How to Build Your Own BIOS Using coreboot

Reza Adinepour

July 19, 2025

Outline

Git

To build and configure coreboot, we need to install several packages:

- Development tools and version control
- Coreboot build dependencies
- Optional firmware tools

Step 1: Install Git (Ubuntu/Debian)

Install git

```
$ sudo apt install git
```

Step 2: Verify Git installation

Check version

```
$ git --version
```

Git (Cont.)

Step 3: Set user identity

Set username

```
$ git config --global user.name "Reza Adinepour"
```

Set email

```
$ git config --global user.email "reza@example.com"
```

Step 4: Set default text editor (optional)

Set vim as editor

```
$ git config --global core.editor "vim"
```

Python

Python is usually pre-installed on most Linux distributions.

Step 1: Check if Python is already installed

Check Python version

```
$ python3 --version
```

If not installed, install Python manually:

Install Python

```
$ sudo apt install python3 python3-pip
```

Installing Coreboot

Step 1: Install tools and libraries needed for coreboot

Install build dependencies

```
$ sudo apt-get install -y bison
       build-essential
       curl
       flex
       gnat
       libncurses-dev
       libssl-dev
       zlib1g-dev
       pkgconf
```

Installing Coreboot (Cont.)

Step 2: Clone the official coreboot repository

Clone coreboot

```
$ git clone https://review.coreboot.org/coreboot
```

Step 3: Enter the coreboot directory

Enter directory

```
$ cd coreboot
```

Step 4: Initialize submodules

Fetch submodules

```
$ git submodule update --init --recursive
```

Installing Coreboot (Cont.)

Step 5: Build the coreboot toolchain

Build

```
$ make crossgcc-i386 CPUS=$(nproc)
```

```
You can now run NASM from /home/rcdat/Downloads/coreboot/util/crossgcc/xgcc.
Welcome to the coreboot cross toolchain builder v2025-07-04_a9e97268fe

Building toolchain using 4 thread(s).

Downloading and verifying tarballs ...
 * acpica-unix-20250404.tar.gz (downloading from https://downloadmirror.intel.com/852044/a
)
Downloaded tarballs ... ok
Unpacking and patching ...
 * acpica-unix-20250404.tar.gz
   o acpica-unix-20250404_iasl.patch
Unpacked and patched ... ok
Building packages ...
Building IASL v20250404 for host ... ok
Packages built ... ok
Copied EDK2 tools template ... ok
Cleaning up temporary files... ok

You can now run IASL ACPI compiler from /home/rcdat/Downloads/coreboot/util/crossgcc/xgcc.
rcdat@rcdat:~/Downloads/coreboot$
```

Installing Coreboot (Cont.)

Step 6: Build the payload - coreinfo

Build

```
$ sudo apt-get install -y bison
$ make -C payloads/coreinfo
```

```
    CC          curses/colors.libcurses.o
    AR          /home/rcdat/Downloads/coreboot/payloads/coreinfo/libpayload/libcurses.a
    CC          arch/x86/boot_media.libcbfs.o
    CC          coreboot/src/commonlib/bsd/cbfs_private.libcbfs.o
    CC          coreboot/src/commonlib/bsd/cbfs_mcache.libcbfs.o
    CC          libcbfs/cbfs.libcbfs.o
    AR          /home/rcdat/Downloads/coreboot/payloads/coreinfo/libpayload/libcbfs.a
    CC          liblzma/lzma.liblzma.o
    AR          /home/rcdat/Downloads/coreboot/payloads/coreinfo/libpayload/liblzma.a
    CC          liblz4/lz4_wrapper.liblz4.o
    AR          /home/rcdat/Downloads/coreboot/payloads/coreinfo/libpayload/liblz4.a
    AR          /home/rcdat/Downloads/coreboot/payloads/coreinfo/libpayload/libpayload.a
  LDSCRIPT  /home/rcdat/Downloads/coreboot/payloads/coreinfo/libpayload/libpayload.ldscrip
    LPGCC       coreinfo.bin
/home/rcdat/Downloads/coreboot/util/crossgcc/xgcc/lib/gcc/i386-elf/14.2.0/../../../../i386
ack
/home/rcdat/Downloads/coreboot/util/crossgcc/xgcc/lib/gcc/i386-elf/14.2.0/../../../../i386
 the linker
/home/rcdat/Downloads/coreboot/util/crossgcc/xgcc/lib/gcc/i386-elf/14.2.0/../../../../i386
    DEBUG       coreinfo.debug
    STRIP       coreinfo.elf
make: Leaving directory '/home/rcdat/Downloads/coreboot/payloads/coreinfo'
```

Congratulations

You're Ready!

- Git and Python are configured
- Required packages and libraries are installed
- Coreboot source is cloned and ready
- You're now ready to start building and customizing coreboot firmware!

Good luck and happy hacking!

Coreboot Build Workflow Overview

Here are the general steps to build your custom BIOS:

1. Select your mainboard
2. Choose and configure the payload (e.g., SeaBIOS, Edk2)
3. Customize build options using `menuconfig`
4. Build the toolchain and coreboot image
5. Flash the image to the target board (with caution!)
6. Verify the boot process and debug if needed

Let's go step by step!

Select Your Mainboard

After downloading coreboot, the first configuration step is selecting your target mainboard.

Step 1: Open the configuration menu

Open coreboot menuconfig

```
$ make menuconfig
```

Step 2: Navigate to the mainboard selection

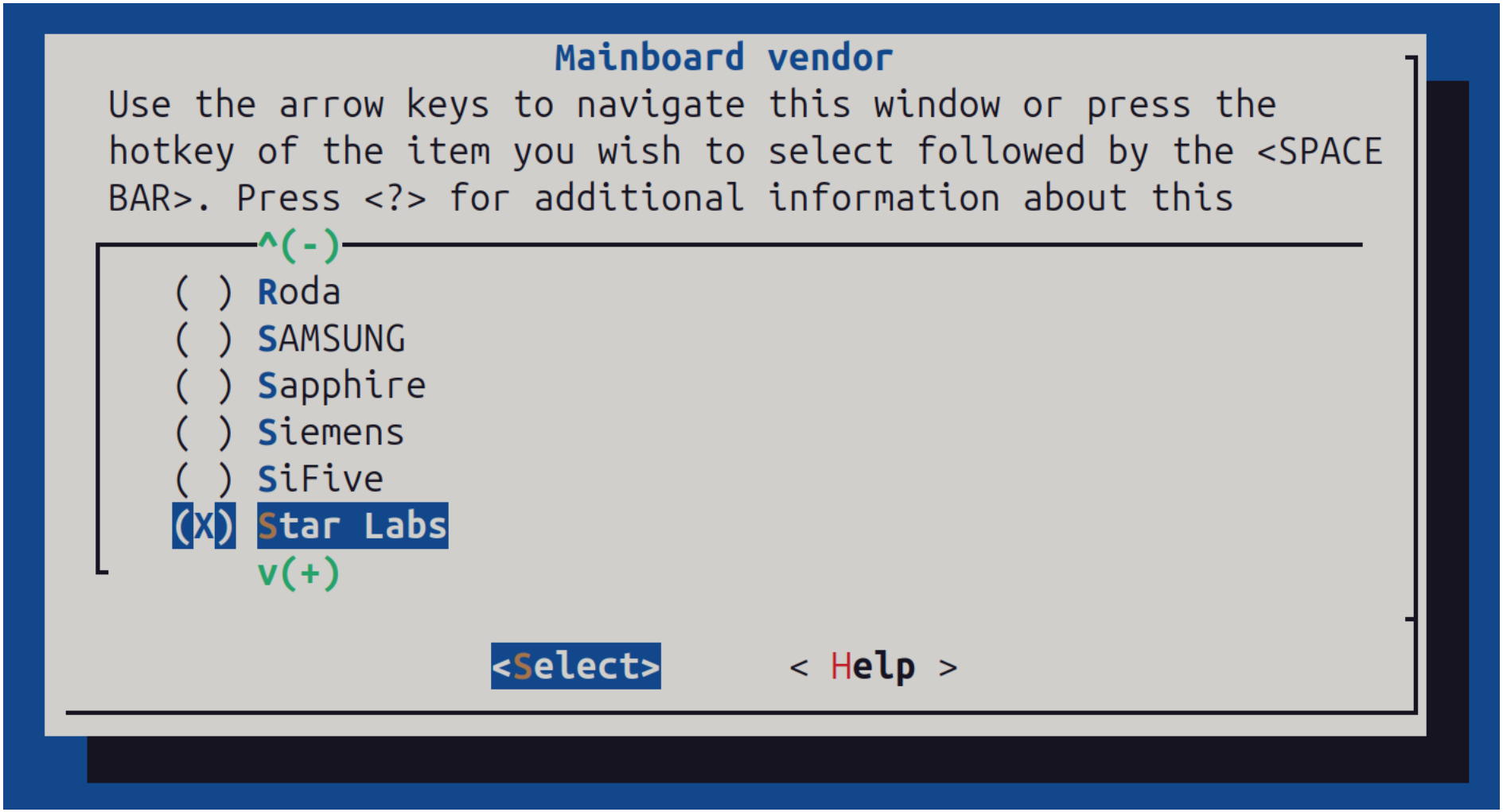
- Choose: `Mainboard` → `Mainboard vendor`
- Then choose your specific board model

Tip: You must know the exact vendor and model of your board before continuing.

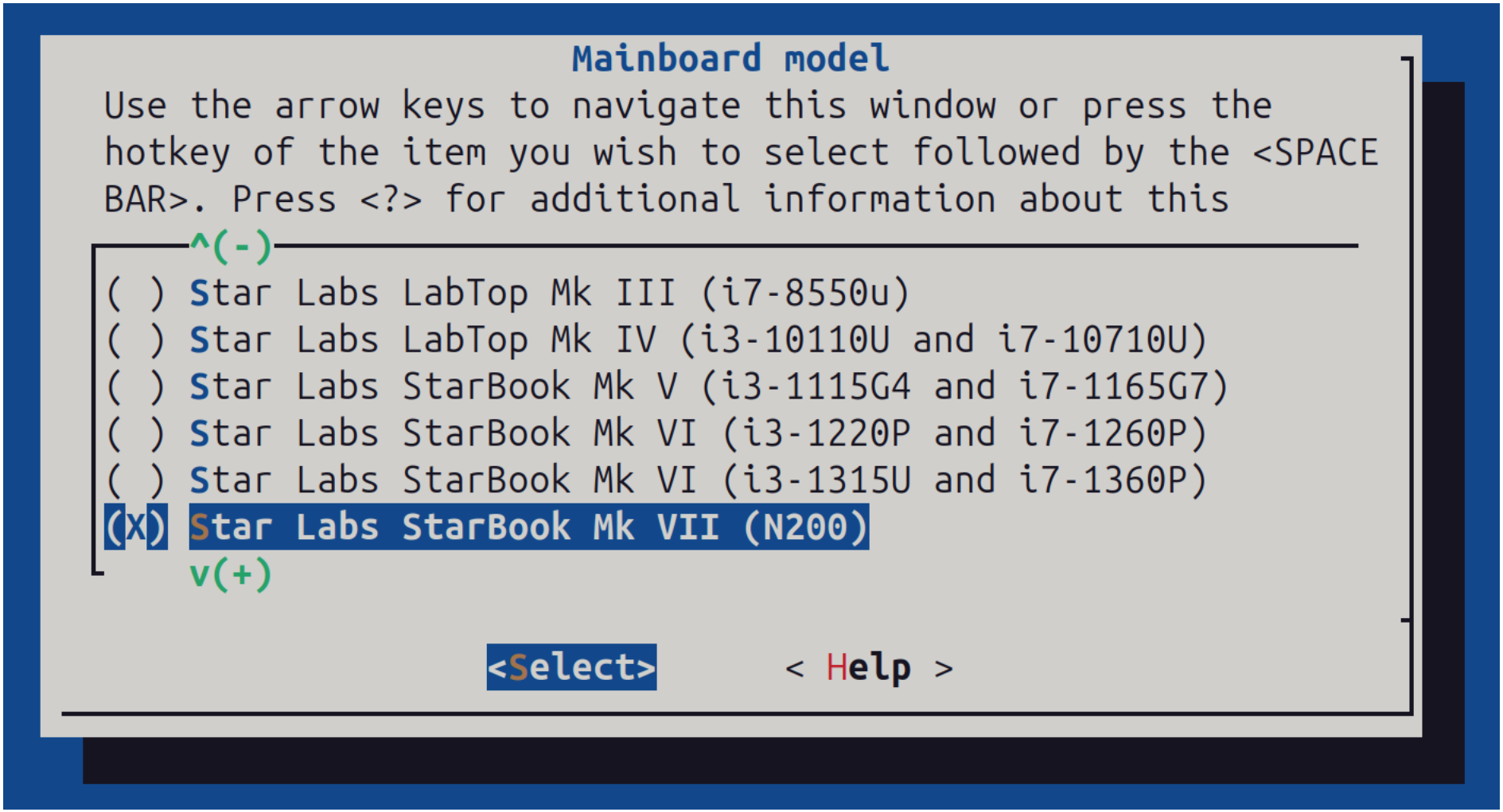
Select Your Mainboard (Cont.)

```
    General setup  --->
    Mainboard  --->
    Chipset  --->
    Devices  --->
    Generic Drivers  --->
    Security  --->
    Console  --->
    System tables  --->
    Payload  --->
    Debugging  --->
    Boot Logo Configuration  --->
```

Select Your Mainboard (Cont.)



Select Your Mainboard (Cont.)



Saving Configuration and Build

Step 1: Save your configuration after running `make menuconfig`

Save config to default location

Configuration is saved to: `.config`
using: `make savedefconfig`

Step 2: Display the saved config (optional)

View config

`$ cat defconfig` (or `cat .config`)

Step 3: Build coreboot

Start build

```
$ make -j$(nproc)
```

Fix Python Error

After building, you might see below error:

Error message

/bin/sh: 1: python: not found

```
    CC          ramstage/lib/bootmode.o
    CC          ramstage/lib/cbfs.o
    CC          ramstage/lib/cbmem_common.o
    CC          ramstage/lib/cbmem_console.o
    CC          ramstage/lib/coreboot_table.o
  Building ld scripts
/bin/sh: 1: python: not found
make[2]: *** [Makefile:169: out/romlayout16.lds] Error 127
make[1]: *** [Makefile:84: build] Error 2
make: *** [payloads/external/Makefile.mk:66: payloads/external/SeaBIOS/seabios/out/bios.bin.elf] Error 2
make: *** Waiting for unfinished jobs....
rcdat@rcdat:~/Downloads/coreboot$
```

Fix Python Error (Cont.)

Cause:

- Some build scripts (e.g., SeaBIOS) expect the command `python` to exist.
- On modern Linux systems, only `python3` is installed by default.
- The legacy `python` command is missing, causing the build to fail.

Solution: Create the `python` symlink pointing to `python3`

Install compatibility package

```
$ sudo apt install python-is-python3
```

This will make `python` point to `python3`, fixing the error.

Alternative (if package not available):

Manual symlink (advanced)

```
$ sudo ln -s /usr/bin/python3 /usr/bin/python
```

Fixing EDK2 Payload Build Error

Problem: Missing tools required by EDK2 (TianoCore UEFI payload)
Common Errors:

NASM not found

EDK2: Checking nasm: Not found!
ERROR: Please install nasm.

Missing NSS (optional)

Missing NSS. PKCS11 signing not supported. Install libnss3 to enable this feature.

Fixing EDK2 Payload Build Error (Cont.)

```
EDK2: Checking nasm:        Not found!
ERROR: Please install nasm.
make[1]: *** [Makefile:253: checktools] Error 1
make[1]: *** Waiting for unfinished jobs....
Makefile:308: Missing NSS. PKCS11 signing not supported. Install libnss3 to enable this feature.
vboot SHA256 built with tight loops (slower, smaller code size)
    AR          ramstage/external/vboot_reference-ramstage/vboot_fw.a
    AR          postcar/external/vboot_reference-postcar/vboot_fw.a
    AR          romstage/external/vboot_reference-romstage/vboot_fw.a
    AR          bootblock/external/vboot_reference-bootblock/vboot_fw.a
    LINK        cbfs/fallback/postcar.debug
    LINK        cbfs/fallback/romstage.debug
    LINK        cbfs/fallback/bootblock.debug
    LINK        cbfs/fallback/ramstage.debug
    OBJCOPY     cbfs/fallback/romstage.elf
    OBJCOPY     cbfs/fallback/bootblock.elf
    OBJCOPY     bootblock.raw.elf
    OBJCOPY     bootblock.raw.bin
    Checking out edk2 revision origin/uefipayload_2502
HEAD is now at feaf6b976b UefipayloadPkg/SmmStoreLib: Set capabilities for store region
make: *** [payloads/external/Makefile.mk:160: build/UEFIPAYLOAD.fd] Error 2
rcdat@rcdat:~/Downloads/coreboot$
```

Fixing EDK2 Payload Build Error (Cont.)

Solution: Install required packages

Fix command (Ubuntu/Debian)

```
$ sudo apt install nasm uuid-dev iasl build-essential
libnss3
```

Then rebuild again:

Retry the build

```
$ make -j$(nproc)
```

Build Successful

Congratulations!

Coreboot has been successfully built with your selected configuration and payload.

```
Built starlabs/starbook (StarBook Mk VII)

        ** WARNING **
coreboot has been built without the ITE EC Firmware.
Do not flash this image.  Your laptop's power button
may not respond when you press it.


        ** WARNING **
ADD_FSP_BINARIES isn't selected even though this SoC relies on the FSP.
The resulting image won't contain the FSP binaries and will not boot unless
they are added later.

        ** WARNING **
coreboot has been built without an Intel Firmware Descriptor.
Never write a complete coreboot.rom without an IFD to your
board's flash chip! You can use flashrom's IFD or layout
parameters to flash only to the BIOS region.

rcdat@rcdat:~/Downloads/coreboot$
```

Post-Build Warnings (Read Carefully)

- **Missing ITE EC Firmware:** Power button may stop working. Do not flash this image as-is.
- **FSP Binaries Not Included:** SoC requires FSP. Image will not boot without it.
- **No Intel Firmware Descriptor (IFD):** Never flash full ROM without IFD! Use `flashrom` with layout or region flags.

Fix these issues before flashing the image.

Adding FSP and IFD to Your Coreboot Image

To ensure your image boots safely:

- **FSP (Firmware Support Package):**
  - Download FSP binary from Intel or vendor.
  - Enable `ADD_FSP_BINARIES` in `make menuconfig`.
  - Place FSP file in `3rdparty/blobs/...` as instructed.
- **IFD (Intel Firmware Descriptor):**
  - Extract it from your original ROM using:

Extract IFD from dump

```
$ ifdtool -x original.rom
```

  - Merge it into your Coreboot build using:

Insert IFD

```
$ ifdtool -i fd:flashregion_0_descriptor.bin coreboot.rom
```

Only flash after confirming both FSP and IFD are present.

Flashing Coreboot Image Safely

Never overwrite the entire flash chip if IFD is missing!

Step 1: Dump and inspect your original ROM

Backup current firmware

```
$ flashrom -p <programmer> -r [YOUR/PATH/]/original.rom
```

Step 2: Extract and reuse layout

Extract layout (optional)

```
$ ifdtool -x original.rom
```

Flashing Coreboot Image Safely (Cont.)

Step 3: Flash only the BIOS region

Flash BIOS region safely

```
$ flashrom -p <programmer> -w coreboot.rom
      --ifd -i bios
```

Use external flasher (e.g. CH341A) if unsure or testing first time.

First Boot: Verifying Coreboot Works

After flashing, verify that Coreboot boots successfully.
Signs of success:

- System powers on and screen initializes
- Payload (e.g., SeaBIOS, Tianocore) shows a boot screen
- You can boot into OS or see a boot device menu

For deeper verification:

- Use UART/serial output for debug logs
- Check if EC (Embedded Controller) responds to power/reset
- Confirm BIOS region was correctly flashed with `flashrom -v`

First Boot: Verifying Coreboot Works (Cont.)

If system does not boot:

- Reflash original firmware backup (Externally)
- Recheck EC firmware, FSP, and IFD
- Review serial logs if available

Questions?

Any Questions?

Thank you for your attention!

Slides available at:

https://github.com/rezaAdinepour/ISC-Workshop.git