Porting Coreboot to new x86 Targets.

Herein we describe the various technical aspects to consider while porting Coreboot to a

Prepared for Altera Praxis Pty Ltd;

Abstract

???

Keywords

disk, security, personal, server, UNIX, laptop, embedded, coreboot, BIOS, firmware, anti-tamper, integrity

Contents

Current Document Revision.

The current document revision was produced by the following commiter:

git author name: (None)
git author date: (None)
git hash: (None)
git references: (None)
git version tag:

Chapter 1

Firmware

1.1 Coreboot

Coreboot Structural Overview

The basic consistences of *coreboot.rom* is illustrated in figure 1.1.

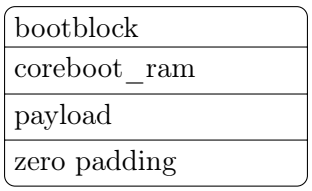

Figure 1.1: The *coreboot.rom* structure.

Bootblock

The *bootblock* is the earliest initialisation code. It's function is rudimentary, handling only the *reset vector* and enough to *jump* to the *ROMstage*. The *bootblock* code contains the first *instruction fetch* that is a *jump instruction* from 4GB to the initialisation code, then an immediate change to $32-$ bit *protected mode*. Once *protected mode* has been toggled, the *bootblock* provisions the minimal *north* and *south* bridge setup required to accomplish a *jump* to the *ROMstage*. The *jump instruction* and subsequent switch to *protected mode* can be found in:

`coreboot/src/cpu/x86/16bit`

1. At **0xFFFFFFF0**, start execution at **reset_vector** from **src/cpu/x86/16bit/reset16.inc**, which simply jumps to **_start**.

2. **_start** from **src/cpu/x86/16bit/entry16.inc**, invalidates the TLBs, sets up a GDT for selector 0x08 (code) and 0x10 (data), switches to protected mode, and jumps to **__protected_start** (setting the CS to the new selector 0x08). The selectors provide full flat access to the entire physical memory map.

3. **__protected_start** from **src/cpu/x86/32bit/entry32.inc**, sets all other segment registers to the 0x10 selector.

4. Execution continues with various mainboardinit fragments:

   - **__fpu_start** from **cpu/x86/fpu_enable.inc**.
   - **(unlabeled)** from **cpu/x86/sse_enable.inc**

- Some CPUs enable their on-chip cache to be used temporarily as a scratch RAM (stack), e.g. **cpu/amd/model_lx/cache_as_ram.inc**.

The *bootblock* code is written in C and is built into *stackless* assembler by a custom toolchain. The custom toolchain *ROMCC* is required due to the restrictive environment of a cold power-up of the hardware.

ROMstage

The *ROMstage* contains the early chipset and motherboard initialisation, Cache As RAM setup, memory initialisation, and decompression code for the coreboot *RAMstage*. Upon *ROMstage* completetion, *RAMstage* is executed in system memory.

The final mainboardinit fragment is mainboard-specific, in C, called **romstage.c**. For non-cache-as-RAM targets, it is compiled with **romcc**. It includes and uses other C-code fragments for:

1. Initializing MSRs, MTRRs, APIC.
2. Setting up the southbridge minimally ("early setup").
3. Setting up Super I/O serial.
4. Initializing the console.
5. Initializing RAM controller and RAM itself.

Execution continues at **__main** from **src/arch/x86/init/crt0_romcc_epilogue.inc**, where the non-romcc C coreboot code is copied (possibly decompressed) to RAM, then the RAM entry point is jumped to.

RAMstage

1. The RAM entry point is **_start** in **src/arch/x86/lib/c_start.S**, where new descriptor tables are set up, the stack and BSS are cleared, the IDT is initialized, and **hardwaremain()** is called (operation is now full 32-bit protected mode C program with stack).
2. **hardwaremain()** is from **src/boot/hardwaremain.c**, the console is initialized, devices are enumerated and initialized, configured and enabled.
3. The payload is called, either via **elfboot()** from **boot/elfboot.c**, or **filo()** from **boot/filo.c**.

1.2 Coreboot Development

Development Tools

- **Minicom** - terminal emulator
- **SAGE Embedded Development Kit (EDK)** - Eclipse-based integrated development environment (IDE).
- **SAGE SmartProbe** - Comprehensive hardware interface for controlling and viewing the system under development.

Source Development

The source is located in the following directory:

```
coreboot/src/vendorcode
```

..

Critical Hardware Information

SuperIO

The Super I/O chip is found on the mainboards which is responsible for the serial ports and so first thing to support. Since serial debugging output from the mainboard via a *null-modem* cable and *minicom* is critically important to bringing up other components in the process of the Coreboot port.

Adding support for a new Super I/O chip involves the following:

1. Add a directory **src/superio/vendor/device**.
2. In that directory, add a file **device_early_serial.c** where *device* is the actual part name.
3. In this file you now declare a function **device_enable_serial()** which enables the requested serial port.

The **device_early_serial.c** file is where the serial port on the mainboard is initialised. The serial output will work even before the RAM is first initialized, thus is required for debugging the RAM initialization code in a port.

An example of the **device_enable_serial()** function is given:

Listing 1.1: The **device_enable_serial()** function.

```
static void w83627ehg_enable_serial(device_t dev, unsigned int iobase)
{
        pnp_enter_ext_func_mode(dev);
        pnp_set_logical_device(dev);
        pnp_set_enable(dev, 0);
        pnp_set_iobase(dev, PNP_IDX_IO0, iobase);
        pnp_set_enable(dev, 1);
        pnp_exit_ext_func_mode(dev);
}
```

Mainboards which have this Super I/O chip, can then call this function in their **romstage.c** file. For example:

Listing 1.2: The **romstage.c** file.

```
#include "superio/winbond/w83627ehg/w83627ehg_early_serial.c"
[...]
#define SERIAL_DEV PNP_DEV(0x2e, W83627EHG_SP1)
[...]
w83627ehg_enable_dev(SERIAL_DEV, TTYS0_BASE);
uart_init();
console_init();
```

Remark. The Super I/O is usually at config address *0x2e* or *0x4e*, however is mainboard-dependent and so could be different. A way find out the address is by running the *superiotool*.

Getting further

You can reach the Coreboot project using the following communication means.

1.3 IRC

You can chat with us on IRC using the Freenode Network by joining the #coreboot or #flashrom channels.

1.4 Mailing-lists

You can get access to the mailing-lists at `https://lists.coreboot.com`. To ask questions about the something development please use the something-devel mailing list.

1.5 Website

The coreboot website is at `http://www.coreboot.org` has news and access to the source repositories.