Porting NetBSD to the RISC-V

Zachary McGrew, Philip A. Nelson
Western Washington University Computer Science Department

I. Abstract

While NetBSD runs on 16 different types of CPU architectures, it did not run on the RISC-V. In order to live up to the slogan “Of course it runs NetBSD” the project of completing the port of the NetBSD kernel to the new RISC-V architecture was started. Adapting the kernel to take advantage of the new platform features while still maintaining NetBSD’s portability was challenging, but became and interesting problem to solve. While many issues were discovered in the process, the final outcome of booting a kernel on a new architecture was informative and rewarding.

II. Introduction

NetBSD [1] is an open source Unix-like operating system with roots in both 386BSD [2] and the BSD Net/2 [3] release. NetBSD-1.0 (the second formal release of NetBSD) supported five CPU architectures [4]. Since that release in 1994 it has been ported to over 16 different CPU architectures [5], which has lead to the slogan of “Of course it runs NetBSD” [1]. It is this slogan that advanced NetBSD on many platforms that may not even be considered as a computer, such as the Dreamcast [6] video game console, or a toaster [7]. One of NetBSD’s original focuses was architecture independence [8], and this has made it an excellent choice for porting to new architectures. To continue on the tradition of running NetBSD on as many different architectures as possible, many people have ported NetBSD to most architectures that are capable of supporting it. When a new open source architecture that is capable of running NetBSD became available it became only a matter of time until a port to that architecture would take place.

The RISC-V [9] architecture is a free and open instruction set architecture (ISA). The original design came from the University of California, Berkeley for research and educational purposes [10]. Currently under the control of the RISC-V Foundation, it is comprised of two specifications: The User Level ISA Specification [11], and the Privileged ISA Specification [12]. Designed as an extensible architecture, the User Level ISA specification defines a set of extensions that may be implemented. Some extension examples are: the base integer extension (I), the multiplication and division extension (M), the atomic extension (A), the single-precision floating-point extension (F), and the double-precision floating-point extension (D). These extensions are what the User Level ISA collectively refers to as the general-purpose ISA, known as G. The basic set of operations on integers are defined as an extension, because they are not required to be present. Instead, an embedded form (E) of these basic instructions may be used instead, but this also presents a smaller number of registers. Even things that many would consider to be standard on a processor such as multiply and divide are an optional extension, as this allows for flexibility when creating chips for embedded systems. The general extensions that define what would commonly be found on a desktop-class or server-class processor are bundled into the general extension (G), which encompasses I, M, A, F, and D. Additionally the Privileged ISA specification defines the machine (M), hypervisor (H), supervisor (S), and usermode (U) specifications for operating modes. The S and U modes allow for the Unix model of security, separating the kernel mode from the user mode, to be enforced in hardware. The RISC-V comes in four varieties: RV32I, RV32E, RV64I, and RV128I, with the RV128I specification not completed as of the time of this paper.

This project was a Master’s project for the Computer Science Department at Western Washington University. It aimed to complete the port of the NetBSD kernel to the RISC-V architecture. In particular, the port targeted the RV64GSU, or the expanded form known as RV64IMAFDSU, variety of the RISC-V. While there are many more extensions that are defined in the User Level ISA specification, they are not needed for NetBSD to function, though NetBSD could be extended to use them in the future as they become available. The current User Level ISA specification 2.2 defines both 32-bit and 64-bit modes, the 64-bit mode was selected for the NetBSD kernel port as the initial target, though there is a possibility of supporting both in the future. This project used a single version of both specifications, the User Level ISA specification 2.2 and the Privileged ISA specification 1.10. While these specifications are subject to change in the future, they are not expected to change in a manner that breaks existing code.

The NetBSD port to the RISC-V had already started when this project was proposed. Initially contributed by Matt Thomas in early 2015, the committed code was a good starting point, but was neither compiling nor functionally correct. While some of the bugs addressed in the initial code were truly bugs, many of them weren't. When the port was initially started it was targeting a different User Level ISA specification as well as a different Privileged ISA specification. It remains unclear exactly which previous version of the specifications Matt Thomas was using, but after reading older specifications it appears to be at least a few versions behind for both. While the specifications were mostly compatible, some key components have changed. For example, the bit fields have changed in some of the crucial control and status registers (csr). Some of these settings now live in different bit positions, different registers altogether, or have been removed entrely in favor of another option. One such example of this is in the way that floating point support is disabled. Another large change was the page table entries in the Privileged ISA specification. Both their bit fields as a whole have changed, as well as how the leaf nodes are indicated in the page tables. There were many subtle changes that went unnoticed for extended periods of time during this port. There was even an instruction that was removed from the Privileged ISA specification, but the assembler would still emit the opcode for it. The "`SFENCE.VM`" instruction was replaced with the "`SFENCE.VMA`" instruction. The subtle one character difference is all it took between actually emitting a valid instruction that flushes sections of the TLB and causing an invalid instruction fault. These specification changes caused the majority of the locore.S `start` function to need to be rewritten in order to initialize and activate the page tables needed for virtual memory to function.

III. Simulators, Toolchains, and Build Scripts, Oh My!

When the project started no physical RISC-V hardware existed. In order to run RISC-V binaries a simulator needed to be built for the development system. To the benefit of researchers and developers, the RISC-V Foundation released a simulator supporting both the User Level ISA and Privileged ISA specifications named Spike [13]. However, Spike has both build and runtime dependencies on the "Frontend Server" (fesvr) [14]. Fesvr enables access to the host's resources inside the simulator. Spike and Fesvr were both compiled with the host's native toolchain, as the resulting programs will run directly on the host. Spike does not have many requirements outside of Fesvr and the device-tree-compiler (dtc) [15]. Fortunately the dtc is readily available in NetBSD's Pkgsrc [16], or pre-built on certain platforms. Spike does not currently have a BIOS or UEFI-like [17] system in place to share system details and allow for quick hardware interaction. System details are presented at boot time via a Flattened Device Tree (fdt) [18]. It also relies on a boot loader to load the kernel and provide these hardware interaction services. The Berkeley Boot Loader (BBL) [19] was used when developing the NetBSD port.

BBL loads the kernel at a known address, and begins running code at the pre-defined entry point in the ELF binary that it loads. The BBL was not just built once and given a kernel file to load. Rather it is required to be compiled each time a new kernel was built, in order to embed the kernel inside. Additionally, BBL required a C library in order to compile, as it utilizes the string manipulation functions. This meant that an additional toolchain needed to be built, one just for BBL, as the toolchain that compiled the NetBSD kernel and additional tools needed to not link against or reference a C library and required a special naming scheme.

Once Spike was built, two cross-compiling toolchains were needed. The first was used to build BBL, and the second to build the NetBSD kernel. As NetBSD's in-tree version of GCC [20] was currently too old to support the RISC-V architecture, this dual external toolchain approach was used. The first toolchain was identified by the triplet prefix [21] "riscv64-unknown-elf" and consisted of Binutils and GCC that was built as a two-stage compile. The initial compile of GCC was built with the Newlib [22] flag but no path to Newlib's headers was defined. This was followed by the GNU Newlib version of the C library being compiled using the stage-1 compiler. Finally, GCC was compiled again with the Newlib flag, but was also given the path to the recently built Newlib headers. This toolchain was used just for building BBL.

The second toolchain used the triplet prefix "riscv64--netbsd," which was form that NetBSD was already setup to use. It consisted of Binutils and a single stage build of GCC without a C library. The "riscv64--netbsd" toolchain was used to compile the NetBSD kernel, and to provide additional libraries for the tools that NetBSD build system needed to compile the kernel.

Due to the complexity of building cross-toolchains, a build script was created just to manage compiling the simulator, the boot loader, and the two toolchains. This allowed changes to the configuration options of the various tools to be performed quickly, as many re-configurations were needed before everything functioned as intended. The "with-arch" option was changed significantly before finally becoming "rv64imafd." Again, the "imafd" is what the RISC-V ISA specification refers to as "g," but using "g" as the setting gave problems when code explicitly looked for the expanded form of "imafd." Another problem was encountered with the "c" extension, which generated invalid code in certain circumstances. Rather than debug this compiler issue, the choice was made to move forward without that feature enabled in the compiler.

Having the “riscv64--netbsd” toolchain working was the last step involving source external to NetBSD that was needed in order to build the tools target of NetBSD’s build.sh [23] tool. Build.sh is the tool that is utilized to build the entire NetBSD operating system in a platform agnostic manor. It can be given various configuration flags, and information about targeted hosts. Cross-toolchains can be automatically built and used to compile the entire operating system using this tool. As part of its normal build process build.sh should build the needed cross-toolchain using the in-tree Binutils and GCC, but they were currently too old and lacked support for RISC-V. This required that build.sh be told to use an external toolchain. The EXTERNAL_TOOLCHAIN and TOOLCHAIN_MISSING variables were used to inform it not to build these tools, but this introduced another problem. The dbsym and mdsetimage tools needed to know exactly where the build folder for libbfd and libiberty were. The makefiles that build.sh uses to build those tools were extended in order to work around that issue. The makefiles for both dbsym and mdsetimage were modified to allow the BFDDIR and IBERTYDIR variables to be set externally, which enabled passing them to build.sh. The patch enabling these settings was accepted into the NetBSD tree in December 2017, and is now available for others too use.

With the build.sh tools operation completed, the kernel build itself was attempted. NetBSD had very portable code and tried to conform to strict standards where possible. This could be seen through the fact that it can be compiled on many platforms with various versions both GCC and Clang [24]. However, when this project began NetBSD was utilizing the GCC 5.x release series, which did not include as many warning checking functions as the newer GCC 7.3 compiler that the RISC-V Foundation had added RISC-V support to. Something as critical as an operating system kernel needed to ensure it is correct as possible, so the NetBSD kernel was initially built with the “-Werror” flag. This turned all warnings into errors and prevented the build from progressing. The additional warnings introduced into GCC were beneficial for detecting problems that can occur, but hindered the ability to get further in the build. Enabling the -V NOGCCERROR=”yes” flag for build.sh allowed progress while other NetBSD developers worked on the fallout of having additional warnings enabled in the compiler. This build.sh flag allowed the build to only produce warnings and errors in the sys/arch/riscv directory, which was the intended place to finally start writing and fixing code. These warnings and errors that came from that directory unfortunately meant that the code that had been committed was either incorrect or had bit rotted from not being touched in the years since its addition. The problem would later be determined to be a combination of those issues as well as the RISC-V specification having changed since the initial code was committed to the source tree.

IV. Virtual Memory

The entry point of the kernel is defined in the file kern.ldscript as “ENTRY(start).” The aptly titled symbol `start` function referenced by entry is defined as a label in locore.S. This assembly language file contains almost all of the lower level code used by the port that needs to be written by hand in assembly. The job that the `start` function performs is to enable virtual memory, get to the virtual address where the kernel will run, and do any remaining machine dependent initialization that needed to happen before the machine independent function `main` will be called.

The RISC-V utilizes a standard multilevel page table approach for virtual memory. As of Privileged ISA specification 1.10 there are three specification for page tables: Sv32, Sv39, and Sv48. The “Sv” stands for Supervisor, which is the mode or access level the chip is running the operating system in, while the number suffix is the size of the virtual address space. Because NetBSD RISC-V port targets the RV64I specification as its primary target, the Sv39 system was chosen. While the the 39-bit virtual address (VA) allows for a maximum of 512GB of memory, the larger 48-bit VA space would extend the maximum to 256TB of memory, but also requires one additional page table entry lookup. The 512GB limit seemed to vastly exceed the current systems at the time of this writing, and in the future if need arises it can be switched to Sv48 without too much additional work. This would only require the addition of an extra layer in the page tables, which amounts to just updating the machine dependent helper functions that create, modify, and perform lookups on the page tables.

When the kernel is initially loaded into memory by BBL it is put into one, large, contiguous block of memory. This predetermined information permits the use of the RISC-V’s MMU feature that allows any PTE entry to be a leaf node in the tree. Another way to think of this is that it has support for multiple page sizes. However, the Privileged ISA specification defines only a 4K page size on both the 32-bit and 64-bit systems, and when the page table is marked as the final entry before the last level in the page table it is specifying a contiguous block of 4K pages. The Sv39 spec allows for 1GB and 2MB contiguous blocks of pages. When the start function builds the page tables instead of mapping the kernel on 4KB pages it maps them by 2MB contiguous areas. At the time the system was first successfully booted, a non-debug kernel was slightly larger than 8MB, which means only five entries in the second-level page table need to be created. Unfortunately, the space after the kernel up to the next 2MB boundary is wasted, and is something that will be addressed in future work. In order to create the initial page tables in the start function, only two 4K pages are reserved at compile-time.

The first of the those pages, the L1 page table, has two entries; one for the kernel and one for physical memory. Currently, the kernel is mapped at `0xffffffff00000000`, which makes identifying kernel space easy. Physical memory is mapped starting above the end of the reserved kernel space, which allows for direct access to all physical memory. This technique is in use on many platforms, but was recently removed from some of the key platforms such as i386 and AMD64 due to the Meltdown and Spectre hardware bugs that are present there. However, for development purposes this made doing manual lookups in page tables faster because of the direct access to the entire physical memory. There is a limitation to this direct mapping method, and that is it will require more of the virtual memory space to be occupied that otherwise would not. Additionally, as more memory is mapped in the kernel will need to be mapped starting at a lower address in order to accommodate this larger physical memory space that is mapped above it.

The second of the reserved pages, the L2 page table, is initialized to point at the five contiguous 2MB blocks that hold the kernel. These entries are marked as a leaf node in the page table, even though they are not at the maximum depth. Taking advantage of this hardware feature allowed for creating a single entry to map out a contiguous block of 2MB of memory, instead of an entry per 4KB section. It also saved and additional five 4KB pages from being allocated to hold all those entries.

Once the start function initializes the page tables, the Supervisor Address Translation and Protection (satp) register needs to be set to activate them. This register's job is to point to the physical page number (PPN) of the top level of the page table, define the currently activated address space identifier (ASID) of the current process, and the selected memory mode. The memory mode is set to Sv39, the ASID is set to 0 as this will eventually be referred to as PID 0, and the PPN of the L1 page table that was created is set.

Writing to the satp register will cause a page fault. The machine status register (mstatus) contains a bit labeled Trap Virtual Memory (TVM) to control this behavior, but the operating system is running in supervisor mode, and is unable to override the setting that BBL initialized. As such, the common technique of mapping both the high virtual address and the low physical address to the same L2 entry, which would allow you to continue running at the existing physical address, and then jump to the high address does not work as expected. However, the FreeBSD [25] developers found an impressive workaround to use the fault resulting from the write to the satp register to their advantage. The technique involves first loading the value of the virtual address of a label that immediately follows the write to satp in the Supervisor Trap Vector Base Address Register (stvec), then writing to the satp register. The system will fault to the virtual address and continue running. While this initially seemed less elegant than jumping to the correct virtual address, it does seem faster, as it requires less setup and tear-down for the page table entries. Finally, once the system has faulted to the virtual address the correct address of the true fault handler can be set in stvec for when an actual fault occurs, of which there can be many. At this point in the start function the system has memory mapping enabled, and is running at its virtual address. The remaining hardware dependent initialization can occur, such as bootstrapping the physical memory mapping manager.

V. Pmap Common

The NetBSD pmap(9) [26] man page states that pmap is "machine-dependent portion of the virtual memory system." By definition pmap is machine dependent, but NetBSD tries to reduce machine dependent code to as small of sections as possible. The vast majority of the pmap code among ports performs the same tasks as other platforms and only deviates when it needs to perform a special hardware function. Enter NetBSD's pmap common. Initially written by Matt Thomas in early 2011, pmap common is an implementation of pmap designed to be shared among ports. It was created to simplify the machine dependent code to sections that are as small as possible and reduce common problems that continuously appear, requiring the same fix to be applied throughout the source tree. Examining one of the smaller functions in pmap common, such as pmap_create() this abstraction becomes apparent.

Listing 1 shows in the source for pmap_create() that no matter if the platform has hardware support for page tables (Defined as `PMAP_HWPAGEWALKER`) or not, initialization starts the same. All pmap structures come from a standard pool, have their reference count set to one, their minimum and maximum addresses initialized, etc. Once the common resources of the pmap structure are initialized, a smaller helper function is called to initialize for the machine dependent portions. The RISC-V has hardware based page tables, so the pmap_md_pdetab_init() function will be called. Note that NetBSD refers to page tables as page directories, except for the final entry, which is referred to as a page table.

When this project started, pmap common only functioned on platforms that didn't have hardware page table support. At that time it was in use on MIPS and on

Listing 1. pmap_create()

```
pmap_t
pmap_create(void)
{
  UVMHIST_FUNC(__func__);
     UVMHIST_CALLED(pmaphist);
  PMAP_COUNT(create);

  pmap_t pmap = pool_get(&
     pmap_pmap_pool, PR_WAITOK);
  memset(pmap, 0, PMAP_SIZE);

  KASSERT(pmap->pm_pai[0].pai_link.
     le_prev == NULL);

  pmap->pm_count = 1;
  pmap->pm_minaddr = VM_MIN_ADDRESS;
  pmap->pm_maxaddr =
     VM_MAXUSER_ADDRESS;

#ifndef PMAP_HWPAGEWALKER
  pmap_segtab_init(pmap);
#else
  pmap_md_pdetab_init(pmap);
#endif

#ifdef MULTIPROCESSOR
  kcpuset_create(&pmap->pm_active,
     true);
  kcpuset_create(&pmap->pm_onproc,
     true);
  KASSERT(pmap->pm_active != NULL);
  KASSERT(pmap->pm_onproc != NULL);
#endif

  UVMHIST_LOG(pmaphist, "␣<--␣done␣(
     pmap=%#jx)", (uintptr_t)pmap,
      0, 0, 0);

  return pmap;
}
```

some Power PC ports, both of which directly manipulate their TLBs to achieve the results of virtual memory. This meant that a large amount of work needed to be completed outside of the RISC-V platform specific directory. Working outside of the platform directory requires great care to ensure that it does not negatively impact other platforms. This additional work amounted to not writing the expected pmap common functions, but creating new functions specific to hardware page tables, and then hooking them into the existing pmap common code. Along with pmap common only functioning on non-hardware page table based MMUs at the project's beginning, pmap common was itself in a very incomplete state. It only compiled on the MIPS and PowerPC platforms due to an issue with conditional preprocessor code. The RISC-V pmap common helper functions that were in the source tree at this project's beginning had been written to interface with a version of pmap common that had not been committed into the NetBSD source tree leading to much confusion.

Due to the complexities of pmap, it can be understandable that abstracting it further and ensuring not to break other platforms is not an easy task. The side effect of this is that now that this project has extended pmap common to include hardware page table support, more platforms can be ported to it as well. Pmap common, while a key element of the project, was just one task. Some miscellaneous parts still needed to be worked out before the system could get to forking and running a new process.

VI. Miscellaneous

One required element of the port was the creation of a new console device driver. This driver is currently named "sbicons" for the System Binary Interface (SBI) [27] that it uses. This interface is provided by BBL and allows for the operating system to access it via an environment call (ecall) [28]. This ecall traps to the next privileged mode above what is currently running and optionally passes additional data with the call. In the case of the new sbicons console driver it utilizes two helper functions that wrap these ecalls into the BBL. The `sbi_console_putchar()` function will make an ecall with the SBI_ID set to 1, and the first argument holds the ASCII value of the character to write to the console. The `sbi_console_getchar()` functions similarly, except the SBI_ID is set to 2, and no arguments are passed, instead it returns the value from register a0 to pass back the character value that was read from the console. Utilizing the SBI was a great benefit for developing, in that it allowed for a very simple and quick console device driver to be written. The working console provides for an extra debugging method besides relying on Spike's limited memory dumping and register printing capabilities. In the end, printing to the console was required in order to continue debugging, because Spike is not currently capable of printing the contents of virtual memory.

Another issue that was discovered was context switching with soft interrupts. NetBSD uses the concept of soft interrupts to be able to provide low-priority callbacks. When a soft interrupt triggers the process that was interrupted is "pinned" and the process that needs the callback message is notified. The pinned process does not have its accounting time modified while the other process is notified of the interrupt. When the soft interrupt handler finishes, control is returned to the pinned process. Soft interrupts can be implemented one of two ways: pure software based or hardware accelerated. The existing code had set out to use hardware accelerated soft interrupts, and that path was continued. The hardware accelerated version takes advantage of a context switch to check if a soft interrupt has occurred. If one or more have occurred, the context switch it will use that opportunity to call a soft interrupt before completing the original context switch. The code in `cpu_fast_switchto()` had an issue where it was passing the soft interrupt handler to itself as the pinned process, rather than the previously running process. This lead to an issue where an assertion was triggered due to the pinned process not being in a running state. Additionally the code in the `cpu_fast_switchto_cleanup()` function was directly modifying a mutex, which caused problems elsewhere when the mutex value was verified. Correcting this last issue brought the port to where it currently is: running.

VII. Current Status

In its current state (as of the writing of this paper) NetBSD on the RISC-V is: capable of initializing and utilizing the virtual memory system, has a working console device driver, a working pmap implementation courtesy of an updated pmap common with hardware page table support, is capable of creating and context switching between light weight processes, creating a new process (PID 1), and switching between processes while also passing soft interrupts.

The system will boot, initialize virtual memory, initialize the console, bootstrap pmap, create additional kernel threads, fork and create the first process that will eventually become init. However, the system will prompt for the root file system path to be provided as there is not one available. Unfortunately there is not a way to compile the userland tools and populate a root file system. This lack of userland tools is caused by the use of an external toolchain, which will be addressed in additional work happening in the overall NetBSD project.

VIII. Future Work

As mentioned previously, the version of NetBSD's GCC that lives in the source tree is too old to support the RISC-V architecture. The external toolchain was a valid approach for building and getting the kernel to boot because it does not need to include the C library. However, in order to build the userland tools needed to populate the root file system there are two possible solutions. Either an extensive amount of work to hook deeper into the build system with an external toolchain needs to take place, or the in-tree GCC needs to be updated. At least one NetBSD developer is currently working to import GCC 7.3 into the NetBSD tree, but updating a tool as crucial as the compiler is not a small task. With each update to GCC new features are gained, such as support for additional platforms, like the RISC-V. But each update also brings the possibility of the deprecation of older architectures; Something that can greatly affect an operating system like NetBSD that supports so many of them. It of course also means new bugs and idiosyncrasies to find in both the warnings generated and problems with code generation. The immediate benefit for the RISC-V port is that the userland binaries can be built and a root filesystem can be produced to allow further development and use of the port.

One short term goal is getting the port booting on the SiFive HiFive Unleashed [29] board. Running on real hardware could allow for testing things that are not possible in Spike, such as device drivers and networking.

As Spike supports multiple cores, and the HiFive Unleashed board has four cores, it would be great to get SMP support. Currently only the first core does any work. Additional cores are initialized by BBL, but shortly after control is given to them they each jump into an infinite loop that waits for an interrupt that will never arrive. Enabling SMP will require inter process interrupts (IPI) so that each core can signal to the others.

As mentioned in the virtual memory section the RISC-V has support for big pages. An area of future work and research is the allocation of these big pages once the system has been up and running. As the physical memory gets fragmented it may be difficult to find 512 contiguous 4KB pages that also start on a 2MB boundary. Various ideas of how to track this have been discussed and it could lead to interesting research in the future, as well a nice performance enhancement for applications that perform large allocations.

Yet another area of work could be mapping as much of the kernel as possible on 2MB pages, but use the smaller 4K pages to map out the area after the last full 2MB page. Reclaiming this space could allow for the system to reuse this memory in a meaningful way, instead of it sitting untouched as it currently is.

The current method used for debugging is nothing more than calls to `printf()`. NetBSD has an existing cross-platform in-kernel debugger known as DDB. Getting DDB to work would be a great benefit for debugging, but the current issue is decoding backtraces. The RISC-V hardware has a return address register (ra) that holds the return address of a function call. This leads to the pattern where the return address is only written to the stack in the preamble of the function that is called if that function calls another function. Without a consistent way to walk back up the call stack it is currently unclear how to get a stack trace.

Flattened Device Tree (fdt) support would be a great benefit for running on both actual hardware and on other simulators that do not configure themselves identical to the way that Spike does. FDT support will allow NetBSD to read the device tree information and know the amount of memory that is available and at what physical addresses. The FDT will also contain for information such as hardware clocks, interrupt controllers, etc. FDT support already exists on at least one other supported platform of NetBSD. This platform's code can be used as a working example of how to support FDT on RISC-V.

NetBSD has a great deal of compatibility code in it already and this trend could and should be continued with the RISC-V. As the main focus is RV64 it would be nice to enable compatibility for RV32 on RV64 chips. The AMD64 port can run i386 binaries, and recently the aarch64 port gained compatibility with the arm32 port's binaries. Continuing with this would be a nice added feature if the need ever arises.

Floating point support was present and presumably working for the older specification when the initial code was committed. As was mentioned previously, the specifications have changed and the floating point control registers have been altered. In the port's current state the floating point registers are completely left alone for context switches, which will break floating point support in userland. Enabling floating point and handling these additional registers in context switches should not be an enormous task, it was just a non-critical issue when getting the kernel booting.

IX. Acknowledgments

In no particular order, Zach would like to thank:

- Phil Nelson — For all the guidance he provided, and for the inspiration to even attempt this port
- Aran Clauson — For believing in me enough to purchase a SiFive HiFive Unleashed board for testing
- Matt Thomas — For starting the RISC-V port, and PMAP Common
- Nick Hudson — For sharing the updated but not committed pmap common code, and answering my pmap questions because I was never able to speak with Matt Thomas

X. References

[1] The NetBSD Foundation. *The NetBSD Project*. Oct. 25, 2018. URL: https://netbsd.org (visited on 11/18/2018).

[2] William Frederick Jolitz and Lynne Greer Jolitz. "Porting Unix to the 386: the Basic Kernel". In: (Nov. 1, 1991). URL: http://www.drdobbs.com/open-source/porting-unix-to-the-386-the-basic-kernel/184408655 (visited on 11/18/2018).

[3] Computer Systems Research Group. *Net/2*. June 1991. URL: https://archive.org/details/net.2 (visited on 11/18/2018).

[4] Chris G. Demetriou. *NetBSD 1.0 Release Announcement*. Nov. 8, 1994. URL: https://www.netbsd.org/releases/formal-1.0/NetBSD-1.0.html (visited on 11/18/2018).

[5] The NetBSD Foundation. *Platforms supported by NetBSD*. Nov. 18, 2018. URL: https://wiki.netbsd.org/ports/ (visited on 11/18/2018).

[6] The NetBSD Foundation. *NetBSD/dreamcast*. Aug. 19, 2018. URL: https://wiki.netbsd.org/ports/dreamcast/ (visited on 11/18/2018).

[7] Jesse Off. *NetBSD Toaster Powered by the TS-7200 ARM9 SBC — Technologic Systems Blog*. Nov. 10, 2016. URL: https://www.embeddedarm.com/blog/netbsd-toaster-powered-by-the-ts-7200-arm9-sbc/ (visited on 11/18/2018).

[8] The NetBSD Foundation. *The History of the NetBSD Project*. Sept. 5, 2018. URL: https://www.netbsd.org/about/history.html (visited on 11/18/2018).

[9] RISC-V Foundation. *RISC-V Foundation — Instruction Set Architecture (ISA)*. Nov. 18, 2018. URL: https://riscv.org (visited on 11/18/2018).

[10] Krste Asanović. *Home Page for Krste Asanović*. Sept. 13, 2018. URL: https://people.eecs.berkeley.edu/~krste/ (visited on 11/18/2018).

[11] Andrew Waterman and Krste Asanović, eds. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2*. RISC-V Foundation. May 7, 2017. URL: https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf (visited on 11/18/2018).

[12] Andrew Waterman and Krste Asanović, eds. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*. RISC-V Foundation. May 7, 2017. URL: https://content.riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf (visited on 11/18/2018).

[13] RISC-V Foundation. *Spike, a RISC-V ISA Simulator*. Nov. 18, 2018. URL: https://github.com/riscv/riscv-isa-sim/ (visited on 11/18/2018).

[14] RISC-V Foundation. *RISC-V Frontend Server*. Nov. 18, 2018. URL: https://github.com/riscv/riscv-fesvr (visited on 11/18/2018).

[15] *Index of /pub/software/utils/dtc/*. Nov. 18, 2018. URL: https://mirrors.edge.kernel.org/pub/software/utils/dtc/ (visited on 11/18/2018).

[16] The NetBSD Foundation. *pkgsrc*. Oct. 5, 2018. URL: https://pkgsrc.org/ (visited on 11/18/2018).

[17] UEFI Forum. "The UEFI Primer". In: (Aug. 15, 2014). URL: http://www.uefi.org/sites/default/files/resources/UEFI%20Primer_Aug%2015%202014_Final.pdf (visited on 11/18/2018).

[18] devicetree.org. "Devicetree Specification". Version 0.2. In: (Dec. 20, 2017). URL: https://github.com/devicetree-org/devicetree-specification/releases/download/v0.2/devicetree-specification-v0.2.pdf (visited on 11/18/2018).

[19] RISC-V Foundation. *RISC-V Proxy Kernel*. Nov. 18, 2018. URL: https://github.com/riscv/riscv-pk (visited on 11/18/2018).

[20] The GCC Team. *GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF)*. Nov. 13, 2018. URL: https://gcc.gnu.org/ (visited on 11/18/2018).

[21] Jnc100. *Target Triplet - OSDev Wiki*. Jan. 19, 2017. URL: https://wiki.osdev.org/Target_Triplet (visited on 11/18/2018).

[22] *The Newlib Homepage*. Mar. 17, 2016. URL: https://www.sourceware.org/newlib/ (visited on 11/18/2018).

[23] Luke Mewburn and Matthew Green. "build.sh: Cross-building NetBSD". In: (Aug. 21, 2003). URL: https://www.usenix.org/legacy/publications/library/proceedings/bsdcon03/tech/full_papers/mewburn/mewburn_html/index.html (visited on 11/18/2018).

[24] llvm-admin-team. *Clang C Language Family Frontend for LLVM*. Nov. 18, 2018. URL: https://clang.llvm.org/ (visited on 11/18/2018).

[25] The FreeBSD Project. *The FreeBSD Project*. Nov. 18, 2018. URL: https://netbsd.org (visited on 11/18/2018).

[26] *PMAP(9) Kernel Developer's Manual*. Version NetBSD 8.99.25. Feb. 16, 2012.

[27] RISC-V Foundation. "RISC-V SBI specification". In: (Oct. 17, 2018). URL: https://github.com/riscv/riscv-sbi-doc/blob/master/riscv-sbi.md (visited on 11/18/2018).

[28] Andrew Waterman and Krste Asanović, eds. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*. RISC-V Foundation. May 7, 2017. Chap. 3.2. URL: https://content.riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf (visited on 11/18/2018).

[29] *SiFive - HiFive Unleashed*. Nov. 18, 2018. URL: https://www.sifive.com/boards/hifive-unleashed (visited on 11/18/2018).