

# An Overview of FreeBSD/mips

M. Warner Losh  
*Cisco Systems*  
*Broomfield, CO*  
imp@freebsd.org

## 1 Introduction

Embedded FreeBSD[1] has been expanding its reach over the past few years. FreeBSD now boots on more than just server class x86 machines. While most of the work in this area has only recently been part of the public FreeBSD, the history here is much longer and older. BSD has a long history in the embedded space. This heritage has been reflected in both OpenBSD and NetBSD for some time now. FreeBSD borrows from this rich heritage and adds to it. Companies have been enhancing FreeBSD's platform support to take advantage of the other features in FreeBSD[2].

FreeBSD/mips may have only been committed to the tree this year. However, FreeBSD has been running on the MIPS architecture since the FreeBSD 3.x time frame (1999!). Numerous efforts over the years have been made to get FreeBSD running on MIPS hardware. These efforts don't start from scratch but reuse code from past efforts and other systems. Recently all these efforts have come together, resulting in FreeBSD support for MIPS in 8.0.

The same currents that have caused the FreeBSD/mips port to come together after many years have also been pushing development of other areas important to the embedded world. A number of new devices found only in embedded systems have been added to FreeBSD. Hardware vendors are starting to fund FreeBSD improvement projects (mostly for their hardware). Build system improvements have made it easier to deploy FreeBSD in smaller and smaller footprints. All of these changes have made FreeBSD a more viable embedded platform.

## 2 MIPS overview

The MIPS architecture has been around for a long time. In this time, it has undergone a steady evolution. De-

scribing all these changes in detail is beyond the scope of this paper. A few highlights of the history, however, will aid in understanding some of the issues and challenges with FreeBSD/mips.

In 1984, MIPS Computer Systems Inc. was formed to bring the MIPS architecture developed at Stanford to market. The MIPS architecture started out as a 32-bit architecture, which defined only the user level instructions. The CPU control details necessary for kernel operations were left to the implementor to decide. These early iterations were labeled MIPS I and MIPS II. In 1991, 64-bit extensions were added in MIPS III with the introduction of the R4000. Again, the details of the CPU control were left to the implementor. In 1992, Silicon Graphics Incorporated (SGI) acquired MIPS Computer Systems Inc. after the hoped-for markets for the R4000 failed to materialize. Finally, MIPS IV and MIPS V was defined to include many SIMD operations, and a few other exotic instructions. Neither one of these Instruction Set Architectures (ISA) gained widespread use. Each of these ISA levels was a strict superset of the prior one with no optional parts. After all these were defined, the market crashed almost taking SGI down with it, and the future of the MIPS architecture was in doubt.

While SGI was in the process of failing, other vendors began integrating MIPS cores into Application-Specific Integration Circuits (ASICs). These vendors created a market for the MIPS architecture in embedded systems. They used MIPS cores they had created, which differed from the R4000 in their CPU control details. Software vendors found that the differences difficult to deal with.

In 1998, SGI spin MIPS Technologies Inc. out to raise cash. Learning from the mistakes of the past, MIPS Technologies' architects realized that the strict superseting of the ISAs was a liability. It wasn't possible to pick and choose for a particular application which parts could be implemented. In addition, the differing details of CPU control (especially the TLB/MMU control) had a higher software cost than was saved in hardware im-

plementation. MIPS Technologies defined MIPS32 and MIPS64 ISAs with these lessons in mind.

First, instead of having an all-or-nothing approach, a core set of functionality was defined as mandatory, with additional optional features layered on top via Application-Specific instruction Extensions (ASEs). ASEs allowed an ordered way to expand the ISA for such things as Digital Signal Processing (DSP) operations, vector processing or the MIPS-16e mini-instructions without burdening all implementations with these extra limited-use features.

Second, the CPU control model was defined to increase OS portability between members of the MIPS32 and MIPS64 families. MIPS32 was defined to be all of the instructions in MIPS II, plus a 32-bit MMU that is very similar to the 32-bit section of the R4000's MMU. MIPS64 is a superset of MIPS IV and contains MIPS V instructions as an option. Its MMU is directly from the R4000's MMU, the 32-bit portion of which is compatible with MIPS32.

A few years later, MIPS Technologies revised the ISAs to come up with MIPS32r2 and MIPS64r2. These additions and changes further helped the OS writer by providing single instructions to flush the instruction pipeline, deal with interrupts and otherwise correct many places where the programmer would need a seemingly arbitrary number of NOPs to avoid pipeline hazards<sup>1</sup>. [3]

As far as I know, all ASICs with embedded MIPS cores produced today conform to one of these last four ISAs. The few pseudo-MIPS cores which introduced since 2000 have quickly disappeared and been replaced by real MIPS implementations after becoming legitimate licensees of the MIPS technology.

### 3 FreeBSD/mips History

FreeBSD/mips has a long history. It owes its origins to the MIPS support that was in 4.2BSD for DEC's DECstations, also known as pmax. This original port was updated for subsequent releases and has made its way into both NetBSD and OpenBSD. The original FreeBSD port to MIPS was done around 1999 for FreeBSD 3.x by Juniper's products. An independent effort to bring MIPS

---

<sup>1</sup>NOP instructions are used to clear the pipeline and parallel execution units when memory mappings change. Too many will slow down everything. Too few causes crashes.

support to the FreeBSD community started around the same time frame, but it was never completed and removed 3 years later. So FreeBSD on MIPS remained confined to hardware from Juniper for many years. Efforts started in the Perforce repository in 2002 and again in 2006. These latter efforts resulted in FreeBSD/mips booting multiuser. In parallel, Juniper kept up their port and the two code bases were merged and committed in 2008. Since it has been committed, developers have enhanced the port with new features and platform support.

#### 3.1 FreeBSD 3.x Juniper MIPS

Juniper Networks[4] started life in the late 1990s to build routers to compete with Cisco Systems[5]. Juniper based their routers on FreeBSD and ported FreeBSD to MIPS to support one of their platforms. This port's details aren't known outside of Juniper. They bundled the hardware and software together and marketed their version of FreeBSD under the name JunOS.

Juniper wanted to donate this port to the FreeBSD project, but was unable to complete the process. The port was to a specialized MIPS processor and Non-Disclosure Agreements (NDA) issues prevented a simple release of the sources. In addition, the project desired a publicly available reference platform for FreeBSD/mips. This presented a chicken and egg problem to both Juniper and any would-be port maintainer. After the Internet bubble burst, efforts to contribute the code took a lower priority than survival.

#### 3.2 FreeBSD/mips in CVS 1999

Initially independent of the Juniper efforts, I committed some initial support for MIPS to the tree. I added basic tool chain and libc support. I started porting the kernel, but I never finished that work. When I found out about the possible Juniper work, I stopped working on the port, and never restarted. The code was removed from the tree in 2002.

#### 3.3 Juli Mallett MIPS

Ironically, the removal of the mips code lead to a revival of interest in FreeBSD/mips. Juli Mallett, a FreeBSD developer, ported FreeBSD to R4000-based SGI machines. She created a Perforce project called simply

“mips” in late 2002. The port booted to the single user prompt, but she stopped work before the port was stable enough to run in multi-user mode. Although not ultimately successful, the trail blazed by Juli helped the later mips2 efforts.

### 3.4 Cabal MIPS

Shortly after Juli stopped working on the port, the so-called “Cabal MIPS” effort was started. At the BSDCan 2006 conference, I gave a talk on embedding FreeBSD on arm platforms. Several people at the conference were interested in FreeBSD on MIPS and lobbied several senior kernel developers to work on a port. The “Cabal MIPS” effort tried to independently recreate the MIPS port, not using Juli’s work. These efforts were done in secret to avoid a public failure. However, the secrecy was a mistake and when these kernel developers were busy with other things, the project ground to a halt. The build tools and basic kernel framework were the only things to be completed.

### 3.5 mips2

Wojciech Koszek and Oleksandr Tymoshenko took the “Cabal MIPS” base and added some of Juli Mallett’s code and the “mips2” project was born. The project was named for the path in the Perforce repository where the code resided: `//depot/projects/mips2`. Myself and Ollivier Houchard mentored Wojciech and Oleksander’s efforts as they progressed, providing guidance on how to limit the scope to make it easier to finish the port. This project achieved single user boot in late 2006 on the emulator and multi-user boot on real hardware in early 2007. Cavium Networks took a snapshot of this work published in April 2007 and ported it to their Octeon processors. More on these efforts later. After this early success, the project started losing steam as efforts switched from stability to merging the changes back into the FreeBSD main line of development (-current).

### 3.6 FreeBSD 6.1 Juniper MIPS

Meanwhile, Juniper had forward ported their MIPS code to 4.x and later 6.x. They added support for MIPS ASICs in their products. Once they had a stable port for their real hardware, they produced a stripped down version for standard MIPS hardware with all code covered by

NDA removed. They approached me to review the code in September of 2007, but I lacked the time to do a thorough job. I determined that they ported FreeBSD 6.1 and not -current, and there would be work needed to bring it up to FreeBSD -current. All the years that Juniper had spent hardening its MIPS port might not benefit the FreeBSD community due only to my lack of free time.

### 3.7 Merged mips2 and Juniper MIPS

In December 2007, Cisco Systems hired me to complete the FreeBSD port to MIPS. This effort targeted one of the Cavium Networks Octeon chips. Andre Hedrick, Randall Stewart, Ollivier Houchard and I took code from Cavium’s port, and the “mips2” project and merged the two together. Since both of these code bases were based on recent FreeBSD -current snapshots, we felt that it would be easier to port to FreeBSD -current than Juniper’s 6.1 port. The Cavium base was from April 2006, and the mips2 base was from November 2007. Cavium’s port was from April 2006, while the mips2 branch had last been updated in November. Andre and I worked to forward port Cavium’s changes to the mips2 code base, but we encountered many problems.

In reaction to these difficulties, I took a fresh look at Juniper code. Juniper allowed me to share the code for its port for to and evaluate the best path forward for the FreeBSD project. Juniper’s code was very strong for the core architectural support, but lacked support for any of the System-on-Chip (SoC) ASICs that the mips2 port supported.

We created a merged port called `jnpr-mips` which took the best from both code lines. We took the boot sequence, SoC framework, SoC support and peripheral drivers from the mips2 code. We took the processor and vm support from the Juniper base. Most of the code for `src/sys/mips/mips` and `src/sys/mips/include` came from the Juniper tree, while much of the rest came from the mips2 tree. We merged Cavium’s Octeon support code into this merged code base and was used to validate the results. Once that system was working multiuser, we validated the other systems supported by mips2 and corrected problems.

After the port stabilized, we committed the code to FreeBSD -current in April 2008. The merged system ran well both in simulation and on real Octeon hardware. The MALTA board worked in emulation. The RouterBoard RB/532 board worked. I presented a talk at BSDcan 2008, and based on feedback from the talk I

made a number of tweaks to the port. The Octeon support code has not yet been committed to the tree.

The project owes a large debt of gratitude to Juniper Networks for its code base and persistence in getting its code into the tree. The port is much better for Juniper's efforts.

### 3.8 Post commit updates

Since the merged port was committed, a number of developments have happened relating to FreeBSD/mips. Bruce Simpson has added support for Broadcom MIPS cores. Oleksandr Tymoshenko has started working on Atheros AR71xx/AR91xx support. He's currently debugging the Ethernet driver for this SoC. I'm working to support the Alchemy Au1xxx chip, and the PlatHome OpenMicroServer board. Finally, Cavium Networks has made available to select developers its older mips2-based port. Cavium's port also supports n32 and n64 ABIs, as well as SMP. Finally, other MIPS vendors have contracted with developers to add support for FreeBSD/mips for their ASICs.

## 4 Current FreeBSD/mips Status

FreeBSD/mips runs on a variety of platforms. Support for the o32 ABI is present in the base system, with n32 and n64 ABI support on deck. Currently, the system is single core, but multicore support is present in the Octeon support code. A number of new platforms are being worked. All the normal FreeBSD features are supported in FreeBSD/mips, except dtrace, gdb and ficl support in the boot loader.

### 4.1 MIPS Platform Support

FreeBSD/mips targets the recent generation of MIPS processors. The port targets processors that conform to either revision of the MIPS32 or MIPS64 ISAs. Older processors that were used to create these ISAs may also work but aren't the primary focus of the port at this time. Multicore support will be added with the Cavium integration. In addition to the processor support, code for many MIPS-based system on chips' peripherals are supported. The IDT RC32432, Infineon ADM5120, and Broadcom bcm5365 mips32-based SoCs are supported. Work is underway to add support for the Atheros

AR71xx/AR91xx family, the Alchemy/RMI Au1550, and MIPS64 Cavium Octeon family.

#### 4.1.1 Infineon ADM5120

The Infineon ADM5120 ASIC has a MIPS 4Kc core implementing the MIPS32 ISA. The core runs at 175MHz giving a performance 227 MIPS. Integrated into the ASIC are a switch engine, a 10/100M PHY, an embedded PCI bridge, an embedded USB 1.1 host, and controllers for UART, SDRAM, and Flash. The design is targeted at SOHO/SME Gateway, wireless routers and access points, NAT wired routers, print servers, and VPN gateways. Many boards integrating the ADM5120 are inexpensive.[6]

The platform's reference hardware is the ADM5120-based EdiMax reference boards. The code supports the built-in NIC, Ethernet switch engine and the serial console. Both the PCI and non PCI versions of the ADM5120 are supported. NOR Flash, USB, and I2C support are in progress.

Most of the ADM5120 boards available today are based on the EdiMax reference, so FreeBSD/mips usually works on them. Oleksandr did all his ADM5120 work MicroTik's RouterBoard RB/112 board. Other products are also believed to work. Any MiniPCI card that works in these boards and has a FreeBSD driver will work. The Atheros Wireless driver, ath(4), works well on this board.

#### 4.1.2 IDT RC32432

The IDT Interprise RC32432 ASIC is based on a MIPS 4Kc core, implementing the MIPS32 ISA. The processor runs at speeds from 200MHz to 400MHz depending on the model. The ASIC integrates an Ethernet switch, a 10/100M PHY, an embedded PCI bridge, memory controller, DMA engines, a UART, SPI, and I2C bus. This part target the mid-range of VPNs, wireless APs and routers built-in to the ASIC as well[7].

The MicroTik RouterBoard 532 incorporates an IDT RC32432 ASIC into its design. It provides a NAND memory, a CF interface, and additional Ethernet ports via the VIA/Rhine VT6105 chip set[8][9]. The built-in NIC, UART and PCI bus are supported. The VT6105 is supported by the vr driver. Any MiniPCI card with a FreeBSD driver works. The ath(4) driver works well.

### 4.1.3 Broadcom bcm5365

Broadcom makes a wide variety of ASICs with embedded MIPS processors. These ASICs are used in a wide range of networking applications that range from managed switches to wireless access points. These processors implement the MIPS32 ABI. In addition, they have the typical devices found in embedded targets, including NOR and NAND Flash support, 10/100M or 10/100/1000M Ethernet PHYs, PCI host bridge, SDRAM, and UARTs. Sometimes switch engines, I2C, or SPI interfaces are also provided. The BCM5365 is used primarily in managed switches [10].

The Sentry5 is a family of managed switches[11]. FreeBSD/mips supports the built-in NIC and switch, the UART and SiBus enumeration. The latter is unusual in the embedded world as it allows for different embedded devices to be enumerated in a manner similar to the PCI bus. Typically, embedded devices are known to the OS only through tables that are compiled into the OS based on the exact chip model.

Although initially targeted at the managed switches, FreeBSD/mips' support for the Broadcom MIPS is done in a generic fashion that allows it to work on a wider range of chips from Broadcom. The exact range is difficult to determine due to Broadcom's technical detail disclosure policies. One known issue is that the core in the bcm47xx ASICs has pipeline bugs requiring compiler modifications to work around. Patches for gcc are available, but they apply to a small number of gcc 3.x releases and are difficult to use with more recent gcc 4.x releases. BCM53xx-based ASICs do not have this bug, and should work.

### 4.1.4 Coming Soon

Efforts are underway to support the Atheros AR71xx/AR91xx, the Alchemy Au1xxx, and the Cavium Octeon families of ASICs. In addition, some companies with MIPS-based ASICs are bringing up FreeBSD on their hardware, but do not wish the specifics to be public until they are ready to release. The following describes what each publicly announced porting effort is targeting. Since these efforts are ongoing, only the hardware is described.

The Atheros AR71xx/AR91xx series of ASICs integrate a MIPS 24K core running at speeds ranging from 300 to 680MHz. The core implements a MIPS32r2 ISA and has a number of built-in peripherals for gigabit networking,

USB, audio, UART, Flash and RAM. This device targets the high-end wireless market when paired with Atheros' 802.11a/b/g or 802.11n radio cards or chip sets[12].

The RMI/Alchemy Au1xxx series of ASICs is a network security processor targeted at wireless and wired applications. This includes wireless access points, network-attached storage, or set top boxes. The Alchemy Au1 core is common to all members of the family and implements the MIPS32 ISA with enhancements. The family includes a number of integrated peripherals for 10/100M Ethernet, USB 1.1 (both host and device), sound, security (crypto) as well as the traditional UART, Flash and RAM controllers. The processor runs at between 333MHz and 500MHz[13].

The Cavium Octeon network engines target high-end routers and switches. They implement the MIPS64r2 ISA with the cnMIPS cores tied together on coherent peripheral bus. The engines run from 300 to 800MHz and have between 1 and 16 cores. These chips span the range from 100Mbps to full-duplex 20Gbps and have an extensive set of peripherals to optimize network throughput. In addition, a rich set of cryptographic features is present[14]. The Octeon port will likely drive SMP support in FreeBSD/mips. Cavium networks has donated code for a version of the FreeBSD/mips dating from about April 2007 which supports all Octeon ASICs, SMP, full 64-bit mode operation, additional ABIs (n32 and n64) and building FreeBSD on a Linux host.

## 4.2 Feature Support

The FreeBSD/mips port supports all the normal features in FreeBSD. The FreeBSD Handbook[2] documents FreeBSD, its features, use and configuration. There are a few caveats presently for the MIPS port. Systems running on the MIPS architecture support three ABIs, but FreeBSD/mips supports only one of them. Support for SMP for multicore ASICs is in the works. Linux emulation is not supported. A few programs need to be ported to MIPS still.

### 4.2.1 MIPS ABI Support

The MIPS architecture has a rich history. Unfortunately, this rich history has led to a confusing accumulation of ABIs. On most architectures, there's only one ABI, but in MIPS there are three. The original MIPS ABI was defined by AT&T with its original port to the MIPS pro-

cessor. It is a 32-bit ABI called “o32” or sometimes just “32.” The “o32” ABI mandates a MIPS II ISA, with some caveats.

When SGI shipped the 64-bit R4000-based systems, it invented two new ABIs: “n32” and “n64.” In “n64,” registers, pointers and longs are all 64-bits in length. SGI discovered that this broke many applications, so it created “n32” to bridge the gap between “o32” and “n64.” In “n32,” all registers are still 64 bits, but ints, longs and pointers are 32 bits. The ABI also optimizes stack traffic and alignment, as well as allowing newer MIPS ISAs.[3]

FreeBSD implements “o32.” Support of “n32” and “n64” is in progress. Support for “multilib” is also planned.

#### 4.2.2 SMP Support

SMP support for MIPS hasn’t been the primary focus of the project. Most of the chips supported by FreeBSD/mips have been single core only. For single core, SMP is completely useless, and compiling an SMP kernel adds nothing but overhead. However, many of the high-end embedded network processors have been moving to multi-core for some time now. These processors benefit from an SMP kernel. The Cavium Networks FreeBSD code has support for SMP included in it. While there are some code base differences, as noted above, SMP support is being integrated along with the Octeon support.

#### 4.2.3 Rough Edges

A number of programs in the system require machine-dependent code to function properly. Many of these programs have been updated with MIPS-specific code already, but a few haven’t.

- Debugging with gdb program isn’t supported, although work is in progress to make it work.
- The dtrace feature doesn’t have the MIPS-specific code necessary to make it work.
- The rescue and sysinstall don’t build. Crunchgen creates .o’s that can’t be linked.

Hardware floating point for MIPS isn’t supported on those processors that actually have hardware to do floating point. None of the currently supported processors

have floating point, but such processors exist. The kernel doesn’t implement hardware floating point emulation, so programs compiled for those processors won’t work.

Linux/mips emulation is not present. The Linuxulator in FreeBSD consists of three parts:

1. The machine independent portion, to implement semantics that are common to all Linux architectures.
2. The machine depend portion, to implement machine specific details like system call translation and register calling conventions.
3. A set of libraries from the FreeBSD ports collection which provide applications the shared libraries to run.

The MIPS-specific code has not been written, nor has a set of Linux/mips shared libraries been added to the current Linux emulation port.

Finally, since ports system does not support cross compilation, one needs to have a real or emulated MIPS machine to build ports or packages. There are patches circulating for limited cross compilation support. However, much of the cross compilation support must reside in the actual source packages. Many do not support cross building at all. Since there is no standard for configuring cross compilation, work must be repeated for each package that is cross buildable. There are many logistical issues, such as packages that run natively to build binaries for a target system, that make this problem difficult.

### 4.3 Embedded Device Additions

FreeBSD has traditionally been a server operating system. In the embedded space, the mix of devices and busses is different. With the move into this space, FreeBSD has enhanced its support for these technologies. NOR Flash devices have a standard interface called CFI. SD and SDHC cards are often used when larger storage is required. A number of different serial bus connections are more common in the embedded world. Finally, the USB stack now supports device-mode operation in addition to host-mode operation.

### 4.3.1 NOR Flash

NOR Flash uses a parallel interface to the host. This allows the host to memory map the NOR device into its address space and sometimes execute code directly from the NOR flash. The NOR devices have a standardized control set called Common Flash Interface (cfi). This interface is defined by an industry standards group, but many vendors make proprietary additions to the standard. FreeBSD supports these devices, and usually one only needs to write a tiny amount of glue code to make them work in a new platform. FreeBSD supports reading and writing these devices and will erase pages as necessary to accomplish these tasks.

### 4.3.2 SD Card

An SD Card is a small, postage-stamp-sized card. The current standard defines two classes of cards: SD cards are smaller than 4GB; SDHC cards as 32GB, although minor revisions to the latest standard are expected to push that limit to 2TB. SD Cards are popular in embedded environments because they can be implemented with a very low pin count. FreeBSD fully supports both SD and SDHC cards. The newer miniSD and microSD cards are also support, since they are compatible with either SD or SDHC cards. SD Cards are cheap, readily available and easy to swap in and out of a system. They are good when you need more data than NOR Flash can provide and can afford the expense of having a connector for the flash.

### 4.3.3 Serial Bus Technologies

Serial busses are more common in embedded systems. They keep the pin count down, which reduces cost. I2C is the most common of these technologies. The I2C bus is a two-wire bus that multiplexes the single data line between sender and receiver. It is used for everything from EEPROMs to temperature or environmental sensors. It has many cousins that use I2C as a transport for a higher level of organization. One of those cousins is the SM-Bus, used for environmental sensors. FreeBSD supports I2C and its variations.

Another class of serial bus is those busses where data flows bi-directionally between the master and the slave. The SPI bus falls into this category. Data is clocked bi-directionally between the master and the slave device after the master initiates the transaction. NAND-type flash

is the primary use for the SPI bus, although SD Cards also support being read/written over the SPI bus. Closely related to the SPI bus is the I2S bus, used in audio applications to stream digital audio to a slave chip that converts the stream to analog signals to play over speakers or headphones. Both of these busses are supported in FreeBSD and have a few drivers for each. However, at this time, reading SD cards via the SPI bus is not yet supported.

The final class of serial bus technologies may already be familiar to desktop and server computer users. These are busses that have a hierarchy of sorts. The best known examples of this are Firewire and USB. Firewire, a peer-to-peer bus, has been supported in FreeBSD for some time. It is used in some embedded applications such as MP3 players, but its use is declining. USB is organized in a tree fashion, with USB hubs acting to fan out the single connection to multiple connections for things like printers, scanners, disks, thumb drives, serial ports, and more. As a host, FreeBSD has supported USB for many years. Recently, a new USB stack has been integrated into FreeBSD which will allow it to act as a USB device. USB device mode is also called “USB Gadget support” by some systems. It allows the FreeBSD box to emulate a USB Ethernet connection, a USB TTY connection, etc. This functionality is required for things like printers and scanners that interface to the outside world via USB.

## 4.4 Build System Enhancements

As FreeBSD has evolved from primarily a server and desktop system into an embedded OS, the build system has evolved with it. Initially, NanoBSD targeted producing a small version of FreeBSD that matched the host’s architecture. NanoBSD now supports both native and cross building targets. In addition, FreeBSD’s base system now has the ability to install cross compilers into onto the system so that software outside of the base (`/usr/src`) can use them.

### 4.4.1 NanoBSD

NanoBSD is a collection of scripts to produce smaller FreeBSD images. It works within the FreeBSD build paradigm to remove those parts of FreeBSD you don’t need on the target system. It also provides a stylized framework for dealing with run-time configuration data. If you have enough space, it even allows for in-place upgrades using a ping-pong strategy to bounce between

two partitions. One partition is active, and the other is used for upgrades. This allows bad upgrades to be reverted to the last working configuration.

NanoBSD has been enhanced with cross-compilation support. It is now possible with NanoBSD to build an entire ARM system on an x86 box, for example. This allows one to use very fast development machines to build images for an embedded system. NanoBSD relies on a number of other tools in the system to create its images. A number of bugs in these tools have been corrected.

#### 4.4.2 Cross Compiler installation

One of the nicer features of FreeBSD is its ability to cross build itself. It does so by a rather complicated mechanism, leveraging Makefile magic to accomplish its goals. However, unless you're well integrated with the FreeBSD build system, and have a built tree laying around, you're not able to leverage this technology.

*Until now, that is.* Recently, the build system was patched to allow creation *and installation* of cross compilers. These cross compilers follow the gnu autoconf conventions to allow proper probing for cross-building support. These compilers offer advantages over the normal cross-build tools that can be installed from the ports collection. Since they are the actual system compilers, their behavior exactly matches a native FreeBSD build. They have also been tested with FreeBSD natively in numerous builds and cross builds that are done as part of the project's normal infrastructure.

## 5 Conclusion

Although a relative newcomer to the tree, FreeBSD/mips has a long history behind it. The port is gaining momentum, and promises to be with us for quite a while. New code to support different ASICs is entering the tree all the time. Vendors are starting to approach the FreeBSD project to integrate their code into the tree. FreeBSD supports more and more devices important in the embedded world, which makes deploying FreeBSD/mips easier. FreeBSD build system has been enhanced to better support embedded development. FreeBSD/mips is an exciting new possibility when considering operating systems to run your MIPS-based embedded device.

## References

- [1] *FreeBSD Home Page*. n.d. FreeBSD Project. February 2009. <http://www.freebsd.org/>.
- [2] *FreeBSD Handbook*. n.d. FreeBSD Doc Project. February 2009. [http://www.freebsd.org/doc/en\\_US.ISO8859-1/books/handbook/](http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/).
- [3] Dominic Sweetman, *See MIPS®Run*. San Francisco: Morgan Kaufmann, second ed., 2007.
- [4] *Juniper Networks*. n.d. Juniper Networks. February 2009. <http://www.juniper.net/>.
- [5] *Cisco Systems, Inc.* n.d. Cisco Systems. February 2009. <http://www.cisco.com/>.
- [6] *ADM5120 Network Processor Data Sheet, Rev 1.1*. München, Germany, March 2005.
- [7] *Interprise Integrated Communication Processor 79RC32434*. November 2002. IDT. February 2009. <http://www.idt.com/index.cfm?genID=79RC32434>.
- [8] *Microtik Routerboard RB/532*. n.d. DD-WRT Project. February 2009. [http://www.dd-wrt.com/wiki/index.php/Mikrotik\\_Routerboard\\_RB/532](http://www.dd-wrt.com/wiki/index.php/Mikrotik_Routerboard_RB/532).
- [9] *Microtik Routerboard RB/532*. n.d. Microtik. February 2009. <http://www.routerboard.com/rb500.html>.
- [10] Broadcom Corporation, "BCM6365/BCM5365P Product Brief." PDF Product Brief, 2004.
- [11] *BCM5365/5365P Sentry5™ Secured Switch Processor*. n.d. Broadcom Corporation. February 2009. <http://www.broadcom.com/products/Wireless-LAN/802.11-Wireless-LAN-Solutions/BCM5365-5365P>.
- [12] *Products & Technology : AR7100*. n.d. Atheros Communications. February 2009. <http://www.atheros.com/pt/AR7100.htm>.
- [13] *Alchemy Processor Family*. n.d. RMI Corporation. February 2009. <http://www.rmicorp.com/products/Au1550.htm>.
- [14] *Products: Octeon MIPS64*. n.d. Cavium Networks. February 2009. [http://www.caviumnetworks.com/OCTEON\\_MIPS64.html](http://www.caviumnetworks.com/OCTEON_MIPS64.html).