Google

Building and Flashing Chrome OS AP Firmware:

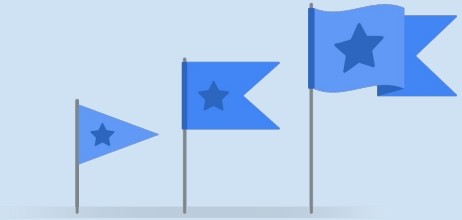
A Hands-On Dive into Coreboot

POCs:

rishikaraj@, digehlot@, jayvik@, pranavayn@

OOSC'24
August 2024

Objectives



This workshop offers a hands-on exploration of the Application Processor (AP) firmware in Chromebooks, traditionally referred to as the BIOS. Participants will delve into the open-source “coreboot” framework, gaining practical experience in building custom AP firmware images.

Recap

- *What is firmware?*
- *Chromebook EC, AP and GSC firmware*

Real hardware demo

- *Hardware requirements*
- *Host & DUT connection*
- *Firmware development and debugging*
- *Firmware flashing*
- *Different consoles and system tools*

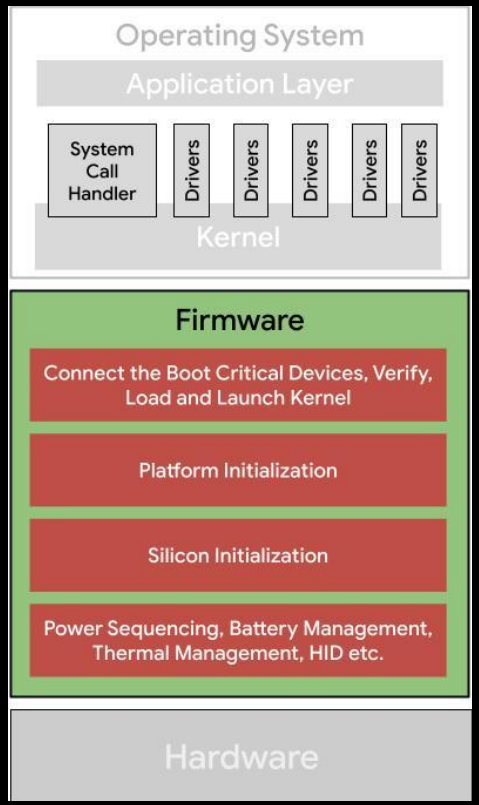
QEMU demo

- *Understanding QEMU*
- *Steps to build the Coreboot QEMU demo*
- *Workshop*

First step towards Open Source Firmware

https://tinyurl.com/rvxdz792

What is Firmware?



As per IEEE 610.12-1990, the definition of Firmware is:

The *tiny* block that combines hardware device, computer instructions and data, reside as *read-only* software on the device.

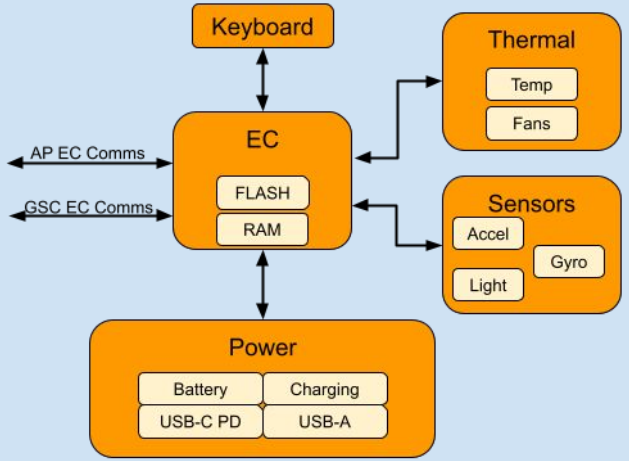
"Tiny" and *"read-only"* terms are misleading while defining the scope of the modern firmware.

An *essential* piece of code that is responsible for performing the underlying hardware initialization prior to handing over to the operating system.

EC Firmware

EC refers to the Embedded Controller. Major responsibilities of EC firmware are,

- AP Power Management
- Peripheral Management
- Keyboard Control
- Thermal Management
- Power Management

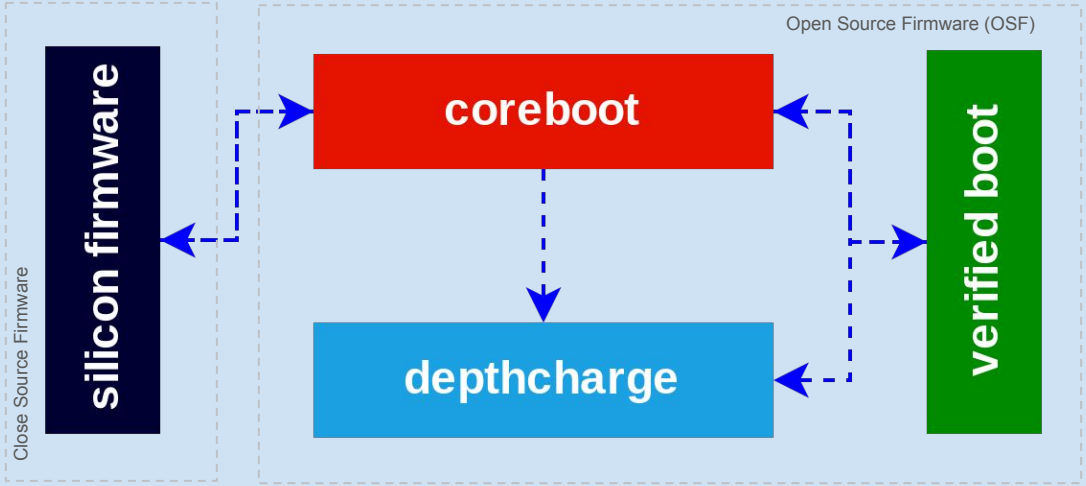


AP Firmware

AP refers to the Application Processor or the CPU, AP Firmware is the firmware running on the AP.

Components:

- coreboot
- depthcharge
- verified boot (vboot)
- silicon firmware



GSC (Google Security Chip)

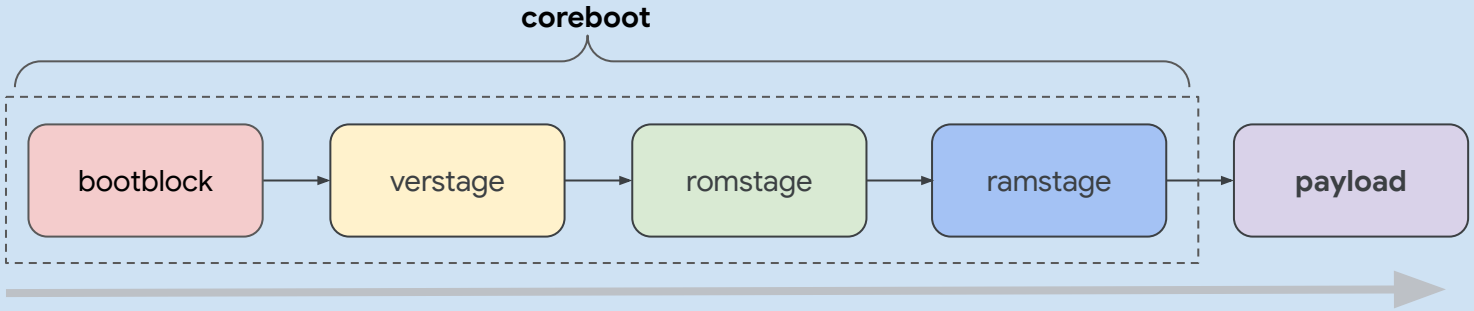
Ti50 is the firmware that runs on the GSC, it has support for **Case Closed Debugging** (CCD).

- Exposes EC, AP and GSC consoles
- Controls hardware write protect
- Allows flashing of EC and AP firmware

coreboot

coreboot is an open source firmware platform that delivers a lightning fast and secure boot experience on modern computers.

- Minimal hardware initialization
- RAM initialization, setting up the I/O devices etc.
- Not a bootloader, but supports various Payloads
- Payloads can boot OS

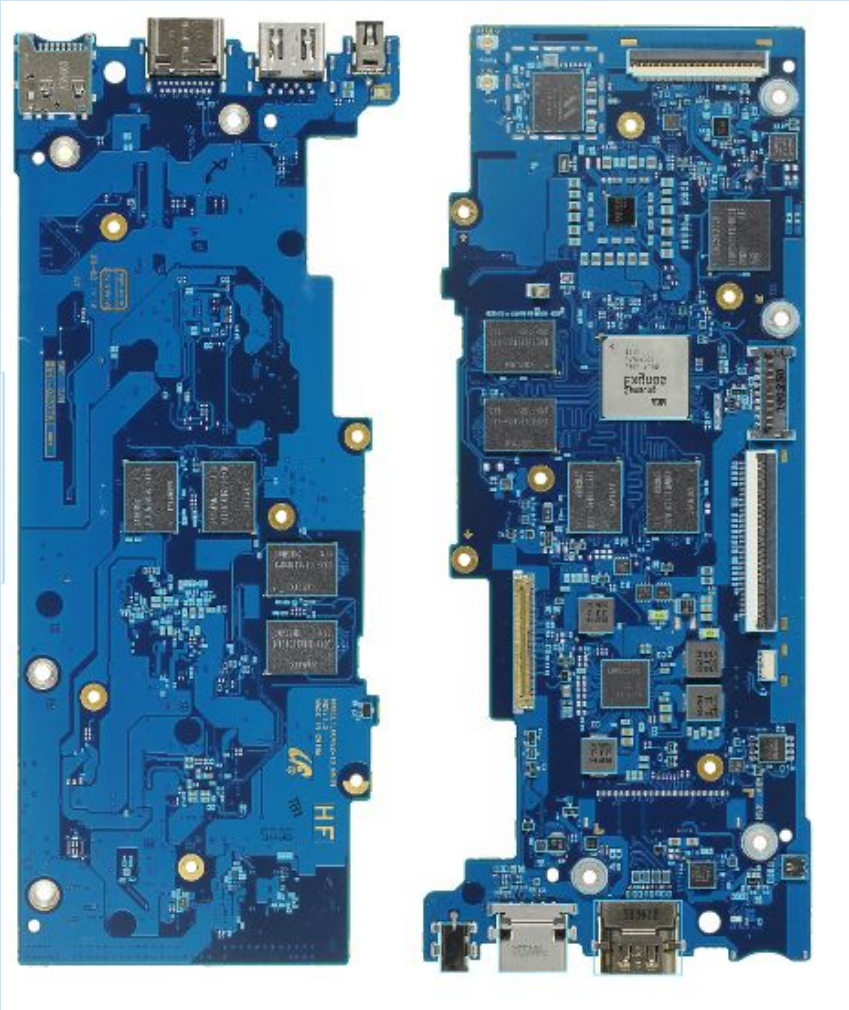


Typical AP FW Boot Flow

“OK, I got the concept...

but how does this look in practice?”

Real Hardware Demo



Hardware Requirements

Host

Host is the development machine

It can be:

- **Linux machine**

or

- **Chromebook** running Linux
  - No VM required

DUT - Device Under Test

A Chromebook under development or testing

- In developer mode
- CCD open (enabled)

SuzyQ

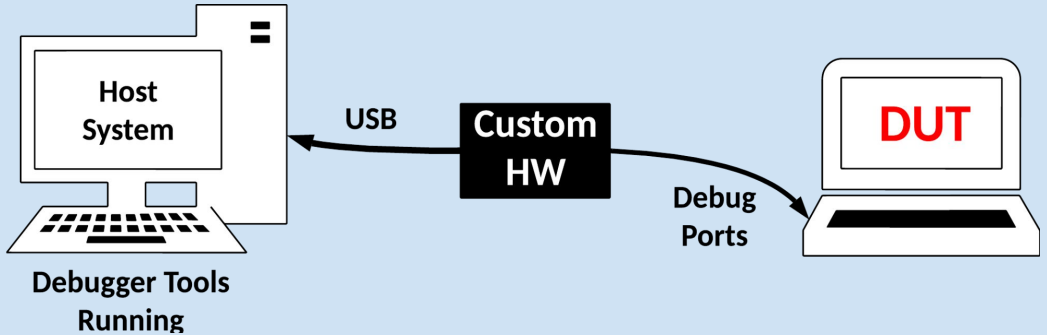
SuzyQ is a debug cable used for ChromeOS firmware development and testing

- Connects to USB-C port on the Chrome OS device
- Serial ports - AP, EC and GSC consoles
- SPI Flash read/write - AP, EC and GSC
- Power control - on/off/reset
- Misc other controls

Host & DUT Connection

Common interfaces include:

- **USB:** Used for data transfer, power supply, and debugging.
- **Serial:** Used for low-level debugging and accessing logs.
- **JTAG:** Used for hardware-level debugging and programming.
- **Network:** Used for remote debugging and accessing the DUT's file system.



Firmware Development/Debugging

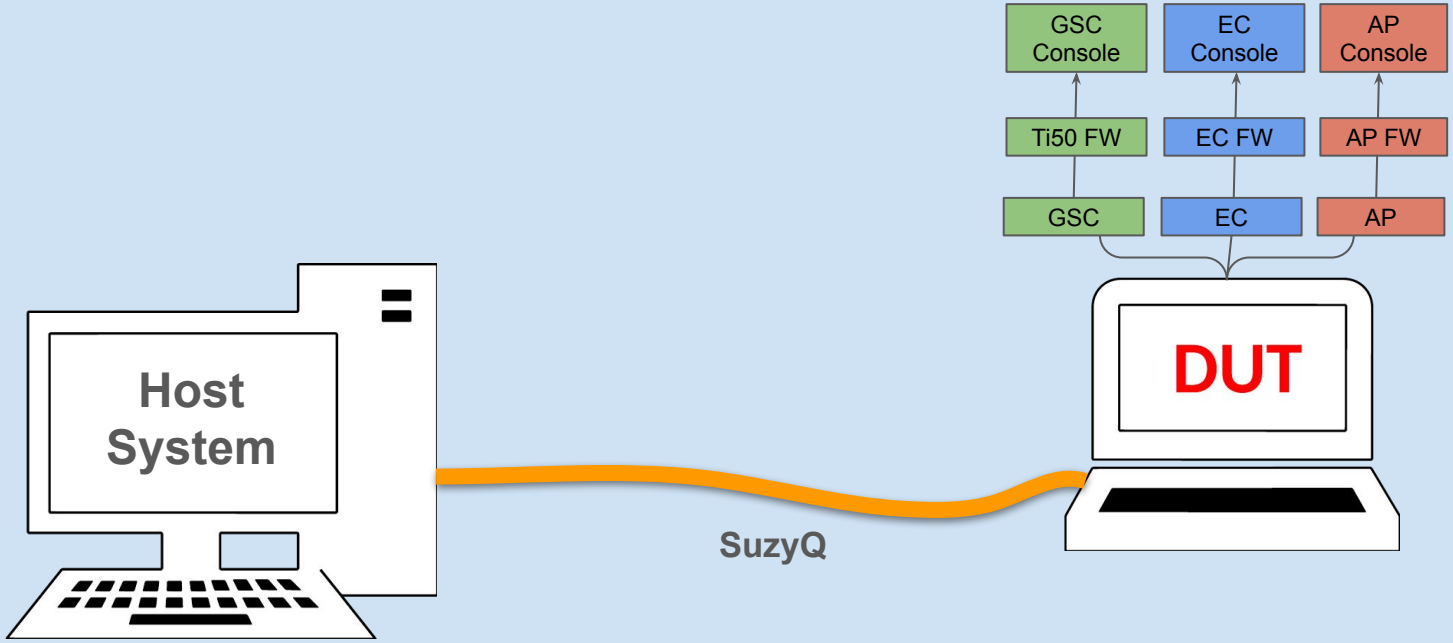
- **Firmware Flashing:** Update or modify the DUT's firmware using specialized tools like flashrom or the Chrome OS Firmware Utility.
- **Console Access:** Access the DUT's console to view boot messages, kernel logs, and interact with the system through a CLI.
- **Log Analysis:** Extract system logs using tools like elogtool (for ChromeOS).
- **Hardware-Level Debugging:** Access the DUT's internal registers and memory, set breakpoints, single-step through code execution, and examine the state of the hardware in real-time.
- **System Tools:** Tools like crossystem (for ChromeOS) or custom scripts can be used to monitor system parameters like CPU usage, temperature, battery status, and more.

flashrom

- Utility for identifying, reading, writing, verifying and erasing flash chips.
- Designed to flash BIOS/firmware images on mainboards, network/graphics/storage controller cards, and various other programmer devices.

https://github.com/flashrom/flashrom

Consoles


GSC Console
EC Console
AP Console
Ti50 FW
EC FW
AP FW
GSC
EC
AP
DUT
Host System
SuzyQ

GSC Console

```
OPTIONS: I18n
Port /dev/modem

Press CTRL-A Z for help on special keys

24-08-20 06:59:16.250
24-08-20 06:59:16.250 >
24-08-20 06:59:16.452 >
24-08-20 06:59:16.606 > ccd
24-08-20 06:59:19.525 State: Opened
24-08-20 06:59:19.525 Password: none
24-08-20 06:59:19.525 Flags: 0x400005
24-08-20 06:59:19.525 Capabilities: 5555555555010000
24-08-20 06:59:19.525   UartGscRxAPTx   Y 1=Always
24-08-20 06:59:19.525   UartGscTxAPRx   Y 1=Always
24-08-20 06:59:19.525   UartGscRxECTx   Y 1=Always
24-08-20 06:59:19.525   UartGscTxECRx   Y 1=Always
24-08-20 06:59:19.525   FlashAP         Y 1=Always
24-08-20 06:59:19.525   FlashEC         Y 1=Always
24-08-20 06:59:19.530   OverrideWP      Y 1=Always
24-08-20 06:59:19.530   RebootECAP      Y 1=Always
24-08-20 06:59:19.530   GscFullConsole  Y 1=Always
24-08-20 06:59:19.530   UnlockNoReboot  Y 1=Always
24-08-20 06:59:19.530   UnlockNoShortPP Y 1=Always
24-08-20 06:59:19.530   OpenNoTPMWipe   Y 1=Always
24-08-20 06:59:19.530   OpenNoLongPP    Y 1=Always
24-08-20 06:59:19.530   BatteryBypassPP Y 1=Always
24-08-20 06:59:19.530   Unused          Y 1=Always
24-08-20 06:59:19.531   I2C             Y 1=Always
24-08-20 06:59:19.531   FlashRead       Y 1=Always
24-08-20 06:59:19.531   OpenNoDevMode   Y 1=Always
24-08-20 06:59:19.531   OpenFromUSB     Y 1=Always
24-08-20 06:59:19.531   OverrideBatt    Y 1=Always
24-08-20 06:59:19.531   APROCheckVC     Y 1=Always
24-08-20 06:59:19.531 [509981.779447 Console unlock allowed]
24-08-20 06:59:19.531 TPM:
24-08-20 06:59:19.531 Capabilities are modified.
24-08-20 06:59:19.531 Use 'ccd help' to print subcommands
24-08-20 06:59:19.531 >
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.9 | VT102 | Offline | pts/6
```

Google

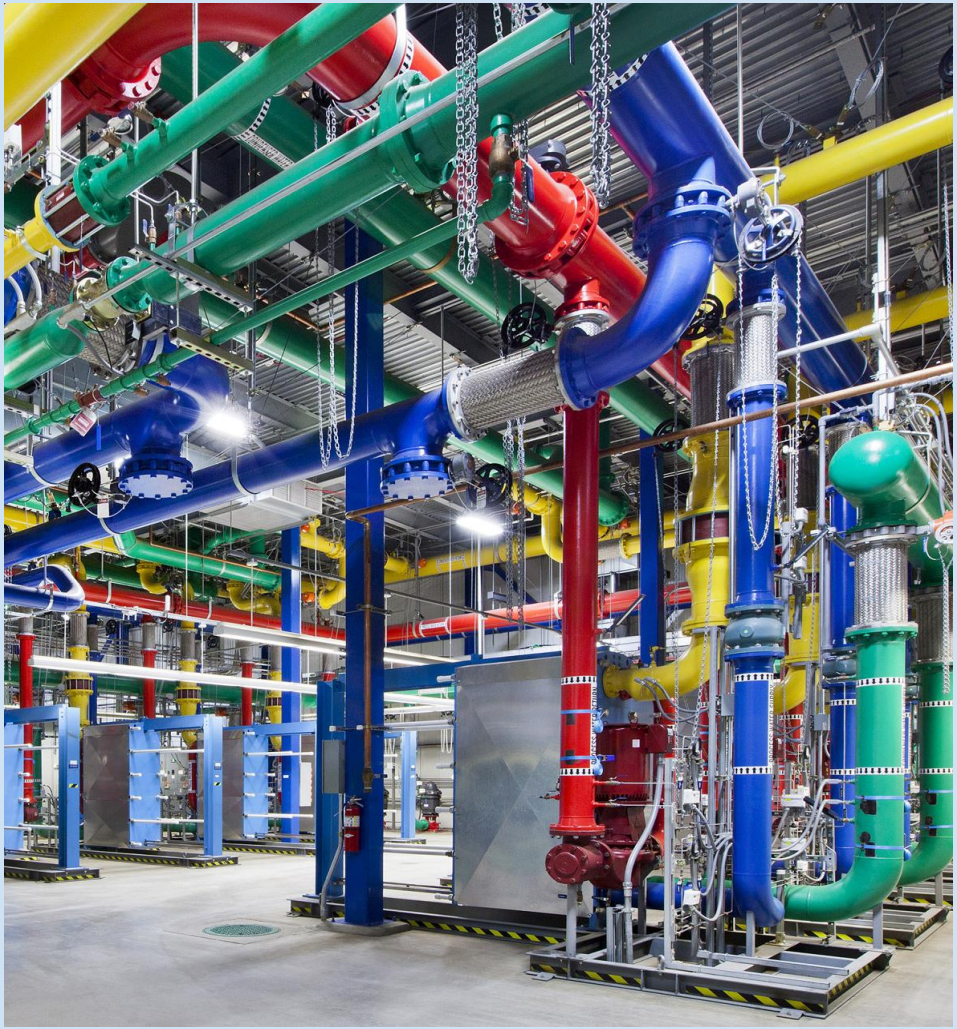
EC Console

```
24-08-20 06:56:36.203 > battery
24-08-20 06:56:44.083   Status:       0x00e0 FULL DCHG INIT
24-08-20 06:56:44.087   Param flags:  00000002
24-08-20 06:56:44.091   Temp:         0x0bc5 = 301.3 K (28.2 C)
24-08-20 06:56:44.095   V:            0x3305 = 13061 mV
24-08-20 06:56:44.098   V-desired:    0x0000 = 0 mV
24-08-20 06:56:44.101   I:            0xff57 = -169 mA(DISCHG)
24-08-20 06:56:44.110   I-desired:    0x0000 = 0 mA
24-08-20 06:56:44.110   Charging:     Not Allowed
24-08-20 06:56:44.110   Charge:       95 %
24-08-20 06:56:44.112     Display:    98.0 %
24-08-20 06:56:44.112   Manuf:        LGC KT0030G024
24-08-20 06:56:44.113   Device:       AP19B8M
24-08-20 06:56:44.116   Chem:         LION
24-08-20 06:56:44.119   Serial:       0x8cc1
24-08-20 06:56:44.119   V-design:     0x2d5a = 11610 mV
24-08-20 06:56:44.122   Mode:         0x6081
24-08-20 06:56:44.126   Abs charge:   100 %
24-08-20 06:56:44.126   Remaining:    4701 mAh
24-08-20 06:56:44.130   Cap-full:     4942 mAh
24-08-20 06:56:44.133     Design:     4683 mAh
24-08-20 06:56:44.133   Charge Cycle: 18
24-08-20 06:56:44.135   Time-full:    0h:0
24-08-20 06:56:44.138     Empty:      24h:21
24-08-20 06:56:44.138   Full Factor:  0.97
24-08-20 06:56:44.142   Shutdown SoC: 4 %
24-08-20 06:56:44.142   C-FET:        1
24-08-20 06:56:44.145 > [1546.203887 KB disable_scanning_mask changed: 0x00000002]
24-08-20 06:56:58.409 [1546.204376 KB Clear Buffer]
24-08-20 06:56:58.434 [1546.234117 power button pressed]
24-08-20 06:56:58.434 [1546.234850 SW 0x07]
24-08-20 06:56:58.436 [1546.235156 PB pressed]
24-08-20 06:56:58.440 [1546.235474 PB task 1 = pressed]
24-08-20 06:56:58.443 [1546.235834 PB PCH pwrbtn=LOW]
24-08-20 06:56:58.446 [1546.237120 PB task 2 = t0, wait 30355]
24-08-20 06:56:58.450 [1546.237579 mkbp buttons: 1]
24-08-20 06:56:58.467 [1546.268083 PB task 2 = t0]
24-08-20 06:56:58.467 [1546.268451 PB PCH pwrbtn=HIGH]
24-08-20 06:56:58.473 [1546.268786 PB task 3 = t1, wait 3967297]
24-08-20 06:56:58.550 [1546.351751 KB disable_scanning_mask changed: 0x00000000]
24-08-20 06:56:58.554 [1546.352349 power button released]
```

Google

AP Console

```
omnigul-rev2 ~ # cbmem -t
81 entries total:

 990:CSME ROM started execution                      0
 944:CSE sent 'Boot Stall Done' to PMC               119,000
 945:CSE started to handle ICC configuration         119,000 (0)
 946:CSE sent 'Host BIOS Prep Done' to PMC           123,000 (4,000)
 947:CSE received 'CPU Reset Done Ack sent' from PMC 281,000 (158,000)
   0:1st timestamp                                   320,718 (39,718)
  11:start of bootblock                              331,007 (10,289)
  12:end of bootblock                                400,753 (69,745)
   5:start of verified boot                          400,849 (96)
 503:starting to initialize TPM                      401,473 (624)
 504:finished TPM initialization                     544,477 (143,003)
 505:starting to verify keyblock/preamble (RSA)      604,436 (59,958)
 506:finished verifying keyblock/preamble (RSA)      713,434 (108,998)
 507:starting to verify body (load+SHA2+RSA)         722,610 (9,176)
 508:finished loading body                           1,157,041 (434,430)
 509:finished calculating body hash (SHA2)           1,185,780 (28,738)
 510:finished verifying body signature (RSA)         1,205,153 (19,372)
 511:starting TPM PCR extend                         1,209,390 (4,237)
 512:finished TPM PCR extend                         1,307,722 (98,331)
 513:starting locking TPM                            1,307,722 (0)
 514:finished locking TPM                            1,326,989 (19,267)
   6:end of verified boot                            1,349,834 (22,845)
  13:starting to load romstage                       1,365,495 (15,660)
  14:finished loading romstage                       1,390,678 (25,183)
   1:start of romstage                               1,398,570 (7,892)
 515:starting EC software sync                       1,413,076 (14,506)
 516:EC vboot hash ready                             1,489,391 (76,314)
 517:waiting for EC to allow higher power draw       1,513,879 (24,488)
 518:finished EC software sync                       1,520,619 (6,739)
 948:starting CSE firmware sync                      1,591,125 (70,505)
 949:finished CSE firmware sync                      1,663,621 (72,495)
 970:loading FSP-M                                   1,679,128 (15,507)
   2:before RAM initialization                       1,687,411 (8,282)
 950:calling FspMemoryInit                           1,822,666 (135,254)
 951:returning from FspMemoryInit                    1,865,222 (42,555)
 550:starting to load ChromeOS VPD                   1,877,946 (12,724)
   3:after RAM initialization                        1,942,223 (64,277)
   4:end of romstage                                 2,017,857 (75,634)
 100:start of postcar                                2,060,820 (42,962)
 101:end of postcar                                  2,060,820 (0)
   8:starting to load ramstage                       2,063,737 (2,916)
  15:starting LZMA decompress (ignore for x86)       2,097,841 (34,104)
  16:finished LZMA decompress (ignore for x86)       2,136,416 (38,575)
   9:finished loading ramstage                       2,153,429 (17,012)
  10:start of ramstage                               2,175,408 (21,979)
 971:loading FSP-S                                   2,278,749 (103,340)
  17:starting LZ4 decompress (ignore for x86)        2,287,129 (8,380)
```



System Tools

elogtool

- Retrieve and analyze system logs from a ChromeOS device to troubleshoot hardware or software issues.
- **Example:** elogtool list to collect and display the latest system logs.

```
omnigul-rev2 ~ # elogtool list
0 | 2024-08-18 11:29:42-0700 | Log area cleared | 4088
1 | 2024-08-18 11:29:42-0700 | Memory Cache Update | Normal | Success
2 | 2024-08-18 11:30:00-0700 | System boot | 145
3 | 2024-08-18 11:30:00-0700 | Firmware Splash Screen | Enabled
4 | 2024-08-18 11:30:02-0700 | System Reset
5 | 2024-08-18 11:30:02-0700 | Firmware vboot info | boot_mode=Developer | fw_tried=A | fw_try_count=0 | fw_prev_tried=A
omnigul-rev2 ~ #
```

https://github.com/coreboot/coreboot/blob/main/util/cbfstool/elogtool.c

crossystem

- Interact with and manage various aspects of ChromeOS devices from the command line.
- **Example:** crossystem dev_boot_usb to enable booting from a USB drive on a Chromebook.

```
omnigul-rev2 ~ # crossystem
act_fwver               = 0x00010001                # [RO/int] Active firmware version
act_kernver             = 0x00010001                # [RO/int] Active kernel version
arch                    = x86                       # [RO/str] Platform architecture
backup_nvram_request    = 1                         # [RW/int] Backup the nvram somewhere at the next boot. Cleared on success.
battery_cutoff_request  = 0                         # [RW/int] Cut off battery and shutdown on next boot
block_devmode           = 0                         # [RW/int] Block all use of developer mode
board_id                = 2                         # [RO/int] Board hardware revision number
clear_tpm_owner_done    = 0                         # [RW/int] Clear TPM owner done
clear_tpm_owner_request = 0                         # [RW/int] Clear TPM owner on next boot
cros_debug              = 1                         # [RO/int] OS should allow debug features
dbg_reset               = 0                         # [RW/int] Debug reset mode request
debug_build             = 1                         # [RO/int] OS image built for debug features
```

https://github.com/coreboot/vboot/blob/main/utility/crossystem.c

cbmem

- Access and analyze coreboot CBMEM (coreboot memory) region, which stores system information and logs.
- **Example:** cbmem -list to list the available CBMEM regions and their contents.

https://github.com/coreboot/coreboot/tree/main/util/cbmem

```
omnigul-rev2 ~ # cbmem --list
CBMEM table of contents:
    NAME                 ID         START     LENGTH
 0. FSP MEMORY           46535052   76afe000  00500000
 1. CONSOLE              434f4e53   76abe000  00040000
 2. RW MCACHE            574d5346   76abd000  000004f0
 3. RO MCACHE            524d5346   76abc000  00000efc
 4. FMAP                 464d4150   76abb000  000005f6
 5. TIME STAMP           54494d45   76aba000  00000910
 6. VBOOT WORK           78007343   76aa6000  00014000
 7. MEM INFO             494d454d   76aa5000  00000f48
 8. AFTER CAR            c4787a93   76a97000  0000e000
 9. RAMSTAGE             9a357a9e   76921000  00176000
10. ACPI BERT            42455254   76911000  00010000
11. CHROMEOS NVS         434e5653   76910000  00000f00
12. REFCODE              04efc0de   768b1000  0005f000
13. SMM BACKUP           07e9acee   768a1000  00010000
14. RAMOOPS              05430095   767a1000  00100000
15. FSP LOGO             4c4f474f   766a1000  00100000
16. IGD OPREGION         4f444749   7669c000  00004203
17. COREBOOT             43425442   76694000  00008000
18. ACPI                 41435049   76670000  00024000
19. TPM2 TCGLOG          54504d32   76660000  00010000
20. SMBIOS               534d4254   76658000  00008000
21. FSP RUNTIME          52505346   76ffebe0  00000004
22. CSE BP INFO          42455343   76ffeb60  00000068
23. CSE SPECIFIC INFO    4553435f   76ffeb40  00000020
24. POWER STATE          50535454   76ffeae0  00000044
25. FSPM VERSION         56505346   76ffeac0  00000004
26. ROMSTAGE             47545352   76ffeaa0  00000004
27. ROMSTG STCK          90357ac4   76ffe9e0  000000a8
28. ACPI GNVS            474e5653   76ffe9a0  00000038
29. TYPE_C INFO          54595045   76ffe980  0000000c
```

dmidecode

- Gather detailed information about a system's hardware components directly from the BIOS/firmware.
- Know more

- **Example:** dmidecode -t memory to display information about the system's RAM modules.

https://github.com/mirror/dmidecode/tree/master

```
omnigul-rev2 ~ # dmidecode -t memory
# dmidecode 3.4
Getting SMBIOS data from sysfs.
SMBIOS 3.0 present.

Handle 0x000B, DMI type 16, 23 bytes
Physical Memory Array
        Location: System Board Or Motherboard
        Use: System Memory
        Error Correction Type: None
        Maximum Capacity: 128 GB
        Error Information Handle: Not Provided
        Number Of Devices: 2

Handle 0x000C, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x000B
        Error Information Handle: Not Provided
        Total Width: 16 bits
        Data Width: 16 bits
        Size: 2 GB
        Form Factor: Row Of Chips
        Set: None
        Locator: Channel-0-DIMM-0
        Bank Locator: BANK 0
        Type: LPDDR5
        Type Detail: Unknown Synchronous
        Speed: 8400 MT/s
        Manufacturer: Micron
        Serial Number: 00000000
        Asset Tag: Channel-0-DIMM-0-AssetTag
        Part Number: MT62F1G32D2DS-023 WT:B
```

iotools

- Low-level interaction with hardware devices, particularly useful for debugging and testing.
- **Example:** iotools –list-cmds to list the available commands to perform different operations.

https://code.google.com/archive/p/iotools/

```
omnigul-rev2 ~ # iotools --list-cmds
MSR: commands to access CPU model specific registers
  rdmsr
  wrmsr
LOGIC: commands to perform boolean algebra operations
  or
  and
  xor
  shl
  shr
  not
  btr
  bts
PCI: commands to access PCI registers
  pci_read8
  pci_write8
  pci_read16
  pci_write16
  pci_read32
  pci_write32
  pci_list
MISC
  rdtsc
  cpuid
```

QEMU Demo

QEMU

- A powerful and versatile open-source machine emulator and virtualizer.
- Allows you to run a virtual machine or emulate different computer architectures.

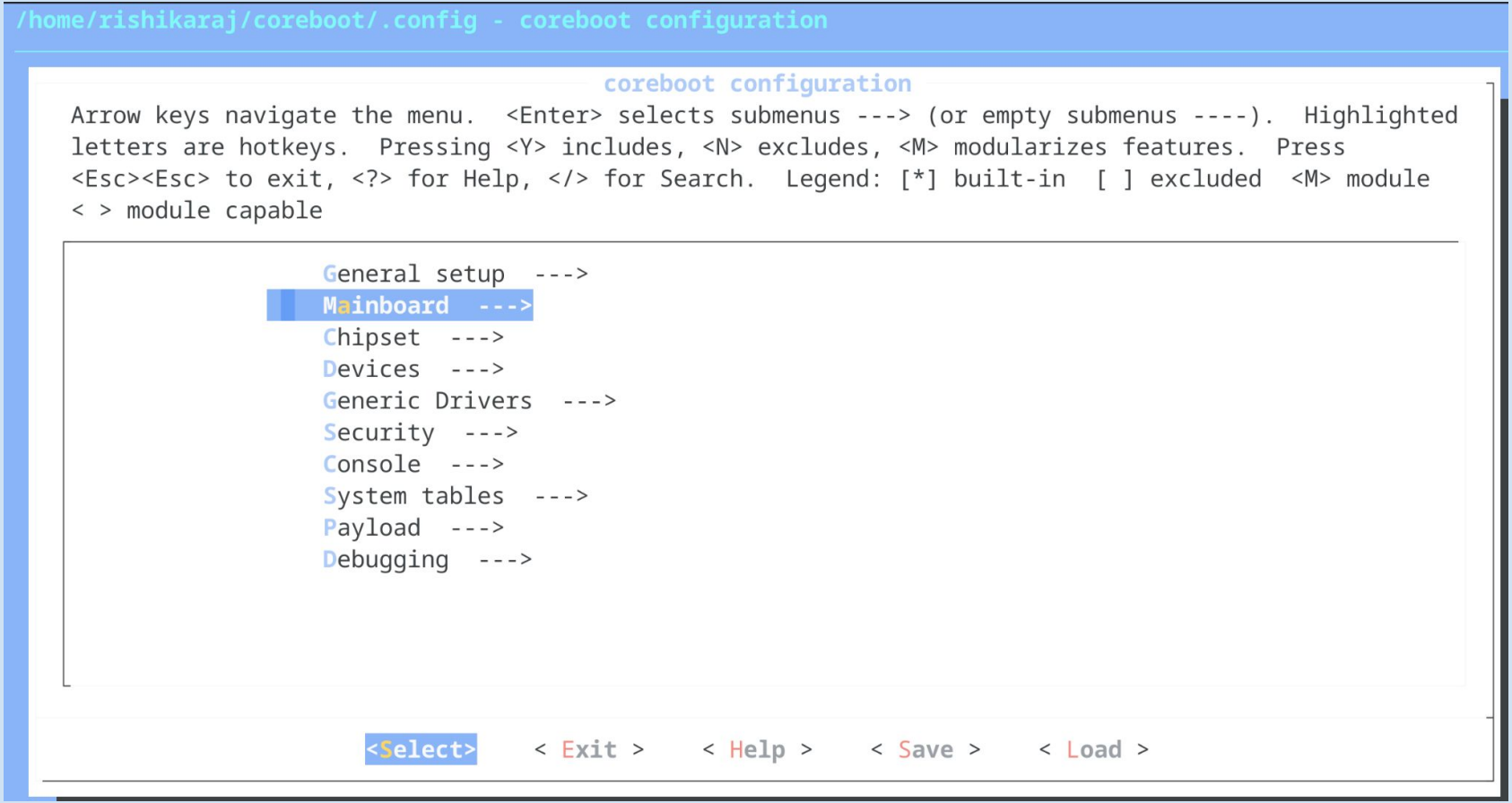
Provides a sandboxed environment where you can run and experiment with different software and operating systems without affecting your main system.

Steps to build & test coreboot

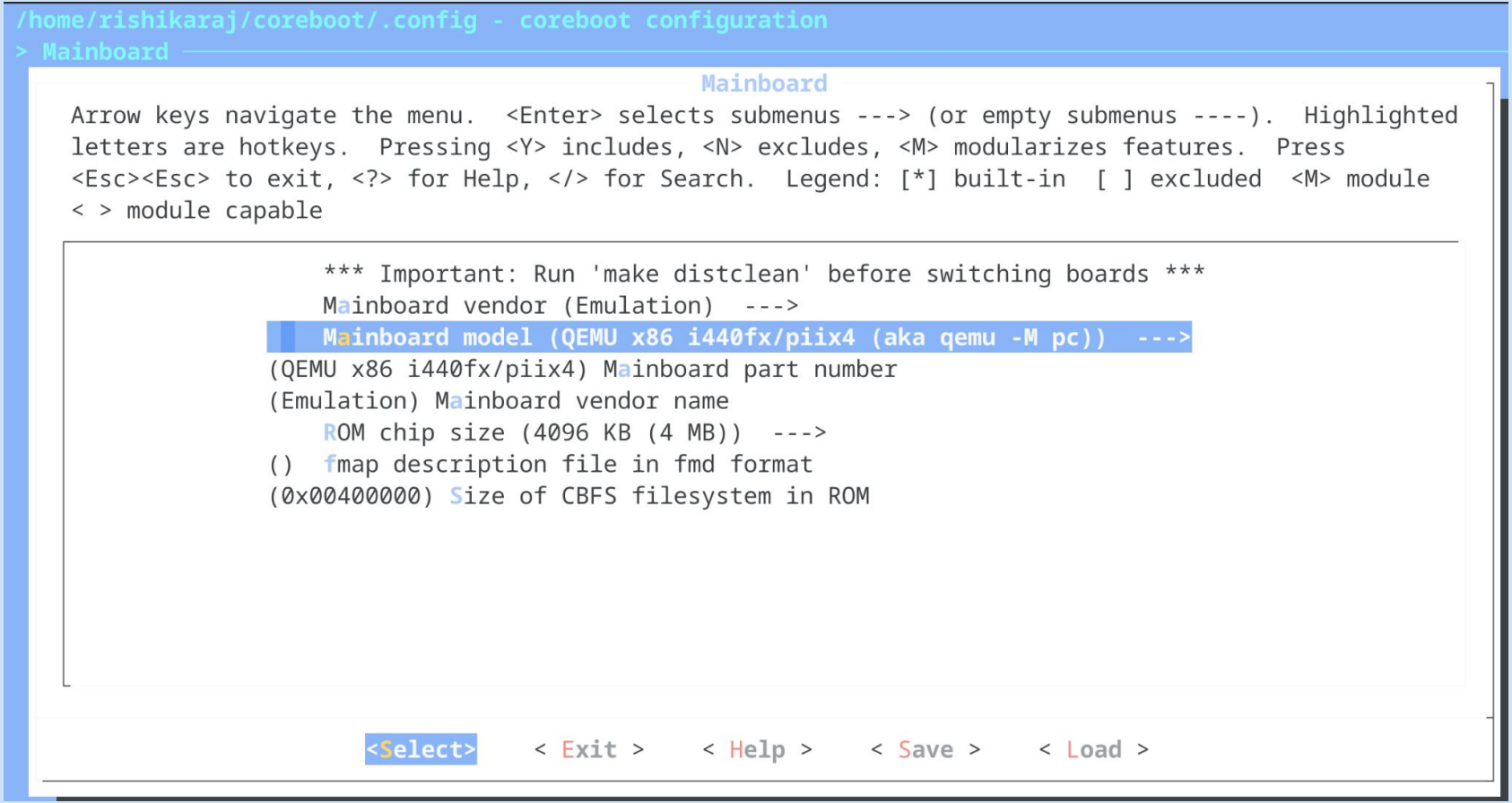
- Prerequisites steps
- Configure the build
  - Configure your mainboard
    `make menuconfig`
  - Select the payload:
    - Payload path
      payloads/coreinfo/build/coreinfo.elf
- Build coreboot
  `make`

- Test the image using QEMU
  `qemu-system-x86_64 -bios build/coreboot.rom -serial stdio`

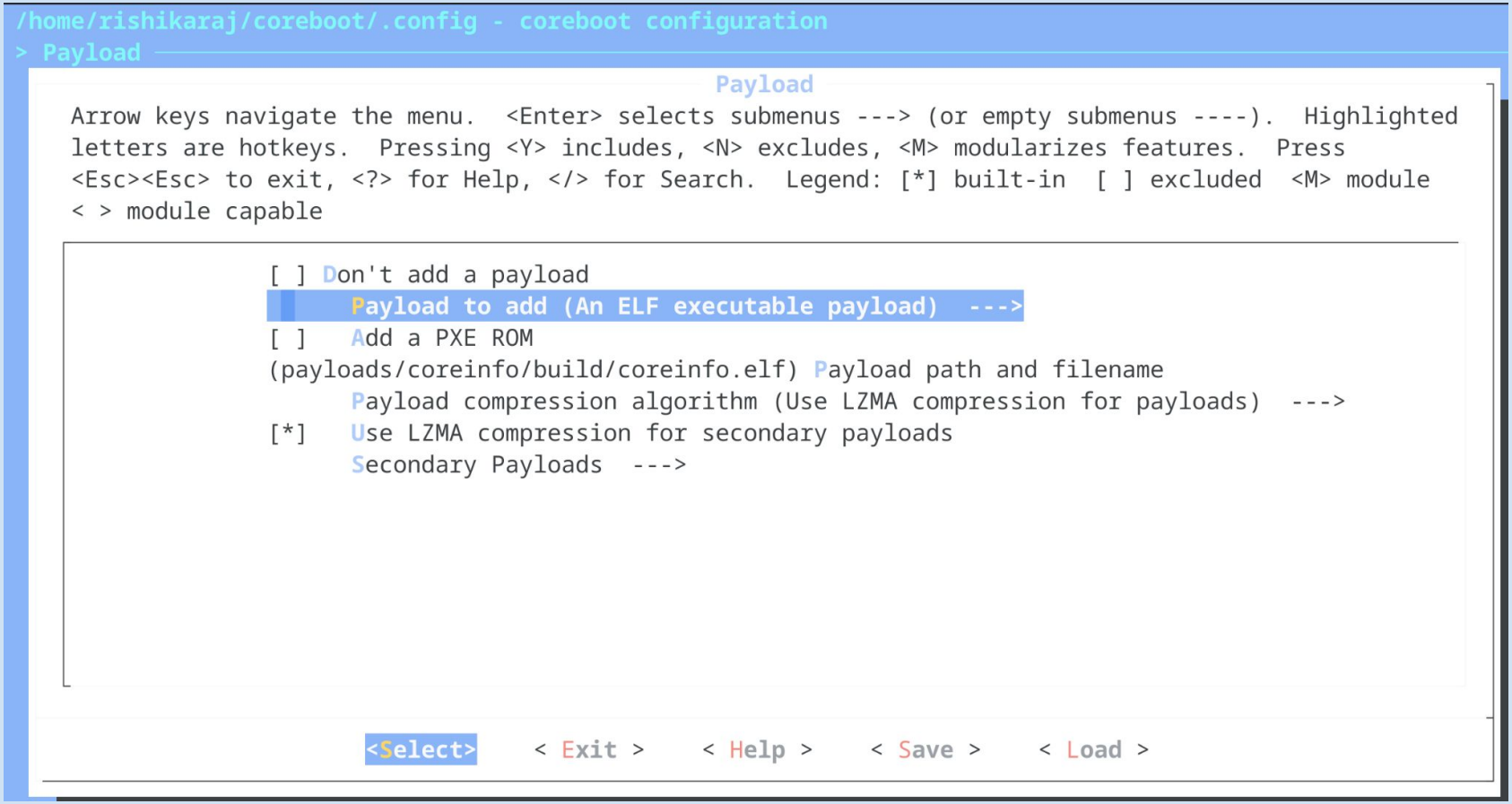
Scan to get the Prerequisite Doc



Menu configuration for coreboot



Mainboard configuration



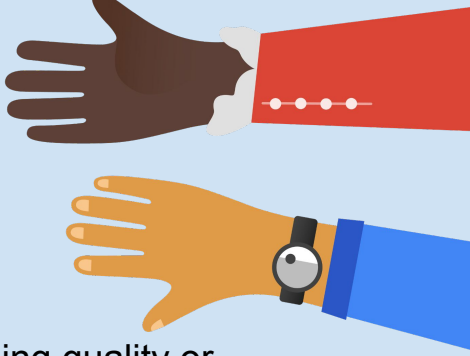
Selecting the payload

Provide payload path

```
                                coreinfo 0.1
coreboot Timestamps
------------------------------------------------------------------------------

25 entries total:

  0: 1st timestamp                                  53867
 11: start of bootblock                             56113 (2246)
 12: end of bootblock                               122314 (66200)
 13: starting to load romstage                      122401 (87)
 14: finished loading romstage                      128439 (6037)
  1: start of romstage                              128994 (555)
  4: end of romstage                                141009 (12014)
100: start of postcar                               148052 (7043)
101: end of postcar                                 148176 (123)
  8: starting to load ramstage                      148182 (6)
 15: starting LZMA decompress (ignore for x86)      149812 (1630)
 16: finished LZMA decompress (ignore for x86)      168749 (18936)
  9: finished loading ramstage                      170991 (2242)
 10: start of ramstage                              178023 (7031)
 30: device enumeration                             178370 (347)
 40: device configuration                           188338 (9967)
 50: device enable                                  220177 (31839)

[A: coreboot] [B: Multiboot] [C: Bootlog] [D: CBFS] [E: Timestamps]
F1: System F2: Firmware                                 08/20/2024 - 09:41:33
```

coreboot stages timestamp

First Step towards OSF



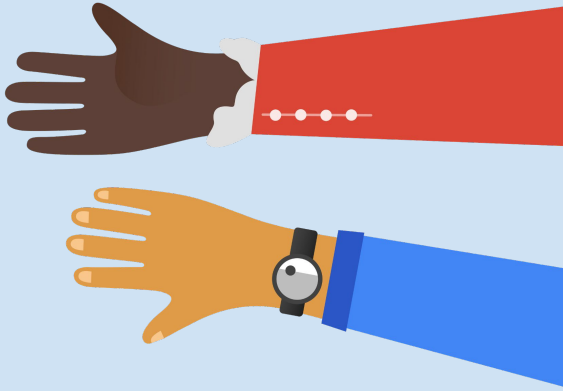
- Login in to https://review.coreboot.org/login with your GitHub.
- Crop your photo to a perfect square & resize the file to <50kb by degrading quality or dimensions
- Clone the repo which contains all user avatars -

  git clone "https://${USER}@review.coreboot.org/a/gerrit-avatars" && (cd "gerrit-avatars" && mkdir -p .git/hooks && curl -Lo `git rev-parse --git-dir`/hooks/commit-msg https://${USER}@review.coreboot.org/tools/hooks/commit-msg; chmod +x `git rev-parse --git-dir`/hooks/commit-msg)

- The name of the image should be “<ID>.jpg”. You can find out this ID under Profile section at https://review.coreboot.org/settings
- Move the image to the repo directory -

  mv <ID>.jpg gerrit-avatars/

First Step towards OSF



- Commit your changes

  cd gerrit-avatars

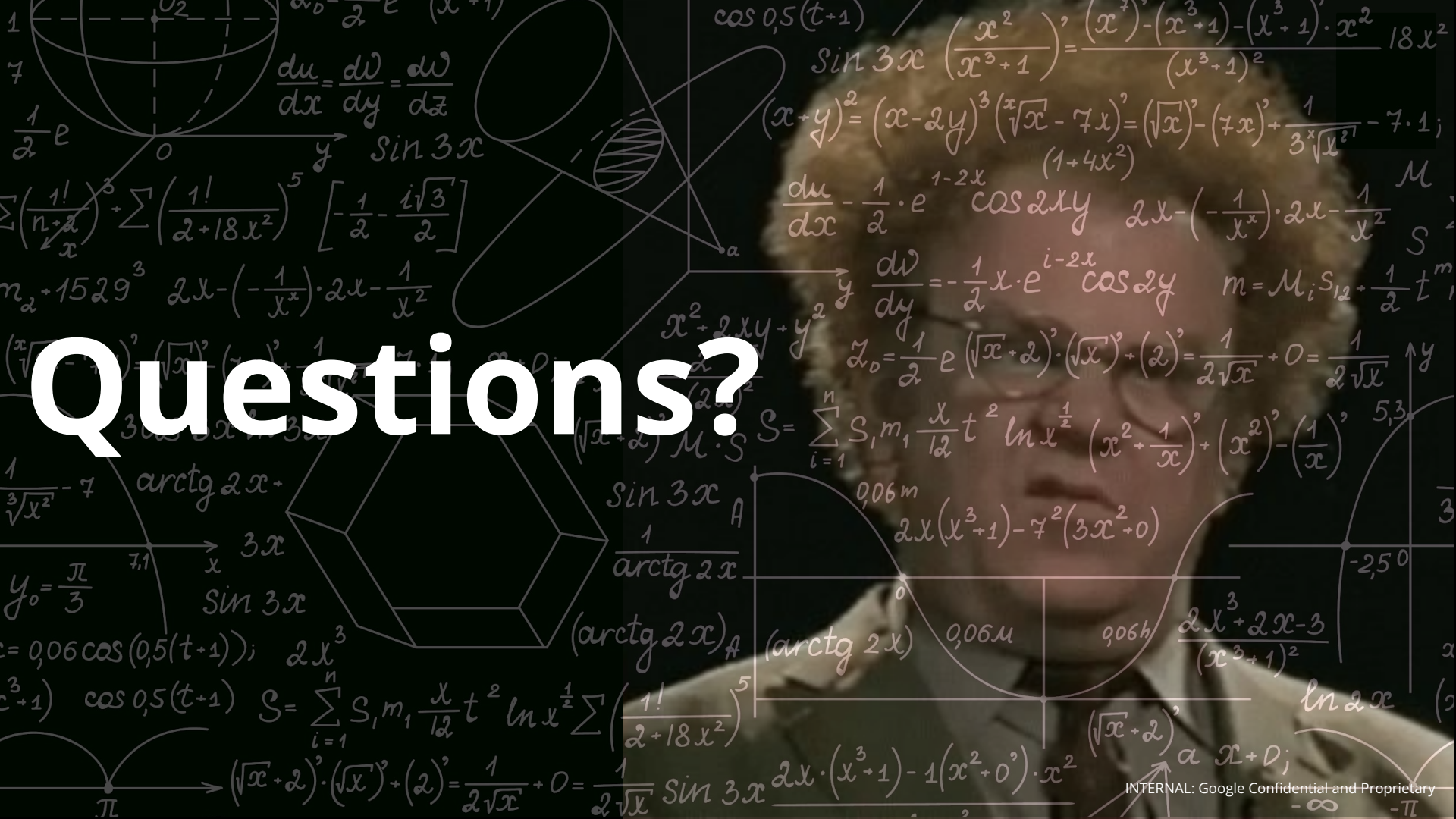
  git add <ID>.jpg

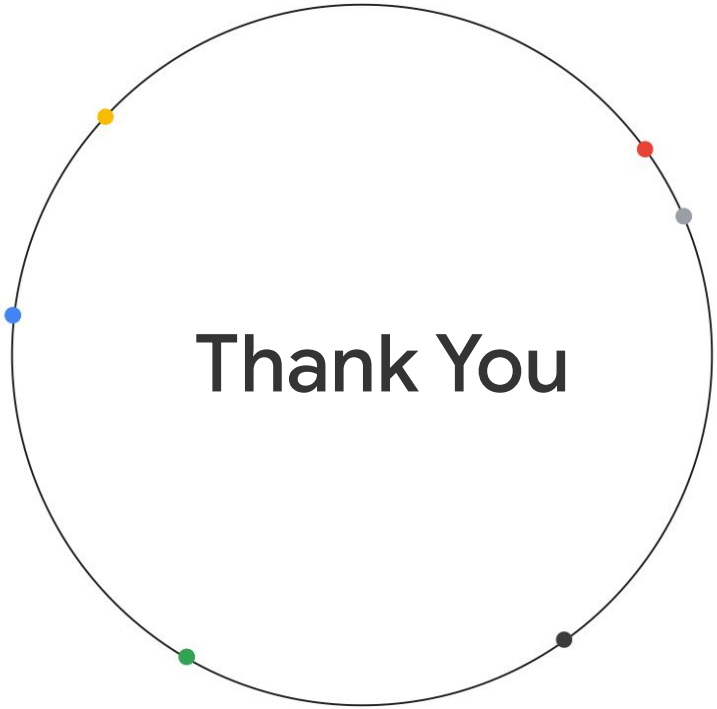
  git commit -vs -m “add avatar for <ID>” -m “${USER}’s avatar”cd

  git push origin HEAD:refs/for/main

- Hurray!! You have added your first change onto coreboot.

Questions?




Thank You