Porting NetBSD to JavaStation–NC

Valeriy E. Ushakov
`<uwe@ptc.spbu.ru>`

Abstract

Porting NetBSD to a new platform that has its CPU already supported is simplified by clean interfaces of the NetBSD kernel and a wide range of machine–independent drivers for different buses and devices. This paper summarizes experience in porting NetBSD to the JavaStation–NC and gives an overview of machine–dependent code that was necessary.

1. Introduction

The JavaStation–NC is a network computer class machine built on the microSPARC–IIep processor. The microSPARC–IIep is a sun4m with an integrated PCI controller [3]. This makes it unique amongst 32-bit sparc systems as other sun4c and sun4m models are SBus-based. It is sufficiently similar to sun4m to reuse much of the existing code and sufficiently different to require a non-trivial porting effort. Generic and flexible machine–independent infrastructure and drivers provided by NetBSD were crucial to completing the port in short time despite the author's lack of any previous experience with NetBSD kernel internals.

It is hoped that this experience will be interesting to people who need a modern OS for the device they develop or to enthusiasts who want to port BSD to their favorite gadget. Since NetBSD already supports almost all modern (and not so modern) processors, chances are that most of the hard work, like MMU and cache support, is already done. Most likely there are already machine–independent device drivers for some of the devices found in the target platform, thus further reducing your porting time and costs.

Porting NetBSD to a completely new platform is described in [2] that also outlines key features of NetBSD that contribute to its great portability.

2. The target machine

In late 1990s Sun was aggressively pushing the concept of "Network Computer". It developed several network computer class machines of which only two were more or less widely available in the wild — JavaStation-1, codename "Mr. Coffee", and JavaStation–NC, codename "Krups"[1]. All JavaStations were shipped with JavaOS.

Mr. Coffee is a "chimeric" machine. It is a straight sun4m, except equipped with commodity PS/2 keyboard and mouse. It uses the Sun TCX framebuffer, but the video connector is standard 15-pin VGA D-SUB so that it can be used with any PC monitor.

For Krups, Sun used the microSPARC–IIep processor, where 'e' stands for "embedded" and 'p' stands for "PCI". Being "embedded", the microSPARC–IIep is very low–heat, so Krups has no fan, making it a dead-silent machine. Its integrated PCI controller makes the microSPARC–IIep unique amongst other 32-bit sparc machines, but in all other respects it is a SPARC v8. Krups uses a PCIO chip that provides "Happy Meal" Ethernet and EBus (8-bit peripheral bus). The latter is used to connect PS/2 keyboard and mouse, the serial port, and CS4231 audio.

NetBSD/sparc already had complete support for SPARC v8 MMU and caches, so when the Krups port was started, most of the hard stuff was already there. As far as peripherals are concerned, Krups is sufficiently similar to PCI-based Sun Ultra machines, so the plan was to reuse as much device support code from NetBSD's sparc64 port as possible.

3. Firmware and boot loader

The first thing that was needed was a boot loader. Like Ultra machines Krups uses Open Firmware (OFW). Fortunately, NetBSD/sparc port already borrowed OFW support from the sparc64 port, though there were few minor bugs and missing bits here and there because JavaStations are probably the only 32-bit sparcs with OFW and so the OFW support had just never been tested in the 32-bit sparc port (other 32-bit sparcs use Open Boot PROM (OBP), a predecessor of OFW).

The nasty surprise was that OFW in Krups has many quirks. The worst one was that OFW, in violation of the standard, was located at `f000.0000` — the address at which the NetBSD/sparc kernel expects to be loaded. Nor-

[1]This paper will refer to JavaStations by their codenames for brevity and clarity.

mally, OFW is located in high virtual addresses, so the NetBSD/sparc kernel has a fundamental assumption that it has the space between its own end and the beginning of OFW at its disposal. As a workaround the kernel was relocated to a lower address and memory bootstrap code was tweaked to start its heap past OFW. This is a kludge, but it required changing only few lines in a couple of files and all was ready to proceed with porting, leaving the question of how to properly deal with this situation to a better time. This work was actually done on a Mr. Coffee that also had OFW with this problem, but had the benefit of having a minimal working support already, so it was easy to test the kernel relocation on it with otherwise working kernel.

There were also a few other problems, mostly related to device nodes and their properties. To avoid polluting the kernel with numerous special cases and workarounds for OFW quirks, the boot loader was modified to "patch" the OFW before loading the kernel. Since OFW is a full-fledged Forth environment, this was easily achievable with small pieces of forth code that boot loader passes to OFW to execute.

Linux took a different path. Pete Zaitcev implemented PROLL, a small OBP simulator that provided to the kernel the device tree and a minimal subset of OBP entry points that Linux sparc kernel used. PROLL completely replaces machine's OFW. However having OFW around is quite handy to be able to inspect hardware state interactively, so for the NetBSD port, the case was decided in favor of OFW despite the necessary kludges described above.

4. Overall port plan

When the boot loader is written the next milestone is to mount the root filesystem. There are two popular choices. One possibility is to embed a minimal root filesystem into the kernel, the other is to mount root from NFS server. For both approaches, some sort of console support is required, and for the diskless boot, a network driver is also necessary. The latter approach is often attractive because once the network driver is complete the system will be able to boot multiuser directly with both root and swap on NFS, thus passing two milestones in one leap.

NetBSD/sparc can use firmware for its console input and output. As noted in the previous section OFW support was completed during the work on the boot loader and as that code is shared by the boot loader and the kernel the console was functional even before the kernel can do anything useful. That was very helpful during development of early kernel bootstrap code.

Happy Meal Ethernet at PCI is supported by the `hme`(4) driver, so diskless boot was chosen for the next milestone. Thus the only things missing were the most low level code to deal with interrupts, timers and the like, and the machine–dependent parts of the PCI framework.

5. Low level code

The microSPARC–IIep has totally different system registers to raise software interrupts, report pending interrupts, control system and processor timers, etc. While learning the intricacies of system operation and writing support for those low level things was, perhaps, the most interesting part of the project, it is also the most boring part of the project to describe, so this section will only give a short summary of things done.

While some assembly hacking was required, only three short assembler routines were written. `sparc_interrupt4m` — the interrupt trap handler. `raise()` — the function to raise a software interrupt. `microtime()` — the function that reports current time in μs.

There is, actually, another assembler routine that needs to be written but hasn't been yet — the routine to handle non-maskable interrupts that indicate system malfunction. But it is not necessary for the *normal* system operation after all, so it was postponed to some later time.

Bootstrap code and the initial autoconfiguration process were tweaked to reflect new CPU variant support. Details of mapping between PCI and physical address spaces and interrupt routing were encapsulated in a driver that provided usual `bus_space`(9) interface [1]. Finally, kernel clocks that use system and processor counters were implemented.

6. PCI framework

NetBSD has a machine–independent PCI framework that needs only few typedefs and functions provided by the machine–dependent code[2]. These are usually declared in the port's `<machine/pci_machdep.h>` header file. Please refer to `pci`(9) and `pci_intr`(9) for function signatures.

[2] Section 9 of the NetBSD manual does not (yet) fully document what types and functions must be provided by machine–dependent PCI code. This section is intended to summarize the current situation.

An important type that the port must define is `pci_chipset_tag_t`. It is a chipset tag for the PCI bus. Effectively, it describes a root for a hierarchy of PCI buses. The chipset tag is passed to almost all machine–dependent functions described below.

Since the microSPARC–IIep has an integrated PCI controller there is no need to provide for different possible PCI chipsets, and the chipset tag just carries some private data. But e.g. on Alpha the chipset tag also contains pointers to functions that implement machine–dependent methods for each PCI chipset that can be found in Alpha machines.

6.1. Autoconfiguration

`pci_attach_hook()`
: The hook called right before each pci bus is attached during autoconfiguration.

`pci_bus_maxdevs()`
: Returns a maximum number of devices for the given PCI bus.

`pci_enumerate_bus()`
: Necessary if the port needs some special bus enumeration. For the microSPARC–IIep, it is a macro that just calls machine–independent `pci_enumerate_bus_generic()`.

6.2. Device tags

`pcitag_t`
: Configuration tag describing the location and function of the PCI device. Opaque to the PCI framework. On sparc, the `pcitag_t` is a 64-bit integer that encodes OFW device node for this PCI device and the tuple <*bus*, *device*, *function*> in a form used for PCI configuration accesses.

`pci_make_tag()`
: Construct `pcitag_t` value for bus, device, function.

`pci_decompose_tag()`
: Return bus, device, function for the PCI tag.

6.3. Conf space access

`pci_conf_read()` and `pci_conf_write()` are used to access PCI configuration space. The microSPARC–IIep uses standard mode 1 configuration accesses so implementation of these function is straightforward.

6.4. Interrupt manipulation

`pci_intr_handle_t`
: A handle describing an interrupt source. Opaque to the PCI framework that uses the following functions to manipulate interrupts.

`pci_intr_map()`
: The function takes a pointer to `struct pci_attach_args` and maps it to a `pci_intr_handle_t`.

`pci_intr_string()`
: Returns a string describing interrupt source that the driver can use if it wishes to refer to it in an attach or error message.

`pci_intr_establish()`
: Actually establish the interrupt handler for `pci_intr_handle_t` mapped with `pci_intr_map`. Returns a cookie that can be passed to `pci_intr_disestablish()`.

`pci_intr_disestablish()`
: Disestablish the interrupt handler previously established with `pci_intr_establish()`.

`pci_intr_evcnt()`
: Returns the event counter that is the parent for all interrupt–related counters associated with the given PCI bus hierarchy. Refer to `evcnt`(9) for the description of the NetBSD generic event counter framework.

7. First boot

After the steps outlined in preceding sections were completed, the Krups was able to boot multiuser off the NFS. Of course it lacked a lot of device drivers, most annoying was the lack of driver for the time–of–day clock that in Krups is connected via EBus, but nonetheless the machine was self-hosting at this point.

It took about a month from the beginning of the project to the first boot. However it should be noted that this was author's very first experience with both NetBSD kernel programming and with programming something *that* low–level. A seasoned NetBSD hacker could have probably done it in under a week.

While clean interfaces of the NetBSD kernel that support code portability were crucial in completing the Krups port very quickly, they also allowed this small project to contribute back to the NetBSD more then just yet another platform support. The remainder of this article gives some examples of code that was developed for Krups, but was immediately useful for other platforms.

8. Audio

Device drivers in NetBSD are split into bus–independent code that drives the device and bus–specific attachment code. Bus–independent code uses `bus_space`(9) abstraction layer [1]. While EBus is commonly found in PCI-based Ultra machines and NetBSD/sparc64 has a driver for it, unfortunately the driver can not be used for Krups because OFW properties of EBus bus node and its children are very different and often incomplete in Krups. However it is desirable to share the EBus–specific drivers' attachment code between two sparc ports.

A notable example is `audiocs`(4), a driver for CS4231 audio that is found under both SBus and EBus in sparc and sparc64 machines. At the time the driver supported only SBus and only playback. Some rudimentary EBus support was written for sparc64 but was far from complete. Also some SBus specific code was polluting the machine–independent part of the driver.

The driver was refactored so that bus–specific details are removed from machine–independent code and EBus playback support was completed. At that point it turned out that the driver internal interfaces allow to add capture support almost trivially. This is a good smaller–scale example of how clean interfaces of the NetBSD kernel contribute to its unparalleled portability by greatly simplifying development.

The refactored driver was developed on JavaStations, Mr. Coffee (SBus) and Krups (EBus), and when it was complete the sparc64 port automatically got full CS4231 support as well.

9. Graphic card

The graphic chip in Krups is IGA 1682 from Integraphics Systems (now Tvia). Fortunately, the good folks at Tvia kindly provided technical docs for it. It was decided to use the machine–independent "workstation console" subsystem (`wscons`(9)) for it. The rest of the NetBSD/sparc port doesn't use wscons yet, but wscons is intended as the standard console subsystem, and there was no existing code for the IGA 1682 at the time, so it made sense to write the new driver to support the intended standard. As a side note — compare this to Mr. Coffee, that uses Sun TCX framebuffer for which the driver already exist. For Mr. Coffee it was faster to develop PS/2 keyboard and mouse drivers that conformed to old Sun interfaces. That made Mr. Coffee supported with stock `Xsun`(4) binary.

For writers of framebuffer drivers NetBSD provides generic raster operations (`rasops`(9)) that implement text rendering and unaccelerated blitting. The driver shall implement `wsdisplay`(9) interface so that that upper layers of wscons can attach to it. With completion of the drivers, Krups got a real console.

More recent Integraphics chips, CyberPro2000 series, are also used in several other machines that NetBSD runs on. Matt Thomas provided a Corel Netwinder machine for developing CyberPro support in the driver. It turned out that only minimal extensions were required. The biggest problem was that a complete chip init is required for Netwinder, a task that on Krups is performed by the firmware and so can be skipped in the driver. Details of the chip initialization (on which Integraphics docs are extremely scarce) were mostly learned from the Forth code of Krups firmware.

10. Acknowledgements

Pete Zaitcev, who did the Linux port to the microSPARC–IIep, kindly provided a lot of hints on hardware operation. Matthew Green, Eduardo Horvath and Paul Kranenburg provided valuable insights into obscure corners of the low level sparc code. Eduardo Horvath and Jason Thorpe helpfully clarified details of generic NetBSD kernel interfaces, the PCI subsystem in particular, and patiently replied to numerous questions. Martin Husemann has done a lot of testing and debugging for Krups in general and for `audiocs`(4) on sparc64 as well.

References

[1] Chris Demetriou. `bus_space`(9) manual page. Originally in NetBSD 1.3, 1997.

[2] Frank van der Linden. Porting NetBSD to the AMD x86-64: a case study in OS portability. In *Proceedings of the BSDCon 2002 Conference*. Usenix, 2002.

[3] Sun Microelectronics. *microSPARC™–IIep User's Manual*. Part number #802-7100-01. Sun Microsystems, April 1997.