Recent Plan 9 Work at Bell Labs

Geoff Collyer
geoff@plan9.bell-labs.com

Bell Laboratories
Murray Hill, New Jersey 07974
USA

ABSTRACT

Bell Labs recently ported Plan 9 to new systems with non-PC architectures. This is a status report on those ports and a summary of what helped, what didn't and what could.

1. Introduction

In my group of Plan 9 developers and users at Bell Labs in Murray Hill during the last two years (2008—2010) or so, much of our work has been based on ports of Plan 9 to machines other than the IBM PC. This talk will describe the ports, what we use them (or plan to use them) for, and what we learned.

As usual, the bulk of Plan 9 code is sufficiently portable that only the kernel needs to be moved to a new system, and even that only involves populating a subdirectory of `/sys/src/9` and possibly creating a bootstrap program in `/sys/src/boot`. A port to a system for which we have no compiler and no similar kernel would require substantially more work, as described in reference [6].

2. Non-PC Ports: Common Problems and Techniques

We ported Plan 9 to PowerPC 405, 440 and 450 machines and to ARM 926 and Cortex-A8 machines. They all share certain classes of obstacles.

2.1. Memory Barriers and Caches

A fairly common memory architecture is a that *store* instructions queue data in a *write buffer* in the processor, which is gradually drained to a first-level cache, then to a larger but slower second-level cache, and finally to DRAM. The IBM PC model provides cache-coherent memory, so that we rarely have to worry about cache maintenance in the PC port. Cache-coherent memory largely hides the speed tricks pulled by modern processors, which include aggressive caching through multiple levels of cache, and out-of-order (or delayed) memory stores and instruction execution. Even on the PC, we still occasionally have to execute data or instruction barrier instructions by calling *coherence* to ensure that previous instructions and memory stores have completed; for example, in the implementation of locks (to accelerate notification of other processors trying to acquire the lock), and before or after DMA transfers.

By contrast, the PowerPC has long exposed the above processor tricks to the programmer, and ARM processors are now catching up (alas). So in the ports that I have done,

partly out of paranoia, I was fairly heavy-handed in the use of barrier instructions to avoid surprises. Now that the ports are up and running, some of those barriers are being removed and *kprof* has been helpful in finding hot spots due to barriers.

The full sequence of operations needed to guarantee that a write to memory address *addr* has completed (or at least is visible to all peripherals and other processors) in these systems is:

- an instruction barrier, to force the store instruction to complete
- a data barrier, to flush the data from the processor's memory write buffer to level 1 (L1) data cache (or memory if uncached)
- flush *addr*'s level 1 data cache line to the level 2 (L2) cache, if any
- flush *addr*'s level 2 data or unified cache line to RAM or further cache levels

This sequence could have to be extended in future to additional levels of cache. (The ARM v7 architecture cache control registers permit eight levels of cache!) The full sequence is only needed for cached (actually write-back cached) memory, and device registers are mapped as uncached, so they only require the barrier instructions to ensure that stores have completed. Cache lines are often 32 or 64 bytes long.

In a few cases, for example after DMA input, it's necessary to invalidate cache lines rather than flushing them.

Whenever possible, we've configured caches to be write-back for maximum speed.

We imported USB code from the PC port to the ARM ports and had to first add barriers and cache flushing to it. The USB in-memory data structures maintained partly by the USB drivers and partly by the host controllers are currently allocated from uncached memory because managing the caches in the presence of such mixed data structures was too hard: consider a data structure, the first part maintained by hardware, the second maintained by software, and the boundary between them is not a cache-line boundary, yet the data structure is required by hardware to be aligned to a power of 2.

2.2. Kernel Memory Mapping

In the Virtex and ARM ports, the maximum physical memory was 512MB or less (though not always starting at physical address zero), so by mapping the kernel base (KZERO) to virtual addresses `0x80000000` (PowerPC), `0xC0000000` (TI OMAP35 system-on-a-chip (SoC)) or `0x60000000` (Marvell Kirkwood SoC), we are able to address all of physical memory from the kernel using trivial versions of *kmap* and *kunmap*, unlike the PC kernel, which is mapped at virtual `0xF0000000` yet needs to be able to address up to 3.75 GB of memory using kernel virtual addresses (those above KZERO). The different bases were chosen to permit various devices to be identity-mapped for simplicity.

2.3. Kernel Debugging

The Virtex and IGEPv2 boards include a normal Intel-8250-compatible serial port and a few LEDs under software control, which simplifies initial porting and debugging. The ARM boards other than IGEPv2 have combined JTAG and serial ports with special cables with a USB connector at the other end. `usb/serial` now knows how to drive these ports and there are MS Windows drivers too.

On the ARM ports we have enabled hardware watchdog timers so that if the kernel goes off the rails, the system will reset itself, thus returning control to U-boot. (*Das U-boot* is a 'universal' open-source boot loader derived from Linux and often provided

as the stock boot loader on non-PC systems.) There is similar code in the Virtex ports, but it's currently turned off because resetting the Virtex boards is less useful.

2.4. ARM Kernel Initialisation

U-boot just lays the kernel's image down in memory, without regard for segment boundaries, so the ARM kernels relocate their data segments to the expected physical addresses and then zero their BSS segments.

We use U-boot to load `/cfg/pxe/$ether` into memory before loading the kernel, which parses the in-memory copy to simulate what *9load* would have done on the PC.

U-boot configures the processor and various SoC devices before loading the kernel, which is a help because there are many, many figurative dials and knobs (thousands of pages worth) that can be tweaked and sometimes must be just to get basic functionality. Our kernels make an effort to configure the hardware that they use, but probably don't do the full job. Time is finite but hardware configuration registers and pad and gpio configurations are not (or so it seems).

2.5. Floating Point and Other Emulation

The Xilinx Virtex and Marvell Kirkwood boards have no floating-point units, so the kernel traps and emulates floating-point instructions. This is necessary because floating-point is used in *awk*, for example.

The ARM ports will also emulate *ldrex*, *strex* and locally-invented *cas* instructions if the processor traps them as illegal instructions.

2.6. Math Debugging

We used a port of W. M. Kahan's *paranoia* to watch for errors in floating-point emulation. A new program to verify *vlong* arithmetic found a code-generation bug in *qc* which has since been fixed.

3. PowerPC Ports

These PowerPC ports are all CPU server kernels.

3.1. Blue Gene Ports

This work is being funded by the U.S. Dept. of Energy (DoE) under the project name 'HARE'* and the main participants are Bell Labs, IBM Research, Sandia National Labs and Vitanuova. The intent is to provide full operating system capabilities via comparatively lightweight Plan 9 rather than heavyweight Linux or even more primitive operating systems on very large-scale parallel machines.

The Blue Gene machines are many-core PowerPC systems from IBM with multiple cores sharing memory on a single board, and many boards in a complete system, connected via Ethernet and several types of networks developed by IBM specifically for the Blue Gene machines. Processor clock rates are typically modest (e.g., under 1GHz) to reduce power consumption and heat dissipation. Floating-point instructions differ from the usual PowerPC instructions. The complete system can be partitioned into independent clusters of power-of-2 boards. All the cores of a single cluster are

* Portions of this project supported by the Department of Energy under Award Number DE-FG02-08ER25847.

initially loaded via JTAG with the same kernel image and all started synchronously at once.

The systems we have developed on are at Argonne National Lab. The environment is very much geared to batch processing, so interactive development is a bit odd, and our sessions are limited to at most an hour of elapsed time on the development machine.

An initial port to the BG/L (with 64K PowerPC 440 cores) was done, but current work is on the larger BG/P (with 160K PowerPC 450 cores) and will probably move eventually to the still larger BG/Q (with at least 750,000 PowerPC 460(?) cores) being built now by IBM. Drivers for the various IBM networks have been written. We've done some work on more aggressive 9P caching of infrequently-changed files with user-mode servers; the details are reported elsewhere at this workshop.[5] IBM has granted permission to export a snapshot of the BG/P Plan 9 port, and it can be extracted with one of these commands:

```
hg clone http://bitbucket.org/ericvh/hare
git clone http://git.anl-external.org/plan9.git
```

Note that this is not a stock Plan 9 kernel but is an internal development kernel variant named *9k*.

3.2. Virtex 4 and 5 Ports

The Xilinx Virtex 4 FX and 5 FXT evaluation boards are FPGA-based systems with PowerPC processors (300MHz 405D5X2[12] and 400MHz dual-issue 440X5,[13,7] respectively). They have L1 caches but no L2 caches. We did these ports to demonstrate the feasibility of an idea for a hardware device to be used in embedded systems. The two evaluation boards are similar and the Virtex 5 port is a minor variant of the Virtex 4 port and uses many of its source files. It would be awkward to combine them into a single port because the low-level details of traps and memory management differ between the PowerPC 405 and 440.

The PowerPC 405D5X2 errata list was long and worrying but we were able to work around most of the hardware bugs. The 440X5 had a much shorter and less worrisome errata list.

3.2.1. Virtex Bootstrapping

An FPGA is a Field-Programmable Gate Array. A giant Xilinx CAD tool crunches through a VHDL definition of your specific 'hardware' design for 30—60 minutes, downloads the generated bit-stream, which includes the static RAM contents, into the FPGA, and starts it running by taking the PowerPC(s) out of reset. The normal PowerPC restart sequence sets the PC to the last word of memory (static RAM in this case) and begins execution.

Thus we need to include a stripped-down *9load* bootstrap program in the static RAM contents (128K bytes) of the bit-stream loaded into the FPGA. *9load* loads a Virtex kernel, `/power/9vt[45]cpu`, over Ethernet via BOOTP and TFTP and jumps to it. Since bootstrapping via downloaded bit-stream and *9load* is relatively slow and difficult to trigger remotely, `/dev/reboot` and `#ec` were implemented and debugged early, which allowed us to reboot at will, even remotely. We carried this forward into the ARM ports too, though U-boot is more tractable than the Virtex boot process.

The Virtex 5 can be configured to have two PowerPC cores, but its caches are per-processor and not maintained coherently, so coordinating the two cores is painful and

our existing symmetric multiprocessing code isn't sufficient. If configured for two cores, Plan 9 currently puts the second one to sleep. Ideally, it would be nice to use the second core to run the rather dumb Ethernet controller.

3.2.2. Virtex Performance

These PowerPCs have only 64 translation lookaside buffer (TLB) entries in their memory management units (MMUs), and that number is dropping fast in newer PowerPC designs, so we now round segments on the `power` architecture up to 1MB multiples, which will allow us to use pages larger than 4K bytes and thus take fewer page faults and need to reload the TLBs (in software) less often.

Both boards use a complicated combination of DMA engines, hardware FIFOs (byte or word queues) and Ethernet controller, which just doesn't go very fast. Sometimes the DMA engines aren't much faster than copying the bytes with *memmove*, and using them complicates the Ethernet driver. We do use the DMA engines in the Virtex ports. These ports are noticeably slower than the Marvell Kirkwood port and not just because the processors are slower. We consistently used as our benchmark building the Kirkwood kernel on quiescent diskless machines with their root file systems on our main file server:

```
cd /sys/src/9/kw
# 2nd mk will have file system caches loaded
{ mk clean; mk; mk clean; time mk } >/dev/null
```

(Note that the *mkfile* links the kernel twice: once with symbols and once without.) On the Virtex 5 over gigabit Ethernet, this reports:

```
44.32u 35.60s 99.80r      mk      # virtex 5 gb
```

Small level 1 caches and the lack of a level 2 cache are likely responsible for its overall slowness.

For comparison, this is how long it takes on our main four-core 2.5 GHz Core 2 Xeon PC CPU server using gigabit Ethernet:

```
2.26u 2.46s 4.01r         mk      # quad xeon pc gb
```

on a one-core 2.2 GHz AMD K8 PC terminal using gigabit Ethernet:

```
2.95u 2.34s 8.31r         mk      # amd k8 pc
```

and a 500 MHz Soekris net5501-70 AMD Geode LX 586 PC CPU server using 100Mb/s Ethernet:

```
20.78u 11.42s 49.48r      mk      # 586 pc 100mb
```

3.2.3. Virtex Port Availability and Desirability

Anyone who wants one of the Virtex ports should mail me. Note too that new Virtex FPGA designs (e.g., Virtex 6) use ARM processors rather than PowerPC processors.

4. ARM Ports

These ARM ports are all CPU server kernels so far, though there is work in progress to convert some of the ARM systems into terminals too.

The ARM systems that we ported to all come with *U-boot* as their boot loader, but each vendor customises it, so each instance of U-boot behaves differently, though most can

boot via PXE (BOOTP and TFTP) eventually. The Beagleboard's U-boot doesn't even allow PXE booting over USB Ethernet, so it typically is booted from an SDIO memory card, which makes testing new kernels painful.

The most useful by-product of the work with ARM Cortex-A8 processors may be that we will be closer to driving Cortex-A9 multi-core processors or other v7 architecture ARM systems when they appear.

4.1. OMAP35: Beagleboard

We started porting Plan 9 to the Beagleboard with the intent of using it as a small, portable terminal that could boot quickly (i.e., without the IBM PC's power-on self-test). The processor is a 500 MHz dual-issue ARM v7-a architecture[3] chip (the Cortex-A8)[4] with L1 and L2 caches. The board costs US$149, though that includes **no cables, no case, and no power supply**. By the time you finish adding peripherals, powered USB hub(s), power supply, and cables, it's not so portable (or cheap). The board includes a serial port and (complicated) video controller. We now have a driver for this video controller, based on a first draft by Per Odlund.

Unfortunately the Texas Instruments (TI) OMAP 3530 SoC[8] that the Beagleboard is built around was designed for cell-phone or similar battery-powered operation and is quite complicated and somewhat buggy. Its biggest defect is that it lacks a built-in Ethernet interface. In principle we could use USB Ethernet (disgusting though it is) but we haven't got the 3530's USB to work yet (and there are some nasty-looking USB-related bugs in the 3530 errata list). As a result, the Beagleboard isn't really usable yet.

The Cortex-A8 processor includes VFPv3 floating-point hardware, but the kernel doesn't yet save and restore FPU registers and other state during a context switch. That's okay for now because *5c* doesn't yet generate VFP op-codes for floating-point instructions, but rather op-codes for the old ARM 7500.

4.1.1. IGEPv2 and Gumstix Overo

Meanwhile, we bought an IGEPv2 board from ISEE in Barcelona. The IGEP is also built on the 3530 SoC and is supposed to be quite similar to the Beagleboard except that it adds some external devices such as 100Mb/s Ethernet via the SMSC 9221.[11] Also, the CPU can be run at 720 MHz rather than 500 MHz, and our port does so. We run the same kernel on the Beagleboard, the IGEPv2 board and the 600 MHz Gumstix Overo Earth, which also provides the SMSC 9221 Ethernet controller but is based on the OMAP 3503 (not 3530) SoC. Unlike the Kirkwood, we haven't got access to the NAND flash memory to work yet; it's not well documented. At least on the IGEP board we have in the Unix Room, plugging in a DVI cable disables the Ethernet controller, so the IGEP may never be usable as a terminal. The Gumstix don't have this problem.

Ordering the IGEP from Spain and getting it through US customs was a challenge; I don't recommend buying one lightly if you are in North America. The Gumstix should be easy to obtain instead.

4.1.2. OMAP35 Performance

Unfortunately, the rudimentary 9221 Ethernet controller is somewhat like the Virtex Ethernet controller, and thus consumes enough system time to make the IGEP fairly slow. Currently we do not use DMA but rather programmed I/O to copy data between main memory and the Ethernet FIFOs, which simplifies the driver compared to using DMA. Running our benchmark on the IGEP and Gumstix over 100Mb/s Ethernet yields:

```
24.58u 30.07s 70.65r      mk      # igep 100mb
28.88u 38.38s 81.67r      mk      # gumstix overo 100mb
```

4.2. Kirkwood: Sheevaplug, Guruplug and OpenRD

The other ARM ports were going slowly and we noticed the early advertising for the Sheevaplug, based on Marvell's Kirkwood SoC (88F6281),[10,9] for US$100. The processor is a 1.2 GHz ARM v5 architecture[1] chip (the ARM 926EJ-S)[2] with L1 and L2 caches. The SoC is more tractable than the OMAP35 SoC, though still somewhat tedious to initialise. The Guruplug is particularly interesting because it has two gigabit Ethernet interfaces (and RJ45 connectors), and thus might make an interesting firewall or other network device, or a server of some kind (e.g., secure store in flash memory), possibly standalone.

All of these systems run the same kernel. We started with the Inferno Kirkwood port, which helped us get going. We have got most of the parts that we care about working, including access to flash memory and even the bizarre cryptographic accelerator.

4.2.1. Kirkwood Performance

The integral Marvell 1116 gigabit Ethernet controller autonomously performs DMA in and out of buffer rings, so the port is fairly zippy and quite usable:

```
16.66u 8.39s 31.44r       mk      # guruplug gb
```

5. The Legendary AMD64 PC Port

One port that we haven't done much work on lately is the *amd64* port. For a start, it's based on a non-stock kernel (*9k*) better suited to 64-bit machines. For example, it uses *varargs* to extract system call arguments. The initial target was AMD K8 systems for DoE work several years ago and the port is a minimal CPU server kernel. The DoE work changed direction and there has been little development of the port since.

It does not run 386 binaries. Some Ethernet controllers (mostly higher-performance ones) and UARTs have drivers, typically imported from the Plan 9 PC port. These things do not yet work: audio, video, USB, floppies, disks, and laptop stuff such as PCMCIA. Some of this will be easy to import but it would be nice to take the opportunity to drop legacy support for old devices and processors when importing. However, video will be painful: the kernel either has to run the VESA BIOS in 16-bit real mode or emulate it.

Using this port also requires *6[cal]* and *libmach amd64* support, and a modified *9load*, all of which exist.

6. Performance Summary

Table 1 summarises the above *time*(1) output lines sorted by user-mode time, slowest first, and adds other system characteristics. Of these systems, only the Virtex 5 lacks an L2 cache. The 'issue' column is the claimed maximum number of instructions that may be issued per clock cycle. Machines above the middle dividing line have Ethernet controllers that don't use buffer rings.

A few observations: the sub-GHz dual-issue machines are the slowest in our sample. The slowest machines all have clumsy DMA/FIFO/Ethernet combos and the fastest all have Ethernet controllers that use buffer rings and initiate DMA autonomously (note the variation in system times).

times	name	cpu MHz	Ethernet	issue
44.32u 35.60s 99.80r	virtex 5	400	1Gb	2
28.88u 38.38s 81.67r	gumstix overo	600	100Mb	2
24.58u 30.07s 70.65r	igep	720	100Mb	2
20.78u 11.42s 49.48r	586 pc	500	100Mb	1
16.66u 8.39s 31.44r	guruplug	1,200	1Gb	1
2.95u 2.34s 8.31r	amd k8 pc	2,200	1Gb	1
2.26u 2.46s 4.01r	quad xeon pc	2,500	1Gb	2

Table 1: Performance Summary

7. Hardware Documentation

Documentation for the Virtex parts was uniformly better than for the ARM parts, largely because IBM wrote a lot of the PowerPC documentation. Xilinx's contributions were less clear, with fine points sometimes ignored. So for the Virtex ports, it was usually possible to suspend disbelief, trust the documentation and get things to work.

7.1. ARM Documentation

The ARM ports are another story. There are at least three reference manuals needed for each port: the processor architecture manual from ARM, the processor manual from ARM (and sometimes another from the SoC vendor), and the SoC manual from Marvell or TI. The latter two types of manuals point to the earlier manual(s) for specifics. For example, the SoC manual will point to the processor manual for details of how the processor does something, and it in turn will point to the corresponding architecture manual to describe a common mechanism. If you're lucky, the board manufacturer will include some board documentation too.

The architecture reference manuals from ARM have been getting better: more precise and complete, less slippery. The CPU and SoC technical reference manuals, however, combine extreme verbosity with imprecision, incompleteness, bugginess and a general lack of rigour that is maddening. There is a tendency to write ''this register is used to do X'', without explaining how or why.

One ends up flipping between the three or four manuals, and the OMAP35 manual alone is a single PDF file of 3,500 pages, so having a good PDF viewer is essential. 3,500 pages should be enough to describe almost anything, yet it's incomplete; quite an accomplishment. The table of contents, figures and tables alone is 160 pages long.

Perhaps separate programmers' reference manuals are needed.

7.2. Poor SoC Design and Documentation

The SoC manuals are particularly poor. The SoCs themselves appear to have been slapped together from existing intellectual property that the companies had lying around, without much thought for how they might be used together, and the manuals reflect this: each chapter describes a separate standalone device. The manuals appear to be written for other hardware designers or implementors and make many assumptions about background knowledge. There is little coherent explanation of why this sack of components was slung together or what the interconnections are or how you might need to use them to do something, such as access the flash memory or drive the video or Ethernet controller.

There is a great deal of low-level explanation of the tedious games of 'Mother-may-I'

that one must play to even get the hardware to function. For example, clocks have to be enabled, because they aren't always already enabled. Until they are enabled, accessing some device registers causes address faults. Also, various parts of the SoCs have to have their power turned on. At least one SoC has internal 'firewalls' that have to be disabled or subverted. In general, the hardware doesn't come up in a sane state and U-boot doesn't always fix that (though it does leave some things in a sane state). Failing that, the documentation ought to contain a concise and complete list of exactly what steps are required to initialise a particular device.

7.3. Marvell Documentation

A drawback of using the Kirkwood SoC is that Marvell seem to regard almost all of their manuals as proprietary. Given their low quality, Marvell may just be trying to reduce their embarrassment. Alcatel-Lucent has a non-disclosure agreement with Marvell, but even with that, we are not able to get all the documentation we need. Contrary to Marvell's belief, Linux (or U-boot) drivers are not a substitute for good hardware documentation (nor for understanding!), but we have sometimes had to resort to reading them.

7.4. Board Documentation

Global Technologies fabricated the Sheevaplug, Guruplug, etc. ISEE in Barcelona made the IGEPv2 board, and the Beagleboard was an open-source hardware project. The Beagleboard documentation is pretty thorough. The IGEPv2 documentation is less thorough, specifically when covering access to flash memory (which differs from the Beagleboard's) and the devices added to the stock Beagleboard, notably Ethernet. Global Technologies have done a pretty good job of documenting the plugs, though the Sheevaplug is better covered than the Guruplug.

8. Lessons Learned and Recommendations

8.1. Kernel Debugging

Effort to connect the various JTAG interfaces (via USB) to *db* or *acid* would likely pay off when the processor mysteriously goes away. Even just having U-boot print the processor's state at the instant that U-boot regained control would help.

Debugging can be assisted by having a dead-simple variant of *iprint* that will always work, including before the interrupt controller is initialised, in interrupt-service routines, and when interrupts are masked.

A little debugging code in `/sys/src/9/port/portclock.c` can diagnose infinite loops caused by incorrect implementation of the clock primitives. An added test in `/sys/src/9/port/xalloc.c` can detect a cycle in the `Hole` list and avoid an infinite loop.

8.2. Hardware Sanity (or Lack Thereof)

There is a great market opportunity for someone to build a (cheap) sensible general-purpose ARM system with good documentation, since there appear to be none now.

Hardware cache coherency helps correctness and performance, especially in multiprocessor systems.

The performance of an Ethernet controller (as dictated by its design) can determine the usability of a Plan 9 port. We should be past the era of programmed I/O and FIFOs and

separate DMA engines, and kludges like Ethernet via USB, for Ethernet controllers. All future Ethernet controllers should perform DMA to and from buffer rings without needing host intervention to initiate each packet transfer. It's not necessary to have all the features of the Marvell 1116, which goes somewhat overboard, though. Also, gigabit Ethernet is pretty cheap now and ought to be standard.

Hardware should come out of reset in a sane and usable state.

Hardware documentation should include system programmers as part of the intended audience, and should be public, complete, clear and concise. GPIO (general-purpose I/O) pins and similar ill-defined bits, and their uses, need to be documented in great detail at the CPU, SoC and board levels to avoid the finger-pointing so often seen in existing manuals, which makes it maddeningly difficult to pin down exactly what must be done to get some device to work. In general, comments like 'see your board documentation for specifics' are unhelpful unless the board documentation actually does describe simply and directly what's needed. Too often these pointers are just cop-outs, with no one taking responsibility for getting all the necessary information published. The sort of treasure hunt that we undertook to determine the IRQ number and memory address for the I/O registers of the IGEP's 9221 Ethernet controller should not have to be repeated.

9. Acknowledgements

Salva Peiró and Mechiel Lukkien ported native Inferno to the Kirkwood SoC.

Rae McLellan helped me decrypt voluminous yet often unhelpful hardware documentation. He and Claudia Collyer provided useful comments on drafts of this paper.

These ports were made feasible by the existing Plan 9 PowerPC and ARM compiler suites, and made easier by existing Plan 9 kernels for other PowerPC and ARM systems.

10. References

1. ARM, *ARM Architecture Reference Manual*, DDI 0100I, 2005.
2. ARM, *ARM926EJ-S Technical Reference Manual*, DDI 0198E, 2008.
3. ARM, *ARM Architecture Reference Manual, ARM v7-A and ARM v7-R edition*, DDI 0406B, 2008.
4. ARM, *Cortex-A8 Technical Reference Manual*, DDI 0344J, 2009.
5. Geoff Collyer, Charles Forsyth, ''A Cache to Bash for 9P,'' *Fifth International Workshop on Plan 9*, Seattle (October 2010).
6. Bob Flandrena, ''Adding Application Support for a New Architecture in Plan 9,'' in *Plan 9 Programmer's Manual, Fourth Edition* (April 2002).
7. IBM, *PPC440x5 CPU Core User's Manual, Preliminary*, SA14-2613-03, July 15, 2003.
8. Texas Instruments, *OMAP35x Applications Processor Technical Reference Manual*, SPRUF98D, October 2009.
9. Marvell, *Unified Layer 2 (L2) Cache for Sheeva™ CPU Cores Addendum*, MV-S104858-00, Rev. B, September 28, 2008.
10. Marvell, *88F6180, 88F6190, 88F6192, 88F6280, and 88F6281 Integrated Controller Functional Specifications*, MV-S104860-00, Rev. E, September 17, 2009.

11. SMSC, *High-Performance 16-bit Non-PCI 10/100 Ethernet Controller with Variable Voltage I/O*, LAN9221 datasheet Revision 2.6, 12-04-08.
12. Xilinx, *PowerPC Processor Reference Guide*, UG011 (v1.2), describes 405D5, January 19, 2007.
13. Xilinx, *Embedded Processor Block in Virtex-5 FPGAs Reference Guide*, UG200 (v1.6), describes 440X5, January 20, 2009.