Problems updating FreeBSD's card system from ISA to PCI

M. Warner Losh
Timing Solutions, Inc
Boulder, Co
imp@village.org

Abstract

FreeBSD's 16-bit PC Card implementation has used the ISA legacy interface. PCI support was added by making PCI-CardBus and PCI-PCMCIA bridges behave in ISA compatibility mode. While this technique worked in laptops, it made support of add-in PCI cards with CardBus or PCMCIA bridges impossible. PCI-PC Card bridges are unlike traditional devices because they can have connections to multiple busses, offering both ISA and PCI interrupt routing for them and any devices connected to them. Expanding support to add-in PCI cards with 16-bit PC Cards connected exposed weaknesses in the PC Card implementation of FreeBSD as well as other parts of the system. Vendor BIOS quality, variance in hardware implementation details from standard and weaknesses in the FreeBSD development model made incorporation of these improvements into FreeBSD 4.4-RELEASE difficult. Lessons learned will be incorporated into the 32-bit CardBus support forthcoming in FreeBSD 5.0.

1 Problem

FreeBSD's[FreeBSD] PC Card implementation in FreeBSD 4.3-RELEASE and earlier only routed ISA interrupts to 16-bit PC Card cards. It configured PCI-PC Card bridges to look like old ISA devices, complete with ISA interrupt routing. This strategy worked well for Laptops where the ISA interrupt signals were available to the PCI bridge. However, for add-in PCI-PC Card bridge cards, this strategy failed because the add-in cards do not have connections to the ISA bus' interrupts. Since there was no connection to the ISA interrupts, these add-in cards could not be configured to use them. The growing popularity of these cards in new PCI only systems was a problem. Additionally, ISA interrupts cannot be shared in the absence of specialized hardware which laptops lack, making some laptops very difficult to configure. Since ISA interrupts are also hard to detect for devices that do not have drivers configured for them, mysterious failures happened frequently when a novice user would configure a FreeBSD's PC Card system improperly.

An effort was made to allow PCI routing of interrupts on PCI bridges, and for sharing of interrupts of 16-bit PC Cards when they are connected to a PCI-PC Card bridge. No attempt was made to expand FreeBSD support to include 32-bit CardBus cards as part of this effort. As a result of this effort, 16-bit PC Cards in add-in PCI-PC Card bridge cards now work, as do most of the bridges supported by the previous code. In addition, the support load from improperly configured laptops has dropped with the new automatic interrupt configuration.

2 Implementation History

Prior to the 4.4-Release, FreeBSD's PC Card implementation relied on treating all PC Card[PC Card] bridges as ISA devices. FreeBSD's implementation was written in a time before PCI[PCI] devices that supported PC Card were widely available. Two basic types of PCI bridges have appeared. One is a 16-bit PC Card bridge and called a PCI to PCMCIA bridge. The other is for 16 or 32-bit cards and called a PCI to CardBus bridge. While most of these bridges support a standard configuration space and register set, some older bridges do not and need varying amounts of special case code. This paper uses the term PCI-PC Card bridge to generically refer to any of these bridges.

FreeBSD's PC Card code was able to treat these PCI-PC Card bridges as ISA devices because they generally were connected to the ISA bus or ISA bus' interrupts in the laptop configuration. Since laptop support was the primary target of FreeBSD's PC Card code, little attention was paid to the add-in card problem since most machines had an ISA slot, which was well supported. Microsoft's hardware design guides have ensured that laptops manufacturers made these connections to the ISA bus. FreeBSD was able to program these PCI devices in such a way as to ensure that the ISA compatibility worked for laptop users, at the price that the laptop user would have to hand configure each laptop to list those interrupts not used by other hardware.

In recent years, a number of trends in the industry has made ignoring the add-in card problem less and less appealing. The first trend was toward more and more desktop or server machines having PC Card in them at all. Many users desired to read flash cards from their digital cameras on desktops. One of the solutions to do this was to install PC card slots and insert the flash cards into the PC Card slots, possibly with an adapter. A second reason for having PC Card slots in a desktop or server machine was for wireless support. While some of the early, pre–802.11 wireless cards were available in PC Card, ISA and PCI versions, nearly all of the recent 802.11[WiFi] and 802.11b cards have only PC Card versions. To place an 802.11b wireless card into a desktop or server machine, you needed to have some type of PC Card bridge to provide the slots to insert the wireless card into. ISA, PCI and other solutions exist to solve these needs. The PCI add-in card forced FreeBSD wireless PC users to use ISA PC Card bridges. This solution was adequate for a long time, but other trends made it less viable.

The second trend was the retirement of the ISA bus expansion from PC compatible computers. The number of ISA slots in system had been shrinking for years as the superior PCI bus provided a number of advantages. Microsoft had a vested interest in retiring ISA support from Windows, so created the PC 98 System Design Guide (not to be confused with the PC-98[PC-98] machines made by NEC and sold in Japan). To be compliant with the PC 98 Design Guide, a system could not have any ISA expansion bus slots if it was manufactured after a certain date. Subsequent design guides[PC99, PC2001] reaffirmed this restriction. Many systems now omit ISA bus expansion slots. FreeBSD users that wanted to connect wireless cards to these system were left without a general solution (although specialized solutions remained available to PCI only systems).

The third trend has been towards integration of more and more devices into laptops which can cause a decreased availability of unused interrupts. In some generations of laptops, each of these devices would use an interrupt. Since FreeBSD couldn't share PC Card interrupts with other devices, some machines could not find enough interrupts to support multiple PC cards. Some laptops also could not reserve an interrupt for card change status events. As the number of free interrupts on the typical laptop has declined, the difficulty and complexity of correctly configuring the system grew. While FreeBSD has tried to prevent the user from reusing an interrupt, it cannot protect against devices where no driver has attached and there's no way of knowing which resources are consumed. Laptops release in the last year or two have shown a trend away from ISA devices in favor of PCI devices, which automatically configure resources and which can share IRQs. While the number of devices on laptops continues to grow, their impact has been lessened somewhat, leaving only the problems of highly integrated laptops from a few years ago.

3 Prior Implementation Details

The FreeBSD PC Card implementation prior to 4.4–RELEASE was done in two parts. One part of the implementation was done in the kernel, while another part was done in a user-land daemon.

When the system boots, the kernel detects PCI-PC Card bridges and programs them into ISA compatibility mode. Sometimes, for proper operation, special settings in the BIOS are required. A separate part of the kernel then detects the ISA devices (or the PCI ones in ISA compatibility mode). On attach, things are setup to detect cards arriving into the system, using the traditional ISA interface. This mode uses only the ExCA registers, defined in the PCIC standard, to manipulate the bridge.

When a card is inserted into a PC Card slot, the kernel does a basic power up of the card, and notifies pccardd that a card status has changed. Once the card is ready, pccardd will read its meta data, which it uses to configure the card. It then resets the card, and tells the kernel which resources the card requires, and what driver to attach to the card. The kernel then asks the driver if it really supports the card, and if so, adds it to the FreeBSD device tree. The pccardd daemon is then informed of the success of the operation. Any final user-land configuration of the new device will follow. When a card is ejected, the kernel will tell detach the device and tell pccardd that the device is gone. A program exists to query the state of the PC Card system, read the card's meta data and to force configuration of a card exists.

During the configuration process, both the kernel and pccardd will manipulate the hardware in a number of ways. They set bits in the hardware to cause certain things to happen, then polls the bits for a response. So as to allow other system activity in the interim, there is a small sleep between attempts to check the bits. This polling works well in an ISA system. If an interrupt handler fails to clear the interrupt condition, nothing happens for the typical ISA device (except maybe to miss future interrupts) since the ISA bus uses edge triggered interrupts. While FreeBSD is manipulating the hardware to configure a PC Card, various interrupts may occur. The FreeBSD implementation didn't need to handle these interrupts to appear to be working, since they were one shots.

4 Important Hardware Details

A number of problems present themselves when this solution is attempted. However, to understand them one must understand how PCI hardware works, what 16-bit PC Cards do at each step of their configuration process, and how the PCI-PC Card bridges react. Some of these items are well documented, while others are observed behavior.

The PCI bus differs from the ISA bus in many ways. Unlike the ISA bus, the PCI bus has level sensitive interrupts. The PCI bus also allows for interrupt sharing. The PCI bus defines 4 signals for interrupts that are routed to each device on the PCI bus, **INTA#**, **INTB#**, **INTC#** and **INTD#**. The PCI device can pick one it will use (although that choice is usually hardwired to **INTA#**). When a PCI device asserts an interrupt, it stays asserted until the interrupting condition is cleared. Failure to clear an interrupt condition in the device's ISR will cause the PCI **INTx#** line to remain asserted. Since the the interrupt remains when the ISR exists, it will be called again. Lather. Rinse. Repeat. If the ISR never clears the condition, this causes an interrupt storm. This will cause the system to loop forever in a wedged, or semi-wedged state. If one fails to clear an interrupt condition on an ISA device, no such pathological behavior happens.

PCI-PC Card bridges assert two types of interrupts. The first type is for card status changes, while the second is for function interrupts. The first type can be masked, but the card function cannot be masked. When a PC Card indicates that an interrupt is present, the PCI interrupt line is asserted. Short of instructing the bridge to use ISA signaling for card function interrupts, there is no way to mask this interrupt. Every family of PCI-PC Card bridges have their own method to control how interrupts are routed, which adds complication. The implications of this are that both the bridge driver and children drivers of the bridge must be more careful in their interrupt handling. PCI-PC Card bridges do not offer a way to mask the card function interrupts. Driver writers must ensure that an ISR is present before the card begins to generate interrupts.

A 16-bit card can be reset by writing certain bits to a configuration register (sometimes called a COR reset). When the card is in this reset state, it's **READY** pin will sometimes be pulled low to indicate that the card is not ready (or that the card has finished resetting, depending on the card). The **READY** pin is shared with the **IREQ#** pin for 16-bit PC Cards. The **IREQ#** pin is active low and indicates that the card is interrupting. This signal is held low until the reset bits are turned back off. The COR register varies in location from PC Card to PC Card, and is contained in the CIS (or meta data) that is parsed by pccardd.

When powering a 16-bit card up, an interrupt is generated when the power sequence is complete on many (but apparently not all) PC Card bridges. Driver writers are expected to detect this interrupt, acknowledge it, and continue with their power on sequence.

5 Naive Implementation

A naive implementation strategy to add PCI interrupts to FreeBSD 16-bit implementation would be to create a real PCI attachment for the PC Card bridges, program the PCI bridge into "native" mode, but otherwise leave the previous implementation alone. The author tried a naive implementation in full ignorance of the problems he faced in doing so.

The experienced reader may have already notice several pitfalls from the above descriptions. Nearly all of the problems result in system hangs due to interrupt storms. Some problems are less obvious and more failsafe, but can result in no PC Cards working.

Each of the above problems posed a hurdle in upgrading FreeBSD's PC Card implementation. These problems were dealt with in two ways. Many of the busy wait portions of the code were rewritten and moved entirely into the kernel. They were made interrupt driven and each of the interrupts were acknowledge. These were the easy ones to fix (once you discovered which magic register the interrupt source bits were stored in, as there are more than one). The hard part of fixing these bugs was always finding sufficient documentation to be sure that the interrupt bits in question were the right ones.

Two of the issues were hard to fix and merit further discussion. The first issue is that of the COR reset. In the user-land approach, all knowledge of where to write the magic bits rests in pccardd. Indeed, when level interrupts weren't an issue, it made good sense to have all the knowledge there. The normal PC card bus driver interfaces allowed the right part of the attribute memory to be mapped at the right times to write to the COR. pccardd didn't have to manage that at all, it just would seek to the right location; write to set the reset bit; wait and then seek/write to clear the reset bit. Moving all this functionality into the kernel proved to be difficult. The kernel would have to be told where the COR was located, and the ISR routine would need to write the right value to the COR. Since the old implementation made this tedious to do, I elected to fix the problem in another way. Examination of the BSD/OS, Linux[Linux-CS] and NetBSD[NetBSD] PC Card code showed that none of them had a COR reset at all. It appears to be unique to FreeBSD's PC Card implementation. I eliminated it and so far no ill effects have be traced to its elimination.

The second issue was PC Card function interrupts. During the configuration process, these were asserted at unpredictable times. Each card appears to have its own way of resetting these condition during startup. Since the PC Card bus driver had no knowledge of the plurality of card dependent methods, it could not acknowledge the interrupt, or otherwise force it to go away, thus causing an interrupt storm. The solution adopted by the author was to route the function interrupts to the ISA bus until the driver had attached an interrupt status routine. This solved the transient problem, but may be the cause of strange system hangs on reboot.

6 PCI Interrupt Routing

Once the simple problems were solved, I deployed my code on a number of laptops and desktop system with a variety of chip-sets. The results showed that addition hurdles awaited. However, to understand them, some basic hardware and system boot strap issues must be discussed.

The biggest problem encountered was PCI interrupt routing. On the laptops I had done the initial development the BIOS automatically routed an interrupt to the PCI - Cardbus bridge. It turns out that the BIOS is not required to route the interrupts, and the OS is responsible for doing this in some cases.

The PCI interrupt routing problem is thornier than it may appear on the surface. It is generally not the case that all interrupts can be directed to any pin on any card on the PCI bus. There are usually significant restrictions on which interrupts can be used. These restrictions come from two different sources. First, most PCI Cards are wired using the so-called barber pole arrangement, where **INTA#** of slot 1 is connected to **INTB#** of slot 2, **INTC#** of slot 3, **INTD#** of slot 4, **INTA#** of slot 5, etc. The standard doesn't specify the arrangement, so others exist. These pins are wire-ored together, so they can only be connected to one interrupt at the bus controller. If one is routing interrupts for a given device, and another device has routed an interrupt on the wire that, there is no choice in which interrupt gets routed. The two devices must share that interrupt. Since the wiring diagram of a PCI bus can be arbitrary and complex, the Operating System must have knowledge of this wiring diagram.

The second source of problems is controlling the bridges between the device and the CPU. They may have restrictions on which interrupts can be used, and each chip-set has a slightly different API to route the interrupts. Also, exact knowledge of how the bridges are wired together is often required to program the bridges correctly. The number of different bridge chip-sets found in deployed computers is large. Far larger than one could hope to ever support, even if one could get the wiring knowledge necessary to do the routing, which isn't possible in the absence of auxiliary tables.

To bring order to this chaotic state of affairs, the PCI SIG standardized the BIOS API in the PCI BIOS Specification[PCI BIOS]. This standard includes way to access each PCI Device's config space. version 2.1 added the ability to query $PIR Interrupt Routing Table and to route interrupts. These functions made it possible to do the routing.

Mike Smith wrote a fairly complete implementation of PCI interrupt routing using PCI BIOS. However, his implementation was for the forth coming FreeBSD 5.0, and not for FreeBSD 4.x which my PC Card implementation was targeted at. After porting most of the PCI interrupt code to 4.x, I was able to use many of the machines that had previously had unassigned interrupts for the PC Card bridge. This new interrupt routing was not without its faults.

6.1 Calling the BIOS

The PCI BIOS Specification defines two different ways one could call the PCI BIOS. The OS can look for a special signature in the BIOS ROM area of the computer (Physical address 0E0000h - 0FFFFFFh). Once it finds the signature, the entry point address is located with the entry and can be used to call the PCI BIOS using a standard CALL FAR (callf) instruction. The OS uses the BIOS32 Service Directory to obtain information to build the proper segments for this entry point.

The PCI BIOS also provides a 16-bit real and protected mode interface. The int 1Ah software interrupt is used to access the 16-bit interface, and operates in either real or protected mode. These BIOS functions may also be accessed by calling the industry standard address for int 1Ah (physical address 0FFE6Eh) by simulating a INT instruction (pushf followed by callf).

Unfortunately, the PCI BIOS standard doesn't appear to mandate that both of these interfaces be available to the OS. The PC97 Design Guide appears to rectify this situation by stipulating that both interfaces must be supported. However, the PC 98 Design guide required all machines to migrate towards using ACPI to configure their interrupts, a trend that continued in subsequent Design guides. As such, extremely new machines have also started to exhibit problems with PCI BIOS configuring the interrupts.

Unfortunately, the standard doesn't appear to mandate that all calling methods actually work. With some BIOSes, only a subset of the calling methods are available. If you are lucky, the one you want will work. Some BIOSes appear to support the BIOS calls, but later will perform an illegal access inside the BIOS call. On machines newer than about 1996, these issues appear to have been corrected. The PCI BIOS Specification was published in 1994, but it was not until after Window 95 became widely deployed that BIOS writers get their act together and make these functions work.

The current FreeBSD implementation requires that the BIOS implement the BIOS32 interface. As such, some older machines will not allow interrupts to be routed, and must fall back to using tradition ISA interrupt routing (if possible). Like Windows 98 and newer, it requires that the BIOS be at least PCI 2.1-compliant and provides the $PIR Interrupt routing Table.

6.2 Bad PCI config space

The PCI Standard[PCI] requires that all devices that can have interrupts routed to them, but do not currently, have an INTLINE (Configuration Register offset 0x3c) of 0xff. However, a large number of devices have been seen in the wild with a value of 0. For IBM AT compatible systems, this is an illegal value. There is no way to route IRQ 0 to anything except the programmable interval timer, sometimes used to keep track of time.

Many PCI CardBus bridges either default to a value of 0 for INTLINE, or the BIOS on the machine that they are in writes a value of 0 into INTLINE. The PC Card subsystem in FreeBSD properly doesn't know about these problems. The PCI subsystem had to have code added to it to work around these buggy implementations.

6.3 Standards Deviation

The PC Card standard[PC Card] for PCI bridges (formerly known as YENTA) is incomplete. It does not specify a way, for example, to cause the card function interrupts to be routed using ISA interrupts. And there are a number of minor, but important, variations between vendors on how to do simple things. The author worked around the need to have varying code initialize and manage the bridges by using the fairly standard jump table technique. Each family of PCI-PC Card bridges had its own set of functions, and a table pointing to them. The rest of the FreeBSD PC Card system uses these tables to manage the bridge as needed.

On problem the author had in creating these functions and tables was gaining access to all the different families of devices. There are five major families of "YENTA" compliant PCI CardBus bridges. Cirrus Logic makes one family, Texas Instruments another, Toshiba a third, Ricoh a forth and O2Micro a fifth. The Texas Instrument ones are the most commonly used, and easiest to find and support. While the O2Micros were the hardest to find (the author still doesn't have a machine with an example bridge), they turned out to be the most standard and the generic routines easily supported them. Toshiba's ToPIC family has proven to be the hardest to support (as well as moderately difficult for the author to afford for laptops with the newer members of the ToPIC family).

6.4 3.3V 16-bit PC Cards

The "YENTA" specification spells out a uniform standard to supply voltages to PC Cards. This standard replaced the multitude of pre-existing methods to control card voltage. Prior to "YENTA" there were at least 5 different 3.3V standards: Two from Cirrus Logic, one from Intel, one from Ricoh, and one from Vadem. Since the original PCIC hardware didn't support 3.3V cards, each of these methods extended the original ExCA register set in different, incompatible ways.

One would like to use the "YENTA" interface to configure card voltages. However, many PCI bridges take using this interface to mean that the whole "YENTA" interface will be used to program the card, and where it replaces functionality of the old ExCA register set, that functionality will no longer work. The TI based chip-sets were found to break badly when using only the power interface.

This area is one area that would greatly benefit from further study. Debugging of the code which attempted to implement the "YENTA" power API was unable to turn up the cause of the difficulties. The author failed to make it work, and believes this would be a fruitful area of exploration.

6.5 Those Pesky Legacy Laptops

Before the "YENTA" specification was widely implemented, both Intel and Cirrus Logic produced PCI-PCMCIA bridges that looked more like a PCIC on PCI than "YENTA". PCIC is the name for the original Intel ISA to PCMCIA bridge chip (the 82365). These chip-sets required greater knowledge of how the interrupts are connected to the chip-set than "YENTA" chip-sets.

Cirrus Logic produced one PCI chip that followed this form. The CL-PD6729 glued a PCIC interface on the PCI bus. It used the same I/O index+data registers that the PCIC chip used. It used the Cirrus Logic variant of the ExCA register set, and could either be wired to deliver interrupts on the PCI bus, or wired to deliver interrupts on the ISA bus. There was no easy way for the driver writer to know which way the bridge was physically wired. The author assumed it was ISA, with the intention to make it possible for the user to specify PCI at a later date. This curiosity would have remained an obscure footnote to the computer industry had it not been extremely popular in Pentium 120, 133 and 150 based laptops. Surprisingly, many of these laptops are still in service today.

Intel also produced a similar PCI chip. The 82092AA implemented what its data-sheet calls the PPEC register set. This part has a PCMCIA bridge, as well as an IDE bridge on it. Only evaluation boards were ever made of this device. However, a number of clone makers cloned or licensed this design, and it appears in a few older laptops. Fortunately, its interface is similar enough to the CL-PD6729 that the same code works for its clones as well (and presumably on it, but the author has been unable to locate an existing PCI add-in card or laptop that uses this chip). The PPEC register set differs only in how it implements 3.3V extensions.

6.6 The Newest Laptops

The very newest laptops surprising showed a number of problems. The quality of their PCI BIOS seems to have started to vary widely. The older laptops tended to have good PCI BIOS support due to its mandate in the *PC 97 Design Guide*[PC97]. Older laptops with first generation "YENTA" parts on them had problems with their PCI BIOS implementation, which isn't surprising. It did surprise the author the number of new laptops that have this problem.

Months after the code was committed to "current," ACPI support was added to the tree. One of the many things ACPI does is to route PCI interrupts. The newer laptops that had been having problems using PCI BIOS to route the interrupts suddenly started working when ACPI was used to route the same interrupt. ACPI support has been mandated since the *PC 98 Design Guide*[PC98], so it is not surprising that this code path is tested more by systems integrators than the non-ACPI code path that the PCI BIOS takes. It is believed that the latest versions of Windows use ACPI more than PCI BIOS, so vendors are less likely to discover problems in the PCI BIOS API than they are in the ACPI.

7 Remaining problems

Three problems remain in the code. Most Toshiba ToPIC devices do not work when set to CardBus mode in the BIOS. Some systems will hang on the reboot process due to an interrupt storm. A small number of systems will hang on card insertion events, due to an interrupt storm. The author is aware of these problems, and is attempting to acquire hardware or access to hardware that exhibits these problems. Users experiencing other problems are invited to write to the author, or to the **mobile@freebsd.org** mailing list.

8 Development Model Weakness

Although not related to the hardware, or an existing flaw in the software, one final problem should be noted. FreeBSD's development model, which normally operates effectively failed to catch major problems in the PCI implementation before it was released into the stable tree. Fortunately, the process tends to correct itself, and these flaws didn't appear in 4.4-RELEASE.

FreeBSD keeps its source code in a CVS[CVS] tree. FreeBSD branches major releases in this tree, and then starts on its next major release. The branched trees are called "stable branches" because they aren't supposed to contain fully tested code. The main development branch is called "current" and contains code that might not be finished yet. Code is committed first to the "current" branch (yes, although it is technically the trunk, people often call it a branch). There people test it, fix problems with it and allow it to solidify. Once the code has been in "current" for a while, and appears ready for general users, and it is functionality needed in a prior branch, it is merged to "stable." In theory, this development mode ensures that no bad code is merged into "stable" since it has undergone testing in "current" first.

The author committed all of the changes to implement the functionality described in this paper to FreeBSD's "current" branch. He then prepared patches against "stable" that included his changes, as well as fairly extensive changes to the PCI layer to use PCI BIOS more aggressively. These changes had been in "current" for almost a year at this point. The original author of them had no knowledge of any major problems, except with a few off brand motherboards. The patches were posted to several mailing lists, and while problems were found in the PC Card code, none were reported in the PCI merge I was doing.

Within 48 hours of my committing the highly tested patch set (which itself was tested in current for a long time), I had to back out a large part of the PCI patch. It turns out that a number of PCI BIOS implementations do not report all the PCI devices on a bus; filters the config space in unexpected ways; and sometimes would hang the system unexpectedly. The extent and magnitude of these problems took the author of the PCI code and myself by surprise. Those two days showed that the convergence testing that we thought had been happening in "current" actually was minimal and insufficient for finding these problems.

Within a week of the merge into "stable," about 15 different bugs were reported (and mostly fixed) in the PC Card code I had merged. Many of these bugs were people using less popular PCI-PC Card bridges with the new code. Some were configuration errors, or bad configurations that the FreeBSD PC Card layer could have detected but didn't. While most of these problems were easy to work through, some exist to this day. As large a user base that I had testing these patches before I did the merge was still none the less insufficient to ensure high quality for everyone.

While not directly related to the hardware quirks of using PCI interrupt routing, it does show that code dealing with both the generic PCI bus as well as the PC Card bus must be tested on a huge range of hardware, and be prepared to cope with a larger than expected number of unconforming BIOS and hardware implementations. The number of problem machines, and the nature of their problems surprised the author, who has been committing kernel code to FreeBSD for several years.

9 Acknowledgments

The author would like to thank Monzoon Networking, LLC for funding the development of this software. The author would also like to thank the dozens of testers that helped him debug preliminary versions of this software.

10 Availability

The software described in this paper have been integrated into the FreeBSD 4.4-RELEASE. FreeBSD is available free of charge for download from **http://www.freebsd.org/**.

References

[CVS] **http://www.cvshome.org/**

[FreeBSD] **http://www.freebsd.org/**

[Linux-CS] **http://pcmcia-cs.sourceforge.net/**

[NetBSD] **http://www.netbsd.org/**

[PC97] *PC 97 System Design Guide.* Microsoft Corporation, (1996). **http://www.pcdesguide.org /download/pc97.zip**

[PC98] *PC 98 System Design Guide.* Intel Corporation and Microsoft Corporation, (1997). **http://www.intel.com/design /pc98/draft/pc98.pdf**

[PC99] *PC 99 System Design Guide.* Intel Corporation and Microsoft Corporation, (1998). **http://www.pcdesguide.org /pc99/default.htm**

[PC99a] *PC 99A Addendum.* Intel Corporation and Microsoft Corporation, (1999).

[PC2001] em PC 2001 System Design Guide, A Technical Reference for Designing PCs and Peripherals for the Microsoft Windows Family of Operating Systems. Intel Corporation and Microsoft Corporation, (2000). **http://www.pcdesguide.org /pc2001/default.htm**

[PC Card] *PC Card Standard, Release 7.0*, PCMCIA, (February 1999). **http://www.pcmcia.org**

[PC-98] **http://www.pc98.nec.co.jp/**

[PCI] *PCI Local Bus Specification, Revision 2.2*, PCI Special Interest Group, (December 18, 1998). **http://www.pcisig.com**

[PCI BIOS] *PCI BIOS SPECIFICATION, Revsion 2.1*, PCI Special Interest Group, (August 26, 1994).

[WiFi] **http://www.wi-fi.org/**