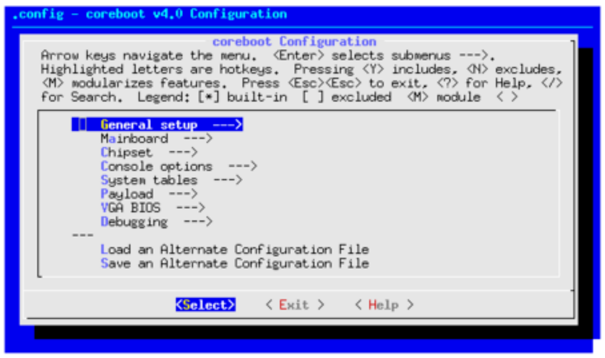
Build HOWTO

The wiki is being retired!

Documentation is now handled by the same processes we use for code: Add something to the Documentation/ directory in the coreboot repo, and it will be rendered to https://doc.coreboot.org/. Contributions welcome!

This page describes how you can build a coreboot image for your specific mainboard.



```
.config - coreboot v4.0 Configuration
                         coreboot Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

        [ ]  General setup  --->
             Mainboard  --->
             Chipset  --->
             Console options  --->
             System tables  --->
             Payload  --->
             VGA BIOS  --->
             Debugging  --->
        ---
             Load an Alternate Configuration File
             Save an Alternate Configuration File

                  <Select>    < Exit >    < Help >
```

make menuconfig in coreboot

Contents

Requirements
- debian

Building a payload (Optional)

Building coreboot
- Intel boards
- AMD boards
- Choose the payload
- VGA support
- Security Notes
- Compiling

Compiling with Clang/LLVM

Known issues

Development version

Flashing coreboot

Requirements

- gcc / g++ / gnat (gcc-multilib is ideal, makes building payloads a lot easier)
- make
- cmake (if using clang/llvm)
- ncurses-dev (for **make menuconfig**)
- iasl (for targets with ACPI support)
- flex and bison (for regenerating parsers)

Optional:

- doxygen (for generating/viewing documentation)

debian

```
apt-get install git build-essential gnat flex bison libncurses5-dev wget zlib1g-dev
```

Building a payload (Optional)

The majority of the payloads supported by coreboot are built automatically once they are selected and configured as described in the next section.

Most beginners want to use the default SeaBIOS payload. It runs Option ROMs, is able to discover boot devices and provides a very simple boot menu.

If, however, you need to build a payload that is currently not included in the coreboot build system:

- First you need to download the source code for the payload of your choice and build it.
- Each payload may have different build instructions and requirements, however most of the time a "make" command will suffice. Please check Payloads and the wiki page for the respective payload for details.
- The result of this step should be an ELF file (e.g. filo.elf, or coreinfo.elf) which you can use with coreboot (see below).

Building coreboot

First, get the latest coreboot version from our git repository:

```
$ git clone https://review.coreboot.org/coreboot
$ cd coreboot
$ git submodule update --init --checkout
```

The last step is important! It checks out a sub-repository in the 3rdparty directory.

It is worth checking the Supported Motherboards page to see if your mainboard is known to work with a particular version of coreboot. If your board is listed and you want to try a known-working version, click the "upstream tree" link to find the commit hash and then check it out in git:

```
$ git checkout <hash>
```

If you do this, run `git submodule update --checkout` again to make sure the 3rdparty modules are in sync with the sources.

Once you have the desired sources (and 3rdparty modules, if needed) checked out, you can configure the build-time options of coreboot:

```
$ make menuconfig
```

OR

```
$ make nconfig (easier to navigate, uses ncurses)
```

In that menu (which may look familiar, as other projects such as the Linux kernel or busybox use the same system), select at least the following options:

- Enter the **Mainboard** menu.
  - In **Mainboard vendor** select the vendor of your board.
  - In **Mainboard model** select your exact mainboard name.
  - In **ROM chip size** select the exact size of the flash ROM chip you want to flash the coreboot image on. (see output of `flashrom` command)

More detailed example (generic) configuration (tweak accordingly): (note that this assumes presence of native graphics initialization on the given board, which is not universally available in coreboot)

```
   General / Use CMOS for configuration values = enable (CMOS defaults are located in your boards
directory src/mainboard/OEM/MODEL/cmos.default)
   Mainboard / Mainboard vendor = Name of manufacturer
   Mainboard / Mainboard model = Model name
   Mainboard / ROM chip size = size of flash chip
   Chipset / Include CPU microcode in CBFS = Do not include microcode updates (NOTE: you probably want to
enable it on some systems)
   Devices / Use native graphics initialization = enable (NOTE: not available on all systems)
   Display / Keep VESA framebuffer = disable (disable for text-mode graphics, enable for coreboot vesa
framebuffer)
   Generic Drivers / USB 2.0 EHCI debug dongle support = Enable
   Generic Drivers / Enable early (pre-RAM) usbdebug = Enable
   Generic Drivers / Type of dongle = Net20DC or compatible
   Generic Drivers / Digitizer = Present
   Console / USB dongle console output = enable
   Payload / Add a payload = An ELF executable payload (change if you want a different payload)
   Payload / Payload path and filename = grub.elf (assumes building GRUB manually. Change this if you
want a different payload)
```

Now go back into Devices (only do this if you didn't enable native graphics initialization; NOTE: instructions for adding a vbios option rom are not mentioned in the above instructions):

```
   Devices / Run VGA Option ROMs = disable
   Devices / Run Option ROMs on PCI devices = disable
```

Intel boards

For Intel boards you have to provide files coreboot can't generate by itself:

- Intel Flash Descriptor region
- Intel Gigabit Ethernet firmware

- Intel Management Engine

Please have a look at Binary situation for a full overview. The files have to be extracted from your vendor bios.

- Enter the **Chipset** menu
  - Do the following based on which blobs you have:
  - Untick **Build with a fake IFD** (descriptor.bin)
  - Tick **Add gigabit ethernet firmware** (gbe.bin)
  - Tick **Add Intel Management Engine firmware** (me.bin)

AMD boards

For AMD boards you may have to provide files coreboot can't generate by itself:

- NIC firmware
- AMD IMC
- AMD XHCI
- AMD PSP (AMD's ME analog)

Please have a look at Binary situation for a full overview. The files have to be extracted from your vendor bios.

Choose the payload

Here's the full list of supported Payloads.

By default, the SeaBIOS payload will be downloaded and built during the coreboot build process.

If you want to use another payload (ELF for example):

- Enter the **Payload** menu.
  - Set the **Add a payload** option to **An ELF executable payload**.
  - Then, specify the file name and path to your payload file (which you built before).

VGA support

In order to see something on your screen the graphic card has to be initialized by the VGA BIOS which is actually an Option ROM.

VGA support is required for payloads such as GRUB or elf-memtest86+-5.01.

It isn't required for operating systems such as GNU/Linux as it initializes the graphic card by itself.

On some platforms there's support for native gfx init. A VGA BIOS isn't required.

Security Notes

On many modern X86 CPU's microcode updates are required for secure and proper CPU operation. Not including microcode updates may pose security risks and stability issues.

As an example, the 33xx, 43xx and 63xx "Piledriver" AMD Opteron CPUs have a fatal NMI to gain root exploit in some versions of the onboard microcode that is easily performed with userspace tool, while the "Bulldozer" CPUs can run without microcode updates.

Compiling

You also need to build the coreboot cross-compiler. This is to compile for the architecture of the platform you are building coreboot FOR, not the system you are building ON. For x86 systems, you currently would use the i386 cross-compiler, not the x64 version.

You can see all of the options available by running 'make help':

```
$ make help
...
*** Toolchain targets ***
 crossgcc        - Build coreboot cross-compilers for all platforms
 crosstools      - Build coreboot cross-compiler and GDB for all platforms
 crossgcc-clean  - Remove all built coreboot cross-compilers
 iasl            - Build coreboot IASL compiler (built by all cross targets)
 clang           - Build coreboot clang compiler
 test-toolchain  - Reports if toolchain components are out of date
 crossgcc-ARCH   - Build cross-compiler for specific architecture
 crosstools-ARCH - Build cross-compiler with GDB for specific architecture
 ARCH can be "i386", "x64", "arm", "aarch64", "mips", "riscv", "power8", "nds32le"
 Use "make [target] CPUS=#" to build toolchain using multiple cores
```

To build the cross-compilers for all architectures using 4 threads (This takes a LONG time):

```
$ make crossgcc CPUS=4
```

To build the cross-compiler for just the x86 architecture with just a single thread:

```
$ make crossgcc-i386
```

You can also invoke the cross compiler build script directly (in this example eight threads). But you probably don't want to, because the makefile builds other things you need too:

```
$ util/crossgcc/buildgcc -j 8
```

If something fails, the build should tell you what file to look in. You can also try to search for the relevant log (`find . -name '*.log' | xargs grep Error`) and examine last few lines of it.

That's the bare minimum. Feel free to adjust the other settings to your needs (see Coreboot Options for the full list), then exit menuconfig and build the coreboot image:

```
$ make
```

The file **build/coreboot.rom** is your final coreboot image you can flash onto a ROM chip or add payloads to with cbfstool.

Compiling with Clang/LLVM

We have been working on building coreboot with clang/llvm and it basically works. Remaining issues can be reported upstream and then block this meta bug here:

META Compiling the Coreboot with clang (http://llvm.org/bugs/show_bug.cgi?id=21691)

The default and recommended flow is still to use crossgcc.

Known issues

Make sure you really have all the requirements installed!

With certain versions of the gcc/ld toolchain shipped in some Linux distributions, it's possible that you'll see the following error when building coreboot:

```
src/arch/x86/coreboot_ram.ld:129 cannot move location counter backwards
```

This is a known bug in those versions of the toolchain. Before sending a complaint message to our mailing list, please try to switch to our reference cross-compilation toolkit then recompile the sources. To switch to the cross-compiler just run

```
$ make crossgcc
```

Then remove the **.xcompile** file and retry the compilation process:

```
$ rm .xcompile
$ make
```

Development version

If you want to contribute a patch or report an issue about coreboot, you will need to set up your environment for full development.

You **must** run **make crossgcc** and rebuild coreboot before reporting an issue or contributing a patch.

To get set up to submit a patch please run **make gitconfig**, then register with gerrit.

Flashing coreboot

You can flash the coreboot image on a flash ROM chip using either an external EEPROM-programmer or a mainboard using the flashrom (http://www.flashrom.org) user-space utility.

Retrieved from "https://www.coreboot.org/index.php?title=Build_HOWTO&oldid=34642"

This page was last edited on 3 May 2018, at 08:16.