Coreboot: the open source BIOS.

Lennart Benschop

Presentation at T-DOSE 2011

Sunday 2011-11-06 10:00

Overview

- Inroduction
- Why Coreboot?
- Architecture
- Practical matters
- Demo (using virtual machine)
- Questions

Introduction

- What is Coreboot?

  - Free firmware for the x86 architecture, alternative to BIOS

- When was the project started?

  - In 1999, at Los Alamos National Laboratory as LinuxBIOS

- Why?

  - Enable unattended booting of cluster nodes

Why Coreboot?

- Applications:

  - Embedded x86 systems

  - Rackmounted servers and clusters

  - Normal PCs

Why for embedded systems?

- Embedded x86 applications:

  - media centers, smart TVs, PVRs

  - Test & measurement, logic analyzers etc.

  - Industrial control

  - Ticket vending machines, voting machines, photo kiosks etc.

But Why?

- No royalty payment (only relevant for high volume products)
- Fast booting
- Reliable untattnded booting

Why for servers and clusters?

- Reliable unattended booting

  - No keyboard

  - Do not depend on CMOS parameters

  - Control of which device to boot from

  - Independent of local harddisks, boot using network protocols

Why for normal PCs?

- Because we can
- Continued updates/bug fixes
- Stay in control of your own hardware (secure boot)

Architecture

- Payload
- Coreboot proper

  - RAM stage (running from RAM)

  - ROM stage (running from ROM)

    - C part

    - Assembly part (boot block)

Payloads

- Hardware independent (comparable to Linux kernel)
- Types of payload

  - SeaBIOS (implementation of legacy BIOS)

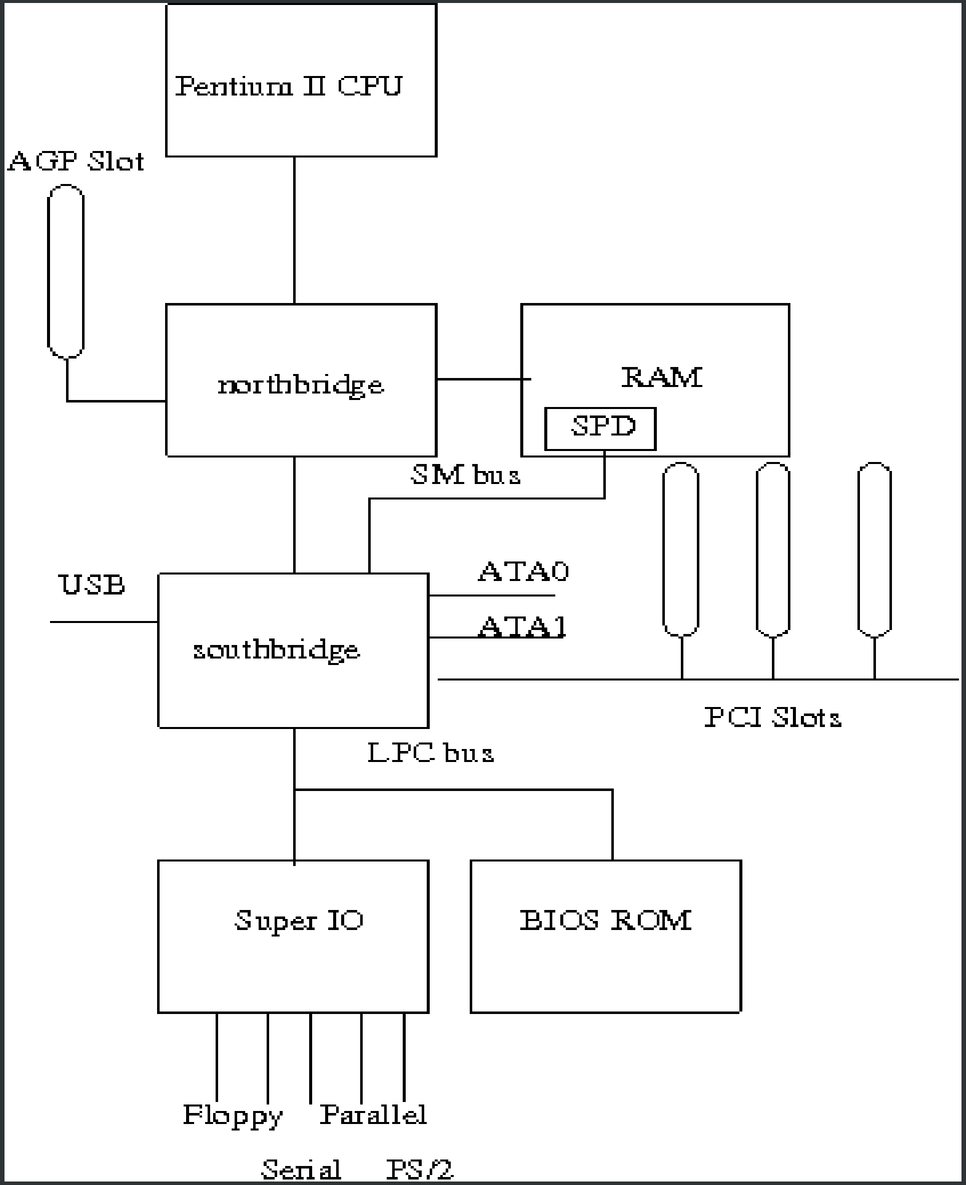
  - Linux kernel

  - Boot loader (FILO)

  - Application (game, diagnostics).

  - In future: Tiano Core (UEFI implementation)

PC Architecture



Startup Sequence

- Switch to protected mode (assembly)
- Enable Cache as RAM (assembly)
- Initialize RAM (C, ROM stage)

  - read SPD ROMs

  - configure memory module in Northbridge
- Initialize PCI devices (C, RAM stage)
- Run option ROMs
- Start payload

CBFS

- Read-only file system in ROM
- Named entries (e.g. fallback/ramstage)
- Entries can be compressed (LZMA)
- Stages, payload, option ROMs, config files

Coding

- Almost completely in C
- Almost completely in 32-bit protected mode

  - Real mode parts restricted to:

    - 16 instructions at startup
    - Option ROMs
    - SeaBIOS (has to provide legacy interface)

Practical Matters

- Obtaining sources

  - Git repository
- Compiling
- Flashing
- Debugging
- Development

Compiling

- Cross-compiler required in most cases
- Configuring (make menuconfig)
- Compile payload separately (for some payloads)
- Make
- Add files to ROM image (cbfstool)

Flashing

- Project flashrom at coreboot.org

  - Allows flashing under Linux

- Need to recover

  - PROM programmer (expensive)

  - Dedicated programmer for one chip (cheap)

  - Hot swapping (dangerous)

  - BIOS Savior

  - ROM emulator

Debugging

- POST codes on port (0x80) easy on ISA, harder on PCI.
- Serial output
- EHCI debug port
- Debug payloads and higher layers under QEMU
- AMD SimNow
- ROM emulator
- In-circuit emulator (expensive)

Development

- Payload develment

  - Libpayload as library for payloads

  - Hardware independent (perfectly fine on QEMU)

- Porting to new hardware (more challenging)

  - Documentatiion hard to obtain

  - Very hardware dependent

  - Can only be debugged on real hardware

Resources

- Website http://www.coreboot.org
- Mailing lists
- Git repository
- Related projects:

  - flashrom

  - SeaBIOS