A Coreboot step-by-step, Intel/D510MO

June 15, 2019

1 Contents:

2 What is coreboot?

Coreboot is an extensible open source hardware initialization firmware designed for desktops, laptops, servers, and embedded systems.
Coreboot is fast and secure thanks to it's minimal codebase. There are strict guidelines in the coreboot coding style.
Coreboot coding style assumes that the source code will be read and modified by a human. This makes porting and patching it far easier.
It supports loading payloads, eg. anything from a standard BIOS loader to loading the Linux kernel directly off the bios chip.

2.1 Pros:

Speed
Stability
Customizability
No device whitelist
Updated processor microcode
Open Source firmware for your computer
Management engine cleaner

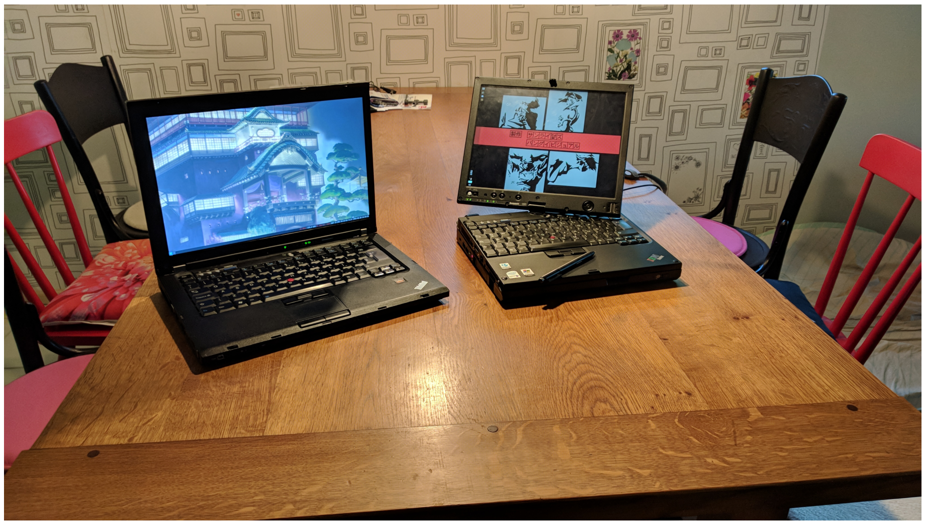
2.2 Cons:

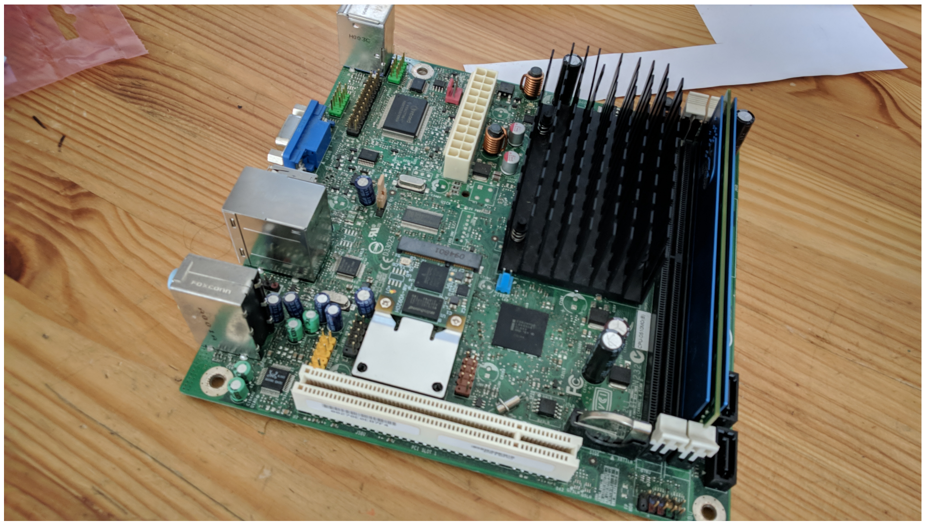
Limited Windows support
Vendor device driver issues
External installation
Bugs on early ports
No native overclocking

3 Acquiring prerequisites:

3.1 Mainboard:

For this i chose the Intel D510MO, it is in the mini ITX form factor which makes it great for small low power applications like a router or a NAS.
If you want a coreboot laptop, Thinkpads are recommended(have a look on the supported mainboards list).


Thinkpad T400 and X60 Tablet

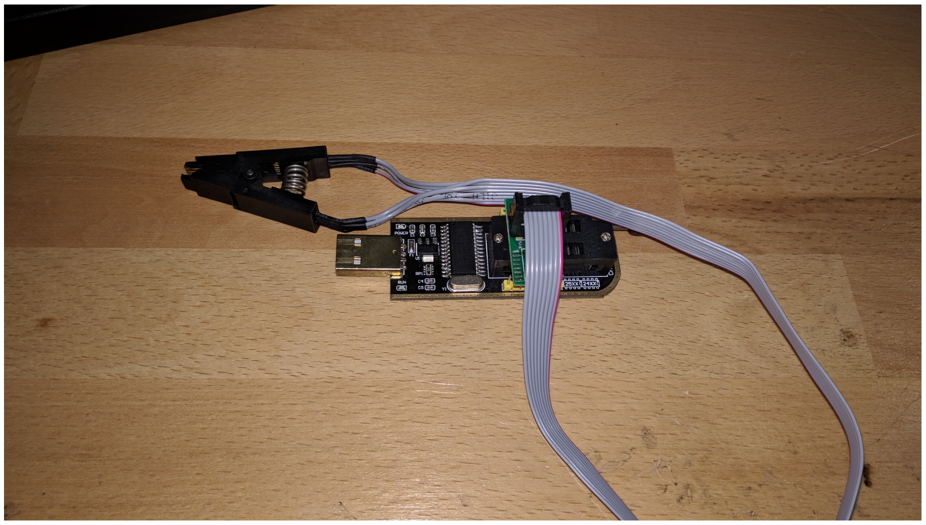

Intel D510MO

3.2 Build host:

The build host can be anything with GNU+Linux. Debian is the recommended distribution for this. Having at least 6 cpu threads and a solid state drive will help out a lot.

3.3 Programmer:

Most mainboards use standard SPI EEPROMs. If you are lucky your mainboard may have a DIP8 chip in a socket.
In this case the chip package is SOIC8, this reqires a programming clip or soldering. This package is often found on laptops.
A CH341a bundle with a clip is recommended. A single board computer like the Raspberry Pi can be used too.



CH341a with a SOIC8 clip

3.4 Vendor firmware backup:

Before proceeding any futher you will need a backup of the vendor firmware.
You might need to extract a few blobs from this backup.
This backup will also be used in case of a bricked mainboard.
To make a backup, look at the flashing section of this document.
Unlike writing a new image you will use the '-r' argument for flashrom. This will tell flashrom to read the image.
Make 3 backups(each with a different filename) and compare their md5sums. If they match open one in a hex editor to make sure there is data.
I have experienced when the chip just read blank.

4 Configuring coreboot

4.1 General cnfiguration:

For most mainboards the default configuration is fine.
In the 'General options' menu allowing the use of proprietary blobs might be necessary for a lot of mainboards.
Depending on if you want to use Windows or GNU+Linux you want to pick between native graphics initialization and VGAROMs.
Native graphics initialization is far faster and always uses the full resolution of the display. VGAROMs are proprietary and can be extracted via UEFITool from the backup of the vendor bios.

4.2 Payload:

There is multiple payloads to choose. Most people use SeaBIOS as it's the simplest.
SeaBIOS is an open source implementation of the legacy BIOS loader.
Tianocore is a much bigger EFI payload that is relatively slow, but it currently is the most stable prefered EFI loader.

4.3 Proprietary components:

Intel based mainboards after Nehalem require the Intel ME, you can strip it down either before or by enabling the me cleaner option in the configuration.
Microcode updates, while proprietary i recommned including them. Updated microcde also contains mitigations for exploits like meltdown. Some CPUs require them to boot at all.

4.4 External blobs:

Intel flash descriptor will be needed, it tells the chipset where to look for the BIOS, iME, and GBE region.
iME, you can use iME extracted from your vendor bios backup and strip it down.
GBE, GBE specifies your mac address. It can be extracted from the vendor bios.

5 Compiling coreboot

5.1 Toolchain:

To compile coreboot you need the toolchain. The utilities in this toolchain are versions verified to produce stable coreboot builds.
This toolchain can be compiled with ‘make crosstools-[your architecture] CPUS=[threads]‘

5.2 Coreboot itself

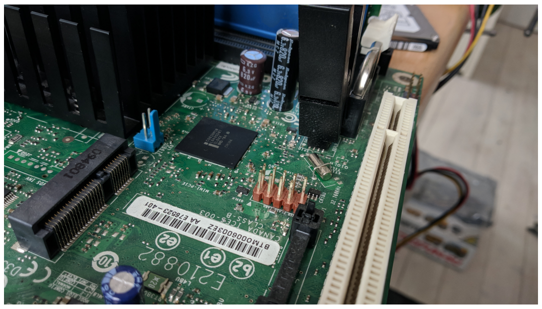
To compile coreboot itself run ‘make CPUS=[threads]‘.

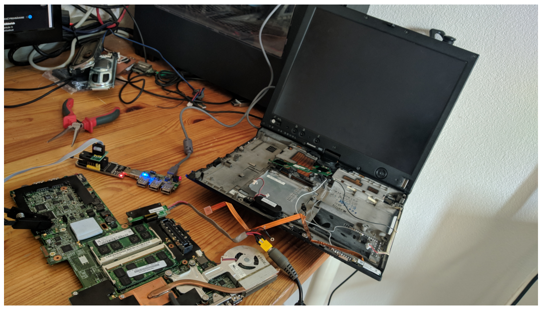
5.3 Troubleshooting

Sometimes you might encounter random errors, when that happens run ‘make distclean‘ to clean your build enviroment.
This command should also be ran when switching targets. WARNING: It will delete your .config!

6 Flashing coreboot

To flash your coreboot rom, connect your programmer via a soic clip or other means depending on your setup. I recommend using flashrom, this software supports many EEPROMs and is very easy to use. ‘flashrom -p ch341a -w build/coreboot.rom‘ Would be the command for my setup.
Some EEPROMs don’t get autodetected and you need to specify the model. Flashrom will throw up a list of possible options, read it off the chip and specify it with -c.
Certain EEPROMs require external power, if you are sure that your electrical connections are reliable and you get no EEROM detected and or failed write/erase functions you might have one of these chips. Macronix and ST chips experience this very often.
For mainboards with these chips, put the board into an S5 state and plug in the ethernet cable. The chip will get power because of WOL from the mainboard and should be sufficient.


Intel D510MO


Thinkpad X60