Proceedings of the Linux Symposium

June 27th–30th, 2007
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc.*
Dirk Hohndel, *Intel*
Martin Bligh, *Google*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
John Feeney, *Red Hat, Inc.*
Len DiMaggio, *Red Hat, Inc.*
John Poelstra, *Red Hat, Inc.*

Breaking the Chains—Using LinuxBIOS to Liberate Embedded x86 Processors

Jordan H. Crouse
Advanced Micro Devices, Inc.
jordan.crouse@amd.com

Marc E. Jones
Advanced Micro Devices, Inc.
marc.jones@amd.com

Ronald G. Minnich
Sandia National Labs

Abstract

While x86 processors are an attractive option for embedded designs, many embedded developers avoid them because x86-based systems remain dependent on a legacy BIOS ROM to set up the system. LinuxBIOS is an open source solution that replaces the proprietary BIOS ROMs with a light-weight loader. LinuxBIOS frees the developer from complex CPU and chipset initialization and allows a variety of payloads to be loaded, including the Linux kernel itself.

This presentation reviews the journey of the AMD Geode™ processors and chipset as they were integrated into LinuxBIOS to become the centerpoint of the One Laptop Per Child (OLPC) project. We also discuss how embedded developers can take advantage of the LinuxBIOS environment for their own x86-based projects.

1 Introduction

Ever since the x86 Personal Computer (PC) architecture was introduced in 1981, it has been accompanied by bootstrap code known as the Basic Input/Output System (BIOS) that executes a Power On Self Test (POST). Almost every year since the PC's introduction, hardware and operating system features have increased in complexity. Each new feature adds complexity to the BIOS, which must maintain compatibility with older operating systems and yet also provide support for new ones. The end result is a convoluted and cryptic combination of old standards (such as software interrupts for accessing the display and storage devices) and new standards (such as Advanced Configuration and Power Interface (ACPI)).

Almost all BIOS implementations are proprietary and many Open Source developers are in conflict with what is perceived to generally be a "black magic" box. Due to the arcane nature of the BIOS, most modern operating systems have abandoned the BIOS hardware interfaces and access the hardware directly. The desktop computer focus of the traditional BIOS model frustrates embedded systems designers and developers, who struggle to get a BIOS that embraces their unique platforms. Due to the very specific requirements for system boot time and resource usage, it is difficult to meet embedded standards with a BIOS designed for two decades of desktop computers.

The LinuxBIOS project exists to address legacy BIOS issues. It is licenced under the GNU Public License (GPL) to promote a transparent and open loader. LinuxBIOS provides CPU and chipset initialization for x86, x86_64, and Alpha systems and allows the flexibility to load and run any number of different payloads.

This paper discusses the development and use of LinuxBIOS for embedded x86 platforms based on AMD Geode processors. The first section examines the history of LinuxBIOS and the AMD Geode processors. The next section moves into detail about the system initialization process. The final section discusses integrating payloads with the LinuxBIOS firmware.

2 History

> "History is a guide to navigation in perilous times. History is who we are and why we are the way we are."
>
> *—David C. McCullough*

2.1 LinuxBIOS History

Ron Minnich started the LinuxBIOS project at Los Alamos National Lab (LANL) in September 1999 to address problems caused by the PC BIOS in large clusters. The team agreed that the ideal PC cluster node would have the following features:

- Boot directly into an OS from non-volatile RAM;
- Configure only the network interfaces;
- Connect to a control node using any working network interface;
- Take action only at the direction of the control node.

At that time, the LANL team felt that Linux® did a better job of running the hardware than the PC BIOS. Their concept was to use a simple hardware bootstrap to load a small Linux kernel from flash to memory. Leveraging work from the OpenBIOS project, the LinuxBIOS team booted an Intel L440GX+ motherboard after approximately six months of development. Early on, the team decided that assembly code would not be the future of LinuxBIOS. OpenBIOS was disregarded because it was based on a great deal of assembly code and a difficult-to-master build structure. The team found a simple loader from STMicroelectronics called STPC BIOS that was written in C and available to be open sourced, so it became the basis for the first version of LinuxBIOS.[1]

In 2000, Linux NetworX and Linux Labs joined the effort. The LinuxBIOS team added Symmetric Multiple Processor (SMP) support, an Alpha port, and created the first 13-node LinuxBIOS-based Supercomputing Clusters. Since 2001, the team has added developers and they continue to port to new platforms, including AMD Opteron™ processor- and AMD Athlon™ processor-based platforms. Interestingly enough, LinuxBIOS was originally designed for clusters, yet LinuxBIOS for non-cluster platforms far exceeds the cluster use.

In 2005, some current and past members of the MIT Media Lab joined together to create the One Laptop Per Child (OLPC) program, dedicated to making a low-cost laptop for educational projects around the globe. The OLPC team designed an x86 platform that incorporates an AMD Geode solution. As low price and open technology were part of the core requirements for the laptop, the designers decided to use a royalty-free open source BIOS solution, ultimately choosing LinuxBIOS. The first two board revisions included the AMD Geode GX processor based LinuxBIOS loader, originally utilizing a Linux-as-bootloader payload. This later transitioned to OpenFirmware after it became available in the middle of 2006. In 2007, AMD Geode LX processor support was added to the loader, making it a freely available reference BIOS design for interested developers of other AMD Geode solutions.

[1] Version 2 started after the addition of Hypertransport™ technology support changed the device model enough to warrant a version bump.

2.2 AMD Geode History

The AMD Geode processor is the offspring of the MediaGX processor released by Cyrix in 1997. The MediaGX saved total system cost by embedding a graphics engine that used one of the first Unified Memory Architecture (UMA) implementations. It also featured an integrated northbridge memory controller and SoundBlaster emulation in the CPU. The MediaGX broke the sub-$1000, sub-$500, and sub-$200 price barrier on the Compaq Presario 2100 in 1996 and 1997. In 1997, Cyrix was purchased by National Semiconductor, who renamed the MediaGX line to Geode. National Semiconductor released the Geode GX2 (today, just called the GX) and CS5535 companion chip in 2002. In 2003, the Geode division was sold to AMD. AMD focused heavily on power, price, and performance, and in 2005 released the AMD Geode LX 800@0.8W processor and CS5536 companion chip, with the LX 900@1.5W processor following in 2007.

The AMD Geode GX and LX processors support the i586 instruction set, along with MMX and 3DNow!™ extensions. The LX features a 64K instruction and a 64K data L1 cache and 128K L2 cache. Both processors have on-board 2D graphics and video accelerators. The LX adds an on-board AES engine and true random number generator. The CS5536 companion chip provides southbridge capabilities: IDE, USB 2.0, SMBus, AC97, timers, GPIO pins, and legacy x86 peripherals.

3 Geode LinuxBIOS ROM image

While the entire image is known as LinuxBIOS, it is constructed of individual pieces that work together. An AMD Geode LinuxBIOS ROM image is made up of three main binary pieces:[2]

- LinuxBIOS Loader: system initialization code;

[2] OLPC also adds additional binary code for its embedded controller.

- VSA2: the AMD Geode proccessor's System Management Interrupt (SMI) handler;
- Payload: the image or program to be loaded to boot the OS.

3.1 LinuxBIOS Architecture

LinuxBIOS version 2 is structured to support multiple motherboards, CPUs, and chipsets. The overall platform configuration is described in `Config.lb` in the `mainboard` directory. The `Config.lb` file contains important information like what CPU architecture to build for, what PCI devices and slots are present, and where code should be addressed. The mainboard also contains the pre-DRAM initialization file, `auto.c`. ROMCC compiles `auto.c` and generates a stackless assembly code file, `auto.inc`. The use of ROMCC works well for small sections of simple C code, but for complicated memory controller initialization, there are some issues with code size and C variable-to-register space conversion.

To work around the ROMCC issues, Yinghai Lu of AMD developed support for the AMD64 architecture's Cache-as-RAM (CAR) feature [1]. Compiled C code makes heavy use of the stack. Only a few lines of assembly code are needed to set up the CPU cache controller to be used as temporary storage for the stack. All the pre-DRAM initialization (including memory initialization) is compiled as normal C code. Once the memory controller is configured the stack is copied to real memory and the cache can be configured as normal. The AMD Geode processors are one of two CPUs to use a CAR implementation in LinuxBIOS version 2.[3]

3.2 LinuxBIOS Directory Structure

The LinuxBIOS source tree can be a bit daunting to a newcomer. The following is a short tour of the LinuxBIOS directory structure, highlighting the parts interesting to a systems developer.

The `cpu/` directory contains the initialization code for VSA2 and the AMD Geode graphics device.

[3] See the *Future Enhancements* section for more details about CAR in LinuxBIOS version 3.

```
linuxbios/src
|-- cpu
    |-- amd
        |-- model_gx2
        |-- model_lx
```

The `mainboard/` directory contains platform-specific configuration and code. The platform `Config.lb` file contains the PCI device configuration and IRQ routing. This directory also contains the source file compiled by ROMCC.

```
linuxbios/src
|-- mainboard
    |-- amd
        |-- norwich
    |-- olpc
        |-- rev_a
```

(**Note**: 'Norwich' is the code name for an AMD Geode development platform).

The source code in `northbridge/` includes memory initialization and the PCI bridge 0 configuration and initialization. In the AMD Geode processor's architecture, the northbridge is integrated into the CPU, so the directory name is the same as the CPU.

```
linuxbios/src
|-- northbridge
    |-- amd
        |-- gx2
        |-- lx
```

The `southbridge/` directory contains the source for SMBus, flash, UART, and other southbridge device configuration and initialization.

```
linuxbios/src
|-- southbridge
    |-- amd
        |-- cs5536
```

The `target/` directory contains the platform build directories. These include the configuration files that specify the build features including ROM size, VSA2

binary size, and the desired payload binary. This is also where the ROM image is built. A script called `buildtarget` in the `linuxbios/target` directory parses the target configuration files and builds the Makefiles for the ROM image in the platform target directory.

```
linuxbios/targets
|-- amd
    |-- norwich
|-- olpc
    |-- rev_a
```

3.3 LinuxBIOS Boot Process

Figure 1 details the process of booting a system with LinuxBIOS.

1. The AMD Geode processor fetches code from the reset vector and starts executing noncompressed (pre-DRAM) LinuxBIOS from the flash ROM. The early CPU, northbridge, and southbridge initialization takes place. Once the memory is initialized, LinuxBIOS decompresses and copies the rest of itself to low memory.

2. LinuxBIOS continues system initialization by walking the PCI tree starting at bus 0. Most of the AMD Geode device's internal configuration and initialization happens at this stage. Cache, system memory, and PCI region properties are configured. The VSA2 code is decompressed into low memory and executed.

3. The VSA2 initialization makes adjustments to the UMA for graphics memory and itself. VSA2 is then copied to the top of RAM and located logically in mid-PCI memory space, 0x80000000. VSA2 initializes the runtime components. Upon completion, control is returned to LinuxBIOS.

4. LinuxBIOS finishes enumeration and initialization of all the PCI devices in the system. PCI devices are allocated memory and I/O (including the VSA2 virtualized headers) and then enabled. During the southbridge PCI configuration, the presence of IDE-versus-flash capability and other configurations not controlled directly in PCI space are set up. The CPU device is the last device to enumerate and an end-of-POST SMI is generated to signal to VSA2 that the system is configured.

5. The last stage of LinuxBIOS is to load the payload. LinuxBIOS copies itself to the top of system memory and then locates and decompresses the payload image into memory. Finally, the payload is executed. See Section 4 for more details about the payload.

3.4 VSA2

Virtual System Architecture (VSA) is the AMD Geode device's System Management Mode (SMM) software. VSA2 is the second generation of VSA that supports GX and LX CPUs and the CS5535 and CS5536 chipsets. In a traditional BIOS, VSA2 handles normal SMI/SMM tasks like bug fixes, legacy USB, and power management (legacy, APM, and ACPI). VSA2 also handles virtual PCI configuration space for the AMD Geode device's internal controllers; graphics, IDE, flash, etc. PCI virtualization translates PCI configuration-space access to the internal device's GeodeLink™ Model-Specific Registers (MSRs). PCI configuration access is infrequent and virtualization is a good way to save silicon real-estate with software.

Since Linux manages most hardware devices on its own, it only requires VSA2 PCI virtualization.

Linux kernel drivers handle the power management, USB, and graphic controllers that would normally be controlled by VSA2 in a legacy BIOS environment. In the embedded Linux environment, only the PCI virtualization portion of VSA2 is required. Omitting the unneeded code saves space in the LinuxBIOS ROM image for larger payloads. VSA2 is in the process of being ported to GNU tools and will be released as open source. This will enable the open source community to write Virtual System Modules (VSMs) for additional features or to replace VSA2 entirely with a new AMD Geode chipset SMI handler.

The VSA2 image is compressed with NRV2B and concatenated to the beginning of the LinuxBIOS (with payload) image.[4]

4 Payloads

Once LinuxBIOS provides CPU and chipset initialization for the platform, it passes control to a payload that

[4] VSA2 is added at the beginning because execution starts at the end of the ROM image, where LinuxBIOS is located.

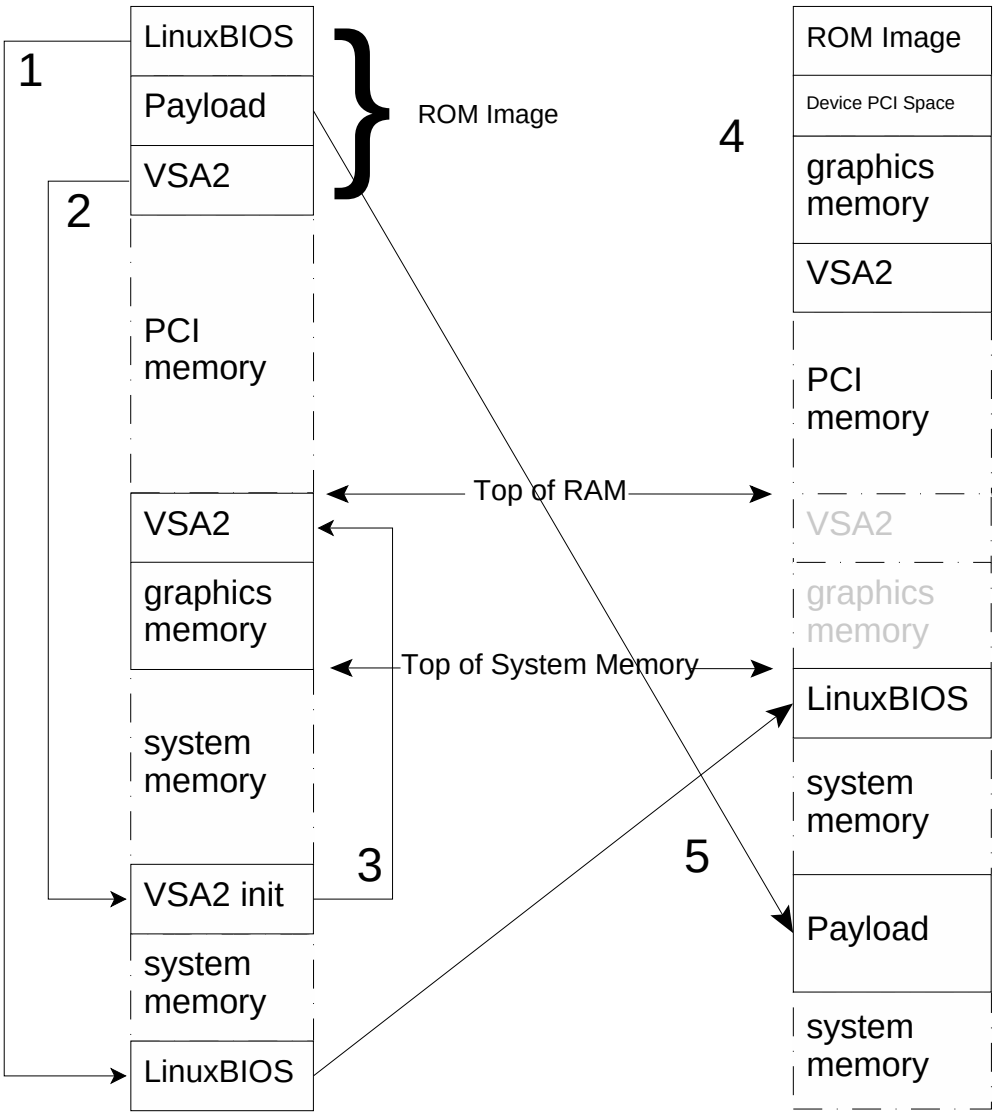

Figure 1: LinuxBIOS Memory Map

can continue the booting process. This is analogous to a traditional BIOS ROM, which also handles CPU and chipset initialization, and then passes control to code that manages the BIOS services (such as the setup screen, access to block devices, and ultimately starting the process that ends up in the secondary bootloader). The traditional BIOS code is tied closely to the loader and only provides support for a limited set of features. By contrast, a LinuxBIOS payload is far more flexible. In theory LinuxBIOS can load and run any correctly formatted ELF file (though in practice, the payload must be able to run autonomously without any operating system services). This allows the developer to choose from any number of available open source options, from simple loaders to the Linux kernel itself. This flexibility also allows embedded developers to easily craft custom solutions for their unique platform—for instance, supporting diskless boot with Etherboot, or loading and running a kernel from NAND flash or other non-traditional media.

When LinuxBIOS has finished initializing and enumerating the system, it passes control to the ELF loader to load the payload. The payload loader locates the stream on the ROM and decodes the ELF header to determine where the segments should be copied into memory. Before copying, the loader first moves the firmware to the very top of system memory to lessen the chance that it will be overwritten by the payload. LinuxBIOS stays

resident in case the ELF loader fails to load the specified payload. Many "standard" payloads (such as memtest86 and the Linux kernel) are designed to run on a system with a traditional BIOS image. Those payloads are loaded into firmware-friendly memory locations such as 0x100000. After copying and formatting the segments, the loader passes control to the entry point specified in the ELF header. System control leaves LinuxBIOS and passes to the payload.

4.1 Linux Kernel Payloads

While there are many different types of loaders, loading the Linux kernel directly from the ROM was the original goal of the LinuxBIOS project. The kernel can either be loaded by itself and mount a local or network-based filesystem, or it can be accompanied by a small RAMdisk filesystem that provides additional services for finding and booting the final kernel image. This is known as "Linux as Bootloader" or simply, LAB.

The Linux kernel is a very compelling payload for several reasons. The kernel already supports a huge number of different devices and protocols, supporting virtually any platform or system scenario. The kernel is also a well known and well supported entity, so it is easy to integrate and extend. Finally, the great majority of LinuxBIOS implementations are booting the Linux kernel anyway, so including it in the ROM greatly simplifies and accelerates the boot process. Using Linux as a bootloader further extends the flexibility by including a RAMdisk with user-space applications that can access a network or provide graphical boot menus and debug capabilities.[5]

The challenge to using a main kernel in a LinuxBIOS payload is that it is often difficult to shrink the size of the kernel to fit in the ROM. This can be mitigated by using a larger ROM. In most cases the additional cost of the flash ROM is offset by the improved security and convenience of having the main kernel in the ROM image. Another concern is the ability to safely and quickly upgrade the kernel in the ROM image. It is a dangerous matter to flash the ROM, since a failed attempt usually results in a "brick" (an unbootable machine). This can be avoided in part by increasing the size of the flash ROM and providing a safe "fallback" image that gets invoked in case of a badly flashed image. The advantages outweigh the costs for embedded applications that rarely upgrade the kernel image.

As may be expected, the standard Linux binary files require some manipulation before they can be loaded. A tool called `mkelfimage`[6] is used to combine the kernel text and data segments and to add setup code and an optional RAMdisk into a single loadable ELF file.

Table 1 shows the program headers read by `readelf` from the loadable ELF file created by `mkelfimage` from a vmlinux file and a 1MB RAMdisk.

The first segment contains code similar to the Linux startup code that de-compresses the kernel and prepares the system to boot. This section also contains setup information such as the kernel command line string. The next segment allocates space for a GDT table that is used by the setup code. Kernel developers will note the familiar `.text` segment loaded to 0x100000 and the subsequent `.data` segment. Finally, the 1MB RAMdisk is copied to address 0x800000.

4.2 Other Payloads

Several popular Open Source Software (OSS) standalone applications have been adapted to run as LinuxBIOS payloads. These include the `memtest86` memory tester and the `etherboot` network boot utility. `etherboot` is particularly interesting since it provides an open source alternative to the PXE protocol. It can easily enable any system to boot an image from a network even with network cards that do not natively support PXE boot. Another interesting option appeared during the early days of the OLPC project when Sun Microsystems unexpectedly released key portions of the OpenFirmware loader. Also known as OpenBoot, OpenFirmware is a firmware package programmed in Forth that serves as the bootloader on SPARC-based workstations and PowerPC-based systems from Apple and IBM. When it became available, OpenFirmware was quickly adapted to load on the OLPC platform as a LinuxBIOS payload.

4.3 Building Payloads

The payload is integrated with the LinuxBIOS loader during the LinuxBIOS build process. During configura-

[5] Some LinuxBIOS developers have been experimenting with fitting an entire root filesystem into the the ROM. See reference [2].

[6] Written by Eric Biderman, Joshua Aune, Jake Page, and Andrew Ip.

```
Type   Offset   VirtAddr   PhysAddr   FileSiz  MemSiz

LOAD   0x000144 0x00010000 0x00010000 0x0561c  0x1ab24
LOAD   0x005760 0x00091000 0x00091000 0x00000  0x00070
LOAD   0x005760 0xc0100000 0x00100000 0x264018 0x264018
LOAD   0x269778 0xc0365000 0x00365000 0x4b086  0xaf000
LOAD   0x2b47fe 0x00800000 0x00800000 0x100000 0x100000
```

Table 1: ELF Sections from Loadable Kernel

```
# option CONFIG_COMPRESSED_ROM_STREAM_NRV2B=0
option CONFIG_COMPRESSED_ROM_STREAM_LZMA=1
option CONFIG_PRECOMPRESSED_ROM_STREAM=1
# Need room for VSA
option ROM_SIZE=(1024*1024)-(64*1024)
...
romimage "fallback"
      ...
      payload /tmp/payload.elf
end
```

Figure 2: OLPC LinuxBIOS Configuration

tion, the developer specifies the size of the LinuxBIOS ROM image and a pointer to the payload binary. Optionally, the payload can be NRV2B- or LZMA-compressed to conserve space at the expense of speed. Figure 2 shows a set of example configuration options for an AMD Geode processor-based target with a 1MB flash ROM and a compressed payload.

During the LinuxBIOS build, the payload is compressed (if so configured), and integrated in the final ROM image as shown previously in Figure 1.

4.4 BuildROM

Constructing a LinuxBIOS ROM image from start to finish can be a complicated and tedious process involving a number of different packages and tools. BuildROM is a series of makefiles and scripts that simplify the process of building a ROM image by consolidating tasks into a single `make` target. This provides a reproducible build that can be replicated as required. BuildROM was inspired by Buildroot,[7] and was originally designed to build Linux-as-bootloader (LAB) based ROM images for the OLPC project. The OLPC LAB used a simple RAM filesystem that was based on `uClibc` and `Busybox`, and ran a simple graphical tool that could use `kexec` to load a kernel from USB, NAND, or from the network. This involved no less than six packages and a number of tools—a nightmare for the release manager, and very difficult for the core team to duplicate and run on their own platforms. BuildROM simplified the entire process and makes it easy to build new ROM image releases as they are required. More recently, it has been extended to build a number of different platforms and payloads.

[7] `http://buildroot.busybox.org`

5 Advantages and Disadvantages of LinuxBIOS

Like most open source projects, LinuxBIOS continues to be a work in progress, with both positive and negative aspects.

Chief among the positive aspects is that LinuxBIOS is developer-friendly, especially when compared to traditional BIOS solutions. LinuxBIOS is mostly C-based, which greatly simplifies development. However, machine-generated code is almost always larger and slower than hand-tuned assembly, which is a liability, especially in the pre-DRAM section where speed and size are of the essence. As mentioned before, ROMCC does an amazing job of generating stackless assembly

code, but due to the complexity of its task, it is difficult to optimize the code for minimum size and maximum efficiency.

Even though the LinuxBIOS code is written in C, the developer is not freed from having to look through the generated assembly to verify and debug the solution. Assembly code in LinuxBIOS is written in the AT&T format (as are all GNU tools-based projects), but many traditional BIOS projects and x86 CPU debuggers use the Intel format. This may cause a learning barrier for developers transitioning to LinuxBIOS, as well as making it somewhat difficult to port existing source code to LinuxBIOS.

The current AMD Geode LinuxBIOS implementation is slower then expected. Benchmarks show that decompression and memory copy are slower then other ROM implementations. More investigation is needed to determine why this happens.

The positive aspects of LinuxBIOS more than make up for these minor issues. LinuxBIOS uses a development environment familiar to embedded Linux developers. It is written in C and uses 32-bit flat mode. There is no need to worry about dealing with 16-bit real or big real modes.

In the end, while LinuxBIOS is backed by a strong open source community, it cannot exist without the support of the hardware vendors. The growth of LinuxBIOS will ultimately depend on convincing hardware companies that there is a strong business case for developing and supporting LinuxBIOS ports for their platforms.

6 Future Enhancements

There is still much to be done for the AMD Geode chipset LinuxBIOS project. LinuxBIOS version 3 promises to to be a great step forward. Among the changes planned include:

- A new configuration system based on the the kernel config system;
- Replacing remaining stackless pre-DRAM code with Cache-as-RAM (CAR) implementations;
- Speed and size optimizations in all facets of the boot process.

The AMD Geode chipset code will be transitioned to work with LinuxBIOS version 3, including better integration with the default CAR mode, and speed optimizations. Also, more work needs to be done to support a fallback image to reduce the chance that a failed ROM flash will break the target machine.

Changes are also in store for VSA2. The code will be ported to compile with GNU tools, and fully released so that others can build on the existing SMI framework. Further VSA2 work will center around power management, which will be new ground for LinuxBIOS-based ROMs. Finally, continuing work will occur to enhance BuildROM and help make more diagnostic tools available to validate and verify LinuxBIOS in an open source environment.

7 Conclusion

LinuxBIOS is an exciting development in the world of the AMD Geode chipsets and x86 platforms in general. It facilitates the efforts of developers by avoiding the pitfalls of a traditional BIOS and provides great flexibility in the unique scenarios of embedded development. There is a great advantage for the AMD Geode processors in supporting LinuxBIOS because LinuxBIOS allows designers to consider AMD Geode solutions in ways they never before thought possible (as evidenced by the early success story of the very non-traditional OLPC platform). We look forward to continuing to participate with LinuxBIOS as it transitions into version 3 and beyond.

8 Acknowledgements

The authors would like to extend a special thank you to all the people who helped make the AMD Geode LinuxBIOS image possible: Ollie Lo and the LinuxBIOS team, Yinghai Lu, Tom Sylla, Steve Goodrich, Tim Perley and the entire AMD Embedded Computing Solutions team, and the One Laptop Per Child core software team.

9 Legal Statement

References

[1] Yinghai Lu, Li-Ta Lo, Gregory Watson, Ronald Minnich. *CAR: Using Cache as RAM in LinuxBIOS*. http://linuxbios.org/data/yhlu/cache_as_ram_lb_09142006.pdf

[2] LinuxBIOS with X Server Inside, posted to *LinuxBIOS Developers Mailing List*, March 2007. http://www.openbios.org/pipermail/linuxbios/2007-March/018817.html

[3] Ronald Minnich. *LinuxBIOS at Four*, In *Linux Journal* #118, February 2004. http://www.linuxjournal.com/article/7170

[4] Ronald Minnich, *Porting LinuxBIOS to the AMD SC520*, in *Linux Journal* #136, August 2005. http://www.linuxjournal.com/article/8120

[5] Ronald Minnich, *Porting LinuxBIOS to the AMD SC520: A followup Report*, July 2005. http://www.linuxjournal.com/article/8310

[6] Advanced Micro Devices, Inc. *AMD Geode™ LX Processors Data Book*, June 2006. http://www.amd.com/files/connectivitysolutions/geode/geode_lx/33234E_LX_databook.pdf

[7] Advanced Micro Devices, Inc. *AMD Geode™ CS5536 Companion Device Data Book*, March 2006. http://www.amd.com/files/connectivitysolutions/geode/geode_lx/33238f_cs5536_ds.pdf

[8] Advanced Micro Devices, Inc. *AMD Geode™ GX and LX Processor Based Systems Virtualized PCI Configuration Space*, November 2006. http://www.amd.com/files/connectivitysolutions/geode/geode_gx/32663C_lx_gx_pciconfig.pdf

[9] LinuxBIOS, http://linuxbios.org. svn://openbios.org/repos/trunk/LinuxBIOSv2

[10] One Laptop Per Child. http://laptop.org

[11] OpenFirmware. http://firmworks.com svn://openbios.org/openfirmware

[12] Memtest86. http://memtest86.com

[13] Etherboot. http://www.etherboot.org/wiki/index.php