Delivering PCI in HP B-Class and C-Class Workstations: A Case Study in the Challenges of Interfacing with Industry Standards

Ric L. Lewis
Erin A. Handgen
Nicholas J. Ingegneri
Glen T. Robinson

In the highly competitive workstation market, customers demand a wide range of cost-effective, high-performance I/O solutions. An industry-standard I/O subsystem allows HP workstations to support the latest I/O technology.

Industry-standard I/O buses like the Peripheral Component Interconnect (PCI) allow systems to provide a wide variety of cost-effective I/O functionality. The desire to include more industry-standard interfaces in computer systems continues to increase. This article points out some of the specific methodologies used to implement and verify the PCI interface in HP workstations and describes some of the challenges associated with interfacing with industry-standard I/O buses.

PCI for Workstations

One of the greatest challenges in designing a workstation system is determining the best way to differentiate the design from competing products. This decision determines where the design team will focus their efforts and have the greatest opportunity to innovate. In the computer workstation industry, the focus is typically on processor performance coupled with high-bandwidth, low-latency memory connections to feed powerful graphics devices. The performance of nongraphics I/O devices in workstations is increasing in importance, but the availability of cost-effective solutions is still the chief concern in designing an I/O subsystem. Rather than providing a select few exotic high-performance I/O solutions, it is better to make sure that there is a wide range of cost-effective solutions to provide the I/O functionality that each customer requires. Since I/O performance is not a primary means of differentiation and since maximum flexibility with appropriate price and performance is desired, using an

industry-standard I/O bus that operates with high-volume cards from multiple vendors is a good choice.

The PCI bus is a recently established standard that has achieved wide acceptance in the PC industry. Most new general-purpose I/O cards intended for use in PCs and workstations are now being designed for PCI. The PCI bus was developed by the PCI Special Interest Group (PCI SIG), which was founded by Intel and now consists of many computer vendors. PCI is designed to meet today's I/O performance needs and is scalable to meet future needs. Having PCI in workstation systems allows the use of competitively priced cards already available for use in the high-volume PC business. It also allows workstations to keep pace with new I/O functionality as it becomes available, since new devices are typically designed for the industry-standard bus first and only later (if at all) ported to other standards. For these reasons, the PCI bus has been implemented in the HP B-class and C-class workstations.

PCI Integration Effort

Integrating PCI into our workstation products required a great deal of work by both the hardware and software teams. The hardware effort included designing a bus interface ASIC (application-specific integrated circuit) to connect to the PCI bus and then performing functional and electrical testing to make sure that the implementation would work properly. The software effort included writing firmware to initialize and control the bus interface ASIC and PCI cards and writing device drivers to allow the HP-UX* operating system to make use of the PCI cards.

The goals of the effort to bring PCI to HP workstation products were to:

- Provide our systems with fully compatible PCI to allow the support of a wide variety of I/O cards and functionality
- Achieve an acceptable performance in a cost-effective manner for cards plugged into the PCI bus
- Create a solution that does not cause performance degradation in the CPU-memory-graphics path or in any of the other I/O devices on other buses in the system
- Ship the first PCI-enabled workstations: the Hewlett-Packard B132, B160, C160, and C180 systems.

Challenges

Implementing an industry-standard I/O bus might seem to be a straightforward endeavor. The PCI interface has a thorough specification, developed and influenced by many experts in the field of I/O bus architectures. There is momentum in the industry to make sure the standard succeeds. This momentum includes card vendors working to design I/O cards, system vendors working through the design issues of the specification, and test and measurement firms developing technologies to test the design once it exists. Many of these elements did not yet exist and were challenges for earlier Hewlett-Packard proprietary I/O interface projects.

Although there were many elements in the team's favor that did not exist in the past, there were still some significant tasks in integrating this industry-standard bus. These tasks included:

- Designing the architecture for the bus interface ASIC, which provides a high-performance interface between the internal proprietary workstation buses and PCI
- Verifying that the bus interface ASIC does what it is intended to do, both in compliance with PCI and in performance goals defined by the team
- Providing the necessary system support, primarily in the form of firmware and system software to allow cards plugged into the slots on the bus interface ASIC to work with the HP-UX operating system.

With these design tasks identified, there still remained some formidable challenges for the bus interface ASIC design and verification and the software development teams. These challenges included ambiguities in the PCI specification, difficulties in determining migration plans, differences in the way PCI cards can operate within the PCI specification, and the unavailability of PCI cards with the necessary HP-UX drivers.

Architecture

The Bus Interface ASIC

The role of the bus interface ASIC is to bridge the HP proprietary I/O bus, called the general system connect (GSC) bus, to the PCI bus in the HP B-class and C-class workstations. **Figures 1** and **2** show the B-class and C-class workstation system block diagrams with the bus interface ASIC bridging the GSC bus to the PCI bus. The Runway bus shown in **Figure 2** is a high-speed processor-to-memory bus.[1]

The bus interface ASIC maps portions of the GSC bus address space onto the PCI bus address space and vice versa. System firmware allocates addresses to map between the GSC and PCI buses and programs this information into configuration registers in the bus interface ASIC. Once programmed, the bus interface ASIC performs the following tasks:

- Forward writes transactions from the GSC bus to the PCI bus. Since the write originates in the processor, this task is called a processor I/O write.
- Forward reads requests from the GSC bus to the PCI bus, waits for a PCI device to respond, and returns the read data from the PCI bus back to the GSC bus. Since the read originates in the processor, this task is called a processor I/O read.
- Forward writes transactions from the PCI bus to the GSC bus. Since the destination of the write transaction is main memory, this task is called a direct memory access (DMA) write.
- Forward reads requests from the PCI bus to the GSC bus, waits for the GSC host to respond, and returns the read data from the GSC bus to the PCI bus. Since the source of the read data is main memory, this task is called a DMA read.

Figure 3 shows a block diagram of the internal architecture of the bus interface ASIC. The bus interface ASIC uses five asynchronous FIFOs to send address, data, and transaction information between the GSC and PCI buses.

Figure 1

HP B-class workstation block diagram.

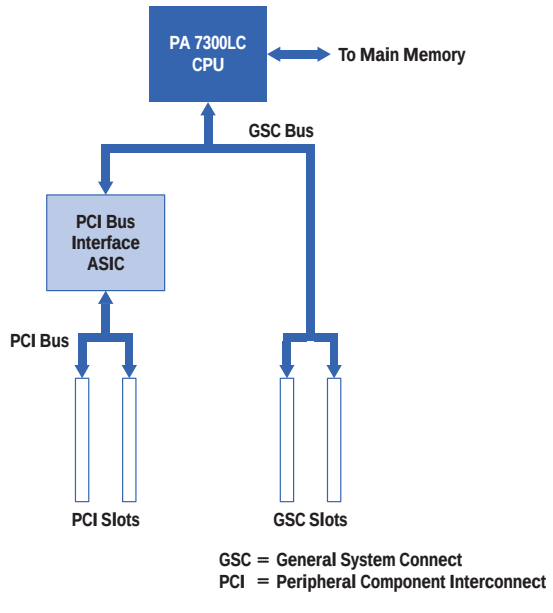


Figure 2

HP C-class workstation block diagram.

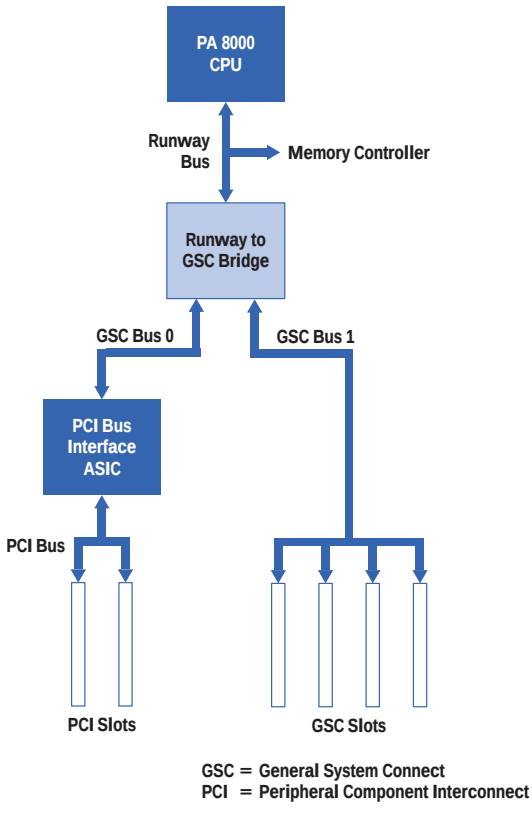
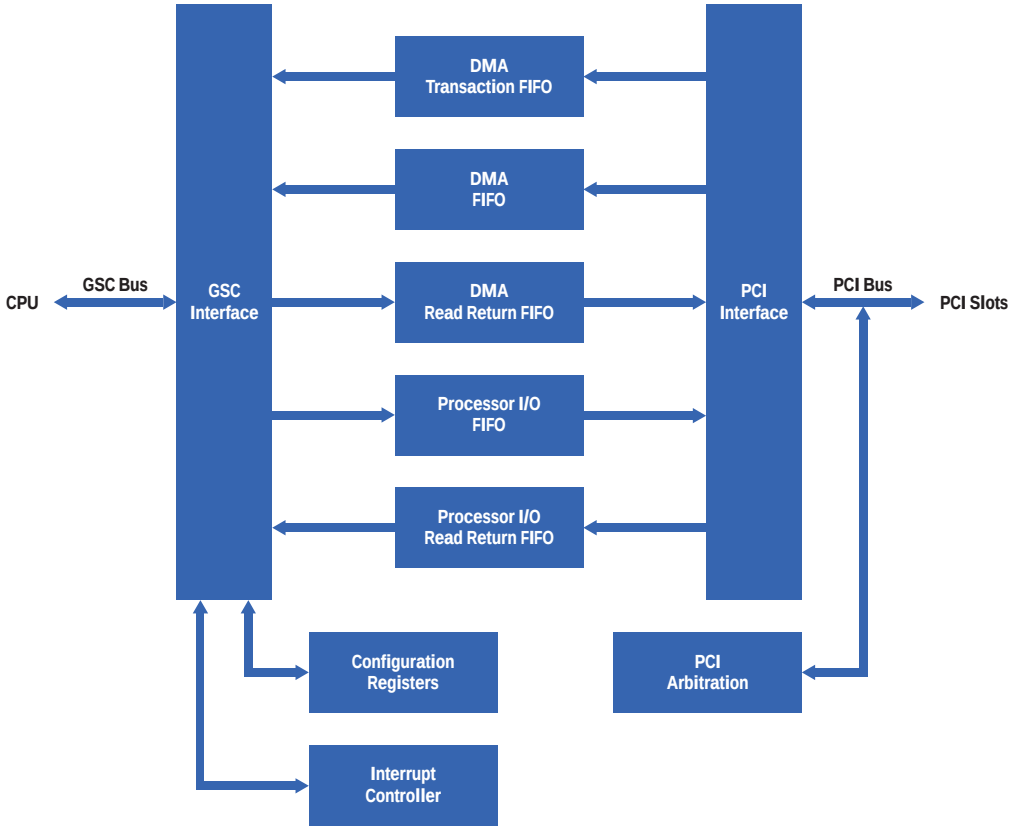


Figure 3

A block diagram of the architecture for the bus interface ASIC.



A FIFO is a memory device that has a port for writing data into the FIFO and a separate port for reading data out of the FIFO. Data is read from the FIFO in the same order that it was written into the FIFO. The GSC bus clock is asynchronous to the PCI bus clock. For this reason, the FIFOs need to be asynchronous. An asynchronous FIFO allows the data to be written into the FIFO with a clock that is asynchronous to the clock used to read data from the FIFO.

Data flows through the bus interface ASIC are as follows:

- Processor I/O write:
  - The GSC interface receives both the address and the data for the processor I/O write from the GSC bus and loads them into the processor I/O FIFO.
  - The PCI interface arbitrates for the PCI bus.
  - The PCI interface unloads the address and data from the processor I/O FIFO and masters the write on the PCI bus.
- Processor I/O read:
  - The GSC interface receives the address for the processor I/O read from the GSC bus and loads it into the processor I/O FIFO.
  - The PCI interface arbitrates for the PCI bus.
  - The PCI interface unloads the address from the processor I/O FIFO and masters a read on the PCI bus.
  - The PCI interface waits for the read data to return and loads the data into the processor I/O read return FIFO.
  - The GSC interface unloads the processor I/O read return FIFO and places the read data on the GSC bus.

- DMA Write:
  - The PCI interface receives both the address and the data for the DMA write from the PCI bus and loads them into the DMA FIFO.
  - The PCI interface loads control information for the write into the DMA transaction FIFO.
  - The GSC interface arbitrates for the GSC bus.
  - The GSC interface unloads the write command from the DMA transaction FIFO, unloads the address and data from the DMA FIFO, and masters the write on the GSC bus.
- DMA Read:
  - The PCI interface receives the address for the DMA read from the PCI bus and loads it into the DMA FIFO.
  - The GSC interface arbitrates for the GSC bus.
  - The GSC interface unloads the address from the DMA FIFO and masters a read on the GSC bus
  - The GSC interface then waits for the read data to return and loads the data into the DMA read return FIFO.
  - The PCI interface unloads the DMA read return FIFO and places the read data on the PCI bus.

Architectural Challenges

One of the difficulties of joining two dissimilar I/O buses is achieving peak I/O bus performance despite the fact that the transaction structures are different for both I/O buses. For example, transactions on the GSC bus are fixed length with not more than eight words per transaction while transactions on the PCI bus are of arbitrary length. It is critical to create long PCI transactions to reach peak bandwidth on the PCI bus. For better performance and whenever possible, the bus interface ASIC coalesces multiple processor I/O write transactions from the GSC bus into a single processor I/O write transaction on the PCI bus. For DMA writes, the bus interface ASIC needs to determine the optimal method of breaking variable-size PCI transactions into one-, two-, four-, or eight-word GSC transactions. The PCI interface breaks DMA writes into packets and communicates the transaction size to the GSC interface through the DMA transaction FIFO.

Another difficulty of joining two dissimilar I/O buses is avoiding deadlock conditions. Deadlock conditions can occur when a transaction begins on both the GSC and PCI buses simultaneously. For example, if a processor I/O read begins on the GSC bus at the same time a DMA read begins on the PCI bus, then the processor I/O read will wait for the DMA read to be completed before it can master its read on the PCI bus. Meanwhile, the DMA read will wait for the processor I/O read to be completed before it can master its read on the GSC bus. Since both reads are waiting for the other to be completed, we have a deadlock case. One solution to this problem is to detect the deadlock case and retry or split one of the transactions to break the deadlock. In general, the bus interface ASIC uses the GSC split protocol to divide a GSC transaction and allow a PCI transaction to make forward progress whenever it detects a potential deadlock condition.

Unfortunately, the bus interface ASIC adds more latency to the round trip of DMA reads. This extra latency can have a negative affect on DMA read performance. The C-class workstation has a greater latency on DMA reads than the B-class workstation. This is due primarily to the extra layer of bus bridges that the DMA read must traverse to get to memory and back (refer to **Figures 1** and **2**). The performance of DMA reads is important to outbound DMA devices such as network cards and disk controllers. The extra read latency is hidden by prefetching consecutive data words from main memory with the expectation that the I/O device needs a block of data and not just a word or two.

Open Standard Challenges

The PCI bus specification, like most specifications, is not perfect. There are areas where the specification is vague and open to interpretation. Ideally, when we find a vague area of a specification, we investigate how other designers have interpreted the specification and follow the trend. With a proprietary bus this often means simply contacting our partners within HP and resolving the issue. With an industry-standard bus, our partners are not within the company, so resolving the issue is more difficult. The PCI mail reflector, which is run by the PCI SIG at www.pcsig.com, is sometimes helpful for resolving such issues. Monitoring the PCI mail reflector also gives the

benefit of seeing what parts of the PCI specification appear vague to others. Simply put, engineers designing to a standard need a forum for communicating with others using that standard. When designing to an industry standard, that forum must by necessity include wide representation from the industry.

The PCI specification has guidelines and migration plans that PCI card vendors are encouraged to follow. In practice, PCI card vendors are slow to move from legacy standards to follow guidelines or migration plans. For example, the PCI bus supports a legacy I/O* address space that is small and fragmented. The PCI bus also has a memory address space that is large and has higher write bandwidth than the I/O address space. For obvious reasons, the PCI specification recommends that all PCI cards map their registers to the PCI I/O address space and the PCI memory address space so systems will have the most flexibility in allocating base addresses to I/O cards. In practice, most PCI cards still only support the PCI address I/O space. Since we believed that the PCI I/O address space would almost never be used, trade-offs were made in the design of the bus interface ASIC that compromised the performance of transactions to the PCI I/O address space.

Another example in which the PCI card vendors follow legacy standards rather than PCI specification guidelines is in the area of PCI migration from 5 volts to 3.3 volts. The PCI specification defines two types of PCI slots: one for a 5-volt signaling environment and one for a 3.3-volt signaling environment. The specification also defines three possible types of I/O cards: 5-volt only, 3.3-volt only, or universal. As their names imply, 5-volt-only and 3.3-volt-only cards only work in 5-volt and 3.3-volt slots respectively. Universal cards can work in either a 5-volt or 3.3-volt slot. The PCI specification recommends that PCI card vendors only develop universal cards. Even though it costs no more to manufacture a universal card than a 5-volt card, PCI card vendors are slow to migrate to universal cards until volume platforms (that is, Intel-based PC platforms) begin to have 3.3-volt slots.

Verification

Verification Methodology and Goals

The purpose of verification is to ensure that the bus interface ASIC correctly meets the requirements described in the design specification. In our VLSI development process this verification effort is broken into two distinct parts called phase-1 and phase-2. Both parts have the intent of proving that the design is correct, but each uses different tools and methods to do so. Phase-1 verification is carried out on a software-based simulator using a model of the bus interface ASIC. Phase-2 verification is carried out on real chips in real systems.

Phase-1. The primary goals of phase-1 verification can be summarized as correctness, performance, and compliance. Proving correctness entails showing that the Verilog model of the design properly produces the behavior detailed in the specification. This is done by studying the design specification, enumerating a function list of operations and behaviors that the design is required to exhibit, and generating a suite of tests that verify all items on that function list. Creating sets of randomly generated transaction combinations enhances the test coverage by exposing the design to numerous corner cases.

Performance verification is then carried out to prove that the design meets or exceeds all important performance criteria. This is verified by first identifying the important performance cases, such as key bandwidths and latencies, and then generating tests that produce simulated loads for performance measurement.

Finally, compliance testing is used to prove that the bus protocols implemented in the design will work correctly with other devices using the same protocol. For a design such as the bus interface ASIC that implements an industry-standard protocol, special consideration was given to ensure that the design would be compatible with a spectrum of outside designs.

Phase-2. This verification phase begins with the receipt of the fabricated parts. The effort during this phase is primarily focused on testing the physical components, with simulation techniques restricted to the supporting role of duplicating and better understanding phenomenon seen on the bench. The goals of phase-2 verification can be summarized as compliance, performance, and compatibility. Therefore, part of phase-2 is spent proving that the physical device behaves on the bench the same as it did in simulation. The heart of phase-2, however, is that the design is finally tested for compatibility with the actual devices that it will be connected to in a production system.

* Legacy refers to the Intel I/O port space.

Verification Challenges

From the point of view of a verification engineer, there are benefits and difficulties in verifying the implementation of an industry-standard bus as compared to a proprietary bus. One benefit is that since PCI is an industry standard, there are plenty of off-the-shelf simulation and verification tools available. The use of these tools greatly reduces the engineering effort required for verification, but at the cost of a loss of control over the debugging and feature set of the tools.

The major verification challenge (particularly in phase-1) was proving compliance with the PCI standard. When verifying compliance with a proprietary standard there are typically only a few chips that have to be compatible with one another. The design teams involved can resolve any ambiguity in the bus specification. This activity tends to involve only a small and well-defined set of individuals. In contrast, when verifying compliance with an open standard there is usually no canonical source that can provide the correct interpretation of the specification. Therefore, it is impossible to know ahead of time where devices will differ in their implementation of the specification. This made it somewhat difficult for us to determine the specific tests required to ensure compliance with the PCI standard. In the end, it matters not only how faithfully the specification is followed, but also whether or not the design is compatible with whatever interpretation becomes dominant.

The most significant challenge in phase-2 testing came in getting the strategy to become a reality. The strategy depended heavily on real cards with real drivers to demonstrate PCI compliance. However, the HP systems with PCI slots were shipped before any PCI cards with drivers were supported on HP workstations. Creative solutions were found to develop a core set of drivers to complete the testing. However, this approach contributed to having to debug problems closer to shipment than would have been optimal. Similarly, 3.3-volt slots were to be supported at first shipment. The general unavailability of 3.3-volt or universal (supporting 5 volts and 3.3 volts) cards hampered this testing. These are examples of the potential dangers of "preenabling" systems with new hardware capability before software and cards to use the capability are ready.

An interesting compliance issue was uncovered late in phase-2. One characteristic of the PA 8000 C-class system is that when the system is heavily loaded, the bus interface ASIC can respond to PCI requests with either long read latencies (over 1 µs before acknowledging the transaction) or many (over 50) sequential PCI retry cycles. Both behaviors are legal with regard to the PCI 2.0 bus specification, and both of them are appropriate given the circumstances. However, neither of these behaviors is exhibited by Intel's PCI chipsets, which are the dominant implementation of the PCI bus in the PC industry. Several PCI cards worked fine in a PC, but failed in a busy C-class system. The PCI card vendors had no intention of designing cards that were not PCI compliant, but since they only tested their cards in Intel-based systems, they never found the problem. Fortunately, the card vendors agreed to fix this issue on each of their PCI cards. If there is a dominant implementation of an industry standard, then deviating from that implementation adds risk.

Firmware

Firmware is the low-level software that acts as the interface between the operating system and the hardware. Firmware is typically executed from nonvolatile memory at startup by the workstation. We added the following extensions to the system firmware to support PCI:

- A bus walk to identify and map all devices on the PCI bus
- A reverse bus walk to configure PCI devices
- Routines to provide boot capability through specified PCI cards.

The firmware bus walk identifies all PCI devices connected to the PCI bus and records memory requirements in a resource request map. When necessary, the firmware bus walk will traverse PCI-to-PCI bridges.* If a PCI device has Built-in Self Test (BIST), the BIST is run, and if it fails, the PCI device is disabled and taken out of the resource request map. As the bus walk unwinds, it initializes bridges and allocates resources for all of the downstream PCI devices.

Firmware also supports booting the HP-UX operating system from two built-in PCI devices. Specifically, firmware can load the HP-UX operating system from either a disk attached to a built-in PCI SCSI chip or from a file server attached to a built-in PCI 100BT LAN chip.

* A PCI-to-PCI bridge connects two PCI buses, forwarding transactions from one to the other.

Firmware Challenges

The first challenge in firmware was the result of another ambiguity in the PCI specification. The specification does not define how soon devices on the PCI bus must be ready to receive their first transaction after the PCI bus exits from reset. Several PCI cards failed when they were accessed shortly after PCI reset went away. These cards need to download code from an attached nonvolatile memory before they will work correctly. The cards begin the download after PCI reset goes away, and it can take hundreds of milliseconds to complete the download. Intel platforms delay one second after reset before using the PCI bus. This informal compliance requirement meant that firmware needed to add a routine to delay the first access after the PCI bus exits reset.

Interfacing with other ASICs implementing varying levels of the PCI specification creates additional challenges. Compliance with PCI 2.0 (PCI-to-PCI) bridges resulted in two issues for firmware. First, the bridges added latency to processor I/O reads. This extra latency stressed a busy system and caused some processor I/O reads to timeout in the processor and bring down the system. The firmware was changed so that it would reprogram the processor timeout value to allow for this extra delay. The second issue occurs when PCI 2.0 bridges are stacked two or more layers deep. It is possible to configure the bridges such that the right combination of processor I/O reads and DMA reads will cause the bridges to retry each others transactions and cause a deadlock or starve one of the two reads. Our system firmware fixes this problem by supporting no more than two layers of PCI-to-PCI bridges and configuring the upstream bridge with different retry parameters than the downstream bridge.

Operating System Support

The HP-UX operating system contains routines provided for PCI-based kernel drivers called *PCI services*. The first HP-UX release that provides PCI support is the 10.20 release. An infrastructure exists in the HP-UX operating system for kernel-level drivers, but the PCI bus introduced several new requirements. The four main areas of direct impact include context dependent I/O, driver attachment, interrupt service routines (ISR), and endian issues. Each area requires special routines in the kernel's PCI services.

Context Dependent I/O

In the HP-UX operating system, a centralized I/O services context dependent I/O (CDIO) module supplies support for drivers that conform to its model and consume its services. Workstations such as the C-class and B-class machines use the workstation I/O services CDIO (WSIO CDIO) for this abstraction layer. The WSIO CDIO provides general I/O services to bus-specific CDIOs such as EISA and PCI. Drivers that are written for the WSIO environment are referred to as WSIO drivers. The services provided by WSIO CDIO include system mapping, cache coherency management, and interrupt service linkage. In cases where WSIO CDIO does need to interface to the I/O bus, WSIO CDIO translates the call to the appropriate bus CDIO. For the PCI bus, WSIO CDIO relies on services in PCI CDIO to carry out bus-specific code.

Ideally, all PCI CDIO services should be accessed only through WSIO CDIO services. However, there are a number of PCI-specific calls that cannot be hidden with a generic WSIO CDIO interface. These functions include PCI register operations and PCI bus tuning operations.

Driver Attachment

The PCI CDIO is also responsible for attaching drivers to PCI devices. The PCI CDIO completes a PCI bus walk, identifying attached cards that had been set up by firmware. The PCI CDIO initializes data structures, such as the interface select code (ISC) structure, and maps the card memory base register. Next, the PCI CDIO calls the list of PCI drivers that have linked themselves to the PCI attach chain.

The PCI driver is called with two parameters: a pointer to an ISC structure (which contains mapping information and is used in most subsequent PCI services calls) and an integer containing the PCI device's vendor and device IDs. If the vendor and device IDs match the driver's interface, the driver attach routine can do one more check to verify its ownership of the device by reading the PCI subsystem vendor ID and subsystem ID registers in the configuration space. If the driver does own this PCI device, it typically initializes data structures, optionally links in an interrupt service routine, initializes and claims the interface, and then calls the next driver in the PCI attach chain.

Interrupt Service Routines

The PCI bus uses level-sensitive, shared interrupts. PCI drivers that use interrupts use a WSIO routine to register their interrupt service routine with the PCI CDIO. When a PCI interface card asserts an interrupt, the operating system calls the PCI CDIO to do the initial handling. The PCI CDIO determines which PCI interrupt line is asserted and then calls each driver associated with that interrupt line.

The PCI CDIO loops, calling drivers for an interrupt line until the interrupt line is deasserted. When all interrupt lines are deasserted, the PCI CDIO reenables interrupts and returns control to the operating system. To prevent deadlock, the PCI CDIO has a finite (although large) number of times it can loop through an interrupt level before it will give up and leave the interrupt line disabled. Once disabled, the only way to reenable the interrupt is to reboot the system.

PCI Endian Issues

PCI drivers need to be cognizant of endian issues.* The PCI bus is inherently little endian while the rest of the workstation hardware is big endian. This is only an issue with card register access when the register is accessed in quantities other than a byte. Typically there are no endian issues associated with data payload since data payload is usually byte-oriented. For example, network data tends to be a stream of byte data. The PCI CDIO provides one method for handling register endian issues. Another method lies in the capability of some PCI interface chips to configure their registers to be big or little endian.

Operating System Support Challenges

We ran into a problem when third-party card developers were porting their drivers to the HP-UX operating system. Their drivers only looked at device and vendor identifiers and claimed the built-in LAN inappropriately. Many PCI interface cards use an industry-standard bus interface chip as a front end and therefore share the same device and vendor IDs. For example, several vendors use the Digital 2114X family of PCI-to-10/100 Mbits/s Ethernet LAN controllers, with each vendor customizing other parts of the network interface card with perhaps different physical layer entities. It is possible that a workstation could be configured with multiple LAN interfaces having the same vendor and device ID with different subsystem IDs controlled by separate drivers. A final driver attachment step was added to verify the driver's ownership of the device. This consisted of reading the PCI subsystem vendor ID and subsystem ID registers in the configuration space.

The HP-UX operating system does not have the ability to allocate contiguous physical pages of memory. Several PCI cards (for example, SCSI and Fibre Channel) require contiguous physical pages of memory for bus master task lists. The C-class implementation, which allows virtual DMA through TLB (translation lookaside buffer) entries, is capable of supplying 32K bytes of contiguous memory space. In the case of the B-class workstation, which does not support virtual DMA, the team had to develop a workaround that consisted of preallocating contiguous pages of memory to enable this class of devices.

Conclusion

PCI and Interoperability. We set out to integrate PCI into the HP workstations. The goal was to provide our systems with access to a wide variety of industry-standard I/O cards and functionality. The delivery of this capability required the creation and verification of a bus interface ASIC and development of the appropriate software support in firmware and in the HP-UX operating system.

Challenges of Interfacing with Industry Standards. There are many advantages to interfacing with an industry standard, but it also comes with many challenges. In defining and implementing an I/O bus architecture, performance is a primary concern. Interfacing proprietary and industry-standard buses and achieving acceptable performance is difficult. Usually the two buses are designed with different goals for different systems, and determining the correct optimizations requires a great deal of testing and redesign.

Maintaining compliance with an industry standard is another major challenge. It is often like shooting at a moving target. If another vendor ships enough large volumes of a nonstandard feature, then that feature becomes a de facto part of the standard. It is also very difficult to prove that the specification is met. In the end, the best verification techniques for us involved simply testing the bus interface ASIC against as many devices as possible to find where the interface broke down or performed poorly.

* Little endian and big endian are conventions that define how byte addresses are assigned to data that is two or more bytes long. The little endian convention places bytes with lower significance at lower byte addresses. (The word is stored "little-end-first.") The big endian convention places bytes with greater significance at lower byte addresses. (The word is stored "big-end-first.")

Finally, it is difficult to drive development and verification unless the functionality is critical to the product being shipped. The issues found late in the development cycle for the bus interface ASIC could have been found earlier if the system had required specific PCI I/O functionality for initial shipments. The strategy of preenabling the system to be PCI compatible before a large number of devices became available made it difficult to achieve the appropriate level of testing before the systems were shipped.

Successes. The integration of PCI into the HP workstations through design and verification of the bus interface ASIC and the development of the necessary software components has been quite successful. The goals of the PCI integration effort were to provide fully compatible, high-performance PCI capability in a cost-effective and timely manner. The design meets or exceeds all of these goals. The bandwidth available to PCI cards is within 98 percent of the bandwidth available to native GSC cards. The solution was ready in time to be shipped in the first PCI-enabled HP workstations B132, B160, C160, and C180.

The bus-bridge ASIC and associated software have since been enhanced for two new uses in the second generation of PCI on HP workstations. The first enhancement provides support for the GSC-to-PCI adapter to enable specific PCI functionality on HP server GSC I/O cards. The second is a version of the bus interface supporting GSC-2x (higher bandwidth GSC) and 64-bit PCI for increased bandwidth to PCI graphics devices on HP C200 and C240 workstations.

Acknowledgments

This article is a summary of some of the challenges experienced by numerous team members involved in the integration of PCI into workstations. We would like to specifically thank several of those team members who helped contribute to and review this article. George Letey, Frank Lettang, and Jim Peterson assisted with the architecture section. Vicky Hansen, Dave Klink, and J.L. Marsh provided firmware details. Matt Dumm and Chris Hyser reviewed the operating system information.

References

1. W. Bryg, K. Chan, and N. Fiduccia, "A High-Performance, Low-Cost Multiprocessor Bus for Workstations and Midrange Servers," *Hewlett-Packard Journal*, Vol. 47, no. 1, February 1996, p. 18.

Ric L. Lewis

A project manager at the HP Workstation Systems Division, Ric Lewis was responsible for managing the development of the PCI bus interface ASIC. He came to HP in 1987 after receiving a BSEE degree from Utah State University. He also has an MSEE degree (1993) from Stanford University and an MBA (1992) from Santa Clara University. Ric was born in Twin Falls, Idaho, and he is married and has one son. His outside interests include basketball, mountain biking, and skiing.

Erin A. Handgen

Erin Handgen is a technical contributor at the HP Workstation Systems Division working on ASICs for HP workstations. He was the lead engineer for the PCI bus interface ASIC during the shipment phase. He has a BS degree in computer and electrical engineering (1986) and an MSEE degree (1988) from Purdue University. He joined HP in 1988. Born in Warsaw, Indiana, he is married and has three children. His outside interests include astronomy, camping, and hiking.

Nicholas J. Ingegneri

A hardware design engineer at the HP Fort Collins Systems Laboratory, Nicholas Ingegneri was the lead verification engineer for the PCI bus interface ASIC. He has a BSEE degree (1989) from the University of Nevada at Reno and an MSEE degree (1994) from Stanford University. He came to HP in 1990. His outside interests include travel and camping.

Glen T. Robinson

Glen Robinson is a technical contributor at the HP Workstation Systems Division. He came to HP in 1979 and was responsible for the subsystem test for the PCI network and kernel drivers. He has an MBA degree (1979) from Chapman College. Born in Santa Monica, California, he is married and has two grown children. He enjoys biking and AKC hunting tests with Labrador retrievers.

▶ Go to Next Article

▶ Go to Journal Home Page