PA-RISC 2.0 Firmware Architecture Reference Specification

Version 1.1E

Printed in U.S.A. July 22, 2004

Notice

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information that is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company.

Copyright © 1983-2003 by HEWLETT-PACKARD COMPANY All Rights Reserved

5. IODC

The purpose of IODC is to provide a uniform, architected mechanism to obtain module-type dependent information from a module. IODC is composed of two parts. The first part is a set of up to 16 bytes that identify and characterize the module. The second part is a set of entry points that provide a standard procedural interface for performing module-type dependent operations. IODC is typically contained in a ROM on the module.

The IODC data bytes contain sufficient module-type dependent information to allow a configuration to be determined automatically during system initialization. This allows new modules and I/O devices to be installed without modification of the processor configuration and boot ROM.

Each operating system should establish a software convention to control access to IODC. The IODC entry points provide a consistent interface for operations such as module initialization and testing. Special IODC entry points are defined to support boot.

All software access to IODC is obtained through the PDC_IODC procedure. PDC_IODC accesses a module's IODC by writing the desired IODC address to its IO_DC_ADDRESS register, and reading the addressed data from its IO_DC_DATA register.

In variable configuration systems, each native module is required to support the IO_DC_ADDRESS and IO_DC_DATA registers in its HPA.

However, in fixed configuration systems, PDC_IODC need not access the module at all: instead it may return the appropriate fixed IODC directly to its caller. This is also true for PCI devices under a PCI host bridge. In such a configuration, a module need not implement the IO_DC_ADDRESS and IO_DC_DATA registers in its HPA if the functionality is emulated by the PDC_IODC procedure. Further, for PCI devices, the HPA need not be a valid 64-bit I/O address, but may be a 32-bit PCI Function Address (PFA). Bit 32 of the PFA must always be zero. (the high order bit)

The sections that follow give a complete definition of IODC. Section 5.1 specifies the format of IODC and gives the meaning of first 16 data bytes. Section 5.2 shows how the entry point code blocks are organized. Section 5.3 specifies the calling conventions for the IODC entry points. The IODC entry points defined by the architecture are described in section 5.4. Finally, Section 5.5 describes the module-specific aspects of IODC for all modules.

5.1 IODC Data Bytes

The architecturally specified locations of the IODC data bytes are shown below:

TABLE 5-1. IODC Data Bytes

Byte Address	Name	Description
0 - 1	IODC_HVERSION	Hardware version number
2	IODC_SPA	Soft physical address capability
3	IODC_TYPE	Type of module
4 - 7	IODC_SVERSION	Software version number
8	IODC_REV	IODC revision
9	IODC_DEP	HVERSION dependent
10	IODC_FEATURES	Optional features supported
11	RESERVED	Reserved
12 - 13	IODC_CHECK	Checksum
14 - 15	IODC_LENGTH	Length of entry point table
16 - 19	IODC_ENTRY_1	First entry point
. . .	. . .	. . .
n-4 - n-1	IODC_ENTRY_N	Last entry point

where n is the address of the byte following the last word in the entry point table.

Modules provide one of three subsets of the data shown above. Every module provides at least byte 3 of IODC. The information in byte 3 is sufficient to identify how many IODC bytes the module provides:

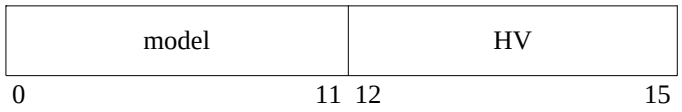
- EXACTLY 1 BYTE: Byte 3, the IODC_TYPE byte, is the only byte the module provides. Category A and category B processors are the only modules in this class.
- EXACTLY 8 BYTES: The module provides bytes 0 through 7.
- AT LEAST 16 BYTES: The module provides at least bytes 0 through 15. Bytes 14 and 15 imply how much larger the IODC is.

ENGINEERING NOTE

Some bus specifications allow more than one module to be implemented on one physical card. The I/O Architecture requires one IODC for each module; the card can emulate this functionality by mapping multiple IO_DC_ADDRESS and IO_DC_DATA registers to separate portions of one physical ROM.

IODC_HVERSION (bytes 0 - 1)

Format:



Purpose: To specify the hardware version number (HVERSION) for the module.

An HVERSION is relevant only when compared to the HVERSION of a module which shares the same IODC_SVERSION[model]. Therefore, when IODC_SVERSION[model] differs there is no guarantee that the HVERSIONs have the same interpretation.

Fields: The *model* field specifies the module hardware implementation. If a hardware implementation is updated in a way which is visible to HVERSION-dependent software (e.g., diagnostics) but transparent to SVERSION-dependent software (e.g., drivers), IODC_HVERSION[model] must be incremented. As a result of this type of module change, IODC_SVERSION[rev] and IODC_SVERSION[model] remain unchanged.

If a hardware implementation is updated in a way which is transparent to all HVERSION-dependent and SVERSION-dependent software, IODC_SVERSION[rev] and IODC_SVERSION[model] must remain unchanged.

SUPPORT NOTE

When a change is made to an implementation that is not visible to all HVERSION-dependent and SVERSION-dependent software, IODC_HVERSION[12..15] should be updated if the change is considered to be a design change, and not just a simple hardware part change. Examples of typical design changes are:

- Updating components on a card from TTL to CMOS.
- Going from a full size card to a half size card with SMT.
- Using increased integration to reduce parts count.

The goal of updating the value of IODC_HVERSION[12..15] is to provide support with a means of tracking design changes that are not visible to diagnostics. Contact the support organization to determine the initial value of IODC_HVERSION[12..15] and whether IODC_HVERSION[12..15] should be updated for a particular implementation change. Note that numerically increasing values for IODC_HVERSION[12..15] typically denote chronologically later hardware versions.

If a hardware implementation is changed in a way which is visible to SVERSION-dependent software, then IODC_SVERSION[rev] or IODC_SVERSION[model] must change.

In order to determine the correct fields to update as a result of a module change, the primary consideration is the impact on HVERSION-dependent and SVERSION-dependent software. The following table shows the fields that must be updated as a result of a module change.

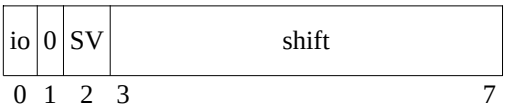
SV Software	HV Software	SV[model]	SV[rev]	HV[model]
Y	X	C	Typically 0	4
N	N	U	U	U
N	Y	U	U	C
S	N	U	C	U
S	Y	U	C	C

Where:

N = change to the module is not visible to software
Y = change to the module is not compatible with existing software
S = change to the module is visible to software, but existing software will continue to work (i.e., the new module is a strict superset)
X = don’t care
C = changed
U = unchanged

IODC_SPA (byte 2)

Format:



Purpose: To specify the soft physical address capabilities of a module.

Fields: The *io* bit indicates whether the SPA is in the memory (*io* = 0) or I/O (*io* = 1) address space. Each SPA must be entirely in the memory address space or entirely in the I/O address space. Only memory modules are allowed to have SPA in the memory address space. If a module does not have an SPA, the *io* bit is SVERSION dependent.

The *shift* field specifies the maximum SPA space size (in bytes) and the alignment requirement (in bytes) as 2^{shift}. If a module does not have an SPA, the *shift* field must be 0.

The following table shows the relationships among module type, *io* bit, and *shift* field:

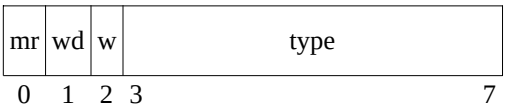
Module Type	*io* bit	*shift* field	Description
memory	0	0	processor-dependent satellite
memory	0	1 - 11	illegal
memory	0	12 - 31	maximum SPA = 2^{shift}
memory	1	0 - 31	illegal
coherent i/o module	0	0 - 31	TLB entries = 2^{shift}
coherent i/o module	1	0 - 31	illegal
other module types	0	0	module has no SPA
other module types	0	1 - 31	illegal
other module types	1	0	module has no SPA
other module types	1	1 - 11	illegal
other module types	1	12 - 26	maximum SPA = 2^{shift}
other module types	1	27 - 31	illegal

If software sees an illegal combination of module type, *io* bit, and *shift* field, it must presume an error and should not attempt to enable the SPA of the module in question.

For modules where the architecture defines the SVERSION (i.e., native processor, memory, and bus converter port modules), the value of IODC_SPA{2} is HVERSION dependent.

IODC_TYPE (byte 3)

Format:



Purpose: To identify the type of the module.

Fields: If the *mr* ("more") bit is 1, the module provides at least the first 16 bytes of IODC. If *mr* is 0, the module provides no more than the first 8 bytes of IODC.

If the *w* ("wide") bit is 1, the modules IODC entry points should be called with the PSW W-bit = 1 (wide mode). If the *w* bit is 0, the modules entry points should be called with PSW W-bit = 0 (narrow mode).

If the *wd* ("word") bit is 1, the module provides a full word of address-justified data in IO_DC_DATA. If *wd* is 0, the module provides a single byte of right-justified data in IO_DC_DATA. The locations of the valid and HVERSION-dependent data bytes provided in IO_DC_DATA are illustrated below:

IO_DC_ADDRESS	wd = 0				wd = 1			
0	HV	HV	HV	DATA0	DATA0	DATA1	DATA2	DATA3
1	HV	HV	HV	DATA1	DATA0	DATA1	DATA2	DATA3
2	HV	HV	HV	DATA2	DATA0	DATA1	DATA2	DATA3
3	HV	HV	HV	DATA3	DATA0	DATA1	DATA2	DATA3
4	HV	HV	HV	DATA4	DATA4	DATA5	DATA6	DATA7
5	HV	HV	HV	DATA5	DATA4	DATA5	DATA6	DATA7
6	HV	HV	HV	DATA6	DATA4	DATA5	DATA6	DATA7
7	HV	HV	HV	DATA7	DATA4	DATA5	DATA6	DATA7
	0	1	2	3	0	1	2	3

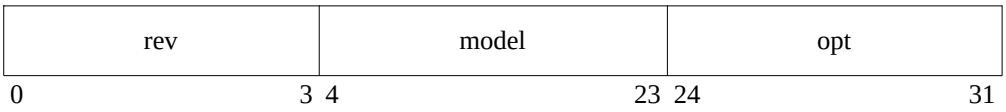
The *type* field specifies the module type, as follows:

Value	Name	Description
0	TP_NPROC	Native Processor
1	TP_MEMORY	Memory
2	TP_B_DMA	Type-B DMA I/O
3	Obsolete	Reserved
4	TP_A_DMA	Type-A DMA I/O
5	TP_A_DIRECT	Type-A Direct I/O
6	Obsolete	Reserved
7	TP_BCPORT	Bus Converter Port
8	TP_CIO	HP-CIO Adapter
9	TP_CONSOLE	Console
10	TP_FIO	Foreign I/O Module
11	TP_BA	Bus Adapter
12	TP_IOA	
13	TP_BRIDGE	Bus Bridge to Foreign Bus
14	TP_FABRIC	Fabric ASIC
15	TP_MC	Management Controller
16 - 30	Reserved	Future Types
31	TP_FAULT	Faulty Module

Modules with an internal hardware fault may return TP_FAULT instead of their true type.

IODC_SVERSION (bytes 4 - 7)

Format:



Purpose: To specify the software version number (SVERSION) for the module.

SVERSIONs are unique across all module types. Therefore, SVERSIONs do not need to be qualified by module type.

Fields: If an enhanced feature is added to a module, the value of the *rev* field must be changed. An increased value of *rev* implies that the newer revision retains all of the previously released features (i.e., the new module is a strict superset). Existing SVERSION-dependent software will still function on a module with an increased value of *rev*. As a result of this type of module change, IODC_HVERSION[model], and IODC_SVERSION[model] remain unchanged.

Bit 27 of the *opt* field define the Module Category, also referred to as the *mc* bit. Bit 26 of the *opt* field is reserved. The remainder of the *opt* field is defined differently for each module type. The definitions are presented in Section 5.5, Module Specific IODC.

The *model* field specifies the software interface to a module. If a change is made to a module which results in incompatible SVERSION-dependent software capabilities, the value of *model* must be changed. As a result of this type of module change, IODC_HVERSION[model] must be set to 4, and IODC_SVERSION[rev] is typically set to 0.

ENGINEERING NOTE

If a reduced functionality module is to be released in the future IODC_SVERSION[rev] may optionally be set to a value greater 0 when IODC_SVERSION[model] is updated. This allows for a future module which is a subset of the existing module.

In order to determine the correct fields to update as a result of a module change, the primary consideration is the impact on HVERSION-dependent and SVERSION-dependent software. The following table shows the fields that must be updated as a result of a module change.

SV Software	HV Software	SV[model]	SV[rev]	HV[model]
Y	X	C	Typically 0	4
N	N	U	U	U
N	Y	U	U	C
S	N	U	C	U
S	Y	U	C	C

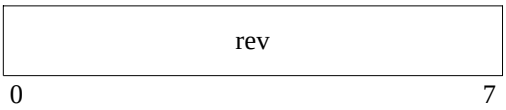
Where:

N = change to the module is not visible to software
Y = change to the module is not compatible with existing software
S = change to the module is visible to software, but existing software will continue to work (i.e., the new module is a strict superset)
X = don't care
C = changed
U = unchanged

The IODC_SVERSION[model] value 0x00FFF is allocated for use by Type A-Direct, Type A-DMA, and Type B-DMA modules developed independent of HP. Modules with this IODC_SVERSION[model] value must have an IODC_SVERSION[rev] value of 0. For these modules, the contents and meaning of the IODC_HVERSION[model] field are dependent on the particular module implementation, and do not have their normal architectural definition.

IODC_REV (byte 8)

Format:



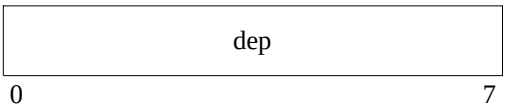
Purpose: To specify the revision of the IODC entry points.

Fields: The *rev* field contains the revision of the IODC entry points.

If a module implements any of the IODC entry points, IODC_REV[rev] must be incremented if the IODC entry point code is modified. If a change is made to a module which implements any of the IODC entry points, and that change results in IODC_SVERSION[model] being updated, IODC_REV[rev] must be set to 0. For modules that do not implement any IODC entry points, the value of IODC_REV[rev] is HVERSION dependent. Note that IODC_REV is an unsigned byte.

IODC_DEP (byte 9)

Format:

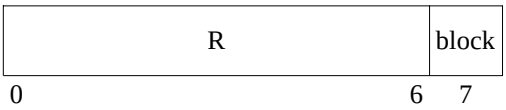


Purpose: To provide HVERSION-dependent information about the module.

Fields: The *dep* field contains HVERSION-dependent information about the module.

IODC_FEATURES (byte 10)

Format:

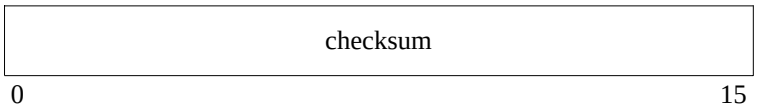


Purpose: To specify which optional IODC feature are supported by this module.

Fields: The *block* field specifies whether ENTRY_IO support Book block input (ARG1=16) and Boot block output (ARG1=17). A 1 in the *block* indicates the feature is supported. A 0 indicates the feature is not supported.

IODC_CHECK (bytes 12 - 13)

Format:



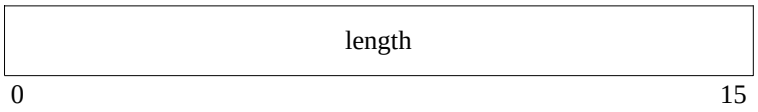
Purpose: To provide a checksum for the first n bytes of IODC.

PDC_IODC is required to compute this checksum when it is called (with ARG1=0 and ARG4=0) to get the first 16 bytes of IODC.

Fields: The *checksum* field is adjusted to guarantee that the 16-bit arithmetic sum of the halfwords at locations 0 through n-2 is zero, where n is the number of bytes in the previous IODC and the following entry point table.

IODC_LENGTH (bytes 14 - 15)

Format:



Purpose: To specify the length in words of the entry point table, which is comprised of words IODC_ENTRY_1 through IODC_ENTRY_N.

Fields: The *length* field specifies the length in words of the entry point table.

This indirectly specifies the "n" bytes covered by the checksum IODC_CHECK, as follows:

$$n = (4 * length) + 16$$

Thus *length* has the same value as N, the number of entry points contained in the IODC. If *length* is zero, then there are no entry points in the IODC, just exactly 16 bytes of data.

5.2 IODC Entry Point Table

The words IODC_ENTRY_1 through IODC_ENTRY_N comprise the entry point table. The table describes the entry points that the module provides. Each word in the entry point table specifies the index and address of an entry point in the module's IODC, in the following format: Note that IODC was first defined with the PA-Risc 32 bith architecture. The use of "word" in the context of the length of the IODC entry point table and the size of items in the table always refers to a 32-bit word. Similarly "halfword" refers to a 16-bit quantity.

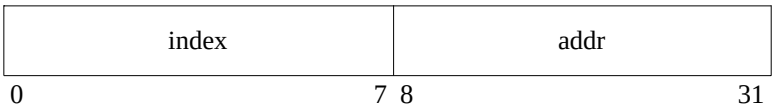


Figure 5-1. Entry Point Table Word

The *index* byte identifies the entry point. The index values in the entry point table are distinct, and are in ascending order.

The 3-byte *addr* field is the word address of the entry point code block (the byte address is 4 * *addr*). The format of the code block is as shown below. The addresses show byte offsets from the start of the block.

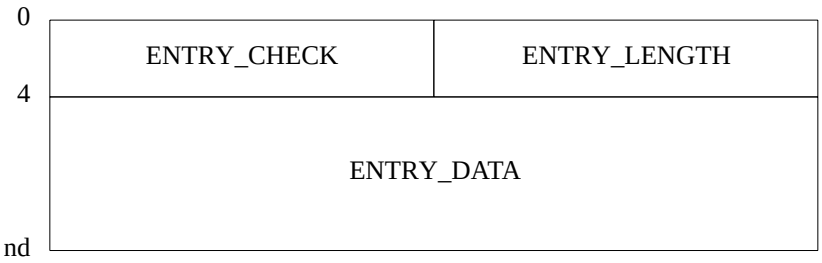


Figure 5-2. Entry Point Code Block

The ENTRY_CHECK halfword ensures that the 16-bit arithmetic sum of the *nd*/2 halfwords in the entry point code block is zero.

The ENTRY_LENGTH halfword is the number of words of data which follow. This value indirectly specifies the parameter *nd*, the number of bytes in the entry point code block, as follows:

nd = (4 * ENTRY_LENGTH) + 4

The words in ENTRY_DATA comprise a position-independent (relocatable) entry point which can be transferred by PDC_IODC from the module to memory. Following relocation, the entry point is called using the calling conventions described in Section 5.3, IODC Calling Conventions.

5.3 IODC Calling Conventions

IODC provides a procedural interface to module-type dependent code for modules other than native processors. The calling convention used for these entry points is a subset of the one defined in the *PA-RISC 64-bit Runtime Architecture*, Version 3.3.

The applicable portions of the calling convention are summarized here; refer to the *Runtime Architecture Document* for details. This subset of the calling convention used by IODC is frozen in the I/O Architecture. Indirect calls through special stubs will be required if the convention changes in the future.

The architected operation of a module (including execution of IODC) must not require the use of any non-architected IODC entry points. It must also not require the use of any non-architected options in architected IODC entry points.

The converse is also true: the execution of a non-architected IODC entry point (or a non-architected option of an architected entry point) must not affect the architected operation of a module.

5.3.1 Processor Entry/Exit State

The processor must be in the following state when calling IODC entry points:

- The processor must be at Privilege Level 0 at entry, during the call, and at exit.
- The following rules govern the use of the PSW for calls to IODC entry points:
  a. ENTRY STATE: The caller must ensure that the PSW W bit and Q bit are 1 and the T, H, L, N, B, C, M, R, E and D bits are 0. IODC may be called with the I bit set to 0 or 1. The values of the other bits are defined by the caller.
  b. DURING THE CALL: The entry point must not change the values of the S, T, H, L, C, M, R, Q, P, W, E, and D bits. IODC may optionally modify the N, X, B, V and C/B bits. See Section 5.3.3, IODC and Interruptions, for the state of the I bit.
  c. EXIT STATE: The N, X, E and B bits must be 0. The W bit must be 1. The V and C/B bits are IODC dependent. The I bit has the value at entry, upon exit of the IODC call.
- The caller of IODC must provide a doubleword-aligned value in GR 30, the stack pointer, which points to the following data:

SP-96	SAVE_ARG7
SP-88	SAVE_ARG6
SP-80	SAVE_ARG5
SP-72	SAVE_ARG4
SP-64	SAVE_ARG3
SP-56	SAVE_ARG2
SP-48	SAVE_ARG1
SP-40	SAVE_ARG0
SP-32	frame marker
SP	stack space for IODC (IODC occupies a maximum of 7 Kbytes)

  The values of SAVE_ARG0 through SAVE_ARG7 are defined by the caller at entry and IODC dependent at exit. They are used to save the first four arguments to the procedure, which are passed to the procedure in registers GR26 through GR19, respectively. Additional arguments are passed on the stack in successive locations (ARG8 at SP-104, ARG9 at SP-112, ARG10 at SP-120, etc.)

  When the called procedure returns, the value of SP must be restored.
- The memory at the stack pointer address SP and the next 7 Kbytes of larger physical addresses are available for temporary use by the IODC entry point. It is required that SP always point beyond the last byte of storage used by the caller.

PROGRAMMING NOTE

An IODC entry point allocates 7 Kbytes of temporary space on the stack to any PDC procedure that it calls (see Section 4.1.1, Entry/Exit State). Note that the IODC entry point only has 7 Kbytes of temporary space on the stack allocated to itself. The IODC entry point may optionally get this extra space by allocating a buffer within its own code space. ENTRY_TEST has another choice; it may optionally request an extra buffer space from its caller when called with ARG1 = 0 (see Section 5.4, IODC Entry Points, ENTRY_TEST page).

- The following four tables show the state of the CRs, GRs, SRs, and FPRs:

CR	Entry state	State during call	Exit state
0	Defined by caller	No WRITE	Value at Entry
1-7	HV(processor)	No READ/WRITE	HV(processor)
8	Defined by caller	No WRITE	Value at Entry
9	Defined by caller	No WRITE	Value at Entry
10	Defined by caller	No READ/WRITE	Value at Entry
11	Defined by caller	IODC dependent	IODC dependent
12	Defined by caller	No WRITE	Value at Entry
13	Defined by caller	No WRITE	Value at Entry
14	Defined by caller	No READ/WRITE	Value at Entry
15	See Section 5.3.2, Use of the EIR and EIEM by IODC		
16	Defined by caller	No WRITE	Value at Entry plus time elapsed during IODC call
17-22	Undefined	No READ/WRITE	Undefined
23	See Section 5.3.2, Use of the EIR and EIEM by IODC		
24-31	Defined by caller/IH	No READ/WRITE	Defined by caller/IH

GR	Entry state	State during call	Exit state
0	Zero	Zero	Zero
1	Defined by caller	IODC dependent	IODC dependent
2	Return address of the called procedure	IODC dependent	IODC dependent
3-18	Defined by caller	IODC dependent	Value at Entry
19	ARG7/Defined by caller	IODC dependent	IODC dependent
20	ARG6/Defined by caller	IODC dependent	IODC dependent
21	ARG5/Defined by caller	IODC dependent	IODC dependent
22	ARG4/Defined by caller	IODC dependent	IODC dependent
23	ARG3/Defined by caller	IODC dependent	IODC dependent
24	ARG2/Defined by caller	IODC dependent	IODC dependent
25	ARG1/Defined by caller	IODC dependent	IODC dependent
26	ARG0/Defined by caller	IODC dependent	IODC dependent
27	Defined by caller	IODC dependent	Value at Entry
28	Defined by caller	IODC dependent	Return Status
29	Defined by caller	IODC dependent	IODC dependent
30	Defined by caller	IODC dependent	Value at Entry
31	Defined by caller	IODC dependent	IODC dependent

SR	Entry state	State during call	Exit state
0-2	Defined by caller	IODC dependent	IODC dependent
3-7	Defined by caller	IODC dependent	Value at Entry

FPR	Entry state	State during call	Exit state
0-31	Defined by caller	No WRITE	Value at Entry

The definitions of the terms used in the above tables are as follows:

— Defined by caller. The caller of the IODC entry points can provide any value in the register/bit field, and IODC entry points must not rely on the value provided by the caller.

— IODC dependent. The state of the register/bit field depends on the IODC entry point. IODC entry points are allowed to read from and write to the registers/bit fields that are IODC dependent during the call. IODC entry points may provide any value upon exit for those registers/bit fields that are IODC dependent at exit.

— Value at Entry. The state of the register/bit field is the same as the value provided by the caller for the IODC call. For those registers/bit fields with this exit state, IODC entry points must restore the value defined by the caller upon exit of the IODC entry points, even if the state during the call was IODC dependent.

— HV(processor). The state of the register/bit field is dependent on the HVERSION of the processor.

- Registers GR26 through GR19 contain ARG0 through ARG7. If a particular entry point does not define one or more of the arguments ARG0 through ARG7, the corresponding registers are defined by the caller at entry and are IODC dependent at exit.
- The Interruption Vector Table (IVT) is defined by the caller at entry. IODC entry points must not change the Interruption Vector Table (IVT) pointed to by the IVA (CR14).

PROGRAMMING NOTE

Interruption handlers cannot rely on the contents of any register that IODC is allowed to modify during the procedure, even if IODC is required to restore the contents of the register upon exit. Specifically, the interruption handlers cannot rely on the contents of GR 30 to be the Stack Pointer.

5.3.2 Use of the EIR and EIEM by IODC

If ENTRY_INIT or ENTRY_IO can cause its module to send an interrupt message, then it must ensure that the target of that interrupt is EIR{3} in the global broadcast address space. For ENTRY_TEST, the module must interrupt only on the EIR bit specified by the *EIM_addr* argument in the ENTRY_TEST call. ENTRY_SPA and ENTRY_CONFIG must never cause its module to send interrupts.

IODC must not allow external interrupts generated by its module to be seen by the caller. Callers are not required to set EIEM{2..31} to any value. If IODC causes a module to send an interrupt, it must clear the target bit on the EIR and EIEM registers. IODC must not set any bit on these two registers to 1. Additionally, IODC must not modify any other bit on these two registers.

IODC can determine that the module sent an interrupt by observing registers like IO_II_DATA[ii] on the module. Therefore, IODC need not change the value of the IVA or the Interruption Vector Table(IVT) to which the IVA points.

PROGRAMMING NOTE

To increase supportability, IODC is encouraged to also ensure that EIR{3} is set to verify that its module sent an interrupt when expected.

If the caller has set the PSW I-bit and the target bit on the EIEM register to 1, then the caller's interruption handler must not take any action like clearing the IO_II_DATA[ii] bit on the target module, that could cause IODC to miss the interrupt it was waiting for.

Upon exit of the IODC call, the EIR register is IODC dependent. The EIEM register has the value at entry, upon exit of the IODC call.

5.3.3 IODC and Interruptions

IODC may be called with the PSW I-bit set to 0 or 1. The PSW I-bit has the value at entry, upon exit of the IODC call.

If the PSW I-bit is set to 1, then EIEM{1} must also be set to 1. IODC must set PSW I-bit to 0 if and only if it calls PDC. In such a case, the PSW I-bit must be set to 0 in the delay slot on the branch to PDC code, and IODC must also set PSW I-bit to 1 during the first instruction after the PDC call completes.

If PSW I-bit is set to 0, then there is no guarantee that the powerfail budget requirements are met if a powerfail warning occurs during the IODC call. IODC must not set the PSW I-bit to 1.

Callers must not resume an offline IODC entry point that was interrupted by a powerfail, but instead may restart the entry point from the beginning.

IODC must not be re-entered.

The execution of IODC must not cause any Group 3 or Group 4 interruptions.

5.3.4 Online IODC

The following IODC entry points can be called online in real mode:

- ENTRY_CONFIG
- ENTRY_SPA
- ENTRY_TEST (online test lists only)

Any IODC entry point that can be executed online must meet the following restrictions:

- must never set the PSW I-bit to 0 (except ENTRY_CONFIG[Get/Set SCSI Parms])
- must never call PDC procedures (except ENTRY_CONFIG[Get/Set SCSI Parms])
- must never cause its module to send interrupts
- must not change the EIEM or EIR registers
- must have a maximum size of 32 Kbytes

In addition to these restrictions, online IODC must provide support for interrupt handling, as online IODC may be called with interruptions enabled.

Following the processing of an interruption, online IODC entry points must allow the operating system to take either of the following actions:

- Resume execution of the interrupted online IODC entry point. The operating system can only do this if it can guarantee that the state of the module is unchanged as a result of interruption processing. If the state of the module can not be guaranteed to be unchanged, as is typically the case after a powerfail interrupt, the entry point must not be resumed.
- Discontinue the execution of the IODC entry point. The operating system may optionally choose not to return to the IODC entry point after interruption processing completes.

  If execution of an online IODC entry point is discontinued, the entry point must operate correctly when it is restarted, provided that the target module is in the required entry state. For example, when an online IODC entry point has been discontinued by the caller, IODC must allow the caller to restart execution of any online IODC entry point on the module, including the one that was interrupted.

5.3.5 IODC and the Operating System

The requirements on the operating system when calling IODC entry points are:

- must be called in real mode
- must run at the highest privilege level
- The operating system interruption handlers must not take any action that could cause IODC to miss the interrupt it was waiting for

Offline IODC entry points are intended to be used during boot and system configuration, not during normal system operation. Therefore, an operating system must understand the effects on system state of making offline IODC calls. The operating system may call offline IODC entry points during initial configuration and also during system shutdown. Offline IODC entry points must not be resumed after a powerfail.

The additional requirements for the operating system when calling online IODC are:

- The operating system must guarantee that the module is in a quiescent state before invoking an online IODC entry point for that module. (except ENTRY_CONFIG(Get/Set SCSI PARMS) If ENTRY_TEST is called online with *scope* = 1, all modules in the same module set as the module being tested must also be made quiescent.
- For multiprocessor systems, the operating system must guarantee that the execution of an IODC entry point is either completed or discontinued before allowing any software other than IODC to access the module. The operating system must not execute multiple IODC entry points on a module simultaneously, including multiple instances of the same entry point. If ENTRY_TEST is called online with *scope* = 1, these restrictions apply to all modules in the same module set as the module being tested.
- The operating system must not resume execution of an online IODC entry point following an interruption if it can not guarantee the state of the module is unchanged as a result of processing the interruption.

PROGRAMMING NOTE

Caution must be exercised by the operating system if an online IODC entry point is resumed after a powerfail, since during the course of a powerfail the state of any module may be affected.

The operating system may optionally restart execution of the interrupted online IODC entry point after interruption processing completes.

PROGRAMMING NOTE

Online IODC entry points are restricted from actions that would result in problems during normal system operation. Offline IODC entry points are not subject to these additional restrictions. Therefore, the operating system should be aware that if offline IODC entry points are called during normal system operation, problems may result.

5.3.6 IODC and PDC

IODC entry points are allowed to call PDC procedures. Therefore, the caller of an IODC entry point must guarantee that the following conditions are satisfied before calling the entry point:

— MEM_PDC must contain the address of the monarch processor's PDCE_PROC entry point.

— MEM_10MSEC must contain the monarch processor's number of clock ticks in 10 msec.

Only the monarch processor can execute offline IODC entry points, since these IODC entry points can call PDC. Any processor in a multiprocessor system can execute online IODC entry points, as online IODC entry points must not call PDC.

5.3.7 Standard Arguments

The base address of the module's HPA to which the IODC corresponds is specified in **ARG0**. It must be 4 Kbyte aligned. ENTRY_CONFIG[Get/Set SCSI Parms] does not use the standard argument for ARG0.

The option of the entry point is selected by **ARG1**. For architected IODC entry points, options 0 through 511 are architected or reserved; the remaining options (512 through 0xFFFFFFFF) are for SVERSION-dependent use (for bus converter ports and memory modules, these options are for HVERSION-dependent use). For SVERSION/HVERSION-dependent IODC entry points, all options are for SVERSION/HVERSION-dependent use.

Many IODC entry points use the standard argument, *spa*, to specify the base address of the module's SPA space. Its 21 most significant bits have the same format as the SPA registers. However, bits 21 through 31 of the *spa* argument must be 0. For modules that do not have SPA space, *spa* must be 0. If the *spa* argument is provided, it is **ARG2**.

Before making any IODC call which has *spa* as an argument, the caller must allocate SPA space for the target module (if the target module has SPA). That is, the *spa* argument (ARG2) must be properly aligned for the target module and the I/O address space between *spa* and *spa* + SPA size is free of address conflicts, where the SPA size is the value from the IODC_SPA[shift] field. The caller is not required to enable SPA on the target module.

Many IODC entry points use the standard argument, *ID_addr*, to specify the desired device. This argument is a pointer to a six-word LAYER data structure which describes the portion of the path to a device which is beyond the module and/or contains device-dependent information. See the format of the Primary Boot Path given in the PDC_STABLE specification for details. If the *ID_addr* argument is provided, it is **ARG3**. It must be doubleword aligned.

All IODC entry points use the standard argument *R_addr* to designate the return parameter buffer. This buffer is a doubleword-aligned block of 32 doubleword allocated by the caller. The entry point can return parameters to its caller by storing into the buffer. The *R_addr* argument is **ARG4**.

The notation 'R' is used to indicate an argument passed to an IODC entry point which is reserved for future extensions. Reserved arguments must be set by all current callers to 0, and must be ignored by all current callees. Reserved arguments may be architected in the future, with the value 0 defined to preserve compatibility with previous versions.

The notation 'HV' is used to indicate that the value of the argument is not specified by the architecture and so may be freely chosen by the caller. By contrast, arguments denoted by '---' are nonexistent: the caller is not required to provide such arguments at all. Callees must not attach any significance to 'HV' arguments and must not attempt to access '---' arguments.

5.3.8 Data Types

The data types of the standard arguments and return parameters are as follows:

- All signed integers are represented in two's complement (64-bit) format.
- The status value returned by all IODC entry points in GR28 is a signed integer.
- All addresses, which are passed as arguments, or returned as parameters, are 64-bit unsigned integers.
- The data type of ARG0 in all IODC entry points is a 64-bit unsigned integer.
- The data type of ARG1 in all IODC entry points is a 64-bit unsigned integer.

5.3.9 Return Parameters

If the entry point returns parameters to its caller, they are stored in the return parameter buffer specified by *R_addr*. The 32 returned parameters are called RET[0] through RET[31]. At least RET[0] through RET[15] are designated for architected return parameters. Return parameters in RET[16] through RET[31] which are not architecturally defined may be used for SVERSION-dependent purposes. All return parameters neither architected nor used for SVERSION-dependent purposes must be set to 0 by the IODC entry point upon return. The notation 'R' indicates a return value that must be set to 0 by the IODC entry point.

If an IODC implementation defines a new dependent return word for an entry point, the value 0 must be used to indicate "not implemented" to preserve compatibility with previous versions.

For SVERSION/HVERSION-dependent IODC entry points and SVERSION/HVERSION-dependent options of architected IODC entry points, all 32 return values (RET[0] through RET[31]) are SVERSION/HVERSION dependent.

Unless specified otherwise in the respective IODC entry point sections:

- All RET values are valid with a zero return status.
- All RET values are valid with any positive return status.
- All RET values are HVERSION dependent with any negative return status.

5.3.10 Status

The status of IODC entry points is returned as a signed integer value in register GR28.

The rest of this section applies only to architected options of architected IODC entry points. For SVERSION/HVERSION-dependent IODC entry points and SVERSION/HVERSION-dependent options of architected IODC entry points, all status values are SVERSION/HVERSION dependent.

The following status values are common to all IODC entry points:

Value	Description
0	OK
-2	Nonexistent option
-3	Cannot complete call without error
-10	Invalid argument

Positive status values (1 to 0x7FFFFFFFFFFFFFFF) are used for warnings whose meaning is dependent on the entry point that was called.

The other negative status values (-1, -4 to -9, and -11 to -0x8000000000000000) are used for errors whose meaning is dependent on the entry point that was called.

Status values other than those listed for an IODC entry point are reserved. Each IODC entry point may return only the values specifically defined for it. Reserved values can be assigned architected meanings in the future. Therefore, callers must treat the reserved negative values the same as -3 (Cannot complete call without error) and the reserved positive values the same as 0 (OK).

For status values marked as REQUIRED, all implementations of the IODC entry point are required to detect the condition specified by the status value, and to return the status value whenever the condition is detected. Values are designated as required when necessary to support the functionality of the entry point.

For status values marked as OPTIONAL, each implementation of the particular IODC entry point can choose whether or not it will detect the condition specified by the status value.

For status values marked as CONDITIONAL, each value is accompanied by a specification of the cases in which the condition must be detected and reported. There will be some IODC implementations for which those cases do not apply; they must not use the given value at all.

ENGINEERING NOTE

It is expected that those IODC implementations that are able to detect optional conditions will do so (and will return the appropriate status value).

IODC implementations are encouraged to recognize as many specific error conditions as they can.

If an implementation cannot isolate an error to one of the more specific conditions, then it must report the error by returning the general status value -3 (indicating that an indeterminate error was detected). If it cannot isolate one of the specific warning conditions, then they must return status 0 for "OK".

5.4 IODC Entry Points

This section provides a detailed description of the IODC entry points.

The indices of the IODC entry points are shown in the following table:

TABLE 5-2. IODC Entry Points

Index	Name	Description
0-2		Obsolete - HVERSION-dependent
3	ENTRY_INIT	Initialize module
4	ENTRY_IO	Module input/output
5	ENTRY_SPA	Module's extended space requirements
6-63		
64-127		Allocated for module-type dependent use
128-255		Allocated for SVERSION dependent use

ENTRY_CONFIG (index 6)

Purpose: To search for and identify entities connected to a module. Also to store and retrieve SCSI initialization parameters on SCSI cards.

Options: Options 0, 1, and 2 are required. Options 3 and 4 are required for SCSI device configuration only. If either of options 3 and 4 are implemented, then both must be implemented.

Restrictions: None

Arguments:

Description	ARG0	ARG1	ARG2	ARG3	ARG4	ARG5
Search first	hpa	0	spa	ID_addr	R_addr	R
Search next	hpa	1	spa	ID_addr	R_addr	prev_layer
Probe address	hpa	2	spa	ID_addr	R_addr	layer
Get SCSI Parms	modaddr	3	Physloc	R	R_addr	R
Set SCSI Parms	modaddr	4	SCSI_ID	Xfer	Bwidth	Autoterm

Description	ARG6	ARG7
Search first	name_addr	tic_10ms
Search next	name_addr	tic_10ms
Probe address	name_addr	tic_10ms
Get SCSI Parms	R	R
Set SCSI Parms	Physloc	R

The data type of *prev_layer*, *layer*, *SCSI_ID*, *Xfer*, *Bwidth*, *Autoterm*, and *tic_10ms* is a 64-bit unsigned integer. The *modaddr* argument is a 64-bit address variable. The *PhysLoc* argument is an 8 byte formatted entry in Physical Location format. The *name_addr* argument must be byte aligned.

Returns:

Description	RET[0]	RET[1]	RET[2]	RET[3]	RET[4]
Search first	ent_id	name_len	layer	R	R
Search next	ent_id	name_len	layer	R	R
Probe address	ent_id	name_len	R	R	R
Get SCSI Parms	SCSI_ID	Xfer	Bwidth	Autoterm	Physloc

The data type of *ent_id*, *name_len*, and *layer* is a 64-bit unsigned integer.

Status:

Value	Description
4	Autotermination setting not suported. Programmatic control of autotermination is not supported on this device. CONDITIONAL. Must be used for devices which do not support programmatic control of autotermination.
3	Unidentifiable entity An entity whose location is specified by *ID_addr* and *layer* was found, but the entity is not identifiable. Return parameter *name_len* equals zero. The contents of *ent_id*, and the buffer pointed to by *name_addr* are HVERSION dependent. CONDITIONAL. Must be used if error recovery is performed.
2	Recoverable error The call completed normally and the returned results are valid. The entry point encountered an error which it was able to correct completely. In the case of the **Get SCSI Parms** entry point, it means that the NVRAM data area on the card is uninitialized, that there is not NVRAM on the card, or that the Physical Location is invalid. In all cases, default values are returned. CONDITIONAL. Must be used if error recovery is performed.

0	OK The call completed normally and the entry point detected no error. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-4	Unrecoverable hardware error A hardware error prevented the call from completing correctly. CONDITIONAL. Must be used if hardware errors are isolated.
-6	Illegal entity address The address specified by ID_addr is invalid and cannot be used. One or more of the LAYER fields is out of range and could never be a valid entity address. OPTIONAL. Checking for illegal addresses will increase supportability.
-7	Nonexistent entity The address specified by ID_addr is valid, but it points to an entity that is not present, or an entity that does not respond. The contents of the buffer pointed to by *name_addr* are HVERSION dependent. CONDITIONAL. Must be used if nonexistent entities can be identified.
-9	Cannot locate an entity on the module. The contents of the LAYER structure and the buffer pointed to by *name_addr* are HVERSION dependent. REQUIRED.
-10	Invalid argument An argument other than ARG1 was invalid. OPTIONAL. The entry point may assume that its caller is perfect and so need not check arguments for correctness.

Entry State: The required architected state of the target module (module specified by ARG0) upon entry to ENTRY_CONFIG is listed below. IO_FLEX is initialized in all cases, and all module state other than those listed is HVERSION dependent.

- Option ARG1=0: IO_FLEX[enb] = 1
- Option ARG1=1: state unchanged from that established by the most recent previous call to ENTRY_CONFIG options ARG1=0 or 1
- Option ARG1=2: IO_FLEX[enb] = 1
- Option ARG1=3: IO_FLEX[enb] = 1
- Option ARG1=4: IO_FLEX[enb] = 1

Exit State: The required architected state of the target module (module specified by ARG0) upon exit from ENTRY_CONFIG is listed below. IO_FLEX is unchanged and all module state other than those listed is HVERSION dependent.

- Option ARG1=0: state expected by ENTRY_CONFIG option ARG1=1
- Option ARG1=1: state expected by ENTRY_CONFIG option ARG1=1
- Option ARG1=2: unchanged from state at entry
- Option ARG1=3: unchanged from state at entry
- Option ARG1=4: NVRAM on card updated to expected values

Description: ARG7, *tic_10ms*, is the number of clock ticks per 10 msec on the executing processor. This argument is intended for use in multiprocessor systems to establish timeouts.

Purpose: To locate the first entity connected to the given module.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 0
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG6	name_addr	pointer to 80-byte product name buffer
ARG7	tic_10ms	number of clock ticks per 10 msec

Returns:

Number	Name	Description
RET[0]	ent_id	entity identifier
RET[1]	name_len	length of name
RET[2]	layer	number of valid layers in the LAYER structure

Status:

Value	Description
3	Entity present, but not identifiable
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-9	Cannot locate an entity
-10	Invalid argument

Description: This option searches for the first entity connected to a module. The first entity is the one which has the lowest possible value in the *layer* return value and the lowest values in the LAYER structure pointed to by *ID_addr*.

Upon entry to the option, *ID_addr* points to a LAYER structure, but the contents of the structure are HVERSION dependent. If the option finds an entity, it deposits the path to the entity in the LAYER structure specified by ID_addr and sets the return value *layer* to the number of valid layers in the LAYER structure. *layer* must have a value greater than zero and less than seven.

The option returns an ASCII string identifying the entity in *name_addr*, and an entity identifier in *ent_id*. Note that the string is not terminated by a newline or null character.

The name may be up to 80 characters long and the length is returned in *name_len*. If *name_len* is zero, no name is available.

ENGINEERING NOTE

The name will typically be the HP product name, and is primarily intended for use by a person (not software), via utilities like IO_MAP.

It is recommended that ENTRY_CONFIG obtain the ASCII name from the entity, rather than maintain a lookup table in IODC. For example, the CS/80 Describe command, the SCSI Inquiry command, and the Esc *sˆ escape sequence for HP compatible terminals can all return an ASCII string identifying the device.

The *ent_id* return value identifies the software interface to the entity (similar to SVERSION for modules). It is unique only amongst all entities that can be connected to modules with the same SVERSION.

ENGINEERING NOTE

The *ent_id* value is primarily intended for use by software, not by any person. Many entities have the ability to return such an identifier. For example, the Amigo Id returned by many HP-IB peripherals is a 16-bit number which is unique for a particular type of peripheral.

Only SVERSION-dependent software can interpret the meaning of a particular *ent_id*, although generic software could associate a driver with the entity without necessarily understanding the exact meaning.

The path returned by this option must be used in subsequent calls to the "Search Next" option to identify other entities connected to the module.

The status -9 indicates that no entity can be located on the module, in which case the contents of the LAYER structure and the buffer pointed to by *name_addr* are HVERSION dependent.

Purpose: To locate the next entity connected to the given module.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 1
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG5	prev_layer	number of valid layers in the LAYER structure
ARG6	name_addr	pointer to 80-byte product name buffer
ARG7	tic_10ms	number of clock ticks per 10 msec

Returns:

Number	Name	Description
RET[0]	ent_id	entity identifier
RET[1]	name_len	length of name
RET[2]	layer	number of valid layers in the LAYER structure

Status:

Value	Description
3	Entity present, but not identifiable
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-9	Cannot locate an entity
-10	Invalid argument

Description: This option searches for the next entity connected to a module. The search must be depth first, that is, the search must be conducted down to the first leaf of the first branch before searching across to find other branches.

Upon entry to the option, *prev_layer* and *ID_addr* point to the entity located in the previous search using ARG1=0 or ARG1=1. If the option finds another entity, it deposits the path to the entity in the LAYER structure specified by *ID_addr*, and sets the return value *layer* to the number of valid layers in the LAYER structure. Both *prev_layer* and *layer* are greater than 0 and less than 7.

The option returns an ASCII string identifying the entity in *name_addr*, and an entity identifier in *ent_id*.

The name may be up to 80 characters long and the length is returned in *name_len*.

The *ent_id* return value identifies the software interface to the entity (similar to SVERSION for modules). It is unique only amongst all entities that can be connected to modules with the same SVERSION.

The status -9 indicates that previous calls located all the entities connected to the module.

Purpose: To identify the entity connected to the given module at the specified address.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 2
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG5	layer	number of valid layers in the LAYER structure
ARG6	name_addr	pointer to 80-byte product name buffer
ARG7	tic_10ms	number of clock ticks per 10 msec

Returns:

Number	Name	Description
RET[0]	ent_id	entity identifier
RET[1]	name_len	length of name

Status:

Value	Description
3	Entity present, but not identifiable
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-6	Illegal entity address
-7	Nonexistent entity
-10	Invalid argument

Description: This option probes for an entity connected to a module at the address specified by the *layer* parameter and the LAYER structure pointed to by the *ID_addr* parameter.

Upon entry to the option, *ID_addr* points to a LAYER structure, containing the address of entity to be identified. The entry point is not allowed to modify the contents of ID_addr. The *layer* parameter is the number of valid layers in the LAYER structure, and must have a value greater than zero and less than seven.

The option returns an ASCII string identifying the entity in *name_addr*, and an entity identifier in *ent_id*.

The name may be up to 80 characters long and the length is returned in *name_len*. If *name_len* is zero, no name is available. Note that the name is not terminated by a newline or null character. If name_len is zero, or for a negative status return, the contents of the buffer pointed to by name_addr is HVERSION dependent.

The *ent_id* return value identifies the software interface to the entity (similar to SVERSION for modules). It is unique only amongst all entities that can be connected to modules with the same SVERSION.

Purpose: To obtain the parameters stored on a SCSI card required for a SCSI device initialization.

Arguments:

Number	Name	Description
ARG0	modaddr	pointer to the path structure of the I/O card
ARG1	option	value is 3
ARG2	Physloc	Physical Location of the I/O card
ARG3	-	Reserved
ARG4	R_addr	pointer to return buffer
ARG5	-	Reserved
ARG6	-	Reserved
ARG7	-	Reserved

Returns:

Number	Name	Description
RET[0]	SCSI_ID	SCSI Initiator ID
RET[1]	Xfer	Maximum Transfer Rate
RET[2]	Bwidth	Bus Width
RET[3]	Autoterm	Autotermination Indicator
RET[4]	Physloc	I/O Card Physical Location

Status:

Value	Description
2	NVRAM uninitialized or invalid Physical Location, returning default values
0	OK
-3	Cannot complete call without error
-5	No NVRAM on card
-10	Invalid argument

Description: This option gets the SCSI parms stored on the I/O card at the module path in the path structure pointed to by the *modaddr* pointer. The Physical Location of the card is passed in ARG2, the *Physloc* input argument. This argument is checked against the Physical Location stored in NVRAM to validate the data. This option is required for I/O cards that contain a SCSI controller. It must not be implemented on other I/O cards.

The *modaddr* input argument is a memory pointer to an 8-byte structured value containing the module path of the I/O card whose SCSI parms are to be returned. The parms are returned in the memory buffer supplied by the caller in RET[0] through RET[4].

RET[0], *SCSI_ID*, is the SCSI initiator ID of the SCSI controller on the I/O card to which the request is made. *SCSI_ID* must be an integer in the range 0..15. The default value is 7.

RET[1], *Xfer*, is the maximum transfer rate (in mega-transfers per second) on the SCSI bus. The valid transfer rates depend on the particular SCSI device, and are contained in the PDC ERS for the particular platform on which the I/O card or cards are used. In any case, the parameter is an integer. The default value is the maximum rate that the SCSI controller is capable of, except on Ultra SCSI controllers (which are maximally capable of 20 mega-transfers per second) where the default is 10.

RET[2], *Bwidth*, indicates the width of the SCSI bus connected to the SCSI port on the adapter on the card. This parameter is an unsigned integer, and valid values are 8 or 16, depending on the width of the bus on the card. The default value is the bus width that the SCSI controller is capable of.

RET[3], *Autoterm* indicates whether the autotermination feature of the I/O card is enabled or disabled. This parm is an integer. A value of 1 indicates that autotermination is enabled. A value of 0 indicates that autotermination is disabled. A value of 2 indicates that the autotermination setting is not programmatically readable on the card and is hence unknown. For

programmatically readable cards the default value is 1, indicating autotermination is enabled; otherwise the default is unknown.

RET[4], *Physloc* is the Physical Location of the I/O card. It is an 8-byte formatted quantity in Physical Location format. See Section 1.2.3 of the PDC PAT ARS, Version 2.5 or later, for the format of this field. The default value is the input argument value.

When this procedure is called to return the SCSI parms, it should check the system serial number, which **Set SCSI Parms** has previously stored in card NVRAM before returning values. If the data area is uninitialized or the serial number is not correct, the call should return default values for all the parms, and a Return Status of 2, to indicate that default values are being returned. This procedure should also check for invalid Physical Location by checking the stored value versus the ARG2, *Physloc*. If they do not match, the procedure should return default values for all the parms and a Return Status of 2.

Purpose: To store initialization parameters required for SCSI devices on NVRAM located on the I/O card. This information will be subsequently read at each boot when the card is initialized.

Arguments:

Number	Name	Description
ARG0	modaddr	pointer to structure contain module path of I/O card
ARG1	option	value is 4
ARG2	SCSI_ID	SCSI initiator ID
ARG3	Xfer	Maximum Transfer Rate
ARG4	Bwidth	Width of SCSI bus
ARG5	Autoterm	Autotermination indicator
ARG6	Physloc	I/O Card Physical Location
ARG7	-	Reserved

Returns: This options has no return parameters.

Status:

Value	Description
4	Autoterm disabling not supported
2	Invalid Physical Location
0	OK
-3	Cannot complete call without error
-5	No NVRAM on card
-10	Invalid argument

Description: This option stores the SCSI parms required for the OS to initialize an I/O card at the module path in the path structure pointed to by the *modaddr* pointer. The parms are stored in NVRAM on the card for subsequent use by the OS. This option is required for I/O cards that contain a SCSI controller. It must not be implemented on other I/O cards.

The parms stored are ARG2 through ARG6. These are the same variables as are returned in RET[0] through RET[4] for **Get SCSI Parms**.

For each parm being stored, a value of -2 in the corresponding input argument means do not change the parm, a value of -1 in the input argument means to set the parm to its default value. These values of input arguments are not valid for ARG6, *Physloc*. It must always be the Physical Location of the card.

ARG2, *SCSI_ID*, is the SCSI initiator ID of the SCSI controller. This input argument is a signed integer. Valid values are 0 through 15, or -2 indicating don't change, or -1 indicating use the default. The default is 7.

ARG3, *Xfer*, is the maximum transfer rate on the SCSI bus. Units are mega-transfers per second. This argument is a signed integer. Valid values for Xfer depend on the SCSI protocol being used, and are given in the PDC ERS for each individual platform. The input argument must either be a valid value, or -2 indicating don't change, or -1 indicating use the default value as defined in **Get SCSI Parms**.

ARG4, *Bwidth* is the width of the SCSI bus on the card. *Bwidth* is a signed integer. Valid values for this argument are 8, 16, or -2, indicating don't change or -1 indicating use the default, or 0 indicating that the card default should be used (chose by the driver). The ENTRY_CONFIG default is the capable width of the adapter.

ARG5, *Autoterm* is the indicator of whether autotermination should be enabled or disabled. Autoterm is a signed integer. Valid values for this argument are 0, indicating to disable autotermination, 1, indicating to enable autotermination, -1 indicating to use the default value ("enabled"), or -2, indicating not to change the autotermination indicator.

ARG6, *Physloc* is the Physical Location of the I/O card in the complex. *Physloc* is an 8-byte formatted value in Physical Location format. There is no default value.

When this option is called to write the SCSI parms to the NVRAM on the card, it is expected to obtain the system serial number of the computer that is is running on and store it along with the parms. Subsequent calls to the **Get SCSI Parms** option should check the stored serial number against the serial number of the computer as obtained from PDC_MODEL.

NOTE

This option and the option **Get SCSI Parms** should be used in combination with the **PDC_SCSI_PARMS** call. Please see the documentation of that call for more information on the calling requirements.

ENTRY_INIT (index 3)

Purpose: To initialize and test a console module or boot module so that ENTRY_IO can be used to transfer data to/from the module and to establish a module-device connection.

Options: Options ARG1=2 and ARG1=3 are optional. However, if either of them is implemented, the other option must be implemented as well. The three options ARG1=4 through ARG1=6 are all required. The option ARG1=9 is optional.

Restrictions: The length of the ENTRY_INIT entry point must not exceed 16 Kbytes.

ENTRY_INIT must not enable the SPA space(s) to any value outside of the range determined by the *spa* argument and IODC_SPA[shift].

Arguments:

Description	ARG0	ARG1	ARG2	ARG3	ARG4	ARG5
Search first	hpa	2	spa	ID_addr	R_addr	---
Search next	hpa	3	spa	ID_addr	R_addr	---
Init & test mod & dev	hpa	4	spa	ID_addr	R_addr	HV
Init & test dev	hpa	5	spa	ID_addr	R_addr	HV
Init & test mod	hpa	6	spa	HV	R_addr	HV
Return message	hpa	9	spa	ID_addr	R_addr	data_addr

Description	ARG6	ARG7	ARG8	ARG9	ARG10	ARG11
Search first	---	---	---	---	---	---
Search next	---	---	---	---	---	---
Init & test mod & dev	HV	HV	lang	---	---	---
Init & test dev	HV	HV	lang	---	---	---
Init & test mod	HV	HV	lang	---	---	---
Return message	msg_addr	R	lang	R	R	R

The data type of *lang* is a 32-bit unsigned integer. The arguments *data_addr* and *msg_addr* must be word aligned.

Returns:

Description	RET[0]	RET[1]	RET[2]	RET[3]
Search first	R	class	net_id_hi	net_id_lo
Search next	R	class	net_id_hi	net_id_lo
Init & test mod & dev	stat	class	net_id_hi	net_id_lo
Init & test dev	stat	class	net_id_hi	net_id_lo
Init & test mod	stat	R	R	R
Return message	msg_size	R	R	R

The data type of *net_id_lo*, *net_id_hi*, *stat* and *msg_size* is a 32-bit unsigned integer. The data type of *class* is a 4-bit unsigned integer.

Status:

Value	Description
2	Recoverable error The call completed normally and the returned results are valid. The entry point encountered an error which it was able to correct completely. CONDITIONAL. Must be used if error recovery is performed.
0	OK The call completed normally and the entry point detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option implemented by the entry point. REQUIRED.

Value	Description
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-4	Unrecoverable hardware error A hardware error on the module or device prevented the call from completing correctly. CONDITIONAL. Must be used if hardware errors are isolated.
-5	Unrecoverable data error The entry point encountered an error while transferring data to or from the device. Returned only by options ARG1=4 and ARG1=5. OPTIONAL. May be used if hardware has the capability to isolate data errors.
-6	Illegal device address The device address specified by *ID_addr* is invalid and cannot be used. One or more of the LAYER fields is out of range and could never be a valid device address. Returned only by options ARG1=4 and ARG1=5. OPTIONAL. Checking for illegal addresses will increase supportability.
-7	Nonexistent device The device address specified by *ID_addr* is a valid device address. However, it points to either a device that is not installed or a device that does not respond. Returned only by options ARG1=4 and ARG1=5. CONDITIONAL. Must be used if nonexistent devices can be identified.
-8	Module/device not ready The module or device is not ready to be initialized or tested. A module may not be ready because it is still performing its selftest after reset or power-on. An example of device not ready is a disk drive that is still spinning up. Returned only by options ARG1=4 through ARG1=6. OPTIONAL. Implementations may use this status if waiting for the module or device to become ready would make the call take longer than 5 seconds.
-9	Cannot locate a console device or boot device The search could not locate a console device or boot device on the module. Returned only by options ARG1=2 and ARG1=3. CONDITIONAL. Must be used if the search options are provided.
-10	Invalid argument An argument other than ARG1 was invalid. OPTIONAL. The entry point may assume that its caller is perfect and so need not check arguments for correctness.
-13	Protocol error A protocol violation was encountered on the module-device connection while transferring data to or from the device. Returned only by options ARG1=4 and ARG1=5. CONDITIONAL. Must be used if the implementation can detect a protocol error.
-14	Invalid or uninitialized SCSI parms in NVRAM. when initializing a SCSI module, either the SCSI parms in system NVRAM or module NVRAM were unitialized, or they did not agree. Returned only by options ARG1=4 and ARG1=6. CONDITIONAL. Must be used both the card and system supports NVRAM for SCSI parms.

ENGINEERING NOTE

The status returns -3, -4, and -5 are progressively more specific about the nature of an error. If the device or module hardware provides an indication to IODC that a problem with data transfer has occurred, then a status return of -5 is appropriate. If the hardware indicates that a hardware failure occurred, then a status of -4 is returned. It may also

correspond to a data error that was not isolated. A status return of -3 indicates an unspecified error which might have been a data error or a hardware error.

Entry State: The required architected state of the target module (module specified by ARG0) upon entry to ENTRY_INIT is listed below. IO_FLEX is initialized in all cases, and all module state other than those listed is HVERSION dependent.

- Option ARG1=2: state unchanged from that established by the most recent previous call to ENTRY_INIT options ARG1=4, 5, or 6
- Option ARG1=3: state unchanged from that established by the most recent previous call to ENTRY_INIT options ARG1=2 or 3
- Option ARG1=4: IO_FLEX[enb] = 1
- Option ARG1=5: state unchanged from that established by the most recent previous call to ENTRY_INIT options ARG1=4, 5, or 6
- Option ARG1=6: IO_FLEX[enb] = 1
- Option ARG1=9: IO_FLEX[enb] = 1

Exit State: The required architected state of the target module (module specified by ARG0) upon exit from ENTRY_INIT is listed below. IO_FLEX is unchanged and all module state other than those listed is HVERSION dependent.

- Option ARG1=2: state expected by ENTRY_INIT options ARG1=3 and 5
- Option ARG1=3: state expected by ENTRY_INIT options ARG1=3 and 5
- Option ARG1=4: state expected by ENTRY_IO options ARG1=0 and 1 for boot devices; state expected by ENTRY_IO options ARG1=2 and 3 for console devices.
- Option ARG1=5: state expected by ENTRY_IO options ARG1=0 and 1 for boot devices; state expected by ENTRY_IO options ARG1=2 and 3 for console devices. The state of devices other than the one specified by *ID_addr* is not altered.
- Option ARG1=6: state expected by ENTRY_INIT options ARG1=2 and 5
- Option ARG1=9: unchanged from state at entry

Description: **Time Limits**

There is no time limit on the Search options (ARG1=2 and ARG1=3) of ENTRY_INIT. Those options take as much time as is necessary to locate a console device or boot device on the given module. However, the device that is found need not be ready for use. Thus the entry point should return immediately after locating the device rather than wait for it to become ready for use.

The Init & test options (ARG1=4 through ARG1=6) of ENTRY_INIT do not have any architecturally specified time limit. However, the entry point must know the upper bound required to complete initialization and test and establish a timeout accordingly. That is, ENTRY_INIT must not allow a defective module or device to hang the system. In some cases, the entry point may not be able to begin the initialization process because a module or device is not ready. For example, ENTRY_INIT may not be able to begin until a disk has spun up or a modem connection is established. The Init & test options should make an effort to detect cases in which a device is not yet ready. If the entry point determines that its device is preparing for use, then it should immediately return -8 (Module/device not ready). ENTRY_INIT should not wait for its device to become ready, rather it should immediately return -8 whenever its device is not yet ready. The case of a disk being offline is NOT a situation in which the device is not ready; ENTRY_INIT should return -4 (Unrecoverable hardware error) when its device is offline. (The difference between status values -4 and -8 is that when the status is -8, it is expected that a future call to ENTRY_INIT can succeed without any operator intervention, whereas when the status is

-4, operator intervention is required to fix the problem.)

Required Initialization Functions

The Init & test options that pertain to devices (ARG1=4 and ARG1=5) are required to initialize the state of the device. If the device is a console, the cursor position and the display status are HVERSION dependent after ENTRY_INIT completes. In the case of a boot device of the sequential class, the position of the medium is HVERSION dependent after ENTRY_INIT completes.

The options ARG1=4 and ARG1=5 establish module-device connections. A module-device connection (for example, the link between the boot module and the boot device) is a communication link that is established between a module and a device. A module-device connection is closed by ENTRY_IO option ARG1=4.

ENGINEERING NOTE

If required for testing, ENTRY_INIT may disable the SPA. The original SPA base address must be restored after a successful test.

Return Values

The Search First, Search Next, Init & Test mod & dev, and Init & Test dev (ARG1=2 through ARG1=5) of ENTRY_INIT return in RET[2] and RET[3] the value of the station address for I/O modules with network connections:

Number	Name	Description
RET[2]	net_id_hi	Most significant 16 or 32 bits of station address
RET[3]	net_id_lo	Least significant 16 or 32 bits of station address

net_id_hi and *net_id_lo* are optional, but if they are returned by one of these four options to ENTRY_INIT, they must be returned by any of the four options which are implemented. They must not be returned by any other ENTRY_INIT options.

If *net_id_hi* and *net_id_lo* both have a value of zero, then either the I/O module does not have a network connection or a station address has not been returned.

The station address may be a 16, 48, or 64 bit unsigned integer.

If *net_id_hi{0..31}* is zero and and *net_id_lo{0..15}* is zero, but *net_id_lo{15..31}* is non-zero, then a 16 bit station address has been returned.

If *net_id_hi{0..15}* is zero, but *net_id_hi{16..31}* or *net_id_lo* is non-zero, then a 48 bit station address has been returned.

If *net_id_hi{0..15}* is non-zero, then a 64 bit station address has been returned.

ENGINEERING NOTE

PDCE_RESET should display the station address on the console in hexadecimal. One of the following formats is suggested:

Length	net_id_hi	net_id_lo	Display
16 bits	`00000000`	`0000bbaa`	`000000-00bbaa`
48 bits	`0000ffee`	`ddccbbaa`	`ffeedd-ccbbaa`

64 bits	hhggffee	ddccbbaa	hhgg-ffeedd-ccbbaa

ENGINEERING NOTE

Note that while the station address is potentially a double-word quantity, it is not guaranteed that it will be double-word aligned, since R-addr need only be word aligned.

PROGRAMMING NOTE

The intended use of the Search options (ARG1=2 and ARG1=3) is shown in the following algorithm to perform autosearch for boot device:

1. Call ENTRY_INIT with ARG1=6 to initialize and test the boot module to be searched.
2. Call ENTRY_INIT with ARG1=2 to search for the first console device or boot device on the module.
3. If the status is -9, then terminate the search unsuccessfully.
4. Check the return parameter *class* to verify that the device is appropriate for boot. If *class* is not appropriate (because the search found a console device), skip step 5 and continue with step 6.
5. Call ENTRY_INIT with ARG1=5 to initialize and test the device. If the call was successful, then terminate the search having found the boot device, else continue with step 6.
6. Call ENTRY_INIT with ARG1=3 to search for the next console device or boot device on the module.
7. Return to step 3.

Purpose: To locate the first console device or boot device on the given module.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 2
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer

Returns:

Number	Name	Description
RET[1]	class	device class
RET[2]	net_id_hi	most significant 16 or 32 bits of station address
RET[3]	net_id_lo	least significant 16 or 32 bits of station address

Status:

Value	Description
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-9	Cannot locate console device or boot device
-10	Invalid argument

Description: Upon entry to the option, *ID_addr* points to a LAYER structure, but the contents of that structure need not be initialized to any particular value. If the option finds a console device or boot device, it deposits the path to the device in the LAYER structure specified by *ID_addr*. This path can be used in subsequent calls to ENTRY_INIT or ENTRY_IO. The status -9 indicates that no console device or boot device can be located on the module, in which case the contents of the LAYER structure are HVERSION dependent.

Return value *class* indicates the device class. Device classes are defined in Section C.2, Data Format of Page Zero; the relevant table is reproduced below for convenience.

Value	Name	Description
0	CL_NULL	Invalid
1	CL_RANDOM	Random access media (as in disk)
2	CL_SEQU	Sequential record access media (as in tape)
3 - 6	Reserved	Reserved
7	CL_DUPLEX	Full duplex point-point communication (as in RS-232)
8	CL_KEYBD	Half-duplex console (Keyboard In)
9	CL_DISPL	Half-duplex console (Display Out)
10 - 15	Reserved	Reserved

Purpose: To locate the next console device or boot device on the module.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 3
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer

Returns:

Number	Name	Description
RET[1]	class	device class
RET[2]	net_id_hi	most significant 16 or 32 bits of station address
RET[3]	net_id_lo	least significant 16 or 32 bits of station address

Status:

Value	Description
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-9	Cannot locate console device or boot device
-10	Invalid argument

Description: Upon entry to the option, *ID_addr* points to the device located in the previous search using ARG1=2 or ARG1=3. When the call returns, the LAYER structure specified by *ID_addr* points to the next console device or boot device on the module. The status -9 indicates that previous calls located all console devices and boot devices on the module.

Return parameter *class* indicates the device class. (See the table of device classes on the page for the "Search first" option.)

Purpose: To test and initialize the module and a specific device.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 4
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG8	lang	future use as language specifier

Returns:

Number	Name	Description
RET[0]	stat	module status
RET[1]	class	device class
RET[2]	net_id_hi	most significant 16 or 32 bits of station address
RET[3]	net_id_lo	least significant 16 or 32 bits of station address

Status:

Value	Description
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-5	Unrecoverable data error
-6	Illegal device address
-7	Nonexistent device
-8	Module/device not ready
-10	Invalid argument
-13	Protocol error

Description: The option tests and initializes the module specified by *hpa* and the device pointed to by *ID_addr*. The duration of the test is known by ENTRY_INIT, but no time limit is imposed by the I/O Architecture. This test is a GO/NO GO test: it need not isolate the failed FRU. A test which passes does not guarantee totally correct module operation but means that errors which occur in ENTRY_IO will be detected. The initialization leaves the device in a state so that ENTRY_IO can be used to transfer data. The only permitted value for *lang* is 0.

This option establishes a module-device connection.

For all positive and zero status returns, RET[0] contains the value of IO_STATUS at the time of exit. For a status return of -5, -7, -8, or -13, RET[0] contains the value of IO_STATUS at the the time of exit. RET[0] is HVERSION dependent for a status return of -2, -3, -4 -6, or -10.

Return parameter *class* indicates the device class. (See the table of device classes on the page for the "Search first" option.)

Purpose: To test and initialize a specific device.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 5
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG8	lang	future use as language specifier

Returns:

Number	Name	Description
RET[0]	stat	module status
RET[1]	class	device class
RET[2]	net_id_hi	most significant 16 or 32 bits of station address
RET[3]	net_id_lo	least significant 16 or 32 bits of station address

Status:

Value	Description
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-5	Unrecoverable data error
-6	Illegal device address
-7	Nonexistent device
-8	Module/device not ready
-10	Invalid argument
-13	Protocol error

Description: The option tests and initializes the device pointed to by *ID_addr*. The duration of the test is known by ENTRY_INIT, but no time limit is imposed by the I/O Architecture. This test is a GO/NO GO test: it need not isolate the failed FRU. A test which passes does not guarantee totally correct module operation but does guarantee that errors which occur in ENTRY_IO will be detected. The initialization leaves the device in a state so that ENTRY_IO can be used to transfer data. The only permitted value for *lang* is 0.

This option establishes a module-device connection.

For all positive and zero status returns, RET[0] contains the value of IO_STATUS at the time of exit. For a status return of -5, -7, -8, or -13, RET[0] contains the value of IO_STATUS at the the time of exit. RET[0] is HVERSION dependent for a status return of -2, -3, -4, -6, or -10.

Return parameter *class* indicates the device class. (See the table of device classes on the page for the "Search first" option.)

Purpose: To test and initialize the module.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 6
ARG2	spa	SPA of the module
ARG4	R_addr	pointer to return buffer
ARG8	lang	future use as language specifier

Returns:

Number	Name	Description
RET[0]	stat	module status

Status:

Value	Description
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-8	Module/device not ready
-10	Invalid argument

Description: The option tests and initializes the module specified by *hpa*. The duration of the test is known by ENTRY_INIT, but no time limit is imposed by the I/O Architecture. This test is a GO/NO GO test: it need not isolate the failed FRU. A test which passes does not guarantee totally correct module operation but does guarantee that errors which occur in ENTRY_IO will be detected. The only permitted value for *lang* is 0.

For all positive and zero status returns, RET[0] contains the value of IO_STATUS at the time of exit. For a status return of -5, -7, -8, or -13, RET[0] contains the value of IO_STATUS at the the time of exit. RET[0] is HVERSION dependent for a status return of -2, -3, -4, -6, or -10.

Purpose: To ask for a message when the status from the previous call to the entry point was nonzero.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 9
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG5	data_addr	pointer to previous return buffer
ARG6	msg_addr	pointer to message buffer
ARG8	lang	future use as language specifier

Returns:

Number	Name	Description
RET[0]	msg_size	number of bytes returned in the message buffer

Status:

Value	Description
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-10	Invalid argument

Description: The old *R_addr* pointer from the previous call is supplied as argument *data_addr* to the new option. The SVERSION-dependent part of the old *R_addr* (the last 16 words) contains all the information the option needs to return the appropriate message (the entry point does NOT need to look at the state of its module; the values in *data_addr* are sufficient). The arguments *hpa*, *spa*, and *ID_addr* must be the same as in the previous call which yielded the nonzero status. The size of the buffer pointed to be *msg_addr* must be 2 Kbytes. The only allowed value for *lang* is 0.

If *msg_size* is 0, no message was returned. The text of the returned message must not exceed one screen (24 lines).

ENTRY_IO (index 4)

Purpose: To perform basic I/O with boot devices and console devices so that the system can boot in a device-independent fashion, and to close module-device connections.

Calls to the boot device and to the console device have been distinguished, to allow a single device to be used for either functionality (possibly using different protocols).

Options: The option ARG1=0 is required of all boot modules, but option ARG1=1 is optional. For a duplex console module, the pair of options ARG1=2 and ARG1=3 are required. A simplex console module need implement only the appropriate half of the pair. The option ARG1=4 must be implemented by a module if it opens a module-device connection that needs to be closed in the future. The option ARG1=9 is optional.

Restrictions: The length of the ENTRY_IO entry point must not exceed 16 Kbytes.

Arguments:

Description	ARG0	ARG1	ARG2	ARG3	ARG4	ARG5
Boot input	hpa	0	spa	ID_addr	R_addr	devaddr
Boot output	hpa	1	spa	ID_addr	R_addr	devaddr
Console input	hpa	2	spa	ID_addr	R_addr	HV
Console output	hpa	3	spa	ID_addr	R_addr	HV
Close connection	hpa	4	spa	ID_addr	R_addr	R
Return message	hpa	9	spa	ID_addr	R_addr	data_addr

Description	ARG6	ARG7	ARG8	ARG9	ARG10	ARG11
Boot input	memaddr	reqsize	maxsize	---	---	---
Boot output	memaddr	reqsize	---	---	---	---
Console input	memaddr	reqsize	lang	---	---	---
Console output	memaddr	reqsize	lang	---	---	---
Close connection	R	R	R	R	R	R
Return message	msg_addr	R	lang	R	R	R

The data type of *reqsize*, *maxsize*, and *lang* is a 32-bit unsigned integer. The arguments *data_addr* and *msg_addr* must be word aligned.

Returns:

Description	RET[0]
Boot input	count
Boot output	count
Console input	count
Console output	count
Return message	msg_size

The data type of *count* and *msg_size* is a 32-bit unsigned integer.

Status:

Value	Description
3	EOF encountered The call completed normally and the returned results are valid. The entry point encountered an EOF (end of file) during the last boot input operation. Returned only by option ARG1 = 0, and only when the device has class CL_SEQU. OPTIONAL. Implementations may use this status if they can detect an EOF when executing boot input from a CL_SEQU device.
2	Recoverable error The call completed normally and the returned results are valid. The entry point encountered an error which it was able to correct completely.

Value	Description
	CONDITIONAL. Must be used if error recovery is performed.
1	Inexact I/O transfer The amount of data transferred was not exactly the same as specified by the *reqsize* argument. Returned only by options ARG1=0 and ARG1=1. REQUIRED.
0	OK The call completed normally and the entry point detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option implemented by the entry point. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-4	Unrecoverable hardware error A hardware error prevented the call from completing correctly. CONDITIONAL. Must be used if hardware errors are isolated.
-5	Unrecoverable data error An error was encountered while transferring data to or from the device. CONDITIONAL. May be used if hardware has the capability to isolate data errors.
-6	Illegal device address The device address specified by *ID_addr* is invalid and cannot be used. One or more of the LAYER fields is out of range and could never be a valid device address. Returned only by options ARG1=0 through ARG1=4. OPTIONAL. Checking for illegal addresses increases supportability.
-7	Nonexistent device The device address specified by *ID_addr* is a valid device address. However, it points to either a device that is not installed or a device that does not respond. Returned only by options ARG1=0 through ARG1=4. CONDITIONAL. Must be used if nonexistent devices can be identified.
-8	Module/device not ready The module or device is not ready to do I/O. Returned only by options ARG1=0 through ARG1=4. OPTIONAL. Implementations may use this status if waiting for the module or device to become ready would make the call take longer than 5 seconds.
-10	Invalid argument An argument other than ARG1 was invalid. OPTIONAL. The entry point need not check arguments for correctness.
-11	Data buffer too small Medium is formatted with a record size larger than the data buffer. Returned only by option ARG1=0, and only when the device has class CL_SEQU. REQUIRED.
-12	Unsupported record size Requested record size not supported by the device. Returned only by options ARG1=0 and ARG1=1, and only when the device has class CL_SEQU. CONDITIONAL. Must be used if legal values of *reqsize* are not supported.
-13	Protocol error A protocol violation was encountered on the module-device connection while transferring data to or from the device. Returned only by options ARG1=0 through ARG1=4.

Value	Description
	CONDITIONAL. Must be used if the implementation can detect a protocol error.
-14	Illegal device block size The device is formatted with a block size that is not a factor of 2 Kbytes. Returned only by options ARG1 = 0 and ARG1 = 1, and only when the device has class CL_RANDOM. CONDITIONAL. Must be used if illegal device block sizes are detected.

ENGINEERING NOTE

The status returns -3, -4, and -5 are progressively more specific about the nature of an error. If the device or module hardware provides an indication to IODC that a problem with data transfer has occurred, then a status return of -5 is appropriate. If the hardware indicates that a hardware failure occurred, then a status of -4 is returned. It may also correspond to a data error that was not isolated. A status return of -3 indicates an unspecified error which might have been a data error or a hardware error.

Entry State: The required architected state of the target module (module specified by ARG0) upon entry to ENTRY_IO is listed below. IO_FLEX is initialized in all cases, and all module state other than those listed is HVERSION dependent.

- Option ARG1=0: state unchanged from that established by the most recent previous call to ENTRY_INIT options ARG1=4 or 5 or ENTRY_IO options ARG1=0, 1, 2, or 3
- Option ARG1=1: state unchanged from that established by the most recent previous call to ENTRY_INIT options ARG1=4 or 5 or ENTRY_IO options ARG1=0, 1, 2, or 3
- Option ARG1=2: state unchanged from that established by the most recent previous call to ENTRY_INIT options ARG1=4 or 5 or ENTRY_IO options ARG1=0, 1, 2, or 3
- Option ARG1=3: state unchanged from that established by the most recent previous call to ENTRY_INIT options ARG1=4 or 5 or ENTRY_IO options ARG1=0, 1, 2, or 3
- Option ARG1=4: state unchanged from that established by the most recent previous call to ENTRY_IO options ARG1=0, 1, 2, or 3
- Option ARG1=9: IO_FLEX[enb] = 1

Exit State: The required architected state of the target module (module specified by ARG0) upon exit from ENTRY_IO is listed below. IO_FLEX is unchanged and all module state other than those listed is HVERSION dependent.

- Option ARG1=0: state expected by ENTRY_IO options ARG1=0, 1, 2, 3, and 4
- Option ARG1=1: state expected by ENTRY_IO options ARG1=0, 1, 2, 3, and 4
- Option ARG1=2: state expected by ENTRY_IO options ARG1=0, 1, 2, 3, and 4
- Option ARG1=3: state expected by ENTRY_IO options ARG1=0, 1, 2, 3, and 4
- Option ARG1=4:
  - — If the console connection was closed, then the state expected by ENTRY_INIT options ARG1=2, 3, 4, and 5
  - — If the boot connection was closed, then the state expected by ENTRY_INIT options ARG1=2, 3, 4, and 5
- Option ARG1=9: unchanged from state at entry

Description: **Responsibilities of the Caller**

The caller of ENTRY_IO is responsible for verification of input parameters. If alignment or value constraints are not met, the effects of the call are HVERSION dependent and errors will not necessarily be detected.

If ENTRY_IO returns with status -3 (Cannot complete call without error) or -4 (Unrecoverable hardware error) or -5 (Unrecoverable data error), the caller must call ENTRY_INIT to initialize the module and device before retrying ENTRY_IO.

If the boot device has class CL_SEQU, the caller must ensure that the medium is rewound before accessing the device. Rewind is accomplished by calling ENTRY_IO (ARG1=0 or 1) with *devaddr*=0. (The call that rewinds can also transfer data.)

When reading IPL from a device of class CL_SEQU for which the record size is unknown, the caller of ENTRY_IO with ARG1=0 is advised to allocate an input buffer of 64 Kbytes (*maxsize* should be 64 Kbytes). That ensures that the buffer is large enough, because the IPL record size can never exceed 64 Kbytes.

The caller of ENTRY_IO with ARG1=0 or ARG1=1 must always check the return parameter *count* and react appropriately if ENTRY_IO did not transfer all the data that was requested.

PROGRAMMING NOTE

Here is an example of the steps that a caller of ENTRY_IO could follow to transfer a block of data of SIZE bytes which is stored in memory at location MEM to address DEV on an output device of class CL_RANDOM:

```
while (SIZE > 0) {
      status ← ENTRY_IO(1, spa, ID_addr, R_addr, DEV, MEM, SIZE);
      if status < 0 then handle_error(....);
      SIZE ← SIZE – RET[0];
      MEM  ← MEM  + RET[0];
      DEV  ← DEV  + RET[0];
}
```

Conflicts with Data Contained in the Cache(s)

For data output, ENTRY_IO must flush the data buffer in the D-cache before DMA begins. For data input, ENTRY_IO must flush/purge the data buffer in the D-cache before DMA begins and must also flush/purge the data buffers in the D-cache and the I-cache after DMA completes.

The flush/purge before DMA begins must be to lines in the data buffer between *memaddr* and *memaddr* + *count*, if the value of *count* is known before DMA begins. If the value of *count* is not known, the flush/purge must be to lines in the data buffer between *memaddr* and *memaddr* + *maxsize*. After DMA completes, the flush/purge must be to lines in the data buffer between *memaddr* and *memaddr* + *count*.

A SYNC instruction must follow each flush/purge or groups of flush/purge instructions.

Following the flush/purge to the I-cache, IODC must ensure that there are at least 8 instructions between the SYNC instruction and the end of the IODC code, or IODC must execute at least 8 instructions before returning to the caller.

Timeouts

The ENTRY_IO entry point is required to timeout all requests. The timeout can be based on the length of the requested transfer size, or can be a fixed value based on the maximum supported transfer size.

The caller of ENTRY_IO must not put a timeout on the duration of the call.

ENGINEERING NOTE

Implementations are encouraged to complete each ENTRY_IO call within 5 seconds whenever possible. Adherence to this guideline allows the caller of ENTRY_IO the opportunity to report forward progress through the console and/or chassis display. However, there are a number of events that may make it impossible for ENTRY_IO to guarantee that it will return within 5 seconds. Listed below are some examples of these events:

- Transactions to or from the console that are paused by a pacing mechanism, such as XON/XOFF.
- Tape rewind.
- Lock-outs on I/O systems that are shared by multiple masters. A-LINK and LANs are examples of I/O systems that can have multiple masters.
- Device seek time.
- Automatic read/write retries not explicitly controlled by ENTRY_IO.
- Device internal maintenance. As an example, HP793x disk drives can go offline for 2 or 3 seconds to perform maintenance.

Console Model

The following features are true of the console I/O options to ENTRY_IO:

- All input characters are returned to the caller as a series of ASCII bytes.
- All output characters are passed to the entry point as a series of ASCII bytes.
- Maintenance of the console cursor is SVERSION dependent.
- The minimum screen size is 24 lines x 80 columns per line
- The action of the following ASCII characters is defined:

ASCII Char	Value	Action
BEL	07	Audible beep or does nothing
BS	08	Moves the cursor one character position to the left. The erasure of the backspaced character is HVERSION dependent. If the cursor is at the beginning of a line there is no action.
LF	10	Moves the cursor to the same position in the next line. If the next line is beyond the bottom of the screen, the action is to scroll the screen one line up. The top line of the screen could be lost.
CR	13	Moves cursor to the beginning of the current line.
SP	32	The character is displayed at the current cursor position and the cursor moves on position to the right. If the cursor is at the rightmost position in the line, the action is HVERSION dependent.

Definitions of values 33 through 126 are similar to that of SP (they are the usual printing characters). IODC is not required to make any checks for unspecified characters.

Console Input: The receipt of unspecified characters is console input device dependent.
Console Output: The output of unspecified characters is console output device dependent.

PROGRAMMING NOTE

Only 79 characters can be printed in each line, since the resulting action when the cursor is in the rightmost position is HVERSION dependent.

ENGINEERING NOTE

For bit-mapped displays, the cursor position must be available to ENTRY_IO. The cursor position can be stored in module registers or in a nonvisible part of the display RAM.

Console Flow Control

The ENTRY_IO options are modeled after half-duplex devices. The ENTRY_IO caller is assumed to provide all character echoing, erase, and new line processing.

Some console/display devices may use special flow control characters (e.g., XON/XOFF) to regulate the pace at which data can be transmitted to the console on output. If the hardware device can generate those flow control characters, then the ENTRY_IO code must handle them in a manner transparent to its caller. Specifically, ENTRY_IO must pause its output when the stop signal is received, and continue only when the start signal is received. When ENTRY_IO is called, it starts in a state in which transmission is enabled, and it returns only after its entire output message has been sent and transmission is once again enabled.

On the other hand, it is the caller of ENTRY_IO for console output that must ensure that a new screenful of data does not overwrite previous output before the user is ready for it. The architecture does not require support for any special user flow control characters, such as Control_Q/Control_S. Rather, the pace of console output is governed by a paging model. Before a caller can send a console message that would overwrite a previous screen or cause lines to be scrolled off the screen, it must first prompt the user to give some input indicating that it is all right to do so. Note that the minimum console/display is required to have 24 lines, so a program is required to prompt the user whenever more than 24 lines of output are generated without any intervening user input. The IODC entry points that can return error messages for display must therefore limit those messages to strictly less than 24 lines. PDC must ensure that it can complete an automatic boot (autoboot or autosearch) without requiring any user input at the console. Callers which do not send more than 24 lines to the console do not need to implement the paging model.

ENTRY_IO is not required to support "type ahead". That is, it is allowed for ENTRY_IO to cause a duplex console device to discard characters, if those characters were typed in during a "Console Output" call.

To simplify ENTRY_IO implementations, the caller must convert a carriage return (CR) into a carriage return and line feed sequence (CR, LF) before echoing.

Purpose: To perform input from a boot device.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 0
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG5	devaddr	address on device medium
ARG6	memaddr	address of data buffer
ARG7	reqsize	size of data transfer requested
ARG8	maxsize	size of maximum allowable data transfer

Returns:

Number	Name	Description
RET[0]	count	actual size of data transfer

Status:

Value	Description
3	EOF encountered
2	Recoverable error
1	Inexact I/O transfer
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-5	Unrecoverable data error
-6	Illegal device address
-7	Nonexistent device
-8	Module/device not ready
-10	Invalid argument
-11	Data buffer too small
-12	Unsupported record size
-13	Protocol error
-14	Illegal device block size

Description: The argument *reqsize* specifies the amount of data that the caller would like to read. It must be a multiple of 2 Kbytes, but is otherwise unconstrained. The argument *maxsize* is the maximum amount of data that the caller is prepared to accept (i.e., it is the size of the data buffer that has been allocated. The data buffer is the area of memory between *memaddr* and *memaddr* + *maxsize*.) The value of *memaddr* must be a multiple of 64 bytes. The caller must ensure that *reqsize* is not greater than *maxsize*, or else the results are HVERSION dependent. The address on the device medium, *devaddr*, must be 2 Kbyte aligned. If the call returns a nonnegative status, the return parameter *count* is the number of bytes actually input; it must be a multiple of 2 Kbytes. The input data is in memory between *memaddr* and *memaddr* + *count*. If *maxsize* is greater than *count*, the contents of the remainder of the data buffer is HVERSION dependent. If the call returns a negative status, the value of *count* and the contents of the data buffer are HVERSION dependent. In the case of inexact I/O transfer(status return = 1), *count* must be a multiple of 2 Kbytes if there is more data to be transferred. The value of *count* need not be a multiple of 2 Kbytes if there is no more data to be transferred.

ENGINEERING NOTE

If status return = 1 and *count* is a multiple of 2 Kbytes, the caller makes another call if more data has to be transferred. If no more data has to be transferred, the caller does not make another call.

If status return = 1 and *count* is not a multiple of 2 Kbytes, then the caller does not make another call for there is no more data to be transferred. For example, the end of file has been reached.

If the boot device has class CL_SEQU, the call reads one record from the device. Each IODC implementation is free to select the maximum record size that it supports, but that maximum size must be at least 64 Kbytes. The maximum size can be greater than 64 Kbytes, but implementations are encouraged to choose it so that a record can be input within 5 seconds whenever possible. If the sequential medium is formatted with a record size greater than *maxsize*, then the call returns -11 (Data buffer too small). If the sequential medium is formatted with a record size greater than the implementation maximum record size, then the call returns -12 (Unsupported record size). As long as the record size on the sequential medium is not greater than *maxsize* or the implementation maximum, the call transfers one input record into memory. If *reqsize* was not equal to the record size, then the call returns 1 (Inexact I/O transfer). The caller must ensure that the value of *devaddr* is either 0 or equal to the sum of the previous *devaddr* and the value *count* returned by the previous call (*devaddr*=0 causes the medium to be rewound). If the boot device has class CL_SEQU and the call completed normally, ENTRY_IO may optionally return a status of 3 (EOF encountered) when it detects an EOF, instead of 0. ENTRY_IO reads a sequential medium in a strictly continuous fashion; it does not support a "skip" to a higher device address. If the call returns a negative status, then the position of the medium is HVERSION dependent.

If the boot device has class CL_RANDOM, then the IODC implementation is free to select its own maximum transfer size. This maximum value may be chosen based on device characteristics or in order to meet timeout guidelines. The maximum data transfer must be a multiple of 2 Kbytes. For each request, the call transfers the smaller of *reqsize* and its implementation maximum size. If *reqsize* is greater than the maximum size, then the call returns 1 (Inexact I/O transfer).

In order for a CL_RANDOM device to be supported as a boot device, the device must be formatted with a block size that is a factor of 2 Kbytes (i.e., 2 Kbytes is a multiple of the device block size). ENTRY_IO may optionally check that the CL_RANDOM device has been formatted with a legal block size. If IODC detects an illegal block size on the device, the call must return a status of -14 (Illegal device block size).

It is assumed that ENTRY_IO for a boot device of class CL_RANDOM supports both random and record sequential access methods.

Purpose: To perform output to a boot device.

It is not required that every implementation support this option, but it should be supported when possible and not unduly complicated.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 1
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG5	devaddr	address on device medium
ARG6	memaddr	address of data buffer
ARG7	reqsize	size of data transfer requested

Returns:

Number	Name	Description
RET[0]	count	actual size of data transfer

Status:

Value	Description
2	Recoverable error
1	Inexact I/O transfer
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-5	Unrecoverable data error
-6	Illegal device address
-7	Nonexistent device
-8	Module/device not ready
-10	Invalid argument
-12	Unsupported record size
-13	Protocol error
-14	Illegal device block size

Description: The argument *reqsize* specifies the amount of data that the caller would like to write. It must be a multiple of 2 Kbytes, but is otherwise unconstrained. The data to write is in memory at the address specified by *memaddr*. The value of *memaddr* must be a multiple of 64 bytes. The address on the device medium, *devaddr*, must be 2 Kbyte aligned. If the call returns a nonnegative status, the return parameter *count* is the number of bytes actually output; it must be a multiple of 2 Kbytes. If the call returns a negative status, the value of *count* and the data written are HVERSION dependent.

If the boot device has class CL_SEQU, *reqsize* specifies the record size and length of transfer. ENTRY_IO always writes exactly one record, with the record size equal to the length of the transfer. Each IODC implementation is free to choose the set of record sizes that it supports. If *reqsize* is equal to an unsupported record size, then the call returns status -12 (Unsupported record size). It is the responsibility of the caller to know which record sizes are supported by a device of class CL_SEQU. If the call returns a negative status, then the position of the medium is HVERSION dependent. This option never returns status 1 (Inexact I/O Transfer) for output to a device of class CL_SEQU. The caller must ensure that the value of *devaddr* is either 0 or equal to the sum of the previous *devaddr* and the value *count* returned by the previous call (*devaddr*=0 causes the medium to be rewound). ENTRY_IO writes a sequential medium in a strictly continuous fashion; it does not support a "skip" to a higher device address.

If the boot device has class CL_RANDOM, then the IODC implementation is free to select its own maximum transfer size. This maximum value may be chosen based on device characteristics or in order to meet timeout guidelines, and may be different from the maximum limit imposed on

input transfers. The maximum data transfer must be a multiple of 2 Kbytes. For each request, the call transfers the smaller of *reqsize* and its implementation maximum size. If *reqsize* was greater than the maximum size, then the call returns 1 (Inexact I/O transfer).

In order for a CL_RANDOM device to be supported as a boot device, the device must be formatted with a block size that is a factor of 2 Kbytes (i.e., 2 Kbytes is a multiple of the device block size). ENTRY_IO may optionally check that the CL_RANDOM device has been formatted with a legal block size. If IODC detects an illegal block size on the device, the call must return a status of -14 (Illegal device block size).

Purpose: To perform input from a console device.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 2
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG6	memaddr	address of data buffer
ARG7	reqsize	size of data transfer requested
ARG8	lang	future use as language specifier

Returns:

Number	Name	Description
RET[0]	count	actual size of data transfer

Status:

Value	Description
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-5	Unrecoverable data error
-6	Illegal device address
-7	Nonexistent device
-8	Module/device not ready
-10	Invalid argument
-13	Protocol error

Description: The argument *reqsize* specifies the amount of data that the caller would like to read. It must be greater than 0. The address of the data buffer allocated by the caller is specified by *memaddr*. The value of *memaddr* must be a multiple of 64 bytes and the size of the data buffer must be at least as large as *reqsize* rounded up to the next multiple of 64. The only allowed value for *lang* is 0.

This call is required to establish timeouts so that it completes within a reasonable time (typically within 5 seconds). Thus, the call may not be able to provide the full number of bytes requested by the caller in the *reqsize* argument. If there is no input available from the console device, the call must return *count* = 0. If there is input available from the console device, the call may return with *count* strictly less than *reqsize* in order to meet timeout requirements.

PROGRAMMING NOTE

Since echoing characters to the display is the responsibility of the caller, it will be most effective if the console input option returns as soon as it has any input characters available, rather than waiting until the request can be satisfied in its entirety.

The input data is in memory between *memaddr* and *memaddr* + *count*. If *count* is not a multiple of 64 bytes, the data up to the next 64-byte boundary is HVERSION dependent. If the call returns a negative status, the value of *count* and the contents of the data buffer are HVERSION dependent.

This call is defined for the classes CL_DUPLEX and CL_KEYBD.

Purpose: To perform output to a console device.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 3
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG6	memaddr	address of data buffer
ARG7	reqsize	size of data transfer requested
ARG8	lang	future use as language specifier

Returns:

Number	Name	Description
RET[0]	count	actual size of data transfer

Status:

Value	Description
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-5	Unrecoverable data error
-6	Illegal device address
-7	Nonexistent device
-8	Module/device not ready
-10	Invalid argument
-13	Protocol error

Description: The argument *reqsize* specifies the amount of data that the caller would like to write. It must be greater than 0. The data to write is in memory at the address specified by *memaddr*. The value of *memaddr* must be a multiple of 64 bytes. The only allowed value for *lang* is 0.

This call must not return a *count* value that is less than *reqsize*. If the call was unable to output all the bytes specified by the caller in the *reqsize* argument, it must not output any of them; the appropriate negative status value must be also returned. If the call returns a negative status, then the value of *count* and the data written are HVERSION dependent.

This call is defined for the classes CL_DUPLEX and CL_DISPL.

Purpose: This option closes the ENTRY_IO module-device connection that was established by ENTRY_INIT.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 4
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer

Returns: None

Status:

Value	Description
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-5	Unrecoverable data error
-6	Illegal device address
-7	Nonexistent device
-8	Module/device not ready
-10	Invalid argument
-13	Protocol error

Description: A module must implement this option if ENTRY_INIT opens a module-device connection that will need to be closed at a future point in time. An example of this type of module-device connection is one that is established solely for the purposes of boot and is not needed when boot is complete.

After this option has been called, ENTRY_INIT must be called to re-establish the connection with the device if additional module-device communication is required.

Purpose: To ask for a message when the status from the previous call to the entry point was nonzero.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 9
ARG2	spa	SPA of the module
ARG3	ID_addr	pointer to LAYER structure
ARG4	R_addr	pointer to return buffer
ARG5	data_addr	pointer to previous return buffer
ARG6	msg_addr	pointer to message buffer
ARG8	lang	future use as language specifier

Returns:

Number	Name	Description
RET[0]	msg_size	number of bytes returned in the message buffer

Status:

Value	Description
2	Recoverable error
0	OK
-3	Cannot complete call without error
-4	Unrecoverable hardware error
-10	Invalid argument

Description: The old *R_addr* pointer from the previous call is supplied as argument *data_addr* to the new option. The SVERSION-dependent part of the old *R_addr* (the last 16 words) contains all the information the option needs to return the appropriate message (the entry point does NOT need to look at the state of its module; the values in *data_addr* are sufficient). The arguments *hpa*, *spa*, and *ID_addr* must be the same as in the previous call which yielded the nonzero status. The size of the buffer pointed to be *msg_addr* must be 2 Kbytes. The only allowed value for *lang* is 0.

If *msg_size* is 0, no message was returned. The text of the returned message must not exceed one screen (24 lines).

ENTRY_SPA (index 5)

Purpose: To return the number, size, and alignment specifications of the module's SPA space(s).

Options: The option ARG1=0 is required.

Restrictions: ENTRY_SPA is allowed to access its module's HPA only to ascertain the SPA specifications. It is not allowed to cause its module to generate interrupts nor to mask the occurrence of interrupts by changing CR15 (EIEM).

Arguments:

Description	ARG0	ARG1	ARG2	ARG3
Return Info	hpa	0	tic_10ms	R

Description	ARG4	ARG5	ARG6	ARG7
Return Info	R_addr	R	R	R

The data type of *tic_10ms* is a 32-bit unsigned integer.

Returns:

Description	RET[0]	RET[1]
Return Info	spa0_spec	spa1_spec

The data type of *spa0_spec* and *spa1_spec* is a 32-bit unsigned integer.

Status:

Value	Description
0	OK The call completed normally and the entry point detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option implemented by the entry point. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG1 was invalid. OPTIONAL. The entry point may assume that its caller is perfect and need not check arguments for correctness.

Entry State: The required architected state of the target module (module specified by ARG0) upon entry to ENTRY_SPA is listed below. IO_FLEX is initialized in all cases, and all module state other than those listed is HVERSION dependent.

IO_FLEX[enb] = 1

Exit State: The required architected state of the target module (module specified by ARG0) upon exit from ENTRY_SPA is listed below. IO_FLEX is unchanged and all module state other than those listed is HVERSION dependent.

Unchanged from state at entry.

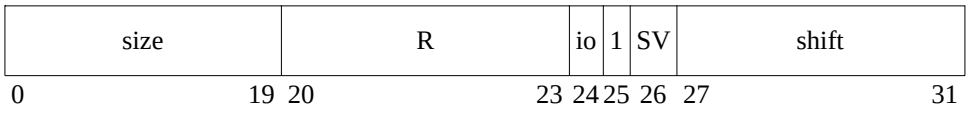
Description: An I/O module that has more than one SPA or a single SPA which is not a power of two in size is required to implement ENTRY_SPA.

Modules that provide ENTRY_SPA and which can be a boot module or a console module must ensure that ENTRY_INIT and ENTRY_IO function correctly when called by PDC.

ARG2, *tic_10ms*, is the number of clock ticks per 10 msec on the executing processor. This argument is intended for use in multiprocessor systems to establish timeouts.

Depending on whether the module has one or two SPA space(s), ENTRY_SPA will return one or two nonzero parameters. If a module has one SPA space, *spa0_spec* describes that SPA and *spa1_spec* is 0. If the module has two SPA spaces, *spa0_spec* describes the first SPA and *spa1_spec* describes the second SPA.

The format of the return parameters is as follows:



The *size* specifies the number of pages in the SPA space. The valid range for *size* is $1 \leq size \leq 61{,}312$ if *io* is 1 and $1 \leq size \leq 978{,}944$ if *io* is 0. If an invalid *size* value is returned, software may consider the module broken.

The *io* field indicates whether the SPA space is in the memory (*io*=0) or I/O (*io*=1) address space. Each SPA space must be either entirely in the memory address space or entirely in the I/O address space. At present, only memory modules are allowed to have an SPA space in the memory address space and any other request to have an SPA space in the memory address space should be considered illegal.

The *shift* field specifies the alignment requirement for the SPA as 2^{shift}. The valid range for *shift* is $12 \leq shift \leq 26$ if *io* is 1 and $12 \leq shift \leq 31$ if *io* is 0. All modules which do not have an SPA space should return 0 for the *shift* field, even if the module type does not support SPA capabilities. All other values for the *shift* field are illegal and if an invalid *shift* value is returned, software may consider the module broken.

For processor, memory, and bus converter port modules where the architecture defines the SVERSION, the value of the SV bit is HVERSION dependent.

ENGINEERING NOTE

PDCE_RESET is discouraged from calling ENTRY_SPA as it would be forced to track any future changes to the ENTRY_SPA specification. A module will function correctly even if ENTRY_SPA is not called.

ENTRY_TEST (index 8)

Purpose: To perform extensive tests on a module. In the case of an I/O module, these tests are more complete than those in ENTRY_INIT for that module and provide isolation to the FRU (Field Replaceable Unit) level. ENTRY_TEST (unlike ENTRY_INIT) may diagnose faults not involved in the basic operations of the module used by ENTRY_IO. In the case of a memory module, ENTRY_TEST provides tests that are more complete than the simple read/write test performed by PDC.

Options: The options ARG1=0 and ARG1=1 are required. The options ARG1=2 and ARG1=9 are optional.

Restrictions: None

Arguments:

Description	ARG0	ARG1	ARG2	ARG3	ARG4	ARG5
Return Info	hpa	0	spa/io_low	ID_addr/io_high	R_addr	list_addr
Execute Step	hpa	1	spa/io_low	ID_addr/io_high	R_addr	data_addr
Describe Section	hpa	2	spa/io_low	ID_addr/io_high	R_addr	data_addr
Return Message	hpa	9	spa/io_low	ID_addr/io_high	R_addr	data_addr

Description	ARG6	ARG7	ARG8	ARG9	ARG10
Return Info	R	list_type	R	scope	layer_valid
Execute Step	inbuf_addr	EIM_addr	test_id	scope	layer_valid
Describe Section	R	list_type	test_id	scope	layer_valid
Return Message	msg_addr	msg_type	lang	scope	layer_valid

Description	ARG11	ARG12	ARG13	ARG14	ARG15
Return Info	tic_10ms	R	R	R	R
Execute Step	tic_10ms	R	R	R	R
Describe Section	tic_10ms	R	R	R	R
Return Message	tic_10ms	R	R	R	R

The data type of *io_low*, *list_type*, *msg_type*, *scope*, and *tic_10ms* is a 32-bit unsigned integer. The arguments *list_addr*, *msg_addr*, and *EIM_addr* must be word aligned. The *inbuf_addr* parameter must be word aligned.

For all modules except bus converter ports, ARG2 and ARG3 represent the *spa* and *ID_addr*, respectively. For bus converter ports, ARG2 and ARG3 are *io_low* and *io_high*, respectively.

For all options, the module alone is referenced if *scope* is 0. If *scope* is 1, the module and the shared hardware are referenced. All other values are reserved. If the module is not part of a module set, both 0 and 1 refer to the module. When running ENTRY_TEST with *scope* = 1, all status values apply to the referenced module and the shared hardware.

For all options, if *layer_valid* is 1, then *ID_addr* points to a valid layer structure. If *layer_valid* is 0, then the value of *ID_addr* is defined by the caller. If an implementation of ENTRY_TEST wishes to use *ID_addr* as a pointer to a layer structure, it must first check the value of *layer_valid*. ENTRY_TEST can only test beyond the module when it has been passed a valid *ID_addr* (i.e., *layer_valid* is 1). Note that ENTRY_TEST may optionally ignore *ID_addr*, in which case it can only test the module.

The caller of ENTRY_TEST must set *layer_valid* to 0 if a pointer to a valid layer structure is not available or if testing must be restricted to the module.

For online test lists, ARG11, *tic_10ms*, is the number of clock ticks per 10 msec on the executing processor. This argument is intended for use in multiprocessor systems to establish timeouts. This argument must be 0 for offline test lists.

Returns:

Description	RET[0]	RET[1]
Return Info	dbuf_size	mbuf_size
Execute Step	r_fixed	R
Describe Section	R	R
Return Message	msg_size	R

The data type of *dbuf_size*, *r_fixed*, *msg_size*, and *mbuf_size* is a 32-bit unsigned integer.

Status:

Value	Description
2	Error detected, FRU not isolated An error was detected, but the FRU was not isolated. The caller should continue testing in order to isolate the problem. Returned only by option ARG1=1. CONDITIONAL. Must be used if the test cannot always isolate the FRU.
1	Returning for user input, next call must be sequential Returned only by option ARG1=1. CONDITIONAL. Must be used if the entry point has a test which expects user input.
0	OK The call completed normally and the entry point detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option implemented by the entry point. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-4	Invalid user input The caller must re-execute the previous step to specify new user input. Returned only by option ARG1=1. CONDITIONAL. Must be used if the entry point has a test which expects user input.
-5	Error detected, FRU isolated, module is usable The module may be usable in a degraded state; that is, the module is fully functional but performance may be lessened. Returned only by option ARG1=1. CONDITIONAL. Must be used if errors may leave the module usable.
-6	Error detected, FRU isolated, module is not usable The module has a problem and is not fully functional. Returned only by option ARG1=1. CONDITIONAL. Must be used if errors may leave the module unusable.
-7	Fatal error detected, damage is possible The error that was detected is serious enough to cause damage if testing is continued. The FRU may or may not be isolated, but further testing must be aborted. Returned only by option ARG1=1. CONDITIONAL. Must be used if testing may cause damage.
-8	Module/device not ready The module or device is not ready to be tested. A module may not be ready because it is still performing its selftest after reset or power-on. An example of device not ready is a disk drive that still is spinning up. Returned only by option ARG1 = 1. OPTIONAL. Implementations may use this status if waiting for the module or device to become ready would make the call take longer than 10 seconds.
-10	Invalid argument An argument other than ARG1 was invalid. OPTIONAL. The entry point may assume that its caller is perfect and so need not check

Value	Description
	arguments for correctness.
-11	Illegal device address The device address specified by *ID_addr* is invalid and cannot be used. One or more of the LAYER fields is out of range and could never be a valid device address. Returned only by option ARG1 = 1 and only if ARG10 = 1. OPTIONAL. Checking for illegal addresses increases supportability.
-12	Nonexistent device The device address specified by *ID_addr* is a valid device address. However, it points to either a device that is not installed or a device that does not respond. Returned only by option ARG1 = 1 and only if ARG10 = 1. CONDITIONAL. Must be used if nonexistent devices can be identified.

Entry State: The required architected state of the target module (module specified by ARG0) upon entry to ENTRY_TEST is listed below. IO_FLEX is initialized in all cases, and all module state other than those listed is HVERSION dependent.

- Option ARG1=0: IO_FLEX[enb] = 1 (for bus converter ports, the target module must have been given CMD_RESET.ST)
- Option ARG1=1:

  If ARG8 indicates first step of a section: IO_FLEX[enb] = 1

  If ARG8 not first step of a section: state unchanged from that established by the most recent previous call to ENTRY_TEST option ARG1=1 for the previous step in the test list
- Option ARG1=2: IO_FLEX[enb] = 1
- Option ARG1=9: IO_FLEX[enb] = 1

Exit State: The required architected state of the target module (module specified by ARG0) upon exit from ENTRY_TEST is listed below. IO_FLEX is unchanged and all module state other than those listed is HVERSION dependent.

- Option ARG1=0: unchanged from state at entry
- Option ARG1=1: state expected by next step in the test list; for bus converter ports, at the last step in a section, the required exit conditions must be met

  If this option is called with ARG9=1 (*scope*=1) for a module in a multi-module set, then the state of other non-processor modules in the set is HVERSION dependent. The state of processor modules in the module set must be unchanged.
- Option ARG1=2: unchanged from state at entry
- Option ARG1=9: unchanged from state at entry

Description: ENTRY_TEST may be used in an interactive (user input) or noninteractive (no user input) mode, and in a real (offline) or virtual (online) environment. Modules may provide a simplified ENTRY_TEST with only a single noninteractive offline test list and no messages, or take advantage of the complete definition to provide a flexible, interactive diagnostic, or choose any subset of the interface between these two extremes. The complexity of the ENTRY_TEST for a particular implementation is determined by the support requirements and functional limitations of the product.

PROGRAMMING NOTE

System reliability may be hampered if memory ENTRY_TEST is called online. Consider the following case:

Memory ENTRY_TEST may want to test the double-bit error detection circuitry in the memory array. This may be achieved by placing the memory module in a mode that inserts double-bit errors every time a transaction is directed to the array. Unwanted double-bit errors could be inserted if any of the following events occurs while the module is in "double-bit error insertion mode":

- The processor services an interruption (for example, an external interrupt, or a powerfail warning) and invokes a handler that issues a write.
- The processor performs an asynchronous cache flush. (Asynchronous in the sense that the flush is not related to any instruction. The processor performs the flush for performance reasons).

By calling memory ENTRY_TEST offline, the probability of inserting unwanted double-bit errors decreases. Note, however, that ENTRY_TEST cannot prevent any of the above events, even when called offline, because:

- ENTRY_TEST must not disable the PSW I-bit.
- There is no way to prevent asynchronous cache flushes.

ENTRY_TEST may want to test the error signalling mechanisms of its module. However, ENTRY_TEST is not allowed to build its own HPMC handler. Therefore, ENTRY_TEST must call PDC_ADD_VALID to detect the assertion of the error signalling mechanisms that otherwise would generate an HPMC.

ENTRY_TEST may optionally use the information in the module's IO_STATUS, IO_ERR_RESP, IO_ERR_REQ, and IO_ERR_INFO registers to increase the probability that the condition that PDC_ADD_VALID detected was the condition that ENTRY_TEST introduced.

Test Lists

A test list is a collection of tests that are appropriate for a particular testing environment, such as interactive or noninteractive. A test list is composed of one or more one-word entries called test ids. A given test id represents the same test independent of the test list that it was returned in.

A test list is terminated by a word containing zero. Thus, a test list consisting of a single word containing zero is a null list that contains no tests.

A test list is divided into a series of consecutive words that form sections. Therefore, a test list consists of zero or more test sections followed by a word containing zero.

The most significant unsigned halfword of each test id contains the section number and the least significant unsigned halfword contains the test step number. Both the section number and test step number are positive integers (neither may have a value of zero except when both are, which marks the end of the list). The section number uniquely identifies the section within an implementation of ENTRY_TEST. That is, a section number may appear on multiple test lists, but it always indicates the same series of test ids.

A section is the smallest independent entity, and contains one or more test steps. That a section is independent implies that it can be executed without advance preparation, and can be executed repetitively in a loop. A test step is not necessarily independent; it may rely on earlier steps in the section to establish the proper context. Following the execution of a test step, the module under test is in the state expected by the next test step in the section. The consecutive test ids in a section indicate successive test steps in a section. Test step numbers must be distinct within a section. Test steps must be executed in the order that they appear in the test list except under the

following circumstances:

- When a status of -4 (invalid user input) has been returned, the caller of ENTRY_TEST must re-execute the previous test step in the test list to specify user input.
- When a status of -8 (module/device not ready) has been returned, the caller of ENTRY_TEST must re-execute the current test step or abort the test.

Test steps are not required to be in increasing order within a section. Sections are not required to be in increasing order within a test list. A test step can either perform a test or prompt for user input, but it must not do both operations.

An example of a valid test list (all words in hexadecimal):

00050001, 00050002, 00050003, 001a0011, 001a0007, 00020001, 00000000

This is a test list with three sections, containing three, two, and one test steps, respectively. If any of the sections in this test list appear in other test lists, the sequence of test ids that comprise that section must be identical for all test lists in this ENTRY_TEST implementation. For example, if the following test list were part of the same ENTRY_TEST implementation as the one shown above, it would not be valid because section 0005 differs from the test list above:

00050001, 00050002, 001a0011, 001a0007, 00020001, 00000000

However, the following test list would be valid:

00070004, 00020001, 00050001, 00050002, 00050003, 00000000

The user input test lists may contain steps which request user input; noninteractive input test lists must not request user input. Offline test lists run in an environment (e.g., during boot) where interruptions are typically masked. Online test lists run in an environment (e.g., normal system operation) where interruptions are typically enabled. The standard test lists are given below.

TABLE 5-3. Standard Test Lists

list_type	Description
0	Complete Offline Test List
1	Default User Input Offline Test List
2	Default Non User Input Offline Test List
3	Complete Online Test List
4	Default User Input Online Test List
5	Default Non User Input Online Test List
6-63	Reserved
64-127	SVERSION dependent
>127	Reserved

Test lists 0 and 2 are both required. Test list 0, the complete offline test list, represents the union of all offline test lists.

Test list 5, the default non user input online test list, is optional. Test list 3, the complete online test list, is required if any online test lists are implemented. When it is implemented, test list 3 is the union of all online test lists.

To test shared hardware in a multi-module set, a caller retrieves a test list by calling ENTRY_TEST(Return Info) with ARG9=1 (*scope*=1).

The following pseudocode illustrates the calling sequence of sections and steps for a test list that does not include user input:

```
list_type ← n;                              /* select list type */
entry_test (hpa_mod_under_test, 0,....);    /* call return info */
test_id{0..15} ← section;                   /* select test section */
step_number ← first_step_in_section();      /* initialize step number */
count ← number_of_steps_in_section();       /* initialize loop counter */

while (count > 0) {
  test_id{16..31} ← step_number;            /* test step to execute */
  entry_test (hpa_mod_under_test, 1,....);  /* call execute step */

  switch (status) {

    case 0 or >= 2:                         /* continue test */
      step_number ← next_step_in_section();
      count ← count – 1;
      break;

    case –8:   /* module/device not ready, re–execute current step */
      break;   /* loop counter is not altered */
               /* for a –8, the caller may optionally abort the test */

    default:   /* the call to return message is optional */
      entry_test (hpa_mod_under_test, 9,....);  /* call return message */
      entry_io (hpa_display, 3,....);           /* output returned message to console */
      abort_test;
  }
}
```

User Input

User input to a test step is performed by having a test step that requests user input. A caller of ENTRY_TEST must not call the "Execute Step" option with a user input test list if there is no console. If any of its test lists include user input, ENTRY_TEST must implement the "Return Message" option.

ARG6 (*inbuf_addr*) is only valid for a step immediately following a step that returned a status of 1 (returning for user input). Otherwise, its value is defined by the caller.

The size of the buffer pointed to by *inbuf_addr* is a maximum of 80 characters followed by a carriage return. The caller of ENTRY_TEST must append a carriage return to the buffer if it does not already contain one.

If an "Execute Step" call returns a status of -4, the input passed in at *inbuf_addr* was invalid. If this occurs, the caller must call the "Return Message" option, display the message on the console, and re-execute the previous step.

Each user input step may optionally supply its own default values to the caller following the detection of invalid user input, instead of returning a status of -4.

If an "Execute Step" call returns a status of 1, a test step is returning for user input. If this occurs, the caller must call the "Return Message" option, display the message on the console, and retrieve a new line of input.

If status = 1 or -4 is returned from a call to "Execute Step", ENTRY_TEST must enter the message pending state (i.e., a call to the "Return Message" option will return a message). ENTRY_TEST may optionally enter the message pending state when other status values are returned from a call to "Execute Step".

The following pseudo-code illustrates the calling sequence of sections and steps for a test list that may include user input:

```
list_type ← n;                          /* select list type */
entry_test (hpa_mod_under_test, 0,....); /* call return info */
test_id{0..15} ← section;               /* select test section */
step_number ← first_step_in_section();  /* initialize step number */
count ← number_of_steps_in_section();   /* initialize loop counter */

while (count > 0) {
  test_id{16..31} ← step_number;              /* test step to execute */
  entry_test (hpa_mod_under_test, 1,....);     /* call execute step */

  switch (status) {

    case 1:                           /* returning for user input */
      entry_test (hpa_mod_under_test, 9,....);  /* call return message */
      entry_io (hpa_display, 3,....); /* output returned message to console */
      input_count ← 0;               /* initialize input loop counter */
      while (input_character != carriage_return && input_count < 80) {
        entry_io (hpa_keyboard, 2,....,1,0); /* perform single character
                                                input from console */
        entry_io (hpa_display, 3,....,1,0);  /* echo input character */
        input_count ← input_count + 1;
      }
      if (input_count = 80)           /* 80 characters input from console */
         append_carriage_return();    /* append carriage return to the buffer
                                         if it does not already contain one */
      step_number ← next_step_in_section();
      count ← count – 1;
      break;

    case 0 or >= 2:                   /* continue test */
      step_number ← next_step_in_section();
      count ← count – 1;
      break;

    case –8: /* module/device not ready, re–execute current step */
      break; /* loop counter is not altered */
             /* for a –8, the caller may optionally abort the test */

    case –4: /* invalid input, re–execute previous step */
      entry_test (hpa_mod_under_test, 9,....);   /* call return message */
      entry_io (hpa_display, 3,....); /* output returned message to console */
      step_number ← previous_step_in_section();
      count ← count + 1;              /* increment loop counter */
      break;

    default:                /* the call to return message is optional */
      entry_test (hpa_mod_under_test, 9,....);  /* call return message */
      entry_io (hpa_display, 3,....); /* output returned message to console */
      abort_test;
  }
}
```

Timeouts

ENTRY_TEST must guarantee that no call takes longer than 10 seconds to complete. (Note that memory ENTRY_TEST has stricter requirements. See Section 5.5.2.2, IODC Entry Points.) The caller of ENTRY_TEST must not put a timeout on the duration of the call.

If an error prevents ENTRY_TEST from completing a call in the required 10 seconds, ENTRY_TEST must return:

- status value -3 if ARG1 = 0, 2, or 9,
- the appropriate status value corresponding to the state of the test when the timeout occurred if ARG1 = 1.

PROGRAMMING NOTE

Note that IODC entry points are not allowed to write to CR16, the Interval Timer. Therefore, ENTRY_TEST is not allowed to determine the 10 second timeout by detecting an Interval Timer External Interrupt. An alternate method that ENTRY_TEST may use is to divide the number of counts that CR16 has taken since ENTRY_TEST was entered by MEM_10MSEC, the word in Page Zero that contains the number of clock ticks in 10 msec.

Error Handling

The "Execute Step" option may return a variety of status values indicating its progress in isolating a failure to the FRU.

A status of 2 indicates that an error was found, but the FRU has not been isolated. This value is advisory only, since the caller should still continue testing in order to isolate the problem.

Status values -5 and -6 indicate that an error has been isolated to a FRU. If status -5 is returned, the module may be usable in a degraded state; that is, the module is fully functional but performance may be lessened. ENTRY_INIT must be invoked before using ENTRY_IO on the module. Status value -6 indicates that the module is not fully functional; ENTRY_INIT and ENTRY_IO are not guaranteed to function correctly, or even detect that an error exists.

SUPPORT NOTE

If a status equal to -5 or -6 is detected following a call to Execute Step, the caller may use the following sequence of steps to identify the FRU:

1. Call ENTRY_TEST(Return Message)
2. If a status of -2 (i.e., unimplemented option) is returned, the caller should output the SVERSION-dependent locations in the return buffer (i.e., RET[16] through RET[31]) to the console.

   Callers are strongly encouraged to identify the FRU since this increases supportability. In addition, implementations of ENTRY_TEST which do not provide the Return Message option are strongly encouraged to provide FRU isolation information in RET[16] through RET[31].

Status value -7 has been included to handle situations where the error that was detected is serious enough to possibly cause damage if testing is continued. In this case the FRU may or may not be isolated, but further testing must be aborted. The status of FRU isolation can be indicated in a message returned using the ENTRY_TEST "Return Message" option.

Status value -8 indicates that the device is not ready for use. If ENTRY_TEST determines that its device is preparing for use, it should not wait for the device to become ready. Instead, it should immediately return -8.

Purpose: For I/O modules, to retrieve a test list and required buffer sizes for the entity selected by the *hpa*, *spa*, and *ID_addr* arguments. For all other modules, to retrieve a test list and required buffer sizes for the entity selected by the *hpa* argument.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 0
ARG2	spa/io_low	SPA of the module/lower bound of I/O range
ARG3	ID_addr/io_high	pointer to LAYER structure/upper bound of I/O range
ARG4	R_addr	pointer to return buffer
ARG5	list_addr	pointer to list buffer
ARG7	list_type	type of test list
ARG9	scope	scope of test
ARG10	layer_valid	qualifier for *ID_addr*
ARG11	tic_10ms	number of clock ticks per 10 msec

Returns:

Number	Name	Description
RET[0]	dbuf_size	required size for data buffer
RET[1]	mbuf_size	required size for message buffer

Status:

Value	Description
0	OK
-3	Cannot complete call without error
-10	Invalid argument

Description: The caller must allocate at least 8 Kbytes at *list_addr* for the test list. The test list selected by *list_type* is returned at *list_addr*. If the value of the *list_type* argument corresponds to a value that is unimplemented, ENTRY_TEST must return a null list (i.e. a single word containing zero).

The return parameters *dbuf_size* and *mbuf_size* specify the size (in bytes) of the data buffer and message buffer, respectively, that must be allocated by the caller in subsequent calls to ENTRY_TEST.

Purpose: To execute a single test specified by *test_id*.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 1
ARG2	spa/io_low	SPA of the module/lower bound of I/O range
ARG3	ID_addr/io_high	pointer to LAYER structure/upper bound of I/O range
ARG4	R_addr	pointer to return buffer
ARG5	data_addr	pointer to data buffer
ARG6	inbuf_addr	pointer to user input buffer
ARG7	EIM_addr	value for IO_EIM register
ARG8	test_id	identifier of test to execute
ARG9	scope	scope of test
ARG10	layer_valid	qualifier for *ID_addr*
ARG11	tic_10ms	number of clock ticks per 10 msec

Returns:

Number	Name	Description
RET[0]	r_fixed	fixed address of remote port

The return value *r_fixed* is defined only for bus converter port modules. For all other modules, this parameter is reserved.

Status:

Value	Description
2	Error detected, FRU not isolated
1	Returning for user input
0	OK
-3	Cannot complete call without error
-4	Invalid user input
-5	Error detected, FRU isolated, module may be usable
-6	Error detected, FRU isolated, module is not usable
-7	Fatal error detected, damage is possible
-8	Module/device not ready
-10	Invalid argument

Description: This option executes the test specified by *test_id*. The *test_id* argument has two parts: the 16-bit unsigned test section number in the most significant halfword and the 16-bit unsigned test step number in the least significant halfword. The *inbuf_addr* argument points to a buffer containing user input and is used when ENTRY_TEST is interacting with the console.

The *data_addr* argument points to a global data buffer of size *dbuf_size* in the memory address space. The caller of ENTRY_TEST must always pass the same buffer to all calls of ENTRY_TEST (ARG1 = 1, 2, or 9) during the execution of a given test list. The caller must not alter the contents of this buffer between calls to ENTRY_TEST during the execution of a given test list.

The caller establishes ARG7 (*EIM_addr*) to specify the target of interrupts caused by this option. The format of *EIM_addr* is the same as that of the SRS IO_EIM register in Type-A DMA modules. If the caller wishes to prevent ENTRY_TEST from generating interrupts, the caller must set ARG7=0. If ARG7 is nonzero, the caller must be prepared to accept interrupts at the group and address specified. If the caller wishes to allow ENTRY_TEST for a Type-A Direct module to generate interrupts, the caller must set *EIM_addr* to 0xFFFE0003.

If *EIM_addr* is nonzero, ENTRY_TEST for Type-A DMA modules or HP-CIO Adapters can write *EIM_addr* to IO_EIM to establish the target for interrupts.

ENTRY_TEST for Type-B DMA modules must adjust the format of *EIM_addr* to suit CCMD_LINK_I.

ENTRY_TEST for Type-A Direct modules is allowed to generate interrupts if and only if *EIM_addr* is nonzero. ENTRY_TEST for Type-A Direct modules may optionally return status -10 (Invalid parameter) if *EIM_addr* is nonzero but not equal to 0xFFFE0003.

ENTRY_TEST for memory modules must ignore the *EIM_addr* argument.

For bus converter ports, the return parameter *r_fixed* specifies the fixed address of the remote port. A value of -1 means that the fixed address of the remote port is not known, and a value from 0 to 63 means that the remote port has that value for its fixed address. All other values are illegal.

Purpose: To generate a message describing a test section in the test list specified by *list_type*.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 2
ARG2	spa/io_low	SPA of the module/lower bound of I/O range
ARG3	ID_addr/io_high	pointer to LAYER structure/upper bound of I/O range
ARG4	R_addr	pointer to return buffer
ARG5	data_addr	pointer to data buffer
ARG7	list_type	type of test list
ARG8	test_id	identifier of test to describe
ARG9	scope	scope of test
ARG10	layer_valid	qualifier for *ID_addr*
ARG11	tic_10ms	number of clock ticks per 10 msec

Returns: None

Status:

Value	Description
0	OK
-3	Cannot complete call without error
-10	Invalid argument

Description: The message is retrieved by a subsequent call to the "Return Message" option. Following a call to "Describe Section", if status is greater than or equal to 0, ENTRY_TEST must enter the message pending state. For status values less than 0, ENTRY_TEST may optionally enter the message pending state.

The least significant halfword of *test_id* must always be 0 and the effect when it is nonzero is SVERSION dependent. If *test_id* is greater than 0, a description of the section specified by the most significant halfword is generated and *list_type* is ignored. If *test_id* is 0, a short description of all sections in the test list specified by *list_type* is generated.

The *data_addr* argument points to a global data buffer of size *dbuf_size* in the memory address space. The caller of ENTRY_TEST must always pass the same buffer to all calls of ENTRY_TEST (ARG1 = 1, 2, or 9) during the execution of a given test list. The caller must not alter the contents of this buffer between calls to ENTRY_TEST during the execution of a given test list.

Purpose: To retrieve a status message, summary error report, detailed error report, test section description, or a user input prompt.

Arguments:

Number	Name	Description
ARG0	hpa	HPA of the module
ARG1	option	value is 9
ARG2	spa/io_low	SPA of the module/lower bound of I/O range
ARG3	ID_addr/io_high	pointer to LAYER structure/upper bound of I/O range
ARG4	R_addr	pointer to return buffer
ARG5	data_addr	pointer to data buffer
ARG6	msg_addr	pointer to message buffer
ARG7	msg_type	type of message
ARG8	lang	future use as language specifier
ARG9	scope	scope of test
ARG10	layer_valid	qualifier for *ID_addr*
ARG11	tic_10ms	number of clock ticks per 10 msec

Returns:

Number	Name	Description
RET[0]	msg_size	number of bytes returned in the message buffer

Status:

Value	Description
0	OK
-3	Cannot complete call without error
-10	Invalid argument

Description: This option must be called to retrieve any pending message after each call to the "Execute Step" option that returns a status of 1 or -4. The "Return Message" option may optionally be called after every call to the "Execute Step" or "Describe Section" options. If *msg_type*=0, a short (summary) message is returned, while a longer (detailed) message is returned if *msg_type*=1. The option interprets values of *msg_type* greater than 1 to be the same as *msg_type*=1 (to allow for future extensions). The message is returned at *msg_addr* and will not be longer than *mbuf_size*.

The *data_addr* argument points to a global data buffer of size *dbuf_size* in the memory address space. The caller of ENTRY_TEST must always pass the same buffer to all calls of ENTRY_TEST (ARG1 = 1, 2, or 9) during the execution of a given test list. The caller must not alter the contents of this buffer between calls to ENTRY_TEST during the execution of a given test list.

The only allowed value for *lang* is 0.

The actual size of the message is returned as *msg_size*. When there is no message pending, "Return Message" is required to return *msg_size* = 0. When there is a message pending, "Return Message" is required to return *msg_size* > 0. If *msg_size* > 0, the caller must write the message to the console. The pending message must be cleared after each call to "Return Message".

Purpose: To return the size of a Translating Bus Converter's TLB.

Options: The option ARG1=0 is required.

Restrictions: ENTRY_TBL is not allowed to access its module at all. It is not allowed to cause its module to generate interrupts nor to mask the occurrence of interrupts by changing CR15 (EIEM). It is implemented only for upper ports.

Arguments:

Description	ARG0	ARG1	ARG2	ARG3
Return Info	hpa	0	R	R

Description	ARG4	ARG5	ARG6	ARG7
Return Info	R_addr	R	R	R

Returns:

Description	RET[0]	RET[1]
Return Info	tlb_size	R

The data type of **tlb_size** is a 32-bit unsigned integer.

Status:

Value	Description
0	OK The call completed normally and the entry point detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option implemented by the entry point. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG1 was invalid. OPTIONAL. The entry point may assume that its caller is perfect and need not check arguments for correctness.

Entry State: The architected state of the target module (module specified by ARG0) upon entry to ENTRY_SPA is immaterial, and must not be changed by ENTRY_TLB.

Description: A Translating Bus Converter module must implement ENTRY_TLB for its upper port unless the number of TLB entries is equal to the number in the U2 Translating Bus Converter module.

ENTRY_TLB returns in the **tlb_size** return parameter the number of TLB entries in the module's upper port.

5.5 Module Specific IODC

The previous sections have described the aspects of IODC that are independent of module type. This section details the module-specific aspects for each module type.

The only IODC byte which is module-type dependent is the IODC_SVERSION[opt] byte. Some of the other bytes have special attributes based on module type.

SVERSION-dependent information for memory modules and bus converter ports will be given in this section.

5.5.1 Native Processor Specific IODC

5.5.1.1 IODC Data Bytes

Category A and category B processors implement only one byte of IODC, namely the IODC_TYPE byte. The IODC_TYPE byte must be 0, which identifies the module as a native processor and specifies the correct values of the *mr* and *wd* bits. Category A and category B modules also have an HVERSION and an SVERSION, but those values are accessed through the PDC_MODEL procedure rather than through the IODC.

5.5.1.2 IODC Entry Points

No IODC entry points are defined for native processor modules.

5.5.2 Memory Module Specific IODC

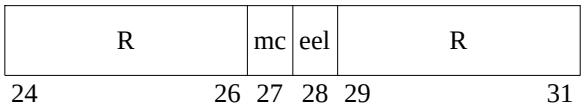
5.5.2.1 IODC Data Bytes

IODC_SVERSION

Two SVERSION model numbers are defined for memory modules (and these are the only SVERSION model numbers that will ever be defined):

IODC_SVERSION[model]	Description
8	Architected memory module
9	Processor-dependent memory module

The definition of the IODC_SVERSION[opt] byte for memory modules is as follows:



mc Defines the module category. Has values 0 and 1 for category A and B modules respectively.

eel Validates the IO_ERR_REQ, IO_ERR_RESP, and IO_ERR_INFO registers for architected errors.

5.5.2.2 IODC Entry Points

The following table describes the IODC entry points defined for Memory Modules:

Index	Mode	Name
0-2	R	Obsolete
3-6	R	Reserved for entry points
7	R	Obsolete
8	SV[1]	ENTRY_TEST
9-63	R	Reserved for architected expansions
64-127	???	Allocated for module-type dependent use
128-255	HV	Allocated for HVERSION-dependent use

Notes:

1. This entry point is HVERSION dependent for architected memory modules and reserved for processor-dependent memory modules.

The only entry point defined for memory modules is ENTRY_TEST; it applies to architected memory modules only.

The following are the objectives for ENTRY_TEST for architected memory modules:

- Verify operation of RAM error detection circuitry.
- Verify operation of RAM error correction circuitry.
- Verify operation of error reporting mechanisms in the memory module's bus interface.
- Verify operation of error logging circuitry.
- Verify operation of all I/O registers.
- Complete the test in approximately 5 seconds.

The use of ENTRY_TEST by architected memory modules is governed by these rules:

- ENTRY_TEST may be implemented or not depending on the HVERSION.
- The Default Non User Input Offline Test List (*list_type*=2) is required if ENTRY_TEST is implemented. Other test lists may be implemented or not depending on the HVERSION.
- All calls to ENTRY_TEST must be made with ARG3 (*ID_addr*) equal to 0.

Memory ENTRY_TEST must use the data buffer requested from its caller if it needs to write to the memory module

under test. (ENTRY_TEST requests this buffer when called with ARG1 = 0). The caller of memory ENTRY_TEST must allocate this data buffer in the memory module to be tested and in memory locations that have undergone a Destructive Array Test. (Note that during Boot this corresponds to memory locations lower than or equal to *fast-size*).

To guarantee that memory ENTRY_TEST runs in an environment where the only memory module is the IMM, the following relation must be valid:

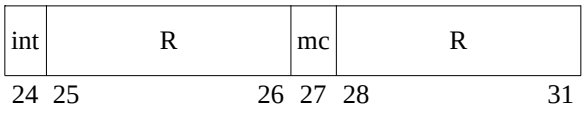
(size of the memory module for which ENTRY_TEST is written)
≥ (maximum value of MEM_FREE)
+ (ENTRY_TEST code size)
+ (8 Kbytes for ENTRY_TEST's test list)
+ (ENTRY_TEST data buffer)
+ (ENTRY_TEST message buffer)

5.5.3 Type-B DMA Specific IODC

5.5.3.1 IODC Data Bytes

IODC_SVERSION

The definition of the IODC_SVERSION[opt] byte for Type-B DMA I/O modules is:



int Module has interrupt capability if the bit is set.

mc Defines the module category. Has values 0 and 1 for category A and B modules respectively.

In addition to the IODC_SVERSION[opt] byte which explicitly encodes the existence or nonexistence of certain functionality in the module, the IODC_TYPE byte for Type-B DMA I/O modules also implicitly encodes the following functionality:

- The SRS has an IO_STATUS register.
- No effects on the responder can occur on a READ operation from any register (address).
- The module uses DMA I/O data transfers with command chaining.
- The module will only interrupt after the insertion of a completion list entry.

5.5.3.2 IODC Entry Points

The following table describes the IODC entry points defined for Type B DMA I/O Modules:

Index	Mode	Name
0-2	R	Obsolete
3	A[1]	ENTRY_INIT
4	A[1]	ENTRY_IO
5	SV[2]	ENTRY_SPA
6	SV	ENTRY_CONFIG
7	R	Obsolete
8	SV	ENTRY_TEST
9-63	R	Reserved for architected expansions
64-127	???	Allocated for module-type dependent use
128-255	SV	Allocated for SVERSION-dependent use

Notes:

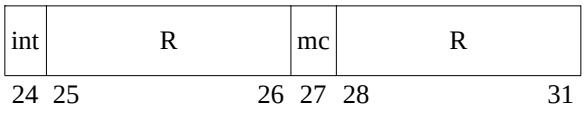
1. This entry point is architected if the module is a boot or console module; otherwise, it is reserved.
2. This entry point is architected if the module has more than one SPA or a single SPA which is not a power of two in size; otherwise, it is reserved.

5.5.4 Type-A DMA Specific IODC

5.5.4.1 IODC Data Bytes

IODC_SVERSION

The definition of the IODC_SVERSION[opt] byte for Type-A DMA I/O modules is:



int Module has interrupt capability if the bit is set.

mc Defines the module category. Has values 0 and 1 for category A and B modules respectively.

In addition to the IODC_SVERSION[opt] byte which explicitly encodes the existence or nonexistence of certain functionality in the module, the IODC_TYPE byte for Type-A DMA I/O modules also implicitly encodes the following functionality:

- The SRS has an IO_STATUS register.
- Effects on the responder can occur on a read only from any non-architected register (address).
- The module uses DMA I/O data transfers without command chaining.
- The module will only interrupt via a programmable IO_EIM register after setting the IO_II_DATA[ii] bit in the SRS.

5.5.4.2 IODC Entry Points

The following table describes the IODC entry points defined for Type A DMA I/O Modules:

Index	Mode	Name
0-2	R	Obsolete
3	A[1]	ENTRY_INIT
4	A[1]	ENTRY_IO
5	SV[2]	ENTRY_SPA
6	SV	ENTRY_CONFIG
7	R	Obsolete
8	SV	ENTRY_TEST
9-63	R	Reserved for architected expansions
64-127	???	Allocated for module-type dependent use
128-255	SV	Allocated for SVERSION-dependent use

Notes:

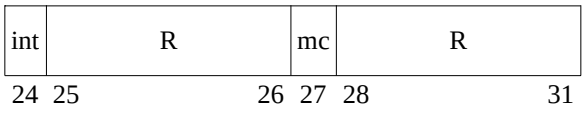
1. This entry point is architected if the module is a boot or console module; otherwise, it is reserved.
2. This entry point is architected if the module has more than one SPA or a single SPA which is not a power of two in size; otherwise, it is reserved.

5.5.5 Type-A Direct Specific IODC

5.5.5.1 IODC Data Bytes

IODC_SVERSION

The definition of the IODC_SVERSION[opt] byte for Type-A Direct I/O modules is:



int Module has interrupt capability if the bit is set.

mc Defines the module category. Has values 0 and 1 for category A and B modules respectively.

In addition to the IODC_SVERSION[opt] byte which explicitly encodes the existence or nonexistence of certain functionality in the module, the IODC_TYPE byte for Type-A Direct I/O modules also implicitly encodes the following functionality:

- The SRS may not have an IO_STATUS register; generic software will assume that it does not.
- Effects on the responder can occur on a read only from any non-architected register (address).
- The module uses direct I/O data transfers.
- The module will only interrupt via asserting the PATH_INT signal (producing a global broadcast interrupt to EIR{3}) after setting the IO_II_DATA[ii] bit in the SRS.

5.5.5.2 IODC Entry Points

The following table describes the IODC entry points defined for Type A Direct I/O Modules:

Index	Mode	Name
0-2	R	Obsolete
3	A[1]	ENTRY_INIT
4	A[1]	ENTRY_IO
5	SV[2]	ENTRY_SPA
6	SV	ENTRY_CONFIG
7	R	Obsolete
8	SV	ENTRY_TEST
9-63	R	Reserved for architected expansions
64-127	???	Allocated for module-type dependent use
128-255	SV	Allocated for SVERSION-dependent use

Notes:

1. This entry point is architected if the module is a boot or console module; otherwise, it is reserved.
2. This entry point is architected if the module has more than one SPA or a single SPA which is not a power of two in size; otherwise, it is reserved.

5.5.6 Bus Converter Port Specific IODC

An upper bus converter port must provide at least bytes 0 through 15 of IODC data and at least the ENTRY_TEST entry point. A lower bus converter port must provide at least bytes 0 through 7 of IODC data.

5.5.6.1 IODC Data Bytes

IODC_HVERSION

IODC_HVERSION bits {0..4} give the bus ID of the port's bus, according to the **Bus Identifier** table in Section B.2.2. of the PA-RISC 1.1 I/O Architectural Reference Specification.

PROGRAMMING NOTE

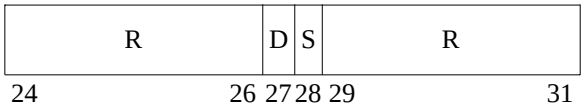
Bits 0..4 of IODC_HVERSION[model] of a bus converter port can be used to identify the bus connected to that port. This is useful for determining the optimal system configuration (based on bus performance) and executing diagnostics.

IODC_SVERSION

The assigned value for IODC_SVERSION[rev] is 0x0.

The assigned value for IODC_SVERSION[model] is 0x0000C.

The definition of the IODC_SVERSION[opt] byte is:



D Indicates that the Bus converter is D-coherent. If the D-bit is 1, then purges and flushes do not need to be done after I/O operations. If the bit is set to 0, then purges and flushes of the memory range DMAed to must be done after an I/O operation.

S Indicates that the Bus converter does Synchronzation on I/O. If the S-bit is 1, then no SYNCDMA instructions must be executed in conjunction with a DMA operation. If the S-bit is 0, then each I/O operation requires SYNCDMA instructions.

5.5.6.2 IODC Entry Points

The following table describes the IODC entry points defined for Bus Converter Modules:

Index	Mode	Name
0-2	R	Obsolete
3-6	R	Reserved for entry points
7	R	Reserved
8	A[1]	ENTRY_TEST
9-63	R	Reserved for architected expansions
64-127	???	Allocated for module-type dependent use
128-255	HV	Allocated for HVERSION-dependent use

Notes:

1. This entry point is architected for ports that are not hardwired and hardwired upper ports. It is reserved for lower ports.

ENTRY_TEST is the only IODC entry point allowed for bus converter ports.

ENTRY_TEST is required for all ports that are not hardwired. Additionally, it is required for hardwired upper ports. It is not allowed on a hardwired lower port. In the case of independent ports, ENTRY_TEST will exist in each port, but only the one accessed through the upper port will be executed. However, ENTRY_TEST must be able to test both ports of the bus converter.

When hard booting, PDC must call ENTRY_TEST for all bus converters in the boot and console paths and may optionally call ENTRY_TEST for all other bus converters; PDC must not call ENTRY_TEST at any other time. The successful execution of ENTRY_TEST for a bus converter decreases the likelihood that the bus converter will impede the data transfer between the boot/console module and memory on the central bus.

SUPPORT NOTE

To minimize undetected failures, the operating system is encouraged to call ENTRY_TEST during system configuration after a hard boot for all bus converters not tested by the PDC boot code. However, if ENTRY_TEST is called during a soft boot, OS_PFR, or OS_PFW_REMOTE, the OS should be aware that it may need to allocate some portion of memory to load the ENTRY_TEST code into, keeping in mind that there is no architectural limit on the size of ENTRY_TEST (offline test lists).

The only system configuration that bus converter ENTRY_TEST may assume, in addition to the bus converter itself, is the monarch processor and a memory module on the central bus. In particular, ENTRY_TEST must not assume that a module on the lower bus could serve as a slave to transactions coming from the upper port or as a master for transactions going to the upper port.

The *EIM_addr* and *data_addr* arguments allow ENTRY_TEST to test the bus converter's master circuitry on the upper port. ENTRY_TEST may obtain the monarch processor's HPA either by calling PDC_HPA, or by deriving it from the *EIM_addr* argument (if the *EIM_addr* argument is not in the BPA space). The *EIM_addr* argument must be either the monarch processor's IO_EIR address, or the IO_EIR address in the global broadcast address space. The *data_addr* argument points to a buffer in a memory module's SPA. ENTRY_TEST may use these arguments to generate the slave address of a transaction directed to the central bus. (Another way for ENTRY_TEST to test the bus converter port's master circuitry is to issue transactions to itself.)

The *io_low* and *io_high* arguments provide a range of addresses that can be used to test transactions going through the BC. Both *io_low* and *io_high* must be 64 Kbyte aligned. The range must be such that ((*io_high* & 0xFFFC0000) - *io_low*) >= 256 Kbytes, in order to guarantee that ENTRY_TEST always finds an address range to use as the HPA of the remote bus.

SUPPORT NOTE

ENTRY_TEST should test every possible circuit including the following:

- Modes (OFF, INCLUDE, EXCLUDE, and PEEK)
- Master and slave functionality
- Error signalling circuitry
- Error logging circuitry
- The link connecting both ports
- Transaction queues
- All implemented registers in the I/O and broadcast address spaces

---

In addition to the normal IODC calling conventions, the state of both ports of the bus converter between the execution of sections in a test list is the following:

- The IO_FLEX register on the upper port must not be changed.
- On the lower port, IO_FLEX[flex] on the lower port must be set to any 256 Kbyte-aligned address between *io_low* and *io_high* and IO_FLEX[enb] must be zero.
- IO_IO_LOW, IO_IO_HIGH, and IO_CONTROL registers may be modified by ENTRY_TEST, provided the changes in these registers do not cause multiple slaves to acknowledge a directed transaction. However, in order to prevent unwanted transactions from modules on the lower bus, a broadcast flex disable must be issued before enabling the remote port.
- On exit, the values of IO_CONTROL, IO_IO_LOW, and IO_IO_HIGH are HVERSION dependent if ENTRY_TEST returns with a negative status. Otherwise, the following restrictions apply:

— IO_CONTROL[mode] on the upper port must be set to INCLUDE; IO_CONTROL[mode] on the lower port must be set to EXCLUDE.

— IO_IO_LOW and IO_IO_HIGH must be set to the values passed in as ARG2 and ARG3.

- If ENTRY_TEST returns with a negative status value, IO_STATUS and the extended error logging registers must reflect the state of the bus converter ports at the time ENTRY_TEST failed. Otherwise, the IO_STATUS register must indicate the port is in ST_READY, IO_STATUS[lp] must be 0 on the upper port and 1 on the lower port, IO_STATUS[pl] must be 0 on both ports, and IO_STATUS[pw,pf] must reflect the state of the remote power.
- The remaining registers may be modified and contain HVERSION-dependent data upon exit of each call to ENTRY_TEST.

Further, ENTRY_TEST must not cause any modules on the upper and the lower bus to generate bus traffic through the bus converter during its execution. Also, ENTRY_TEST must not affect the state of any module "above" the bus converter other than the contents of memory from *data_addr* to *data_addr+dbuf_size*-1 and sending an interrupt to *EIM_addr*.

For an offline test list, the state of all other modules "below" the BC being tested is a function of the IODC_HVERSION and IODC_REV bytes of the upper BC port.

SUPPORT NOTE

It is recommended that affecting the state of other modules "below" the bus converter should be avoided if possible.

In addition to the requirements listed above, the caller of "Return Info" option of ENTRY_TEST needs to establish the following state:

- Modules on the upper bus have been reset (with CMD_RESET.ST), have their HPA initialized, and have their bus mastership enabled.

PROGRAMMING NOTE

A recommended sequence of steps for ENTRY_TEST is as follows:

1. Test the upper bus converter port.
2. Issue a CMD_CLEAR to the upper bus converter port and put it in PEEK mode.
3. Initialize the HPAs and disable mastership on the lower bus.
4. Identify the bus converter lower port (this step is easy if its fixed address is hardwired).
5. Put the upper port in INCLUDE mode.
6. Test the lower bus converter port.

5.5.7 Translating Bus Converter (IOA) Specific IODC

An upper translating bus converter port must provide at least bytes 0 through 15 of IODC data and at least the ENTRY_TEST entry point. It may optionally also provide the ENTRY_TLB entry point. A lower bus converter port must provide at least bytes 0 through 7 of IODC data.

5.5.7.1 IODC Data Bytes

IODC_HVERSION

IODC_HVERSION bits {0..4} give the bus ID of the port's bus, according to the **Bus Identifier** table in Section B.2.2. of the PA-RISC 1.1 I/O Architectural Reference Specification.

PROGRAMMING NOTE

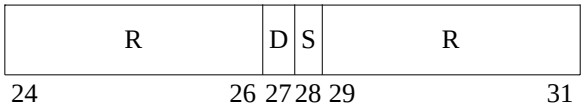
Bits 0..4 of IODC_HVERSION[model] of a bus converter port can be used to identify the bus connected to that port. This is useful for determining the optimal system configuration (based on bus performance) and executing diagnostics.

IODC_SVERSION

The assigned value for IODC_SVERSION[rev] is 0x0.

The assigned value for IODC_SVERSION[model] is 0x0000B.

The definition of the IODC_SVERSION[opt] byte is:



D Indicates that the Bus converter is D-coherent. If the D-bit is 1, then purges and flushes do not need to be done after I/O operations. If the bit is set to 0, then purges and flushes of the memory range DMAed to must be done after an I/O operation.

S Indicates that the Bus converter does Synchronzation on I/O. If the S-bit is 1, then no SYNCDMA instructions must be executed in conjunction with a DMA operation. If the S-bit is 0, then each I/O operation requires SYNCDMA instructions.

5.5.7.2 IODC Entry Points

The following table describes the IODC entry points defined for Translating Bus Converter Modules (IOAs):

Index	Mode	Name
0-2	R	Obsolete
3-6	R	Reserved for entry points
7	R	Reserved
8	A[1]	ENTRY_TEST
9	O[2]	ENTRY_TLB
10-63	R	Reserved for architected expansions
64-127	???	Allocated for module-type dependent use
128-255	HV	Allocated for HVERSION-dependent use

Notes:

1. This entry point is architected for ports that are not hardwired and hardwired upper ports. It is reserved for lower ports.
2. This entry point is optional for upper ports. If the number of TLB entries is the same as the U2 IOA module, it is not required, otherwise, it is. It is reserved for lower ports.

ENTRY_TEST and ENTRY_TLB are the only IODC entry points allowed for Translating Bus Converter (IOA) modules.

ENTRY_TEST is required for all ports that are not hardwired. Additionally, it is required for hardwired upper ports. It is not allowed on a hardwired lower port. In the case of independent ports, ENTRY_TEST will exist in each port, but only the one accessed through the upper port will be executed. However, ENTRY_TEST must be able to test both ports of the bus converter.

When hard booting, PDC must call ENTRY_TEST for all bus converters in the boot and console paths and may optionally call ENTRY_TEST for all other bus converters; PDC must not call ENTRY_TEST at any other time. The successful execution of ENTRY_TEST for a bus converter decreases the likelihood that the bus converter will impede the data transfer between the boot/console module and memory on the central bus.

SUPPORT NOTE

To minimize undetected failures, the operating system is encouraged to call ENTRY_TEST during system configuration after a hard boot for all bus converters not tested by the PDC boot code. However, if ENTRY_TEST is called during a soft boot, OS_PFR, or OS_PFW_REMOTE, the OS should be aware that it may need to allocate some portion of memory to load the ENTRY_TEST code into, keeping in mind that there is no architectural limit on the size of ENTRY_TEST (offline test lists).

The only system configuration that bus converter ENTRY_TEST may assume, in addition to the bus converter itself, is the monarch processor and a memory module on the central bus. In particular, ENTRY_TEST must not assume that a module on the lower bus could serve as a slave to transactions coming from the upper port or as a master for transactions going to the upper port.

The *EIM_addr* and *data_addr* arguments allow ENTRY_TEST to test the bus converter's master circuitry on the upper port. ENTRY_TEST may obtain the monarch processor's HPA either by calling PDC_HPA, or by deriving it from the *EIM_addr* argument (if the *EIM_addr* argument is not in the BPA space). The *EIM_addr* argument must be either the monarch processor's IO_EIR address, or the IO_EIR address in the global broadcast address space. The *data_addr* argument points to a buffer in a memory module's SPA. ENTRY_TEST may use these arguments to generate the slave address of a transaction directed to the central bus. (Another way for ENTRY_TEST to test the bus converter port's master circuitry is to issue transactions to itself.)

The *io_low* and *io_high* arguments provide a range of addresses that can be used to test transactions going through the BC. Both *io_low* and *io_high* must be 64 Kbyte aligned. The range must be such that ((*io_high* & X'FFFC0000) - *io_low*) >= 256 Kbytes, in order to guarantee that ENTRY_TEST always finds an address range to use as the HPA of the remote bus.

SUPPORT NOTE

ENTRY_TEST should test every possible circuit including the following:

- Modes (OFF, INCLUDE, EXCLUDE, and PEEK)
- Master and slave functionality
- Error signalling circuitry
- Error logging circuitry
- The link connecting both ports
- Transaction queues
- TLBs and caches
- All implemented registers in the I/O and broadcast address spaces

---

In addition to the normal IODC calling conventions, the state of both ports of the bus converter between the execution of sections in a test list is the following:

- The IO_FLEX register on the upper port must not be changed.
- On the lower port, IO_FLEX[flex] on the lower port must be set to any 256 Kbyte-aligned address between *io_low* and *io_high* and IO_FLEX[enb] must be zero.

- IO_IO_LOW, IO_IO_HIGH, and IO_CONTROL registers may be modified by ENTRY_TEST, provided the changes in these registers do not cause multiple slaves to acknowledge a directed transaction. However, in order to prevent unwanted transactions from modules on the lower bus, a broadcast flex disable must be issued before enabling the remote port.
- On exit, the values of IO_CONTROL, IO_IO_LOW, and IO_IO_HIGH are HVERSION dependent if ENTRY_TEST returns with a negative status. Otherwise, the following restrictions apply:
  - — IO_CONTROL[mode] on the upper port must be set to INCLUDE; IO_CONTROL[mode] on the lower port must be set to EXCLUDE.
  - — IO_IO_LOW and IO_IO_HIGH must be set to the values passed in as ARG2 and ARG3.
- If ENTRY_TEST returns with a negative status value, IO_STATUS and the extended error logging registers must reflect the state of the bus converter ports at the time ENTRY_TEST failed. Otherwise, the IO_STATUS register must indicate the port is in ST_READY, IO_STATUS[lp] must be 0 on the upper port and 1 on the lower port, IO_STATUS[pl] must be 0 on both ports, and IO_STATUS[pw,pf] must reflect the state of the remote power.
- The remaining registers may be modified and contain HVERSION-dependent data upon exit of each call to ENTRY_TEST.

Further, ENTRY_TEST must not cause any modules on the upper and the lower bus to generate bus traffic through the bus converter during its execution. Also, ENTRY_TEST must not affect the state of any module "above" the bus converter other than the contents of memory from *data_addr* to *data_addr+dbuf_size*-1 and sending an interrupt to *EIM_addr*.

For an offline test list, the state of all other modules "below" the BC being tested is a function of the IODC_HVERSION and IODC_REV bytes of the upper BC port.

SUPPORT NOTE

It is recommended that affecting the state of other modules "below" the bus converter should be avoided if possible.

In addition to the requirements listed above, the caller of "Return Info" option of ENTRY_TEST needs to establish the following state:

- Modules on the upper bus have been reset (with CMD_RESET.ST), have their HPA initialized, and have their bus mastership enabled.

PROGRAMMING NOTE

A recommended sequence of steps for ENTRY_TEST is as follows:

1. Test the upper bus converter port.
2. Issue a CMD_CLEAR to the upper bus converter port and put it in PEEK mode.
3. Initialize the HPAs and disable mastership on the lower bus.
4. Identify the bus converter lower port (this step is easy if its fixed address is hardwired).
5. Put the upper port in INCLUDE mode.
6. Test the lower bus converter port.

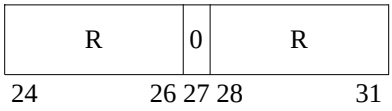
ENTRY_TLB is used to inform the operating system the number of TLB entries contained in the upper port of the translating bus converter.

5.5.8 Console Pseudo-Module Specific IODC

5.5.8.1 IODC Data Bytes

A Console psuedo-module must implement the first 16 bytes of IODC, as described below:

- IODC_HVERSION must be implemented as already defined for other module types.
- The shift field in the IODC_SPA byte must always be zero, since a Console pseudo-module can never have an SPA.
- IODC_TYPE must be implemented as already defined for other module types. The IODC_TYPE[type] field must equal TP_CONSOLE, value 9.
- IODC_SVERSION must be implemented as already defined for other module types. The IODC_SVERSION[model] field must equal 0x0001C, since all current Console pseudo-modules have the same software interface, and hence the same SVERSION.
- The definition of the IODC_SVERSION[opt] byte for console modules is as follows:



- IODC_REV, IODC_DEP, IODC_CHECK, and IODC_LENGTH must be implemented as already defined for other module types.

5.5.8.2 IODC Entry Points

The following table describes the IODC entry points defined for Console Pseudo-modules:

Index	Mode	Name
0-2	R	Obsolete
3	A	ENTRY_INIT
4	A	ENTRY_IO
5	R	Reserved for entry points.
6	HV	ENTRY_CONFIG
7	R	Obsolete
8	HV	ENTRY_TEST
9-127	R	Reserved
128-255	HV	Allocated for HVERSION-dependent use

A Console pseudo-module must implement its IODC entry points as follows:

- A Console pseudo-module must implement ENTRY_INIT as currently defined, except that it must ignore the *hpa* and *spa* arguments to each option.
- A Console pseudo-module must implement ENTRY_IO as currently defined, but with the following restrictions:
  - A Console pseudo-module must implement options ARG1=2 (Console input) and ARG1=3 (Console output).
  - A Console pseudo-module may optionally implement option ARG1=9 (Return message).
  - A Console pseudo-module must not implement options ARG1=0 (Boot input) or ARG1=1 (Boot output).
  - A Console pseudo-module must ignore the *hpa* and *spa* arguments to each option.
- If a Console pseudo-module implements ENTRY_TEST, it must implement it as currently defined, except that it must ignore the *hpa*, *spa*, and *EIM_addr* arguments to each option.
- The ARG1=0 option of PDC_IODC must be used to fetch IODC from a Console pseudo-module by passing in its pseudo-HPA.

5.5.9 Independently-developed Module Specific IODC

The IODC_SVERSION[model] value 0x00FFF is allocated for use by Type A-Direct, Type A-DMA, and Type B-DMA I/O modules developed independent of HP. This SVERSION differs from all other SVERSIONs in that it does not correspond to a unique software interface.

Modules with this IODC_SVERSION[model] value must have an IODC_SVERSION[rev] value of 0. For these modules, the contents and meaning of the IODC_HVERSION[model] field are dependent on the particular module implementation, and do not have their normal architectural definition.

All other IODC fields, for example, IODC_TYPE, IODC_SPA, IODC_SVERSION[opt], must be implemented as already defined in the I/O Architecture.

ENGINEERING NOTE

Unique SVERSIONs are available to both internal HP developers and outside vendors. An outside vendor who contacts HP will be allocated a unique SVERSION.

The IODC_SVERSION[model] value of 0x00FFF is intended exclusively for use by outside vendors who wish to develop modules independent of HP, and who do not wish to contact HP for a unique SVERSION.

LIST OF FIGURES

LIST OF TABLES

This page intentionally left blank