formula:

```
ipt = module number * 16 + n
```

On each subsequent call, n is incremented by one and then data bit n is examined in the stored module data. When n reaches 16, it is reset to 0 and an attempt is made to read the next interrupt entry from the circular buffer. If a valid entry is not found, ipt is set negative and ict (if specified) is either assigned a value of 0 or an overrun count maintained by the UDC11 driver. If ict is 0, no further entries remain. A nonzero value indicates that the driver received more data than could be stored in the buffer, and ict represents the number of entries that were discarded.

The RDDI call is issued as follows:

```
CALL RDDI (ipt,ival,[ict])
```

ipt

A variable to which the digital input point number is returned; it may be set as follows:

- ipt 0 if no valid entry is found (that is, no interrupt data currently in buffer, or buffer empty). One of the following values is returned to indicate the condition detected:

  -1=Buffer empty
  -2=Overrun detected

- ipt = 0 if the value indicated is a point number; the state is returned to ival

ival

A variable to which the state of the point is returned; it may be set as follows:

- .FALSE. (0) if the point is open
- .TRUE. (-1) if the point is closed

ict

A variable to which the overrun count may be returned; a nonzero positive count indicates that the driver was unable to store the number of entries indicated.

16.5.18 RDTI: Reading Timer Interrupt Data from a Circular Buffer

The RDTI FORTRAN subroutine reads timer interrupt data from a circular buffer that was specified in a CTTI call (see Section 16.5.9). It does no actual input or output, but rather performs a scan of each entry in the buffer, returning the timer value for each call. The trigger event flag that was specified in the CTTI call is also cleared.

When a timer module interrupt occurs, the UDC11 driver resets the count to an initial value, usually that specified in the iv array on the CTTI call. The initial value can be modified for a specific module by calling the subroutine SCTI (see Section 16.5.19).

The RDTI call is issued as follows:

```
CALL RDTI (imod,itm,[ivrn])
```

imod

A variable to which the module number is returned; it may be set as follows:

- imod 0 if no valid entry is found (that is, no interrupt data currently in buffer, or buffer empty). One of the following values is returned to indicate the condition detected:

  -1=Buffer empty
  -2=Overrun detected

- imod > 0 if the entry is valid, indicating a module number; the value of the timer module is returned in itm

itm

A variable to which the timer value is returned.

ivrn

A variable to which the overrun count may be returned; a nonzero positive count indicates that the driver was unable to store the number of values indicated.

16.5.19 RDWD: Read Full Word of Contact Interrupt Data From Circular Buffer

The RDWD FORTRAN subroutine reads a full word of contact interrupt and change-of-state data from the circular buffer that was specified in a CTDI call (see Section 16.5.8). It does no actual input or output, but rather performs a scan of each entry, returning the state of a module and, optionally, the change-of-state data for each call. The trigger event flag specified in the call to CTDI is cleared.

The call to RDWD is issued as follows:

```
CALL RDWD (imod,ist,[ivrn],[icos])
```

imod

A variable to which the module number is returned; it may be set as follows:

- imod 0 if no valid entry is found (that is, no interrupt data currently in buffer or overrun detected). One of the following values is returned to indicate the condition detected:

  -1=Buffer empty
  -2=Overrun detected

ist

A variable to which the module data is returned.

ivrn

A variable to which the overrun count may be returned; a nonzero, positive count indicates that the driver was unable to store the number of entries indicated.

icos

A variable to which the change-of-state data is returned. One bit is set for each point that has changed state in the direction indicated by the "point open" (POP) or "point closed" (PCL) jumpers on the module.

16.5.20 RSTI: Reading a Timer Module

The RSTI FORTRAN subroutine reads a single timer module. It is issued as follows:

```
CALL RSTI (imod,isb,[lun])
```

imod

The module number of the timer to be read.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number (ignored if present).

The isb array has the standard meaning defined in Section 16.5.2.

16.5.21 SCTI: Initializing a Timer Module

The SCTI FORTRAN subroutine sets a timer module to an initial value. It is issued as follows:

```
CALL SCTI (imod,ival,[isb],[lun])
```

imod

The module number of the timer to be set.

ival

The initial timer value.

isb

A 2-word integer array to which the subroutine status is returned.

lun

The logical unit number.

The isb array has the standard meaning defined in Section 16.5.2.

Calls to initialize a counter are valid only if the issuing task has connected a buffer for receiving counter interrupts by a call to CTTI.

16.6 STATUS RETURNS

Table 16-6 lists the error and status conditions that are returned by the UDC11 driver described in this chapter:

Table 16-6
UDC11 Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of samples completed or converted.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The specified I/O operation was canceled with IO.KIL while still in the I/O queue.
IE.BAD	**Bad parameter** An illegal specification was supplied for one or more of the device-dependent QIO parameters (words 6-11). For the UDC11, this code indicates an illegal channel number or gain code for the ADU01.

(continued on next page)

Table 16-6 (Cont.)
UDC11 Status Returns

Code	Reason
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a buffer but only word alignment is legal for the UDC12. Alternately, the length of a buffer was not an even number of bytes.
IE.CON	**Connect error** The task attempted to connect to contact or timer interrupts, but the interrupt was already connected to another task. Only one task can be connected to a timer or contact interrupt. Alternately a task that was not connected attempted to disconnect from contact or timer interrupts.
IE.DNR	**Device not ready** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. For the ADU01, this code is returned if an interrupt time out occurred or the power failed.
IE.IEF	**Invalid event flag number** An invalid trigger event flag number was specified in a connect function.
IE.IFC	**Illegal function** A function code was included in an I/O request that is illegal for the UDC11, or a request to initialize a counter (IO.ITI) was issued by a task that was not connected to receive counter interrupts. The function may also refer to a UDC11 feature that was not specified at system generation.
IE.MOD	**Invalid UDC11 module** On latching output, your task specified a starting point number that was not legal (defined at system generation) or was not evenly divisible by 16.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.

(continued on next page)

Table 16-6 (Cont.)
UDC11 Status Returns

Code	Reason
IE.PRI	**Privilege violation** The task that issued the request was not privileged to execute that request. For the UDC11, this code indicates that a checkpointable task attempted to connect to timer or contact interrupts.
IE.SPC	**Illegal address space** The specified control, data, or interrupt buffer was partially or totally outside the address space of the issuing task. Alternately, the interrupt buffer was too small for a single data entry (six words for timer interrupts and seven words for contact interrupts), or a byte count of 0 was specified.

FORTRAN interface values for these status returns are presented in Section 16.6.1.

16.6.1 FORTRAN Interface Values

The values listed in Table 16-7 are returned in FORTRAN subroutine calls.

Table 16-7
FORTRAN Interface Values

Status Return	FORTRAN Value
IS.SUC	+01
IS.PND	+00
IE.ABO	+315
IE.ADP	+101
IE.BAD	+301
IE.BYT	+319
IE.DAO	+313
IE.DNR	+303
IE.IEF	+100
IE.IFC	+302
IE.ILU	+99
IE.MOD	+321
IE.ONP	+305
IE.PRI	+316
IE.RSU	+317
IE.SDP	+102
IE.SPC	+306
IE.ULN	+08
IE.UPN	+04

16.7 PROGRAMMING HINTS

This section contains important information about programming the UDC11 driver described in this chapter.

16.7.1 Numbering Conventions

Numbering is relative. Module numbers start at 0, beginning with the first module of a given type.

Channel numbers also start at 0, with channel 0 as the first channel on the first module of a given type. For the ADU01, channel 8 is the first channel on the second analog output module.

Each IAD-IA module installed in an ICS11 subsystem occupies 120 channels (regardless of the number of multiplexers installed). In this case, channel 120 is the first channel on the second IAD-IA A/D converter.

Point numbers start at 0, with point 0 as the first point on the first module of a given type. For instance, point 20(8) is the first point of the second contact sense module (that is, relative module number 1).

16.7.2 Processing Circular Buffer Entries

Circular buffer entries should be processed in the following sequence.

1. Execute a WAITFOR system directive using the trigger event flag specified in the subroutine called to connect the circular buffer (CTTI or CTDI).

2. Repeatedly call the appropriate subroutine to read the circular buffer until all entries have been obtained and ipt indicates that the buffer is empty (-1).

3. Perform any other processing and return to step 1.

CHAPTER 17

LABORATORY PERIPHERAL SYSTEMS DRIVERS

17.1 INTRODUCTION

The LPS11 and AR11 Laboratory Peripheral Systems are modular, real-time subsystems that acquire or output laboratory analog data. (Laboratory Peripheral Systems drivers are not supported on RSX-11M-PLUS systems.) Table 17-1 compares the LPS11 with the AR11.

Table 17-1
Laboratory Peripheral Systems

	LPS11	AR11
Analog-to-Digital Conversion (with Sample and Hold Circuitry)	12 bits of precision 16-channel multiplexer with gain ranging Maximum of 64 channels without gain ranging	10 bits of precision 16-channel multiplexer without gain ranging
Programmable Real-Time Clock	Yes	Yes
Digital-to-Analog Output	12 bits of precision 10 channels (including display)	10 bits of precision 2 channels (including display)
Display Control	4096 by 4096 dot matrix	1024 by 1024 dot matrix
Digital I/O Option	16 digital points and programmable relays	16 digital points (available with DR11-K option)

At system generation, you can specify the following:

- The number of A/D channels
- The presence or absence of the gain-ranging option (LPSAM-SG) (LPS11 only) and the polarity of each channel (uni- or bipolar)
- The presence or absence of the external D/A option (LPSVC and LPSDA), and if present, the number of D/A channels
- The clock preset value

17.1.1 AR11 Laboratory Peripheral System

The AR11 is a 1-module, real-time analog subsystem that interfaces to the PDP-11 family of computers by a "hex" small peripheral controller slot. The system is a subset of the LPS11 and, as such, enjoys the same degree of flexibility. The AR11 includes a 16-channel, 10-bit A/D converter with sample-and-hold, a programmable real-time clock with one external input, and a display control with two 10-bit D/A converters.

17.1.2 LPS11 Laboratory Peripheral System

The LPS11 is a high-performance, modular, real-time subsystem with the flexibility of serving a variety of applications, including biomedical research, analytical instrumentation, data collection and reduction, monitoring, data logging, industrial testing, engineering, and technical education. The basic subsystem has an easy interface with external instrumentation and includes a 13-bit A/D converter, a programmable real-time clock, with two Schmitt triggers, a display controller with two 12-bit D/A converters, and a 16-bit digital I/O option. Up to nine different option types may be added to the basic package.

17.2 GET LUN INFORMATION MACRO

If a Get Lun Information system directive is issued for a LUN associated with a Laboratory Peripheral System, word 2 (the first characteristics word) contains all 0s words 3 and 4 are undefined, and word 5 contains a 16-bit buffer preset value that controls the rate of the real-time clock interrupts, as explained in Section 17.6.1.

17.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for the Laboratory Peripheral System drivers.

17.3.1 Standard QIO Function

Table 17-2 lists the standard function of the QIO macro that is valid for the Laboratory Peripheral Systems.

Table 17-2
Standard QIO Function for
Laboratory Peripheral Systems

Format	Function
QIO$C IO.KIL,...	Cancel I/O requests

IO.KIL cancels all queued and in-progress I/O requests.

17.3.2 Device-Specific QIO Functions (Immediate)

Except for IO.STP (see Section 17.3.4), all device-specific functions of the QIO macro that are valid for the Laboratory Peripheral Systems are either immediate or synchronous. Each immediate function performs a complete operation, whereas each synchronous function simply initiates an operation synchronized to the real-time clock. Table 17-3 lists the immediate functions.

Table 17-3
Device-Specific QIO Functions for the
Laboratory Peripheral Systems (Immediate)

Format	Function
QIO$C IO.LED,...,<int,num>	DISPLAY number in LED lights (LPS11 only)
QIO$C IO.REL,...,<rel,pol>	LATCH output relay (LPS11 only)
QIO$C IO.SDI,...,<mask>	READ digital input register
QIO$C IO.SDO,...,<mask,data>	WRITE digital output register

int

The 16-bit signed binary integer to display.

num

The LED digit number where the decimal point is to be placed.

rel

The relay number (0 or 1).

pol

The polarity (0 for open, nonzero for closed).

mask

The mask word.

data

The data word.

The following subsections describe the functions listed above.

17.3.2.1 **IO.LED - Display 16-bit Signed Integer** - This LPS11-only function displays a 16-bit signed binary integer in the light-emitting diode (LED) lights. The number is displayed with a leading blank (positive number) or minus sign (negative number), followed by five non-zero-suppressed decimal digits that represent the magnitude of the number. LED digits are numbered from right to left, starting at 1.

The number may be displayed with or without a decimal point. If the parameter num is a number from 1 to 5, then the corresponding LED digit is displayed with a decimal point to the right of the digit. If the LED digit number is not a number from 1 to 5, then no decimal point is displayed.

17.3.2.2 **IO.REL - Open or Close Relays** - This LPS11-only function opens or closes the programmable relays in the digital I/O status register. Approximately 300 milliseconds are required to open or close a relay. The driver imposes no delays when executing this function. Thus it is the responsibility of your task to ensure that adequate time has elapsed between the opening and closing of a relay.

17.3.2.3 **IO.SDI - Read Data from Digital Input Register** - This function reads data qualified by a mask word from the digital input register. The mask word contains a 1 in each bit position from which data is to be read. All other bits are zero filled and the resulting value is returned in the second I/O status word.

The operation performed is:

```
RETURN VALUE=MASK.AND.INPUT REGISTER
```

17.3.2.4 **IO.SDO - Write Data into Digital Output Register** - This function writes data qualified by a mask word into the digital output register. The mask word contains a 1 in each bit position that is to be written. The data word specifies the data to be written in corresponding bit positions.

The operation performed is:

```
NEW REGISTER=<MASK.AND.DATA>.OR.<<.NOT.MASK>.AND.OLD REGISTER>
```

17.3.3 Device-Specific QIO Functions (Synchronous)

Table 17-4 lists the synchronous, device-specific functions of the QIO macro that are valid for the Laboratory Peripheral Systems.

Table 17-4
Device-Specific QIO Functions for the
Laboratory Peripheral Systems (Synchronous)

Format	Function
QIO$C IO.ADS,...,<stadd,size,pnt, ticks,bufs,chna>	INITIATE A/D sampling
QIO$C IO.HIS,...,<stadd,size,pnt, ticks,bufs>	INITIATE histogram sampling (LPS11 only)
QIO$C IO.MDA,...,<stadd,size,pnt, ticks,bufs,chnd>	INITIATE D/A output
QIO$C IO.MDI,...,<stadd,size,pnt, ticks,bufs,mask>	INITIATE digital input sampling
QIO$C IO.MDO,...,<stadd,size,pnt, ticks,bufs,mask>	INITIATE digital output

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes (must be greater than 0 and a multiple of four bytes).

pnt

The digital point numbers (byte 0 - starting input/output point number; byte 1 - input point number to stop the function). Points are numbered from 0 to 15, allowing a maximum of 16 points to be specified.

ticks

The number of real-time clock ticks between samples or data transfers, as appropriate.

bufs

The number of data buffers to transfer.

chna

The analog-to-digital conversion specification. Byte 0 contains the starting channel number. For LPS11 this must be in the range of 0-63; for AR11 the range is 0-15. If the LPS11 gain-ranging option is present, the channel number must be in the range of 0-15, and bits 4 and 5 specify the gain code.

Byte 1 contains the number of consecutive analog-to-digital channels to sample. For LPS11 this must be in the range of 1-64; for AR11 or the LPS11 with gain-ranging, the range is 1-16.

chnd

The digital-to-analog output channel specification. Byte 0 contains the starting channel number. For LPS11 this must be in the range of 0-9; for the AR11 the range is 0-1.

Byte 1 contains the number of consecutive channels to be output. For LPS11 this must be in the range of 1-10; for AR11 the range is 1-2.

mask

The mask word.

The following subsections describe the functions listed above.

17.3.3.1 **IO.ADS - Read A/D Channels at Timed Intervals** - This function reads one or more A/D channels at precisely timed intervals, with or without auto gain-ranging. If you specify two or more channels, all are sampled at approximately the same time, once per interval.

Sampling may be started when the request is dequeued or when a specified digital input point is set. A digital output point may optionally be set when sampling is started. Sampling may be terminated by a program request (IO.STP or IO.KIL), by the clearing of a digital input point, or by the collection of a specified number of buffers of data.

All input is double buffered with respect to your task. Each time a half buffer of data has been collected, your task is notified (by the setting of an event flag) that data is available to be processed while the driver fills the other half of the buffer. If your task does not respond quickly enough, a data overrun may result. This occurs if the driver attempts to put another data item in the task's buffer when no space is available.

The subfunction modifier bits are identical to those described in Section 17.3.3.2. In addition, setting bit 3 to a 1 means LPS11 auto gain-ranging is requested. Bit 3 is ignored for the AR11. If bits 7 and 6 are both set to 1, the digital input point and digital output point number are assumed to be the same.

If your task uses LPS11 auto gain-ranging, the LPSAM-SG hardware option must be present and specified at system generation. The auto gain-ranging algorithm causes a channel to be sampled at the highest gain at which saturation does not occur. If the gain-ranging option is present and auto gain-ranging is not specified in bit 3 of the subfunction code, then bits 4 and 5 of the starting channel number specify the gain at which samples are to be converted. Gain codes are as follows:

Code	Gain
00	1
01	4
10	16
11	64

Data words written into your task's buffer contain the converted value in bits 0-11 and the gain code, as shown below, in bits 12-15:

Code	Gain
0000	1
0001	4
0010	16
0011	64

If the LPSAM-SG option is present, then each channel must have been defined as uni- or bipolar at system generation. In addition, if bandwidth filtering is enabled (and so indicated at system generation time), a software delay is imposed by the driver when the multiplexer channel is changed. This delay must have been specified at system generation. See the LPS11 Laboratory Peripheral System User's Guide.

The AR11 always returns data that is equivalent to an LPS11 gain of 1. Channel polarity must always be specified for the AR11 at system generation, because this operation is software selectable at the time sampling is initiated.

17.3.3.2 **IO.HIS - Measure Elapsed Time Between Events** - This LPS11-only function measures the elapsed time between a series of events by means of Schmitt trigger 1. Each time a sample is to be taken, a counter is incremented and Schmitt trigger 1 is tested. If it has fired, the counter is written into your task's buffer and reset to 0. Thus, the data item returned to your task is the number of sample intervals between Schmitt trigger firings.

If the counter overflows before Schmitt trigger 1 fires, then a 0 value is written into your task's buffer. Sampling may be started and stopped as described in Section 17.3.3.1. All input is double buffered with respect to your task.

The subfunction modifier bits appear below. A setting of 1 indicates the action listed in the right-hand column.

Bit	Meaning
0-3	Unused
4	Stop on number of buffers
5	Stop on digital input point clear
6	Set digital output point at start of operation
7	Start on digital input point set (a 0 specification means start immediately). Points are numbered from 0 to 15, allowing a maximum of 16 points to be specified.

17.3.3.3 **IO.MDA - Write Data to D/A Converter at Timed Intervals** - This function writes data into one or more external D/A converters at precisely timed intervals. If two or more channels are specified, all are written at approximately the same time, once per interval. Output may be started or stopped as described in Section 17.3.3.1. All output is double buffered with respect to your task.

D/A converters 0 and 1 correspond to the X and Y registers of the display control. Note that there are no specific driver functions to set the display status register. This is reserved for your task. D/A converters 2 through 9 correspond to the LPS11, LPSDA external D/A option.

The subfunction modifier bits are identical to those described in Section 17.3.3.2.

17.3.3.4 **IO.MDI - Read Data from Input Register at Timed Intervals** - This function provides the capability to read data qualified by a mask word from the digital input register at precisely timed intervals. Sampling may be started and stopped as described in Section 17.3.3.1. All input is double buffered with respect to your task.

The mask word contains a 1 in each bit position from which data is to be read. All other bits are 0.

The subfunction modifier bits are identical to those described in Section 17.3.3.2.

17.3.3.5 **IO.MDO - Write Data into Output Register at Timed Intervals** - This function writes data qualified by a mask word into the digital output register at precisely timed intervals. Output may be started and stopped as described in Section 17.3.3.1. All output is double buffered with respect to your task.

The subfunction modifier bits are identical to those described in Section 17.3.3.2.

17.3.4 Device-Specific QIO Function (IO.STP)

Table 17-5 lists the device-specific function of the QIO macro, which is valid for the Laboratory Peripheral Systems.

Table 17-5
Device-Specific QIO Function for the
Laboratory Peripheral Systems (IO.STP)

Format	Function
QIO$C IO.STP,...,<STADD>	STOP in-progress request

stadd

The buffer address of the function to stop (must be the same as the address specified in the initiating request).

17.3.4.1 IO.STP - Stop In-Progress Synchronous Request - IO.STP stops a single, in-progress synchronous request. It is unlike IO.KIL in that it cancels only the specified request, whereas IO.KIL cancels all requests.

17.4 FORTRAN INTERFACE

A collection of FORTRAN-callable subroutines provide FORTRAN programs access to the Laboratory Peripheral Systems. These routines are described in this section.

Some of these routines may be called from FORTRAN as either subroutines or functions. All are reentrant and may be placed in a resident library.

17.4.1 The isb Status Array

The isb (I/O status block) parameter is a 2-word integer array that contains the status of the FORTRAN call, in accordance with ISA convention. This array serves two purposes:

1. It is the 2-word I/O status block to which the driver returns an I/O status code on completion of an I/O operation.

2. The first word of the isb receives a status code from the FORTRAN interface in ISA-compatible format, with the exception of the I/O pending condition, which is indicated by a status of 0. The ISA standard code for this condition is +2.

The meaning of its contents varies, depending on the FORTRAN call that has been executed, but Table 17-6 lists certain general principles that apply. The sections describing individual subroutines provide more details.

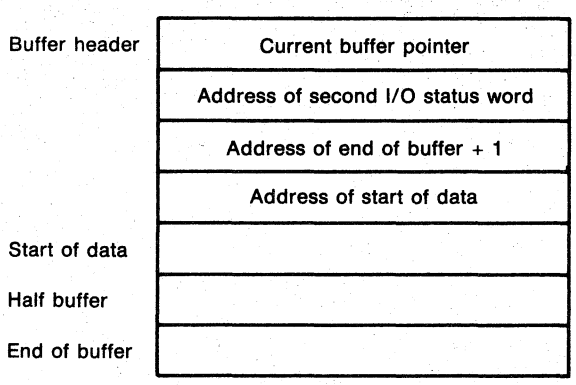
Table 17-6
Contents of First Word of isb

Contents	Meaning
isb(1) = 0	Operation pending; I/O in progress
isb(1) = 1	Successful completion
isb(1) = 3	Interface subroutine unable to generate QIO directive, or illegal time or buffer value
3 < = isb(1) < 300	QIO directive rejected and actual error code = -(isb(1) - 3)
isb(1) > 300	Driver rejected request and actual error code = -(isb(1) - 300)

FORTRAN interface routines depend on asynchronous system traps to set their status. Thus, if the trap mechanism is disabled, proper status cannot be set.

17.4.2 Synchronous Subroutines

RTS, DRS, HIST (LPS11 only), SDO, and SDAC are FORTRAN subroutines that initiate synchronous functions. When you use the appropriate Laboratory Peripheral System driver and the FORTRAN program, they communicate by means of a caller-specified data buffer of the following format:



ZK-008-81

The buffer header is initialized when the synchronous function initiation routine is called. The length of the buffer must be even and greater than or equal to 6. An even length is required so that the buffer is exactly divisible into half buffers.

The drivers perform double buffering within the half buffers. Each time a driver fills or empties a half buffer, it sets a event flag that you specified to notify your task that more data is available or needed. Your task responds by putting more data into the buffer or by removing the data now available.

If your task does not respond quickly enough, a data overrun may result. This occurs if the driver attempts to put another data item in your task's buffer when no space is available (that is, the buffer is full of data), or if the driver attempts to obtain the next data item from your task's buffer when none is available (that is, the buffer is empty).

All synchronous functions can be initiated immediately or when a specified digital input point is set (that is, a start button is pushed).

They can be terminated by any combination of a program request, the processing of the required number of full buffers of data, or the clearing of a specified digital input point (that is, a stop button is pushed). A digital output point may also optionally be set at the start of a synchronous function. You could use this, for example, as a signal to start a test instrument.

17.4.3 FORTRAN Subroutine Summary

Table 17-7 lists the FORTRAN interface subroutines supported for the Laboratory Peripheral Systems under RSX-11M. S and F indicate whether they can be called as subroutines or functions, respectively.

Table 17-7
FORTRAN Interface Subroutines for Laboratory Peripheral Systems

Subroutine	Function
ADC	Read a single A/D channel (F,S)
ADJLPS	Adjust buffer pointers (S)
ASARLN	Assign a LUN to AR0: (S)
ASLSLN	Assign a LUN to LS0: (S)
CVSWG	Convert a switch gain A/D value to floating point (F)
DRS	Initiate synchronous digital input sampling (S)
HIST	Initiate histogram sampling (S) (LPS11 only)
IDIR	Read digital input (F,S)
IDOR	Write digital output (F,S)
IRDB	Read data from a synchronous function input buffer (F,S)
LED	Display number in LED lights (S) (LPS11 only)
LPSTP	Stop an in-progress synchronous function (S)
PUTD	Put data into a synchronous function output buffer (S)
RELAY	Latch an output relay (S) (LPS11 only)
RTS	Initiate synchronous A/D sampling (S)
SDAC	Initiate synchronous D/A output (S)
SDO	Initiate synchronous digital output (S)

The following subsections briefly describe the function and format of each FORTRAN subroutine call.

17.4.4 ADC: Read a Single A/D Channel

The ADC FORTRAN subroutine or function reads a single converted value from an A/D channel. If the gain-ranging option is present in the LPS11 hardware, the channel may be converted at a specific gain or the driver can select the best gain (the gain providing the most significance). The converted value is returned as a normalized floating-point number. The call is issued as follows:

```
CALL ADC (ichan,[var],[igain],[isb])
```

ichan

The A/D channel to be converted.

var

A floating-point variable that receives the converted value in floating-point format.

igain

The gain at which the specified A/D channel is to be converted. The default is 1. If specified, igain may have the following values:

igain	Gain
0	Auto gain-ranging (driver selects gain that provides most significance)
1	1
2	4
3	16
4	64

The AR11 driver always uses a gain of 1.

isb

A 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in Section 17.4.1.

When your task uses the function form of the call, the value of the function is the same as that returned in var. If this value is negative, an error has occurred during the A/D conversion (see Section 17.5.3). Otherwise, this value is a floating-point number calculated from the following formula:

```
var = (64 * converted value) / conversion gain
```

17.4.5 ADJLPS: Adjust Buffer Pointers

The ADJLPS FORTRAN subroutine adjusts buffer pointers for a buffer that a laboratory peripheral system driver is either synchronously filling or emptying. Your task should call it when your task uses indexing for direct access to the data in a buffer.

When data in a buffer is to be processed only once, your task may use the IRDB and PUTD routines. In some cases, however, it is useful to leave data in the buffer until processing is complete. Your task can process the data directly, and then call ADJLPS to free half the buffer. Using the routine for synchronous output functions is quite similar. When a half buffer of data is ready for output, ADJLPS is called to make the half buffer available.

When your task uses ADJLPS for either input or output, care must be taken to insure that the program stays in sync with the driver. If the program loses its position with respect to the driver, the function must be stopped and restarted. An attempt to over-adjust causes a 3 to be returned in isb(1) and no adjustment to take place.

The call is issued as follows:

```
CALL ADJLPS (ibuf,iadj,[isb])
```

ibuf

An integer array that was previously specified in a synchronous input or output function.

iadj

The adjustment to be applied to the buffer pointers. For an input function, this specifies the number of data values that have been removed from the data buffer. For an output function, this specifies the number of data values that have been put into the data buffer.

isb

A 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in Section 17.4.1.

17.4.6 ASLSLN: Assign a LUN to LS0:

The ASLSLN FORTRAN subroutine assigns a logical unit number (LUN) to the LPS11. It must be called prior to executing any other Laboratory Peripheral Systems FORTRAN function or subroutine. Subsequent calls to other interface routines then implicitly reference the LPS11 with the LUN assigned.

The call is issued as follows:

```
CALL ASLSLN (lun,[isb],[iun])
```

lun

The number of the LUN to be assigned to LS0:

isb

A 2-word integer array to which the subroutine status is returned.

iun

The unit number of the device to be assigned (defaults to 0 if not specified).

The isb array has the standard meaning described in Section 17.4.1.

17.4.7 ASARLN: Assign a LUN to AR0:

The ASARLN FORTRAN subroutine assigns a logical unit number (LUN) to the AR11. It must be called prior to executing any other laboratory peripheral system FORTRAN function or subroutine. Subsequent calls to other interface routines then implicitly reference the AR11 with the LUN assigned.

The call is issued as follows:

```
CALL ASARLN (lun,[isb],[iun])
```

lun

The number of the LUN to be assigned to AR0:.

isb

A 2-word integer array to which the subroutine status is returned.

iun

The unit number of the device to be assigned (defaults to 0 if not specified).

The isb array has the standard meaning described in Section 17.4.1.

17.4.8 CVSWG: Convert a Switch Gain A/D Value to Floating-Point

The CVSWG FORTRAN subroutine converts an A/D value from a synchronous A/D sampling function to a floating-point number. Each data item returned by a laboratory peripheral system driver consists of a gain code and converted value packed in a single word (see Section 17.3.3.1). This form is not readily usable by FORTRAN, but is much more efficient than converting each value to floating point in the driver. This routine unpacks the gain code and value, and then converts the result to a floating-point number. Your task can use it with the IRDB routine (see Section 17.4.13).

The call is issued as follows:

```
CVSWG (ival)
```

ival

The value to be converted to floating point. Its format must be that returned by a synchronous A/D sampling function. The conversion is performed according to the following formula:

```
var = (64 * converted value)/conversion  gain
```

For the various gain codes,

```
var = x * converted value
```

as shown below:

Gain	x
1	64
4	16
16	4
64	1

17.4.9 DRS: Initiate Synchronous Digital Input Sampling

The DRS FORTRAN subroutine reads data qualified by a mask word from the digital input register at precisely timed intervals. Sampling may be started or stopped as for RTS (see Section 17.4.18) and all input is double buffered with respect to your task. Data may be sequentially retrieved from the data buffer by the IRDB routine (see Section 17.4.13), or the ADJLPS routine (see Section 17.4.5). You may use either for direct access to the input data. The call is issued as follows:

```
CALL DRS (ibuf,ilen,imode,irate,iefn,imask,isb,[nbuf],
          [istart],[istop])
```

ibuf

An integer array that is to receive the input data values.

ilen

The length of ibuf (must be even and greater than or equal to 6).

imode

The start, stop, and sampling mode. Its value is encoded by adding together the appropriate function selection values shown below.

Function Selection Value	Meaning
128	Start on digital input point set
64	Set digital output point at start
32	Stop on digital input point clear
16	Stop on number of buffers

Thus, a value of 192 for imode specifies:

- The sampling is to start when a specified digital input point is set.
- A digital output point is to be set when sampling is started.
- Sampling is stopped by a program request.

irate

A 2-word integer array that specifies the time interval between digital input samples. The first word specifies the interval units as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit unsigned integer.

iefn

The number of the event flag that is to be set each time a half buffer of data has been collected.

imask

The digital input points to be read.

isb

A 2-word integer array to which the subroutine status is returned.

nbuf

The number of buffers of data to be collected. It is needed only if a function selection value of 16 has been added into imode.

istart

The digital input pointer number to use to trigger sampling, or the digital output point number to set when sampling is started, or both. It is needed only if a function selection value of 128 or 64 has been added into imode.

istop

The digital input point number to use to stop sampling. It is needed only if a function selection value of 32 has been added into imode.

When sampling is in progress, the first word of the isb array is zero and the second word contains the number of data values currently in the buffer.

17.4.10 HIST: Initiate Histogram Sampling (LPS11 only)

The HIST FORTRAN subroutine measures the elapsed time between a series of events with Schmitt trigger 1.

Each time a sample is to be taken, a counter is incremented and Schmitt trigger 1 is tested. If it has fired, then the counter is written into your task's buffer and the counter is reset to 0. Thus the data returned to your task is the number of sample intervals between Schmitt trigger firings. If the counter overflows before Schmitt trigger 1 fires, a 0 value is written into your task's buffer. Sampling may be started and stopped as for RTS (see Section 17.4.18) and all input is double buffered with respect to your task. The call is issued as follows:

```
CALL HIST (ibuf,ilen,imode,irate,iefn,isb,[nbuf],
           [istart],[istop])
```

ibuf

An integer array that is to receive the input data values.

ilen

The length of ibuf (must be even and greater than or equal to 6).

imode

The start, stop, and sampling mode. Its value is encoded by adding the appropriate function selection values shown below:

Function Selection Value	Meaning
128	Start of digital input point set
64	Set digital output point at start
32	Stop on digital input point clear
16	Stop on number of buffers

irate

A 2-word integer array that specifies the time interval between samples. The first word specifies the interval units as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit signed integer.

iefn

The number of the event flag that is to be set each time a half buffer of data has been collected.

isb

A 2-word integer array to which the subroutine status is returned.

nbuf

The number of buffers of data to be collected. It is needed only if a function selection value of 16 has been added into imode.

istart

The digital input point number to use to trigger sampling or the digital output point number to set when sampling is started, or both. It is needed only if a function selection value of 128 or 64 has been added into imode.

istop

The digital input point number to use to stop sampling. It is needed only if a function selection value of 32 has been added into imode.

The isb array has the standard meaning described in Section 17.4.1.

When sampling is in progress, the first word of the isb array is 0 and the second word contains the number of data values currently in the buffer.

17.4.11 IDIR: Read Digital Input

The IDIR FORTRAN subroutine or function reads the digital input register as an unsigned binary integer, or as four binary-coded decimal (BCD) digits. In the latter case, the BCD digits are converted to a binary integer before the value is returned to the caller. The call is issued as follows:

```
CALL IDIR (imode,[ival],[isb])
```

imode

The mode in which the digital input register is to be read. If imode equals 0, then the digital input register is read as four BCD digits and converted to a binary integer. Otherwise, it is read as a 16-bit unsigned binary integer.

ival

A variable that receives the value read.

isb

A 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in Section 17.4.1.

When your task uses the function form of the call, the value of the function is the same as that returned in ival.

17.4.12 IDOR: Write Digital Output

The IDOR FORTRAN subroutine or function clears or sets bits in the digital output register. The caller provides a mask word and output mode. Bits in the digital output registers corresponding to the bits specified in the mask word are either set or cleared according to the specified mode. The call is issued as follows:

```
CALL IDOR (imode,imask,[newval],[isb])
```

imode

Whether the bits specified by imask are to be cleared or set in the digital output register. If imode equals 0, then the bits are to be cleared. Otherwise, they are to be set.

imask

The bits to be cleared or set in the digital output register. It may be conveniently specified as an octal constant.

newval

A variable that receives the updated (actual) value written into the digital output register.

isb

A 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in Section 17.4.1.

When your task uses the function form of the call, the value of the function is the same as that returned in newval.

17.4.13 IRDB: Read Data from an Input Buffer

The IRDB FORTRAN subroutine or function retrieves data sequentially from a buffer that a laboratory peripheral system driver is synchronously filling. If no data is available when the call is executed, the contents of the next location in the data buffer are returned without updating the buffer pointers. The call is issued as follows:

```
CALL IRDB (ibuf,[ival])
```

ibuf

An integer array that was previously specified in a synchronous input sampling request (that is, DRS, HIST, or RTS).

ival

A variable that receives the next value in the data buffer.

When your task uses the function form of the call, the value of the function is the same as that returned in ival.

17.4.14 LED: Display in LED Lights (LPS11 only)

The LED FORTRAN subroutine displays a 16-bit signed binary integer in the LED lights. The number is displayed with a leading blank (positive number) or minus (negative number), followed by five non-zero-suppressed decimal digits that represent the magnitude of the number. LED digits are numbered right to left starting at 1 and continuing to 5. The number may be displayed with or without a decimal point. The call is issued as follows:

```
CALL LED (ival,[idec],[isb])
```

ival

The variable whose value is to be displayed.

idec

The position of the decimal point. A value of 1 to 5 specifies that a decimal point is to be displayed. All other values specify that no decimal point is to be displayed.

isb

A 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in Section 17.4.1.

For example, the following call:

```
CALL LED (-55,2)
```

would cause -0005.5 to be displayed in the LED lights.

17.4.15 LPSTP: Stop an In-Progress Synchronous Function

The LPSTP FORTRAN subroutine selectively stops a single synchronous request. The call is issued as follows:

```
CALL LPSTP (ibuf)
```

ibuf

An integer array that specifies a buffer that was previously specified in a synchronous initiation request.

17.4.16 PUTD: Put Data into an Output Buffer

The PUTD FORTRAN subroutine puts data sequentially into a buffer that a laboratory peripheral system driver is synchronously emptying. If no free space is available, no operation is performed. The call is issued as follows:

```
CALL PUTD (ibuf,ival)
```

ibuf

An integer array that was previously specified in a synchronous output request (SDO or SDAC).

ival

A variable whose value is to be placed in the next free location in the data buffer.

17.4.17 RELAY: Latching an Output Relay (LPS11 only)

The RELAY FORTRAN subroutine opens or closes the LPS11 relays. The call is issued as follows:

```
CALL RELAY (irel,istate,[isb])
```

irel

Which relay is to be opened or closed (0 for relay one, 1 for relay two).

istate

Whether the relay is to be opened or closed. If istate equals 0, the relay is to be opened. Otherwise, it is to be closed.

isb

A 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in Section 17.4.1.

17.4.18 RTS: Initiating Synchronous A/D Sampling

The RTS FORTRAN subroutine reads one or more A/D channels at precisely timed intervals, with or without auto gain-ranging. The auto gain-ranging algorithm (LPS11 only) causes the channels to be sampled at the highest gain at which saturation does not occur.

Sampling can be started when the interface subroutine is called or when a specified digital input point is set. A digital output point can optionally be set when sampling is started. Sampling can be terminated by a program request (stop-in-progress request or kill I/O), the clearing of a digital input point, or the collection of a specified number of buffers of data.

All input is double buffered with respect to your task. Each time a half buffer of data has been collected, your task is notified (by the setting of an event flag) that data is available to be processed while the driver fills the other half of the buffer. Data can be sequentially retrieved from the data buffer with the IRDB routine (see Section 17.4.13), or the ADJLPS routine (see Section 17.4.5). Your task can use for direct access to the input data. The call is issued as follows:

```
CALL RTS (ibuf,ilen,imode,irate,iefn,ichan,nchan,isb,
          [nbuf],[istart],[istop])
```

ibuf

An integer array that is to receive the converted data values.

ilen

The length of ibuf (must be even and greater than or equal to 6).

imode

The start, stop, and sampling mode. Its value is encoded by adding together the appropriate function selection values as shown below:

Function Selection Value	Meaning
128	Start on digital input point set
64	Set digital output point at start
32	Stop on digital input point clear
16	Stop on number of buffers
8	Auto gain-ranging (LPS11 only)

irate

A 2-word integer array that specifies the time interval between A/D samples. The first word specifies the interval unit as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit unsigned integer.

iefn

The number of the event flag that is to be set each time a half buffer of data has been collected.

ichan

The starting A/D channel of the block of channels to be sampled synchronously (must be between 0 and 63 for LPS11 and between 0 and 15 for AR11).

nchan

The number of A/D channels to be sampled (must be between 1 and 64 for LPS11 and between 1 and 16 for AR11).

isb

A 2-word integer array to which the subroutine status is returned.

nbuf

The number of buffers of data that are to be collected. It is needed only if a function selection value of 16 has been added into imode.

istart

The digital input point number that triggers sampling or the digital output point number set when sampling is started. It is needed only if a function selection value of 128 or 64 has been added into imode. Points are numbered from 0 to 15, allowing a maximum of 16 points to be specified.

istop

The digital input point number that stops sampling. It is needed only if a function selection value of 32 has been added into imode.

The values listed for ichan and nchan above are the maximum allowable for each of the devices. In practice, they are constrained by the number of channels available as specified during system generation.

The isb parameter has the standard meaning described in Section 17.4.1.

When sampling is in progress, the first word of the isb array is 0 and the second word contains the number of data values currently in the buffer.

17.4.19 SDAC: Initiating Synchronous D/A Output

The SDAC FORTRAN subroutine writes data into one or more external D/A converters at precisely timed intervals. Output may be started and stopped as for RTS (see Section 17.4.18), and all input is double buffered with respect to your task. One full buffer of data must be available when synchronous output is initiated.

After SDAC has initiated output and the specified event flag has been set to notify the task that free buffer space is available, your task can use the PUTD routine (see Section 17.4.16) to put data values sequentially into the output data buffer. Your task can use the ADJLPS routine (see Section 17.4.5) for direct access to the output data buffer. Your task issues the SDAC call as follows:

```
CALL SDAC (ibuf,ilen,imode,irate,iefn,ichan,nchan,isb,
           [nbuf],[istart],[istop])
```

ibuf

An integer array that contains the output data values.

ilen

The length of ibuf (must be even and greater than or equal to 6).

imode

The start, stop, and sampling mode. Its value is encoded by adding together the appropriate function selection values as shown below:

Function Selection Value	Meaning
128	Start on digital input point set
64	Set digital output point at start
32	Stop on digital input point clear
16	Stop on number of buffers

irate

A 2-word integer array that specifies the time interval between D/A outputs. The first word specifies the interval units as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit unsigned integer.

iefn

The number of the event flag that is to be set each time a half buffer of data has been output.

ichan

The starting D/A channel of the block of channels to be written into synchronously (must be between 0 and 9 for LPS11, and be 0 or 1 for AR11).

nchan

The number of D/A channels to be written into (must be between 1 and 10 for LPS11, and be 1 or 2 for AR11).

isb

A 2-word integer array to which the subroutine status is returned.

nbuf

The number of buffers of data to be output. It is needed only if a function selection value of 16 has been added into imode.

istart

The digital input point number that triggers sampling or the digital output point number that must be set when your task starts sampling. It is needed only if a function selection value of 128 or 64 has been added into imode. Points are numbered from 0 to 15, allowing you to specify a maximum of 16 points.

istop

The digital input point number that stops sampling. It is needed only if you added a function selection value of 32 into imode.

The isb array has the standard meaning described in Section 17.4.1.

When sampling is in progress, the first word of the isb array is 0 and the second word contains the number of free positions in the buffer.

17.4.20 SDO: Initiating Synchronous Digital Output

The SDO FORTRAN subroutine writes data qualified by a mask word into the digital output register at precisely timed intervals. Sampling may be started and stopped as for RTS (see Section 17.4.18) and all input is double buffered with respect to your task. One full buffer of data must be available when output is initiated.

After SDO initiated output, and your task set the specified event flag to notify the task that free buffer space is available, you may use the PUTD routine (see Section 17.4.16) to put data values sequentially into the output data buffer. You may use the ADJLPS routine (see Section 17.4.5) for direct access to the output data buffer. Issue the SDO call as follows:

```
CALL SDO (ibuf,ilen,imode,irate,iefn,imask,isb,[nbuf],
          [istart],[istop])
```

ibuf

An integer array that contains the digital output values.

ilen

The length of ibuf (must be even and greater than or equal to 6).

imode

The start, stop, and sampling mode. Encode its value by adding together the appropriate function selection values as shown below:

Function Selection Value	Meaning
128	Start on digital input point set
64	Set digital output point at start
32	Stop on digital input point clear
16	Stop on number of buffers

irate

A 2-word integer array that specifies the time interval between digital outputs. The first word specifies the interval units as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit unsigned integer.

iefn

The number of the event flag that is to be set each time a half buffer of data has been output.

imask

The digital output points that are to be written. It may be conveniently specified as an octal constant.

isb

A 2-word integer array to which the subroutine status is returned.

nbuf

The number of buffers of data to be output. It is needed only if a function selection value of 16 has been added into imode.

istart

The digital input point number that triggers sampling or the digital output point number set to start sampling. You need it only if a function selection value of 128 or 64 was added into imode. Points are numbered 0 through 15, allowing a maximum of 16 points to be specified.

istop

The digital input point number that stops sampling. You need it if a function selection value of 32 has been added into imode.

The isb parameter has the standard meaning described in Section 17.4.1.

When sampling is in progress, the first word of the isb array is 0 and the second word contains the number of free positions in the buffer.

17.5 STATUS RETURNS

The error and status conditions listed in Table 17-8 are returned by the Laboratory Peripheral System drivers described in this chapter.

Table 17-8
Laboratory Peripheral Systems Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of data values processed.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been completed.
IE.ABO	**Operation aborted** The specified I/O operation was canceled (by IO.KIL or IO.STP) while in progress.
IE.BAD	**Bad parameter** An illegal specification was supplied for one or more of the device-dependent QIO parameters (words 6-11). The second I/O status word is filled with 0s.
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a data buffer but only word alignment is legal for Laboratory Peripheral Systems. Alternatively, the length of a buffer is not an even number of bytes.
IE.DAO	**Data overrun** For Laboratory Peripheral Systems, the driver attempted to get a value from your task's buffer when none was available or attempted to put a value in your task's buffer when no space was available.
IE.DNR	**Device not ready** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. For Laboratory Peripheral Systems, this code is returned if a device time-out occurs while a function is in progress. The second I/O status word contains the number of free positions in the buffer, as appropriate.

(continued on next page)

Table 17-8 (Cont.)
Laboratory Peripheral Systems Status Returns

Code	Reason
IE.IEF	**Invalid event flag number** An invalid event flag number was specified in a synchronous function.
IE.IFC	**Illegal function** A function code was included in an I/O request that is illegal for the LPS11 or AR11.
IE.NOD	**Insufficient buffer space** Dynamic storage space has been depleted, and there is insufficient buffer space available to allocate a secondary control block for a synchronous function.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.ONP	**Option not present** An option dependent function or subfunction was requested, and the required feature was not specified at system generation. For example the gain-ranging option or D/A option is not present. The second I/O status word contains 0.
IE.PRI	**Privilege violation** The task that issued the request was not privileged to execute that request. For Laboratory Peripheral Systems, a checkpointable task attempted to execute a synchronous sampling function.
IE.RSU	**Resource in use** A resource needed by the function requested in the QIO directive was being used (see Section 17.5.1).
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately a byte count of 0 was specified. The second I/O status word contains 0.

FORTRAN interface values for these status returns are presented in Section 17.5.4.

17.5.1 IE.RSU

IE.RSU is returned if a function requests a resource that is currently being used. Your task may repeat the request at a later time or take any alternative action required.

Because certain functions do not need such resources, they never cause this code to be returned. Other functions return this code under the following conditions:

Function	When IE.RSU Is Returned
IO.SDO	One or more specified digital output bits are in use.
IO.ADS	Digital output point (if specified) is in use.
IO.HIS	Digital output point (if specified) is in use.
IO.MDA	Digital output point (if specified) is in use.
IO.MDI	Digital output point (if specified) or digital input points to be sampled are in use.
IO.MDO	Digital output point (if specified) or output bits to be written are in use.

The following components of the Laboratory Peripheral Systems are each considered a single resource:

Resource	When Shareable
The A/D Converter and clock	Always shareable
Each bit in the digital output register	Never shareable.
Each bit in the digital input register	Always shareable when used by IO.SDI or for start/stop conditions (specified in subfunction modifier bits), even when in use by another function; when specified by a synchronous digital input function, not shareable with another such function

Each resource is allocated on a first-come-first-served basis. (That is, when a conflict arises, the most recent request is rejected with a status of IE.RSU).

17.5.2 Second I/O Status Word

On successful completion of a function specified in a QIO macro call, the IS.SUC code is returned to the first word of the I/O status block.

Table 17-9 lists the contents of the second word of the status block, on successful completion for each function.

Table 17-9
Returns to Second Word of I/O Status Block

Successful Function	Contents of Second Word
IO.KIL	Number of data values before I/O was canceled
IO.LED	0
IO.REL	0
IO.SDI	Masked value read from digital input register
IO.SDO	Updated value written into digital output register
IO.ADS	Number of data values remaining in buffer
IO.HIS	Number of data values remaining in buffer
IO.MDA	Number of free positions in buffer
IO.MDI	Number of data values remaining in buffer
IO.MDO	Number of free positions in buffer
IO.STP	0

When IE.BAD is returned, the second I/O status word contains 0/ Laboratory Peripheral Systems drivers return the IE.BAD code under the following conditions:

Function	When IE.BAD is Returned
IO.REL	Relay number not 0 or 1
IO.ADS IO.MDA	No I/O status block, illegal digital I/O point number, or illegal channel number
IO.HIS IO.MDI IO.MDO	No I/O status block or illegal digital I/O point number

17.5.3 IO.ADS and ADC Errors

While IO.ADS or the ADC FORTRAN subroutine is converting a sample, two error conditions may arise. Both of these conditions are reported to your task by placing illegal values in the data buffer. A -1 (177777(octal) is placed in the buffer if an A/D conversion does not complete within 30 microseconds. A -2 (177776(octal) is placed in the buffer if an error occurs during an A/D conversion (LPS11 only).

17.5.4 FORTRAN Interface Values

The values listed in Table 17-10 are returned in FORTRAN subroutine calls.

Table 17-10
FORTRAN Interface Values

Status Return	FORTRAN Value
IS.SUC	+01
IS.PND	+00
IE.ABO	+315
IE.ADP	+101
IE.ALN	+334
IE.BAD	+301
IE.BYT	+319
IE.DAO	+313
IE.DNR	+303
IE.IEF	+100
IE.IFC	+302
IE.ILU	+99
IE.NOD	+323
IE.OFL	+365
IE.ONP	+305
IE.PRI	+316
IE.RSU	+317
IE.SDP	+102
IE.SPC	+306
IE.ULN	+08
IE.UPN	+04

17.6 PROGRAMMING HINTS

This section contains important information about programming the Laboratory Peripheral Systems drivers described in this chapter.

17.6.1 The LPS11/AR11 Clock and Sampling Rates

The basic real-time clock frequency (count rate) for all synchronous functions is always 10KHz. Device characteristics word 4 contains a 16-bit buffer preset value -- set dynamically or at system generation, that controls the rate of "ticks" (that is, the rate at which the clock interrupts). The quotient that results when this value is divided into 10KHz is the rate of "ticks." For example, if this value is 2, the "tick" rate is 5KHz. You can use a Get LUN Information system directive to examine the value and a SET /BUF MCR function to modify it while the system is running.

The ticks parameter in a synchronous function specifies the number of "ticks" between samples or data transfers. The value of ticks is a 16-bit number. Thus 65,536 discrete sampling frequencies are possible for each of 65,536 different "tick" rates. This provides a maximum single-channel sample rate of 1 sample every 100 microseconds (possible in hardware but impractical in software) and a minimum of 1 sample every 429,495 seconds. A single-channel rate greater than 2KHz is possible, but not recommended.

The figures below represent initial timing tests run under RSX-11M. It should be noted that no computation was performed on the data other than continuously removing it from or inserting it into the data buffer.

The following data is for the LPS11 on a PDP-11/40 with memory management, with no gain-ranging option, and with digital I/O option.

Analog rates:

1 request for 1 channel at 2.5KHz

1 request for 2 channels at 2.0KHz (aggregate 4KHz)

2 requests for 1 channel at 2.0KHz (aggregate 4KHz)

Digital rates:

1 request for 2 channels at 2.5KHz (aggregate 5KHz)

The following data is for the AR11 on a PDP-11/40 with no memory management, no digital I/O option, and no unipolar sampling.

Analog rates:

1 request for 1 channel at 3.3KHz

1 request for 2 channels at 2.5KHz (aggregate 5.0KHz)

2 requests for 1 channel at 2.5KHz (aggregate 5.0KHz)

Digital rate:

2 requests for 2 channels at 3.3KHz (aggregate 6.6KHz)

17.6.2 Importance of the I/O Status Block

An I/O status block must be specified with every synchronous function. If the first I/O status word is nonzero, the request has been terminated and the value indicates the reason for termination. Otherwise, the request is in progress, and the second I/O status word contains the number of data values remaining in the buffer (or the number of free positions in the buffer for IO.MDA and IO.MDO).

17.6.3 Buffer Management

The buffer unload protocol for synchronous input functions is described below. You construct a 5-word block that contains the following:

```
IOSB:     .BLKW    2           ; I/O STATUS DOUBLE-WORD
CURPT:    .WORD    BUFFER      ; ADDRESS OF BUFFER
LSTPT:    .WORD    BUFFER+n    ; ADDRESS OF END OF BUFFER
FSTPT:    .WORD    BUFFER      ; ADDRESS OF BUFFER
```

Two of these words are required by the driver (I/O status block) and the remaining three by your task to unload data values from the buffer.

Your task then issues the I/O request with the appropriate parameters and the address of the above block as the I/O status block. The QIO directive 0s both I/O status words to initialize them.

If the driver accepts the request, it sets up a write pointer to the first word in your task's buffer. Thus, your task has a buffer read pointer and the driver has a buffer write pointer. Your task and the driver share the second I/O status word, which is the number of data words in the buffer that contain data.

Each time the driver attempts to put a sample value into the buffer, it increments the second I/O status word and compares the result with the size of the buffer. If the result is greater, buffer overrun has occurred and the request is terminated. Otherwise, the value is stored in the buffer at the address specified by the driver's write pointer and the write pointer is updated.

If the value stored in your task's buffer fills half of the buffer, the event flag specified in the I/O request is set to notify your task that a half buffer of data is available to be processed. Each time your task is awakened, it should execute the following code:

```
5$:       CLEF$S    #EFN            ;CLEAR EFN
10$:      TST       IOSB+2          ;ANY DATA IN BUFFER?
          BEQ       30$             ;IF EQ NO
          MOV       @CURPT,R0       ;GET NEXT VALUE FROM BUFFER
          DEC       IOSB+2          ;REDUCE NUMBER OF ENTRIES
          ADD       #2,CURPT        ;UPDATE BUFFER READ POINTER
          CMP       CURPT,LSTPT     ;END OF BUFFER?
          BLOS      20$             ;IF LOS NO
          MOV       FSTPT,CURPT     ;RESET BUFFER READ POINTER
20$:      Process data value        ;
          BR        10$             ;TRY AGAIN
30$:      TSTB      IOSB            ;REQUEST TERMINATED?
          BNE       40$             ;IF NE YES
          WTSE$S    #EFN            ;WAIT FOR EFN
          BR        5$              ;
40$:      Determine reason for termination
```

For IO.MDA and IO.MDO, this protocol differs slightly. Your task maintains a write pointer and the driver a read pointer. The entire buffer must be full when the request is executed.

17.6.4 Use of ADJLPS for Input and Output

The following FORTRAN example illustrates the proper protocol for using ADJLPS for synchronous input and output.

Synchronous input:

```
      DIMENSION IBF(1004),IERR(2),INTVL(2)
C
C INITIATE SYNCHRONOUS A/D SAMPLING,
C
      INTVL(1)=2
      INTVL(2)=5
      CALL RTS(IBF,1004,160,INTVL,IEFN,6,6,IERR,50,14,15)
C
C INITIALIZE HALF BUFFER INDEX
C
      INDX=4
C
C WAIT FOR HALF BUFFER OF DATA
C
 10   CALL WAITFR(IEFN)
C
C CLEAR EVENT FLAG
C
 15   CALL CLREF(IEFN)
C
C PROCESS HALF BUFFER OF DATA
C
      SUM=0
      DO 20 I=1,500
      SUM=SUM+CVSWG(IBF(I+INDX))
 20   CONTINUE
      AVERG=SUM/500
C
C FREE HALF BUFFER FOR MORE DATA
C
      CALL ADJLPS(IBF,500)
C
C ADJUST BUFFER INDEX
C
      INDX=INDX+500
      IF(INDX.GE.1004) INDX=4
C
C CHECK IF ANOTHER HALF BUFFER OF DATA IS AVAILABLE
C
      IF(IERR(2).GE.500 GO TO 15
      IF(IERR(1).NE.0) GO TO end of sampling
      GO TO 10
```

Synchronous output:

```
      DIMENSION IBF(1004),IERR(2),INTVL(2)
C
C FIRST BUFFER OF DATA MUST BE AVAILABLE AT START
C
C THIS EXAMPLE ASSUMES FIRST BUFFER IS FULL AT START
C
C START SYNCHRONOUS DIGITAL OUTPUT FUNCTION
C
      INTVL(1)=2
      INTVL(2)=5
      CALL SDO(IBF,1004,160,INTVL,IEFN,MASK,IERR,50,14,15)
C
C INITIALIZE HALF BUFFER INDEX
C
      INDX=4
C
C WAITFOR ROOM IN BUFFER
C
 10   CALL WAITFR(IEFN)
C
C CLEAR EVENT FLAG
C
 15   CALL CLREF(IEFN)
C
C CALCULATE VALUES TO PUT IN BUFFER
C
      X=(Y+2)*Z
      DO 20 I=1,500
      IBF(I+INDX)=X**5/A
 20   CONTINUE
C
C SIGNIFY ANOTHER HALF BUFFER IS FULL
C
      CALL ADJLPS(IBF,500)
C
C ADJUST BUFFER INDEX
C
      INDX=INDX+500
      IF(INDX.GE.1004) INDX=4
C
C CHECK IF ANOTHER HALF BUFFER IS EMPTY
C
      IF(IERR(2).GE.500) GO TO 15
      IF(IERR(1).NE.0) GO TO end of sampling
      GO TO 10
```

NOTE

In both of the examples above, care is taken to ensure that the program stay "in sync" with the driver. If the program "loses" its position with respect to the driver, the function must be stopped and restarted, because this is the only way to recover. Excercise caution to ensure that the program sequence above avoids a possible loss of data.

CHAPTER 18

PAPER TAPE READER/PUNCH DRIVERS

18.1 INTRODUCTION

The RSX-11M/M-PLUS paper tape reader/punch drivers support the PC11 paper tape reader/punch and the PR11 paper tape reader. The PC11 is a high-speed reader/punch capable of reading 8-hole, unoiled, perforated paper tape at 300 characters per second, and punching tape at 50 characters per second. The PR11 has the same characteristics as those of the paper tape reader portion of the PC11. All transfers are image mode only, with no interpretation of data.

18.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains the following information for paper tape devices. A bit setting of 1 indicates that the described characteristic is true for these devices.

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device

Bit	Setting	Meaning
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, which is 64 bytes for paper tape devices.

18.3 QIO$ MACRO

Table 18-1 lists the standard functions of the QIO macro that are valid for the paper tape reader/punch.

Table 18-1
Standard QIO Functions for the Paper Tape Reader/Punch

Format		Function
QIO$C	IO.ATT,...	Attach device
QIO$C	IO.DET,...	Detach device
QIO$C	IO.KIL,...	Cancel I/O requests
QIO$C	IO.RLB,...,<stadd,size>	READ logical block (reader only)
QIO$C	IO.RVB,...,<stadd,size>	READ virtual block (reader only)
QIO$C	IO.WLB,...,<stadd,size>	WRITE logical block (punch only)
QIO$C	IO.WVB,...,<stadd,size>	WRITE virtual block (punch only)

stadd

The starting address of the data buffer (may be on a byte boundary)

size

The data buffer size in bytes (must be greater than 0)

IO.KIL never cancels an in-progress read request. In-progress write requests are canceled only when the punch driver is waiting for the punch to become ready at the start of a transfer.

The paper tape drivers support no device-specific functions.

18.4 STATUS RETURNS

Table 18-2 lists error and status conditions that are returned by the paper tape reader/punch drivers.

Table 18-2
Paper Tape Reader/Punch Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The I/O request was canceled while in progress or while still in the I/O queue.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	**Device not ready** The reader and punch drivers return this code when a time-out occurs. The reader driver also returns this code when an error condition (see Section 18.4.1) is encountered before the initiation of the first transfer after an ATTACH command has been issued.
IE.EOF	**End-of-file encountered** The reader driver encountered an error condition (see Section 18.4.1) at a time other than the initiation of the first read after a valid ATTACH command. The second word of the I/O status block contains a count of bytes successfully read before the error condition was encountered.

(continued on next page)

Table 18-2 (Cont.)
Paper Tape Reader/Punch Status Returns

Code	Reason
IE.IFC	**Illegal function** An illegal function code was specified in an I/O request that is not legal for the respective paper tape drivers.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternatively, a byte count of 0 was specified.
IE.VER	**Unrecoverable hardware error (punch only)** The punch driver encountered an error condition (see Section 18.4.1) at a time other than the initiation of a transfer. Section 18.4.2 describes the action of the punch driver when an error condition is encountered upon the initiation of a transfer.

18.4.1 Error Conditions

There are four error conditions that are indistinguishable to the paper tape drivers. These conditions are:

- No tape
- Reader off line
- Power low
- Hardware malfunction

18.4.2 Ready Recovery

When the punch driver encounters an error condition upon the initiation of a transfer, the following message is displayed:

```
*** PPn:  -- NOT READY
```

n

The unit number of the paper tape punch that is not ready.

This message is repeated every 15 seconds until the error condition is corrected, or until the I/O request is canceled. When the error condition has been corrected, the transfer begins within 1 second.

18.5 PROGRAMMING HINTS

This section contains important information about programming the paper tape drivers described in this chapter.

18.5.1 Special Action Resulting from Attach and Detach

When an Attach or Detach is issued to the punch, the punch driver initiates a transfer of 170 (decimal) nulls. Upon the first read after an attach to the reader, all nulls preceding the first non-null character on the tape are read and discarded by the reader driver.

18.5.2 Reading Past End-of-Tape

When the reader driver reads past the physical end-of-tape, it normally generates at least two incorrect data bytes. These bytes are included in the byte count returned by the driver. Those of your tasks that does not prevent reads past the physical end-of-tape should discard at least the last six characters in the buffer when IE.EOF is returned by the driver.

CHAPTER 19

INDUSTRIAL CONTROL SUBSYSTEMS

19.1 INTRODUCTION

This chapter describes RSX-11M drivers for two process I/O subsystems: the ICS/ICR11 and the DSS/DRS11. (Driver support for these I/O subsystems is not provided in RSX-11M-PLUS systems.)

ICS11 and ICR11 are local and remote process I/O subsystems, respectively. They operate under program control as devices capable of interrogating digital and analog input, and driving digital and analog output.

DSS11 and DRS11 are digital input and output subsystems, respectively. Under program control they drive digital output and interrogate digital input.

19.1.1 Hardware Configuration

A single ICS or ICR controller can handle up to 16 I/O modules in any configuration; a module contains 16 bits of input or output data, providing a total of 256 digital points. Up to 12 ICR or ICS units are supported. You tailor the ICS/ICR driver to your needs, interactively, through the system generation dialogue. The driver can handle any combination of ICR or ICS controllers installed on a single system.

The DSS11 provides 49 optically isolated inputs, including 48 nonbuffered, sense-data inputs and one interrupt input. The DRS11 provides 48 open-collector, buffered outputs plus one interrupt input. You shape the DSS/DRS driver to your system's configuration in the system generation dialog. The driver supports up to 16 DSS11 or DRS11 modules, or combinations thereof.

19.1.1.1 **ICS/ICR Address Assignments** - Each ICR11A Unibus interface or ICS11 file box must be configured at system generation time for individually addressable interrupt vectors, Control and Status Registers (ICSR), and module Address Registers (ICAR), as shown in Table 19-1.

Table 19-1
ICS/ICR Address Assignments

ICS/ICR Unit No.	Module Addresses	ICSR/ICAR Addresses	Interrupt Vectors
0	171000-171036	171770-171776	234-236
1	171040-171076	171760-171766	xxx-xxx+2
2	171100-171136	171750-171756	xxx+4-xxx+6
3	171140-171176	171740-171746	xxx+10-xxx+12
4	171200-171236	171730-171736	xxx+14-xxx+16
5	171240-171276	171720-171726	xxx+20-xxx+22
6	171300-171336	171710-171716	xxx+24-xxx+26
7	171340-171376	171700-171706	xxx+30-xxx+32
10	171400-171436	171670-171676	xxx+34-xxx+36
11	171440-171476	171660-171666	xxx+40-xxx+42
12	171500-171536	171650-171656	xxx+44-xxx+46
13	171540-171576	171640-171646	xxx+50-xxx+52

NOTES

nnnnn6 = Control and Status Register
nnnnn4 = Address Register

Additional controllers are assigned vector addresses above 300.

19.1.1.2 **DSS/DRS Address Assignments** - Unlike the ICS/ICR subsystem, DSS/DRS devices are not restricted to specified bus addresses. However, the following constraints apply:

1. All DSS11 modules must occupy a contiguous set of bus addresses.
2. All DRS11 modules must occupy a contiguous set of bus addresses.
3. The total number of DSS11 and DRS11 modules may not exceed 16.
4. If both module types are installed in a system, the DRS11 must occupy the lower set of bus and interrupt-vector addresses.
5. Bus request priority is BR4.

19.1.1.3 **Supported ICS/ICR I/O Modules** - The following modules, all optional, are supported by the ICS/ICR driver:

D/A Converters

IDA-OA - 4-channel digital-to-analog converter

A/D Converters

IAD-IA - 8-channel wide-range differential analog-to-digital converter
IMX-IA - 16-channel flying capacitor relay multiplexer

Counters

IDC-IC - 16-bit binary counter

Bistable Digital Outputs

IDC-OA - D/C flip-flop driver
IAC-OA - A/C flip-flop driver
IRL-OA - Bistable relay output
IRL-OB - Flip-flop relay output

Momentary Digital Output

IDC-OB - D/C momentary driver
IAC-OB - A/C momentary driver

Digital Inputs (Noninterrupting) [1]

IDC-IA - D/C voltage sense input
IDC-ID - D/C voltage input module
IAC-IA - A/C voltage input module

Digital Inputs (Interrupting)

IDC-IB - D/C voltage interrupt input
IAC-IB - A/C voltage interrupt input

Terminal Input/Output

110 CPS Remote Terminal Interface to ICR11

---

1. Note that noninterrupting input modules are accessed directly by a task. Hence, while FORTRAN interface routines are available, no support for such modules is included in the driver.

19.1.2 Alternate ICS11 Support

The ICS11 Industrial Control Subsystem is supported either by the UDC11 or ICS/ICR11 device driver. If the system does not have an ICR11 controller, and if a driver of minimum size is required, then UDC11 support should be considered. The hardware requirements for such support are as follows:

1. Each file box must be assigned to the same interrupt vector address (normally 234).

2. The control and status register within each file box must appear at the same address within the I/O page (normally 171776).

If support of the IAD-IA A/D converter is required, the following module addressing and installation conventions are imposed:

1. Each IAD-IA converter and associated IMX-IA relay multiplexers are assigned a fixed block of 120 logical channel numbers. No more than 32 IAD-IA converters may be installed in a single system. Based on this convention, A/D converter 0 occupies channels 0-119, A/D converter 1 occupies 120-239, and so forth.

2. Regardless of the actual number of IMX-IA multiplexers installed, each converter preempts a block of eight contiguous module slots.

3. The slots reserved for all A/D converters and multiplexers must occupy a block of contiguous module slots.

If necessary, Field Service personnel can make the vector and address changes. Assuming the hardware configuration is correct, you can implement the desired UDC11 software support by answering in the affirmative all system generation questions relating to the UDC11.

If the additional ICS/ICR-11 driver features are required (at a commensurate increase in the memory requirements), then each ICS11 file box must be configured for individually addressable interrupt vectors and control status registers. This change can be performed by Field Service personnel. The necessary software support is incorporated by answering in the affirmative all system generation questions relating to the ICS/ICR11.

The additional ICS11 capabilities provided by the ICS/ICR11 driver may be summarized as follows:

1. Multicontroller, parallel operation

2. Increased A/D conversion throughput

3. Activation of tasks directly from digital interrupts or counters

4. No requirement to install modules of the same type in contiguous slots

Section 19.7 summarizes the software differences between the UDC and ICS/ICR drivers in detail.

19.1.3 Software Support

Both ICS/ICR and DSS/DRS operations are divided into two categories:

1. Functions performed directly by any task

2. Functions requiring driver services

Direct functions are accomplished through memory references to the ICS/ICR or DSS/DRS registers on the I/O page. In a protected system any task may gain restricted access to the device registers by linking to a global common block that resides within the appropriate physical memory limits. Direct functions consist of:

- Reading counter modules
- Reading any digital input module (DSS)

NOTE

All functions listed in this subsection apply to ICS/ICR modules. Those that also apply to the DSS or DRS subsystems are so marked.

Driver requests are divided into the following categories:

- Noninterrupting output functions
  - Bistable (flip-flop) digital output (DRS)
  - Analog output
  - Momentary (single-shot) digital output
- Requests for interrupting functions
  - Analog input
  - Remote terminal output
- Requests for unsolicited interrupts
  - Digital interrupts (DSS/DRS)
  - Counter interrupts
  - Remote terminal input
  - Remote unit or serial line errors

With the exception of A/D input and remote terminal output, all functions are complete upon return to your task.

Under RSX-11M, noninterrupting output functions are immediately submitted to the controller through a circular buffer that is filled at driver level and emptied at interrupt level. A QIO is considered successfully completed when the request is inserted in the circular buffer.

The following operations are in this category:

1. Bistable digital outputs
2. Analog outputs
3. Momentary digital outputs

Interrupting functions are those operations that generate an interrupt within some fixed time after initiation. The driver allows a list of multiple transactions to be specified in a single QIO. Each transaction is initiated in sequence without waiting for the preceding interrupt, until either the list is exhausted or all modules of the specified type are active. The following operations are in this category:

1. A/D inputs
2. Remote terminal output

Unsolicited interrupts may require no initiation by the processor and occur at indeterminate intervals. The following functions are in this category:

1. Interrupting digital inputs (DSS/DRS)
2. Counter modules
3. Remote terminal input
4. Error interrupts

All unsolicited interrupt data, except for errors, may be placed in a task-provided circular buffer. On interrupt, an event flag specified by the task is set. Such data for each module type is supplied to only one task per controller. In addition, the driver activates selected tasks on the occurrence of digital or terminal input interruptions.

Error interrupts are described later in this chapter.

Terminal support is restricted to passing terminal data between the device and a task. The only special character is Control-C (003), which may cause a task that you specify to be made active. There is no other special processing for terminal I/O except that the parity bit is removed. This is similar to the terminal driver function of IO.RAL.

1. MCR is not invoked.
2. Characters are not echoed.
3. Carriage control is not performed.
4. TABs, RUBOUTs, and so forth are not recognized.
5. Line terminators are not recognized.
6. Fill characters are not generated.

19.1.4 UDC11 Software Compatibility

Many of the MACRO and FORTRAN interfaces described in the following paragraphs are fully compatible with existing UDC11 applications software; however, you should consult Section 19.7 for a summary of differences that do exist between UDC and ICS/ICR software.

19.1.5 Module Addressing Conventions

Table 19-2 illustrates the relationship between physical slot numbers, bus addresses and relative addresses for a given ICS/ICR configuration. It is referred to in the following discussion.

Each A/D converter is assigned a block of 120 channels. The number of channels in use within the block depends on the number of multiplexers installed. Specifically, each A/D converter has eight channels, and each associated multiplexer has 16.

Table 19-2
Sample ICS/ICR Configuration

Unit: 0

Module Number	Slot Number	Type	Bus Address	Relative Addresses
0.	9.	D/A converter	171000	0-3.
1.	10.	A/D converter	171002	0-119.
2.	11.	A/D multiplexer	------	-----
3.	12.	Counter	171006	0
4.	13.	Flip-flop driver	171010	0-15.
5.	14.	D/A converter	171012	4-7.
6.	15.	Flip-flop driver	171014	16.-31.
7.	16.	Counter	171016	1.
8.	17.	A/D converter	171020	120.-239.

As noted, a block of 120 relative addresses is reserved for each A/D converter. The converter and multiplexer in slots 10 and 11 contain channels 0 through 23. The converter in slot 17 contains channels 120 through 127. The driver rejects an attempt to access a nonexistent channel (for example, channel 30 or channel 129).

You should observe that the bistable drivers in slots 13 and 15 contain relative point numbers 0 through 15, and 16 through 29 although the modules are not physically adjacent. In general, the relationship between slot number, module type, bus address, and relative address is as follows:

1. A set of contiguous relative addresses is defined for each module of a given type that is installed. Each relative address, when qualified by type, uniquely identifies a digital point or channel.

2. A set of slot numbers and bus addresses, possibly not contiguous, is occupied by all modules of a given type. Such addresses may be assigned solely on the basis of hardware and installation considerations. Increasing relative addresses correspond to increasing bus addresses.

Table 19-3 is an example of the relationship among bus addresses, interrupt points, and point numbers for a sample DSS11/DRS11 configuration.

Table 19-3
Sample DSS/DRS Configuration

Bus Addresses	Module Type	Points	Interrupt Point
160030-160036	DRS11	0-47.	0
160040-160046	DRS11	48.-95.	1
170010-170016	DSS11	0-47.	2
170020-170026	DSS11	48.-95.	3

All addressing is by point number. Except for the interrupts, all points are numbered sequentially by type (DSS or DRS), starting with the first point on the lowest address assigned to a given module type. Interrupt points are defined by means of a 16-bit mask word. Each bit in the mask defines an interrupting module; high-order bits correspond to increasing bus addresses.

19.2 LUN INFORMATION

A request for logical unit information returns the following device-dependent data in words 2 through 5 of the buffer:

WD 02 - 0

WD 03 - Undefined

WD 04 - Undefined

WD 05 - 0

19.3 ASSEMBLY LANGUAGE INTERFACE

Table 19-4 summarizes standard and device-specific QIO functions supported by the ICS/ICR driver. Only the five functions indicated by a footnote are supported by the DSS/DRS driver.

Table 19-4
Summary of Industrial Control QIO Functions

	Format	Function
QIO$C	IO.CCI,...,<stadd,sizb,tevf>	CONNECT a buffer to digital interrupts
QIO$C	IO.CTI,...,<stadd,sizb,tevf, arv>	CONNECT a buffer to counter interrupts
QIO$C	IO.CTY,...,<stadd,sizb,tevf>	CONNECT a buffer to terminal interrupts
QIO$C	IO.DCI,...	Disconnect a buffer from digital interrupts
QIO$C	IO.DTI,...	Disconnect a buffer from counter interrupts
QIO$C	IO.DTY,...	Disconnect a buffer from terminal interrupts
QIO$C	IO.FLN,...	Set controller off line
QIO$C	IO.ITI,...,<mn,ic>	INITIALIZE a counter
QIO$C	IO.LDI,...,<tname,[,tevf], pn,csm> [1]	LINK task to digital interrupts

1. These functions are supported by the DSS/DRS driver.

(continued on next page)

Table 19-4 (Cont.)
Summary of Industrial Control QIO Functions

	Format	Function
QIO$C	IO.LKE,...,<tname,[,tevf]>	LINK task to error interrupts
QIO$C	IO.LTI,...,<tname,[,tevf] cn[,arv]>	LINK task to counter interrupts
QIO$C	IO.LTY,...,<tname,[,tevf]>	LINK task to remote terminal interrupts
QIO$C	IO.MLO,...,<opn,pp,dp> [1]	OPEN or close bistable digital output points
QIO$C	IO.MSO,...,<opn,dp>	PULSE momentary digital output points
QIO$C	IO.NLK,...,<tname> [1]	UNLINK a task from all interrupts
QIO$C	IO.ONL,...	Place ICS/ICR controller online
QIO$C	IO.RAD,...,<stadd> [1]	READ activating data
QIO$C	IO.RBC,...,<stadd,size, stcnta>	INITIATE multiple A/D conversions
QIO$C	IO.SAO,...,<chn,vout>	PERFORM analog output
QIO$C	IO.UDI,...,<tname> [1]	UNLINK a task from digital interrupts
QIO$C	IO.UER,...,<tname>	UNLINK a task from error interrupts
QIO$C	IO.UTI,...,<tname>	UNLINK a task from counter interrupts
QIO$C	IO.UTY,...,<tname>	UNLINK a task from terminal interrupts.
QIO$C	IO.WLB,...,<stadd,sizb>	TRANSMIT data to the ICR remote terminal

1. These functions are supported by the DSS/DRS driver.

arv

The starting address of a buffer containing initial or reset counter values. The buffer must be aligned on a word boundary.

chn

The D/A channel number.

cn

The counter number.

csm

The change-of-state mask.

dp

The binary data pattern.

ic

The initial count.

mn

The module number.

opn

The first bistable (latching) digital output point number. This value must be on a module boundary (evenly divisible by 16).

pn

The point number (must be assigned on a module boundary).

pp

A 16-bit mask.

sizb

The data buffer size in bytes. For a circular buffer connected to unsolicited interrupts, this value must be even and large enough to include one entry plus the 2-word header.

size

The data and control buffer size in bytes. This value must be an even number that is greater than 0.

stadd

The starting address of the data buffer (must be on a word boundary).

staddb

The starting address of the terminal output buffer (may be aligned on a byte boundary).

stcnta

The starting address of the control buffer (must be on a word boundary); each control buffer word must be constructed as described in Table 19-5 (Section 19.3.2).

tevf

An event flag number in the range 0 to 96, (if the group-global event flag system generation option was selected), or 0 to 64 if group-global event flags are not supported.

tname

A task name composed of 1 to 6 alphanumeric characters in a 2-word RADIX-50 format. Two arguments, each containing three characters, are required for this parameter. For example, the task name ICNAME is specified as:

```
<^RICN,^RAME,...>
```

If the task name is less than four characters, a null argument must be specified as follows for task ABC:

```
<^RABC,,...>
```

vout

A binary number between 0 and 1023. that is to be converted to an analog output.

The following sections contain a detailed description of each function. In the discussion of QIO request parameters, the following conventions apply.

All numbering is relative.

Module numbers start at 0 beginning with the first module of a given type. Increasing module numbers correspond to increasing physical bus addresses.

Channel numbers start at 0, with channel 0 as the first channel on the first module of a given type.

Point numbers start at 0 with point 0 as the first point on the first module of a given type. Points within a module are numbered "from right to left" in increasing order.

It should be remembered that there is no requirement for ICS/ICR modules of a given type to occupy contiguous slots; thus, for example, digital points 15(decimal) and 16(decimal) need not reside on physically adjacent modules. This restriction does apply to DSS/DRS modules, however.

It is assumed that the number of points or channels per module is a constant for each generic type. Specifically, the following weights apply:

1. Each ICS/ICR Digital I/O Module contains 16 points.
2. Each DSS/DRS Digital I/O Module contains 48 points.
3. Each Counter Module contains 1 channel.
4. Each D/A Module contains 4 channels.
5. Each A/D Converter contains 120 channels.

As stated above, an A/D converter is assigned a block of 120 channels. The number of channels in use within the block depends on the number of multiplexers installed. The driver rejects an attempt to address a nonexistent channel.

19.3.1 General Error Status Returns

The system recognizes and handles two kinds of status conditions when they occur in I/O requests:

- Directive conditions, which indicate the acceptance or rejection of the QIO directive itself
- I/O status conditions, which indicate the success or failure of the I/O operation

Table 19-7 lists numerical values of returns for both assembly language and FORTRAN interfaces.

The following directive and I/O status returns apply uniformly to all requests.

19.3.1.1 Directive Conditions -

IS.SUC - Directive accepted.

The first six parameters of the QIO directive were valid, and sufficient dynamic memory was available to allocate an I/O packet. The directive is accepted.

IE.ADP - Invalid address.

The I/O status block or the QIO DPB was outside of the issuing task's address space or was not aligned on a word boundary.

IE.IEF - Invalid event flag number.

IE.ILU - Invalid logical unit number.

The lun specification in a QIO directive was invalid for the issuing task. For example, there were only five logical unit numbers associated with the task, and the value specified for lun was greater than five.

IE.NOD - Insufficient dynamic memory.

There was not enough dynamic memory to allocate an I/O packet for the I/O request. Your task can try again later by blocking the task with a WAITFOR SIGNIFICANT EVENT directive. Note that WAITFOR SIGNIFICANT EVENT is the only effective way for the issuing task to block its execution, because other directives that accomplish this purpose themselves require dynamic memory for their execution (for example, MARK TIME).

IE.SDP - Invalid DIC number or DPB size.

The directive identification code (DIC) or the size of the directive parameter block (DPB) was incorrect; the legal range for a DIC is from 1 through 127, and all DIC values must be odd. Each individual directive requires a DPB of a certain size. If the size is not correct for the particular directive, this code is returned.

IE.ULN - Unassigned LUN.

The logical unit number in the QIO directive was not associated with a physical device unit. Your task may recover from this error by issuing a valid Assign LUN directive and then reissuing the rejected directive.

19.3.1.2 I/O Conditions -

IE.ABO - Operation aborted.

The specified operation was canceled by IO.KIL or the request timed out while the unit was off line.

IE.OFL - Controller off line.

The physical device unit associated with the LUN specified in the QIO directive was not on line. An ICS/ICR controller may be off line because a device check during bootstrap load has indicated that the controller is not in the configuration.

IE.DNR - Controller not ready.

A nonrecoverable controller error has been detected.

IE.IFC - Illegal function.

A function code was included in an I/O request that is illegal for the ICS/ICR. The function may also refer to an ICS/ICR module type or function that was not specified during system generation.

19.3.2 A/D Input - Read Multiple A/D Channels

This function provides the capability of reading several A/D channels at any permissible gain. The driver is capable of initiating parallel transfers when more than one A/D converter is installed in a file box; however, only one interrupt module request (remote terminal or A/D) may be in progress at a given time.

QIO DPB format:

QIO$C IO.RBC,...,<stadd,size,stcnta>

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes (must be even and greater than 0); the control buffer is the same size.

stcnta

The starting address of the control buffer (must be on a word boundary); each control buffer word must be constructed as shown in Table 19-5.

Return Status:

IS.SUC - Function successfully completed.

IE.BAD - Illegal channel or gain code specified.

IE.BYT - Data buffer is byte aligned. Alternatively, the length of the buffer is not an even number of bytes.

IE.DNR - Device not ready. A/D converter interrupt time-out occurred.

Note that the second I/O status word contains a count of the number of conversions successfully completed.

One control word is paired with each data word. That is, the data appearing in a data array element is obtained using the gain and channel number specified in the corresponding element of the control array. Control words specify the gain and channel in the format shown in Table 19-5.

Upon receiving and validating the parameters within the I/O packet, the driver initiates the following sampling procedure:

1. The control word is fetched and tested for validity (that is, for legal gain and channel). If an error is encountered or no further control words remain, processing is terminated as described in Step 4.

2. Assuming the A/D converter board is idle, the driver starts the conversion, sets this resource busy, and returns to step 1. If the converter is busy, the driver returns control to the system after saving the data required to initiate the conversion when the channel becomes idle.

3. On the occurrence of an A/D interrupt, the interrupt service routine initiates the appropriate processing at the non interrupt level that either sets the channel idle or initiates a previous request stored during step 2. The occurrence of the latter results in processing of additional control words as described in step 1.

Table 19-5
A/D Conversion Control Word

Bits	Meaning				
0-11	Channel Number range: 0-1919				
12-15	Gain value for this sample. The binary value is as follows:				Gain
	15	14	13	12	
	0	0	0	0	1
	0	0	0	1	2
	0	0	1	0	illegal
	0	0	1	1	illegal
	0	1	0	0	10
	0	1	0	1	20
	0	1	1	0	illegal
	0	1	1	1	illegal
	1	0	0	0	50
	1	0	0	1	100
	1	0	1	0	illegal
	1	0	1	1	illegal
	1	1	0	0	200
	1	1	0	1	1000
	1	1	1	0	illegal
	1	1	1	1	illegal

4. The converted value is returned as 12 bits, left-justified, in a 16-bit word, with the low-order 4 bits set to 0.

5. A/D requests are terminated under any of the following conditions:

   a. All control words have been processed.

   b. A hardware error has occurred.

   c. An error in a control word has been detected.

Regardless of the cause, the driver cannot complete request processing until all pending A/D transfers have gone to completion.

Because of overlapped processing, multiple errors can occur (for example, a hardware error and an erroneous control word). The driver returns the status associated with the earliest transaction that caused an error condition. Thus, at the interface to your task, the driver appears to execute all conversions sequentially.

19.3.3 **Analog Output**

This function provides the capability of setting a single analog output channel to a specified voltage.

QIO DPB format:

QIO$C IO.SAO,...,<chn,vout>

chn

The output channel number.

vout

The output voltage representation.

Output voltage varies linearly with the binary input to the channel, where 0 to plus 10 volts (+10v.) is represented by integers from 0 to 1023.

Return Status:

IS.SUC - Function submitted for output to controller.

IE.MOD - Nonexistent D/A channel was specified.

The second I/O status word is 0.

19.3.4 Momentary Digital Output - Multi-Point

This function provides the capability of pulsing a field of up to 16 momentary (single-shot) digital output points. Fields must be aligned on module boundaries.

QIO DPB format:

QIO$C IO.MSO,...,<opn,dp>

opn

The starting digital output point number. Point number must be aligned on a module boundary (that is, must be a multiple of 16).

dp

The 16-bit mask. One point is pulsed corresponding to each bit set in the mask word.

Return Status:

IS.SUC - Function submitted for output to the controller.

IE.MOD - Invalid starting point number specified. Point is nonexistent or not aligned on a module boundary.

19.3.5 Bistable Digital Output - Multi-Point

This function provides the capability of setting or resetting a field of up to 16 bistable digital output points. Fields must be aligned on a module boundary.

QIO DPB format:

```
QIO$C IO.MLO,...,<opn,pp,dp>
```

opn

The starting digital output point number. Point number must be aligned on a module boundary (that is, must be a multiple of 16).

pp

The 16-bit mask.

dp

The data pattern.

A bit is set in the mask word for each point that may change state. The state of points corresponding to reset mask bits is unaltered. When the mask bit is set, the output is "closed" if the data bit is set and "open" if the data bit is clear.

Return Status:

IS.SUC - Function submitted for output to the controller.

IE.MOD - Invalid starting point number specified. Point does not exist or is not aligned on a module boundary.

19.3.6 Unsolicited Interrupt Processing

Unsolicited interrupts consist of the following:

1. Digital interrupts
2. Counter interrupts
3. Remote terminal input
4. Hardware errors

Based on the type of interrupt, the driver may dispose of the interrupt data in one or more of the following ways:

1. The data may be furnished to a task that has issued a request to monitor such information continually. This alternative is not available in the DSS/DRS driver.
2. A task may be activated by a specific input. That is, a dormant task can be requested to run, or an event flag may be set if the task is currently active.

The driver allows continual monitoring for digital, counter, and terminal inputs with the provision that, for each controller, only one task per module type may receive such inputs.

Task activation is permitted for digital, terminal, and error interrupts. The processing related to hardware errors is discussed in Section 19.5. Activation of tasks by digital, counter, and terminal inputs is covered in Section 19.3.7.

The driver functions described in the following paragraphs allow a task to continually receive interrupt data. To monitor such data, a task must provide:

1. A buffer that is filled by the driver and emptied by the task in circular fashion

2. An event flag that is set upon the occurrence of each interrupt

The driver connects a single task per controller to receive interrupts from a specific module type.

The buffer to be connected has the format shown below:

FORTRAN Index	Contents
1	driver index
2	your task's index
3	word 0 of entry
4	word 1 of entry
.	.
.	.
.	.

The buffer consists of a 2-word header containing the driver and your task's index, as shown, followed by a data area that is subdivided into fixed-length entries. Each entry consists of a word containing the entry existence indicator followed by one or more words of device-dependent data. Such information usually consists of module data, relative module number, and a code identifying a module type. On the occurrence of an interrupt, the driver enters data in the location currently indicated by the driver index. This index can be considered as a FORTRAN index into the buffer. That is, the first location in the buffer is associated with the index 1. The beginning of the data area is associated with the first entry, index 3. Entries are made in a circular fashion starting at the beginning of the data area, filling in order of increasing memory address, and wrapping around to the beginning of the data area when there is insufficient space for an entry at the end. Note that the size of the data area must be an integer multiple of the entry size.

It is expected that the connected task maintains the index, ensuring that it indicate where, in the buffer, the task is to process interrupt data next.

When the task is activated by the driver, it should process data in the buffer starting at the location indicated by its pointer, and continuing in circular fashion until an existence indicator is encountered that is 0.

The existence indicator is set to +1 when a buffer entry is made. Except to record a hardware error, the contents of an entry are not altered by the driver if the indicator is nonzero. Hence, when a requester has removed or processed the entry, he must clear the existence indicator to free the buffer entry position. If the driver detects a nonzero indicator, (that is, data input has occurred in a burst sufficient to overrun the buffer), the data is discarded and a count of data overruns is incremented. The count is maintained in the entry existence indicator, which, as noted above, is set to +1 to indicate no overruns between entries, +2 to indicate a hardware error entry, or a negative value recording the two's complement of the number of times data has been discarded between entries. The overrun count is never allowed to wrap around to a positive value.

In the event of a nonrecoverable controller error (remote unit power-fail or hard data error) all connected tasks are activated with the following entry in the circular buffer:

```
WD 00          Hardware error indicator (+2)
    .
    .
    .
WD nn          Contents of ICSR register
WD nn+1        Physical unit number
WD nn+2        Generic code indicator
               set to 177770(octal)
```

nn

The offset to module data word.

This entry is always placed in the buffer regardless of overflow status.

The error flags are obtained from the controller ICSR word at the time the error was detected (see Table 19-7).

19.3.6.1 **Connect to Digital Interrupts** - This function allows a single task to receive digital interrupt data.

QIO DPB format:

```
QIO IO.CCI,...,<stadd,sizb,tevf>
```

stadd

The starting address of buffer to be connected (must be word aligned).

sizb

The length of buffer in bytes (must be even). Minimum buffer length is 14 bytes.

tevf

The trigger event flag number.

Return Status:

IS.SUC - Function successfully completed. Second I/O status word contains the number of words passed per interrupt in the low byte, and the initial FORTRAN index in the high byte.

IE.BYT - Buffer address is byte aligned or length is an odd number of bytes.

IE.CON - Interrupt already connected to another task.

IE.IEF - Invalid event flag number.

IE.PRI - Task checkpointable and not fixed in memory.

IE.SPC - Interrupt circular buffer was not wholly within the address space of the task. Alternatively, the buffer was too small for a single data entry (seven words minimum).

Entry Format:

WD 00 - Existence Indicator
WD 01 - Change of state indicator
WD 02 - Module data
WD 03 - Relative module number
WD 04 - Generic Code 1, 2, or 3, indicating a digital interrupt

The contents of the existence indicator have been described previously.

The change-of-state indicator records those bits for which a change of state in the direction of interest has been detected. The direction of the change may be from 0 to 1 (point closed (PCL)) or 1 to 0 (point open (POP)) depending upon the PCL or POP jumper connections on the digital interrupt module. The driver assumes that at least one of these signals is always asserted.

The relative module number indicates the module on which the change of state was recognized.

The module data word records data received at the time the interrupt was serviced.

The generic code identifies the type of module that caused the interrupt. A digital interrupting module may have the value 1, 2, or 3 as selected by jumpers that you install on the module.

19.3.6.2 **Disconnect from Digital Interrupts** - This function allows a task to terminate the processing of digital interrupt data.

QIO DPB format:

```
QIO$C IO.DCI,...
```

Return Status:

IS.SUC - Function successfully completed. Second I/O status word is 0.

IE.CON - Task was not connected. Second I/O status word is 0.

19.3.6.3 **Connect to Counter Module Interrupts** - This function allows a single task to receive counter interrupt data.

QIO DPB format:

QIO$C IO.CTI,...,<stadd,sizb,tevf,arv>

stadd

The starting address of circular buffer (must be word aligned).

sizb

The length of buffer in bytes (must be even). Minimum buffer length is 12 bytes.

tevf

The trigger event flag number.

arv

The starting address of table of initial counter values (must be word aligned).

Word 03 defines an array of initial counter values. One entry is required for each counter installed in a physical unit. Entries are paired with modules in logically ascending sequence. The counter is set to the initial value upon receipt of the connect function and whenever an overflow interrupt occurs (that is, when the count reaches 0).

Return Status:

IS.SUC - Function successfully completed. The second I/O status word contains the number of words passed per interrupt in the low byte, and the initial FORTRAN index in the high byte.

IE.BYT - Buffer address is byte aligned or length is an odd number of bytes.

IE.CON - Interrupt already connected to another task.

IE.IEF - Invalid event flag number.

IE.PRI - Task checkpointable and not fixed in memory.

IE.SPC - Interrupt circular buffer or table of initial values was not wholly within the address space of the task. Alternatively, the buffer was too small for a single data entry (six words minimum).

Entry Format:

WD 00 - Existence indicator

WD 01 - Module data

WD 02 - Relative module number

WD 03 - Generic code (4, 5, or 6)

19.3.6.4 **Set Counter Initial Value** - This function allows a counter initial value to be established. A task need not be connected to counter interrupts to perform this function.

QIO DPB format:

QIO$C IO.ITI,...,<mn,ic>

mn

The relative module number.

ic

The new initial count.

Return Status:

IS.SUC - New value submitted for output to the controller. The second word of I/O status is set to 0.

IE.MOD - Nonexistent module number specified.

Upon receipt of the request, the new initial value is immediately queued for output to the controller. The counter is reinitialized with this value on overflow if a task is connected to counter interrupts.

19.3.6.5 **Disconnect from Counter Interrupts** - This function allows a task to terminate counter interrupt processing.

QIO DPB format:

QIO$C IO.DTI,...

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is set to 0.

IE.CON - Task was not connected to timer interrupts.

After disconnect is complete, counters are not reset to the initial value at the time of the interrupt.

19.3.6.6 **Connect to Terminal Interrupts** - This function allows a task to receive terminal inputs from the selected ICR11 controller.

QIO DPB format:

QIO$C IO.CTY,...,<stadd,sizb,tevf>

stadd

The address of the circular buffer (must be word aligned).

sizb

The length of buffer (must be even). The minimum buffer length is 12 bytes.

tevf

The trigger event flag number.

Return Status:

IS.SUC - Function successfully completed. The second I/O status word contains the number of words passed per interrupt in the low byte, and the initial FORTRAN index in the high byte.

IE.BYT - Buffer is byte aligned or length is an odd number of bytes

IE.CON - Interrupt already connected to another task.

IE.IEF - Invalid event flag number.

IE.MOD - Nonexistent device. Controller is ICS11.

IE.SPC - Interrupt circular buffer was not wholly within the address space of the task. Alternatively, the buffer was too small for a single entry (six words minimum).

Entry Format:

WD 00 - Existence indicator

WD 01 - High byte = 0, low byte = terminal input character

WD 02 - Relative module number (normally 0)

WD 03 - Generic code indicator (normally 0)

Note that words 2 and 3 are nonzero only when the entry was made as the result of a nonrecoverable controller error.

All remote terminal data is conveyed to the requesting task as input, but with the parity bit removed.

NOTE

Remote terminal input is not echoed by the driver.

19.3.6.7 **Disconnect from Terminal Input** - This function allows a task to discontinue the processing of terminal input.

QIO DPB format:

```
QIO$C IO.DTY,...
```

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is set to 0.

IE.CON - Task was not connected to remote terminal interrupts.

19.3.7 Activating a Task by Unsolicited Interrupts

The functions described in the following paragraphs provide the capability of:

1. Activating a task in response to unsolicited interrupts
2. Interrogating the driver to determine the reason for activation
3. Removing a task from the activation list

The QIO DPB parameters specify the task name, an optional trigger event flag to be set if the task is active, and device-dependent parameters that identify the interrupt source. A task is linked to interrupts (that is, made eligible for activation) provided that:

1. The resource exists.
2. The task is installed.
3. No other task is linked to the resource.

If another task is linked to the resource, the driver rejects the request with a status of resource-in-use (IE.RSU). A resource is defined as a single interrupt point, remote terminal (Control-C input only), or counter module.

On the occurrence of the appropriate interrupt, the task is made active if dormant; otherwise, a trigger event flag, if specified, is set. The task may interrogate the driver to determine the conditions that caused activation, and to signify interrupt recognition. The function of the event flag is to allow such a task to recognize an event that has occurred while the task was active. Recognition is ensured prior to the completion of task execution by issuing the Exit If system directive followed by the Clear Event Flag directive.

The linkage between a task and a specific interrupt is removed by issuing the appropriate unlink request with the QIO directive.

Only one task may be associated with each interrupt source (that is, one task per digital interrupt point, terminal input, or counter module.)

NOTE

The MCR command REMOVE unlinks a task from all interrupts.

19.3.7.1 **Link a Task to Digital Interrupts** - This function allows a task to be activated on the occurrence of digital interrupts.

QIO DPB format:

```
QIO$C IO.LDI,...,<tname,[,tevf],pn,csm>
```

tname

A 1- to 6-character alphanumeric task name in 2-word, Radix-50 format.

tevf

The trigger event flag (0 = none).

pn

The point number (must be aligned on a module boundary).

csm

The change-of-state mask.

The change-of-state mask indicates those bits for which a change of state in the direction specified by the PCL and POP jumpers causes the task to be activated. Only one task may be linked to a given interrupt point. A 0 change-of-state mask is not permitted.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is set to 0.

IE.BAD - Change-of-state mask set to 0.

IE.IEF - Invalid event flag number.

IE.MOD - Nonexistent module or point not aligned on a module boundary.

IE.NOD - Insufficient dynamic memory to allocate secondary control block.

IE.NST - Task "tname" is not installed.

IE.RSU - One or more of the specified points is in use by other tasks.

19.3.7.2 **Link a Task to Counter Interrupts** - This function allows a task to be activated by means of an interrupt from a single counter module.

QIO DPB format:

```
QIO$C IO.LTI,...,<tname,[,tevf],cn[,ic]>
```

tname

A 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

tevf

The trigger event flag (0 = none).

cn

The relative module number.

ic

The counter value (optional).

The counter value, if nonzero, reinitializes the module in a manner similar to that described for the Set Counter function in Section 19.3.6.4. Initialization may be bypassed by setting this parameter to 0.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is set to 0.

IE.IEF - Invalid event flag number.

IE.MOD - Nonexistent module specified.

IE.NOD - Insufficient dynamic memory to allocate a secondary control block.

IE.RSU - Counter is linked to another task.

19.3.7.3 **Link a Task to Terminal Interrupts** - This function allows a task to be activated by means of an interrupt from a remote terminal. The task is activated only in response to the Control-C character (octal 003).

QIO DPB format:

QIO$C IO.LTY,...,<tname,[,tevf]>

tname

A 1- to 6-character alphanumeric task name in 2-word, Radix-50 format.

tevf

The trigger event flag (0 = none).

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is 0.

IE.IEF - Invalid event flag number.

IE.MOD - Nonexistent module (unit is ICS11 controller).

IE.NOD - insufficient dynamic storage to allocate secondary control block.

IE.NST - Task "tname" is not installed.

IE.RSU - Remote terminal is linked to another task.

19.3.7.4 **Link a Task to Error Interrupts** - This function allows a single task to be activated whenever a remote unit power-fail or nonrecoverable serial line error is detected on any or all remote units in a system. Only one task within a system may be linked to error interrupts. Once linked, the selected task may receive error reports from any ICR controller.

QIO DPB format:

QIO$C IO.LKE,...,<tname,[,tevf]>

tname

A 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

tevf

The trigger event flag (0 = none).

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is 0.

IE.IEF - Invalid event flag number.

IE.IFC - No ICR11 subsystems are installed.

IE.NOD - Insufficient dynamic storage to allocate secondary control block.

IE.NST - Task "tname" is not installed.

IE.RSU - Another task is linked to error interrupts.

19.3.7.5 **Read Activating Data** - This function allows a task to determine the conditions that caused it to be activated.

QIO DPB format:

```
QIO$C IO.RAD,...,<stadd>
```

stadd

The address of 6-word buffer to receive activation data (must be word aligned).

The buffer receives data in the following format:

WD 00 - Activation indicator

WD 01 - Physical unit number

WD 02 - Generic code

WD 03 - Relative module number

WD 04 - Hardware dependent data

WD 05 - Hardware dependent data

The activation indicator is similar in function to the existence indicator that the task uses when it reads circular buffer entries. The occurrence of an interrupt to which the requesting task is linked sets the indicator to +1, and the appropriate data is stored. The indicator is cleared when the data is solicited by the task. If an interrupt linked to the task occurs and the parameter is nonzero then the previously stored data is not modified and the driver sets this element with the two's complement of the number of linked interrupts not recorded.

The physical unit number specifies the controller that received the interrupt.

The generic code is identical to that specified for circular buffer entries, namely:

0 - Terminal (Control-C)

1,2,3 - Digital interrupt

4,5,6 - Counter interrupt

177770 - Fatal controller error

Hardware-dependent data is associated with generic code and consists of the following:

Terminal:

WD 04 - Terminal buffer contents (low byte)

WD 05 - Undefined

Digital Interrupts:

WD 04 - Module data

WD 05 - Change-of-state indicator

Counter:

WD 04 - Module data

WD 05 - Undefined

Fatal Controller Error:

WD04 - Contents of ICSR register (see Table 19-7)

WD05 - Contents of ICAR register (see Table 19-8)

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is 0.

IE.BYT - Buffer address is aligned on an odd byte boundary.

IE.NLK - Task "tname" was not linked to interrupts.

IE.SPC - Buffer not totally within the task's address space.

19.3.8 Unlink a Task from Interrupts

The functions described in the following paragraphs provide the capability of:

1. Unlinking a task from all interrupts on a controller
2. Selectively unlinking a task from interrupts by module type

19.3.8.1 **Unlink a Task from All Interrupts** - This function unlinks a task from all interrupts on a given controller and from error interrupts.

QIO DPB format:

```
QIO$C IO.NLK,...,<tname>
```

tname

A 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is 0.

IE.NLK - Task "tname" was not linked to interrupts.

19.3.8.2 **Unlink a Task from all Digital Interrupts** - This function provides the capability of unlinking a task from all digital interrupt points on a controller.

QIO DPB format:

```
QIO$C IO.UDI,...,<tname>
```

tname

a 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is 0.

IE.NLK - Task "tname" was not linked to the specified class of interrupt.

IE.NST - Task not installed.

IE.MOD - Nonexistent module type specified.

19.3.8.3 **Unlink a Task from Counter Interrupts** - This function provides the capability of unlinking a task from all counter module interrupts.

QIO DPB format:

```
QIO$C IO.UTI,...,<tname>
```

tname

A 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is 0.

IE.NLK - Task "tname" was not linked to the specified interrupts.

IE.NST - Task not installed.

IE.MOD - Nonexistent module type specified.

19.3.8.4 **Unlink a Task from Terminal Interrupts** - This function provides the capability of unlinking a task from terminal interrupts.

QIO DPB format:

```
QIO$C IO.UTY,...,<tname>
```

tname Tname parameter IO.UTY function (ICDRV/ISDRV)

A 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is 0.

IE.NLK - Task "tname" was not linked to the specified interrupts.

IE.NST - Task not installed.

IE.MOD - Nonexistent module specified (that is, device is an ICS11 controller).

19.3.8.5 **Unlink a Task from Error Interrupts** - This function provides the capability of unlinking a task from all error interrupts.

QIO DPB format:

```
QIO$C IO.UER,...,<tname>
```

tname

A 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is 0.

IE.IFC - No ICR11 controllers exist in the system.

IE.NLK - Task "tname" was not linked to error interrupts.

IE.NST - Task not installed.

19.3.9 Terminal Output

This function allows a task to perform output to the terminal device. Characters are output exactly as they appear in the buffer. The carriage control parameter is not recognized. It should be noted that only one interrupt module request per controller (terminal or A/D) may be in progress at a given time. Thus, the driver does not initiate an A/D operation on a given controller, until any terminal output in progress for that controller has been completed.

QIO DPB format:

```
QIO$C IO.WLB,...,<staddb,sizb>
```

staddb

The buffer address (may be odd).

sizb

The byte count (may be odd).

Return Status:

IS.SUC - Function successfully completed. Second word of I/O status contains the number of bytes output.

IE.MOD - Nonexistent hardware function. Request was issued for an ICS11 controller.

19.3.10 Maintenance Functions

The functions described below allow a privileged task to enable and disable error reporting while troubleshooting or maintenance on a remote unit is in progress.

19.3.10.1 **Disable Hardware Error Reporting** - This function allows a privileged task to disable error reporting and error interrupts, and restrict access to the controller while remote unit troubleshooting or module calibration is in progress (see Section 19.5.1). Upon receipt and validation of the request, error interrupts are disabled and subsequent controller time-outs are ignored. The occurrence of device time-out while A/D conversion or remote terminal input is in progress results in termination of the request with the error code IE.ABO. When error reporting is disabled in this manner, access to the controller for input or output to I/O modules is restricted to privileged tasks. All other requests not requiring the transmission of data to or from the device are permitted for all tasks. Such requests are as follows:

1. Disconnect from digital, counter, or remote terminal interrupts
2. Unlink from interrupts
3. Read activating data

4. Link to digital, remote terminal, or error interrupts

5. Connect a buffer to digital or remote terminal interrupts

All other requests not issued by a privileged task are rejected with the error code IE.DNR.

QIO DPB format:

```
QIO$C IO.FLN,...
```

Return Status:

IS.SUC - Function successfully completed

IE.FLN - Unit already off line

IE.PRI - Task not privileged

19.3.10.2 **Enable Hardware Error Reporting** - This function allows a privileged task to enable error reporting and device error interrupts. Upon receipt and validation of the function, all device error interrupts are enabled and the unit is marked on line. These actions are performed regardless of the current state of the unit.

QIO DPB format:

```
QIO$C IO.ONL,...
```

Return Status:

IS.SUC - Function successfully completed

IE.PRI - Task not privileged

19.3.11 Special Functions

19.3.11.1 **I/O Rundown** - An I/O rundown request from the Executive causes the task to be disconnected from all interrupts. The rundown operation is not finished until any A/D input in progress for the task has been completed.

19.3.11.2 **Kill I/O** - The kill I/O function allows a task to initiate I/O rundown processing for itself on any device. Request processing is identical to that described for I/O rundown.

QIO DPB format:

```
QIO$C IO.KIL,...
```

Return Status:

IS.SUC - Function successfully completed

19.4 FORTRAN INTERFACE

Table 19-6 lists the FORTRAN interface subroutines supported for the ICS/ICR subsystem. (D) indicates a direct access call. The six subroutines supported by the DSS/DRS driver are indicated by a footnote.

Unless specifically noted, all subroutines are reentrant (but not necessarily position-independent) and may be placed in an absolute resident library.

Table 19-6
FORTRAN Interface

Subroutine	Function
AIRD/AIRDW	Input analog data from multiple channels in random sequence
AISQ/AISQW	Read a series of sequential analog input channels at random gain
AO/AOW	Perform analog output on several channels
ASICLN/ ASUDLN	Assign a LUN to an ICS/ICR controller
ASISLN [1]	Assign a LUN to a DSS/DRS controller
CTDI	Connect a circular buffer to receive digital interrupt data
CTTI	Connect a circular buffer to receive counter interrupt data
CTTY	Connect a circular buffer to receive ICR11 remote terminal data
DFDI	Disconnect a buffer from digital interrupts
DFTI	Disconnect a buffer from counter interrupts
DFTY	Disconnect a buffer from remote terminal interrupts
DI/DIW [1]	Read several 16-point digital sense fields (D)
DOL/DOLW [1]	Latch or unlatch several 16-point bistable output fields
DOM/DOMW	Pulse multiple 16-point momentary digital output fields
LNK [1]	Link a task to unsolicited interrupts
OFLIN	Suppress error reporting. Place unit in not ready status
ONLIN	Enable error reporting. Return unit to ready status
RCIPT	Read a single digital interrupt point (D)

1. These subroutines are supported by the DSS/DRS driver.

(continued on next page)

Table 19-6 (Cont.)
FORTRAN Interface

Subroutine	Function
RDACT [1]	Read interrupt activation data
RDDI	Read the digital interrupt circular buffer
RDTI	Read the counter interrupt circular buffer
RDCS	Read digital interrupt circular buffer; return data on only those points for which a change of state has been recognized
RDWD	Read digital interrupt circular buffer; return a full data word
RSTI	Read a single counter module (D)
RTO/RTOW	Perform output to a remote ICR11 terminal
UNLNK [1]	Unlink a task from unsolicited interrupts

1. These subroutines are supported by the DSS/DRS driver.

19.4.1 Synchronous and Asynchronous Process Control I/O

The Instrument Society of America (ISA) standard provides for synchronous and asynchronous I/O. Synchronous I/O is indicated by appending a W to the name of the subroutine (for example, AO/AOW). Except for analog input and terminal output, all QIOs issued by the process control subroutines are service immediately by the driver and are complete upon return to the issuing task. In such cases, there is no functional difference between the synchronous and asynchronous forms; however, both forms of the name are recognized. In the case of A/D input and terminal output, the subroutines are functionally distinct. If the task uses the asynchronous form, execution continues and the task must periodically test the status word for completion.

19.4.2 Return Status Reporting

The I/O status parameter is a 2-word integer array. The first element of the array receives the status of the FORTRAN call in accordance with ISA convention.

This array serves two purposes:

1. It is the 2-word I/O status block to which the driver returns an I/O status code on completion of an I/O request.

2. The first word of the status block receives a status code from the FORTRAN interface subroutine in ISA-compatible format, with the exception of the I/O pending condition, which is indicated by a status of 0. The ISA standard code for this condition is +2.

For asynchronous analog input and terminal output, status is set by means of an asynchronous trap; therefore, the trap mechanism must be enabled while these functions are in progress.

For compatibility, the 2-word status block is also required for status returned by the direct access calls. Errors of this type that may be returned are:

Word 1 = 3	Number of points requested is 0.
Word 1 = +321	Invalid ICS/ICR module.

The status code must be interpreted in the context of the function requested; however, the following general conditions apply:

Contents of Status Word 1	Meaning
0	Operation pending, I/O in progress
+1	Successful completion
+3	Error in a calling argument has been detected by the interface subroutine
3<Word 1<300.	QIO directive rejected. Actual error code = -(WORD 1 - 3)
Word 1 > 300	Request rejected by driver. Actual error code = -(WORD 1 - 300)

Table 19-7 lists all possible status values: the FORTRAN value, assembly language mnemonic, actual value, and related definition.

Table 19-7
Return Status Summary

FORTRAN Interface Value	Assembly Language Value	Assembly Language Mnemonic	Definition
+0	+0	IS.PND	Operation pending
+1	+1	IS.SUC	Successful completion
+3	none	none	Error detected in FORTRAN calling sequence
+4	-1	IE.UPN	Insufficient dynamic storage to allocate I/O packet
+8	-5	IE.ULN	Unassigned LUN
-6	-6	IE.LNL	LUN usage interlocked
+99	-96	IE.ILU	Invalid LUN
+100	-97	IE.IEF	Invalid event flag number

(continued on next page)

Table 19-7 (Cont.)
Return Status Summary

FORTRAN Interface Value	Assembly Language Value	Assembly Language Mnemonic	Definition
+101	-98	IE.ADP	Part of DPB out of your task's addressing space
+102	-99	IE.SDP	Invalid DIC or DPB size
+301	-1	IE.BAD	Bad parameters
+302	-2	IE.IFC	Invalid I/O function code
+303	-3	IE.DNR	Device not ready
+306	-6	IE.SPC	Illegal buffer
+315	-15	IE.ABO	Request aborted
+316	-16	IE.PRI	Privilege violation
+317	-17	IE.RSU	Resource in use
+319	-19	IE.BYT	Buffer address or length is odd
+321	-21	IE.MOD	Illegal module number
+322	-22	IE.CON	Another task already connected to interrupts
+323	-23	IE.NOD	Insufficient dynamic memory to allocate secondary control block
+379	-79	IE.NLK	Task not linked to interrupts
+380	-80	IE.FLN	ICR11 already off line
+381	-81	IE.NST	Task is not installed
+397	-97	IE.IEF	Invalid event flag number

19.4.3 Optional Arguments

The calling sequences discussed in subsequent sections frequently contain optional arguments. These arguments are enclosed in square brackets within the calling sequence description. A statement containing such arguments may be written with these parameters deleted by truncating the argument list if the optional parameters are at the end of the calling sequence, or by replacing them with commas if they are embedded elsewhere in the list. Consider the routine XYZ below having two optional arguments:

```
CALL XYZ(ibuf [,ilen] [,ival])
```

If the argument ival is to be omitted, the calling sequence would be:

```
CALL XYZ(IBUF,ILEN)
```

When your task is to omit an optional argument in the middle of the list, it is replaced with a comma. Consider the routine XYZ, above. Use the following statement to omit the parameter ilen:

```
CALL XYZ(IBUF,,IVAL)
```

NOTE

In some subroutines, lun -- the logical unit number -- is indicated to be an optional argument. It is optional only if one of the Assign LUN subroutines has been called (ASICLN, ASUDLN, ASISLN). Otherwise, the lun argument is mandatory.

19.4.4 Assigning Default Logical and Physical Units for Input and Output - ASICLN/ASUDLN (ICS/ICR) and ASISLN (DSS/DRS)

The following subroutines must be called to assign and record a default LUN and physical unit if either parameter is to be unspecified in subsequent FORTRAN calls for which these parameters are optional.

Calling Sequence:

```
CALL ASICLN([lun] [,idsw] [,iunt])
CALL ASUDLN([lun] [,idsw] [,iunt])
CALL ASISLN(lun[,idsw][,iunt])
```

Before a task can issue the call to ASUDLN, the ASN command must be issued through MCR to assign logical device UDnn to the appropriate physical ICS/ICR unit.

Argument Description:

lun - An integer variable whose value is the number of the LUN to be assigned to the physical unit specified by iunt or unit 0. If you do not specify a LUN, none is assigned. The lun argument is mandatory for ASISLN (used for DRS11 only).

idsw - An optional integer variable to receive the result of the assign lun directive.

iunt - An optional integer variable that specifies the unit number to be assigned. Assumed to be 0 if omitted.

Return Status:

The following values are returned to idsw:

+1 - Assignment or function successfully completed.

-5 - LUN usage is interlocked because LUN is assigned to a device that is attached to another device, or a file is currently open on the LUN.

-96 - Invalid LUN.

The call to ASUDLN assigns a LUN to logical device UD: and is provided for compatibility with existing UDC11 software. The call to ASICLN assigns a LUN to device IC:. The call to ASISLN assigns a LUN to device IS:.

Upon successful issuance of the Assign LUN directive, the subroutine executes a Get LUN Information directive to obtain the actual unit numbers to be saved. It is therefore possible to alter the default physical unit referenced in a direct access call, by means of the ASN MCR function, provided that such logical assignments are done before the task is made active.

Examples:

1. Assign LUN 5 to ICR unit 3.

```
     CALL ASICLN (5,IERR,3)
     IF(IERR) 20,10,10
10   --------------
```

2. Assign LUN 1 to logical device UD:, unit 0

   a. The following MCR command is issued to create logical device UD0:, and assign all references to physical device IC1:.

      >ASN IC1: = UD:

   b. The FORTRAN call

      CALL ASUDLN (1)

      assigns logical device UD0: to LUN 1. Because of the previous ASN command, the Executive assigns this LUN to physical device IC1: and return a value of 1 for the unit number in response to the GET LUN Information directive. This value is stored and later referenced whenever the physical unit number is unspecified in any of the FORTRAN calls that reference the I/O page directly.

3. Assign LUN 6 to logical device IS:, unit 2.

```
CALL ASISLN(6,,2)
```

19.4.5 Analog Input

The following routines provide the capability of performing A/D input:

AIRD/AIRDW - ISA Standard call to read multiple channels in random order. This call requires one or more control variables containing A/D channel and gain in the format shown in Table 19-5 (Section 19.3.2).

AISQ/AISQW - ISA Standard call to read multiple channels in sequential order.

19.4.5.1 **AIRD/AIRDW: Analog Input - Specified Channel Sequence** - The ISA standard call provides the capability of reading multiple A/D channels in a specified sequence.

Calling Sequence:

```
CALL AIRD(inm,icont,idata[,isb],[lun])
```

or

```
CALL AIRDW(inm,icont,...etc.)
```

Argument Descriptions:

inm - Integer variable specifying the number of channels to be read.

icont - An integer array of size inm containing control data in the format shown in Table 19-5 (Section 19.3.2).

idat - An integer array of dimension inm to receive the converted values. Each element in the array is paired with a control element in icont that defines the channel and gain.

isb - An optional 2-word integer array to receive the results of the call as follows:

- +1 - Conversion successfully completed. The second word contains the number of channels converted.
- +3 - Number of channels requested was 0.
- +4 - Insufficient dynamic storage to allocate I/O packet.
- +8 - LUN was not assigned.
- +99 - Invalid LUN.
- +301 - At least one invalid control word was specified. The second I/O status word contains the number of channels successfully converted.
- +303 - Device not ready. Interrupt response was not received from an A/D channel within one second after initiation. The second word of I/O status contains the number of channels successfully converted.
- +306 - Control or data buffer not wholly within your task's addressing space.
- +319 - Control or data buffer is byte aligned.

lun - An integer variable specifying the ICS/ICR logical unit number. This parameter is optional.

Example:

The following example illustrates how A/D throughput can be increased when several IAD-IA A/D Converters are in a system. This is accomplished by means of interleaved samples that initiate parallel conversions on each module. Samples are to be obtained from 12 channels on 3 IAD-IA A/D converter modules at a gain of 1.

```
C
C PROGRAM TO SAMPLE 12 A/D CHANNELS
C IN RANDOM SEQUENCE FOR MAXIMUM
C THRUPUT.
C
C CHANNELS TO BE SAMPLED:
C
C         0
C         1      -A/D MODULE 0
C         2
C         3
C       120
C       121      -A/D MODULE 1
C       122
C       123
C       240
C       241      -A/D MODULE 2
C       242
C       243
C
C INTERLEAVED SEQUENCE FOR MAXIMUM
C THRUPUT.
C
C         0
C       120
C       240
C         1
C       121
C       241
C         2
C       122
C       242
C         3
C       123
C       243
C
C THE FORTRAN CONVENTION FOR ARRAY
C STORAGE CAN REPRESENT THE ABOVE SEQUENCE
C IN AN N X I INTEGER
C CONTROL ARRAY.  WHERE:
C
C       N = NUMBER OF MODULES TO BE SAMPLED
C       I = NUMBER OF SAMPLES PER/MODULE
C
C ALLOCATE STORAGE FOR CONTROL ARRAY
C
       DIMENSION ICONT (3,4)
C
C INITIALIZE CONTROL ARRAY FOR IAD-IA MODULE 0
C
       DATA ICONT(1,1),ICONT(1,2),ICONT(1,3),ICONT(1,4)/0,1,2,3/
C
C INITIALIZE CONTROL ARRAY FOR IAD-IA MODULE 1
C
       DATA ICONT(2,1),ICONT(2,2),ICONT(2,3),ICONT(2,4)/120,121,122,123/
C
C INITIALIZE CONTROL ARRAY FOR IAD-IA MODULE 2
C
       DATA ICONT(3,1),ICONT(3,2),ICONT(3,3),ICONT(3,4)/240,241,242,243/
C
C ALLOCATE STORAGE FOR DATA ARRAY
C IN SIMILAR FASHION TO FACILITATE
C CHANNEL REFERENCES
```

```
C
      DIMENSION IDATA (3,4)
C
C     BEGIN EXECUTABLE STATEMENTS
C
                .
                .
                .
C
C INITIATE A/D SYNCHRONOUS CONVERSION ON LUN 3
C
      CALL AIRDW(12,ICONT,IDATA,,3)
                .
                .
                .
```

19.4.5.2 **AISQ/AISQW: Analog Input - Sequential Channel Sequence -** The ISA standard call described below provides the capability of sampling multiple A/D channels in sequential order. Channels are sampled in increments of one, beginning with the channel specified in icont(1).

Calling Sequence:

```
CALL AISQ(inm,icont,idata [,isb],[lun])
```

or

```
CALL AISQW(inm,icont...etc.)
```

Argument Descriptions:

inm - Integer variable specifying the number of elements to be read.

icont - An integer array of size inm containing initial channel in the first element only, and gain in the format shown in Table 19-5 in the remaining elements.

idat - An integer array of size inm to receive the converted values. Each element is paired with the corresponding control element in icont that defines the gain parameter.

Channels are sampled sequentially starting with the first channel specified in element 1 of icont.

isb - An optional 2-word integer array to receive the results of the call as follows:

+1 - Conversion successfully completed. The second word contains the number of channels converted.

+3 - Number of channels requested was 0.

+4 - Insufficient dynamic storage to allocate I/O packet.

+8 - LUN was not assigned.

+99 - Invalid LUN.

+301 - At least one invalid control word was specified. The second I/O status word contains the number of channels successfully converted.

+303 - Device not ready. Interrupt response was not received from an A/D channel within one second after initiation. The second word of I/O status contains the number of channels successfully converted.

+306 - Control or data buffer is not wholly within your task's addressing space.

+319 - Control or data buffer is byte aligned.

lun - An integer variable containing the logical unit number. This parameter is optional.

Example:

The following example illustrates the procedure for sequential sampling. Five channels are converted at gains of 1, 2, 20, 50, and 1000, starting at channel 3.

```
C
C ALLOCATE SPACE FOR STATUS ARRAY
C
      DIMENSION ISB (2)
C
C ALLOCATE SPACE FOR CONTROL ARRAY
C AND ESTABLISH INITIAL VALUES
C
      DIMENSION ICONT(5)
      DATA ICONT(1),ICONT(2),ICONT(3)/0000003,0010000,0050000/
      DATA ICONT(4),ICONT(5)/0100000,0150000/
C
C ALLOCATE SPACE FOR DATA ARRAY
C
      DIMENSION IDAT (5)
              .
              .
              .
C
C INITIATE SEQUENTIAL, ASYNCHRONOUS CONVERSION
C VIA LUN 1
C
      CALL AISQ(5,ICONT,IDAT,ISB,1)
10    IF(ISB(1).NE.0) GO TO 20

      (continue processing)
              .
              .
              .
C
C TEST CONVERSION STATUS
C
      GO TO 10
20    (test for errors or process converted data)
              .
              .
              .
      END
```

19.4.6 AO/AOW: Analog Output - Multichannel

This ISA standard routine is called to output voltage from multiple D/A channels.

Calling Sequence:

```
CALL AO(inm,icnt,idat[,isb][,lun])
```

or

```
CALL AOW(inm,icnt...and so forth)
```

Argument Descriptions:

inm - Integer variable containing the number of channels to be output.

icnt - Integer array containing the channel numbers to receive output.

idat - Integer array containing the output voltage setting as a value between 0 and 1023 where:

0 = 0 volts dc and

1023 = +9.99 volts (full scale).

isb - Optional 2-word integer array to receive status. One of the following values is returned in isb(1). The second element is always 0.

- +1 - Function successfully completed.
- +3 - No channels requested.
- +4 - Insufficient dynamic storage to allocate an I/O packet.
- +8 - LUN was not assigned.
- +99 - Invalid LUN.
- +303 - Controller not ready.
- +321 - Nonexistent channel specified.

lun - Integer variable containing the logical unit number.

Example:

Output the variable voltages contained in IV(1) and IV(2) to D/A channels 2 and 3, respectively.

```
C
C ALLOCATE DATA ARRAY
C
      DIMENSION IV(2)
C
C ALLOCATE CONTROL ARRAY
C
      DIMENSION ICNT(2)
C
C ALLOCATE STATUS ARRAY
C
      DIMENSION ISB(2)
C
C INITIALIZE CONTROL ARRAY
C
      DATA ICNT(1),ICNT(2)/2,3/
                .
                .
                .
C
C PERFORM A/D OUTPUT VIA LUN 3
C
      CALL AOW(2,ICNT,IV,ISB,3)
      IF (ISB(1).GE.3) go to error processor
```

19.4.7 DOL/DOLW: Digital Output - Bistable Multiple Fields

The following ISA standard call provides the capability of latching or unlatching multiple 16-point bistable digital output fields.

Calling Sequence:

```
CALL DOL(inm,icnt,idat,imsk[,isb][,lun])
```

or

```
CALL DOLW(inm,icnt...and so forth)
```

Argument Descriptions:

inm - Integer variable specifying the number of fields to be latched or unlatched.

icnt - Integer array containing the initial point within each field.

idat - Integer array containing binary data that defines points within the field to be latched or unlatched. The state of each bit is interpreted as follows:

1 = Latch point

0 = Unlatch point

imsk - Integer array containing binary data that defines points within the field for which a change of state is permitted.

A bit set to 1 defines a point that may assume the state defined by the corresponding bit in idat. A 0 bit specifies a point for which no change of state is permitted.

isb - Optional 2-word integer array to receive the results of the call. Status is returned in isb(1) as shown below. isb(2) is always 0.

+1 - Function successfully completed.

+3 - No points specified.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - LUN was not assigned.

+99 - Invalid LUN.

+303 - Controller not ready.

+321 - Nonexistent point number specified. One or more points within the field do not exist.

lun - Integer specifying the Logical Unit Number.

Example:

Reset points 0,1,20 and 21

```
      DIMENSION ICNT(2),IDAT(2),IMSK(2)
C
C INITIALIZE THE CONTROL ARRAY
C
      DATA ICNT(1),ICNT(2)/0,20/
C
C INITIALIZE MASK ARRAY TO EFFECT A
C CHANGE-OF-STATE ONLY ON THE SPECIFIED
C POINTS.
C
      DATA IMSK(1),IMSK(2)/O0000003,O0000003/
            .
            .
            .
C
C RESET THE SPECIFIED POINTS.  ICR IS ASSIGNED
C TO LUN 3.
C
      CALL DOLW(2,ICNT,IDAT,IMSK,,3)
            .
            .
            .
```

19.4.8 Digital Input

Both of the following subroutines perform their functions through direct access to the ICS/ICR hardware registers. Therefore, the physical unit number replaces LUN in the calling sequences described below. Note that any need for conversion of BCD encoded digital input into binary can be accomplished through the FORTRAN function

```
IBIN=KBCD2B (IBCD).
```

Binary data can be converted to BCD through the FORTRAN function.

```
IBCD=KB2BCD (IBIN).
```

The maximum input value for conversion is 9999.

NOTE

When the physical unit number is explicitly included in the calling sequence, it cannot be reassigned by the MCR command ASN.

19.4.8.1 DI/DIW: Digital Input - Digital Sense Multiple Fields - This ISA standard subroutine provides the capability of reading multiple 16-point digital sense fields.

Calling Sequence:

```
CALL DI(inm,icnt,idat[,isb][,iun])
```

or

```
CALL DIW(inm,icnt...and so forth)
```

Argument Descriptions:

- inm - Integer variable specifying the number of fields to be read.
- icnt - Integer array containing the initial point number of each field.
- idat - Integer array to receive the input data.
- isb - Optional, 2-word integer array to receive the results of the call. The status is returned in isb (1) as follows:
  - +1 - Function successfully completed.
  - +3 - No points requested.
  - +321 - Nonexistent point requested. One or more points within the 16-bit field does not exist.
- iun - Optional integer variable specifying the physical unit number.

Example:

Read two contact sense fields starting at points 3 and 27 on physical unit IC2:.

```
DIMENSION  ICNT(2), IDAT(2),ISB(2)
DATA   ICNT(1),ICNT(2)/3,27/
       .
       .
CALL DI (2,ICNT,IDAT,ISB,2)
IF (ISB(1).GE.3) go to error procedure
       .
       .
       .
```

19.4.8.2 **RCIPT: Digital Input - Digital Interrupt Single-Point** - The following subroutine returns the state of a single digital interrupt point as a logical value.

Calling Sequence:

```
CALL RCIPT (ipt,isb[,iun])
```

Argument Descriptions:

ipt - Integer variable defining the point to be read.

isb - a 2-word integer array to receive status and data as follows. Status is returned to isb(1).

+1 - Function successfully completed. Data is returned to isb(2) as a logical value, where:

.TRUE. (-1) = Point closed.

.FALSE. (0) = Point open.

+321 - Nonexistent point specified.

iun - Optional integer variable defining the physical unit number.

Example:

Read the state of contact interrupt point 3 on unit 0.

```
DIMENSION ISB (2)
     .
     .
     .
CALL RCIPT (3,ISB,0)
IF (ISB(2).EQ..FALSE.) go to point open routine.
```

19.4.9 DOM/DOMW: Digital Output Momentary - Multiple Fields

This ISA standard call allows multiple 16-bit fields to be pulsed.

Calling Sequence:

```
CALL DOM (inm,icnt,idat[,idx][,isb][,lun])
```

or

```
CALL DOMW (inm,icnt...and so forth)
```

Argument Descriptions:

- inm - Integer variable specifying the number of fields to be pulsed.
- icnt - Integer array containing the initial point in each field.
- idat - Integer array defining the points to be pulsed. A bit is set corresponding to each point that is to be triggered.
- idx - Optional dummy integer variable retained for compatibility with the standard form of the call.
- isb - Optional 2-word integer array to receive the results of the call as follows in isb(1), isb(2) is set to 0.
  - +3 - Number of fields to be output is 0.
  - +4 - Insufficient dynamic storage to allocate on I/O packet.
  - +8 - LUN not assigned.
  - +99 - Invalid LUN.
  - +303 - Controller not ready
  - +321 - Nonexistent point specified. One or more points within a field do not exist.
- lun - Integer variable defining the logical unit number.

Example:

Pulse momentary digital output fields defined by points 20, 37, and 0 on LUN 1.

```
DIMENSION ICONT(3),IDAT(3)

DATA ICONT(1),ICONT(2),ICONT(3)/20,37,0/

CALL DOM(3,ICONT,IDAT,,1)
```

19.4.10 RTO/RTOW: Remote Terminal Output

The following function provides the capability of transmitting a character string to a remote ICR11 terminal. Both synchronous and asynchronous forms are supported.

Calling Sequence:

```
CALL RTO (ibc,idat[,isb][,lun])
```

or

```
CALL RTOW (ibc,idat....etc.)
```

Argument Descriptions:

ibc - Integer variable specifying the number of bytes to output.

idat - Byte array (LOGICAL * 1) containing the character string to be output.

isb - Optional, 2-word integer array to receive the results of the call in isb(1) as follows. isb(2) is set to the number of bytes actually transferred to the device.

- 0 - Operation pending.
- +1 - Function successfully completed.
- +3 - No bytes to be transmitted.
- +4 - Insufficient dynamic storage to allocate I/O packet.
- +8 - Unassigned LUN.
- +99 - Invalid LUN.
- +303 - Device not ready. Terminal failed to respond within 1 second after character was transmitted.
- +306 - Part or all of buffer is out of the issuing task's addressing space.
- +321 - Nonexistent module. Device is ICS11.

lun - Integer variable defining the logical unit number.

Example:

Output a character string to a remote terminal by the ICR unit assigned to LUN 3.

```
CALL RTOW(32,'APPLY +5 VOLTS TO A/D CHANNEL 10',,3)
```

19.4.11 Unsolicited Interrupt Data - Continual Monitoring

Subroutines are provided that permit a FORTRAN program to continually monitor unsolicited interrupt data supplied to your task's circular buffer, as described in Section 19.3.6. Such routines allow the program to connect a buffer for input, disconnect the buffer upon completion, and read and return the buffer contents in a format suitable for FORTRAN processing. The calls summarized below perform these functions for interrupting digital input modules, counters, and remote terminal inputs:

Interrupting Digital Inputs

- CTDI - Connect a buffer to receive digital interrupts
- RDDI - Read the state of a single interrupting point
- RDCS - Read the state of a single interrupting point for which a change of state has been detected
- RDWD - Read 16 bits of interrupt data from the circular buffer
- DFDI - Disconnect a buffer from digital interrupts

Counter Modules

- CTTI - Connect a buffer to receive counter interrupts
- RDTI - Read the counter circular buffer
- DFTI - Disconnect a buffer from counter interrupts

Remote Terminal Input

- CTTY - Connect a buffer to receive remote terminal inputs
- RDTY - Read remote terminal data from the circular buffer
- DFTY - Disconnect a buffer from remote terminal interrupts

19.4.11.1 CTDI: Connect a Buffer for Receiving Digital Interrupt Data

- The following routine allows a task to provide a circular buffer that receives digital interrupt data, and to define an event flag that is set upon the occurrence of each interrupt.

Calling Sequence:

```
CALL CTDI (ibuf,isz,iev[,isb][,lun])
```

Argument Descriptions:

- ibuf - An integer array making up the circular buffer that is to receive interrupt data.
- isz - Integer variable specifying the length of the circular buffer in words.

iev - Integer variable specifying the event flag that is to be set whenever the driver receives an interrupt from a digital input module.

isb - Optional, 2-word integer array to receive the results of the call. The status values specified below are returned to isb(1).

+1 - Function successfully completed. isb(2) receives the number of words passed per interrupt in the low byte.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+306 - Part of buffer is out of your task's address space or buffer is too small to accommodate a single entry.

+316 - Privilege violation - task is check-pointable and not fixed in memory.

+319 - Buffer address or length is an odd number of bytes.

+322 - Another task is already connected to interrupts.

+397 - Invalid event flag specified.

lun - Integer variable specifying the logical unit number.

The space allocated for the circular buffer must be large enough to accommodate at least one 5-word entry plus an additional 10 words of storage that are required by the subroutines that read circular buffer contents. Thus, the buffer allocation specified by the integer variable isz may be computed as

```
isz = (10 + 5 * n)
```

n

The number of entries to be contained in the buffer.

isz

Expressed in words.

19.4.11.2 **Reading Digital Interrupt Data** - Each of the following routines reads data that has been stored in the circular buffer and performs the following common processing:

1. Detects, and optionally reports, the occurrence of an error entry that has been placed in the buffer by the driver because of a nonrecoverable device fault (for example, fatal serial line error or remote power-fail).

2. Clears the trigger event flag when no further entries remain to be processed.

3. Clears and optionally reports any overrun conditions.

Only one of the following three routines can be invoked by a single task:

1. RDDI: Read Digital Interrupt Data from a Circular Buffer

   The RDDI FORTRAN subroutine reads contact interrupt data from a circular buffer that was specified in a CTDI call (see 19.4.11.1). It does no actual input or output, but rather performs a point-by-point scan of an interrupt entry in the buffer, returning the state of each point as a logical value.

   On the initial call to RDDI, the module number and data of the next interrupt entry are read from the circular buffer and stored for subsequent reference. The subroutine then sets the current data bit number n to 0, examines the state of data bit n, and converts bit n to a point number by the following formula:

   ipt = module number * 16 + n

   On each subsequent call, n is incremented by one and then data-bit n is examined in the stored module data. When n reaches 16, it is reset to 0 and an attempt is made to read the next interrupt entry from the circular buffer. If a valid entry is not found, ipt is set negative and ict (if specified) is either assigned a value of 0 or an overrun count that is maintained by the ICS/ICR driver. If ict is 0, no further entries remain. A nonzero value indicates that the driver received more data than could be stored in the buffer, and ict represents the number of entries that were discarded.

   The variable ict receives the control register contents that are set by the driver -- as described in Section 19.3.6 -- whenever a nonrecoverable controller error occurs.

   Calling Sequence:

   CALL RDDI (ipt,ival[,ict])

   Argument Descriptions:

   ipt - A variable to which the digital input point number is returned. It may be set as follows:

   1. ipt 0 if no valid entry is found

      The specific value of ipt reflects the error that was detected as follows:

      -1 - no data (that is, no interrupt data currently in buffer)

      -2 - overrun

      -3 - hardware error

   2. ipt = 0 if the value indicated is a point number; the state is returned to ival.

ival - A variable to which the state of the point is returned; it may be set as follows:

1. .FALSE. (0) if the point is open

2. .TRUE. (-1) if the point is closed

ict - Optional integer variable to receive the overrun count or the contents of the CSR register on the occurrence of a fatal controller error. Otherwise, set to 0.

NOTE

A task reading the circular buffer should not issue a Wait-For directive until a buffer-empty condition is reported. See Section 19.4.11.11 for an example of how to read circular-buffer entries.

2. RDCS: Read Digital Interrupt Points That Have Changed State

The RDCS FORTRAN subroutine returns data in the format of subroutine RDDI -- as described above -- except that only points that have changed state are processed, resulting in significantly improved throughput and reduced processing overhead for the calling task.

Processing specific to the routine is as follows:

On the initial call, the module number, module data, and change of state information are read from the circular buffer and stored for later reference. The subroutine then sets the current data bit number n to 0 and begins scanning the change-of-state word until a nonzero bit is found. The point number and current state are then reported as previously described. If no change of state is found or when no further bits remain to be processed, the next entry is fetched as described above.

The processing of error conditions is identical to subroutine RDDI.

Calling Sequence:

```
CALL RDCS (ipt,ival[,ict])
```

Argument Descriptions:

ipt - Integer variable to receive the digital input point number. It may be set as follows:

1. ipt 0 if no valid entry is found (that is, overrun, error, or no data in buffer). The specific value of ipt reflects the error that was detected as follows:

-1 - no data
-2 - overrun
-3 - hardware error

2. ipt = 0 if the value indicated is a point number, the state is returned to ival.

ival - Integer variable to receive the state of the point as a logical value where:

1. .FALSE. (0) = Point open

2. .TRUE. (-1) = Point closed

ict - Optional integer variable. A nonzero value indicates that the variable has been set with an overrun count returned by the driver, or with the contents of the CSR register on the occurrence of a fatal controller error. Otherwise, set to 0.

NOTE

A task reading the circular buffer should not issue a Wait-For directive until a buffer-empty condition is reported. See Section 19.4.11.11 for an example of how to read circular-buffer entries.

3. RDWD: Read a Full Word of Digital Interrupt Data

The following subroutine is called to return a full word of digital interrupt data from the circular buffer, and optionally change of state information. A new entry is read for each call; hence, throughput is high when processing is contingent upon several possible conditions within a module.

Calling Sequence:

```
CALL RDWD (imod,ival[,ict][,icos])
```

Argument Descriptions:

imod - Integer variable to receive the module number or status as follows:

1. imod 0 if no data is present or an overrun condition or error was detected

   The specific value of ipt reflects the error that was detected as follows:

   -1 - no data
   -2 - overrun
   -3 - hardware error

2. imod = 0 Module number. Interrupt data is in ival

ival - Integer variable to receive the digital interrupt data.

ict - Optional integer variable. A nonzero value indicates that the variable has been set with an overrun count returned by the driver, or with the contents of the CSR register on the occurrence of a fatal error. Otherwise, set to 0.

icos - Optional integer variable to receive change-of-state information. Bits set to a 1 correspond to points for which a change of state has been recorded.

NOTE

A task reading the circular buffer should not issue a Wait-For directive until a buffer-empty condition is reported. See Section 19.4.11.11 for an example of how to read circular-buffer entries.

19.4.11.3 **DFDI: Disconnect a Buffer from Digital Interrupts** - The following routine is called to connect a task's circular buffer from digital interrupts.

Calling Sequence:

```
CALL DFDI ([isb][,lun])
```

Argument Descriptions:

isb - Optional 2-word integer array to receive the results of the call as follows. isb(2) is always 0.

+1 - Function successfully completed

+4 - Insufficient dynamic storage to allocate I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+322 - Task not connected to interrupts.

lun - Integer variable containing logical unit number.

19.4.11.4 **CTTI: Connect a Buffer for Receiving Counter Data** - The following subroutine may be called to connect a circular buffer that is to receive counter data, and to define an event flag that is to be set upon occurrence of each interrupt.

Calling Sequence:

```
CALL CTTI (ibuf,isz,iev,iv[,isb][,lun])
```

Argument Descriptions:

ibuf - An integer array making up the circular buffer that is to receive interrupt data.

isz - Integer variable specifying the length of the circular buffer in words.

iev - Integer variable defining an event flag that is to be set whenever the driver receives an interrupt from a counter module.

iv - Integer array of initial counter values. Each counter in the physical unit requires one element. The value initializes and resets the counter when a value of 0 is reached. You may reset this parameter for a specific module through a call to SCTI.

isb - Optional 2-word integer array to receive the results of the call. The status values specified below are returned to isb(1).

+1 - Function successfully completed. isb(2) receives the number of words passed per interrupt in the low byte.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+303 - Controller not ready.

+306 - Part of buffer is out of your task's address space or buffer is too small to accommodate a single entry.

+316 - Privilege violation -- task is checkpointable and not fixed in memory.

+319 - Buffer address or length is an odd number of bytes.

+322 - Another task is already connected to interrupts.

+397 - Invalid event flag specified.

lun - Integer variable specifying the logical unit number.

The space allocated for the circular buffer must be large enough to accommodate at least one 4-word entry plus an additional 8 words of storage required by the subroutine that reads buffer contents (RDTI). The buffer allocation specified by the variable isz may be computed as

isz = (8 + 4 * n)

n

The number of entries to be contained in the buffer.

NOTE

A task reading the circular buffer should not issue a Wait-For directive until a buffer-empty condition is reported. See Section 19.4.11.11 for an example of how to read circular-buffer entries.

19.4.11.5 **RDTI: Read Counter Data from the Circular Buffer** - The following call returns counter interrupt data from the circular buffer. A new entry is read on each call.

Calling Sequence:

```
CALL RDTI (imod,ival[,ict])
```

Argument Descriptions:

imod - Integer variable to receive module number and status as follows:

1. imod 0 No data in buffer, data overrun or error condition detected The specific value of ipt reflects the error that was detected as follows:

   -1 - no data
   -2 - overrun
   -3 - hardware error

2. imod = 0 Module number of counter. Interrupt data is in ival.

ival - Integer variable to receive the counter data at interrupt.

ict - Optional integer variable to receive the overrun count, or the ICSR contents returned by the driver on the occurrence of a fatal hardware error. Otherwise, set to 0.

NOTE

A task reading the circular buffer should not issue a Wait-For directive until a buffer-empty condition is reported. See Section 19.4.11.11 for an example of how to read circular-buffer entries.

19.4.11.6 **Miscellaneous Counter Routines**

1. RSTI: Read a Counter Module

   The following routine directly accesses a counter register to return its current value.

Calling Sequence:

```
CALL RSTI (imod,isb[,iun])
```

Argument Descriptions:

imod - An integer variable containing the number of the counter to be read.

isb - A 2-word integer array to receive status and data as follows. Status is returned to isb(1).

+1 - Function successfully completed. Data is returned to isb(2).

+321 - Nonexistent module specified.

iun - Optional integer variable specifying the ICS/ICR physical unit number.

2. SCTI: Reset a Counter Initial Value

The following routine may be called by any task to revise the initial value that activates a counter.

Calling Sequence:

```
CALL SCTI (imod,ival[,isb][,lun])
```

Argument Descriptions:

imod - Integer variable specifying the relative module number of the counter to be reset.

ival - Integer value specifying the new initial value.

isb - Optional 2-word integer array to receive status as follows. isb(2) is always 0.

+1 - Function successfully completed

+4 - Insufficient dynamic storage to allocate an I/O packet

+8 - Unassigned LUN

+99 - Invalid LUN

+303 - Controller not ready

+321 - Nonexistent module specified

lun - Integer specifying the logical unit number.

19.4.11.7 **DFTI: Disconnect a Buffer from Counter Interrupts** - The following subroutine is called to disconnect the task's circular buffer from interrupts.

Calling Sequence:

```
CALL DFTI ([isb][,lun])
```

Argument Descriptions:

isb - Optional 2-word integer array to receive status as follows. isb(2) is always 0.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+322 - Task was not connected to interrupts.

lun - Integer variable specifying the logical unit number.

19.4.11.8 **CTTY: Connect a Circular Buffer to Terminal Interrupts** - The following routine allows a task to provide a circular buffer to receive remote terminal input data, and to define an event flag that is set on the occurrence of each interrupt.

Calling Sequence:

```
CALL CTTY (ibuf,isz,iev[,isb][,lun])
```

Argument Descriptions:

The following arguments are identical in form and function to those described for subroutine CTDI (see Section 19.4.11.1):

ibuf - An integer array making up the circular buffer that receives interrupt data

isz - Length of the circular buffer in words

iev - Event flag to be set on each terminal interrupt

Buffer size is computed as

```
isz = (8 + 4 * n)
```

n

The number of entries that can be stored in the buffer.

isb - Optional 2-word integer array to receive the results of the call. The status values specified below are returned to isb(1).

+1 - Function successfully completed. isb(2) receives the number of words passed per interrupt in the low byte.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+306 - Part of buffer is out of your task's address space or buffer is too small to accommodate a single entry.

+316 - Privilege violation -- task is checkpointable and not fixed in memory.

+319 - Buffer address not on a word boundary or length is an odd number of bytes.

+321 - Nonexistent module specified. Unit is ICS11.

+322 - Another task is already connected to interrupt.

+397 - Invalid event flag specified.

lun - Logical unit number.

19.4.11.9 **RDTY: Read a Character from the Terminal Buffer** - This subroutine retrieves a single character from the terminal circular buffer on each call.

Calling Sequence:

```
CALL RDTY (ind,ichr[,ivr])
```

Argument Descriptions:

ind - An integer variable to receive status as follows:

1. =0 character retrieved from buffer is in ichr
2. <0 no data in buffer, overrun, or hardware error

The specific value of ind reflects the error that was detected as follows:

-1 - no data
-2 - overrun
-3 - hardware error

ichr - Logical * 1 or integer variable to receive the terminal data. If an integer is specified, only the low byte is set.

ivr - Optional integer variable to receive the overrun count, or the ICSR contents on the occurrence of a fatal hardware error. Otherwise, set to 0.

NOTE

A task reading the circular buffer should not issue a Wait-For directive until a buffer-empty condition is reported. See Section 19.4.11.11 for an example of how to read circular-buffer entries.

19.4.11.10 **DFTY: Disconnect a Circular Buffer from Terminal Input** - The following routine disconnects a task's circular buffer from terminal inputs.

Calling Sequence:

```
CALL DFTY ([isb][,lun])
```

Argument Descriptions:

isb - Optional, 2-word integer array to receive status in isb(1) as follows. isb(2) is always set to 0.

- +1 - Function successfully completed.
- +4 - Insufficient dynamic storage to allocate an I/O packet.
- +8 - Unassigned LUN.
- +99 - Invalid LUN.
- +322 - Task was not connected to interrupts.

lun - An integer specifying the logical unit number.

19.4.11.11 **Programming Example** - The following are excerpts from a FORTRAN program that is to monitor a remote terminal for input and echo the received characters when a carriage return is detected.

```
C
C     SPECIFY BYTE FORMAT FOR TERMINAL DATA
C
      LOGICAL*1 TCHR
C
C     ALLOCATE STORAGE FOR THE TERMINAL
C     BUFFER
C
      DIMENSION IBUF(32)
C
C     ALLOCATE STORAGE FOR THE PACKED
C     INPUT DATA SO THAT IT IS ALIGNED
C     ON A WORD BOUNDARY
C
      DIMENSION ICHR(40)
      DIMENSION TCHR(80)
      EQUIVALENCE (TCHR,ICHR)
C
C     ALLOCATE STORAGE FOR A
C     2-WORD STATUS BLOCK
C
      DIMENSION ISB(2)
C
C     INITIALIZE ICR11 LOGICAL UNIT(7) AND
C     TRIGGER EVENT FLAG NUMBER(2)
C
      DATA IEV, LUN/2, 7/
           .
           .
           .
```

```
C
C     CONNECT THE TASK TO TERMINAL
C     INPUTS. IF CONNECT FAILS--STOP 1
C
      CALL CTTY (IBUF,32,IEV,ISB,LUN)
      IF (ISB(1).GE.3) STOP 1
C
C     10--POLL THE CIRCULAR BUFFER
C         FOR DATA. ECHO THE LINE WHEN
C         80 CHARACTERS ARE RECEIVED
C         OR A CARRIAGE RETURN IS
C         DETECTED.
C

10    DO 70 I = 1,80
C
C     20--WAIT FOR TRIGGER EVENT FLAG
C
20    CALL WAITFR (IEV)
C
C     30--PACK THE CIRCULAR BUFFER DATA
C         INTO THE BYTE ARRAY
C
30    CALL RDTY (ISB,TCHR(I), IVR)
C
C     DISPATCH ON ERROR CONDITION
C
      GO TO (20,50,40)-ISB
      GO TO 60
C
C     40--REPORT HARDWARE FAULT
C
40    CALL ALARM (IVR)
          .
          .
          .
      GO TO 30
C
C     50--REPORT OVERRUN CONDITION
C
50    CALL LOST (IVR)
          .
          .
          .
      GO TO 30
C
C     60--CHECK FOR CARRIAGE RETURN,
C         EXIT TO ECHO ROUTINE IF
C         PRESENT
C
60    IF (TCHR(I).EQ."15) GO TO 80

70    CONTINUE
C
C     80--FALL THROUGH TO ECHO A LINE
C
      CALL RTOW (I,TCHR,,LUN)
C
C     DISCONNECT TERMINAL BUFFER, EXIT
C
      CALL DFTY (,LUN)
      CALL EXIT
      END
```

The procedure for reading the buffer in the example above may be summarized as follows:

1. Wait for the trigger event flag specified in the call to connect the buffer.

2. Upon regaining control, call the appropriate routine to read the buffer until one of the following terminal conditions is detected:

   a. All data has been read.

   b. An overrun count is detected.

   c. A fatal error is encountered.

3. On the occurrence of 2a or 2b, perform any appropriate processing; then return to scan for additional data.

4. If a hardware error is detected, use the ICSR register contents for further fault analysis and warning as appropriate. In the event of such an error, the event flag is not set by the driver again unless normal service is resumed.

5. A calling task should not execute the Wait-For directive until a buffer-empty condition is detected. This is because your task's buffer pointer is advanced after detecting and clearing an overrun condition, and the trigger-event flag is cleared only when a buffer-empty condition is detected.

19.4.12 Unsolicited Interrupt Processing - Task Activation

The following routines provide the capability of linking a task to an interrupt, soliciting information from the driver concerning how the task was activated, and unlinking a task from all interrupts.

19.4.12.1 **LNK: Link a Task to Interrupts** - This subroutine allows any installed task to be activated on the occurrence of any unsolicited interrupt.

Calling Sequence:

```
CALL LNK (tnam,iprm[,isb][,lun])
```

Argument Descriptions:

tnam - Real variable containing task name in RADIX-50 format.

iprm - A 5-word integer array containing the following data:

iprm(1) - Interrupt class. May be one of the following:

0 - Digital interrupts

1 - Counters

2 - Remote terminal (CTRL/C only)

3 - Error interrupts

iprm(2) - Reserved.

iprm(3) - Optional event flag set if task to be activated is not dormant when the interrupt occurs.

iprm(4) - Hardware-dependent parameters as follows:
iprm(5)

Interrupt Class	Parameter Contents
Digital	iprm(4) = Point number iprm(5) = Change-of-state mask
Counter	iprm(4) = Module number iprm(5) = Counter initial value
Remote Terminal	iprm(4) = not used iprm(5) = not used
Error	iprm(4) = not used iprm(5) = not used

isb - Optional 2-word integer array to receive status in isb(1) as follows. isb(2) is always set to 0.

+1 - Function successfully completed.

+3 - Unrecognized interrupt class specified.

+4 - Insufficient dynamic storage to allocate I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+301 - Task tnam not installed.

+303 - Controller not ready.

+317 - Resource in use. Other task already linked to interrupt.

+323 - Insufficient dynamic memory to allocate secondary control block.

+380 - Task tnam not installed.

+397 - Invalid event flag number specified.

lun - Optional integer specifying the logical unit number.

Example:

Link task ALARM to report fatal hardware errors arising from a malfunction on any ICR11 physical unit.

```
      DIMENSION IPRM(5)
      C
      C  INITIALIZE PARAMETER ARRAY WITH:
      C     1.  INTERRUPT CLASS
      C     2.  RESERVED ELEMENT CLEARED
      C     3.  GLOBAL EVENT FLAG
      C

      DATA IPRM(1), IPRM(2), IPRM(3)/3,0,64/

      DATA ALARM/6RALARM /
           .
           .
           .
      CALL LNK (ALARM,IPRM,,7)
           .
           .
           .
```

19.4.12.2 **RDACT: Read Activation Data** - The following call allows a task to determine the interrupt conditions that caused it to become active.

Calling Sequence:

```
CALL RDACT (iprm[,isb][,lun])
```

Argument Descriptions:

iprm - A 6-word integer array to receive activation data in the following format.

iprm(1) - Activation indicator (see Section 19.3.7.5).

iprm(2) - Physical unit number of ICR.

iprm(3) - Generic code. Set to one of the following values:

0 - Remote terminal

1,2,3 - Digital interrupt

4,5,6 - Counter interrupt

177770 - Fatal hardware error

iprm(4) - Relative module number.

iprm(5) - Hardware-dependent data.
iprm(6)

The following data is returned based upon the type of interrupt module:

Module Type	Generic Code	Parameter Contents
Remote Terminal	0	iprm(5) = terminal input character iprm(6) = undefined
Digital Interrupt	1,2,3	iprm(5) = module data iprm(6) = change-of-state data
Counter	4,5,6	iprm(5) = value of the counter at interrupt iprm(6) = undefined
Error	177770	iprm(5) = contents of ICSR iprm(6) = contents of ICAR.

isb - Optional 2-word integer array to receive status in isb(1) as follows. isb(2) is set to 0.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+306 - iprm array not fully within the task's addressing space.

+319 - Address of iprm is odd.

+379 - Task not linked to ICS/ICR interrupts.

lun - Integer variable specifying the logical unit number.

Example:

The following is an excerpt from a program that reads activating data into array IACT and conditionally exits if the event flag (IEFN) specified in a previous link request, issued by another task, is not set.

```
C
C     ALLOCATE SPACE FOR DATA ARRAY
C
      DIMENSION IACT(6)
            .
            .
            .
10    CALL RDACT (IACT,,7)
            .
            .
            .
C
C     CLOSE ALL FILES
C
      CALL CLOSE(1)
      CALL CLOSE(2)
C
C     EXIT IF TRIGGER EVENT FLAG IS NOT SET
C     ELSE CLEAR EVENT FLAG AND RESTART.
C
      CALL EXITIF (IEFN)
C
C     FLAG WAS SET.  CLEAR IT AND
C     CONTINUE.
C
      CALL CLREF (IEFN)
      GO TO 10
      STOP
      END
```

The foregoing example illustrates the following considerations when a task is made active by ICS/ICR interrupts:

1. To avoid race conditions, use the Exit-If directive to test the state of the event flag and conditionally exit. Issuing a Test Event Flag directive followed by an Exit would cause a flag set condition occurring after the test to go unrecognized.

2. Use of the Exit-If directive bypasses the closure of all files that is normally done by the FORTRAN object time system when the program executes a STOP or CALL EXIT statement. Thus, to exit cleanly, the program must explicitly close all files before invoking the directive.

19.4.12.3 **UNLNK: Remove Interrupt Linkage to a Task** - The following call removes all linkage between a task and ICS/ICR interrupts.

Calling Sequence:

```
CALL UNLNK (tnam,iprm[,isb][,lun])
```

Argument Descriptions:

tnam - Real variable containing task name in Radix-50 format.

iprm - Integer variable containing the interrupt class. May be one of the following:

0 - Digital interrupts

1 - Counters

2 - Remote terminal

3 - Error interrupts

4 - All interrupts

isb - Optional, 2-word integer array to receive the results of the call in isb(1) as follows. isb(2) is set to 0.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+379 - Task not linked to ICS/ICR interrupts.

+380 - Task not installed.

lun - Integer variable specifying the logical unit number.

Example:

Remove the linkage between task ALARM and all ICS/ICR interrupts.

```
DATA ALARM/6RALARM /
CALL UNLNK (ALARM,,,7)
```

19.4.13 Maintenance Functions

The following functions cause the ICS/ICR driver to suppress or enable hardware error reporting while on-line maintenance and troubleshooting is in progress, as described in Section 19.3.9.

OFLIN - Place selected unit off line.

ONLIN - Return selected unit to on-line status.

These calls may be issued only by a privileged task.

19.4.13.1 **OFLIN: Place Selected Unit in Offline Status** - The following call is executed to set a controller off line:

```
CALL OFLIN ([isb] [,lun])
```

Argument Descriptions:

isb - Optional 2-word integer array to receive the results of the call in isb(1) as follows. isb(2) is always 0.

- +1 - Function successfully completed.
- +4 - Insufficient dynamic storage to allocate an I/O packet.
- +8 - LUN not assigned.
- +99 - Invalid LUN.
- +316 - Issuing task not privileged.
- +380 - Device already off line.

lun - Integer variable specifying the ICS/ICR logical unit number.

19.4.13.2 **ONLIN: Return a Device to On-line Status** - The following call returns the selected unit to on-line status.

```
CALL ONLIN ([isb] [,lun])
```

Argument Descriptions:

isb - Optional 2-word integer array to receive the results of the call in isb(1) as follows. isb(2) is always 0.

- +1 - Function successfully completed.
- +4 - Insufficient dynamic storage to allocate an I/O packet.
- +8 - LUN not assigned.
- +99 - Invalid LUN.
- +316 - Issuing task not privileged.

lun - Integer variable specifying the logical unit number.

19.5 ERROR DETECTION AND RECOVERY

Error detection and recovery procedures encompass the following contingencies.

1. Nonrecoverable serial line errors
2. Power-fail at the remote station

3. Power recovery at the processor

4. No response from an interrupting module

The first two conditions are dealt with in a manner similar to that of other types of unsolicited interrupts. Specifically, such occurrences may cause a task to be activated, and are reported to all tasks that are connected to digital, counter, or terminal input. The following paragraphs discuss specific driver activity relating to each error condition.

19.5.1 Serial Line Errors

The driver detects nonrecoverable serial line errors. A nonrecoverable error condition is defined as the occurrence of a predetermined number of error interrupts in an interval of 1 second, or no response from the controller upon initiation of an output data transfer by means of the serial line. The occurrence of such a condition causes the driver to perform as follows:

1. Place the controller in a "not ready" status.

2. Disable further error interrupts.

3. Report the condition to the task that is linked to errors, and to any tasks connected to receive unsolicited interrupt data from the faulty unit. Subsequent QIO requests that transfer data to or from the unit are rejected with a status of IE.DNR.

Requests for interrupting modules that are pending (A/D converters and terminal output) are allowed to time out with the error code IE.DNR. You may specify the serial line error rate required to consider the link inoperative at the time of system generation.

After reporting the error as described above, the driver removes the "not ready" status when the error condition is not detected at the end of any 1-second interval. If requested during system generation, the state of the following remote modules is restored as described.

1. Bistable outputs - set to last recorded state

2. Counters - reinitialized to last initial value

3. Analog outputs - restored to last output value

19.5.2 Power-Fail at a Remote Site

The detection of AC-low from the remote site immediately triggers the processing described in Section 19.5.1. The absence of AC-low returns the unit to the "ready" status.

If specified, the state of the following remote modules is then restored as described:

1. Bistable outputs - set to last recorded state
2. Counters - reinitialized to last initial value
3. Analog outputs - restored to last output value

19.5.3 Power Recovery at the Processor

Power recovery by the processor initiates the activity described in Section 19.5.2 for both local and remote file boxes. However, power recovery processing at the processor is not reported to a task that is linked to error interrupts or connected to receive unsolicited interrupt data.

19.5.4 Unit in Off-line Status

A unit that is off line (see Section 19.3.9.1) is considered to be under manual control for purposes of diagnosis and maintenance. Under these conditions, error reporting as described in Section 19.5.1 is unnecessary and frequently undesirable, because fault indications are generally a by-product of these activities (that is, a remote unit is shut down to install an I/O module), not the result of a genuine controller fault.

Furthermore, to permit the operation of diagnostic software, it is advisable to attempt to service all QIO requests regardless of the controller status. Consequently, under these circumstances, error reporting and detection are modified as follows when the controller is off line:

1. Access to the controller with the intention of transmitting data to or from the device is restricted to privileged tasks.
2. The task linked to error interrupts and any tasks receiving interrupt data are not notified of remote power-fail or fatal serial line errors.
3. All device error interrupts become disabled.
4. An attempt is made to service all QIO requests if issued by a privileged task. If such requests time out (that is, A/D converter or remote terminal output), they are terminated with the error code IE.ABO rather than with IE.DNR. No hardware errors are reported for I/O requests that are normally completed immediately (for example, bistable digital output).

19.5.5 Error Data - ICSR and ICAR Registers

Whenever a reportable error occurs, the driver returns the contents of the appropriate control and status register (ICSR) and, in some cases, the contents of the address register (ICAR), to assist in fault diagnosis. Tables 19-8 and 19-9 describe the contents of these registers.

Table 19-8
ICSR Contents

Bit	Name	Read/ Write	Description
15	OUTPUT BUSY	R	Indicates output buffer cannot accept new data.
14	MAINT	R/W	Maintenance.
13	NOT USED	R	Always set to 1.
12	ERROR	R	Indicates occurrence of communication serial line error. Reset when ICAR is read.
11	MAINT	R/W	Maintenance.
10	PWR FAIL	R	Remote Power Supply AC LO indicator.
9	TBMT INT EN	R/W	Enables bit 15 of ICAR to interrupt.
8	MAINT	R/W	Maintenance.
7	MOD INT	R	Indicates I/O Module requires interrupt servicing.
6	RESET	W	Resets all I/O modules. Always read as 0.
5	TTY ENABLE	R/W	Activates TTY mode, disables I/O mode.
4	PWR FAIL INT ENABLE	R/W	Enables bit 10 to interrupt.
3	BMT INT ENABLE	R/W	Enables complement of bit 15 to interrupt.
2	MOD INT ENABLE	R/W	Enables Bit 7 to interrupt.
1	ERROR INT ENABLE	R/W	Enables Bit 12 to interrupt.
0	RIF	R/W	Resets the interrupting module's flag when set and the module is addressed. This clearing action also resets the RIF bit.

Table 19-9
ICAR Contents

Bit	Name	Description
15	TBMT	Indicates TTY output buffer can accept new data.
14	PCL	Pulse closed. This bit is set by a jumper on a digital interrupt module. This jumper is removed if contact closures are not of interest to your task.
13	POP	Pulse Opened. This bit is set by a jumper on a digital interrupt module. This jumper is removed if contacts opening are not of interest to your task.
12	DA	Indicates terminal character has been received. Cleared by reading terminal character.
11-08	Generic Code	A 4-bit binary code that identifies the type of module requesting the interrupt.
07-00	Module Address	8-bit address of the module requesting the interrupt.

19.6 DIRECT ACCESS

Section 19.1.3 notes those ICS/ICR11 functions that may be performed by referencing a module through its physical address in the I/O page. Under RSX-11M such access is accomplished by one of the following methods:

1. A privileged task or any task running in an unmapped system has unrestricted access to the I/O page and may therefore access each module by absolute address.

2. Using the Task Builder, a task may link to a global common area whose physical address limits span a set of locations in the I/O page. This method applies to either a mapped or unmapped system.

The latter method allows a task to be transported to any other system simply by relinking. Moreover, in a mapped system the memory management hardware aborts all references to device registers outside the physical address limits of the common block.

Because the software allows arbitrary module placement, direct reference, in either case, must be accomplished by translating a relative module number to a physical or virtual register address within the I/O page. This translation or mapping is performed by means of a table (ICTAB.MAC) that is created during system generation, and inserted in the system object module library.

The operations required to implement each method may be summarized as follows:

1. Unrestricted access to the I/O page

   a. Based upon your response to the ICS/ICR system generation queries, the system creates the MACRO source file ICTAB.MAC under UIC [11,10] on the source disk. This file contains tables that describe the physical location of each counter, digital interrupt, and digital sense module in the target system.

   b. ICTAB.MAC is assembled for eventual inclusion in the system object module library.

   c. The MACRO source file ICOM.MAC, under UIC [11,10] on the source disk, is assembled to generate global definitions for the first ICS/ICR address on the I/O page and the number of ICS/ICR controllers in the target system. The resulting object file is incorporated in the system library file.

   d. A task is built containing the appropriate global references. Such references are resolved when the Task Builder searches the system library.

   Steps a, b, and c are executed once. Step d is performed each time a task that references the ICS/ICR11 is created.

   NOTE

   ICS/ICR inputs are not valid until 3ms after power recovery at the processor. Tasks that are referencing inputs directly may establish a power recovery AST entry point that suspends task execution for the necessary time interval.

2. Access to the I/O page through a Global Common Block:

   a. Steps a and b are performed.

   b. File ICOM.MAC under UIC [11,10] is assembled to define the first ICS/ICR module address as a relocatable value, the number of I/O page locations required, and the number of controllers present on the target system.

   c. File ICOM.OBJ, created in step b, is linked using the Task Builder to create an image of the device common block on disk.

   d. The SET and INSTALL MCR or VMR commands allocate space for the common block and declare the block resident in the target system.

   e. A task is created containing the appropriate global references to the common block and mapping table. Common block references are resolved by directing the Task Builder to link the task to the device common block (ICOM). The mapping table reference is resolved from the system library module ICTAB.

The detailed procedure for creating the necessary object files and device common block is performed as part of the system generation process, and is described fully in the RSX-11M System Generation and Installation Guide. Therefore, the discussion in the following paragraphs is limited to procedures for linking to the device common block, and using the file ICTAB.MAC to determine module addresses within the I/O page.

19.6.1 Linking a Task to the ICS/ICR Common Block

Once the device common block has been created, a task may access ICS/ICR modules by linking to the common block. This can be done by using the Task Builder commands shown in the following example:

```
TKB>TASK,LP:=TASK.OBJ
TKB>/
ENTER OPTIONS:
TKB> COMMON=ICOM:RO
TKB>/
```

The illustration is valid for either a mapped or unmapped system. In both cases, the Task Builder links the task to the common block by relocating the global symbol definitions contained in the common block symbol table file ICOM.STB located under UIC [1,1]. If memory management is present, the Executive maps the appropriate physical locations into the task's virtual addressing space when the task is made active.

19.6.2 Accessing the I/O Page

After the task has been linked to the I/O page, either directly or through reference to the device common block, access to a specific ICS/ICR counter, or digital input modules during task execution, is a 3-step process:

1. The task generates a request for module data by specifying module type, relative module number, and physical unit number.

2. The data contained in module ICTAB is accessed to translate the arguments of step 1 to a physical offset from the ICS/ICR base address on the I/O page.

3. The ICS/ICR base address, defined in the common block or system library module that was created from file ICOM.MAC, is added to the offset to compute a physical or virtual address and the module data is read.

The next few paragraphs describe the format of the system library module ICTAB, and common block module ICOM in detail. A sample MACRO subroutine that references these modules is then presented.

19.6.2.1 **Mapping Table Format** - The mapping table created by system generation (file ICTAB.MAC) translates module type, relative module number, and physical unit number for counter, digital interrupt, and digital sense modules, to the physical or virtual address of the module on the I/O page. This module must be assembled and inserted in the system object module library before the standard FORTRAN callable routines can read digital input and counter modules. The table contains one set of entries for each physical unit. The entry sets are arranged in ascending unit number order (Figure 19-1). Entries within each unit are in sequence by module type, as shown in this figure.

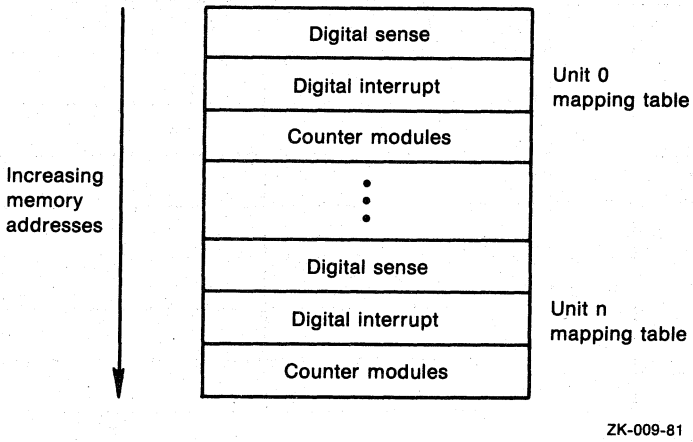


Figure 19-1 Mapping Table Format

The structure of each entry is depicted in Figure 19-2. Entries are 18 bytes long. Byte 0 contains the highest number of modules of a given type that can be referenced for the controller. Bytes 2 through 17, when indexed by relative module numbers, yield a value between 0 and 255 representing the physical location of the module within the set of external page addresses allocated to the ICS/ICR11.

The following global symbols are defined by this module:

.ICTAB = Location of mapping tables

I.CTBL = Length in bytes of one set of entries

19.6.2.2 **I/O Page Global Definitions** - As previously mentioned, module ICOM contains symbolic definitions for I/O page references that are resolved either through unrestricted access or by means of a device common block that is resident on the I/O page. The procedures for implementing either method are carried out during system generation. Upon completion, the following global symbols are defined and later referenced by the FORTRAN callable subroutines:

.ICMD = First ICS/ICR virtual or physical address within the I/O page.

I$$C11 = Number of ICS/ICR controllers

If the global common block was built, the definitions above are contained in the symbol table file that was created by the Task Builder; otherwise, they are included in the system object module library. The definitions are included in module ICOM in the system library or in the STB file ICOM.STB under UIC [1,1] on the system disk. The STB file is referenced by the Task Builder in response to the use of the LIBR keyword.

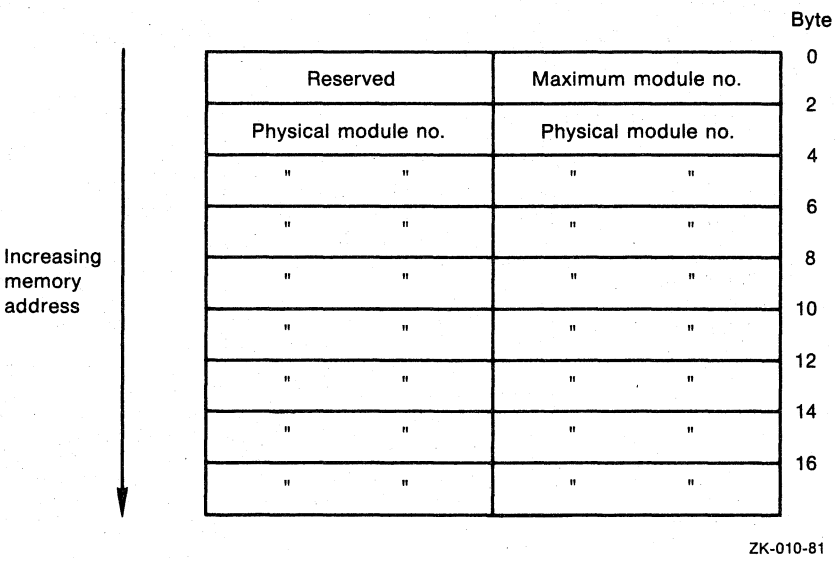


Figure 19-2 Mapping Table Entry Format

19.6.2.3 **Sample Subroutine** - The following subroutine, residing in the system library, uses the modules previously described to read ICS/ICR module data.

```
;
; READ ICS/ICR-11 DIRECT ACCESS INPUTS
;
; LOCAL DATA
;
; ADDRESS OF ICS/ICR-11 MAPPING TABLES
;
        .ENABL  LSB
N=0                                     ;
ICMAP:                                  ;

        .REPT   12.

        .WORD   .ICTAB+I.CTBL*N
N=N+1
        .ENDR

;+
;
```

```
; **-.RDIC-READ ICS/ICR-11 DIRECT ACCESS INPUTS
;
; THIS SUBROUTINE IS CALLED TO TRANSLATE RELATIVE MODULE NUMBER
; TO PHYSICAL EXTERNAL PAGE ADDRESS AND READ THE MODULE DATA.
;
; INPUTS:
;
;       R0 = RELATIVE MODULE NUMBER
;       R1 = MODULE CODE
;
;               WHERE:
;                       0 = CONTACT SENSE
;                       1 = CONTACT INTERRUPTS
;                       2 = COUNTERS
;
;       STACK SETUP IS AS FOLLOWS:
;               (SP)+00 = RETURN TO CALLER
;               (SP)+02 = I/O STATUS BLOCK ADDRESS (NOT
REFERENCED).
;               (SP)+04 = PHYSICAL UNIT NUMBER
; OUTPUTS:
;
;       C/CLEAR
;
;               R0 = MODULE DATA
;
;       C/SET:
;
;               NONEXISTENT PHYSICAL UNIT NUMBER OR MODULE
SPECIFIED
;
;-

.RDIC::                                 ;
        MOV     4(SP),R2                ; GET PHYSICAL UNIT NUMBER
        CMP     #I$$C11-1,R2            ; LEGAL UNIT NUMBER?
        BLO     10$                     ; IF LO NO
        ASL     R2                      ; CONVERT PHYSICAL UNIT NUMBER
TO WORD
                                        ; OFFSET
        MOV     ICMAP(R2),R2            ; GET ADDRESS OF MAPPING TABLE
                                        ; ENTRIES FOR THIS UNIT
        ASL     R1                      ; CONVERT CODE TO WORD OFFSET
        ADD     R1,R2                   ; MULTIPLY OFFSET BY 9 AND ADD
                                        ; TO TABLE ADDRESS
        ASL     R1                      ; ...
        ASL     R1                      ; ...
        ASL     R1                      ; ...
        ADD     R1,R2                   ; COMPUTE OFFSET TO TABLE
        TSTB    (R2)                    ; MODULE EXIST?
        SEC                             ; ASSUME NO
        BEQ     10$                     ; IF EQ NO
        INCB    R0                      ; CONVERT TO NUMBER OF MODULES
        CMPB    (R2)+,R0                ; LEGAL MODULE NUMBER?
        BLO     10$                     ; IF LO NO
        INC     R2                      ; POINT TO TABLE ENTRIES
        ADD     R0,R2                   ; OFFSET TO MODULE NUMBER
        CLR     R0                      ; SET FOR MOVB WITHOUT SIGN
EXTEND
        BISB    (R2),R0                 ; GET INDEX TO MODULE
        ASL     R0                      ; CONVERT TO WORD OFFSET
        MOV     .ICMD(R0),R0            ; GET MODULE DATA
10$:                                    ;
        RETURN                          ;

        .END
```

19.7 CONVERSION OF EXISTING SOFTWARE

The following paragraphs are a guidance in converting existing UDC or ICS software to run under the ICS/ICR-11 driver and associated FORTRAN support routines. The differences described here are restricted to module support and features that would affect existing software. New features, unsupported in previous systems, are not discussed.

19.7.1 Features

Principal features affecting existing software are:

1. Support for the ICS/ICR11 as a multiunit, multicontroller device
2. Removal of software restrictions on the placement of functionally similar modules

Multiunit support affects any software that addresses modules outside the range of a single file box. In general, such software must be modified at the source level.

Unrestricted module placement affects MACRO-11 programs that directly access digital input and counter modules. Such programs may use the library routine described in Section 19.3 to read data from these modules.

19.7.2 Module Support

19.7.2.1 IAD-IA A/D Converter and IMX-IA Multiplexer

MACRO Interface:

Identical to UDC11 driver

FORTRAN Interface:

Same as UDC11

Functional Differences:

The ICS/ICR driver can initiate parallel conversions on each IAD-IA in a file box that is referenced by a single QIO request. The UDC11 driver performs all conversions serially.

The ICS/ICR driver supports any permissible configuration of IAD-IA A/D converters and IMX-IA multiplexers. The UDC11 driver requires that eight module slots be reserved for each IAD-IA in the system regardless of the actual number of multiplexers installed.

19.7.2.2 16-Bit Binary Counter

MACRO Interface:

Identical to UDC11 driver

FORTRAN Interface:

Same as UDC11; however, if the counter is read through a call to RDTI, then the task must be relinked to incorporate the revised FORTRAN Interface routine.

Functional Differences:

The ICS/ICR driver permits any task to reset an initial counter value (with FORTRAN call SCTI or through the IO.ITI QIO function). The UDC11 driver restricts this operation to a task that has connected to counter interrupts.

19.7.2.3 Bistable Digital Output

MACRO Interface:

Identical to UDC11

FORTRAN Interface:

Identical to UDC11

Functional Differences:

None

19.7.2.4 Momentary Digital Output

MACRO Interface:

Your task communicates with the ICS/ICR-11 driver with the QIO IO.MSO. The UDC11 driver does not support this function because the module may be accessed directly through the UDC device common block.

FORTRAN Interface:

Identical to UDC11; however, existing FORTRAN tasks must be relinked to include ICS/ICR11 FORTRAN interface routines.

Functional Differences:

Momentary output operations are now processed by the ICS/ICR driver, rather than through direct access to the I/O page.

19.7.2.5 Noninterrupting Digital Input

MACRO Interface:

MACRO Interface is by means of the ICS/ICR11 device common block and mapping table described in Section 19.6.

FORTRAN Interface:

Identical to UDC11; however, existing tasks must be relinked to include revised ICS/ICR11 FORTRAN interface routines.

Functional Differences:

None

19.7.2.6 Analog Output

MACRO Interface:

Your task communicates with the ICS/ICR driver with the QIO IO.SAO. The UDC11 driver does not support this function because the module may be accessed directly through the UDC device common block.

FORTRAN Interface:

Identical to UDC11; however, existing FORTRAN tasks must be relinked to include ICS/ICR11 FORTRAN interface subroutines.

Functional Differences:

Analog output operations are now processed by the ICS/ICR driver rather than through direct access to the I/O page.

19.7.2.7 Interrupting Digital Input

MACRO Interface:

Identical to UDC11 driver

FORTRAN Interface:

Identical to UDC11 driver; however, if digital inputs are read through the call to RCIPT, then the task must be relinked to incorporate the revised ICS/ICR11 FORTRAN interface routines.

Functional Differences:

None

CHAPTER 20

NULL DEVICE DRIVER

RSX-11M provides a driver for a software construct called the "null device." The mnemonic for the null device is NL:, and its characteristics are as follows:

- A read from NL: returns an end-of-file error (IE.EOF).
- A write to NL: immediately returns success (IS.SUC).

The null device functions as a "black hole" to which your task can direct output, and from which the data so directed never returns. It is particularly useful when you use it with an indirect command file and MCR ASN commands, as in the example below.

Figure 20-1 shows the contents of a Task Builder command file called TESTBLD.CMD. This command file uses symbolic device names for the output file, map file, and input file. These names may be reassigned at task-build time. In particular, the example below assigns the map file to the null device and thus the map file is thrown away.

```
>ASN SY:=OU:

>ASN NL:=MP:

>ASN DK1:=IN:

>TKB @TESTBLD
```

```
OU:TEST,MP:TEST=IN:[200,220]TEST
/
ASG=TI:2
//
```

Figure 20-1 Indirect TKB Command File TESTBLD.CMD.

CHAPTER 21

GRAPHICS DISPLAY DRIVER

21.1 INTRODUCTION

RSX-11M provides support for two graphics display peripherals: the VT11 and the VS60. Graphics display drivers are not supported in RSX-11M-PLUS systems. Each consists of a CRT display, light pen, and display processing unit (DPU). Either may be purchased separately or as part of a complete system. For example, the GT46 is a "starter system" consisting of a VT11 and a PDP-11T/34 with 32K words of memory and disk storage.

21.1.1 VT11 Graphics Display Subsystem

The VT11 is a low-cost, line-drawing graphics display subsystem. It steals cycles asynchronously from the CPU whose UNIBUS it shares. Its DPU instruction set supports the following features:

- Relative and absolute vectors -- solid, long dash, short dash, or dotted
- Point plotting
- Character generation
- Blinking display
- Eight levels of intensity
- Light-pen interaction

21.1.2 VS60 Graphics Display Subsystem

The VS60 supports all these features at a higher rate of performance than the VT11. In addition, the VS60 supports hardware subroutining, scaling, and windowing. A second CRT may be added to the VS60.

21.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains 0s in all bits for graphics display devices. Words 3, 4, and 5 are undefined.

21.3 QIO$ MACRO

Table 21-1 lists the standard and device-specific functions of the QIO macro that are valid for graphics display devices.

The standard QIO functions IO.ATT and IO.DET have little use in the graphics display driver, because the specific functions IO.CON and IO.DIS are available.

Table 21-1
Standard and Device-Specific QIO Functions for Graphics Displays

Format	Function
STANDARD FUNCTIONS	
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
DEVICE-SPECIFIC FUNCTIONS	
QIO$C IO.CON,...,<stadd,size [,lpef][,lpast]>	CONNECT to graphics device (start DPU)
QIO$C IO.CNT,...	Continue (restart DPU)
QIO$C IO.DIS,...	Disconnect from graphics device (halt DPU)
QIO$C IO.STP,...	Stop (halt DPU)

stadd

The starting address of a display file (must be word aligned). The display file must be within the lowest 28K of physical memory for the VT11.

size

The size of the display buffer in bytes.

lpef

The optional number of an event flag to be set upon light-pen hit; in the range 1-64 (10).

lpast

The optional address of an asynchronous system trap (AST) entry point to use upon light-pen hit.

21.4 STATUS RETURNS

Table 21-2 lists error and status conditions that are returned by the graphics display driver in the first word of the I/O status block. The second I/O status word always contains 0.

Table 21-2
Graphics Display Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully.
IE.ABO	**Operation aborted** The I/O operation was canceled by IO.KIL while in progress or in the I/O queue.
IE.CNR	**Connection rejected** The graphics device specified in an IO.CON function was already connected to another task.
IE.DNA	**Device not attached** The graphics device specified in an IO.DET function was not attached to the issuing task.
IE.IEF	**Illegal event flag** An event flag number specified in an IO.CON function (lpef argument) was not in the range 1-64 (decimal).
IE.IFC	**Illegal function code** A function code was specified in an I/O request that is illegal for graphics display devices.
IE.SPC	**Illegal address space** The display buffer specified in an IO.CON function (stadd argument) was not word aligned, or (for VT11 only) was not completely within the lowest 28K of memory.

21.5 PROGRAMMING HINTS

The graphics display driver does not determine what appears on the screen of the VT11 or VS60. The key to what is drawn is the collection of DPU instructions in the display buffer.

Under normal circumstances, the display buffer is generated by calls to a set of FORTRAN graphics subroutines. These subroutines provide a more convenient access to the graphics features of the hardware than do the raw DPU instructions. The subroutines are described in the DECgraphic-11 FORTRAN Reference Manual, order number DEC-11-GFRMA-A-D.

Aborting a VT11 task may cause an RSX-11M system to hang up indefinitely, requiring a bootstrap. The VT11 DPU has no Halt instruction, but the I/O driver must halt the DPU before it can return a success status in response to an IO.KIL request. (IO.KIL is generated when a task is aborted.) This hang situation cannot arise with a VS60.

CHAPTER 22

LABORATORY PERIPHERAL ACCELERATOR DRIVER

22.1 INTRODUCTION

The Laboratory Peripheral Accelerator (LPA11-K) is an intelligent, direct memory access (DMA) controller for DIGITAL's laboratory data acquisition I/O devices. It is a fast, flexible, and easy-to-use microprocessor subsystem that allows analog data acquisition rates up to 150,000 samples per second. The LPA11-K is for applications requiring concurrent data acquisition and data reduction at high rates.

The LPA11-K is supported through a device driver and a set of program-callable routines. The device driver supports multiple controllers and can be configured as resident or loadable. The program-callable support routines are linked with your task at task-build time. These routines are highly modular. Therefore, a particular task need only contain that code necessary for the facilities that it actually uses.

The LPA11-K operates in two distinct modes: dedicated and multirequest. The subsections that follow summarize each mode.

22.1.1 LPA11-K Dedicated Mode of Operation

In dedicated mode, only one task (that is, one request) can be active at a time and only analog I/O data transfers are supported. Up to two analog converters can be controlled simultaneously. Sampling is initiated by an overflow of the real-time clock or by an externally supplied signal.

22.1.2 LPA11-K Multirequest Mode of Operation

In multirequest mode, sampling from all device types is supported. Up to eight user tasks can be simultaneously active. The sampling rate for each user task is a multiple of the common real-time clock rate. Independent rates can be maintained for each task. Both the sampling rate and the device type are specified as part of each data transfer request.

22.2 GET LUN INFORMATION MACRO

If a Get LUN Information system directive is issued for a LUN associated with an LPA11-K, word 2 (the first characteristics word) contains all zeros, words 3 and 4 are undefined, and word 5 contains a 16-bit buffer preset value that controls the rate of the real-time clock interrupts.

22.3 THE PROGRAM INTERFACE

A collection of program-callable subroutines provides access to the LPA11-K. The formats of these calls are fully documented here for FORTRAN programs. MACRO-11 programmers access these same subroutines either through the standard subroutine linkage or through the use of two special-purpose macros. Optionally, MACRO-11 users can control an LPA11-K directly by using device-specific QIO functions. Both FORTRAN and MACRO programs must contain at least one I/O Status Block (IOSB) for retrieval of status information. The following subsections, therefore, describe:

- The FORTRAN interface
- The MACRO-11 interface
- The I/O status block

NOTE

The subroutines documented in this chapter represent the high-level interface to the LPA11-K. Using these subroutines requires an understanding of hardware capabilities, configuration details, and hardware status codes as described in the LPA11-K Laboratory Peripheral Accelerator User's Guide.

22.3.1 FORTRAN Interface

Table 22-1 lists the FORTRAN interface subroutines for accessing the LPA11-K.

The calling sequences of the routines listed in Table 22-1 are compatible with the K-series support routines, described in Chapter 23, except as noted. The following subsections briefly describe the function and format of each FORTRAN subroutine call.

Table 22-1
FORTRAN Subroutines for the LPA11-K

Subroutine	Function
ADSWP	Initiate synchronous A/D sweep
CLOCKA	Set Clock A rate
CLOCKB	Control Clock B
CVADF	Convert A/D input to floating point
DASWP	Initiate synchronous D/A sweep
DISWP	Initiate synchronous digital input sweep
DOSWP	Initiate synchronous digital output sweep
FLT16	Convert unsigned integer to a real constant
IBFSTS	Get buffer status
IGTBUF	Return buffer number
INXTBF	Set next buffer
IWTBUF	Wait for buffer
LAMSKS	Set masks buffer
RLSBUF	Release data buffer
RMVBUF	Remove buffer from device queue
SETADC	Set channel information
SETIBF	Set array for buffered sweep
STPSWP	Stop sweep
XRATE	Compute clock rate and preset

22.3.1.1 **ADSWP: Initiate Synchronous A/D Sweep** - The ADSWP routine initiates a synchronous A/D input sweep through an LPS-11 or an AD11-K (and, if present, the AM11-K).

If differential input is desired for the AD11-K/AM11-K, the channel increment must be set to 2 by calling the SETADC routine. The default channel increment is 1 (single-ended input).

The ADSWP call is as follows:

```
CALL ADSWP (ibuf,lbuf,[nbuf],[mode],[idwell],[iefn],[ldelay],
                [ichn],[nchn],[ind])
```

ibuf

A 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

The size in words of each data buffer. All data buffers must be equal in size and lbuf must be greater than 5. In dedicated mode, lbuf must be at least 257 words.

nbu

The number of buffers to be filled. If nbuf is omitted or set equal to 0, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

The sampling options. The default is 0. The mode bit values listed below that are preceded by a plus sign (+) are independent and can be added or ORed together (assuming that the sampling options are applicable to the mode of operation). Those values not preceded by a plus sign are mutually exclusive and the task can use only one such value at a time. All bit values not listed below are reserved.

The following values can be specified:

0	Absolute channel addressing (default). This mode allows your task to directly access all 64 channels of an A/D converter.
+32	Dual A/D conversion serial/parallel. This option applies to dedicated mode only. It is ignored in multirequest mode.
+64	Multirequest mode. If this value is not specified, the request is for dedicated mode. If the request mode does not match the mode of the hardware (that is, different microcode in the master microprocessor), the LPA11-K rejects the request with an appropriate error code.
+512	External trigger (ST1). Use this mode when you want to use your own external sweep trigger. The external trigger is supplied by a jumper connecting the AD11-K External Start input to the KW11-K Schmitt Trigger 1 output. You can use this external trigger connection only in Dedicated Mode. If you select mode 512, your task must specify a Clock A rate of -1 for proper A/D sampling. This is nonclock-driven sampling.

+1024 Time stamp with Clock B (Multirequest mode only).

+2048 Event marking (Multirequest mode only). LAMSKS must be called to specify an event mark channel and event mark mask.

+4096 Start method. If set, digital input start. If clear, immediate start. LAMSKS must be called to specify a digital start channel and digital start mask (multirequest mode only).

+8192 Dual A/D converter (dedicated mode only).

+16384 Data overrun NON-FATAL/FATAL. If selected, data overrun is considered nonfatal. The LPA11-K defaults to fill buffer 0. (See Section 22.4 for a discussion of buffer management.)

idwell

The number of clock overflows (pulses) between data sample sequences. As an example, if idwell is 20 and nchn is 3, the following occurs: after 20 pulses, one channel is sampled on each of the next three pulses. Then, no sampling takes place for the next 20 pulses. In multirequest mode, this facility permits different sample rates for the same hardware clock rate and preset. In dedicated mode, the clock hardware rate controls sampling and this idwell argument is ignored.

If compatibility with K-series support routines is desired, your task must first establish the clock preset by calling the CLOCKA routine. The sweep start command uses the default idwell value of 1. For the K-series, this procedure sets the rate as desired.

NOTE

This parameter is called iprset in the K-series support routines described in Chapter 23. Its function is different from the idwell parameter described here.

iefn

The event flag (1 to 28, 30 to 96), the name of a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If you select an event flag with iefn, the driver sets the selected event flag as each buffer is filled. Note that the LPA11-K support routines reserve event flag 29 for internal synchronization. If iefn is greater than 96, the driver considers it to be a completion routine that is called with a JSR PC. Such routines must return with an RTS PC (or a FORTRAN RETURN statement).

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors, because error reporting is done through the FORTRAN run-time system
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

If multiple sweeps are initiated, your task should specify different event flags. Adherence to this limitation cannot be enforced by the software.

ldelay

The delay from the start event (DR11-K) until the first sample in IRATE units. This feature is supported in multirequest mode only. Default or 0 indicates no delay.

ichn

The number of the first channel to be sampled. The default of 0 applies only if ichn was not established in a prior call to the SETADC routine.

nchn

The number of channels to sample. The default is 1. nchn may be set up with the SETADC routine. The number of channels specified are sampled at a rate of 1 per clock interrupt. If nchn equals 1, the single channel bit is set in the mode word of the start RDA.

ind

Receives a success or failure code as follows:

1 indicates that the sweep was successfully initialized.

0 indicates an illegal argument list, or SETIBF was not called prior to this call.

-1 indicates a QIO directive failure. The directive error code is placed in IOSB(1) in IBUF.

NOTE

The ind parameter is not supported by the K-series support routines. If compatibility with K-series support routines is desired, this parameter must be ignored.

22.3.1.2 **CLOCKA: Set Clock A Rate** - The CLOCKA routine sets the rate for Clock A. This routine is called as follows:

```
CALL CLOCKA (irate,iprset,[ind],[lun])
```

irate

The clock rate. One of the following must be specified:

-1	Direct-coupled Schmitt Trigger 1 (used only for A/D sweeps in Dedicated Mode +512; not supported by K-series support routines)
0	Clock B overflow or no rate
1	1 MHz
2	100 KHz
3	10 KHz
4	1 KHz
5	100 Hz
6	Schmitt Trigger 1
7	Line frequency

iprset

The two's complement value for clock preset. The clock rate divided by the negative clock preset value yields the clock overflow rate. For example, to obtain a clock overflow rate of 10 Khz with a clock rate of 1Mhz, iprset = -100 (minus 100 decimal). You can use the XRATE routine to calculate a clock preset value.

ind

Receives a success or failure code as follows:

0	indicates an illegal argument list or I/O error. Possible causes are: microcode not loaded; driver not loaded; device off line or not in system.
1	indicates Clock A set to start when sweep requested.

lun

The logical unit number. The default is 7.

22.3.1.3 **CLOCKB: Control Clock B** - The CLOCKB routine gives your task control over the KW11-K Clock B.

The CLOCKB call is as follows:

```
CALL CLOCKB ([irate],iprset,mode,[ind],[lun])
```

irate

The clock rate. When irate is nonzero, the clock is set running at the selected rate after the preset value specified by iprset is loaded. A 0 irate stops the clock. When irate is 0 or default, the iprset and mode parameters are ignored.

The following values are acceptable for irate:

0 Stop clock B
1 1MHz
2 100 KHz
3 10 KHz
4 1 KHz
5 100 Hz
6 Schmitt Trigger 3
7 Line frequency

iprset

The count by which to divide clock rate to yield overflow rate. You can use overflow events to drive clock A. The preset parameter must be specified as 0 or as a negative number in the range -1 to -255. The value in iprset can be established by use of the XRATE routine.

mode

The options. The mode bit values listed below that are preceded by a plus sign (+) are independent and can be added or ORed together. Those values not preceded by a plus sign are mutually exclusive and you can use only one such value at a time. All bit values not listed below are reserved.

1 indicates Clock B operates in noninterrupt mode. The 16-bit clock is not incremented or altered. This allows a greater than 10KHz pulse to be sent to Clock A.

+2 indicates that the feed B to A bit is set in the Clock B status register.

ind

Receives a success or failure code as follows:

0 indicates an illegal argument list or I/O error. Possible causes are: microcode not loaded; driver not loaded; device off line or not in system.

1 indicates Clock B started.

lun

The logical unit number (LUN). The default LUN is 7.

22.3.1.4 **CVADF: Convert A/D Input to Floating Point** - The CVADF routine converts an A/D input value to a floating-point number. The routine can be invoked as a subroutine or a function as follows:

```
CALL CVADF (ival,val)
```

or

```
val = CVADF(ival)
```

ival

A value obtained from A/D input. Bits 12-15 are 0. Bits 0-11 represent the value.

val

(REAL*4) receives the converted value. The converted value is calculated with the following formula:

```
val = (64*ival)/gain
```

22.3.1.5 **DASWP: Initiate Synchronous D/A Sweep** - The DASWP routine initiates synchronous D/A output to an AA11-K.

The DASWP call is as follows:

```
CALL DASWP (ibuf,lbuf,[nbuf],[mode],[idwell],[iefn],ldelay,
                [ichn],[nchn],[ind])
```

ibuf

A 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

The size in words of each data buffer. All data buffers must be equal in size and lbuf must be greater than 5. In dedicated mode, lbuf must be at least 257 words.

nbuf

The number of buffers to be emptied. If nbuf is omitted or set equal to 0, indefinite sweeping occurs. The STPSWP routine terminates indefinite sweeping.

mode

The start criteria. Except where noted, the plus sign (+) preceding mode bit values listed below indicates that they are independent and can be added or ORed together. All bit values not listed below are reserved.

The following values can be specified:

0 indicates immediate start. This is the default.

+64 Multirequest mode. If this value is not specified, the request is for dedicated mode. If the request mode does not match the mode of the hardware (that is, different microcode in master microprocessor), the LPA11-K rejects the request with an appropriate error code.

+4096 Start method. If set, digital input start. If clear, immediate start. LAMSKS must be called to specify a digital start channel and a digital start mask (multirequest mode only).

+16384 Data overrun NON-FATAL/FATAL. If selected, data overrun is considered nonfatal. The LPA11-K empties buffer 0. (See Section 22.4 for a discussion of buffer management.)

idwell

The number of clock overflows (pulses) between data sample sequences. For example, if idwell is 20 and nchn is 3, the following occurs: After 20 pulses, one channel is emptied on each of the next three pulses. Then, no emptying takes place for the next 20 pulses. In multirequest mode, this facility permits different rates for the same hardware clock rate and preset. In dedicated mode, the clock hardware rate controls sampling and idwell in the sweep start command is ignored.

If compatibility with K-series support routines is desired, your task must first establish the clock preset by calling the CLOCKA routine. You must use the default value (1) for the idwell parameter in the sweep start command. For the K-series, this procedure sets the rate as desired. For the LPA11-K, this procedure results in idwell in the sweep call defaulting to 1, thus yielding the same clock rate.

NOTE

This parameter is called iprset in the K-series support routines described in Chapter 23. Its function is different from the idwell parameter described here.

iefn

An event flag number (1 to 28, 30 to 96), or the name of a completion routine, or 0. If you use 0 or default iefn, the driver uses event flag 30 for internal synchronization. If you specify iefn as an event flag, the driver sets the event flag as each buffer is filled. Note that the LPA11-K support routines reserve event flag 29 for internal synchronization. If iefn is greater than 96, it is considered to be a completion routine that is called with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement).

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors, because error reporting is done through the FORTRAN run-time system
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

If multiple sweeps are initiated, your task should specify different event flags. This limitation cannot be enforced by the LPAll driver.

ldelay

The delay from start event (DR11-K) until the first sample in irate units. A minimum delay of 150 microseconds is required (not verified by the LPAll driver). This feature is supported in multirequest mode only.

ichn

The first channel number. Default is channel number 0.

nchn

The number of channels. Default is one channel.

ind

Receives a success or failure code as follows:

1 indicates that the sweep was successfully initialized.

0 indicates an illegal argument list, or SETIBF was not called prior to this call.

-1 indicates a QIO directive failure. The directive error code is placed in IOSB(1) in IBUF.

NOTE

The ind parameter is not supported by the K-series support routines. If compatibility with K-series routines is desired, this parameter must be ignored.

22.3.1.6 **DISWP: Initiate Synchronous Digital Input Sweep** - The DISWP routine initiates a synchronous digital input sweep through a DR11-K. It can be called in multirequest mode only.

The DISWP call is as follows:

```
CALL DISWP (ibuf,lbuf,[nbuf],[mode],[idwell],[iefn],[ldelay],
                [iunit],[nchn],[ind])
```

ibuf

A 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

The size in words of each data buffer. All data buffers must be equal in size and lbuf must be greater than 5.

nbuf

The number of buffers to be filled. If nbuf is 0 or defaulted, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

The sampling options. The default is 0. The plus signs (+) preceding the mode bit values listed below indicate that they are independent and can be added or ORed together. All bit values not listed below are reserved.

The following values can be specified:

0 Immediate start. This is the default mode.

+512 External trigger. The input sampling is triggered by interrupts generated by the DR11-K's external control lines, or its input bits if they are interrupt enabled.

+1024 Time stamped sampling with Clock B. The double word consists of one data word followed by the value of the 16-bit clock at the time of the sample. IOSB(2) contains the number of 2-word samples in the buffer.

+2048 Event marking. LAMSKS must be called to specify an event mark word and an event mark mask.

+4096 Start method. If specified, digital input start. If clear, immediate start. LAMSKS must be called to specify a digital start channel and a digital start mask. The digital start channel need not differ from the input channel (iunit).

+16384 Data overrun NON-FATAL/FATAL. If selected, data overrun is considered nonfatal. The LPA11-K fills buffer 0. (See Section 22.4 for a discussion of buffer management.)

idwell

The number of clock overflows (pulses) between data sample sequences. As an example, if idwell is 20 and nchn is 3, the following occurs: After 20 pulses, one channel is sampled on each of the next three pulses. Then, no sampling takes place for the next 20 pulses. In multirequest mode, this facility permits different sample rates for the same hardware clock rate and preset.

If compatibility with K-series support routines is desired, your task must first establish the clock preset by calling the CLOCKA routine. You must use the default value (1) for the idwell parameter in the sweep start command. For the K-series, this procedure sets the rate as desired. For the LPA11-K, this procedure results in idwell in the sweep call defaulting to 1, thus yielding the same clock rate.

NOTE

This parameter is called iprset in the K-series support routines described in Chapter 23. Its function is different from the idwell parameter described here.

iefn

An event flag number (1 to 28, 30 to 96), or the name of a completion routine, or 0. If you specify 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is a valid event flag, the driver sets the selected event flag as each buffer is filled. Note that the LPA11-K support routines reserve event flag 29 for for internal synchronization. If iefn is greater than 96, it is considered to be a completion routine that is called with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement).

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors, because error reporting is done through the FORTRAN run-time system
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

If multiple sweeps are initiated, your task should specify different event flags. This limitation cannot be enforced by the LPA11 driver.

ldelay

The delay from start event (DR11-K) until the first sample in irate units. Default or 0 indicates no delay.

iunit

The DR11-K unit number. Default is unit number 0. Values 0-4 are valid.

nchn

The number of channels. The LPA11-K treats each DR11-K in its configuration as one channel. Default is 1 channel.

ind

Receives a success or failure code as follows:

1 indicates that the sweep was successfully initialized.

0 indicates an illegal argument list, or SETIBF was not called prior to this call.

-1 indicates a QIO directive failure. The directive error code is placed in IOSB(1) in IBUF.

NOTE

The nchn and ind parameters are not supported by the K-series support routines. If compatibility with K-series support routines is desired, these last two parameters must be ignored.

22.3.1.7 **DOSWP: Initiate Synchronous Digital Output Sweep** - The DOSWP routine initiates a synchronous digital output sweep through a DR11-K. It can be called in multirequest mode only.

The DOSWP call is as follows:

```
CALL DOSWP (ibuf,lbuf,[nbuf],[mode],[idwell],[iefn],ldelay,
                [iunit],[nchn],[ind])
```

ibuf

A 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

The size in words of each data buffer. All data buffers must be equal in size and lbuf must be greater than 5.

nbuf

The number of buffers to be emptied. If nbuf is 0 or default, indefinite emptying occurs. The STPSWP routine terminates indefinite emptying.

mode

The start criteria.

The following values can be specified in the high-order byte of mode:

0	Immediate start. This is the default mode.
+512	External trigger. The output sampling is triggered by interrupts generated by the DR11-K's external control lines or its input bits if they are interrupt enabled.
+4096	Start method. If set, digital input start. If clear, immediate start. LAMSKS must be called to specify a digital start channel and a digital start mask. The digital start channel need not differ from the output channel (iunit).
+16384	Data overrun NON-FATAL/FATAL. If selected, data overrun is considered nonfatal. The LPA11-K fills buffer 0. (See Section 22.4 for a discussion of buffer management.)

idwell

The number of clock overflows (pulses) between data sample sequences. For example, if idwell is 20 and nchn is 3, the following occurs: After 20 pulses, one channel is activated on each of the next three pulses. Then, no output takes place for the next 20 pulses. In multirequest mode, this facility permits different output rates for the same hardware clock rate and preset.

If compatibility with K-series support routines is desired, your task must first establish the clock preset by calling the CLOCKA routine. You must use the default value (1) for the idwell parameter in the sweep start command. For the K-series, this procedure sets the rate as desired. For the LPA11-K, this procedure results in idwell in the sweep call defaulting to 1, thus yielding the same clock rate.

NOTE

This parameter is called iprset in the K-series support routines described in Chapter 23. Its function is different from the idwell parameter described here.

iefn

An event flag number (1 to 28, 30 to 96), or the name of a completion routine, or 0. If you specify 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is a valid event flag, the driver sets the selected event flag as each buffer is emptied. Note that the LPA11-K support routines reserve event flag 29 for internal synchronization. If iefn is greater than 96, it is considered to be a completion routine that is called with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement).

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors, because error reporting is done through the FORTRAN run-time system
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

If multiple sweeps are initiated, your task should specify different event flags. This limitation cannot be enforced by the LPA11 driver.

ldelay

The delay from start event (DR11-K) until the first sample in irate units. A minimum delay of 150 microseconds is required (not verified by the LPA11 driver).

iunit

The DR11-K unit number. Default is unit number 0. Values 0-4 are valid.

nchn

The number of channels. The LPA11-K treats each DR11-K in its configuration as one channel. Default is one channel.

ind

Receives a success or failure code as follows:

1 indicates that the sweep was successfully initiated.

0 indicates an illegal argument list, or SETIBF was not called prior to this call.

-1 indicates a QIO directive failure. The directive error code is placed in IOSB(1) in IBUF.

NOTE

The nchn and ind parameters are not supported by the K-series support routines. If compatibility with K-series support routines is desired, these last two parameters must be ignored.

22.3.1.8 **FLT16: Convert Unsigned Integer to a Real Constant** - The FLT16 routine converts an unsigned 16-bit integer to a real constant (REAL*4). It can be invoked as a subroutine or a function as follows:

```
CALL FLT16 (ival,val)
```

or

```
val=FLT16(ival[,val])
```

ival

An unsigned 16-bit integer.

val

The converted (REAL*4) value.

22.3.1.9 **IBFSTS: Get Buffer Status** - The IBFSTS routine returns information on buffers being used in a sweep.

The IBFSTS call is as follows:

```
CALL IBFSTS (ibuf,istat)
```

ibuf

The 40-word array specified in the call that initiated a sweep.

istat

An array with as many elements as there are buffers involved in the sweep. The maximum is 8. IBFSTS fills each element in the array with the status of the corresponding buffer. The possible status codes are as follows:

+2 indicates that the buffer is in the device queue. That is, RLSBUF has been called for this buffer.

+1 indicates that the buffer is in the task queue. That is, it is full of data (for input sweeps) or is available to be filled (for output sweeps).

0 indicates that the status of the buffer is unknown. That is, it is not the current buffer nor is it in either the device or the user task queue.

-1 indicates that the buffer is currently in use.

22.3.1.10 **IGTBUF: Return Buffer Number** - The IGTBUF routine returns the number of the next buffer to use. This routine should be called by your task's completion routines to determine which is the next buffer to access. Do not use it if an event flag was specified in the sweep-initiating call; if an event flag was specified, use the IWTBUF routine.

IGTBUF can be invoked as a subroutine or a function as follows:

```
CALL IGTBUF (ibuf,ibufno)
```

or

```
ibufno=IGTBUF(ibuf[,ibufno])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

ibufno

Receives the number of the next buffer to access. If there is no buffer in the queue, ibufno contains -1.

On the return from a call to IGTBUF, the following are the possible combinations of ibufno and IOSB contents:

ibufno	IOSB(1)	IOSB(2)	Explanation
n	400(octal)	(Word count)	Normal buffer complete.
n	1	(Word count)	Buffer complete. Sweep terminated. There may be additional buffers in the queue filled and ready for processing.
-1	0	0	No buffers in queue. Request still active.
-1	1	0	No buffers in queue. Sweep terminated.
-1	RSX-11M error code(decimal)	LPA11-K	No buffers in queue. error code(octal) Sweep terminated due to error condition. (Refer to Section 22.3.3 for error code summary.) Note that the error is not returned until there are no more buffers in the task queue.

22.3.1.11 **INXTBF: Set Next Buffer** - The INXTBF routine alters the normal buffer selection algorithm. It allows your task to specify the number of the next buffer to be filled or emptied.

INXTBF can be invoked as a subroutine or a function as follows:

```
CALL INXTBF (ibuf,ibufno[,ind])
```

or

```
ind=INXTBF(ibuf,ibufno[,ind])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

ibufno

The number of the next buffer your task wants filled or emptied. The buffer must already be in the device queue.

ind

Receives an indication of the result of the operation:

0 indicates that the specified buffer was not in the device queue.

1 indicates that the next buffer was successfully set.

22.3.1.12 **IWTBUF: Wait for Buffer** - The IWTBUF routine allows your task to wait for the next buffer to fill or empty. Use IWTBUF with the specification of an event flag in the sweep-initiating call. Do not use this routine if a completion routine was specified in the call to initiate a sweep; when event flags are specified, use the IGTBUF routine.

IWTBUF can be invoked as a subroutine or a function as follows:

```
CALL IWTBUF (ibuf,[iefn],ibufno)
```

or

```
ibufno=IWTBUF(ibuf,[iefn],[ibufno])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

iefn

The event flag on which the task waits. This should be the same event flag as that specified in the sweep-initiating call. If you specify iefn as 0 or default this value, event flag 30 is used.

ibufno

Receives the number of the next buffer to be filled or emptied by your task.

On the return from a call to IWTBUF, the following are the possible combinations of ibufno and IOSB contents:

ibufno	IOSB(1)	IOSB(2)	Explanation
n	400(octal)	(word count)	Normal buffer complete.
n	1	(word count)	Buffer complete. Sweep terminated. There may be additional buffers in the queue filled and ready for processing.
-1	1	0	No buffers in queue. Sweep terminated.
-1	RSX-11M error code(decimal)	LPA11-K	No buffers in queue. error code(octal) Sweep terminated due to error condition. (Refer to Section 22.3.3 for error code summary.) Note that the error is not returned until there are no more buffers in the task queue.

22.3.1.13 **LAMSKS: Set Masks Buffer** - The LAMSKS routine initializes a task buffer containing a LUN, a digital start mask and event mark mask, and channel numbers for the two masks. The routine then assigns the LUN. Each DR11-K is considered to be one channel. Each channel has both input and output capabilities.

LAMSKS must be called if digital input starting or event marking is to be used, or if a LUN other than the default LUN 7 is assigned to LA0. LAMSKS must also be called if your task uses multiple LPA11-Ks. If LAMSKS is to be called, it must be called prior to calling SETIBF. Unlike SETIBF, LAMSKS does not have to be called before each sweep initiation unless one or more parameters are to be changed.

The LAMSKS call is as follows:

```
CALL LAMSKS (lamskb,[lun],[iunit],[idsc],[iemc],[idsw],
             [iemw],[ind])
```

lamskb

A 4-word array.

lun

A logical unit number. Default LUN is 7.

iunit

The physical unit number of the LPA11-K. Default physical unit number is LA0.

idsc

The digital start word channel. Default is channel 0.

iemc

The event mark word channel. Default is channel 0.

idsw

The digital start word mask. Default is 0 (disable digital input starting).

iemw

The event mark word mask. Default is 0 (disable event marking).

ind

Receives a success or failure code as follows:

1 indicates successful initialization.

0 indicates an illegal argument list.

-n indicates a LUN assignment failure. n is the directive error code.

NOTE

If compatibility with K-series support routines is desired, ignore this parameter.

For a discussion of event marking and digital starting, see the LPA11-K Laboratory Peripheral Accelerator User's Guide.

22.3.1.14 **RLSBUF: Release Data Buffer** - The RLSBUF routine declares one or more buffers free for use by the interrupt service routine.

The RLSBUF routine must be called to release buffer(s) to the device queue before the sweep is initiated. The device queue must always contain at least one buffer to maintain continuous sampling. Otherwise, buffer overrun occurs (see Section 22.4 for a discussion of buffer management). Note that RLSBUF does not verify whether the specified buffers are already in a queue.

The RLSBUF call is as follows:

```
CALL RLSBUF (ibuf,[ind],n0[,n1...,n7])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

ind

Receives a success or failure code as follows:

0 indicates illegal buffer number specified, illegal number of buffers specified, or a double buffer overrun has been detected.

1 indicates buffer(s) successfully released.

n0,n1,etc.

The numbers (0-7) of the buffers to be released. A maximum of eight can be specified.

22.3.1.15 **RMVBUF: Remove Buffer from Device Queue** - The RMVBUF routine removes a buffer from the device queue.

The RMVBUF call is as follows:

```
CALL RMVBUF (ibuf,n[,ind])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

n

The number of the buffer to remove.

ind

Receives a success or failure code as follows:

0 indicates that the specified buffer was not in the device queue.

1 indicates that the specified buffer was removed from the queue.

22.3.1.16 **SETADC: Set Channel Information** - The SETADC routine establishes channel start and increment information for all sweeps. The SETIBF routine must be called to initialize the 40-word array (ibuf) before SETADC is called.

If, in the call to SETADC, nchn is 1 or inc is 0, the single channel bit is set in the mode word of the start RDA when the sweep start routine is called.

SETADC can be invoked as a subroutine or a function as follows:

```
CALL SETADC (ibuf,[iflag],[ichn],[nchn],[inc],[ind])
```

or

```
ind = ISTADC(ibuf,[iflag],[ichn],[nchn],[inc],[ind])
```

ibuf

A 40-word array initialized by the SETIBF routine.

iflag

Ignored. It is included for compatibility with K-series support routines.

ichn

The first channel number. Default is 0. If inc equals 0 (or default), ichn is the address of a random channel list. A random channel list is an array of n elements, where each element is a channel number. The final element must have bit 15 set to indicate the end of the list.

nchn

The number of samples to be taken per sequence. Default is one sample.

inc

The channel increment. Default is 1. You should specify an increment of 2 for differential A/D input. If inc equals 0, ichn is an array of random channels to receive input.

ind

Receives a success or failure code as follows:

- 0 indicates an illegal channel number or SETIBF was not called prior to the SETADC call.
- 1 indicates successful recording of channel information for the sweep call.

22.3.1.17 **SETIBF: Set Array for Buffered Sweep** - The SETIBF routine initializes an array required by buffered sweep routines. The SETIBF routine must be called before every call to a buffered sweep routine.

The SETIBF call is as follows:

```
CALL SETIBF (ibuf,[ind],[lamskb],buf0[,buf1...buf7])
```

ibuf

A 40-word array.

ind

Receives a success or failure code as follows:

0 indicates a parameter or buffer error.

1 indicates the array was successfully initialized.

lamskb

The name of a 4-word array. This array allows the use of multiple LPA11-Ks within the same program, because the LUN is specified in the first word of the array. Refer to the description of the LAMSKS routine.

If you want compatibility with K-series software, use the default (LUN 7) lamskb parameter, and LUN 7 is assigned to LA0: in the task-build command file for your task.

buf0, etc.

The name of a buffer. A maximum of eight buffers can be specified. Any buffer names in excess of eight are ignored. At least two buffers must be specified to maintain continuous sampling.

Each buffer specified in the call to SETIBF is assigned a number ranging from 0 to 7.

The assignment of these numbers is based on the order in which buffer names appear in the argument list. The first buffer whose name appears in the list is assigned number 0, the second is assigned number 1, and so forth. In all subsequent calls to other routines involving the set of buffers specified in a call to SETIBF, these numbers, rather than names, refer to particular buffers.

22.3.1.18 **STPSWP: Stop Sweep** - The STPSWP routine stops a sweep that is in progress.

The STPSWP call is as follows:

```
CALL STPSWP (ibuf[,iwhen],[ind])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

iwhen

Specifies when to stop the sweep:

0 stops the sweep, immediately aborting the sweep. This is the default stop method. The sweep is stopped asynchronously by the LPA11-K hardware. When IOSB(1) equals 1, the sweep has been stopped. Call IWTBUF continuously after calling STPSWP until the sweep has actually been stopped. When stopping (aborting) a sweep in this manner, the data contents of the current data buffer cannot be guaranteed.

+n (any positive value) stops the sweep at the end of the current buffer. This is considered to be the normal means for stopping a sweep.

-n (any negative value) is reserved. (Do not use.)

ind

Receives a success or failure code as follows:

1 indicates that the sweep is stopped (at the time indicated by iwhen).

0 indicates an illegal argument list.

-n is a directive error code indicating that the stop sweep QIO failed.

22.3.1.19 **XRATE: Compute Clock Rate and Preset** - The XRATE routine allows your task to compute a clock rate and preset. The clock rate divided by the clock preset yields the desired dwell (intersample interval).

NOTE

You can use the XRATE routine only on systems that have a FORTRAN or BASIC-PLUS-2 compiler. This module is not included with the other LPA11-K support routines in object module format. Rather, it is included in source code format with the K-series source modules in [45,10] on the system disk.

XRATE can be invoked as a subroutine or a function as follows:

```
CALL XRATE (dwell,irate,iprset,iflag)
```

or

```
adwell = XRATE(dwell,irate,iprset,iflag)
```

dwell

The intersample time desired by your task. The time is expressed in decimal seconds (REAL*4).

irate

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

iprset

Receives the clock preset.

iflag

Specifies whether the computation is for Clock A or Clock B:

0 indicates the computation is for Clock A.

nonzero indicates the computation is for Clock B.

adwell

The actual dwell rate for the clock based on the irate and iprset parameters.

22.3.2 MACRO-11 Interface

The MACRO-11 interface to the LPA11-K consists of either the callable routines described in Section 22.3.1 or a set of device-specific QIO functions.

22.3.2.1 **Accessing Callable LPA11-K Support Routines** - MACRO-11 programmers access the LPA11-K support routines through either of two techniques:

1. The standard subroutine linkage mechanism and the CALL op code
2. Special-purpose macros that generate an argument list and invoke a subroutine

These techniques are described in the following subsections.

22.3.2.2 **Standard Subroutine Linkage and CALL Op Code** - LPA11-K routines can be accessed through use of the standard subroutine linkage mechanism and the CALL op code. The format of this procedure is:

```
            .PSECT  code
            MOV     #arglist,R5   ;ARGUMENT ADDRESS TO R5
            CALL    lsubr         ;CALL LPA11-K ROUTINE

            .PSECT  data
arglist:    .BYTE   narg,0        ;NUMBER OF ARGUMENTS
            .WORD   addr1         ;FIRST ARGUMENT ADDRESS
                  .
                  .
                  .
            .WORD   addrn         ;LAST ARGUMENT ADDRESS
```

In this sample, the two PSECT directives are shown only to indicate the noncontiguity of the code and data portions of the linkage mechanism. Within the argument list, any argument that is to be defaulted must be represented by a -1 address (that is, 177777(octal)).

22.3.2.3 **Special-Purpose Macros** - To facilitate the calling of LPA11-K support routines from a MACRO-11 program, two macros are provided in file [45,10]LABMAC.MAC. These macros are:

1. INITS
2. CALLS

INITS is an initialization macro. It must be invoked at the beginning of the MACRO-11 source module.

CALLS invokes an LPA11-K support routine. The format of this macro call is as follows:

```
CALLS lsubr,<arg1,...,argn>
```

lsubr

The name of an LPA11-K support routine.

arg1, and so forth

Arguments to be formatted into an argument list and passed to the routine. Each argument can be either a symbolic name or a constant (interpreted as a positive decimal number) or can be defaulted.

An example showing the use of these macros is as follows:

```
          .TITLE  EXAMPLE
          .IDENT  /01.00/
IBUF:     .BLKW   40.
ISTAT:    .BLKW   5
          INITS             ; INITIALIZATION
            .
            .
            .
            .
            .
START:
            .
            .
            .
            .
            .
;
; FIND STATUS OF 5 SWEEP BUFFERS
; USED IN THE CURRENT SWEEP
;
          CALLS   IBFSTS(IBUF,ISTAT)
            .
            .
            .
            .

          .END    START
```

22.3.2.4 **Device-Specific QIO Functions** - Table 22-2 lists the device-specific functions of the QIO system directive macro that are available for the LPA11-K. Programmers using these functions are entirely responsible for buffer management (refer to Section 22.4) as well as all other interfaces (for example, the request descriptor array). Little (if any) performance improvement over the use of FORTRAN support routines can be expected by using QIOs. Therefore, you should use the routines described in Section 22.3.1.

Table 22-2
Device-Specific QIO Functions for the LPA11-K

QIO Function	Purpose
IO.CLK	Start clock
IO.INI	Initialize LPA11-K
IO.LOD	Load microcode
IO.STA	Start data transfer
IO.STP	Stop request

The MACRO-11 programmer must set up the appropriate Request Descriptor Array (RDA) before the corresponding QIO request is issued. In the case of the IO.STA function (start data transfer), the RDA is set up with buffer virtual addresses. The LPA11-K driver address checks and relocates these buffers, changing them from single-word to double-word addresses. The RDA is fully described in the source code of the driver.

22.3.2.5 **IO.CLK** - The IO.CLK function writes an image into the LPA11-K real-time clock control register and issues a clock start command. The format of the QIO request is:

```
QIO$C IO.CLK,...,<mode,ckcsr,preset>
```

mode

The mode.

ckcsr

The image to be written into the clock control register. To achieve the function of clock rate -1 (see Section 22.3.1.2) for Clock A only, set a clock rate of 0 and set the Schmitt Trigger 1 Interrupt Enable bit in the Clock A Status Register.

preset

The clock preset.

22.3.2.6 **IO.INI** - The IO.INI function initializes the LPA11-K. The task issuing the QIO request must be privileged. The format of the request is as follows:

```
QIO$C IO.INI,...,<irbuf,278.>
```

irbuf

A buffer containing an LPA11-K initialize RDA. The buffer size must be at least 278(decimal) bytes.

22.3.2.7 **IO.LOD** - The IO.LOD function loads a buffer of LPA11-K microcode. The issuing task must be privileged. The function verifies that there are no active tasks for the LPA11-K and resets the hardware. It then loads and verifies the microcode, starts the LPA11-K, and enables interrupts. The function returns to the issuing task when the Ready Interrupt is posted.

The format of the QIO request for the IO.LOD function is as follows:

```
QIO$C IO.LOD,...,<mbuf,2048.>
```

mbuf

A buffer containing microcode to be loaded. The buffer size must be 2048(decimal) bytes.

22.3.2.8 **IO.STA** - The IO.STA function issues an LPA11-K data transfer start command. The format of the QIO request is:

```
QIO$C IO.STA,...,<bufptr,40.>
```

bufptr

A pointer to a buffer containing an LPA11-K sample start RDA. The buffer size must be at least 40(decimal) bytes.

The subfunction codes defined for the IO.STA function are:

Bit 0 = 0 indicates that an AST is to be generated for every buffer (if an AST is specified).

Bit 0 = 1 indicates that an AST is to be generated only for exception conditions.

22.3.2.9 **IO.STP** - The IO.STP function stops a data transfer request. The issuing task must be the same task that initiated the data transfer. The format of the QIO request is as follows:

```
QIO$C IO.STP,...,<userid>
```

userid

The index number associated with the task whose request is to be stopped.

22.3.3 The I/O Status Block (IOSB)

Each active sweep must have its own I/O status block. The I/O status block (IOSB) is a 2-word array allocated in your task. Use it to receive the status of a call to an LPA11-K support routine. When your task calls a data sweep routine, the IOSB is always the first two words of the 40-word array specified as the first argument of the call. The first word of the IOSB contains the status code, and the second word contains the buffer size in words.

NOTE

The LPA11-K driver does not directly use the 2-word IOSB. Instead, the driver uses a 4-word IOSB for internal communications with support routines; this 4-word IOSB is completely transparent to those tasks that use FORTRAN support routines. However, when issuing QIOs, it is the 4-word IOSB that must be referenced.

The first two words of the 4-word IOSB function as a 2-word overall IOSB for returning QIO completion status. The driver returns status such as sweep done, system errors, and LPA11-K hardware errors with this 2-word portion of the IOSB.

The remaining two words function as an intermediate IOSB for passing status information during the data sweeps. MACRO-11 programs using QIO calls always receive the correct 2-word portion of the IOSB in the AST generated by the LPA11-K driver.

The codes that can appear in the first word of an I/O status block are in ISA-compatible format (with the exception of the I/O pending condition). Table 22-3 lists all return codes (except 351; see Section 22.5).

Table 22-3
Contents of First Word of IOSB

IOSB(1)		Meaning
FORTRAN	MACRO	
0	IO.PND	Operation pending; I/O in progress
1	IS.SUC	Successful completion
301	IE.BAD	Invalid arguments
302	IE.IFC	Invalid function code
303	IE.DNR	Device not ready (See Section 22.7)
304	IE.VER	Unrecoverable hardware error caused by power-fail
305	IE.ULN	LUN not assigned to LPA11-K

(continued on next page)

Table 22-3 (Cont.)
Contents of First Word of IOSB

IOSB(1)		Meaning
FORTRAN	MACRO	
306	IE.SPC	Illegal buffer specification
309	IE.DUN	Insufficient UMRs available for request
313 [1]	IE.DAO	Data overrun
315 [1]	IE.ABO	Request terminated; LPA11-K status code in IOSB(2)
316	IE.PRI	Privilege violation
317 [1]	IE.RSU	Resource in use (load microcode only)
320	IE.BLK	Executive blocked driver waiting for UMRs
323	IE.NOD	System dynamic memory exhausted
359 [1]	IE.FHE	Fatal hardware error on device
366	IE.BCC	LPA11-K load microcode error
397	IE.IEF	Invalid event flag specified

1. IOSB(2) contains an LPA11-K status code. Refer to the LPA11-K User's Manual for explanation of status code.

22.4 BUFFER MANAGEMENT

The management of buffers for data transfers by LPA11-K support routines involves the use of two FIFO (First-In, First-Out) queues:

1. The device queue (DVQ)
2. The user task queue (USQ)

The device queue (DVQ) contains the numbers of all buffers that your task has released to the support routines in a call to RLSBUF. The buffers represented by these numbers are ready to be filled with data (input sweeps) or to be emptied of data (output sweeps). Any buffer specified in a call to INXTBF must already be in DVQ.

Your task queue (USQ) contains the numbers of buffers available to your task. For output sweeps, this queue contains the numbers of buffers that have already been emptied by the driver. For input sweeps, the buffers represented by USQ are those which are filled with data. In both instances, your task determines the next buffer to use (that is, it extracts the first element of USQ) by calling IGTBUF or IWTBUF.

Both the DVQ and USQ are initialized to -1 -- indicating no buffers -- when your task calls the SETIBF routine. Your task must call RLSBUF before initiating any sweep, because at least one buffer must be present in DVQ for the first input or output to occur.

For input sweeps, your task should call RLSBUF, specifying the numbers associated with all the buffers to be used in the sweep.

For output sweeps, your task can specify two buffers (for continuous sweeps) in the call to RLSBUF. The first action then taken either in a completion routine or after a call to IWTBUF is to release the next buffer. However, note that this approach does not represent true multiple buffering because data overrun occurs if the second buffer is not released in time.

If a buffer overrun occurs, the LPA11-K normally aborts the affected sweep and returns an appropriate error code. However, the option of having buffer overruns treated as nonfatal error conditions can be selected by specifying the appropriate mode argument in any of the sweep calls. Then, when a buffer overrun occurs, the LPA11-K defaults to buffer 0 for its next data buffer. In this case, the following special considerations regarding buffer management must be observed.

Call RLSBUF before calling any of the sweep control calls. However, if buffer overruns are to be treated as nonfatal conditions, the task should not specify buffer 0 in the initial call to RLSBUF. (It is assumed at the outset that buffer 0 is available for use in this manner and, therefore, should not be released.)

Once a buffer overrun has occurred, the LPA11-K uses buffer 0 and places it on the task's queue just like any other data buffer. At this point, buffer 0 is no longer available for buffer overruns. The task then removes buffer 0 from the task queue by IWTBUF or IGTBUF for possible processing. It is the task's responsibility to release buffer 0 for future buffer overruns by specifying buffer 0 in a call to RLSBUF. Note that the task cannot determine that buffer overrun occurred until it receives buffer 0 from IWTBUF or IGTBUF.

The LPA11-K always uses buffer 0 following a buffer overrun if that condition was specified as nonfatal. Thus, when a second buffer overrun occurs before buffer 0 has been processed and made available for that purpose, a condition called "double buffer overrun" occurs. In this case, buffer 0 is not put on the task queue because the actual contents of buffer 0 cannot be determined at this time, and buffer 0 may actually still be on that queue. The double buffer overrun condition is detected when the task attempts to make buffer 0 available for future buffer overruns with the call to RLSBUF. Note that this is the first time that the task is notified of the condition. If a double buffer overrun condition is detected during the call to RLSBUF, the task must be notified of the condition indicating that the previous processing of buffer 0 contents may have been of no value (the LPA11-K probably changed the buffer's contents while it was being processed).

22.5 LOADING THE LPA-11 MICROCODE

LAINIT is a privileged task that loads all versions of LPA11-K microcode. When called, LAINIT issues an IO.LOD function in a QIO request, followed by IO.INI and IO.CLK function requests. The IO.CLK function starts the clock with a default clock rate of 1 MHz.

During system generation, Phase 1, a command file is generated with LPA11-K support selected through operator response to system generation questions. During system generation, Phase 2, the command file builds LAINIT using additional information obtained through operator response to system generation questions. This information further defines the LPA11-Ks system environment and characteristics for your specific application.

Separate tasks are built during system generation that invoke LAINIT to load appropriate LPA11-K microcode. These tasks are named LAINn, where n corresponds to unit number (starting with unit number 0) for each LPA11-K unit in the system. Thus, you never directly invoke LAINIT.

System generation generates command lines in SYSVMR.CMD that install LAINIT and LAIN0; LAIN1 and subsequent LPA11-K unit-numbered tasks are not included in the command file. Thus, you must install these tasks (if they are required) with VMR or MCR.

Once LAINIT and LAINn tasks have been installed, a particular version of LPA11-K microcode for a specific unit can be loaded by running the corresponding LAINn task. For example:

```
>RUN LAIN2
```

executes LAIN2, loading microcode for LPA11-K unit 2.

When a power-fail recovery occurs, the LPA11-K driver terminates all outstanding activity and requests execution of initiating task(s) (LAINn) for each unit. This provides power-fail recovery for the LPA11-K microprocessor, provided the LAINIT and LAINn tasks are installed. Note that when either the RSX-11M system is bootstrapped or the LPA11-K driver is loaded, a simulated power-fail (resulting in driver power-fail recovery) occurs, loading microcode for each LPA11-K unit. In addition, when the LPA11-K is brought online on an RSX-11M-PLUS system, a simulated power-fail occurs.

If the request for the initiating task (LAINn) fails or the loader fails to load the driver, the LPA11-K unit does not become initialized. Any further attempt to use the LPA11-K fails, with the device not ready (IE.DNR) code returned to the requesting task.

If there is no LPA-11K present at the default address, LAINx returns error code 351 in IOSB(1). This failure occurs if there is more than one LPA-11K and the one at the default address is removed. There must always be an LPA-11K at the default address.

All versions of LAINn set the real-time clock frequency to 1MHz by default. The UCB device characteristics word 4 (U.CW4) contains a 16-bit buffer preset value that controls the rate of ticks (that is, the rate at which the clock interrupts). This value can be set dynamically or during system generation. The quotient resulting when this value is divided into 1 MHz is the rate of ticks. For example, if U.CW4 contains 2, the tick rate is 500kHZ. Your task can issue a Get LUN Information system directive to examine the preset value and the MCR SET /BUF command can modify it while the system is running. This modification takes effect the next time the LPA11-K is reloaded with micro-code by LAINx.

22.6 UNLOADING THE DRIVER

To attain maximum LPA11-K performance, the LPA11-K driver appears not busy to the RSX-11M/M-PLUS Executive. As a result, the potential problem exists that any privileged user can unload the driver while the LPA11-K is servicing other users. Therefore, the privileged user must first determine that the LPA11-K is not being used before he or she unloads the driver.

22.7 TIME-OUT OF THE LPA11-K

The error code IO.DNR means that the LPA11-K timed out while processing your task request. In dedicated mode, this condition can have special meaning.

The LPA11-K driver (LADRV) disables the time-out countdown following LPA11-K acknowledgment of your task request. In all cases in multirequest mode, and in most cases in dedicated mode, this acknowledgment is received almost immediately after your task request is passed to the LPA11-K. The only case when this is not true is when your task requests that a data sweep be started while in dedicated mode. In this case, the LPA11-K waits to transfer the first 256 words of data before acknowledging the sweep request.

If a task is sampling at extremely slow data rates in dedicated mode, the time to transfer that first 256 words may exceed the time-out count for the device. This can be avoided by using the multirequest mode.

If a task must use dedicated mode for high sampling rates, and the start of the sweep is delayed for an extended period of time, the time-out count for the LPA11-K must be disabled. (Refer to the note in LADRV describing this time-out problem and showing where the time-out can be safely disabled for sweep calls.)

NOTE

This procedure disables the detection of real time-outs for sweep calls in dedicated mode.

22.8 22-BIT ADDRESSING SUPPORT

The LPA11-K driver supports 22-bit addressing on systems having that capability. When the system employs 22-bit addressing, certain restrictions are imposed. As a result, tasks written for use with earlier LPA11-K driver versions may not run without modifying your task. These restrictions are discussed in the remainder of this section.

When the LPA11-K driver is executed on 22-bit systems, a certain contiguity of your task's data structures must be established. The task data transfer buffers and the IBUF array must be contiguous. In addition, the task random channel list (if present) and the last data transfer buffer must be contiguous. Thus, the correct sequence for your task data is the IBUF array, followed by the task data transfer buffers, followed by the task random channel list. Failure to structure your task's data in this manner can result in illegal buffer specification errors (IE.SPC) being returned or possible corruption of task address space by data sweeps.

Because the LPA11-K driver can potentially request more buffer space than there is UMR mapping space, a limit must be specified on the total number of UMRs that the LPA11-K driver can use at any time. You specify this limit during system generation, part 1, along with the interrupt vector and CSR address for the LPA11-K.

If a task's UMR requirements cause the total number of UMRs currently in use by the LPA11-K to exceed the limit specified during system generation, the task receives an Insufficient UMRs Available For Request (IE.DUN) error code in IOSB(1) of the IBUF array.

This condition can be avoided by setting the UMR limit to the expected minimum number required for smooth LPA11-K operation for all expected tasks. Because each UMR maps 8K bytes, each task's requirements can be calculated as follows:

- Each IBUF array requires 76(decimal) bytes of UMR mapping.
- Add this result to the byte length of all the contiguous transfer buffers to be used in the sweep.
- Add this result to the byte length of the random channel list (if it exists).
- The number of UMRs your task needs is the total byte count divided by 8192 (8K) and rounded up to the next 8K (if not an exact multiple of 8192).

Because there are only 31 UMRs available for the entire system, it is not desirable to allow the LPA11-K driver (through the limit specified during system generation) to have access to all or nearly all UMRs at any given time. Because other device drivers may also require UMR mapping, the total allocation of UMRs by LADRV can slowly choke a system, and for that reason allocation of UMRs must be carefully considered.

The UMR allocation limit for the LPA11-K can be changed by directly modifying the value in the LPA11-K's UCB word U.LAUB; it is not necessary to do another system generation. Use the OPEN command to access and change the limit to the new value. Possible values can range from 0-31. Then, make the required change, UNLOAD, and then LOAD the LPA11-K driver. If the LPA11-K driver is resident, the value in U.LAUB+2 must also be changed to the new value.

NOTE

Be sure the LPA11-K is idle before attempting to access the UCB.

It is possible for a condition to exist where there may not be enough UMRs available for the Executive to allocate to the driver at the time the request is made, even if the number of UMRs necessary to map your task's request are within the limit specified during system generation. When this happens, the Executive blocks the driver until its UMR request can be granted. Because this condition can introduce sweep timing errors, the current sweep is unconditionally aborted and an appropriate error code (IE.BLK) is returned to the task in IOSB(1).

22.9 SAMPLE PROGRAMS

```
C       LPA11-K SAMPLE PROGRAM
C
C SAMPLE SHOWS THE BASIC FLOW FOR PROGRAMMING THE LPA11-K IN A HIGHER
C LEVEL LANGUAGE.  IT IS EXPECTED THAT YOUR TASK TESTS IOSB RETURNS AND
C ERROR INDICATORS (IND) AS NECESSARY.  SYNCHRONOUS PROGRAM TERMINATION
C IS SUGGESTED. NOTE: THIS SAMPLE PROGRAM DOES NOT EXECUTE CORRECTLY IN
C 22-BIT MODE
C
C       D/A DEDICATED MODE WITH CONTINUOUS SAMPLING
C
C       PROGRAM RUNS 3 LOOPS (BASED ON NCNT).  ON FIRST LOOP,
C       STOPS SYNCHRONOUSLY AT END OF PRESENT BUFFER WHICH HAPPENS
C       TO BE BUFFER #3 BEING FILLED FOR THE 2ND TIME.
C       THE 2ND LOOP TERMINATES ASYNCHRONOUSLY (IWHEN=0).
C       THE 3RD LOOP TERMINATES ASYNCHRONOUSLY ALSO.
C
C
        DIMENSION IBUF(40),IOSB(2),NB(1024,8)
        EQUIVALENCE (IBUF(1),IOSB(1))
        EQUIVALENCE (N0,NB(1,1)),(N1,NB(1,2)),(N2,NB(1,3)),(N3,NB(1,4))
        EQUIVALENCE (N4,NB(1,5)),(N5,NB(1,6)),(N6,NB(1,7)),(N7,NB(1,8))
        CALL CLOCKA (4,-1)
        IWHEN=1
        NCNT=0
2       ICNT=1
    5   CALL SETIBF(IBUF,IND,,N0,N1,N2,N3)
C
C INITIALIZE BUFFERS TO ALL -2'S
C
        DO 10 J=1,8
        DO 10 K=1,1024
   10   NB(K,J)=-2
        CALL RLSBUF(IBUF,IND,1,2,3)
        CALL DASWP(IBUF,1024,,,,20)
20      CALL IWTBUF(IBUF,20,IBUFNO)
        CALL RLSBUF (IBUF,IND,IBUFNO)
        WRITE (1,300) IBUFNO,IOSB(1),IOSB(2),ICNT
        IF (NCNT.EQ.3) GOTO 40
        IF (ICNT.EQ.6) GOTO 2
        ICNT=ICNT+1
        IF (ICNT.NE.4) GOTO 20
        CALL STPSWP (IBUF,IBUFNO)
        IWHEN=0
        NCNT=NCNT+1
        GOTO 20
40      CALL IGTBUF(IBUF,IBUFNO)
        WRITE (1,300) IBUFNO,IOSB(1),IOSB(2),ICNT
300     FORMAT (3X,I10,2O8,I10)
```

```
        STOP
        END
```

The following sample program tests the digital I/O interface of the LPA11-K. It will execute correctly in 22-bit mode.

```
C
C PROGRAM TO TEST DIGITAL INPUT AND OUTPUT FOR LPA11-K
C DIGITAL EQUIPMENT CORPORATION
C
C THIS PROGRAM OUTPUTS A DATA BUFFER TO THE LPA11-K
C DIGITAL I/O INTERFACE AND AT THE SAME INSTANT, FOR EACH SAMPLE
C WORD, READ THE RESULTS BACK. THE DATA BUFFERS ARE COMPARED TO
C MAKE SURE THE TRANSFER IS COMPLETED SUCCESSFULLY.
C
C  ****** NOTE! ******
C       THIS PROGRAM WORKS IF AND ONLY IF THE DIGITAL I/O MODULE
C       UNIT SPECIFIED HAS THE MAINTENANCE JUMPER "WRAP-AROUND"
CABLE
C       INSTALLED !!!!
C
C
C RESERVE STORAGE FOR LPA11-K ROUTINES
C
C       THIS PROGRAM WORKS IN 22-BIT MODE
C
C DATA BUFFERS
C
        INTEGER*2 IBUFI(40),INBUF(300,4)
        INTEGER*2 COMMI(1240)
        EQUIVALENCE(IBUFI(1),COMMI(1))
        EQUIVALENCE(INBUF(1,1),COMMI(41))

        INTEGER*2 IBUFO(40),OUTBUF(300,4)
        INTEGER*2 COMMO(1240)
        EQUIVALENCE(IBUFO(1),COMMO(1))
        EQUIVALENCE(OUTBUF(1,1),COMMO(41))

C RESERVE STORAGE AND EQUIVALENCE FOR RSX I/O STATUS BLOCKS
        LOGICAL*1 INIOS(4),OUTIOS(4)
        EQUIVALENCE (IBUFO(1),OUTIOS(1)),(IBUFI(1),INIOS(1))
C
C SET BUFFER SIZE TO USE FOR THIS REQUEST - MAXIMUM OF 300
WITHOUT
C CHANGING THE DIMENSION STATEMENTS. MUST BE EVEN!
        ISIZE=300
C
C INITIALIZE THE PASS COUNTER FOR THE LOOP
        IPASS=1
C
C SET LPA11-K LOGICAL UNIT NUMBER AND ASSIGN IT TO LA0:
        ILUN=7
        CALL ASSIGN(ILUN,'LA:',0,ISTAT)
        IF(ISTAT .LT. 0)GO TO 100
C
C INITIALIZE THE OUTPUT DATA BUFFER
        DO 2 J=1,4
        DO 2 I=1,ISIZE,2
        OUTBUF(I,J)="125252
        OUTBUF(I+1,J)="052525
2       CONTINUE
C
C STOP LPA11-K REAL TIME CLOCK "A" THIS ENSURES THAT
C NOTHING HAPPENS WHEN WE INITIALIZE THE TWO SWEEPS.
5       CALL CLOCKA(0,0,ISTAT,ILUN)
        IF(ISTAT .NE. 1)GO TO 110
```

```
C
C INITIALIZE THE INPUT DATA BUFFER. ASSUME THE LPA11-K DIGITAL
C I/O INTERFACE IS CONFIGURED IN THE DATA LATCH MODE (AS OPPOSED
C TO SENSE). THUS THE OUTPUT DATA BUFFER MUST CONTAIN A BIT
CHANGE
C FOR EVERY BIT POSITION IN SUCCEEDING DATA WORDS.
        DO 10 J=1,4
        DO 10 I=1,ISIZE
        INBUF(I,J)=0
10      CONTINUE
C
C INITIALIZE DIGITAL OUTPUT SWEEP.  THIS MUST BE DONE BEFORE INIT
C OF DIGITAL INPUT SWEEP! THE LPA11-K PROCESSES THE TRANSFER OF
C DATA IN THE ORDER OF THE SPECIFICATION OF THE SWEEPS.  THUS WE
C WANT TO OUTPUT BEFORE WE INPUT.
        CALL
SETIBF(IBUFO,ISTAT,,OUTBUF(1,1),OUTBUF(1,2),OUTBUF(1,3),
        1   OUTBUF(1,4))
        IF(ISTAT .NE. 1)GO TO 120
C
C RELEASE BUFFER FOR OUTPUT SWEEP
C ALL FOUR BUFFERS -- INDEXES 0,1,2,3 -- ARE RELEASED
        CALL RLSBUF(IBUFO,ISTAT,0,1,2,3)
        IF(ISTAT .NE. 1)GO TO 130
C
C "START" DIGITAL OUTPUT SWEEP. REMEMBER NOTHING HAPPENS UNTIL
C THE REAL TIME CLOCKK STARTS. THE LPA11-K PROCESSES THE REQUEST
C AND BE ALREADY TO TRANSFER DATA WHEN WE RESUME THE CLOCK.
C EVENT FLAG 14 IS SPECIFIED. A DIFFERENT EVENT FLAG MUST BE
C SPECIFIED FOR THE DIGITAL INPUT SWEEP SO THE FORTRAN PROGRAM
C CAN SYNCHRONIZE WITH TWO INDEPENDENT, ASYNCHRONOUS PROCESSES.
        CALL DOSWP(IBUFO,ISIZE,4,0,1,14,30,0)
C
C
C NOW INITIALIZE FOR DIGITAL INPUT SWEEP.  THE SAMPLING
PARAMETERS
C MUST BE THE SAME FOR BOTH THE INPUT AND OUTPUT SWEEP.  WE WANT
C TO WRITE AND READ THE SAME DATA WORD AT THE SAME TIME.
        CALL
SETIBF(IBUFI,ISTAT,,INBUF(1,1),INBUF(1,2),INBUF(1,3),
        1   INBUF(1,4))
        IF(ISTAT .NE. 1)GO TO 140
C
C RELEASE THE INPUT BUFFERS
        CALL RLSBUF(IBUFI,ISTAT,0,1,2,3)
        IF(ISTAT .NE. 1)GO TO 150
C
C "START DIGITAL OUTPUT SWEEP. AGAIN, NOTHING HAPPENS UNTIL
C WE RESUME THE LPA11-K REAL TIME CLOCK.
C EVENT FLAG 15 IS SPECIFIED TO SEPARATE THE INPUT AND OUTPUT
SWEEPS.
        CALL DISWP(IBUFI,ISIZE,4,0,1,15,30,0)
C
C NOW FOR THE BIG EVENT! WE START THE CLOCK AND SEE WHAT HAPPENS.
        CALL CLOCKA(1,-150,ISTAT,ILUN)
        IF(ISTAT .NE. 1)GO TO 150
C
C
C THE LPA11-K SHOULD NOW BEGIN TO TRANSFER DATA
C FIRST WE WAIT FOR THE DIGITAL OUTPUT SWEEP TO FINISH.  IT WAS
C STARTED FIRST AND SHOULD FINISH FIRST.  WE VERIFY THAT IT
C FINISHES CORRECTLY OR CHECK FOR ERRORS.
15      CALL IWTBUF(IBUFO,14,IBUFNO)
C
C IF BUFFER NUMBER IS -1, THEN ERROR
C IF BUFFER NUMBER IS 0,1, OR 2, THEN CONTINUE
```

```
C IF BUFFER NUMBER IS 3, THEN FINISHED
        IF(IBUFNO .LT. 0) GO TO 160
C
C NOW WAIT FOR THE DIGITAL INPUT SWEEP TO FINISH.  THE SAME ERROR
C CONDITIONS APPLY.
        CALL IWTBUF(IBUFI,15,IBUFNO)
        IF(IBUFNO .LT. 0)GO TO 170
        IF(IBUFNO .LE. 2)GO TO 15
C
C THE FACT THAT WE HAVE GOTTEN HERE SAYS THE LPA11-K HAS DONE ITS
C THING.
C CHECK THE INPUT DATA BUFFERS AGAINST THE OUTPUT DATA BUFFERS
        DO 20 J=1,4
        DO 20 I=1,ISIZE
        IF(INBUF(I,J) .NE. OUTBUF(I,J))GO TO 180
20      CONTINUE
C
C SUCCESSFUL COMPLETION, LET EVERYONE KNOW. THEN GO BACK AND DO
IT
C AGAIN.
        WRITE(5,1000)IPASS
1000    FORMAT('  REQUEST COMPLETE!',2X,I6)
        IPASS=IPASS+1
        GO TO 5
C
C REPORT ANY ERRORS THAT HAVE BEEN UNCOVERED IN THE EXAMPLE.
C
100     WRITE(5,1010)ISTAT
1010    FORMAT(//,' ERROR ASSIGNING LUN TO LPA11-K   ',I6)
        CALL EXIT
110     WRITE(5,1020)ISTAT
1020    FORMAT(//,' ERROR STOPPING LPA11-K CLOCKA    ',I6)
        CALL EXIT
120     WRITE(5,1030)ISTAT
1030    FORMAT(//,' ERROR FROM SETIBF - OUTPUT BUFFER   ',I6)
        CALL EXIT
130     WRITE(5,1040)ISTAT
1040    FORMAT(//,' ERROR FROM RLSBUF - OUTPUT BUFFER   ',I6)
        CALL EXIT
140     WRITE(5,1050)ISTAT
1050    FORMAT(//,' ERROR FROM SETIBF - INPUT BUFFER   ',I6)
        CALL EXIT
150     WRITE(5,1060)ISTAT
1060    FORMAT(//,' ERROR FROM RLSBUF - INPUT BUFFER   ',I6)
        CALL EXIT
160     WRITE(5,1070)IBUFNO,(OUTIOS(I),I=1,4)
1070    FORMAT(//,' ERROR FROM DOSWP   ',I2,4(3X,O4))
C
C *** WARNING *** DISWP MIGHT STILL BE ACTIVE WHEN YOU EXIT
C
        CALL EXIT
170     WRITE(5,1080)IBUFNO,(INIOS(I),I=1,4)
1080    FORMAT(//,' ERROR FROM DISWP   ',I2,4(3X,O4))
C
C *** WARNING *** DOSWP MIGHT STILL BE ACTIVE WHEN YOU EXIT
C
        CALL EXIT
180     WRITE(5,1090)I,J,OUTBUF(I,J),INBUF(I,J)
1090    FORMAT(//,' *DATA ERROR* - WORD # ',I4,2X,I4,4X,O6,2X,O6)
        CALL EXIT
        END
```

CHAPTER 23

K-SERIES PERIPHERAL SUPPORT ROUTINES

23.1 INTRODUCTION

K-series laboratory peripheral modules are supported through a set of program-callable routines that are linked with your task at task-build time. These routines are highly modular. Therefore, a particular task contains only that code necessary for the facilities that it actually uses. Additionally, the support routines perform input and output operations through the Connect to Interrupt Vector (CINT$) Executive directive. This directive allows your task to bypass normal QIO processing and perform I/O nearly independent of the Executive.

The following subsections briefly describe the K-series laboratory peripherals, the features provided by the K-series support routines, and the generation and use of these routines.

23.1.1 K-Series Laboratory Peripherals

The K-series peripheral support routines provide single-user, task-level support for the following laboratory peripheral modules:

- AA11-K D/A converter
- AD11-K A/D converter
- AM11-K multiple gain multiplexer
- DR11-K digital I/O interface
- KW11-K dual programmable real-time clock
- AAV11-A D/A converter (LSI-11-bus-compatible)
- ADV11-A A/D converter (LSI-11-bus-compatible)
- DRV11 parallel line unit (LSI-11-bus-compatible)
- KWV11-A programmable real-time clock (LSI-11-bus-compatible)

The maximum supported hardware configuration consists of one KW11-K and sixteen of each of the AA11-K, AD11-K (with optional AM11-K), and DR11-K modules. The minimum configuration, if synchronous sweeps are desired, would be one KW11-K and any one of the three other modules. A single DR11-K supports nonclocked, interrupt-driven I/O sweeps or single digital input or output. A single AD11-K supports single-word A/D input and nonclocked, overflow-driven sampling (provided that the A/D conversion is started with the EXT start input on the AD11-K). An AA11-K supports burst mode output and scope control.

23.1.1.1 **AA11-K D/A Converter** - The AA11-K includes four 12-bit digital-to-analog converters (DACs) and an associated display control. The display control permits your task to display data in the form of a 4096 x 4096 dot array. Under program control, a dot may be produced at any point in this array, and a series of these dots may be programmed sequentially to produce graphical output. The display control may output to chart or X/Y recorder or CRT display unit.

The AAV11-A is an LSI-11-bus-compatible D/A converter with characteristics similar to those of the AA11-K.

23.1.1.2 **AD11-K A/D Converter** - The AD11-K is a 12-bit successive approximation converter that enables your task to sample analog data at specified rates and to store the equivalent digital value for subsequent processing. The basic subsystem consists of an input multiplexer (switch-selectable between 16-channel single-ended or 8-channel differential), sample-and-hold circuitry, and a 12-bit A/D converter. By changing jumpers, the analog inputs can be made bipolar or unipolar.

The ADV11-A is an LSI-11-bus-compatible D/A converter with characteristics similar to those of the AD11-K.

23.1.1.3 **AM11-K Multiple Gain Multiplexer** - The AM11-K is a multiplexer expander that supplements the 16-channel single-end (8 differential) analog input multiplexer in the AD11-K. The expansion is done in three independent groups on the AM11-K. Each group can be set to 16 single-ended or pseudo-differential or 8 differential input channels; each group can have a gain of 1, 4, 16, or 64 assigned to it by a switch in the amplifier.

23.1.1.4 **DR11-K Digital I/O Interface** - The DR11-K is a general-purpose digital input/output interface capable of the parallel transfer of up to 16 bits of data, under program control, between a PDP-11 UNIBUS computer and an external device (or another DR11-K).

The DRV11 is an LSI-11-bus-compatible, general-purpose input/output interface with characteristics similar to those of the DR11-K.

23.1.1.5 **KW11-K Dual Programmable Real-Time Clock** - The KW11-K is a dual programmable real-time clock option that PDP-11 UNIBUS computers use. Features include:

Clock A

- 16-bit counter
- 16-bit programmable preset/buffer register
- Four modes of operation
- Two external inputs (Schmitt triggers)
- Eight clock rates, program selectable
- Five clock frequencies, crystal controlled for accuracy
- Processor actions synchronized to external events

Clock B

- 8-bit counter
- 8-bit programmable preset register
- Repeated interval mode of operation
- One external input (Schmitt trigger)
- Seven clock rates, program selectable
- Five clock frequencies, crystal controlled for accuracy

The KWV11-A is an LSI-11-bus-compatible real-time clock with characteristics similar to those of the KW11-K.

23.1.2 Support Routine Features

The RSX-11M program-callable K-series support routines provide the following features:

- Clock overflow or trigger-driven A/D sweep
- Clock overflow or interrupt-driven digital input sweep
- Clock overflow or interrupt-driven digital output sweep
- Clock overflow or burst mode D/A sweep
- Single digital input
- Single digital output
- Single A/D input

- Scope control
- Histogram sampling
- Schmitt Trigger simulation
- Clock control
- 16-bit software clock
- A/D input to real number conversion
- Buffer control

Immediate digital input or output can be performed at any time. Multiple clock-driven sweeps can be initiated if this optional feature was selected during the K-series generation dialog (see Section 23.1.3.1). Such sweeps, however, are subject to the following restrictions:

1. Regardless of the number of controllers present, there can be only one active A/D sweep at any point in time. The same restriction holds true for D/A sweeps. It is possible, however, to perform digital input and digital output sweeps simultaneously, using the same DR11-K, so long as this feature is selected during the generation dialog.
2. There can be no conflict in clock rates among the sweeps.
3. Only the first sweep can use the delay from start event.
4. The interevent time data-gathering routine cannot run in parallel with any other clock-driven sweeps.

23.1.3 Generation and Use of K-Series Routines

To use K-series support routines, you must do the following three things during system generation:

- Reserve necessary vector space.
- Specify that the CINT$ Executive directive is to be included in the system.
- Specify that AST support is required.

After system generation, you must follow particular procedures for the following:

1. Generation of K-series support routines
2. Program use of K-series routines

These two procedures are detailed in the following subsections.

23.1.3.1 **Generation of K-series Support Routines** - An indirect command file, similar to those that system generation uses, generates the K-series support routine library and other necessary facilities. You invoke this command file by typing the following:

```
>@[200,200]SGNKLAB
```

The dialog initiated by this command determines:

- The device configuration of the subsystem
- The maximum number of buffers that are used on a per-sweep basis
- The inclusion or omission of optional features such as multiple clock-driven sweeps and duplex digital I/O sweeps.

After this information is obtained, the command file creates the following:

1. A prefix file, [45,10]KPRE.MAC, for use during assembly of K-series support routines.
2. A data base file, [45,10]KIODT.MAC, containing control blocks needed to support the devices.
3. A common block file, [45,10]KCOM.MAC, that allows your tasks to access the I/O page. Use this file only on mapped systems.
4. On mapped systems only, two indirect command files:
   a. [45,24]KCOMBLD.CMD, which is a TKB build file for the common block
   b. [1,54]INSKCOM.CMD that installs the common block

At your option, the K-series routines themselves can then be assembled and an object library created. You can specify the name of this library or accept the following default file specification:

```
LB:[1,1]KLABLIB.OLB
```

23.1.3.2 **Program Use of K-series Routines** - The steps required for routine program use of K-series support routines are as follows:

1. Compile or assemble the program. If the task is overlaid, you must ensure that both the buffers used by the K-series support routines and the support routines themselves reside in the root section of the overlay structure.

2. Invoke TKB:

   a. On mapped systems only, use the /PR:0 switch to indicate that the task is privileged.

   b. Include the following indirect command among the responses to the TKB prompt:

```
TKB>@[1,5x]LNK2KLAB
```

      where x is 0 for unmapped systems and 4 for mapped systems.

   c. On mapped systems only, enter the following indirect command in response to the prompt for options:

```
ENTER OPTIONS
@[1,54]LNK2KCOM
      .
      .
      .
//
```

3. On mapped systems only, enter the following indirect command from a privileged terminal before executing the program:

```
>@[1,54]INSKCOM
```

The following is a complete example of the steps previously described:

```
>F4P KTEST,KTEST/-SP=KTEST
>TKB
TKB>KTEST/PR:0,KTEST/-SP=KTEST,[1,1]F4POTS/LB
TKB>@[1,54]LNK2KLAB
TKB>/
ENTER OPTIONS
TKB>@[1,54]LNK2KCOM
   .
   .
   .
TKB>//
```

23.2 THE PROGRAM INTERFACE

A collection of program-callable subroutines provides access to the K-series laboratory peripherals. The formats of these calls are fully documented here for FORTRAN programs. MACRO-11 programmers access these same subroutines either through the standard subroutine linkage or through the use of two special-purpose macros. Both techniques are described in Section 23.2.2. Both FORTRAN and MACRO programs must contain at least one I/O Status Block (IOSB), described in Section 23.2.3, for retrieval of status information.

23.2.1 FORTRAN Interface

Table 23-1 lists the FORTRAN interface subroutines for accessing K-series laboratory peripherals.

Table 23-1
FORTRAN Subroutines for K-series Laboratory Peripherals

Subroutine	Function
ADINP	Initiate single analog input
ADSWP	Initiate synchronous A/D sweep
CLOCKA	Set Clock A rate
CLOCKB	Control Clock B
CVADF	Convert A/D input to floating point
DASWP	Initiate synchronous D/A sweep
DIGO	Digital start event
DINP	Digital input
DISWP	Initiate synchronous digital input sweep
DOSWP	Initiate synchronous digital output sweep
DOUT	Digital output
FLT16	Convert unsigned integer to a real constant
GTHIST	Gather interevent time data
IBFSTS	Get buffer status
ICLOKB	Read 16-bit clock
IGTBUF	Return buffer number
INXTBF	Set next buffer
IWTBUF	Wait for buffer
RCLOKB	Read 16-bit clock
RLSBUF	Release data buffer
RMVBUF	Remove buffer from device queue
SCOPE	Control scope
SETADC	Set channel information
SETIBF	Set array for buffered sweep
STPSWP	Stop sweep
XRATE	Compute clock rate and preset

The calling sequences of the routines listed in Table 23-1 are compatible with the routines for the LPA-11, described in Chapter 22. The following subsections briefly describe the function and format of each FORTRAN subroutine call.

23.2.1.1 **ADINP: Initiate Single Analog Input** - The ADINP routine obtains a single word as input from the A/D converter.

ADINP can be invoked as a subroutine or a function as follows:

```
CALL ADINP ([iflag],[ichan],ival)
```

or

```
ival=IADINP ([iflag],[ichan],[ival])
```

iflag

The gain options:

0 Absolute channel addressing (default). This is the only mode supported on the ADV11 (Q-bus).

1 Sample at a gain of 1. In modes 1, 2, 3, 4, and 5 each AD11-K/AM11-K is treated as 16 channels with channels 17-63 strapped to gains 4, 16, and 64. The 48 multiplexer channels are selected by the software according to the gain specification. Mode values 1, 2, 3, 4, and 5 are not supported on the ADV11 (Q-bus version).

2 Sample at a gain of 4.

3 Sample at a gain of 16.

4 Sample at a gain of 64.

5 Perform auto gain ranging.

ichan

Selects the channel to be sampled. The default is 0.

ival

Receives the sample. The gain bits are inserted if iflag is nonzero.

23.2.1.2 **ADSWP: Initiate Synchronous A/D Sweep** - The ADSWP routine initiates a synchronous A/D input sweep through an AD11-K (and, if present, the AM11-K). The analog input word placed in your task's buffer consists of the 12 bits read from the A/D converter and (except when the mode parameter equals 0) the 2 gain bits read from the A/D status register. A value of 177776(octal) is returned for A/D time-out. A value of 177777(octal) is returned on an A/D conversion error. Such errors are typically caused by conversions occurring too fast.

If differential input is desired, the channel increment must be set to 2 by calling the SETADC routine. The default channel increment is 1 (single-ended input).

NOTE

This routine expects to have the ST1 OUT from the KW11-K or similar trigger jumpered to EXT START on the AD11-K if mode 512 is desired. This also requires the A EVENT OUT from the KW11-K clock trigger jumpered to the KW overflow on the AD11-K if clock driven sweeps are desired.

The format of the ADSWP call is as follows:

```
CALL ADSWP (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[ldelay],
                [ichn],[nchn])
```

ibuf

A 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

The size in words of each data buffer. All data buffers must be equal in size and lbuf must be greater than 0.

nbuf

The number of buffers to be filled. If nbuf is omitted or set equal to 0, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

Sampling options. The default is 0. The mode bit values listed below that are preceded by a plus sign (+) are independent and can be ADDed or ORed together. Those values not preceded by a plus sign are mutually exclusive and you can use only one such value at a time. All bit values not listed below are reserved.

The following values can be specified:

0 Absolute channel addressing (default). This mode allows your task to directly access all 63 channels of an AD11-K/AM11-K combination. This is the only mode that is LPA-11 compatible.

1 Sample with a gain of 1. In modes 1, 2, 3, 4, and 5 each AD11-K/AM11-K is treated as 16 channels with channels 17-63 strapped to gains 4, 16, and 64. The 48 multiplexer channels are selected by the software according to the gain specification. Mode values 1, 2, 3, 4, and 5 are not supported on the ADV11 (Q-bus version).

2 Gain of 4. See also mode value 1.

3 Gain of 16. See also mode value 1.

4 Gain of 64. See also mode value 1.

5 The driver executes auto-gain ranging to return the result with the most significance. Note that using auto-gain ranging may require dual sampling and this impacts performance. See also mode value 1.

7 The driver uses an interrupt routine that you supply. The routine must be named .ADINU and must follow the interrupt service routine coding conventions that this subsystem uses. Refer to the source module KADIN5.MAC for an example of an A/D interrupt routine.

+256 External start (ST1).

+512 Nonclock overflow sampling triggered by ST1.

iprset

The clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE subroutine to calculate a clock preset value. If you omit the iprset argument from the ADSWP call, you must specify a mode value of +512. Otherwise, the driver returns an error status code of 301 (invalid arguments) into the IOSB.

iefn

The event flag (1-96), a completion routine, or 0. If 0 or defaulted, event flag 30 is used for internal synchronization. If iefn is an event flag (1-96), the selected event flag is set as each buffer is filled. If iefn is greater than 96, it is considered to be a completion routine that is called with a JSR#PC. Such routines must return with an RTS PC (or a FORTRAN RETURN statement). Furthermore, FORTRAN completion routines must not do any I/O through the FORTRAN runtime system, because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. Adherence to this limitation cannot be enforced by the software.

ldelay

The delay from the start event (ST1) until the first sample in IRATE units. Default or 0 indicates no delay.

ichn

The number of the first channel to be sampled. The default of 0 applies only if ichn was not established in a prior call to the SETADC routine.

nchn

The number of channels to sample. The default is 1. nchn may be set up with the SETADC routine. All nchn channels are sampled on one clock interrupt.

23.2.1.3 **CLOCKA: Set Clock A Rate** - The CLOCKA routine sets the rate for Clock A. The format of the call to this routine is as follows:

```
CALL CLOCKA (irate,iprset,[ind],[lun])
```

irate

The clock rate. One of the following must be specified:

0	Clock B overflow (not on Q-bus version) or no rate
1	1 MHz
2	100 KHz
3	10 KHz
4	1 KHz
5	100 Hz
6	Schmitt Trigger 1
7	Line frequency

iprset

The clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE routine to calculate a clock preset value. The 2s complement of this value is the one that you must supply.

ind

Receives a success or failure code as follows:

0	indicates illegal arguments.
1	indicates Clock A set to start when sweep requested.

lun

The logical unit number. Present for LPA-11 compatibility. Ignored by K-series software.

23.2.1.4 **CLOCKB: Control Clock B** - The CLOCKB routine gives you control over the KW11-K Clock B, which maintains a 16-bit software clock. This feature is not available on LSI-11-bus versions. The 16-bit clock is incremented once per Clock B interrupt. The maximum value of the clock is 65535.

The format of the call to CLOCKB is as follows:

```
CALL CLOCKB ([irate],[iprset],[mode],[ind],[lun])
```

irate

The clock rate. When irate is nonzero, the clock is set running at the selected rate after the preset value specified by iprset is loaded. The 16-bit software clock is not altered by starting the clock. The initial value of the 16-bit clock is 0 when the program is loaded.

When irate is 0, clock B is stopped but the 16-bit software clock is unaltered.

When irate is defaulted, the 16-bit software clock is zeroed but clock B continues to run.

The following are the acceptable values for irate:

0	Stop Clock B
1	1MHz
2	100 KHz
3	10 KHz
4	1 KHz
5	100 Hz
6	Schmitt Trigger 3
7	Line frequency

iprset

The count by which to divide clock rate to yield overflow rate. You can use overflow events to maintain the 16-bit software clock or drive clock A, or both. The default value is 1. The maximum practical overflow rate in interrupt mode is 10 KHz. The range of iprset is 1-255. The value in iprset can be established by use of the XRATE routine.

mode

The options. Either of the following can be specified:

0 indicates normal operations. This is the default. The 16-bit software clock is updated on Clock B overflow. The overflow rate should not exceed 10KHz. The software does not check the overflow rate.

1 indicates Clock B operates in noninterrupt mode. The 16-bit clock is not incremented or altered. This allows a greater than 10KHz pulse to be sent to clock A.

ind

Receives a success or failure code as follows:

0 indicates a failure to start Clock B.

1 indicates Clock B started.

lun

The logical unit number. This argument is ignored by the K-series routines. It is present for LPA-11 compatibility.

23.2.1.5 **CVADF: Convert A/D Input to Floating Point** - The CVADF routine converts an A/D input value to a floating-point number. The routine can be invoked as a subroutine or a function as follows:

```
CALL CVADF (ival,val)
```

or

```
val = CVADF(ival)
```

ival

A value obtained from A/D input. Bits 12-15 are the gain. Bits 0-11 represent the value.

val

(REAL*4) receives the converted value.

23.2.1.6 **DASWP: Initiate Synchronous D/A Sweep** - The DASWP routine initiates synchronous D/A output to an AA11-K.

The format of the DASWP call is as follows:

```
CALL DASWP (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[ldelay],
                [ichn],[nchn])
```

ibuf

A 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

The size in words of each data buffer. All data buffers must be equal in size and lbuf must be greater than 0.

nbuf

The number of buffers to be emptied. If nbuf is omitted or set equal to 0, indefinite emptying occurs. The STPSWP routine terminates indefinite emptying.

mode

The start criteria. Except where noted, the plus sign (+) preceding mode bit values listed below indicates that they are independent and can be added or ORed together. All bit values not listed below are reserved.

The following values can be specified:

0 indicates immediate start. This is the default.

1 indicates that a group of data words, whose number is specified by nchn, is preceded by a scope control word (refer to Section 23.2.1.22 for a description of scope control words). This bit setting is ignored if +512 is also specified. This feature is not included in the Q-bus (AAV11) version.

The buffer size specified by lbuf must be a multiple of nchn+1 words. The DASWP routine, however, does not enforce this restriction.

2 sets the intensify bit after each pair of channels (nchn must be 2) have been output. This feature is supported on the Q-bus version only. It assumes that bit 0 of DAC3 on the AAV11 is connected to the intensify input on the oscilloscope.

+256 indicates external start (ST1).

+512 indicates non-clock-overflow, non-interrupt-driven output (burst mode). This value cannot be specified with either external start (+256) or a nonzero ldelay value. A completion routine must be specified if nbuf is greater than the number of buffers supplied or if continuous burst output is desired. If nbuf equals -1, burst mode must be stopped by calling STPSWP from the completion routine.

iprset

The clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE subroutine to calculate a clock preset value.

If the iprset argument is omitted, you must specify a mode value of +512. Otherwise, an error status code of 301 (invalid arguments) is returned into the IOSB.

iefn

An event flag number (from 1 to 96), or a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is an event flag from 1 to 96, the driver sets the selected event flag as each buffer is emptied. If iefn is greater than 96, the driver considers it a completion routine and calls it with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement). Furthermore, FORTRAN completion routines must not perform I/O through the FORTRAN run-time system because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. This limitation cannot be enforced by the software.

ldelay

The delay from start event (ST1) until the first sample in irate units. Default or 0 indicates no delay.

ichn

The first channel number. The default is 0.

nchn

The number of channels. The default is 1. When nchn equals 2 and mode does not contain +1, the size of data buffers specified in lbuf must be an even number. The software does not check this requirement.

23.2.1.7 **DIGO: Digital Start Event** - The DIGO routine allows you to specify the digital input bits that, when set, causes the simulation of an external start event and the start of a pending sweep.

The format of the call to DIGO is:

```
CALL DIGO([iunit],[mask],[kount])
```

iunit

The DR11-K unit number. The default is 0.

mask

A logical mask that specifies one or more start bits. If zero, a pending digital start event request is immediately canceled. If defaulted, an ST1 event is immediately simulated and the current value of the 16-bit software clock is returned in kount, if specified.

kount

Receives the current value of the 16-bit software clock when the defaulting of mask causes the simulation of an ST1 event.

23.2.1.8 **DINP: Digital Input** - The DINP routine inputs a single 16-bit word from a DR11-K. Bits read as a 1, can be masked with a 1, causing the clearing of the bit in the DR11-K input buffer.

During the K-series routines generation dialog, it is possible to select one of two versions of the DINP routine:

1. A slow version containing all functions described below
2. A fast version that omits the functions provided by the mask, iosb, and input arguments

The fast version of DINP can be invoked as a function (IDINP) only. The slow version of DINP can be invoked as a subroutine or a function. The formats of the invocations are as follows:

```
CALL DINP ([iunit],[mask],iosb,input)
```

or

```
ind=IDINP(iunit,[mask],iosb,[input])
```

iunit

The DR11-K unit number. This argument is required for the fast version of DINP. For the slow version, the default is 0.

mask

The bit mask that specifies which input bits are cleared in the digital input register. The default is 177777(octal) indicating all bits are cleared.

iosb

A 2-word I/O status block array (see Section 23.2.3).

input

Receives the data input from the DR11-K.

ind

Receives the data input from the DR11-K if DINP is invoked as a function.

23.2.1.9 **DISWP: Initiate Synchronous Digital Input Sweep** - The DISWP routine initiates a synchronous digital input sweep through a DR11-K.

The format of the call to DISWP is:

```
CALL DISWP (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[ldelay],
                [iunit])
```

ibuf

A 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

The size in words of each data buffer. All data buffers must be equal in size and lbuf must be greater than 0.

nbuf

The number of buffers to be filled. If nbuf is 0 or defaulted, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

The sampling options. The default is 0. The plus signs (+) preceding the mode bit values listed below indicate that they are independent and can be added or ORed together.

The following values can be specified:

0 Single-word sample, immediate start. This is the default mode.

+256 External start (ST1).

+512 Nonclock overflow interrupt-driven input. External start and delay are illegal.

+1024 Time-stamped sampling. The double word consists of one data word followed by the value of the 16-bit software clock at the time of the sample. This option is not available if you are not using the KW11-K clock (for example, on the Q-bus).

iprset

The clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE subroutine to calculate a clock preset value.

If the iprset argument is omitted, you must specify a mode value of +512. Otherwise, an error status code of 301 (invalid arguments) is returned into the IOSB.

iefn

An event flag number (from 1 to 96), or a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If you use iefn as an event flag from 1 to 96, the driver sets the selected event flag as each buffer is filled. If iefn is greater than 96, the driver considers it a completion routine and calls it with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement). Furthermore, FORTRAN completion routines must not perform I/O through the FORTRAN run-time system because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. This limitation cannot be enforced by the software.

ldelay

The delay from start event (ST1) until the first sample in irate units. Default or 0 indicates no delay.

iunit

The DR11-K unit number. The default is 0.

23.2.1.10 **DOSWP: Initiate Synchronous Digital Output Sweep** - The DOSWP routine initiates a synchronous digital output sweep through a DR11-K.

The format of the call to DOSWP is as follows:

```
CALL DOSWP (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[ldelay],
                [iunit])
```

ibuf

A 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

The size in words of each data buffer. All data buffers must be equal in size, and lbuf must be greater than 0.

nbuf

The number of buffers to be emptied. If nbuf is 0 or defaulted, indefinite emptying occurs. The STPSWP routine terminates indefinite emptying.

mode

The start criteria. The default is 0.

The following values can be specified in the high-order byte of mode:

0	Immediate start. This is the default.
+256	External event start (ST1).
+512	Nonclock overflow, interrupt-driven output. External start and delay are illegal.

iprset

The clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE subroutine to calculate a clock preset value.

If the iprset argument is omitted, you must specify a mode value of +512. Otherwise, an error status code of 301 (invalid arguments) is returned into the IOSB.

iefn

An event flag number (from 1 to 96), or a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is an event flag from 1 to 96, the driver sets the selected event flag as each buffer is emptied. If iefn is greater than 96, the driver considers it a completion routine and calls it with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement). Furthermore, FORTRAN completion routines must not perform I/O through the FORTRAN run-time system because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. This limitation cannot be enforced by the software.

ldelay

The delay from start event (ST1) until the first sample in irate units. Default or 0 indicates no delay.

iunit

The DR11-K unit number. The default is 0.

23.2.1.11 **DOUT: Digital Output** - The DOUT routine outputs a single 16-bit word to a DR11-K. Only those bits in the output word specified by corresponding ones in a mask field are altered.

During the K-series routines generation dialog, it is possible to select one of two versions of the DOUT routine:

1. A slow version containing all functions described below
2. A fast version that omits the functions provided by the mask and iosb arguments

The slow version of DOUT can be invoked as a subroutine or a function. The fast version of DOUT can be invoked as a subroutine only. The formats of the invocations are as follows:

```
CALL DOUT ([iunit],[mask],iosb,idata)
```

or

```
iout=IDOUT([iunit],[mask],iosb,idata)
```

iunit

The DR11-K unit number. The default is 0.

mask

Selects which bits can be altered. The default is 177777(octal), indicating all bits.

iosb

A 2-word I/O status block (see Section 23.2.3).

idata

The 16-bit output value for the DR11-K. A 1 sets a corresponding bit. A 0 clears the corresponding bit.

iout

Receives a copy of the DR11-K output register after it has been altered.

23.2.1.12 **FLT16: Convert Unsigned Integer to a Real Constant** - The FLT16 routine converts an unsigned 16-bit integer to a real constant (REAL*4). It can be invoked as a subroutine or a function as follows:

```
CALL FLT16 (ival,val)
```

or

```
val=FLT16(ival[,val])
```

ival

An unsigned 16-bit integer.

val

The converted (REAL*4) value.

23.2.1.13 **GTHIST: Gather Interevent Time Data** - The GTHIST routine initiates sampling to measure the elapsed time between events. The value of the Clock A buffer/preset register at the time of ST2 firing is stored in a task buffer that you provide.

GTHIST is an optional facility that must be explicitly selected during the K-series generation dialog prior to its use in any program. The format of the call to GTHIST is as follows:

```
CALL GTHIST (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[kount])
```

ibuf

A 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

The size in words of each data buffer. All data buffers must be equal in size and lbuf must be greater than 0.

nbuf

The number of buffers to be filled. If nbuf is 0 or defaulted, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

The sampling options as follows:

0 indicates external event timing without Zero Base. This is the default.

1 indicates external event timing with Zero Base. This is the only mode supported for the KWV11.

iprset

A null argument. It is present only to maintain compatibility with other sweep routine calling sequences.

iefn

An event flag number (from 1 to 96), or a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is an event flag from 1 to 96, the driver sets the selected event flag as each buffer is filled. If iefn is greater than 96, the driver considers it a completion routine and calls it with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement). Furthermore, FORTRAN completion routines must not perform I/O through the FORTRAN run-time system because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. This limitation cannot be enforced by the software.

kount

A counter used by GTHIST, as described below.

To take Post-Stimulus Time data, set mode to 0. ST1 signals the occurrence of a stimulus and starts the clock (that is, no data is taken until the first ST1 occurs). Each response is signaled by ST2, and the buffer/preset register contents are placed in your task's buffer. Each ST1 resets the counter register to 0, and increments kount by 1. Thus, kount keeps track of the number of stimuli (ST1 events). Clock overflow stops the clock. The clock waits for the next ST1 event before restarting. The maximum stimulus-response interval is a function of the clock rate.

To obtain Inter-Stimulus-Interval data, set mode to 1. The time between successive events, as signaled by ST2, is recorded. The maximum interevent time is a function of the clock rate. When clock overflow occurs, the value returned on the next ST2 firing is 177777(octal) and KOUNT is incremented. Thus, kount represents the number of times the maximum interevent time was exceeded. In general, the user should ignore values of 177777(8).

23.2.1.14 **IBFSTS: Get Buffer Status** - The IBFSTS routine returns information on buffers that the driver is using in a sweep.

The format of the call to IBFSTS is as follows:

```
CALL IBFSTS (ibuf,istat)
```

ibuf

The 40-word array specified in the call that initiated a sweep.

istat

An array with as many elements as there are buffers involved in the sweep. The maximum is 8. IBFSTS fills each element in the array with the status of the corresponding buffer. The possible status codes are as follows:

+2 indicates that the buffer is in the device queue. That is, it is waiting to be filled or emptied.

+1 indicates that the buffer is in the task queue. That is, it is full of data (for input sweeps) or is waiting to be filled (for output sweeps).

0 indicates that the status of the buffer is unknown. That is, it is not the current buffer nor is it in either the device or the user task queue.

-1 indicates that a service routine is currently using the buffer.

23.2.1.15 **ICLOKB: Read 16-bit Clock** - The ICLOKB function returns the contents of the 16-bit software clock as an integer value to your task.

The format of the ICLOKB function call is as follows:

```
itim=ICLOKB(0)
```

itim

Receives the current value of the 16-bit software clock as an unsigned integer.

NOTE

MACRO-11 programmers need not establish an argument block for the ICLOKB function. The current value of the software clock is returned in R0.

23.2.1.16 **IGTBUF: Return Buffer Number** - The IGTBUF routine returns the number of the next buffer to use. This routine should be called by your task's completion routines to determine the next buffer to access. Do not use it if an event flag was specified in the sweep-initiating call. Rather, use the IWTBUF routine with event flags.

IGTBUF can be invoked as a subroutine or a function. The formats of the invocations are:

```
CALL IGTBUF (ibuf,ibufno)
```

or

```
ibufno=IGTBUF(ibuf[,ibufno])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

ibufno

Receives the number of the next buffer to access. If there is no buffer in the queue, ibufno contains -1.

23.2.1.17 **INXTBF: Set Next Buffer** - The INXTBF routine alters the normal buffer selection algorithm. It allows your task to specify the number of the next buffer to be filled or emptied.

INXTBF can be invoked as a subroutine or a function. The formats of the invocations are:

```
CALL INXTBF (ibuf,ibufno[,ind])
```

or

```
ind=INXTBF(ibuf,ibufno[,ind])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

ibufno

The number of the next buffer task wants filled or emptied. The buffer must already be in the device queue.

ind

Receives an indication of the result of the operation:

- 0 indicates that the specified buffer was already active or was not in the device queue.
- 1 indicates that the next buffer was successfully set.

23.2.1.18 **IWTBUF: Wait for Buffer** - The IWTBUF routine allows your task to wait for the next buffer to fill or empty. Use it with an event flag specified in in the sweep-initiating call. Do not use this routine if you specified a completion routine in the call to initiate a sweep. Rather, use the IGTBUF routine with completion routines.

IWTBUF can be invoked as a subroutine or a function. The formats of the invocations are as follows:

```
CALL IWTBUF (ibuf,[iefn],ibufno)
```

or

```
ibufno=IWTBUF(ibuf,[iefn],[ibufno])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

iefn

The event flag for which the task waits. This should be the same event flag as that specified in the sweep-initiating call. If iefn equals 0 or is defaulted, the driver uses event flag 30.

ibufno

Receives the number of the next buffer to be filled or emptied by your task.

23.2.1.19 **RCLOKB: Read 16-bit Clock** - The RCLOKB routine returns to your task the contents of the 16-bit software clock as a real constant.

RCLOKB can be invoked as a subroutine or a function as follows:

```
CALL RCLOKB (rlast,time)
```

or

```
time=RCLOKB(rlast,time)
```

time

Receives the current value of the 16-bit software clock as a real constant (REAL*4).

rlast

A value (REAL*4) to be subtracted from the current 16-bit software clock before it is returned into the time field.

23.2.1.20 **RLSBUF: Release Data Buffer** - The RLSBUF routine declares one or more buffers free for use by the interrupt service routine.

The RLSBUF routine must be called to release buffer(s) to the device queue before the sweep is initiated. The device queue must always contain at least one buffer to maintain continuous sampling. Otherwise, buffer overrun occurs (see Section 23.3 for a discussion of buffer management). Note that RLSBUF does not verify whether the specified buffers are already in a queue.

The format of the call to RLSBUF is as follows:

```
CALL RLSBUF (ibuf,ind,n0[,n1...,n7])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

ind

Receives a success or failure code as follows:

0 indicates illegal buffer number specified.

1 indicates buffer(s) successfully released.

n0,n1,and so forth

The numbers of buffers to be released. A maximum of eight can be specified.

23.2.1.21 **RMVBUF: Remove Buffer from Device Queue** - The RMVBUF routine removes a buffer from the device queue.

The format of the call to RMVBUF is as follows:

```
CALL RMVBUF (ibuf,n[,ind])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

n

The number of the buffer to remove.

ind

Receives a success or failure code as follows:

0 indicates that the specified buffer was not in the device queue.

1 indicates that the specified buffer was removed from the queue.

23.2.1.22 **SCOPE: Control Scope** - The SCOPE routine allows you to control the status register of an AA11-K.

The format of the call to SCOPE is as follows:

```
CALL SCOPE (iunit,icntrl,iosb)
```

iunit

The AA11-K unit number.

icntrl

A combination of bit values as shown in Table 23-2. Any bits not listed in this table are cleared before output to the AA11-K status register

iosb

A 2-word I/O status block (see Section 23.2.3).

Table 23-2
Scope Control Word Values

Decimal Value	Octal Value	Function
4096	10000	Erase storage CRT
2048	4000	Set write-through mode
1024	2000	Set store mode
512	1000	A digital signal available in the AA11-K.
12	14	Intensify on X or Y
8	10	Intensify on Y
4	4	Intensify on X
2	2	Fast intensify enable
1	1	Intensify pulse

The values in Table 23-2 also create scope control words for calls to the DASWP routine with a mode value of 1.

23.2.1.23 **SETADC: Set Channel Information** - The SETADC routine establishes channel start and increment information for an A/D sweep.

SETADC can be invoked as a subroutine or a function as follows:

```
CALL SETADC (ibuf,[iflag],[ichn],[nchn],[inc],[ind])
```

or

```
ind = ISTADC(ibuf,[iflag],[ichn],[nchn],[inc],[ind])
```

ibuf

A 40-word array initialized by the SETIBF routine.

iflag

Equals zero if you want absolute addressing and nonzero for programmable gain addressing. The default is 0.

ichn

The first channel number. The default is 0.

nchn

The number of samples to be taken per interrupt. The default is 1.

inc

The channel increment. The default is 1. You should specify an increment of 2 for differential A/D input.

ind

Receives a success or failure code as follows:

0	indicates an illegal channel number.
1	indicates successful recording of channel information for an A/D sweep.

23.2.1.24 **SETIBF: Set Array for Buffered Sweep** - The SETIBF routine initializes an array required by buffered sweep routines.

The format of the call to SETIBF is as follows:

```
CALL SETIBF (ibuf,[ind],[lamskb],buf0[,buf1...buf7])
```

ibuf

A 40-word array.

ind

Receives a success or failure code as follows:

- 0 indicates an illegal number of buffers was specified. SETIBF initializes the array according to the maximum number of buffers allowed. You specify this maximum number of buffers during the K-series system generation dialog.
- 1 indicates the array was successfully initialized.

lamskb

Present for compatibility with LPA-11 routines. It is ignored by K-series software.

buf0, etc.

The name of a buffer. A maximum of eight buffers can be specified. Any buffer names in excess of eight are ignored. At least two buffers must be specified to maintain continuous sampling.

Each buffer specified in the call to SETIBF is assigned a number from 0 to 7.

The assignment of these numbers is based on the order in which buffer names appear in the argument list. The first buffer whose name appears in the list is assigned number 0, the second is assigned number 1, and so forth. In all subsequent calls to other K-series routines involving the set of buffers specified in a call to SETIBF, these numbers, rather than names, refer to particular buffers.

23.2.1.25 **STPSWP: Stop Sweep** - The STPSWP routine allows your task to stop a sweep that is in progress.

The format of the call to STPSWP is as follows:

```
CALL STPSWP (ibuf[,iwhen],[ind])
```

ibuf

The 40-word array specified in the call that initiated a sweep.

iwhen

Specifies when to stop the sweep:

- 0 indicates at the next sample. This is the default.
- +n (any positive value) indicates at the end of the current buffer.
- -n (any negative value) is reserved.

ind

Receives a success or failure code as follows:

0 indicates that the sweep was not active or no sweep could be found that was associated with the specified ibuf.

1 indicates that the sweep is stopped (at the time indicated by iwhen).

23.2.1.26 **XRATE: Compute Clock Rate and Preset** - The XRATE routine computes an appropriate clock rate and preset that achieves a desired dwell (intersample interval).

NOTE

You can use the XRATE routine only on systems that have a FORTRAN or BASIC-PLUS-2 compiler.

XRATE can be invoked as a subroutine or a function as follows:

```
CALL XRATE (dwell,irate,iprset,iflag)
```

or

```
adwell = XRATE(dwell,irate,iprset,iflag)
```

dwell

The intersample time that you want. The time is expressed in decimal seconds (REAL*4).

irate

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

iprset

Receives the clock preset.

iflag

Specifies whether the computation is for Clock A or Clock B:

0 indicates the computation is for Clock A.

nonzero indicates the computation is for Clock B.

adwell

The actual dwell rate for the clock based on the irate and iprset parameters.

23.2.2 MACRO-11 Interface

MACRO-11 programmers access the K-series support routines described in Section 23.2.1 through either of two techniques:

1. The standard subroutine linkage mechanism and the CALL op code
2. Special-purpose macros that generate an argument list and invoke a subroutine

These techniques are described in the following subsections.

23.2.2.1 Standard Subroutine Linkage and CALL Op Code - K-series routines can be accessed through use of the standard subroutine linkage mechanism and the CALL op code. The format of this procedure is:

```
            .PSECT  code MOV #arglist,R5 ;ARGUMENT ADDRESS TO R5
            CALL ksubr ;CALL K-SERIES ROUTINE .PSECT data
arglist:    .BYTE   narg,0 ;NUMBER OF ARGUMENTS .WORD addr1  ;FIRST
            ARGUMENT ADDRESS
                    .
                    .
                    .
            .WORD   addrn ;LAST ARGUMENT ADDRESS
```

In this sample, the two PSECT directives are shown only to indicate the noncontiguity of the code and data portions of the linkage mechanism. Within the argument list, any argument that is to be defaulted must be represented by a -1 (that is, 177777(octal)).

23.2.2.2 Special-Purpose Macros - To facilitate the calling of K-series support routines from a MACRO-11 program, two macros are provided in file [45,10]LABMAC.MAC. These macros are:

1. INITS
2. CALLS

INITS is an initialization macro. It should be invoked at the beginning of the MACRO-11 source module.

CALLS invokes a K-series support routine. The format of this macro call is as follows:

```
CALLS ksubr,ARG1,...,ARGN
```

ksubr

The name of a K-series support routine.

arg1,etc.

Arguments to be formatted into an argument list and passed to the routine. Each argument can be either a symbolic name or a constant (interpreted as a positive decimal number) or can be defaulted.

23.2.3 The I/O Status Block (IOSB)

Each active sweep must have its own I/O status block. The I/O status block (IOSB) is a 2-word array allocated in your task. It receives the status of a call to a K-series support routine. When a data sweep routine is called, the IOSB is always the first two words of the 40-word array specified as the first argument of the call. The first word of the IOSB contains the status code. The second word contains the buffer size in words.

The codes that can appear in the first word of an I/O status block are in ISA-compatible format (with the exception of the I/O pending condition). Table 23-3 lists all return codes.

Table 23-3
Contents of First Word of IOSB

IOSB word 1	Meaning
0	Operation pending; I/O in progress
1	Successful completion
301	Invalid arguments
305	Hardware or software option not present
306	Illegal buffer specification
313	Data overrun
315	Request terminated
317	Resource in use
397	Invalid event flag

23.3 BUFFER MANAGEMENT

The management of buffers for data sweeps by K-series support routines involves the use of two FIFO (First-In, First-Out) queues:

1. The device queue (DVQ)
2. The user task queue (USQ)

The device queue (DVQ) contains the numbers of all buffers that your task has released to the support routines in a call to RLSBUF. The buffers represented by these numbers are ready to be filled with data (input sweeps) or to be emptied of data (output sweeps). Any buffer specified in a call to INXTBF must already be in DVQ.

The user task queue (USQ) contains the numbers of buffers available to your task. For output sweeps, this queue contains the numbers of buffers that have already been emptied by the driver. For input sweeps, the buffers represented by USQ are those that are filled with data. In both instances, your task determines the next buffer to use (that is, extracts the first element of USQ) by calling IGTBUF or IWTBUF.

Both the DVQ and USQ are initialized to -1, indicating no buffers, when your task calls the SETIBF routine. The task must call RLSBUF before initiating any sweep, because at least one buffer must be present in DVQ for the first input or output to occur.

For input sweeps, the best strategy is to call RLSBUF, specifying the numbers associated with all the buffers to be used in the sweep.

For output sweeps, one approach is to specify two buffers (for continuous sweeps) in the call to RLSBUF. The first action then taken either in a completion routine or after a call to IWTBUF would be to release the next buffer. Note, however, that this approach does not represent true multiple buffering, because data overrun occurs if the second buffer is not released in time.

23.4 SAMPLE FORTRAN PROGRAMS

Two sample FORTRAN programs showing the use of K-series support routines are presented in this section. The first program uses event flags for internal synchronization. The second program demonstrates the use of a completion routine that you supply for synchronization.

NOTE

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors, because error reporting is done through the FORTRAN run-time system
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

23.4.1 Sample Program Using Event Flag

```
      IMPLICIT INTEGER (A-Z)

      DIMENSION BUF(1024,8), IBUF (40), IOSB(2)
      EQUIVALENCE (IBUF(1),IOSB(1))

C
C      INITIALIZE THE IBUF ARRAY FOR THE A/D SWEEP
C
      CALL SETIBF (IBUF,IND, , BUF(1,1), BUF(1,2), BUF(1,3),
     * BUF(1,4), BUF(1,5), BUF(1,6), BUF(1,7), BUF(1,8))

      WRITE (1, 900)
      READ (1, 910) IRATE, IPRSET
C
C      SET THE CLOCK RATE AND PRESET FOR THE SWEEP
C
      CALL CLOCKA (IRATE, IPRSET,IND)
C
C      THIS IS INPUT, SO RELEASE ALL BUFFERS TO SERVICE
C      ROUTINE
C
      CALL RLSBUF (IBUF,IND, 0,1,2,3,4,5,6,7)
C
C      START THE SWEEP. USE 1024 WORD BUFFERS, SAMPLE
C      FOREVER, EXTERNAL START, EVENT FLAG 30, 1 CHANNEL (0).
C
      CALL ADSWP (IBUF, 1024, -1, 256, IPRSET,
     * 30, 0, 0, 1)

C
C      HERE WE COULD CHECK THE I/O STATUS BLOCK TO ENSURE
C      THAT THE SWEEP IS ACTUALLY RUNNING.
C
      IBFCNT=0
C
C      THIS IS THE TOP OF THE DATA PROCESSING LOOP. WE
C      WAIT FOR A BUFFER TO BE COMPLETED, AND THEN DUMP
C      THE FIRST 100 WORDS OF THE BUFFER TO LUN 1.
C
 10   IBUFNO = IWTBUF(IBUF, 30)+1
C
C      IWTBUF RETURNS A POSITIVE BUFFER NUMBER
C      AS LONG AS THERE IS A BUFFER OF DATA AVAILABLE.
C      IF IND IS -1, WE PROBABLY HAD DATA OVERRUN, SO STOP.
C
      IF (IBUFNO .EQ. 0) STOP
      IBFCNT=IBFCNT+1
      WRITE (1,920) IBFCNT
      WRITE (1,930) (BUF(I,IBUFNO), I=1,100)
C
C      RELEASE BUFFER FOR SERVICE ROUTINE TO REFILL
C
      CALL RLSBUF(IBUF,IND,IBUFNO-1)
      GOTO 10
 900  FORMAT (' ENTER IRATE, IPRSET:', $)
 910  FORMAT (I, O)
 920  FORMAT (' DUMP OF BUFFER NUMBER ',I5,/)
 930  FORMAT (1X,10O7)
      END
```

23.4.2 Sample Program Using Completion Routine

```
      IMPLICIT INTEGER (A-Z)
      EXTERNAL AST

      DIMENSION BUF(1024,8), IBUF (40), IOSB(2)
      COMMON /KDATA/ BUF, IBUF, IBFCNT
      EQUIVALENCE (IBUF(1),IOSB(1))

C
C           INITIALIZE THE IBUF ARRAY FOR THE A/D SWEEP
C
      CALL SETIBF (IBUF,IND, , BUF(1,1), BUF(1,2), BUF(1,3),
     * BUF(1,4), BUF(1,5), BUF(1,6), BUF(1,7), BUF(1,8))

      WRITE (1, 900)
      READ (1, 910) IRATE, IPRSET
C
C           SET THE CLOCK RATE AND PRESET FOR THE SWEEP
C
      CALL CLOCKA (IRATE, IPRSET,IND)
C
C           THIS IS INPUT, SO RELEASE ALL BUFFERS TO SERVICE
C           ROUTINE
C
      CALL RLSBUF (IBUF,IND, 0, 1, 2, 3, 4, 5, 6, 7)
C
C           START THE SWEEP. USE 1024 WORD BUFFERS, SAMPLE
C           FOREVER, EXTERNAL START, EVENT FLAG 30, 1 CHANNEL
(0).
C
      IBFCNT = 0
      CALL ADSWP (IBUF, 1024, 0, 256, IPRSET
     * AST, 0, 0, 1)

C
C           HERE WE COULD CHECK THE I/O STATUS BLOCK TO ENSURE
C           THAT THE SWEEP IS ACTUALLY RUNNING.
C
 10   CALL WAITFR (23)
C
C           WHEN EVENT FLAG 23 IS SET THE SWEEP IS COMPLETED.
C           WE MAY EXIT NOW.
C
      STOP

 900  FORMAT (' ENTER IRATE, IPRSET:', $)
 910  FORMAT (I, O)
      END
      SUBROUTINE AST
C
C           THIS SUBROUTINE IS CALLED AT AST LEVEL WHENEVER
C           A BUFFER IS COMPLETED. THIS ROUTINE PROCESSES
C           THE CONTENTS OF THE BUFFER AND THEN RELEASES
C           IT FOR THE SERVICE ROUTINE. IF THE SWEEP IS TO
C           TERMINATE (IOSB NON-ZERO) THEN EVENT FLAG 23. IS
C           SET TO INDICATE TO THE MAINLINE CODE THAT WE ARE
C           DONE.
      IMPLICIT INTEGER (A-Z)
      DIMENSION BUF(1024,8), IBUF(40), IOSB(2)
      COMMON /KDATA/ BUF, IBUF, IBFCNT
      EQUIVALENCE (IBUF(1),IOSB(1))

      IBUFNO = IGTBUF (IBUF) +1

      IF (IBUFNO-1) .GE. 0 GOTO 20
```

```
            IF (IOSB(1) .EQ. 0) PAUSE 'INCONSISTENT STATE'
            CALL SETEF (23)
            RETURN

20          IBFCNT = IBFCNT + 1
C
C           HERE WE WOULD PROCESS THE DATA
C
C
C                RELEASE BUFFER FOR SERVICE ROUTINE
C
            CALL RLSBUF (IBUF, IND, IBUFNO-1)
            RETURN

            END
```

CHAPTER 24

UNIBUS SWITCH DRIVER

24.1 INTRODUCTION

The UNIBUS switch driver supports DT07 UNIBUS switch hardware on RSX-11M-PLUS systems. UNIBUS switches are electronic devices that allow peripherals to be switched from one CPU to another, enabling CPUs to share peripheral devices. UNIBUS switches also facilitate on-line system backup and allow dynamic reconfiguration of systems in which high availability of certain peripherals is required.

24.1.1 DT07 UNIBUS Switches

DT07 UNIBUS switches can provide two, three, or four ports for connecting an external UNIBUS run to one of two, three, or four CPUs.

Any CPU can request connection to a UNIBUS run and receive the connection immediately if the requested UNIBUS run is in the neutral state (it is not connected to another CPU's UNIBUS). If the request is received when the UNIBUS run is connected to another CPU, an interrupt is generated, informing the connected CPU of the pending request, and a watchdog timer is started. The connected CPU normally acknowledges the request, indicating the UNIBUS is still in use. In this case, the UNIBUS remains connected to the CPU. However, if the CPU does not respond to the interrupt within the time limit imposed by the DT07's watchdog timer, the UNIBUS is switched to the requesting CPU. Thus, a CPU that is not operating remains connected to the UNIBUS only until another CPU requests the UNIBUS.

Each DT07 UNIBUS switch port functions as an isolation circuit. When its power is off, it does not affect any CPU operation.

24.1.2 UNIBUS Switch Driver

The UNIBUS switch driver allows you to use the UNIBUS switch in one of two ways:

1. A CPU retains the UNIBUS until the task issuing the directives that connected the UNIBUS to this CPU exits. This is normally accomplished when the task attaches the UNIBUS switch (IO.ATT function) and issues the connect function (IO.CON). When the task exits (for any reason), the system detaches the UNIBUS switch (IO.DET) and performs an implicit disconnect function (IO.DIS), releasing the UNIBUS switch for use by any other task.

The task that attaches the UNIBUS switch can be considered the manager of the UNIBUS switch until the task exits. The task can receive ASTs for certain conditions involving UNIBUS switching (see Section 24.3.1.1).

2. A CPU retains the UNIBUS until a task is executed that explicitly disconnects the UNIBUS. This is normally accomplished when a task issues the IO.CON function and no previous IO.ATT was issued. Once the UNIBUS is connected, the task exits. The UNIBUS then remains connected until either the CPU fails to respond to other CPU requests for the UNIBUS, or a task is executed that explicitly disconnects the UNIBUS. Note that when operating in this manner, no active task is required to retain the UNIBUS.

24.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains all 0s. Words 3, 4, and 5 are undefined.

24.3 QIO$ MACRO

This section summarizes standard and device-specific QIO functions for UNIBUS switches.

24.3.1 Standard QIO Functions

Table 24-1 lists the standard functions of the QIO macro that are valid for UNIBUS switches.

Table 24-1
Standard QIO Functions for UNIBUS Switches

Format		Function
QIO$C	IO.ATT,...,<[ast]>	ATTACH device
QIO$C	IO.DET,...	Detach device
QIO$C	IO.KIL,...	Cancel I/O requests

ast

The address of an optional AST routine that is entered if certain conditions are detected (see Section 24.3.1.1)

IO.ATT does not connect the UNIBUS switch (see device-specific function IO.CON).

IO.DET detaches the UNIBUS switch from the task. If the UNIBUS switch was previously attached by the IO.CON function, an implied disconnect (IO.DIS) function is performed.

The only I/O requests that can be affected by the IO.KIL function are IO.CON and IO.DPT. When IO.KIL is issued during an IO.CON function, further retries are canceled. When IO.KIL is issued during an IO.DPT function, the time-out count is changed, forcing time-out (IE.TMO) to occur.

24.3.1.1 **IO.ATT** - The IO.ATT QIO function attaches the UNIBUS switch to the task issuing the QIO directive. An optional AST address parameter can be specified. However, if it is specified, it must remain valid while the UNIBUS switch remains attached to the task.

The AST service routine for the UNIBUS switch is entered when one of the following conditions occur:

- The UNIBUS switch has become connected to another CPU because:

  1. The operator manually switched the UNIBUS to another CPU, or

  2. This CPU failed to respond to another CPU's request for the UNIBUS within the specified time (the CPU must acknowledge the request by servicing an interrupt, as described in Section 24.1.1).

  UNIBUS switch condition code 1 is passed to the AST routine by the stack, indicating the cause of the AST.

- The UNIBUS switch has disconnected from the CPU because:

  1. A power failure occurred in this CPU (system power failure) and the UNIBUS switch driver was unable to reconnect the bus

  2. A power failure occurred on the connected UNIBUS, causing the driver to disconnect the UNIBUS

  UNIBUS switch condition code 2 (for a system power failure) or condition code 3 (for a UNIBUS power failure) is passed to the AST routine by the stack indicating the cause of the AST.

24.3.1.2 **IO.DET** - The IO.DET function detaches the issuing task from the UNIBUS switch, and in addition, performs an implied disconnect for the issuing task if that task had connected the UNIBUS switch. A detach function is generated by the Executive on behalf of an attached task if that task exits (normally or abnormally) without explicitly detaching the device. For a switched UNIBUS, this causes it to be disconnected if an attached, connected task faults in such a way as to cause it to exit.

24.3.1.3 **IO.KIL** - The IO.KIL function cancels any outstanding IO.CON function that has a nonzero retry count and any outstanding IO.DPT function that has not yet timed out. Other QIO functions in progress are not affected by IO.KIL, and are completed.

24.3.2 Device-Specific QIO Functions

The device-specific functions of the QIO macro that are valid for UNIBUS switches are shown in Table 24-2.

Table 24-2
Device-Specific QIO Functions for UNIBUS Switches

	Format	Function
QIO$C	IO.CON,...,<[rcnt],[cpu]>	Connect UNIBUS switch
QIO$C	IO.DIS,...,<[tout],[port]>	Disconnect UNIBUS switch
QIO$C	IO.DPT,...,<[tout],[port]>	Disconnect UNIBUS switch from specified CPU port
QIO$C	IO.SWI,...,<cpu>	Switch the UNIBUS from current CPU to specified CPU
QIO$C	IO.CSR,...	Read UNIBUS switch CSR

rcnt

The number of additional times the connect is attempted if the IO.CON fails to complete.

cpu

The ASCII letter designating the CPU to receive the UNIBUS switch.

port

The port number, ranging from 0 through 3, of the target CPU that must request the bus prior to the CPU that is currently connected to the UNIBUS actually completing the disconnect. The port number corresponds to the four MANUAL CONNECT switch positions (PORT 0 through PORT 3) marked on the DT07 control panel.

tout

The maximum time (in seconds) allowed (253. maximum) for the function to be completed before an error condition is reported.

Parameter details are included in the following sections.

24.3.2.1 **IO.CON** - The IO.CON (connect) function requests connection of a UNIBUS presently not connected to a specified CPU. It can be issued either by a task previously attached with the IO.ATT function or by a task that is not attached. The IO.CON function has four optional parameters. The use of each parameter is described as follows.

Retry Count -- The retry count specifies the number of additional times the connect function is attempted if the IO.CON fails to complete within the time-out period of the UNIBUS switch. Retry count parameters used in this manner are always nonzero positive values.

The IO.CON function is not completed until either the retry count expires or the UNIBUS switch is successfully connected. Thus, the issuing task having a nonzero retry count is not checkpointed until the IO.CON function is completed.

When a retry count of 0 is specified, the connect function attempts to connect the UNIBUS switch once (no retries) and immediately reports the directive status to the issuing task.

When a retry count of 177777 (-1) is specified, the connect function continues to retry the connection until a successful connection is made or an IO.KIL function is issued.

CPU -- You can use the CPU parameter only with loosely coupled multiprocessor systems to specify the CPU to which the UNIBUS switch should be connected. A loosely coupled system is one in which memory resources are not shared by more than one CPU. Use this function only when the UNIBUS switch is presently not connected (you should use the IO.SWI function to disconnect the UNIBUS switch from a connected closely coupled CPU and connect it to a specified closely coupled CPU). Specify the CPU by a single ASCII letter (A, B, C, or D).

24.3.2.2 **IO.DIS** - The IO.DIS function disconnects the switched UNIBUS from the currently connected CPU.

NOTE

If your task issues the IO.DIS or IO.DPT function, it must determine that all devices on the switched UNIBUS are inactive when it issues the function. The UNIBUS switch driver does not check for active devices on the UNIBUS before completing either the IO.DIS or IO.DPT function.

24.3.2.3 **IO.DPT** - Use the IO.DPT function in a loosely coupled multiprocessor system to allow the UNIBUS to be connected to another CPU on a specified port if the CPU requests connection within a specified time interval. A loosely coupled system is one in which memory resources are not shared by more than one CPU. (Refer to the note at the end of Section 24.3.2.2.)

Time-out -- The time-out parameter specifies the maximum time allowed for the function to complete before an error is reported. Time-out specifications are positive, nonzero values ranging from 1 to 254 seconds. The default time-out value is 2 seconds. If the CPU parameter is included in the IO.DPT function, the driver waits for the specified CPU to request the UNIBUS up to the specified time-out value. If the CPU does not request the UNIBUS during this time, the UNIBUS remains connected and the IE.TMO status is returned to the issuing task.

If a time-out value of 0 is specified, the IO.DIS function does not complete until either the successful disconnect occurs, or an IO.KIL function is issued.

Port -- You can use the port parameter only with loosely coupled multiprocessor systems to specify the port through which the UNIBUS switch should be connected to a CPU. Specify the port by a number ranging from 0 through 3.

24.3.2.4 **IO.SWI** - The IO.SWI function disconnects the UNIBUS switch from the currently connected CPU and connects it to the specified CPU in a closely coupled system. The CPU parameter is required.

IO.SWI is executed without the possibility of a third CPU taking control of the UNIBUS during the switching process.

Use the CPU parameter in closely coupled multiprocessor systems to specify the CPU to which the UNIBUS switch should be connected. Specify the CPU by a single ASCII letter (A, B, C, or D).

24.3.2.5 **IO.CSR** - The IO.CSR function reads maintenance information contained in the device CSR and returns it in the second word of the I/O status block. Information returned is valid only if the UNIBUS switch is connected. Limit the use of this function to diagnostic applications.

24.4 POWER-FAIL RECOVERY

24.4.1 System Power-Fail Recovery

During power-fail recovery, the driver attempts to restore the state of the system prior to the actual power failure. If the UNIBUS switch is found to be disconnected during power-fail recovery, the driver attempts to reconnect the switched UNIBUS. If the first attempt to reconnect the UNIBUS is not successful, an entry is made in the error log and the attached task is notified of the UNIBUS switch state by the AST specified in the IO.ATT function (if previously issued).

If an IO.CON function was in progress when the power failure occurred and a retry count was pending, the UNIBUS switch driver attempts to successfully connect the UNIBUS switch until the retry count expires.

If an IO.DIS or IO.DPT function was in progress when the power failure occurred, the UNIBUS switch driver attempts to complete the operation.

24.4.2 UNIBUS Power-Fail Recovery

If an interrupt is received from the UNIBUS switch indicating a power failure has occurred on the switched UNIBUS, the driver issues an immediate disconnect (IO.DIS). The attached task (if any) is notified by the AST. Note that the system may be corrupted if some of the I/O devices on the switched UNIBUS were active when the power failure occurred, because the drivers for those I/O devices may attempt to access the device registers after the switched UNIBUS (and I/O devices) has become disconnected.

24.5 STATUS RETURNS

Table 24-3 lists the error and status conditions that are returned by the UNIBUS switch driver.

Table 24-3
UNIBUS Switch Driver Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Request aborted** An I/O request was queued (not yet acted upon by the driver) when an IO.KIL was issued.
IE.BAD	**Bad parameters** The parameters specified in the QIO macro were in error.
IE.CNR	**Connect rejected** The connect function did not successfully connect the switched UNIBUS to the specified CPU, and the retry count, if specified, has expired.
IE.DAA	**Device already attached** The device specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.IFC	**Illegal function** A function code was specified in an I/O request that is illegal for the UNIBUS switch driver.

(continued on next page)

Table 24-3 (Cont.)
UNIBUS Switch Driver Status Returns

Code	Reason
IE.NOD	**Insufficient buffer space** Dynamic storage space has been depleted, resulting in insufficient buffer space available to allocate either the I/O packet or the device list buffer.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive (the UNIBUS switch) was not on line, or the CPU specified in the IO.CON or IO.SWI was not on line.
IE.SPC	**Illegal address space** The buffer specified in the IO.CON function was partially or totally outside the address space of the issuing task.
IE.TMO	**Time-out error** The time-out count expired during an IO.DPT operation before the target CPU requested the UNIBUS. This error code is also returned when the DT03/DT07 hardware fails to respond to a request due to a hardware failure.

24.6 FORTRAN USAGE

FORTRAN tasks can use all of the QIO functions described for the UNIBUS switch driver, except AST support is not provided (IO.ATT function with an AST address specified). You can write a macro subroutine, which the FORTRAN task can call, that specifies the AST address.

APPENDIX A

SUMMARY OF I/O FUNCTIONS

This appendix summarizes valid I/O functions for all device drivers described in this manual. Both devices and functions are listed alphabetically. The meanings of the five parameters represented by the ellipsis (...) are described in Section 1.5.1. The meanings of the function-specific parameters shown below are discussed in the appropriate driver chapters. The user may reference these functions symbolically by invoking the system macros FILIO$ (standard I/O functions) and SPCIO$ (special I/O functions), or by allowing them to be defined at task-build time from the system object library.

A.1 ANALOG-TO-DIGITAL CONVERTER DRIVERS

IO.KIL,...	Cancel I/O requests
IO.RBC,...,<stadd,size,stcnta>	Initiate an A/D conversion

A.2 CARD READER DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.RDB,...,<stadd,size>	Read logical block (binary)
IO.RLB,...,<stadd,size>	Read logical block (alphanumeric)
IO.RVB,...,<stadd,size>	Read virtual block (alphanumeric)

A.3 CASSETTE DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.EOF,...	Write end-of-file gap
IO.KIL,...	Cancel I/O requests
IO.RLB,...,<stadd,size>	Read logical block

IO.RVB,...,<stadd,size>	Read virtual block
IO.RWD,...	Rewind tape
IO.SPB,...,<nbs>	Space blocks
IO.SPF,...,<nes>	Space files
IO.WLB,...,<stadd,size>	Write logical block
IO.WVB,...,<stadd,size>	Write virtual block

A.4 COMMUNICATION DRIVERS (MESSAGE-ORIENTED)

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.FDX,...	Set device to full-duplex mode
IO.HDX,...	Set device to half-duplex mode
IO.INL,...	Initialize device and set device characteristics
IO.KIL,...	Cancel I/O requests
IO.RLB,...,<stadd,size>	Read logical block, stripping sync characters
IO.RNS,...,<stadd,size>	Read logical block, transparent mode
IO.SYN,...,<syn>	Specify sync character
IO.TRM,...	Terminate communication, disconnecting from physical channel
IO.WLB,...,<stadd,size>	Write logical block with sync leader
IO.WNS,...,<stadd,size>	Write logical block, no sync leader

A.5 DECTAPE DRIVER

IO.RLB,...,<stadd,size,,,lbn>	Read logical block (forward)
IO.RLV,...,<stadd,size,,,lbn>	Read logical block (reverse)
IO.RVB,...,<stadd,size,,,lbn>	Read virtual block (forward)
IO.WLB,...,<stadd,size,,,lbn>	Write logical block (forward)
IO.WLV,...,<stadd,size,,,lbn>	Write logical block (reverse)
IO.WVB,...,<stadd,size,,,lbn>	Write virtual block (forward)

A.6 DECTAPE II DRIVER

IO.ATT...	Attach device
IO.DET,...	Detach device

IO.KIL,...	Cancel I/O requests
IO.RLB,...,<stadd,size,,,lbn>	Read logical block
IO.WLB,...,<stadd,size,,,lbn>	Write logical block
IO.WLC,...,<stadd,size,,,lbn>	Write logical block with check
IO.RLC,...,<stadd,size,,,lbn>	Read logical block with check
IO.BLS,...,<lbn>	Position tape
IO.DGN,...	Run internal diagnostics

A.7 DEUNA DRIVER

IO.XOP,...,<p1,p2,p3>	Open a Line
IO.XSC,...,<p1,p2>	Set Characteristics (Ethernet)
IO.XIN,...,<p1>	Initialize The Line
IO.XRC,...,<p1,p2,p3,p4,[p5,p6]>	Receive a Message on The Line
IO.XTM,...,<p1,p2,p3,p4,[p5,p6]>	Transmit a Message on The Line
IO.XCL,...	Close The Line
IO.XTL+subfunction,...	Control function

A.8 DISK DRIVER

IO.RLB,...,<stadd,size,,blkh,blkl>	Read logical block
IO.RPB,...,<stadd,size,,,pbn>	Read physical block
IO.RVB,...,<stadd,size,,blkh,blkl>	Read virtual block
IO.SEC,...,<stadd,size,pbn>	Sense characteristics (RX02) only
IO.SMD,...,<density,,>	Set media density (RX02 only)
IO.WDD,...,<stadd,size,,,pbn>	Write physical block (with deleted data mark)
IO.WLB,...,<stadd,size,,blkh,blkl>	Write logical block
IO.WLC,...,<stadd,size,,blkh,blkl>	Write logical block followed by write check
IO.WPB,...,<stadd,size,,,pbn>	Write physical block
IO.WVB,...,<stadd,size,,blkh,blkl>	Write virtual block

A.9 GRAPHICS DISPLAY DRIVER

IO.ATT,...	Attach device
IO.CON,...,<stadd,size,lpef,lpast>	Connect to graphics device

IO.CNT,...	Continue (restart display-processing unit)
IO.DET,...	Detach device
IO.DIS,...	Disconnect from graphics device
IO.KIL,...	Cancel I/O requests
IO.STP,...	Stop (halt display-processing unit)

A.10 INDUSTRIAL CONTROL SUBSYSTEMS

All I/O functions listed below apply to the ICS/ICR subsystem. The five functions supported by the DSS/DRS11 subsystem driver are marked by (D).

IO.CCI,...,<stadd,sizb,tevf>	Connect a buffer to digital interrupts
IO.CTI,...,<stadd,sizb,tevf,arv>	Connect a buffer to counter interrupts
IO.CTY,...,<stadd,sizb,tevf>	Connect a buffer to terminal interrupts
IO.DCI,...	Disconnect a buffer from digital interrupts
IO.DTI,...	Disconnect a buffer from counter interrupts
IO.DTY,...	Disconnect a buffer from terminal interrupts
IO.FLN,...	Set controller off line
IO.ITI,...,<mn,ic>	Initialize a counter
IO.LDI,...,<tname,,[tevf],pn,csm>	Link task to digital interrupts (D)
IO.LKE,...,<tname,,[tevf]>	Link task to error interrupts
IO.LTI,...,<tname,,[tevf],cn,[arv]>	Link task to counter interrupts
IO.LTY,...,<tname,,[tevf]>	Link task to remote terminal interrupts
IO.MLO,...,<opn,pp,dp>	Open or close bistable digital output points (D)
IO.MSO,...,<opn,dp>	Pulse single-shot digital output points
IO.NLK,...,<tname>	Unlink a task from all interrupts (D)
IO.NLN,...	Place ICR controller on line
IO.RAD,...,<stadd>	Read activating data (D)

IO.RBC,...,<stadd,size,stcnta>	Initiate multiple A/D conversions
IO.SAO,...,<chn,vout>	Perform analog output
IO.UDI,...,<tname>	Unlink a task from digital interrupts (D)
IO.UER,...,<tname>	Unlink a task from error interrupts
IO.UTI,...,<tname>	Unlink a task from counter interrupts
IO.UTY,...,<tname>	Unlink a task from terminal interrupts
IO.WLB,...,<staddb,sizb>	Transmit data to the ICR remote terminal

A.11 LABORATORY PERIPHERAL ACCELERATOR DRIVER

IO.CLK,...,<mode,ckcsr,preset>

IO.INI,...,<irbuf,278.>

IO.LOD,...,<mbuf,2048.>

IO.STA,...,<bufptr,40.>

IO.STP,...,<userid>

A.12 LABORATORY PERIPHERAL SYSTEMS DRIVERS

IO.ADS,...,<stadd,size,pnt, ticks,bufs,chna>	Perform A/D sampling
IO.HIS,...,<stadd,size,pnt, ticks,bufs>	Perform histogram sampling
IO.KIL,...	Cancel I/O requests
IO.LED,...,<int,num>	Display number in LED lights
IO.MDA,...,<stadd,size,pnt, ticks,bufs,chnd>	Perform D/A output
IO.MDI,...,<stadd,size,pnt, ticks,bufs,mask>	Perform digital input sampling
IO.MDO,...,<stadd,size,pnt, ticks,bufs,mask>	Perform digital output
IO.REL,...,<rel,pol>	Latch output relay
IO.SDI,...,<mask>	Read digital input register
IO.SDO,...,<mask,data>	Write digital output register
IO.STP,...,<stadd>	Stop in-progress request

A.13 LINE PRINTER DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.WLB,...,<stadd,size,vfc>	Write logical block
IO.WVB,...,<stadd,size,vfc>	Write virtual block

A.14 MAGNETIC TAPE DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.DSE,...	Data security erase (TK50/TU81 only)
IO.EOF,...	Write end-of-file (tape mark)
IO.ERS,...	Erase (TE10 and TU10 not supported)
IO.KIL,...	Cancel I/O requests
IO.RLB,...,<stadd,size>	Read logical block
IO.RLV,...,<stadd,size>	Read logical block reverse
IO.RVB,...,<stadd,size>	Read virtual block
IO.RWD,...	Rewind tape
IO.RWU,...	Rewind and turn unit off line
IO.SEC,...	Read tape characteristics
IO.SMO,...,<cb>	Mount tape and set tape characteristics
IO.SPB,...,<nbs>	Space blocks
IO.SPF,...,<nes>	Space files
IO.STC,...,<cb>	Set tape characteristics
IO.WLB,...,<stadd,size>	Write logical block
IO.WVB,...,<stadd,size>	Write virtual block

A.15 PAPER TAPE READER/PUNCH DRIVERS

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O Requests
IO.RLB,...,<stadd,size>	Read logical block (reader only)
IO.RVB,...,<stadd,size>	Read virtual block (reader only)

IO.WLB,...,<stadd,size>	Write logical block (punch only)
IO.WVB,...,<stadd,size>	Write virtual block (punch only)

A.16 PARALLEL COMMUNICATION LINK DRIVERS

A.16.1 Transmitter Driver Functions

IO.ATX,...,<stadd,size,flagwd, id,retries,retadd>	Attempt message transmission
IO.STC,...,<stadd,size,[state], [mode],,retadd>	Set master section characteristics
IO.SEC,...,	Sense master section status

A.16.2 Receiver Driver Functions

IO.CRX,...,<tef>	Correct for reception
IO.ATF,...,<stadd,size,retadd>	Accept transfer
IO.RTF,...	Reject transfer
IO.DRX,...	Disconnect from reception

A.17 TERMINAL DRIVER

IO.ATA,...,<ast,[parameter2] ,[ast2]>	Attach device, specify unsolicited-character AST [1]
IO.ATT,...	Attach device
IO.CCO,...,<stadd,size,vfc>	Write logical block, cancel CTRL/O
IO.DET,...	Detach device
IO.EIO!TF.RLB,...,<stadd,size>	Extended I/O Read Functions [2]
IO.EIO!TF.WLB,...,<stadd,size>	Extended I/O Write Functions [2]
IO.GTS,...,<stadd,size>	Get terminal support
IO.HNG,...	Hangup remote line
IO.KIL,...	Cancel I/O requests
IO.RAL,...,<stadd,size,[tmo]>	Read logical block and pass all characters [1]
IO.RLB,...,<stadd,size,[tmo]>	Read logical block [1]

---

1. "ast2", "parameter2", and "tmo" parameters are available for full-duplex driver functions only.

2. Full-duplex driver only.

IO.RNE,...,<stadd,size,[tmo]>	Read logical block and do not echo [1]
IO.RPR,...,<stadd,size,[tmo], pradd,prsize,vfc>	Read after prompt [1]
IO.RST,...,<stadd,size,[tmo]>	Read with special terminators
IO.RTT,...,<stadd,size,[tmo], table>	Read logical block ended by specified special terminator [2]
IO.RVB,...,<stadd,size,[tmo]>	Read virtual block [1]
IO.WAL,...,<stadd,size,vfc>	Write logical block and pass all characters
IO.WBT,...,<stadd,size,vfc>	Write logical block and break through any ongoing I/O
IO.WLB,...,<stadd,size,vfc>	Write logical block
IO.WVB,...,<stadd,size,vfc>	Write virtual block
SF.GMC,...,<stadd,size>	Get multiple characteristics
SF.SMC,...,<stadd,size>	Set multiple characteristics

Subfunction bits for terminal-driver functions:

TF.AST	Unsolicited-input-character AST
TF.BIN	Binary prompt
TF.CCO	Cancel CTRL/O
TF.ESQ	Recognize escape sequences
TF.NOT	Unsolicited input AST notification [1]
TF.RAL	Read, pass all characters
TF.RCU	Restore cursor position [1]
TF.RDI	Read with default input (IO.EIO function only) [2]
TF.RES	Read with escape sequence processsing enabled (IO.EIO function only) [2]
TF.RLB	Read logical block (IO.EIO function only) [2]
TF.RLU	Read and convert from lower- to upper-case (IO.EIO function only) [2]
TF.RNE	Read with no echo
TF.RNF	Read with no filter (IO.EIO function only) [2]
TF.RPR	Read after prompt (IO.EIO function only) [2]
TF.RPT	Read in pass-through mode (IO.EIO function only) [2]

1. "ast2", "parameter2", and "tmo" parameters are available for full-duplex driver functions only.

2. Full-duplex driver only.

TF.RST	Read with special terminators
TF.RTT	Read with specified special terminator table (IO.EIO function only) [2]
TF.TMO	Read with time-out [1]
TF.WAL	Write, pass all bits
TF.WBT	Break-through write
TF.WIR	Write with input redisplayed
TF.XCC	CTRL/C starts a command line interpreter [1]
TF.XOF	Send XOFF

1. "ast2", "parameter2", and "tmo" parameters are available for full-duplex driver functions only.

2. Full-duplex driver only.

A.18 UNIBUS SWITCH DRIVER

IO.ATT,...,<[ast]>	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.CON,...,<[rcnt],[cpu]>	UNIBUS switch
QIO$C IO.DIS,...,	Disconnect UNIBUS switch
IO.DPT,...,<[tout],[port]>	Disconnect UNIBUS switch and connect to specified CPU port
IO.SWI,...,<cpu>	Switch UNIBUS from current CPU to specified CPU
IO.CSR,...	Read UNIBUS switch CSR

A.19 UNIVERSAL DIGITAL CONTROLLER DRIVER

IO.CCI,...,<stadd,sizb,tevf>	Connect a buffer to contact interrupts
IO.CTI,...,<stadd,sizb,tevf,arv>	Connect a buffer to timer interrupts
IO.DCI,...	Disconnect a buffer from contact interrupt
IO.DTI,...	Disconnect a buffer from timer interrupts
IO.ITI,...,<mn,ic>	Initialize a timer

IO.KIL,...	Cancel I/O requests
IO.MLO,...,<opn,pp,dp>	Open or close latching digital output points
IO.RBC,...,<stadd,size,stcnta>	Initiate multiple A/D conversions

A.20 VIRTUAL TERMINAL DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O request
IO.RLB,...,<stadd,size>	Read logical block
IO.RVB,...,<stadd,size>	Read virtual block
IO.WLB,...,<stadd,size,stat>	Write logical block
IO.WVB,...,<stadd,size,stat>	Write virtual block
IO.STC,...,<cb,sw2,sw1>	Set terminal characteristics (enable/disable intermediate buffering, or return I/O completion status)

APPENDIX B

I/O FUNCTION AND STATUS CODES

This appendix lists the numeric codes for all I/O functions, directive status returns, and I/O completion status returns. Lists are organized in the following sequence:

- I/O completion status codes
- Directive status codes
- Device-independent I/O function codes
- Device-dependent I/O function codes

Device-dependent function codes are listed by device. Both devices and codes are organized in alphabetical order.

For each code, the symbolic name is listed in form IO.xxx, IE.xxx, or IS.xxx. A brief description of the error or function is also included. Both decimal and octal values are provided for all codes.

B.1 I/O STATUS CODES

This section lists error and success codes which can be returned in the I/O status block on completion of an I/O function. The codes below may be referenced symbolically by invoking the system macro IOERR$.

B.1.1 I/O Error Status Codes

Name	Decimal	Octal	Meaning
IE.2DV	-48	177720	Rename--two different devices
IE.ABO	-15	177761	Operation aborted
IE.ALC	-84	177654	Allocation failure
IE.ALN	-34	177736	File already open
IE.BAD	-01	177777	Bad parameter
IE.BBE	-56	177710	Bad block on device
IE.BCC	-66	177676	Block check error or framing error
IE.BDI	-52	177714	Bad directory syntax

Name	Decimal	Octal	Meaning
IE.BDR	-50	177716	Bad directory file
IE.BDV	-55	177711	Bad device name
IE BHD	-64	177700	Bad file header
IE.BLB	-70	177672	Bad logical buffer
IE.BLK	-20	177754	Invalid block number Logical block number too large
IE.BNM	-54	177712	Bad file name
IE.BTF	-76	177664	Bad tape format
IE.BTP	-43	177725	Bad record type
IE.BVR	-63	177701	Bad version number
IE.BYT	-19	177755	Odd byte count (or virtual address) Byte-aligned buffer specified
IE.CKS	-30	177742	File header checksum failure
IE.CLO	-38	177732	File was not properly closed
IE.CNR	-96	177640	Connection rejected
IE.CON	-22	177752	UDC connect error
IE.DAA	-08	177770	Device already attached
IE.DAO	-13	177763	Data overrun
IE.DFU	-24	177750	Device full
IE.DIS	-69	177673	Path lost to partner
IE.DNA	-07	177771	Device not attached
IE.DNR	-03	177775	Device not ready
IE.DSQ	-90	177646	Disk quota exceeded
IE.DUN	-09	177767	Device not attachable
IE.DUP	-57	177707	Enter--duplicate entry in directory
IE.EOF	-10	177766	End-of-file encountered
IE.EOT	-62	177702	End-of-tape encountered
IE.EOV	-11	177765	End-of-volume encountered
IE.EXP	-75	177665	File expiration date not reached
IE.FEX	-49	177717	Rename--a new file name already in use
IE.FHE	-59	177705	Fatal hardware error
IE.FLG	-89	177647	Event flag already specified

Name	Decimal	Octal	Meaning
IE.FLN	-81	177657	Device already offline
IE.FOP	-53	177713	File already open
IE.HFU	-28	177728	File header full
IE.ICE	-47	177721	Internal consistency error
IE.IES	-82	177656	Invalid escape sequence
IE.IFC	-02	177776	Invalid function code
IE.IFU	-25	177747	Index file full
IE.ILL	-42	177726	Invalid operation on file descriptor block
IE.IQU	-91	177645	Inconsistent qualifier usage
IE.ISQ	-61	177703	Invalid sequential operation
IE.LCK	-27	177745	Locked from read/write access
IE.MII	-99	177635	Media inserted incorrectly
IE.MOD	-21	177753	Invalid UDC or ICS/ICR module
IE.NBF	-39	177731	No buffer space available for file
IE.NBK	-41	177727	File exceeds space allocated, no blocks
IE.NDA	-78	177662	No data available
IE.NDR	-72	177670	No dynamic space available
IE.NFI	-60	177704	File ID was not specified
IE.NFW	-69	177673	Path lost to partner
IE.NLK	-79	177661	Task not linked to specified ICS/ICR interrupts
IE.NLN	-37	177733	No file accessed on LUN
IE.NNC	-77	177663	Not ANSI "D" format byte count
IE.NNN	-68	177674	No such node
IE.NNT	-94	177642	Not a network task
IE.NOD	-23	177751	Caller's nodes exhausted No dynamic memory available
IE.NSF	-26	177746	No such file
IE.NST	-80	177660	Specified task not installed
IE.NRJ	-74	177666	Network connection reject
IE.NTR	-87	177651	Task not triggered
IE.OFL	-65	177677	Device off line

Name	Decimal	Octal	Meaning
IE.ONL	-67	177675	Device on line
IE.ONP	-05	177773	Hardware option not present
IE.OVR	-18	177756	Invalid read overlay request
IE.PES	-83	177655	Partial escape sequence
IE.PRI	-16	177760	Privilege violation
IE.RAC	-44	177724	Invalid record access bits set
IE.RAT	-45	177723	Invalid record attribute bits set
IE.RBG	-40	177730	Invalid record size
IE.RCN	-46	177722	Invalid record number--too large
IE.REJ	-88	177650	Transfer rejected by receiving CPU
IE.RER	-32	177740	File processor device read error
IE.RES	-92	177644	Circuit reset during operation
IE.RNM	-51	177715	Cannot rename old file system
IE.RSU	-17	177757	Shareable resource in use
IE.SNC	-35	177735	File ID, file number check
IE.SPC	-06	177772	Invalid user buffer
IE.SPI	-100	177634	Spindown ignored
IE.SQC	-36	177734	File ID, sequence number check
IE.SRE	-14	177762	Send/receive failure
IE.STK	-58	177706	Not enough stack space (FCS or FCP)
IE.SZE	-98	177636	Unable to size device
IE.TML	-93	177643	Too many links to task
IE.TMO	-95	077641	Time-out on request
IE.UKN	-97	177637	Unknown name
IE.ULK	-85	177653	Unlock error
IE.URJ	-73	177667	Connection rejected by user
IE.VER	-04	177774	Parity error on device
IE.WAC	-29	177743	Accessed for write
IE.WAT	-31	177741	Attribute control list format error

Name	Decimal	Octal	Meaning
IE.WCK	-86	177652	Write check error
IE.WER	-33	177737	File processor device write error
IE.WLK	-12	177764	Write-locked device

B.1.2 I/O Status Success Codes

Name	Decimal Bytes	Octal Word	Meaning
IS.CR	Byte 0: 1 Byte 1: 15	006401	Successful completion with carriage return
IS.CC	Byte 0: 1 Byte 1: 3	001401	Successful completion on read terminated by CTRL/C
IS.ESC	Byte 0: 1 Byte 1: 33	015401	Successful completion with ESCape
IS.ESQ	Byte 0: 1 Byte 1: 233	115401	Successful completion with an escape sequence
IS.PND	+00	000000	I/O request pending
IS.RDD	+02	000002	Deleted data mark read
IS.SUC	+01	000001	Successful completion
IS.TMO	+02	000002	Successful completion on read terminated by time-out
IS.TNC	+02	000002	Successful transfer but message truncated (receiver buffer too small)

B.2 DIRECTIVE CODES

This section lists error and success codes that can be returned in the directive status word at symbolic location $DSW when a QIO directive is issued.

B.2.1 Directive Error Codes

Name	Decimal	Octal	Meaning
IE.ACT	-07	177771	Task not active
IE.ADP	-98	177636	Invalid address
IE.ALG	-84	177654	Alignment error
IE.AST	-80	177660	Directive issued/not issued from AST
IE.CKP	-10	177766	Issuing task not checkpointable

Name	Decimal	Octal	Meaning
IE.FIX	-09	177767	Task already fixed/unfixed
IE.HWR	-06	177772	Device handler not resident
IE.IBS	-89	177647	Invalid send buffer size (.GT. 255.)
IE.IDU	-92	177644	Invalid device or unit
IE.IEF	-97	177637	Invalid event flag (.GT. 64.)
IE.ILU	-96	177640	Invalid logical unit number
IE.ILV	-19	177755	Invaild vector specified
IE.INS	-02	177776	Specified task not installed
IE.IOP	-83	177655	Window has I/O in progress
IE.IPR	-95	177641	Invalid priority (.GT. 250.)
IE.ITI	-93	177643	Invalid time parameters
IE.ITP	-88	177650	Invalid TI parameter
IE.ITS	-08	177770	Directive inconsistent with task state
IE.IUI	-91	177645	Invalid UIC
IE.LNL	-90	177646	LUN locked in use
IE.MAP	-81	177657	Invalid mapping specified
IE.NSW	-18	177756	No swap space available
IE.NVR	-86	177652	Invalid region ID
IE.NVW	-87	177651	Invalid address window ID
IE.PNS	-94	177642	Partition/region not in system
IE.PTS	-03	177775	Partition too small for task
IE.PRI	-16	177760	Privileged violation
IE.RBS	-15	177761	Receive buffer is too small
IE.RSU	-17	177757	Resource in use
IE.SDP	-99	177635	Invalid DIC number or DPB size
IE.TCH	-11	177765	Task is checkpointable
IE.ULN	-05	177773	Unassigned LUN
IE.UNS	-04	177774	Insufficient dynamic storage for send
IE.UPN	-01	177777	Insufficient dynamic storage
IE.WOV	-85	177653	Address window allocation overflow

B.2.2 Directive Success Codes

Name	Decimal	Octal	Meaning
IS.SUC	+01	000001	Directive accepted

B.3 I/O FUNCTION CODES

This section lists octal codes for all standard and device-dependent I/O functions.

B.3.1 Standard I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.ATT	001400	3	0	Attach device
IO.DET	002000	4	0	Detach device
IO.KIL	000012	0	12	Cancel I/O requests
IO.RLB	001000	2	0	Read logical block
IO.RVB	010400	21	0	Read virtual block
IO.WLB	000400	1	0	Write logical block
IO.WVB	011000	22	0	Write virtual block

B.3.2 Specific A/D Converter I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.RBC	003000	6	0	Initiate an A/D conversion

B.3.3 Specific Card Reader I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.RDB	001200	2	200	Read logical block (binary)

B.3.4 Specific Cassette I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.EOF	003000	6	0	Write end-of-file gap
IO.RWD	002400	5	0	Rewind tape

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.SPB	002420	5	20	Space blocks
IO.SPF	002440	5	40	Space files

B.3.5 Specific Communication (Message-Oriented) I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.FDX	003020	6	20	Set device to full-duplex mode
IO.HDX	003010	6	10	Set device to half-duplex mode
IO.INL	002400	5	0	Initialize device and set device characteristics
IO.RNS	001020	2	20	Read logical block, transparent mode
IO.SYN	003040	6	40	Specify sync character
IO.TRM	002410	5	10	Terminate communication, disconnecting from physical channel
IO.WNS	000420	1	20	Write logical block with no sync leader

B.3.6 Specific DECtape I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.RLV	001100	2	100	Read logical block (reverse)
IO.WLV	000500	1	100	Write logical block (reverse)

B.3.7 Specific DECtape II I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.WLC	000420	1	20	Write logical block with check
IO.RLC	001020	2	20	Read logical block with check

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.BLS	004010	10	10	Position tape
IO.DGN	004150	10	150	Run internal diagnostics

B.3.8 Specific Disk I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.RPB	001040	2	40	Read physical block (RX01, RL01, RL02 only)
IO.SEC				Sense characteristics (RX02 only)
	000000	0	0	Single Density
	040000	100	0	Double Density
IO.SMD	002510	5	110	Set media density (RX02 only)
IO.WDD	001140	1	140	Write physical block with deleted data mark (RX02 only)
IO.WLC	001020	1	20	Write logical block followed by write check (all except RX01, RX02)
IO.WPB	000440	1	40	Write physical block (RX01, RX02, RL01, RL02 only)

B.3.9 Specific Graphics Display I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CON	015400	33	00	Connect to graphics device
IO.CNT	017000	36	00	Continue DPU
IO.DIS	016000	34	00	Disconnect from graphics device
IO.STP	016400	35	00	Stop DPU

B.3.10 Specific ICS/ICR, DSS/DR I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CCI	014000	30	0	Connect a buffer to digital interrupt input
IO.CTI	015400	33	0	Connect a counter
IO.CTY	003400	7	0	Connect a remote terminal
IO.DCI	014400	31	0	Disconnect a buffer from digital interrupt input
IO.DTI	016000	34	0	Disconnect a buffer from counter input
IO.DTY	006400	15	0	Disconnect a buffer from terminal input
IO.FLN	012400	25	0	Place selected unit off line
IO.ITI	017000	36	0	Initialize a counter
IO.LDI	007000	16	0	Link a task to digital interrupts
IO.LKE	012000	24	0	Link a task to error interrupts
IO.LTI	007400	17	0	Link a task to counter interrupts
IO.LTY	010000	20	0	Link a task to terminal interrupts
IO.MLO	006000	14	0	Open or close bistable digital output points
IO.MSO	005000	12	0	Pulse single-shot digital output points
IO.NLK	011400	23	0	Unlink a task from all unsolicited interrupts
IO.ONL	017400	37	0	Place selected unit on line
IO.RAD	010400	21	0	Read task activation data
IO.RBC	003000	6	0	Initiate multiple A/D conversions

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.SAO	004000	10	0	Perform analog output to specified channel
IO.UDI	011410	23	10	Unlink a task from digital interrupts
IO.UER	011440	23	40	Unlink a task from error interrupts
IO.UTI	011420	23	20	Unlink a task from counter interrupts
IO.UTY	011430	23	30	Unlink a task from terminal interrupts
IO.WLB	000400	1	0	Output to remote terminal

B.3.11 Specific LPA11-K I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CLK	015000	32	0	Start clock
IO.INI	014400	31	0	Initialize LPA11-K
IO.LOD	014000	30	0	Load microcode
IO.STA	015400	33	1	Start transfer
IO.STP	016400	35	0	Stop request

B.3.12 Specific LPS I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.ADS	014000	30	0	Initialize A/D sampling
IO.HIS	015000	32	0	Initialize histogram sampling
IO.LED	012000	24	0	Display number in LED lights
IO.MDA	016000	34	0	Initialize D/A output
IO.MDI	014400	31	0	Initialize digital input sampling
IO.MDO	015400	33	0	Initialize digital output
IO.REL	013400	27	0	Latch output relay

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.SDI	013000	26	0	Read digital input register
IO.SDO	012400	25	0	Write digital output register
IO.STP	016400	35	0	Stop in-progress request

B.3.13 Specific Magnetic Tape I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.DSE	003040	6	40	Data security erase (TK50 only)
IO.EOF	003000	6	0	Write end-of-file gap
IO.RLV	001100	2	100	Read logical block (reverse)
IO.RWD	002400	5	0	Rewind tape
IO.RWU	002540	5	140	Rewind and unload
IO.SEC	002520	5	120	Sense characteristics
IO.SMO	002560	5	160	Mount and set characteristics
IO.SPB	002420	5	20	Space blocks
IO.SPF	002440	5	40	Space files
IO.STC	002500	5	100	Set characteristics

B.3.14 Specific Parallel Communications Link I/O Function Codes - RSX-11M-PLUS Only

B.3.14.1 Transmitter Driver Functions -

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.ATX	000400	1	0	Attempt message transmission
IO.STC	002500	5	100	Set master section characteristics
IO.SEC	002520	5	120	Sense master section status

B.3.14.2 Receiver Driver Functions -

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CRX	014400	31	0	Connect for reception
IO.ATF	001000	2	0	Accept transfer
IO.RTF	015400	33	0	Reject transfer
IO.DRX	001500	32	0	Disconnect from reception

B.3.15 Specific Terminal I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.ATA	001410	3	10	Attach device, specify unsolicited-input-character AST
IO.CCO	000440	1	40	Write logical block and cancel CTRL/O
IO.EIO	017400	37	0	Extended I/O
IO.GTS	002400	5	00	Get terminal support
IO.HNG	003000	6	0	HANGUP remote line
IO.RAL	001010	2	10	Read logical block and pass all bits
IO.RNE	001020	2	20	Read with no echo
IO.RPR	004400	11	00	Read after prompt
IO.RST	001001	2	1	Read with special terminators
IO.RTT	005001	12	1	Read logical block ended by specified special terminator (Full-duplex driver only)
IO.WAL	000410	1	10	Write logical block and pass all bits
IO.WBT	000500	1	100	Write logical block and break through on-going I/O
SF.GMC	002560	5	160	Get multiple characteristics

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
SF.SMC	002440	5	40	Set multiple characteristics

Subfunction Bits:

With IO.RLB, IO.RPR:

TF.RST	000001
TF.BIN	000002
TF.RAL	000010
TF.RNE	000020
TF.XOF	000100
TF.TMO	000200

With IO.WLB:

TF.RCU	000001
TF.WAL	000010
TF.CCO	000040
TF.WBT	000100
TF.WIR	000200

With IO.ATT:

TF.XCC	000001
TF.NOT	000002
TF.AST	000010
TF.ESQ	000020

With IO.EIO:

TF.WLB	000001	TF.RLB	000002
TF.RCU(1)	000001	TF.RLU(2)	000010
TF.CCO(1)	000040	TF.RTT(2)	000400
TF.WAL(1)	000010	TF.RST(2)	000001
TF.WBT(1)	000100	TF.BIN(2)	000002
TF.WIR(1)	000200	TF.RAL(2)	000010
		TF.RNE(2)	000020
		TF.XOF(2)	000100
		TF.TMO(2)	000200
		TF.RES(2)	010000
		TF.RPR(2)	002000
		TF.RPT(2)	004000
		TF.RNF(2)	020000
		TF.TNE(2)	040000
		TF.RDI(2)	100000

1. Modifiers of the IO.EIO!TF.WLB subfunction. These are specified by you in the item-list buffer.

2. Modifiers of the IO.EIO!TF.RLB subfunction. These are specified by you in the item-list buffer.

B.3.16 Specific UDC I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CCI	014000	30	0	Connect a buffer to contact interrupt digital input
IO.CTI	015400	33	0	Connect a timer
IO.DCI	014400	31	0	Disconnect a buffer from contact interrupt digital input
IO.DTI	016000	34	0	Disconnect a timer
IO.ITI	017000	36	0	Initialize a timer
IO.MLO	006000	14	0	Open or close latching digital output points
IO.RBC	003000	6	0	Initiate multiple A/D conversions

B.3.17 Specific UNIBUS Switch I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CON	15400	33	0	Connect UNIBUS switch
IO.DIS	16000	34	0	Disconnect UNIBUS switch
IO.DPT	16010	34	10	Disconnect UNIBUS switch and connect to specified CPU port
IO.SWI	16400	35	0	Switch UNIBUS from current CPU to specified CPU
IO.CSR	15000	32	0	Read UNIBUS switch CSR

B.3.18 Specific Virtual Terminal I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.STC	002500	5	100	Set terminal characteristics

APPENDIX C

QIO$ INTERFACE TO THE ACPS

This appendix describes the QIO$ level interface to the file processors (ACPs). These include F11ACP for Files-11 disks and MTAACP for ANSI magnetic tape.

F11ACP supports the following functions:

IO.CRE	Create file
IO.DEL	Delete file
IO.ACR	Access file for read only
IO.ACW	Access file for read/write
IO.ACE	Access file for read/write/extend
IO.DAC	Deaccess file
IO.EXT	Extend file
IO.RAT	Read file attributes
IO.WAT	Write file attributes
IO.FNA	Find file name in directory
IO.RNA	Remove file name from directory
IO.ENA	Enter file name in directory
IO.ULK	Unlock block

MTAACP supports the following functions:

IO.FNA	Find file by name
IO.ENA	Enter name in directory (a no-op)
IO.ACR	Access for read only
IO.ACW	Access for read/write
IO.ACE	Access for read/write/extend
IO.DAC	Deaccess file
IO.RVB	Read virtual block
IO.WVB	Write virtual block
IO.EXT	Extend file
IO.CRE	Create file
IO.RAT	Read attributes
IO.APC	ACP control
IO.APV	Privileged ACP control

C.1 QIO$ PARAMETER LIST FORMAT

The device-independent part of a file processing QIO$ parameter list is identical to all other QIO$ parameter lists. The general QIO parameter list is described in detail in Section 1.6 of this manual. The file processor QIO$s require the following six additional words in the parameter lists:

Parameter Word 1	Address of a 3-word block containing the file identifier
Parameter Word 2	Address of the attribute list
Parameter Words 3 & 4	Size and extend control information
Parameter Word 5	Window size information and access control
Parameter word 6	Address of the file name block

NOTE

The Micro/RSX Executive treats File Identifier Blocks, filename blocks, and attribute list entries as read/write data. For this reason, they may not be used in read-only code segments or libraries.

C.1.1 File Identification Block

The File Identification Block is a 3-word block containing the file number and the file sequence number. The format of the File Identification Block is shown in Figure C-1.

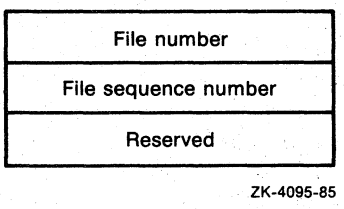


Figure C-1 File Identification Block

F11ACP uses the file number as an index to the file header in the index file. Each time a header block is used for a new file, the file sequence number is incremented. This insures that the file header is always unique. The third word is not currently used but is reserved for the future.

C.1.2 The Attribute List

The file attribute list controls F11ACP reads or writes. File attributes are fields in the file header. These fields are described in detail in the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual.

The attribute list contains a variable number of entries terminated by an all-0 byte. The maximum number of entries in the attribute list is six.

An entry in the attribute list has the following format:

```
.BYTE  <Attribute type>, Attribute size
.WORD  Pointer to the attribute buffer
```

C.1.2.1 **The Attribute Type** - This field identifies the individual attribute to be read or written. The sign of the attribute type code determines whether the transfer is a read or write operation. If the type code is negative, the ACP reads the attribute into the buffer. If the type code is positive, the ACP writes the attribute to the file header. Note that the sign of the type code must agree with the direction implied by the operation. For example, if the type code is positive, the operation must be an IO.WAT or IO.DAC.

The attribute type is one of the following:

- File owner (H.FOWN)

  The file owner UIC is a binary word. The low byte is the owner number and the high byte is the group number.

- File protection (H.FPRO)

  The file protection word is a bit mask with the following format:

  Each of the fields contains four bits, as follows:

  ```
  Bit 1  Read Access
  Bit 2  Write Access
  Bit 3  Extend Access
  Bit 4  Delete Access
  ```

- File characteristics (H.UCHA)

  The following user characteristics are currently contained in the 1-byte H.UCHA field:

  ```
  UC.CON = 200   Logically contiguous file
  UC.DLK = 100   File improperly closed
  ```

- Record I/O Area (U.UFAT)

  This field contains a copy of the first seven words of the file descriptor block. (RMS uses 32 bytes. The first seven are compatible with FCS for sequential files.) See the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual for a description of the FDB.

- File name (I.FNAM)

  The file name is stored as nine Radix-50 characters. the fourth word of this block contains the file type and the fifth word contains the version number.

- File type (I.FTYP)

  The file type is stored as three Radix-50 characters.

- Version number (I.FVER)

  The version number is stored as a binary number.

- Expiration date (I.EXDT)

  Creation date (I.CRDT)
  Revision date (I.RVDT)

  The expiration date is currently unused. When the file is created, the ACP initializes the creation date to the current date and time. It initializes the expiration and revision dates to 0. The ACP sets the revision date to the current date and time each time the file is deaccessed.

- Statistics block

  This block is described in the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual.

- Read entire file header

  This buffer is assumed to be 1000 blocks long. You cannot write this attribute.

- Revision number (I.RVNO)

  The ACP sets the revision number to 0, and increments it every time the file is deaccessed.

- Placement Control

C.1.2.2 **Attribute Size** - This byte specifies the number of bytes of the attribute to be transferred. Legal values are from 1 to the maximum size of the particular attribute. Table C-1 shows the maximum size for each attribute type.

Table C-1
Maximum Size for Each File Attribute

Attribute Type Code	Attribute Type	Maximum Attribute Size in Octal Bytes
1	File owner	6
2	Protection	4
3	File characteristics	2
4	Record I/O area	40
5	File name,type,version number	12
6	File type	4
7	Version number	2
10	Expiration date	7

(Continued on next page)

Table C-1 (Cont.)
Maximum Size for Each File Attribute

Attribute Type Code	Attribute Type	Maximum Attribute Size in Octal Bytes
11	Statistics block	12
12	Entire file header	0
13	Block size (magtape only)	--
15	Revision number and creation/revision/expiration dates	43
16	Placement control	16

C.1.2.3 **Attribute Buffer Address** - The attribute buffer address field contains the address of the buffer in the user's task space to or from which the attribute is to be transferred.

C.1.3 Size and Extend Control

These two parameters specify how many blocks the file processor allocates to a new file or adds to an existing file. These parameters also control the type of block allocation.

The format is as follows:

```
.BYTE  <High 8 bits of size>, <extend control>
.WORD  <Low 16 bits of size>
```

The size field specifies the number of blocks to be allocated to a file on IO.CRE and IO.EXT operations, and the final file size on IO.DEL operations.

The extend control field controls the manner in which an extend operation is to be done. The following bits are defined:

EX.ACl=1	The extend size is to be added as a contiguous block.
EX.AC2=2	Extend by the largest available contiguous piece up to the specified size.
EX.FCO=4	The file must end up contiguous.
EX.ADF=10	Use the default rather than the specified size. The default extend size is the size that was specified when the volume was mounted.
EX.ALL=20	Placement control (see Section C.2).
EX.ENA=200	Enable extend.

C.1.4 Window Size and Access Control

This parameter specifies the window size and access control information in the following format:

```
.BYTE  <window size>, <access control>
```

This word is only processed if the high bit of the access control byte (AC.ENB) is set.

Window size is the number of mapping entries. Specifying a negative window size minimizes window turns. If this byte is zero, the file processor uses the volume default. The size of the window allocated in the dynamic storage region is 6 times the number of mapping entries (each mapping entry is 3 words), plus 10 bytes for the window control block.

On Micro/RSX systems with secondary pool support, the mapping entries are allocated in secondary pool. The window control block and a pointer to secondary pool are located in primary pool.

The following access control bits are defined:

Bit	Definition
AC.LCK=1	Lock out further accesses for Write or Extend
AC.DLK=2	Enable deaccess lock The deaccess lock sets the lock bit in the file header if the file is deaccessed as the result of a task exit without explicitly deaccessing the file. The lock bit is set by the executive. The lock bit is not set when the system crashes.
AC.LKL=4	Enable block locking
AC.EXL=10	Enable explicit block unlocking
AC.ENB=200	Enable access
AC.RWD=10	Rewind the volume (labeled and unlabeled magtape only)
AC.UPD=100	Update mode (labeled magtape only)
AC.POS=20	Do not position to end-of-volume (labeled magtape only)
AC.WCK=40	Initiate driver write-checking

NOTE

Both AC.LKL and AC.EXL must be set if you want block locking. If you do not want block locking, both bits must be clear. Any other combination is an error.

C.1.5 File Name Block Pointer

This word contains the address of a 15-word block in the issuing task's space. This block is called the file name block. The file name block is described in detail in the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual.

The fields of the file name block that are particularly important in file-processing operations are:

- Directory identification (N.DID)

  This field is required for all disk operations. It specifies the directory to which the operation applies. This field is not used for tape operations.

- File identification (N.FID)

  This field is required as input for enter operations. This field is returned as output by find and remove operations.

- File name (N.FNAM), type (N.FTYP), and version number (N.FVER)

  These fields are required as input to enter, find, and remove operations. For find and remove operations, the file processor locates the appropriate entry by matching the information in these fields with the directory entries.

- Status word (N.STAT)

- Wildcard context (N.NEXT)

  This field is required as input for wildcard operations. It specifies the point at which to resume processing. It is updated for the next operation. It must initially be set to 0.

C.2 PLACEMENT CONTROL

The placement control attribute list entry controls the placement of a file in a particular place on the disk. You can specify either exact or approximate placement on IO.CRE and IO.EXT operations.

The placement control entry must be the first entry in the attribute list.

The format of the placement control attribute list entry is as follows:

```
.BYTE  placement control,0
.WORD  high-order bits of VBN or LBN
.WORK  low-order bits of VBN or LBN
.BLKW  4       ; Buffer to receive starting and ending LBN if
                 AL.LBN is set.
```

The following bits are defined for the placement control field:

Bit	Definition
AL.VBN=1	Set if block specified is a VBN; otherwise, the block is the LBN
AL.APX=2	Set if you want approximate placement; otherwise, placement is exact
AL.LBN=4	Set if you want starting and ending LBN information

C.3 BLOCK LOCKING

Block locking only occurs when the user accesses a file with AC.LKL and AC.EXL set in the access control byte of the parameter list. Any read or write operation causes a check to see if the block is locked.

A write access locks a block for exclusive access. A write operation can only access a block that is not locked by any accessor. The only exception to this is an exact match with a previous lock owned by the same accessor.

A read access locks a block for shared access. A read operation can access any block locked for shared access.

The user must unlock a block with an explicit unlock request, IO.ULK. IO.ULK may be used to unlock one or all blocks.

If all accessors to a file have not requested block locking, the F11ACP returns an error (see Table C-2).

When the file is deaccessed, all locks owned by the accessor are released.

Each active lock requires eight bytes from the dynamic storage region. This storage is deallocated when the file is deaccessed.

C.4 SUMMARY OF F11ACP FUNCTIONS

The following is a summary of the functions implemented in F11ACP. A list of accepted parameters follows each function. All parameters are required unless specified as optional. Parameters other than those listed are illegal for that function and must be 0.

IO.CRE **Create file**

- #1 The file identifier block is filled in with the file identifier and sequence number of the created file.
- #2 Write attribute and/or placement control list (optional)
- #3 & #4 Extend control (optional)

  The amount allocated to the file is returned in the high byte of IOST(1) plus IOST(2).
- #5 May be nonzero but must be disabled

IO.DEL **Delete or truncate file**

- #1 Optional if the file is accessed
- #3 & #4 Size to truncate the file to. If not enabled, the file is deleted. If enabled, the remaining 31 bits specify the size the file is to be after truncation. The change in file allocation is returned in the high byte of IOST(1) plus IOST(2). This amount will be zero or negative.

IO.ACR **Access file for read only**

IO.ACW **Access file for read/write**

IO.ACE		**Access file for read/write/extend**
	#1	File identifier pointer
	#2	Read attributes control (optional)
	#5	Access control must be enabled
IO.DAC		**Deaccess file**
	#1	File identifier pointer (optional)
	#2	Write attributes control list
	#5	May be nonzero but must be disabled
IO.EXT		**Extend file**
	#1	Optional if file is accessed
	#2	Placement control attribute list (optional)
	#3 & #4	Extend control
		The amount allocated to the file is returned in the high byte of IOST(1) plus IOST(2).
IO.RAT		**Read attributes**
	#1	Optional if file is accessed
	#2	Read attributes control list
IO.FNA		**Find name in directory**
IO.RNA		**Remove name from directory**
IO.ENA		**Enter name in directory**
	#5	May be nonzero but must be disabled
	#6	File name block pointer
IO.ULK		**Unlock block**
	#2	0 or count of blocks to unlock
	#4 & #5	Starting VBN to unlock or 0 to unlock all blocks.
IO.RVB		**Read virtual block**
IO.WVB		**Write virtual block**
	#1	User buffer
	#2	Buffer length
	#4 & #5	VBN

C.5 SUMMARY OF MTAACP FUNCTIONS

The following is a summary of the functions implemented in MTAACP. A list of accepted parameters follows each function. All parameters are required unless specified as optional. Parameters other than those listed are illegal for that function and must be 0.

IO.FNA Find file by name

#5 AC.RWD set in the access control byte indicates that the volume is to be rewound prior to the search.

#6 Pointer to file name block.
The following fields are used as input:

N.FNAM
N.FTYP
N.FVER
N.STAT

The following fields are returned by MTAACP:

N.FID
N.FNAM
N.FTYP
N.FVER
N.STAT

IO.ENA Enter name in directory -- a no-op for magnetic tape

IO.ACR Access for read only

#1 File identifier pointer. Used to position a tape by file identifier.

#2 Read attribute list (optional)

#5 Ignored

IO.ACW Access for read/write

This function will be rejected with the error code IE.PRI. (Extend access is required.)

IO.ACE Access for read/write/extend

#1 File identifier pointer. Used to position tape by file identifier.

#2 Read attribute list (optional)

#5 AC.UPD (update mode). If AC.UPD is set, the tape will be positioned to overwrite the file and all files beyond the current file will be lost. If AC.UPD is not set, the tape will be positioned for append. If the file is not the last file, MTAACP returns the error code IE.ISQ.

IO.DAC Deaccess file

#1 File identifier pointer is ignored.

#5 AC.RWD set indicates that the volume is to be rewound after the file is closed.

IO.RVB Read virtual block

#1 Buffer address

#2 Buffer size. The buffer size must be greater than 18 bytes and less than the declared block length for the entire file.

#4 High VBN

#5 Low VBN

The virtual block number must be either zero or exactly one greater than the previous block number.

IO.CRE Create File

#1 File identifier pointer. The file sequence and section number will be returned to the user's file identifier block.

#2 Attribute list pointer. Used to write the attributes for the newly created file. Attribute type code must be positive.

#5 If AC.RWD is set, the volume will be positioned at the beginning and will overwrite the first file. This effectively reinitializes the volume.

If AC.RWD is not set and AC.POS is set, the volume set will be positioned to the next file position beyond the current file and will overwrite that file. All files beyond that on the volume will be destroyed.

If neither AC.RWD nor AC.POS is set, the volume set will be positioned at its end and the new file will be appended to the set.

For unlabeled tapes, MTAACP only checks AC.RWD.

#6 Filename block pointer.

IO.RAT Read Attributes

#1 File identifier pointer. Used to position the tape by the file identifier.

#2 Attribute list pointer (see Section C.1.2)

The following attribute list entries are meaningful for magnetic tape:

1,2	UIC
1,4	UIC and protection
1,5	UIC, protection, and characteristics
2,2	Protection
2.3	Protection and characteristics
3,1	Characteristics
4,32	User file attributes
5,6	File name
5,8	File name and type
5,10	File name and type
6,2	File type

6,4	File type and version number
7,2	Version number
8,7	Expiration date
-9,10	Statistics block (read only)
-10,0	Entire header (read only)
11,2	Block size

IO.APC ACP Control

#3 One of the following user control function codes:

1	Rewind volume set.
2	Position to end of volume set.
3	Close current volume and continue processing the next section of the same file on the next volume of the volume set.
4	Space physical records in currently accessed file.
5	Get ACP characteristics.
6	Rewind current file.

IO.APV Privileged ACP Control

This function is used only by the MOUNT and DISMOUNT commands. This interface is subject to change and, therefore, will not be documented until a future release.

C.6 HOW TO USE THE ACP QIO$ FUNCTIONS

Although the operations described in this appendix are normally performed by the file-access methods (RMS and FCS), your application may issue the ACP QIO$s. The required parameters for each QIO$ are described in the preceding section. The necessary steps for common operations are described in the following section:

NOTE

The file identifier is the only way to refer to a file.

C.6.1 Creating a File

To create a file:

- Use IO.CRE to create it.
- Enter it in the Master File Directory (MFD) or a user directory with IO.ENA.

C.6.2 Opening a File

To open a file:

- Use IO.FNA to find the File Identifier of the directory in the MFD.
- Use IO.FNA to find the File Identifier of the file in the directory.
- Access the file with IO.ACR, IO.ACW, or IO.ACE.

C.6.3 Closing a File

To close a file:

- Deaccess the file with IO.DAC.

C.6.4 Extending a File

To extend a file:

- Use IO.FNA to find the file identifier if the file is not accessed.
- Use IO.EXT to extend the file.

C.6.5 Deleting a File

To delete a file:

- Use IO.FNA to find the file identifier.
- Use IO.RNA to remove the directory name.
- Use IO.DEL to delete the file.

C.7 ERRORS RETURNED BY THE FILE PROCESSORS

The error codes returned by F11ACP and MTAACP are shown in Table C-2.

Table C-2
File Processor Error Codes

Error Code	Operations	Explanation
IE.ABO	IO.RVB/IO.WVB	Indicates that not all requested data was transferred by the device.
IE.ALC	Extend or create operation	Indicates that the operation failed to allocate the file because of placement control or because of other related problems.
IE.ALN	An attempt to access a file	Indicates that a file is already accessed on that LUN.
IE.BAD	Any function	Indicates that a required parameter is missing, that a parameter that should not be present is present, that a parameter that must be disabled is enabled, or that a parameter value is invalid.
IE.BDR	Directory operations	Indicates that you attempted a directory operation on a file that is not a directory, or that the specified directory is corrupted. This is usually caused by a 0 version number field.
IE.BHD	Any operation	Indicates that a corrupt file header was encountered, or that the operation required a feature not supported by the FCP (such as multiheader support or support for unimplemented features).

(Continued on next page)

Table C-2 (Cont.)
File Processor Error Codes

Error Code	Operations	Explanation
IE.BVR	Directory operations	Indicates that you attempted to enter a name in a directory with a negative or 0 version number.
IE.BYT	Any function	This error is returned if the buffer specified is on an odd byte boundary or is not a multiple of four bytes.
IE.BTP	Unlabeled Magtape Create	An attempt was made to create an unlabeled tape file with a record type other than fixed.
IE.CKS	Any operation	Indicates that the checksum of a file header is incorrect.
IE.CLO	File access operations	Indicates that the file was locked against access by the "deaccess lock bit."
IE.DFU	An allocation request	Indicates that there is insufficient free disk space for the requested allocation.
IE.DUP	An enter name operation	Indicates that the name and version already exist.
IE.EOF	IO.RVB/IO.WVB/IO.DEL	On read operations, this indicates an attempt to read beyond end of file. On truncate operations, it indicates an attempt to truncate a file to a length longer than that allocated or that the file was already at EOF.

(Continued on next page)

Table C-2 (Cont.)
File Processor Error Codes

Error Code	Operations	Explanation
IE.HFU	An extended operation	Indicates that the file header is full and cannot contain any more retrieval pointers and that adding an extension header is not allowed. When this error code is returned on a create operation, it indicates that the index file could not be extended to allow a file header to be allocated.
IE.IFC	Returned by exec	Illegal function code.
IE.IFU	Create or extend operation	Indicates that there are no file headers available based on the parameters specified when the volume was initialized.
IE.LCK	Returned on file access, directory operations, and on truncate	Indicates that the file is already accessed by a writer and that shared write has not been requested or is not allowed.
IE.LUN	Any operation requiring a file ID	Indicates that file ID has not been supplied and that the file is not accessed on the LUN.
IE.NOD	All file operations that require DSR	Indicates that an I/O request failed because of IE.UPN, and that the FCP was unable to allocate required space from DSR or from secondary pool for data structures.
IE.NSF	All file operations	Indicates that the specified directory entry does not exist, that a file corresponding to the file ID does not exist, or that the file is marked for delete.
IE.OFL	Returned by exec	The device is off line.

(Continued on next page)

Table C-2 (Cont.)
File Processor Error Codes

Error Code	Operations	Explanation
IE.PRI	Any operation	Indicates that the user does not have the required privilege for the requested operation, or that the user has not requested the proper access to the file if the file is already accessed (for example, in an attempt to write to a file that is accessed for read). This error code also indicates an attempt to do file I/O to a device that is not mounted.
IE.RER	Any operation	Indicates that the FCP encountered a fatal device read error during an operation; the operation has been aborted.
IE.SNC	Any operation	Indicates that the file number and the value contained in the header do not agree. This generally means that the header has gone bad because of a crash or a hardware error.
IE.SPC	Returned by exec	Indicates an illegal buffer.
IE.SQC	Any operation	Indicates that the file sequence number does not agree with the file header; usually indicates that the file has been deleted and the header has been reused.
IE.WAC	File access operations	Indicates that the file is already write accessed and lock against writers is requested.
IE.WAT	Write attributes and deaccess	Indicates that the FCP encountered an invalid attribute.

(Continued on next page)

Table C-2 (Cont.)
File Processor Error Codes

Error Code	Operations	Explanation
IE.WER	Any operation	Indicates that the FCP encountered a fatal device write error during an operation. The operation has been aborted, but the disk structure may have been corrupted.
IE.WLK	Any operation requiring write access	Indicates that the volume is software write-locked.