coreboot - the free firmware

vimacs <https://vimacs.lcpu.club>

Beijing GNU/Linux User Group

June 13th, 2017

License

Index

What is coreboot?

coreboot is an extended firmware platform that delivers a lightning fast and secure boot experience on modern computers and embedded systems. As an Open Source project it provides auditability and maximum control over technology.
The word 'coreboot' should always be written in lowercase, even at the start of a sentence.

History: from LinuxBIOS to coreboot

coreboot has a very long history, stretching back more than 15 years to when it was known as LinuxBIOS. While the project has gone through lots of changes over the years, many of the earliest developers still contribute today.

LinuxBIOS v1: 1999-2000

The coreboot project originally started as LinuxBIOS in 1999 at Los Alamos National Labs (LANL) by Ron Minnich. Ron needed to boot a cluster made up of many x86 mainboards without the hassles that are part of the PC BIOS. The goal was to do minimal hardware initilization in order to boot Linux as fast as possible. Linux already had the drivers and support to initialize the majority of devices. Ron and a number of other key contributors from LANL, Linux NetworkX, and other open source firmware projects successfully booted Linux from flash. From there they were able to discover other nodes in the cluster, load a full kernel and user space, and start the clustering software.

LinuxBIOS v2: 2000-2005

After the initial success of v1, the design was expanded to support more CPU architectures (x86, Alpha, PPC) and to support developers with increasingly diverse needs.
One of the design goal is to have little assembly as possible.

- standard C cannot be used because C compiler requires memory for stack
- the new DDR memory controllers required significantly more configuration and a lot more ASM
- solution: ROMCC by Eric Biederman

LinuxBIOS device tree was introduced.
Many target systems had flash that were too small to hold both the hardware initialization code and the Linux kernel. Payloads were created.

- modified etherboot for clusters
- FILO for disk-based boot

LinuxBIOS v2+: 2005-2008

Cache as RAM was introduced in 2005.
Stefan Reinauer formed a company named coresystems GmbH to support LinuxBIOS. Stefan was one of the primary developers and co-leaders of LinuxBIOS with Ron Minnich. His significant contributions included the first AMD64 port, the original ACPI implementation, the original SMM implementation, the flashrom utility, and the FILO payload development and maintainer.
In 2005, FSF started the Free BIOS campaign to support LinuxBIOS development. Ward Vandewege of FSF ported LinuxBIOS to the FSF servers and other mainboards.

LinuxBIOS v3: 2006-2008

By 2006, LinuxBIOS had already supported hundreds of mainboards. With so many boards, there were problems with porting additional silicon and systems.

- Developers fixed and clarified many driver and bus support issues in the device tree.
- Kconfig
- firmware image archive called LAR (LinuxBIOS Archiver), which led to the more refined and flexible concept of CBFS

It wasn't the main development branch; it was essentially an R&D branch, where the best ideas were backported to v2.

2008: LinuxBIOS renamed coreboot

`https://www.coreboot.org/pipermail/coreboot/2008-January/029135.html`

- LinuxBIOS = (core boot code) + (Linux kernel)
- Linux was no longer booted directly from flash

coreboot v4

- In early 2010, coreboot moved from SVN to Git
- during the transition, the community took the opportunity to recognize the advancements of the past 10 years and updated to version 4.0.
- contributions from AMD: AMD Generic Encapsulated Software Architecture (AGESA)
- Google Chromebook
- Intel FSP
- libreboot for ThinkPad T60
- coreboot v4.1

About libreboot

Some firmware components are non-free:

- Intel ME firmware/AMD PSP
- Intel FSP/closed source AGESA
- Option ROMs (including VGA BIOS)
- CPU microcode
- EC firmware

Libreboot is a coreboot distribution that removes proprietary components, including Intel ME, FSP, VGA BIOS, etc. On some laptops, the EC firmware is also free(Chromium EC in Chromebooks).

Why use coreboot

You can see the advantages of coreboot at:
`https://www.coreboot.org/users.html`

- coreboot is free software (see FSF Free BIOS Campaign)
- fast boot times
- it's flexible

Fun stuff on BIOS

- https://www.coreboot.org/Fun_Stuff

How coreboot works

We'll take lenovo/x230 as example to see how a machine boots with coreboot.
We can build coreboot with "`make V=1 > build.log`" to see which files are used to build coreboot for this mainboard.

coreboot stages

The coreboot firmware runs in several stages.

- bootblock: the earliest code that the CPU runs
- romstage: before main memory is ready, only the code in the flash can be run, and no other memory can be used
- ramstage: after main memory can be used, the ramstage code is uncompressed in memory, and do the remaining initialization
- payload: OS or OS loader

bootblock

When the machine starts, PC poionts at reset vector (f000:fff0), the CPU runs the bootblock code.
The bootblock code is in src/arch/x86/bootblock_romcc.S, which includes:

- src/cpu/x86/16bit/reset16.inc: the code at reset vector
- src/cpu/x86/16bit/entry16.inc: the 16-bit code that sets CPU to protected mode
- src/cpu/x86/32bit/entry32.inc: sets segment registers
- generated/bootblock.inc: generated from src/arch/x86/bootblock_simple.c with ROMCC

bootblock_simple.c then runs romstage.

romstage

romstage starts at src/arch/x86/assembly_entry.S, which includes:

- src/cpu/x86/32bit/entry32.inc: loads GDT and sets segment registers
- generated/assembly.inc: generated from src/cpu/intel/model_206ax/cache_as_ram.inc, which sets up CAR and runs the init code in romstage by calling ramstage_main(), then runs ramstage by calling romstage_after_car() which calls run_ramstage().

ramstage_main() is in src/cpu/intel/car/romstage.c, it calls mainboard_romstage_entry() in src/northbridge/intel/sandybridge/romstage.c, which does the DRAM initialization.

ramstage

ramstage starts at src/arch/x86/c_start.S, it calls the “main“ function in src/lib/hardwaremain.c.
There are 12 boot states defined in source code. Functions for each state are run in ramstage. At last payload is loaded and run.

Payloads

There are many coreboot payloads:

- SeaBIOS: a PC BIOS implementation
- GRUB2
- Linux kernel
- Tianocore: a UEFI implementation by Intel
- depthcharge: a bootloader written by Google for Chromebooks
- u-boot

Some useful tools can also be payloads:

- nvramcui: a tool to edit CMOS
- coreinfo: system information
- memtest86+

Supported OSes

coreboot supports many operating systems:

- Linux: boots via GRUB2,SeaBIOS, or using Linux kernel as payload
- OpenBSD: boots via SeaBIOS with VGA option ROM. Now it supports UEFI and don't need VGA BIOS, so it now supports libreboot. See libreboot mailing list.
- Windows: boots via SeaBIOS and Tianocore

Building coreboot and run on QEMU

`https://www.coreboot.org/Lesson1` is a good place to start.

Build a cross toolchain for building coreboot

make crossgcc or `make crossgcc-<arch>`

Generate a configuration

make nconfig

At last do a “make“ to build it, the built firmware is at build/coreboot.rom.

Run coreboot on QEMU

qemu-system-i386 -bios build/coreboot.rom

Hardware flashing

To use coreboot on real machines, we need to know how to flash externally with a hardware flash programmer.

- Most of the mainboards can only be flashed externally with factory firmware running.
- We need to flash externally to unbrick a machine.

We need a programmer and a connector to connect the flash.

Tools to flash a chip

A lot of mainboards use SPI NOR flash, most of them are in SOIC-8 package.

To program SPI NOR flash, we can use one of the following tools:

- Using Linux SPI: Raspberry Pi, Beaglebone
- Micro controllers: Arduino, Bus Pirate, STM32
- Programmers: ch341a, dediprog

To connect with SOIC-8 or SOIC-16 flash chips, we can use a clip.

Flash with flashrom

We can use flashrom to flash the chips.

command line

```
flashrom -p <programmer> [-r <file>] [-w <file>]
```

For Linux SPI

```
flashrom -p linux_spi:dev=/dev/spidev1.0,spispeed=1024
```

For ch341a

```
flashrom -p ch341a_spi
```

Internal flashing with flashrom

Many boards can be flashed internally with coreboot flashed.

internal flashing

```
flashrom -p internal:laptop=force_I_want_a_brick
```

We can also use a layout file to flash part of the ROM.

flashing with layout

```
flashrom -p <prog> --layout layout.txt \
  --image bios -w <file>
```

Utilities and Debugging

To work with coreboot, we have many tools to use:

- nvramtool: dump CMOS contents, display and modify CMOS settings
- cbmem: display coreboot table and logs
- ectool: read and write EC memory, sometimes useful
- autoport: generate the code for a board you want to port, usually needs further changes

To debug coreboot, we can read the cbmem console in a working system. We can also use a EHCI debug dongle.

- Net20DC is an expensive device, and its company is bankrupt.
- We can use a development board with OTG port, e.g. BeagleBone, Cubieboard

For systems with serial output, we can also do the debugging with it.

The coreboot community

You can visit “Community and infrastructure“ section of `https://www.coreboot.org/developers.html` to get an overview of coreboot community.
Some advice:

- Learn to use mailing list and IRC.
- Ask Questions The Smart Way

Community resources

- Homepage: `https://www.coreboot.org`
- Mailing list: coreboot@coreboot.org
- IRC: #coreboot at irc.freenode.net
- Mattermost (bridged to IRC): `https://chat.coreboot.org`
- twitter: @coreboot_org

Reporting bugs and Writing documents

- To join the community, learn to use mailing list and IRC.
- There's a bug tracking system: `http://ticket.coreboot.org/`
- We can apply for a wiki account and write coreboot wiki.

I want to write some code

Gerrit code review is the project management system for coreboot.
To push code to gerrit, you can manually set up the scripts, or just run `make gitconfig`.
Using gerrit is easy:

- To push code: `git push origin HEAD:refs/for/master`
- We can add a topic: `HEAD:refs/for/master%topic=some_topic`
- To push a draft: `HEAD:refs/drafts/master`

I recommend working in a new git branch instead of master.

Google Summer of Code

coreboot has been a GSoC mentoring organization for many years. We can read the project reports at `https://blogs.coreboot.org`.

- GSoC 2016 projects:
  - better RISC-V support
  - serialICE
  - flashrom
- GSoC 2015 projects:
  - H8S Embedded Controller
  - coreboot for AArch64 QEMU
  - end user flash tool

The status of coreboot

The current development of coreboot focuses on:

- improvement on old platforms
- utilities and payloads
- Google Chromebooks and related chips
- new architectures: RISC-V, POWER8, etc.

Chips on a mainboard

coreboot needs to initialize these chips.

- CPU
- northbridge: RAM init and graphics init
- southbridge: PCI, USB, SATA, LPC, GPIO
- super I/O: used to support
- embedded controller

Code for a mainboard

You can see what is needed for a mainboard in the directory for this mainboard.

- Kconfig: specifies what chips and drivers are used
- romstage.c: romstage code, including early southbridge init and reading DRAM SPD data
- devicetree.cb, mainboard.c: mainboard specific data
- gpio.c: GPIO config
- acpi/, dsdt.asl, acpi_tables.c, smihandler.c: ACPI and SMM code, some of the code is EC related
- cmos.layout, cmos.default

Using autoport

autoport is a tool to generate coreboot code for Sandy/Ivy Bridge boards.
It uses inteltool to read the northbridge and southbridge registers.
Manual fixes (see util/autoport/readme.md):

- where to read SPD data
- what is the EHCI debug port
- flash chip size
- EC and super I/O support!

Example

I made coreboot boot on HP Elitebook 2760p (`https://review.coreboot.org/c/18241/`) half a year ago.

- The flash chip is **socketed**, and is very easy to swap!
- Sandy Bridge platform, so use autoport

First, we need to make it boot, but not so easy:

- It needs two blobs, otherwise the EC will not function!
- see util/kbc1126/README.md

Fixes (keyboard)

After adding the blobs, the laptop boots!
Keyboard doesn't work.

- KBC not initialized
- It uses SMSC KBC1126 which provides EC, super I/O, and KBC
- I found an SMSC KBC1122 datasheet
- Also I found src/superio/smsc/kbc1100/, so the keyboard works finally

How to use existing drivers:

- Add it to Kconfig
- Check other boards that use this driver and learn from it

Fixes (fan control)

The laptop fan always runs on full speed, that's because EC is not initialized properly.
Reverse engineering it!

- Use UEFITool to extract the UEFI driver
- Check UEFI specification and related documents (e.g. EFI CPU I/O Protocol Specification) to identify the UEFI protocols

A lot of things to be done...

- ACPI support
- GRUB payload doesn't work
- ...

References

- coreboot history: Embedded Firmware Solutions, by Jiming Sun, Vincent Zimmer, Marc Jones, and Stefan Reinauer
- libreboot: `https://libreboot.org/faq/`