PA-RISC 1.1 I/O Firmware Architecture Reference Specification

Version 1.0

Printed in U.S.A. August 22, 2001

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.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

1.1 Objectives

The following list summarizes the objectives used in developing the PA-RISC I/O Architecture.

- **Software Portability**

  The I/O Architecture must not impact the portability of software. Programs should be transportable from one PA-RISC system to another of similar configuration without requiring modification.

- **High-Level Language I/O Drivers**

  The I/O Architecture must allow operating system I/O drivers to be written in high-level languages, with only minor modifications to the compilers.

- **Scalability**

  The I/O Architecture must permit implementations ranging from very low-cost, relatively low-performance ones, to very capable, relatively expensive ones. To be competitive, it must do this without unduly constraining any particular implementation. The module architectures defined for the PA-RISC I/O Architecture should allow a module to be used on the full range of PA-RISC systems, from the lowest-cost computers to the highest-performance scientific and commercial systems.

- **Multiprocessing**

  The I/O Architecture must allow for support of multiple PA-RISC processor configurations. Systems with processors on separate busses as well as systems with all processors on a single bus must be supported.

- **Multiple Bus Specifications**

  A high-performance PA-RISC system can be configured with high-, medium-, and low-performance I/O modules. A single electrical/mechanical specification cannot provide high-end system performance and meet low-end system cost objectives. The I/O Architecture supports the definition of multiple bus specifications for low-, medium-, and high-performance modules.

- **Transparent Access**

  The PA-RISC I/O Architecture uses bus converters to support scalable configurations of PA-RISC busses without the use of architecturally visible channels or I/O processors.

- **Non-PA-RISC I/O Compatibility**

  The I/O Architecture must allow the construction of adapters which enable non-PA-RISC I/O systems to be connected to a PA-RISC system.

- **Direct User Control of I/O**

  The operating system should be able to give users direct control over devices when appropriate, without compromising system security. Direct control is important because it enhances efficiency in some real-time applications, and because it simplifies the task of debugging new I/O drivers.

- **Fault-Tolerance**

  The I/O Architecture should make it possible to recover from device and path failures. Recovery should be transparent to user programs.

- **Automatic Configuration**

  Software should be able to determine the system configuration automatically, regardless of the relative position of cards in the system. This should not require manual adjustments, such as DIP switches on the cards or daisy chaining between cards.

  The system should be readily reconfigurable at power-up, when reset, and after device or path failures.

  The addition of a new bootable device to a system should not require changes to the system initialization software.

Any system should be bootable from any device which contains enough non-volatile memory to hold the necessary code.

Any system should be bootable over a network.

The boot sequence should be controllable by system managers.

The system configuration should be changeable without requiring that the operating system be terminated or that the system be reinitialized.

A standard interface should exist between the boot code and the operating system.

- **Flexible Interrupt Scheme**

  Interrupt priorities should be software controllable.

  The response time from the occurrence of an interrupt to entry of the appropriate I/O driver should be significantly shorter than the time necessary to perform a context switch.

- **Evolving HP I/O Standards**

  The architecture should support future HP attachment standards.

- **Forward Progress Guarantee**

  Deadlock and starvation on all PA-RISC busses are avoided by bus arbitration and slave service protocols.

- **Reduced Software Development Costs**

  By standardizing the software interface to modules with identical or similar functionality, the costs (in time, dollars, and schedule) of new I/O driver software can be reduced.

- **High Performance**

  The architectural interface to modules should support high-performance applications. Performance is improved by two approaches:

  a. Performance data is used in the design phase of new modules.

  b. The module architecture provides stability for tuning of the software (I/O driver) and hardware (module design).

- **Compatibility**

  Separate module architectures should not be required in order to support different versions of operating system software.

- **Memory Error Logging**

  The memory architecture should support the logging of recoverable (if ECC circuitry is present) and unrecoverable errors.

- **Memory Cost**

  On limited configurations, an architectural option should be provided to minimize the cost of memory.

- **Open Systems Support**

  The architecture should provide the flexibility for the development of modules outside and independent of HP.

1.2 System Organization

1.2.1 System Components

A **module** is an entity which is configured into the system address space and adheres to the PA-RISC I/O Architecture. As the I/O system is memory-mapped, a module can be interrogated and controlled by software via the standard load and store instructions rather than with special I/O instructions.

Section 1.4, Modules provides an introductory definition of the architecturally defined module types.

A **card** is a physical entity containing the components which are necessary for module operation. Cards are not visible to software during normal system operation. Cards may become visible when special diagnostics are run or when a module must be replaced due to component failure. A card has one and only one slot address. A module is always implemented on a single card. During normal operation, the modules on a card must be architecturally independent.

Multiple cards (slot addresses) cannot exist on a single printed circuit board unless at least one native processor module is present. To identify the modules which exist on a board with a native processor, software must call the "Return modules" option (ARG1=1) of PDC_HPA (see Chapter 4, PDC Procedures, for more details).

PROGRAMMING NOTE

These two requirements provide software with enough information to determine which modules are physically associated, so that the appropriate modules can be notified when card or board-level operations are performed (for example, card reset or board replacement).

ENGINEERING NOTE

A card may consist of one or more printed circuit boards.

A **module set** is a group of two or more modules completely contained on a single card. A card containing a module set is a **multi-module card**.

Each bus specification defines the number of cards which can be supported and the number of modules per card. For example, a bus might support up to 16 cards with up to 4 modules

The electrical and mechanical properties of busses are defined in separate bus specification documents. Three examples of bus specifications are: SMB, MID_BUS, and HP-PB.

A **foreign bus** is used to connect cards designed for some other I/O system to a PA-RISC system. A foreign bus does not meet the requirements of the connect protocol.

A **device** is the object to which input and/or output operations are done. Devices are connected to but are not part of an I/O module. They are accessed directly or indirectly through the address space of that I/O module, and they may optionally be independently powered. Each I/O module may have any number of devices connected to it. For the purposes of architectural discussion, the device includes all the entities (such as cables, controllers, link adapters, etc.) between the module and the physical device. Examples of devices are terminals, disks, tape drives, and network connections.

A **system resource** provides the same unique service to each native processor in a system. Examples of system resources are the Time-Of-Day clock and Stable Storage. Each processor has its independent version of a **processor resource**. Accesses to privileged processor/system resources are restricted to architected load and store instructions, HVERSION-dependent mechanisms, and the DIAG instruction, each of which must be privileged.

1.2.2 System Topology

A **system** consists of a collection of modules which share a single consistent view of the system address space. That is, a given physical address accesses the same memory or register location in the same module no matter which module is performing the access.

1.2.3 Configuration Constraints

There are several configuration constraints for PA-RISC systems and they are listed here:

- All processor and memory modules must reside on the central bus.
- Each PA-RISC system must have at least one processor module, memory module, and I/O module.
- The maximum number of processor modules that may be present in a PA-RISC system is 62.
- The maximum number of memory modules that may be present in a PA-RISC system is 62.
- Each PA-RISC system may have up to three levels of bus converters.
- Only the central bus may be a category B bus.
- A category A bus must have only category A modules. A category B bus can have category A and category B modules.
- All bus specifications must provide support for all module types, with exactly one exception: support for Type-A Direct modules is optional.
- All bus specifications must provide support for all SVERSIONs, with exactly one exception: support for architected memory modules on category B busses is optional.

Individual products are free to establish their own configuration constraints above and beyond those established by the architecture and their bus specification. These constraints must be published in the appropriate product documentation and may be enforced by the number and types of slots provided in product boxes. The set of allowable product configuration constraints is listed below:

- The maximum number of modules per bus may be less than 64.
- The product may limit the maximum number of modules of any given type, SVERSION, or HVERSION (as long as there is still at least one each of processor, memory, and I/O module).
- The product may restrict certain module HVERSIONs to specific slots.

1.3 System Address Space

The address space of a PA-RISC system uses 32-bit physical addresses. The **system address space** is partitioned into three major sections, the memory address space, the PDC address space, and the I/O address space, which in turn are divided into smaller subsections as shown in the following figure. The system address space is logically subdivided into **pages**, each 4 Kbytes in size and aligned on a 4-Kbyte boundary. See Section 1.6, Physical Page Attributes for a description of the various kinds of pages.

PROGRAMMING NOTE

Software should be aware that some PA-RISC systems have a 2-Kbyte page size and a 2-Kbyte alignment restriction. The *IT_conf* return parameter in the PDC procedure PDC_CACHE provides the page size of the system.

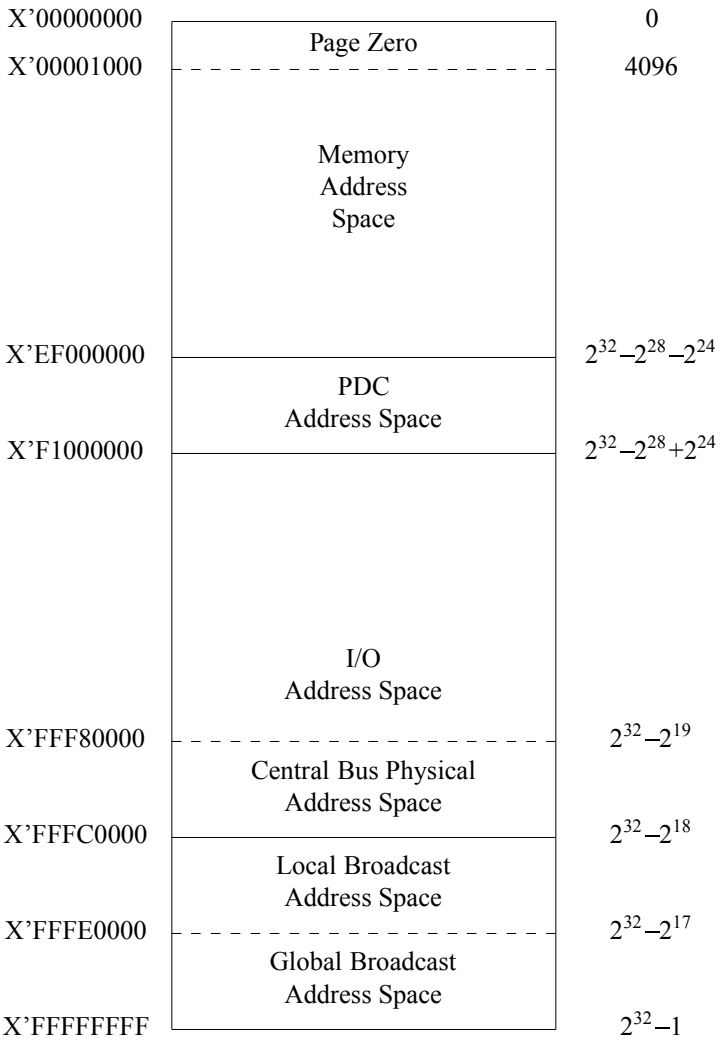

Figure 1-1. System Address Space Layout

1.3.1 Memory Address Space

The **memory address space** extends from address X'00000000 through X'EEFFFFFF. Instructions and data in the memory address space can be cached.

Addresses X'00000000 through X'EEFFFFFF are available only for assignment to memory modules.

ARCHITECTURAL NOTE

Whether other module types may use the memory address space is the subject of

---

The first 4 Kbytes of memory is restricted for use by boot and initial program load (IPL) software. This area is called **Page Zero**.

1.3.2 PDC Address Space

The PDC address space extends from address X'EF000000 through X'F0FFFFFF. It is restricted to use by Processor-Dependent Code (PDC).

Depending on their HVERSION, native processors may optionally fetch instructions from the PDC address space. The effect of such an instruction fetch depends on the HVERSION of the processor.

Depending on their HVERSION, native processors may optionally access the PDC address space via load instructions, store instructions, load and clear instructions, and flush instructions. The effect of such instructions depends on the HVERSION of the processor.

All pages in the PDC address space must be privileged.

1.3.3 I/O Address Space

The **I/O address space** extends from address X'F1000000 through X'FFFBFFFF and is allocated for the configuration of modules only. The I/O address space is never cached.

The 256 Kbyte (2^{18}-byte) space from X'FFF80000 through X'FFFBFFFF is called the **Central Bus Physical Address Space**. This space is restricted for use by modules on the central bus.

1.3.4 Broadcast Address Space

The 256 Kbyte (2^{18}-byte) space from X'FFFC0000 through X'FFFFFFFF is called the **Broadcast Physical Address Space**. The broadcast space is split into two portions, the **local broadcast address space** and the **global broadcast address space**.

A write to the broadcast space is received by all modules on the same bus as the bus requestor of the transaction, and forwarded from the lower port to the upper port by bus converters. It is also received by every module on every bus to which the transaction has been forwarded.

Three functions require the use of the broadcast address space:

1. Broadcast interrupts can be sent to all processors (on the local bus or in the entire system).
2. A global broadcast reset command can be used to reboot the system by resetting all the processors in the system.
3. A local broadcast can be used to assign a portion of the system address space to each of the modules on the local bus.

1.4 Modules

Each module in a PA-RISC system has a type associated with it. This type is stored in the module's IODC ROM in the IODC_TYPE byte. (See Chapter 5, IODC, for details concerning the format and contents of IODC.)

1.4.1 Module Types

A **Native Processor** module executes the Precision instruction set and can execute the standard operating system software (i.e., HP-UX and MPE-iX). For a complete description of the internal processor architecture and instruction set, refer to the *Precision Architecture and Instruction Set Reference Manual*.

A **Memory** module provides directly accessible storage for instructions and data.

1.4.2 Module Categories

The I/O Architecture defines two module categories to describe the module participation in the software-independent coherence algorithms.

Module categories apply to modules as bus requestors, modules which are bus responders only do not have a defined module category.

The two module categories are the following:

- Category A modules do not participate in any software-independent coherence algorithm. As bus requestors category A modules may only issue transaction modes supported by category A busses. As bus responders, category A modules are required to alias all transaction variants to the corresponding default transaction variant.
- Category B modules participate in the software-independent coherence algorithms in a coherence prodocol defined on its category B bus. As bus requestors, category B modules are allowed to issue the set of transaction modes supported by their coherence protocol. As bus responders, category B modules are required to implement the transaction mode functionality specified for their cohrence protocol.

Part of the SVERSION number identifies the category of a module (see the description of IODC_SVERSION in Section 5.1, IODC Data Bytes, and Appendix B, Version and Identification Numbers).

1.5 Software Types

The I/O Architecture recognizes three classes of software: generic software, driver software, and diagnostic software.

1.5.1 Generic Software

Generic software is software which is expected to communicate with a wide variety of module types. Generic software uses IODC data and entry points to determine the specific characteristics of a module. Generic software need only be changed when some drastic change in the I/O Architecture occurs, such as the addition of a new module type or the addition of fault tolerance. Examples of generic software functions are interrupt handling, powerfail preparation and recovery, error handling, and system initialization and configuration.

Generic software may access architected registers of a module and may also write to registers in the BPA space. Generic software is required to specify the value 0 in SVERSION-dependent or HVERSION-dependent fields when writing to an architected register. Architected registers in the broadcast address space do not contain SVERSION-dependent or HVERSION-dependent fields.

Generic software should use only the module capabilities described by the module-type documentation appropriate for the module's type and SVERSION[opt] (recognizing, however, that sometimes hardware incompatibilities will be worked around in software instead of updating the hardware, thereby requiring the generic software to look at other SVERSION and/or HVERSION-dependent information).

1.5.2 Driver Software

Driver software should use only the module capabilities described by the module-type and SVERSION documentation appropriate for the module's type and SVERSION (recognizing, however, that sometimes hardware incompatibilities will be worked around in software instead of updating the hardware, thereby requiring the driver software to look at HVERSION-dependent information).

1.5.3 Diagnostic Software

Diagnostic software can use any module capability described by the documentation appropriate for the module's type, SVERSION, and HVERSION.

1.6 Physical Page Attributes

Each page in the system's address space can be characterized by a set of attributes which hold for each addressable unit in the page. It should be stressed that these attributes are on a per page basis. For example, either all registers in a page of an I/O module exist, or they are all nonexistent. Likewise, if an I/O module has a privileged register in a page, the remaining registers in that page are also privileged.

Mapped Pages

A page is considered **mapped** to a module if it lies within a module's broadcast, hard, or soft physical address space. Note that the pages in a bus converter port's I/O and memory range spaces are not considered to be mapped to that bus converter port, but are mapped to the modules which lie on the other side of that bus converter.

Existent and Nonexistent Pages

All pages which are not mapped to any module are defined to be **nonexistent pages**. Attempting to access a location in a nonexistent page will be signalled as an error. Similarly, all pages which are mapped to some module are defined to be **existent**.

Implemented and Unimplemented Pages

Both implemented and unimplemented pages are existent pages; however, an **implemented page** has some hardware storage associated with its addresses while an **unimplemented page** does not. Attempting to access a location in an unimplemented page must be signalled as an error. Accesses to implemented pages must not generate an error, unless there is a hardware fault. For modules which signal an error on accesses to unimplemented pages, all unimplemented pages must be privileged (see the description of privileged pages below).

A module with both implemented and unimplemented pages is called **partially populated**.

Privileged and Unprivileged Pages

The architecture requires that modules map sensitive addresses (i.e., those that can affect system security) to **privileged pages**. Modules must be designed so that no access to an **unprivileged page** can result in a **security violation**. In particular, this implies that commands written to I/O registers or attempts to access SVERSION/HVERSION-dependent I/O registers in unprivileged pages must not cause an access to a privileged page nor cause a bus error. In this way, all module actions which can produce security violations are restricted to privileged pages.

It is the responsibility of the operating system to set the access rights appropriate for each kind of page and to ensure that privileged pages are not mapped to the virtual address space of an unprivileged user. Permission to access the system address space is given on a page basis and is a function of virtual address translation (privilege levels and protection ids). Permission to access a page can only be granted to an unprivileged user if it is guaranteed that reads and writes of the page cannot compromise system security (e.g., corrupt another user or system process). Therefore, the following pages must always be privileged:

- A module's hard physical address space.
- All pages in the broadcast physical address space.
- Any page which, when accessed, might cause the module to become a bus master, except to send an interrupt which is either broadcast to EIR {3} or sent to an address specified in a privileged page.
- Any page which, when accessed, might cause a register in another privileged page to change, except for the the following registers:
  - — the IO_STATUS register in the SRS, which may be updated to report errors.
  - — the IO_ERR_RESP, IO_ERR_REQ, and IO_ERR_INFO registers in the ARS, which may be updated to report errors.
- Any page which, when accessed, might cause a bus error, except those due to hardware faults.

Cacheable and Uncacheable Pages

Cacheable pages must be accessed only with burst (multiple of 16 bytes) operations, and data from these pages may be placed in caches within processor modules. Uncacheable pages must be accessed only with non-burst (1,2, or 4 bytes) operations, and data from these pages must not be placed in caches within processor modules.

All pages in the I/O address space are uncacheable.

Normally, all pages in the memory address space are cacheable. However some processors may support uncacheable pages in the memory address space on a per page basis. Memory modules and busses designed to be used in systems with these processors must support non-burst accesses to these pages.

The cacheability of pages in the PDC address space is HVERSION dependent.