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.

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 chan nels 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.

A **pseudo-module** is an entity which implements a limited subset of module functionality. It differs from a module in that it does not respond to standard load and store instructions, but only to processor dependent code procedures (PDC) and I/O dependent code entry points (IODC). (see Chapter 5, IODC, for more details).

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

A **card** is a physical entity containing the components which are necessary for module or pseudo-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 or pseudo-module is always implemented on a single card. During normal operation, the modules or pseudo-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 or pseudo-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 or pseudo-modules are physically associated, so that the appropriate modules or pseudo-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 or pseudo-modules completely contained on a single card. A card containing a module set is a **multi-module card**.

A **native bus** is used for the interconnection of pseudo-modules and modules. A native bus meets the requirements of the connect protocol. Modules communicate with each other using transactions on native busses. Psuedo-modules must ignore all of these transactions.

There are at most 64 modules and pseudo-modules on each native bus. **Critical busses** are those busses that the system needs in order to run. How the operating system determines if a given bus is critical is operating system dependent and it could be user specified. If a critical bus is down during boot, the system cannot proceed until that bus is powered up. By contrast, **non-critical busses** are not absolutely necessary to system operation.

Each bus specification defines the number of cards which can be supported and the number of modules or pseudo-modules per card. For example, a bus might support up to 16 cards with up to 4 modules or pseudo-modules per card. Multiple native busses can be connected together by bus converters in order to increase the number of modules in the system or to allow high-speed modules on a high-performance bus to communicate transparently to low-speed modules on a low-performance bus.

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 foreign bus may be connected to a native bus by a bus adapter. The bus adapter must provide for all connect protocal requirements not met by the foreign bus.

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 and pseudo-modules connected together by native busses, with optional connections to foreign busses. All modules in the system 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.

A PA-RISC system can be either a uniprocessor system, or a multiprocessor system with all processors on one bus. The processor-memory bus in a PA-RISC system is called the **central bus**. The central bus forms the root of a tree of busses and must be a native bus.

An example topology is shown in the following figure.

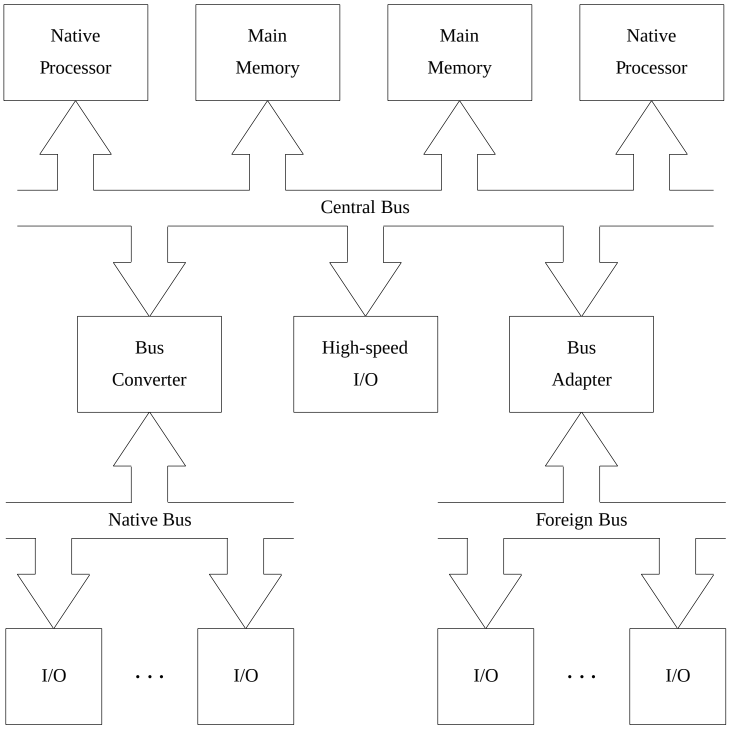

Figure 1-1. System Topology (Example)

In this example, a multiprocessor system is shown which has high speed I/O on the central bus, and which connects to both a native bus and a foreign bus for additional I/O.

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.
- A category A bus must have only category A modules. A category B bus can have category A and category B modules. See Section 1.4.2, Module Categories for definitions.
- Only the central bus may be a category B bus.
- 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 and pseudo_modules per bus may be less than 64.
- The product may limit the maximum number of modules or pseudo-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 64-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.

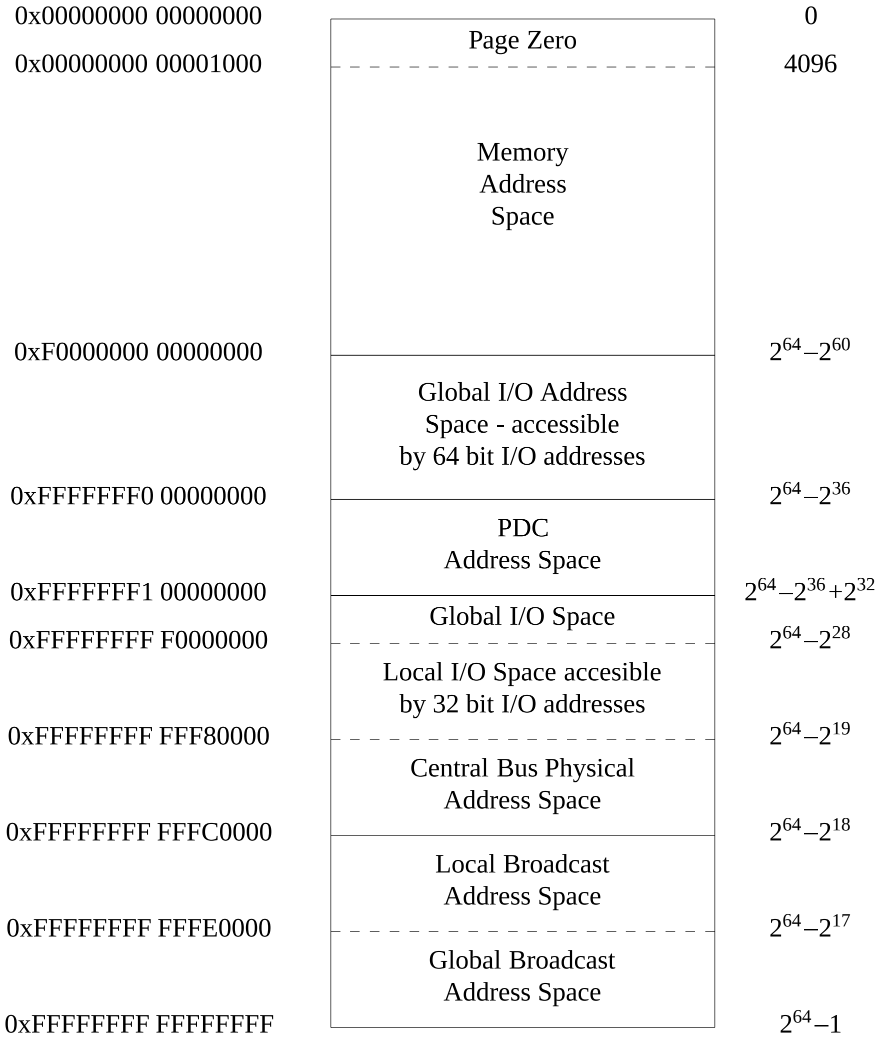

Figure 1-2. System Address Space Layout

1.3.1 Memory Address Space

The **memory address space** extends from address 0x00000000 00000000 through 0xEEFFFFFF FFFFFFFF. Instructions and data in the memory address space can be cached.

Addresses 0x00000000 00000000 through 0xEFFFFFFF FFFFFFFF are available only for assignment to memory modules.

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 0xFFFFFFF0 00000000 through 0xFFFFFFF0 FFFFFFFF. It is restricted to use by Processor-Dependent Code (PDC), PDC semaphores, non-volatile memory managed by PDC, and PDC scratch RAM.

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** may be divided into the *Global I/O address space,* accessible only by 64-bit I/O addresses, and the *Local I/O address space* which is accssible by 32-bit (F extended) I/O addresses. *The Local I/O address* space is provided to allow the system to run 32-bit PA-RISC code. The *Global I/O address space* extends from address 0xF0000000 00000000 through 0xFFFFFFEF FFFFFFFF and from 0xFFFFFFF1 FFFFFFFF through 0xFFFFFFFF EFFFFFFF. All of the *Global I/O address space* is **Direct I/O address sapce**.

The *Local I/O address space* extends from address 0xFFFFFFFF F0000000 through 0xFFFFFFFF FFFFFFFF and may be divided into the **Direct I/O address space** and the **Broadcast I/O address space.** The I/O address space is never cached.

The direct I/O address space extends from address 0xFFFFFFFF F0000000 through 0xFFFFFFFF FFFBFFFF and is allocated for the configuration of modules only.

The 256 Kbyte (2^{18}-byte) space from 0xFFFFFFFF FFF80000 through 0xFFFFFFFF 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 0xFFFFFFFF FFFC0000 through 0xFFFFFFFF 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 requester of the transaction, and, in addition, a write to the global broadcast space is 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 and Pseudo-modules

Each module and pseudo-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.)

The architecturally defined module and pseudo-module types are:

- Native Processor (TP_NPROC)
- Memory (TP_MEMORY)
- Type-B DMA I/O (TP_B_DMA)
- Type-A DMA I/O (TP_A_DMA)
- Type-A Direct I/O (TP_A_DIRECT)
- Foreign I/O (TP_FIO)
- Bus Converter Port (TP_BCPORT)
- Coherent Bus Converter Port (TP_IOA)
- Bus Adapter Port (TP_BA)
- Bus Bridge Port (TP_BRIDGE)
- Fabric connector (TP_FABRIC)
- Console Pseudo-module (TP_CONSOLE)

1.5 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 requesters, 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 requesters 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 protocol defined on its category B bus. As bus requesters, 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 coherence 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)

1.5.1 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.

Type-A Direct I/O modules, **Type-A DMA I/O** modules, and **Type-B DMA I/O** modules are used to perform I/O fuctions on native busses. They differ in the manner in which I/O completion and the generation of interrupts are handled. These differences are the responsibility of the I/O drivers. A **Type-B DMA I/O** module is used to control high-speed I/O devices such as disks, tapes, and networks. Type-B DMA I/O modules can fetch commands from memory, transfer data to or from memory, and return status in completion lists semi-autonomously. Direct memory access (DMA) is used to execute a chain of commands (**command chaining**) and move multiple blocks of data (**data chaining**) without incurring continual processor overhead. Type-B DMA I/O modules access I/O registers in order to generate interrupts and/or reset the system. Type-B DMA I/O modules can be programmed to send interrupts to any bit of any processor's External Interrupt Register (EIR). No effects on a Type-B DMA module due to reads of any registers are allowed in order to fully support error detection, isolation, and recovery. Direct user access can be given only to those pages which do not contain any registers controlling configuration, interrupts, or DMA.

A **Type-A DMA I/O** module implements a simpler form of DMA than does Type-B. Command chaining and completion lists are not supported, but data chaining is. Type-A DMA I/O modules access I/O registers in order to generate interrupts and/or reset the system. Type-A DMA I/O modules can be programmed to send interrupts to any bit of any processor's External Interrupt Register (EIR). Effects on an Type-A DMA module due to reads of non-architected registers are allowed in order to reduce cost with a corresponding loss in the ability to detect, isolate, and recover from errors. Direct user access can be given only to those pages which do not contain any registers controlling configuration, interrupts, or DMA.

A **Type-A Direct I/O** module is used in low-cost, low-performance applications. This module type is intended to allow inexpensive connections of industry-standard I/O chips to Precision systems. Type-A Direct I/O modules cannot master any bus operations. Type-A Direct I/O modules send only broadcast interrupts to a fixed bit of every processor's External Interrupt Register (by asserting the PATH_INT signal). Effects on an Type A Direct module due to reads of non-architected registers are allowed in order to reduce cost with a corresponding loss in support for error detection, isolation, and recovery. Direct user access can be given to all pages except those which contain registers controlling configuration or interrupts.

A **Foreign I/O** module performs I/O functions on a foreign bus connected to a PA-RISC system. A foreign I/O module may only be a Category A module.

A pair of **bus converter port** modules are used to connect two native busses. Each pair of bus converter ports is linked together by a **bus converter link**. The combination of two bus converter ports and the link which joins them is called a **bus converter**. A bus converter can be used, for example, to increase the total number of modules in a system, to connect a high-speed processor/memory bus to a lower-speed I/O bus, or to allow two busses to be physically separated. During normal system operation, bus converters are completely transparent to software. That is, software running on any processor can control the modules on any bus.

In a Bus convertor for which the upper port is a Catogory B module, the upper port is a **Coherent Bus Converter Port**. Lower bus converter ports may only be Category A modules.

A **Bus Adapter** is a connection between a native bus and a foreign bus. The module on the native bus is similar to an upper bus converter port, but only a subset of the architecurally required modes and registers for a bus converter port may be implemented. In addition, there may be implementation dependent modes and registers implemented which are not specified in the architecture. For example, the foreign bus specification may describe a method for locating and initializing I/O modules which does not use architected means.

A **Bus Bridge** is a connection between a native bus and a foreign bus. The module on the native bus is similar to an upper bus converter port. Bus Bridges are required to implement OFF and INCLUDE/EXCLUDE mode, and the full error reporting protocol on the native bus, but only a subset of the architecurally required modes and registers may be implemented for the foreign bus. In addition, there may be implementation dependent modes and registers implemented which are not specified in the architecture. Generally, a Bridge is more Architecturally compliant than an Adapter.

A **Fabric Connector** is a connection between a a native bus or set of native busses and a point-top-point link or set of point-to-point links, or a connection between a set of point-to-point links. Each bus connectgor is a module and resides on that bus, and each link connector is a module, and resides on that link. The module on each bus or link is called a port, and must implement only the architected command and status registers required of all generic modules. In addition it may implement registers specific to the module which route specific transactions to other ports (modules) on that connector. Each bus or port must implement an identical set of command and status registers. In addition, if a **Fabric Connector** implements native bus ports, it may also implement a bus adapter link to a foreign I/O bus, and a link to a memory controller. The bus adapter link must obey the architectural rules for bus adapters, and the memory controller link must obey the architectural rules for memory modules.

A **Console** pseudo-module is used only during system boot, initialization, and failure. The only architected interface to a console module is through processor dependent code (PDC) and I/O-dependent code (IODC); it does not appear in the system's address space.

1.6 Software Types

The PA-RISC Architecture recognizes three classes of software: generic software, SVERSION dependent software, and HVERSION dependent software.

1.6.1 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).

Listed below are some architectural features accessed by generic software.

- IODC

  Identifies the module type, extended address requirements, whether the module interrupts, etc.

- IO_II_DATA/IO_II_CLEAR

  Used by the first-level interrupt handler when polling Type-A DMA and Type-A Direct I/O modules.

- Completion List Entries

  Processed by the first-level interrupt handler for Type-B DMA I/O modules.

- IO_FLEX

  Used to assign module addresses (at power-on and during powerfail recovery). Used to prevent non-processor modules from arbitrating during powerfail preparation.

- IO_COMMAND

  The Reset command is sent to each module during boot and powerfail recovery.

- IO_STATUS

  Power status bits are checked during boot and powerfail recovery. Error bits are checked by OS_HPMC.

1.6.2 SVERSION Dependent Software

SVERSION dependent software should use only the module capabilities described by the module-type and SVERSION documentation appropriate for the module's type and SVERSION. I/O Drivers are the typical example of SVERSION dependent software. This restriction enables the driver for a particular I/O module to be portable from one hardware platform to another. However, 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. This use should be minimized where possible.

1.6.3 HVERSION Dependent Software

HVERSION Dependent software can use any module capability described by the documentation appropriate for the module's type, SVERSION, and HVERSION. The two common types of HVERSION dependent software are diagnostic software and firmware (PDC and IODC).

1.7 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 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.
  - — the IO_II_DATA register in the SRS, which may be updated to report interrupts.
- 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.

1.8 Bus Signals Used to Support Powerfail Recovery

Each native bus specfication is required to define three bus signals to support powerfail recovery. The following table lists the architected bus signals:

Signal	Description
BUS_POW_VALID	Indicates the validity of primary and secondary power
BUS_POW_WARN	Indicates that primary power is about to be lost
BUS_SEC_VALID	Indicates the validity of secondary power

1.8.1 BUS_POW_VALID

BUS_POW_VALID indicates that primary and secondary power are valid (see Section 1.8.1, Power-On).

BUS_POW_VALID is always required.

1.8.2 BUS_POW_WARN

BUS_POW_WARN indicates that primary power will be lost (see Section 1.8.2.2, BUS_POW_WARN Functionality).

In systems that support powerfail recovery, BUS_POW_WARN must be implemented by every bus. In systems that do not support powerfail recovery, the implementation of the signal by any bus is optional.

1.8.3 BUS_SEC_VALID

BUS_SEC_VALID indicates that secondary power is valid (see Section 1.8.3, Secondary Power).

In systems that support powerfail recovery, BUS_SEC_VALID must be implemented by every bus that contains secondary-powered modules. The implementation of BUS_SEC_VALID by a bus is optional if the bus does not contain secondary-powered modules or if the system does not support powerfail recovery.

1.9 Power Requirements

Precision systems are designed to operate correctly even if their source of primary power fails temporarily. The I/O Architecture provides mechanisms so that Precision software can be warned of an impending power failure and be informed of the power status of system components after power is restored. The recovery strategies depend on memory modules preserving their contents throughout the power failure through the use of secondary power. Software can use information stored in memory to reconstruct its state after power is restored.

The processor(s) state is preserved through a powerfail preparation/recovery sequence. Pending I/O operations are generally lost, and must be restarted from the previous checkpoint.

A complex system may consist of several busses, each of which has a separate power supply. In a distributed power environment, any power supply may lose and recover power independently of the other power supplies. The following list briefly describes some of the situations.

- **Normal Power Failure**

  In the normal situation, the failure of primary power is preceded by a warning which allows enough time for the system to save its state in memory. Primary power is lost, but secondary power remains intact until primary power is restored.

- **Extended Power Failure**

  If the duration of the primary power failure is too long, secondary power will eventually fail, destroying the contents of memory.

- **Repeated Power Failures**

  During repeated power failures, primary power is lost and regained many times in quick succession. The interval between failures is not sufficient to allow powerfail recovery to be completed before the next failure occurs. This mode of power failure may occur when the external power is just at the threshold needed for normal operation.

- **Sudden Power Failure**

  If the power system suffers a major hardware fault, it is possible to lose primary power with no warning whatsoever. This is a catastrophic failure, since processor state cannot be preserved. The same effect will be achieved if power is preceded by a warning, but the interval is not long enough for the system to save its state.

- **Peripheral Power Failure**

  While the system is running, an I/O device may experience a power failure.

- **Local Power Failure**

  A local power failure is a power failure which affects the central bus.

- **Remote Power Failure**

  A remote power failure is a power failure which is isolated to one or more remote busses, but does not affect the central bus.

- **Total Power Failure**

  If all the components of a system are located in close proximity, the most likely case will be a total power failure: all busses lose primary power at about the same time.

- **Partial Power Failure**

  If the components of a system are physically remote (as in the case of bus converter ports joined by a fiber optic link), a partial power failure becomes more likely. A partial power failure is one in which some busses lose power while other busses retain power. The busses that lose power may regain it quickly, after a long interval, or not at all.

All bus specifications must define the following three signals: BUS_POW_VALID, BUS_SEC_VALID, and BUS_POW_WARN. BUS_POW_VALID indicates that both primary power and secondary power are valid.

BUS_SEC_VALID indicates that only secondary bus power is valid. The BUS_POW_WARN signal is asserted prior to the deassertion of BUS_POW_VALID to warn of the impending deassertion of BUS_POW_VALID.

1.9.1 Power-On

Power-on is defined as the transition of the BUS_POW_VALID signal from the deasserted to the asserted state. A **soft power-on** occurs if the BUS_SEC_VALID signal remained asserted while the BUS_POW_VALID signal was deasserted. A **hard power-on** occurs if the BUS_SEC_VALID signal was deasserted while the BUS_POW_VALID signal was deasserted.

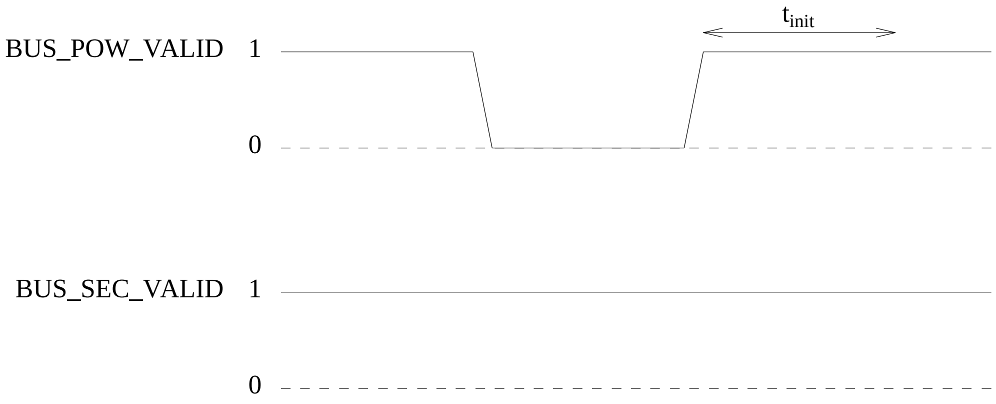

Figure 1-3. Soft Power-On

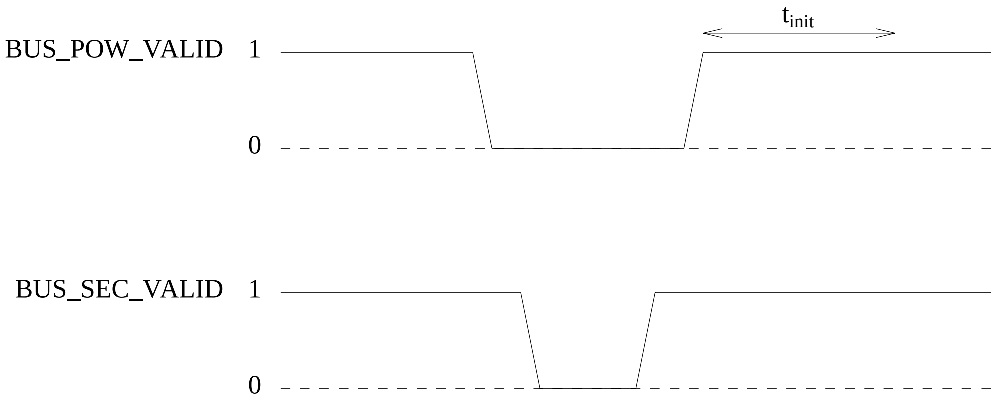

Figure 1-4. Hard Power-On

The **bus init time** is defined as the period of time after the assertion of BUS_POW_VALID during which the initialization of the bus is performed and all modules on the bus are initialized to initiate and respond to system operations. (In the two previous figures, the bus init time is indicated by t_{init}.) Each bus specification must define its bus init time. During the init time implementations may use bus operations solely for the purpose of bus initialization and initialization of the bus operation related functionality of modules.

ENGINEERING NOTE

In addition to the init time specification, bus implementations need to provide all specifications required by module designers to ensure that modules on that bus can be properly initialized to perform system operations. These specifications are dependent upon the particular bus implementation. Deconfigured or faulty modules which have not been deconfigured may be unable to initiate system operations even after the bus init time has elapsed.

Section 2.6, Module State, defines the hard and soft power-on states of a module after the bus init time has elapsed.

1.9.2 Powerfail

1.9.2.1 BUS_POW_VALID Functionality

The BUS_POW_VALID signal is asserted when both primary and secondary power on the bus are valid, otherwise, BUS_POW_VALID is deasserted.

ENGINEERING NOTE

The deassertion of BUS_POW_VALID is not required to be synchronous with the bus clock. However, the design and verification of the slave handshake circuit is easier if the deassertion of BUS_POW_VALID is synchronous with the bus clock.

1.9.2.2 BUS_POW_WARN Functionality

For systems that support powerfail recovery, the BUS_POW_WARN signal is required. With the exception of modules performing a system reset, the BUS_POW_WARN signal must be asserted to indicate the impending deassertion of the BUS_POW_VALID signal. The assertion of BUS_POW_WARN must always be followed by the deassertion of BUS_POW_VALID. The minimum time between the assertion of BUS_POW_WARN and the deassertion of BUS_POW_VALID is called the **powerfail budget**. Each bus specification must define a powerfail budget.

All modules which expect to be interchangeable across all implementations of a given bus must be able to perform all necessary powerfail processing, if any, during the powerfail budget defined by their bus specification. However, modules not designed to be interchangeable (in other words, system-specific modules) are allowed to require a longer powerfail budget. For these modules, each system is free to increase the powerfail budget defined by the bus specification.

SUPPORT NOTE

Modules which do not meet the powerfail budget defined by the bus specification are restricted to systems which guarantee a sufficient powerfail budget for these modules. This, of course, restricts the generality of the module. Therefore, modules are encouraged to complete their powerfail processing within the powerfail budget defined by the bus specification.

In order to guarantee successful powerfail recovery, each system must compute several powerfail budgets: the **local powerfail budget**, which indicates an impending local power failure, and a **remote powerfail budget** for each remote bus, which indicates an impending remote power failure. A **local power failure** is a power failure which affects the central bus. A **remote power failure** is a power failure which is isolated to one or more remote busses, but does not affect the central bus.

The local powerfail budget consists of two components:

- The time from the assertion of BUS_POW_WARN to the detection of the powerfail warning interrupt by all processors.

  This time is dependent on many factors including the maximum operation time, the maximum instruction execution time, and the maximum time that the PSW I-bit is 0.

- The execution time of OS_PFW_LOCAL.

  The execution time of OS_PFW_LOCAL is dependent on many factors including the cache flush time, the maximum operation time, the execution time of PDC_CHASSIS and PDC_POW_FAIL, and the actual code executed which is operating system specific.

The remote powerfail budget consists of one component:

- The time from the assertion of BUS_POW_WARN to the detection of a global broadcast interrupt to EIR{1} by all processors.

> This time is dependent on factors such as the time for a bus converter to recognize the assertion of BUS_POW_WARN on its lower port, and generate an interrupt on its upper bus, the maximum operation time, the maximum instruction execution time, and the maximum time that the PSW I-bit is 0.

In a multiprocessor system, the two components of the local powerfail budget on one processor may overlap those on another processor. For example, one processor may detect the powerfail interrupt quickly and start executing OS_PFW_LOCAL to flush its cache, while another processor is still executing code with the PSW I-bit cleared.

Also, the maximum operation time during powerfail preparation may differ from that during normal system operation, for example, because no bus traffic will come up from remote busses. (I/O ACD, Section 2.4.2, System Operation Completion Time, specifies the method to compute the maximum operation time for a particular system configuration.)

1.9.3 Secondary Power

The BUS_SEC_VALID signal is asserted only when the secondary power on the bus is valid. If secondary power is not valid, both BUS_SEC_VALID and BUS_POW_VALID must be deasserted.

TABLE OF CONTENTS

LIST OF FIGURES