BIOS, UEFI... a może coś innego? Porozmawiajmy o coreboot

Gdańsk Embedded Meetup#11

Karol Zmysłowski

3MDEB

whoami

Karol Zmysłowski
Firmware Developer

- karol.zmyslowski@3mdeb.com
- linkedin.com/in/karolzet

- W 3mdeb od 3 miesięcy
- Pasjonat coreboot od niemal roku
- Wcześniej różne projekty embedded dla branży automotive, IoT, elektroniki
- Od ponad 10 lat użytkownik Wolnego Oprogramowania

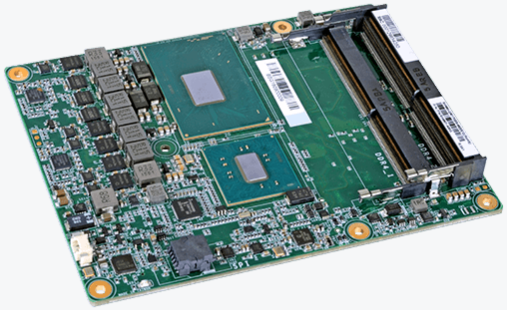
Kim jesteśmy i co robimy ?

- Niewielka firma z Gdańska
- Ponad 7 lat na rynku urządzeń wbudowanych
  - Embedded Linux (Yocto)
  - open-source firmware (coreboot, edk2, slimbootloader ...)
- Aktywni w społeczościach open-source oraz kontrybucji

Agenda

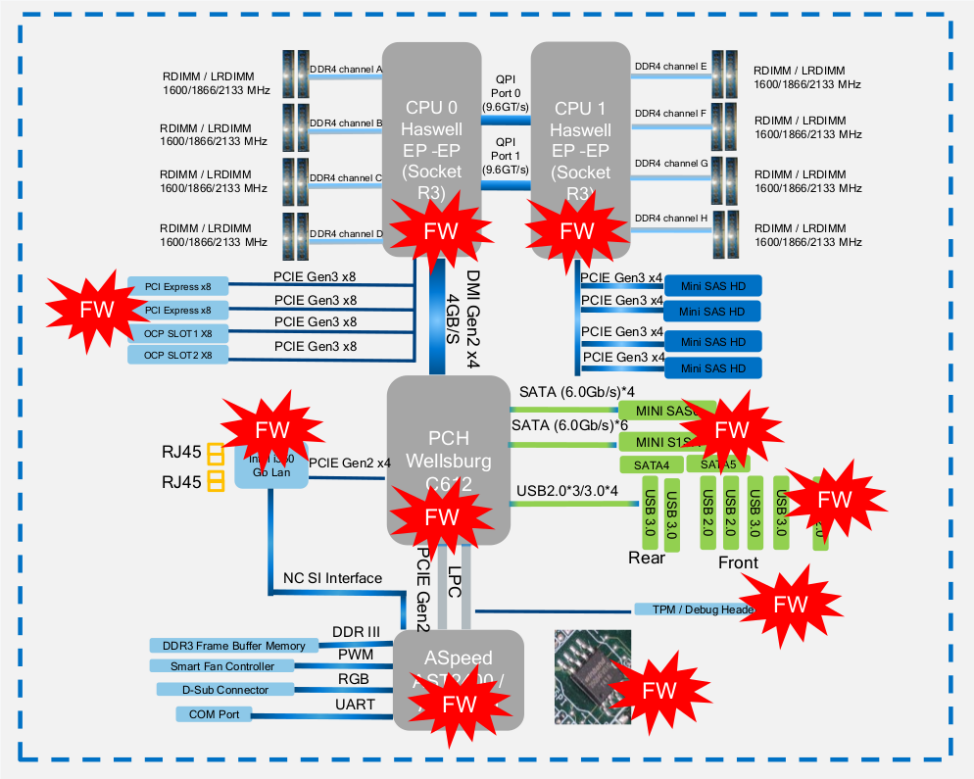
- Dwa słowa o sprzęcie, jaki nas dziś interesuje
- Gdzie jest firmware, oraz co zawiera?
- Co się dzieje po naciśnieciu przycisku Power?
- Skąd bierze się firmware?
- Dlaczego warto zaintersować się open-source firmware?
- coreboot jako przykład open-source firmware
- Q&A

Dwa słowa o sprzęcie



Gdzie jest firmware?

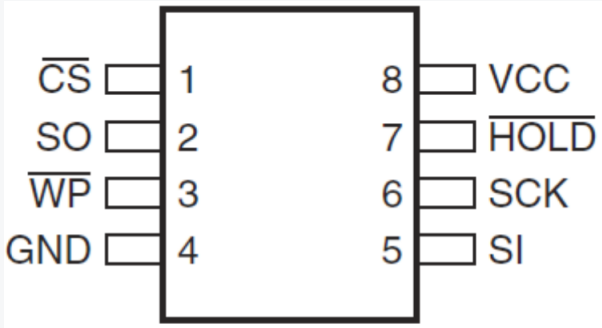
WSZĘDZIE!



Źródło: https://gitlab.com/opensecuritytraining/Arch4031_x86-64_RV_coreboot_slides_and_subtitles

Gdzie jest firmware?

- Firmware jest w pamięci nieulotnej, na płycie głównej komputera
- Interfejs SPI
- Typowe parametry obecnie:
  - SOIC8 / WSON8
  - 3.3V lub 1.8V
  - do 16MB / 32MB pojemności
  - do 133 MHz



Gdzie jest firmware?

- Pamięć połączona interfejsem SPI z CPU poprzez PCH
- PCH (Platform Controller Hub) - znany jako `chipset`
  - znajduje się na płycie głównej
  - bywa zintegrowany w jednym układzie SoC wraz z CPU

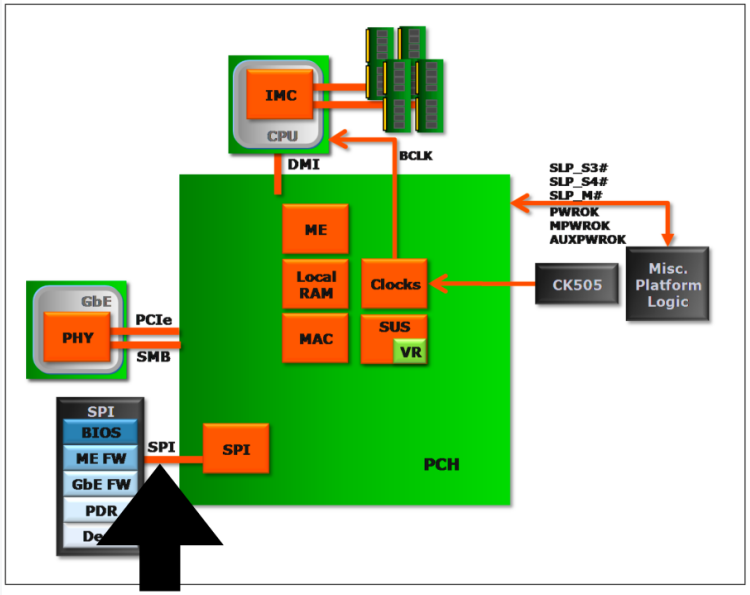


Figure 5-11. PCH Intel® Management Engine High-Level Block Diagram

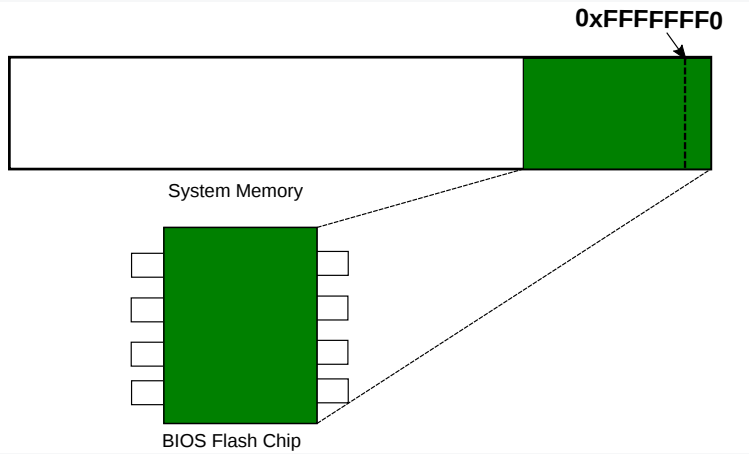
Start systemu, krok po kroku

- Co dzieje się po przyciśnięciu przycisku zasilania w komputerze?
  - w bardzo uproszczonej formie

Źródło: https://www.teknofilo.com/wp-content/uploads/2021/07/Analisis-Huawei-Mate-X-Pro-2021-8.jpg

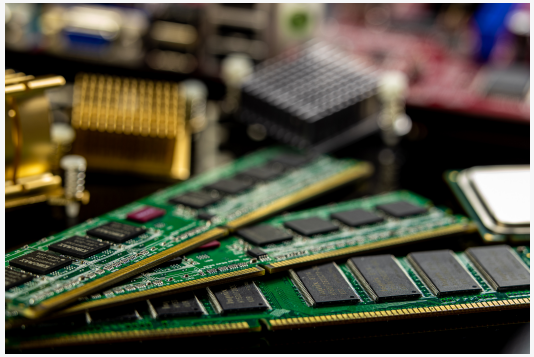
Start systemu, krok po kroku

- Reset vector
  - pamięć SPI flash mapowana do pamięci głównej
  - wykonywanie firmware rozpoczynane jest od adresu fizycznego: `0xFFFFFFF0`



Start systemu, krok po kroku

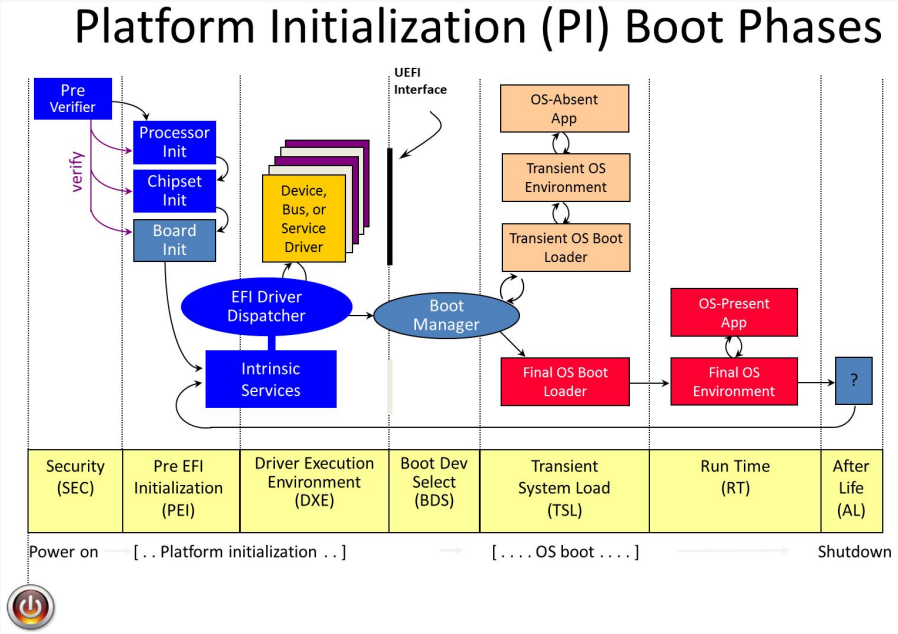
- Inicjalizacja Cache-as-RAM
  - pierwsze instrukcje w firmware musimy napisać w asemblerze
  - aby uzyć języka C, potrzebujemy pamięci dla stosu
  - na początku nie mamy dostępu do pamięci głównej
  - możemy użyć cache wbudowanego w procesor
- Trening pamięci głównej i deinicjalizacja Cache-as-RAM
  - najczęściej pamięć jest w slotach - można ją wymieniać, dokładać
  - nie możemy zapisać "na sztywno" paremetrów kontrolera pamięci
  - konieczny jest "trening pamięci" oraz rekonfiguracja kontrolera



Źródło: https://www.pexels.com/photo/green-and-black-computer-ram-stick-6636474/

Start systemu, krok po kroku

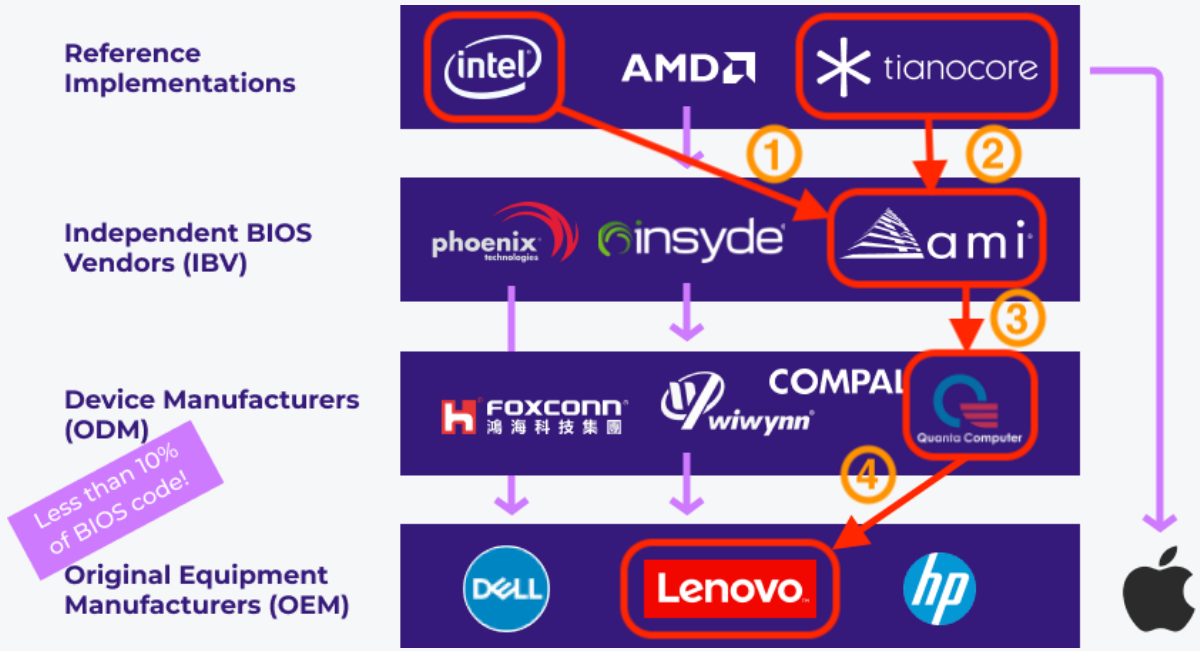
- Enumeracja i inicjalizacja urządzeń PCI(e), peryferiów
- Zapis struktur do pamięci (ACPI, SMBIOS)
- Uruchomienie bootloadera systemu operacyjnego z dysku
- Uruchomienie systemu operacyjnego przez bootloader



Źródło: https://github.com/tianocore/tianocore.github.io/wiki/PI-Boot-Flow

Firmware od IBV

- IBV - Independent BIOS Vendors



Skąd bierze sie firmware?

- Z punktu widzenia użytownika, firmware dostarczany jest wraz ze sprzętem i jest jego integralna i nieodrywalną częścią (jako fragment sprzętu)
- Z punktu widzenia producenta, dostarczany jest przez jednego z kilku IBV (Independent BIOS Vendor) na świecie
- Cechy charakterystyczne:
  - brak dostępu do kodu źródłowego
  - firmware dostarczany w formie binarnej
  - utrudniona modyfikacja - zależna od możliwości narzędzi od IBV
  - jest on licencjonowany - nie jestesmy jego właścicielem
  - skomplikowany proces pozyskania firmware
  - może być nieefektywny dla mniejszych serii urządzeń
  - ograniczone możliwości aktualizacji
  - zwykle, nikt firmware nie aktualizuje, chyba że nie działa określony zasób; ostatecznie aktualizujemy, zgodnie z zasadą: "Press & pray" ;)

Skąd bierze sie firmware?

- Z punktu widzenia użytownika, firmware dostarczany jest wraz ze sprzętem i jest jego integralna i nieodrywalną częścią (jako fragment sprzętu)
- Z punktu widzenia producenta, dostarczany jest przez jednego z kilku IBV (Independent BIOS Vendor) na świecie
- Cechy charakterystyczne:
  - brak dostępu do kodu źródłowego
  - firmware dostarczany w formie binarnej
  - utrudniona modyfikacja - zależna od możliwości narzędzi od IBV
  - jest on licencjonowany - nie jestesmy jego właścicielem
  - skomplikowany proces pozyskania firmware
  - może być nieefektywny dla mniejszych serii urządzeń
  - ograniczone możliwości aktualizacji
  - zwykle, nikt firmware nie aktualizuje, chyba że nie działa określony zasób; ostatecznie aktualizujemy, zgodnie z zasadą: "Press & pray" ;)

Czy jest jakaś alternatywa?

Dlaczego open-source firmware?

- Zaprzeczenie powyższego modelu
- Pełny dostęp do kodu źródłowego; możemy:
  - używać firmware w dowolnym, odpowiadającym nam celu
  - poznać mechanizm działania firmware, sterowania procesem uruchamiana komputera
  - dowolnie go modyfikować
  - rozpowszechnianiać modyfikację, by pomagać innym, i czerpać z pomocy innych
- W skrócie: pełna kontrola nad własym komputerem, niemożliwa do osiągnięcia w przypadku własnościowego firmware
- Bezpieczeństwo
  - łatki bezpieczeństwa mogą zostać udostępnione znacznie szybciej, niż wynika to z cyklu życia oprogramowania
  - dla starszych sprzętów, producent może nigdy ich nie wydać

Dlaczego open-source firmware? - c. d.

- Aspekt komercyjny
  - Producenci, dostawcy usług, takich jak Google, Facebook, Amazon, IBM pracują nad systemami opensource bo uniezależnia to ich od łańcucha dostaw firmware'u, daje pełną kontrolę, a dodatkowo samodzielnie mogą tworzyć wartość dodaną np. wsparcie dla dedykowanych akcelaratorów AI, kryptogarficznych
  - uniezależnienie platformy Chromebook od starego ekosystemu kontrolowanego przez tandem Intel/AMD
  - Elastyczność w kwestii przystosowania do dedykowanych aplikacji (Przykłady: COM Express-Kontron, Advantech)
  - Unikalna wartość dzięki OSF (Purism, System76, Novacustom)

Dlaczego open-source firmware?

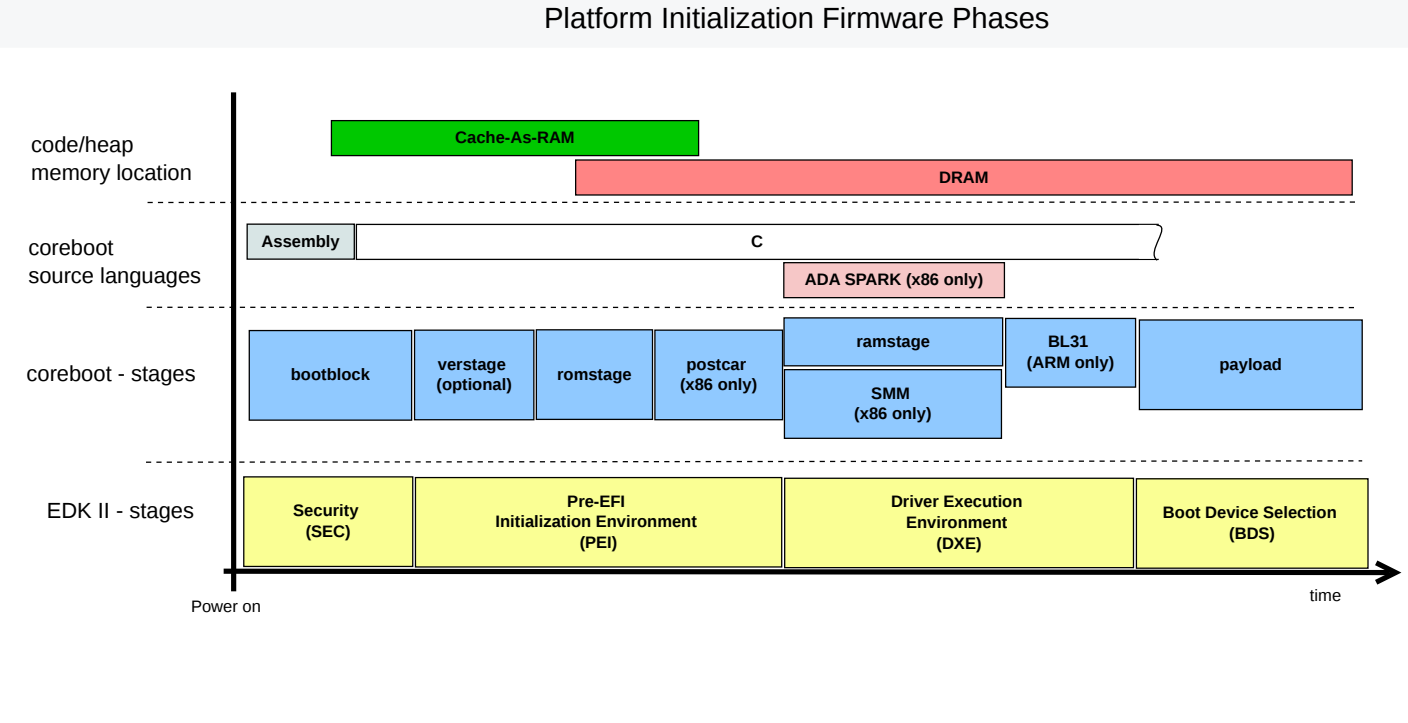
- Funkcjonalność
  - Szeroko pojęta "customizacja" - zmiana logo, zmiana klawiszy wejścia do menu, dodanie nowych opcji, co tylko sobie wymyślimy
  - Dodanie wsparcia dla dodatkowych interfejsów
  - Dodanie wsparcia dla nowych procesorów
- Praktyczne przykłady
  - Bootowanie z dysku NVMe na Dell OptiPlex stało się możliwe dzięki coreboot

Czym jest coreboot?

- Twórcą jest Ron Minnich
- Projekt został zapoczątkowany w 1999 w Los Alamos National Laboratory
- Wspiera następujące platformy: ARM, ARM64, MIPS (do 4.11), PowerPC, RISC-V, x86/x86-64
- nazwę "coreboot" piszemy zawsze z małej litery
- Projekt wykonany zgodnie z zasadą "single responsibility": zrób jedną rzecz, zrób to dobrze - w tym przypadku minimalną inicjalizację platformy, następnie przekaż sterowanie kolejnym elementom (w tym przypadku payload'owi)
- Informacje bieżące na temat projektu dostępne na Twitterze: @coreboot

(1) https://www.osfc.io/2019/talks/coreboot-20th-anniversary

Architektura coreboot



https://doc.coreboot.org/getting_started/architecture.html

- Fragmenty tworzone w assemblerze i C
- Uruchomiony jako pierwszy po sygnale RESET
- wykorzystuje mechanizm Cache-As-RAM do sterty i stosu
- ustawia rejestr SP (Stack Pointer)
- czyści pamięć w segmencie BSS

Dodatkowo, na platformach x86

- dokonuje update'u mikrokodu,
- inicjalizuje timer (bazuje na nim np. uptime)
- dokonuje przełączenia trybu procesora (16-bit, real mode na 32-bit protected mode)

- Tu startuje mechanizm "root-of-trust"
- Przyjmuje się, że nie może zostać nadpisany in-field (wraz z kluczem publicznym)
- Instaluje handler, który weryfikuje plik przed załadowaniem z CBFS czy partycji
- Mechanizm zweryfikowanego bootowania umożliwia zarówno, zaufaną aktualizację jak i tryb odzyskiwania systemu (na wypadek awarii)

- Inicjalizacja pamięci DRAM
- przygotowanie do do inicjalizacji urządzeń peryferyjnych

- Końcowa faza CAR (Cache-as-RAM)
- Rozpoczęcie wykonania kodu z uprzednio zainicjowanej pamięci RAM
- Wyczyszczenie pamięci cache i załadowanie fazy ramstage
- W porównaniu do innych faz bootowania ma najmniejszy rozmiar

Faza, po wykonaniu której zostają zainicjowane:

- urządzenia PCI
- peryferia SoC (I2C, SPI)
- moduł TPM (opcjonalnie, o ile nie został zainicjowany podczas verstage)
- karta graficzna (opcjonalnie)
- CPU

ramstage - c. d.

Po zainicjowaniu wcześniej wymienionych urządzeń zostają wypełnione:

- tabele ACPI (specyficzne dla platformy x86)
- tabele SMBIOS (specyficzne dla platformy x86)
- tabele coreboot
- devicetree (specyficzne dla platformy ARM)

coreboot - DIY

- Źródła coreboot pobieramy z repozytorium komendą:

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

- Następnie wchodzimy do katalogu:

```
cd coreboot
```

- Dodajemy moduły specyficzne dla platformy

```
git submodule update --init --checkout --recursive
```

- Dokumentacja coreboot znajduje się pod adresem: https://doc.coreboot.org/index.html
- Adres serwera Gerrit (służącego do review kodu): https://review.coreboot.org

- Zbudowanie narzędzi

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.
- Konfiguracja

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.
- Konfiguracja
  - `make menuconfig`

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.
- Konfiguracja
  - `make menuconfig`
- kompilacja

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.
- Konfiguracja
  - `make menuconfig`
- kompilacja
  - `make`

coreboot - DIY

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.
- Konfiguracja
  - `make menuconfig`
- kompilacja
  - `make`
- testowanie

coreboot - DIY

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.
- Konfiguracja
  - `make menuconfig`
- kompilacja
  - `make`
- testowanie
  - `qemu`

coreboot - DIY

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.
- Konfiguracja
  - `make menuconfig`
- kompilacja
  - `make`
- testowanie
  - `qemu`
- manipulacja na obrazie

coreboot - DIY

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.
- Konfiguracja
  - `make menuconfig`
- kompilacja
  - `make`
- testowanie
  - `qemu`
- manipulacja na obrazie
  - `cbfstool`

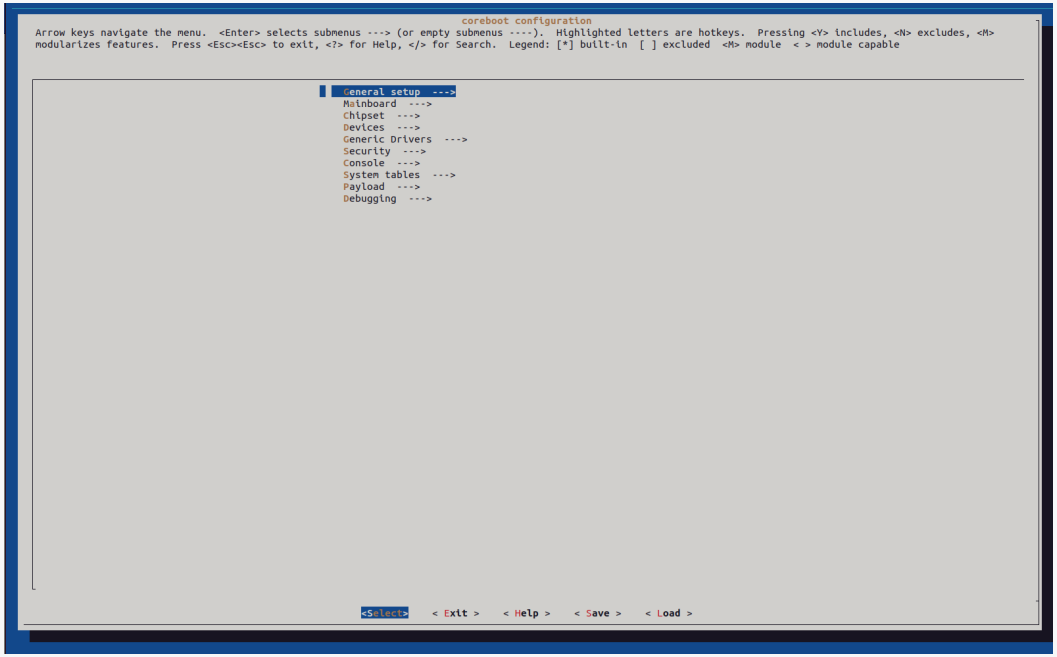
coreboot - DIY

- Zbudowanie narzędzi
  - `crossgcc-i386`, `crossgcc-x64`, `crossgcc-aarch64`, etc.
- Konfiguracja
  - `make menuconfig`
- kompilacja
  - `make`
- testowanie
  - `qemu`
- manipulacja na obrazie
  - `cbfstool`
- Co dalej? Opcjonalnie, Załadowanie obrazu na rzeczywisty sprzęt
  - flashrom

Konfiguracja coreboot

- Wykorzystuje narzędzie podobne w wyglądzie i zachowaniu do konfiguratora kernela Linux (Kconfig)
- Konfigurator uruchamiany po wydaniu polecenia

```
make menuconfig
```



Kompilacja

- Kompilację wykonujemy w standardowy sposób, tj. wydając komendę:

```
make
```

- Wynikiem działania kompilatora są pliki umieszczone w katalogu `build`
- Struktura wygląda jak poniżej:

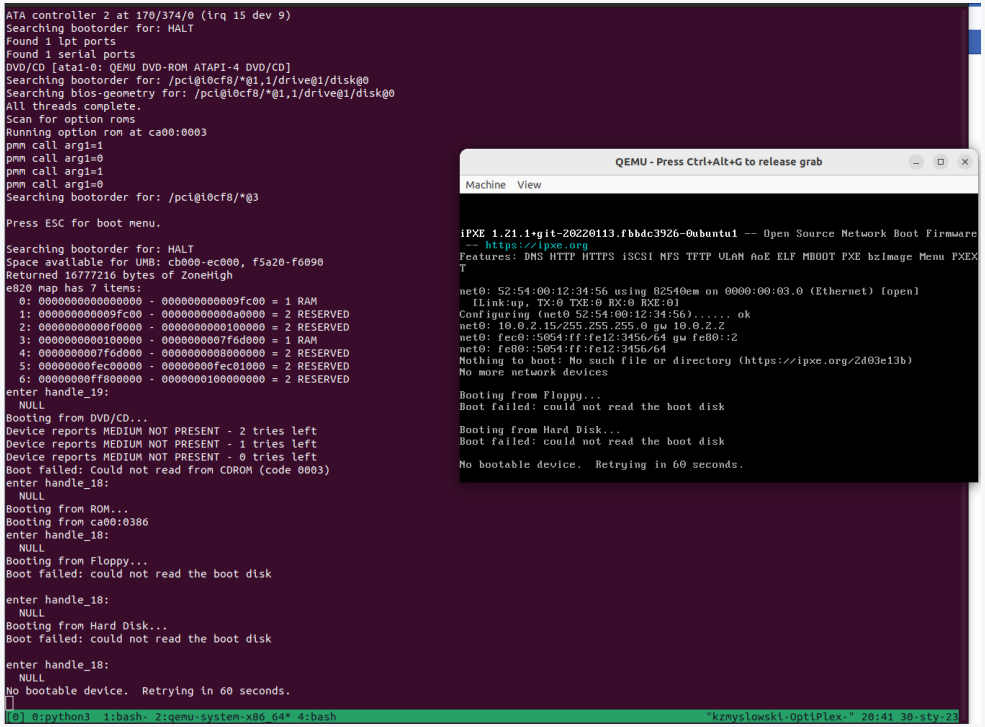
```
$ ls build/
auto.conf        coreboot.pre  fmap_config.h   ramstage         static_devices.h
auto.conf.cmd    coreboot.rom  fmap.desc       rmodtool         static_fw_config.h
bootblock        cpu           fmap.fmap       rmodules_arm     static.h
build.h          cse_fpt       fmap.fmd        rmodules_arm64   util
build_info       cse_serger    generated       rmodules_ppc64   verstage
cbfs             decompressor  ifwitool        rmodules_riscv   xcompile
cbfstool         dsdt.aml      libgnat-x86_32  rmodules_x86_32
cmos_layout.bin  dsdt.asl      mainboard       rmodules_x86_64
config           dsdt.d        option_table.h  romstage
config.h         dsdt.dsl      postcar         smm
```

- Wśród nich znajduje się interesujący nas plik: `coreboot.rom`. W nim zawarte są wszystkie niezbędne elementy, umożliwiające uruchomienie naszego firmware'u.

Uruchamianie w QEMU

- Emulację wykonujemy komendą:

```
qemu-system-x86_64 -bios build/coreboot.rom -serial stdio
```



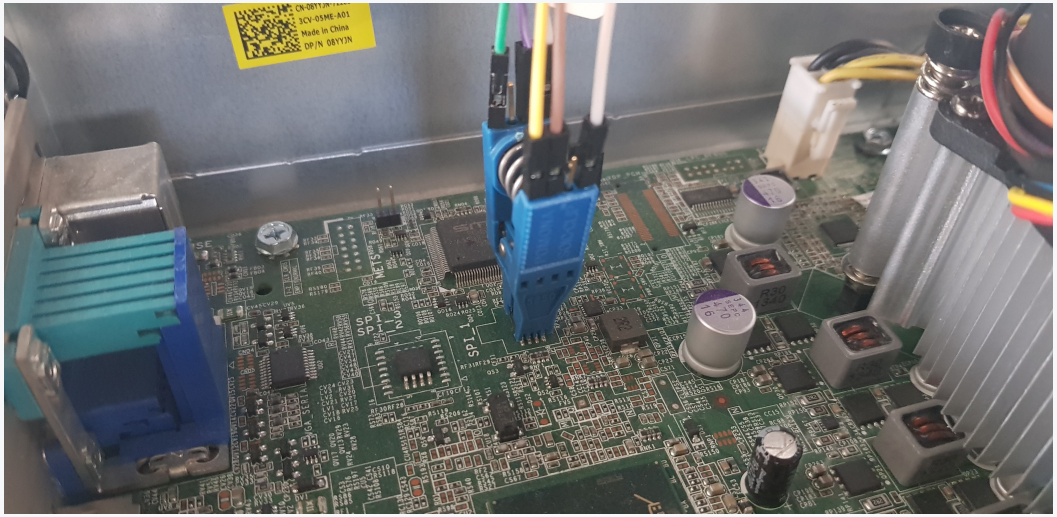
cbfstool - manipulacja obrazem firwmare

- Narzędzie do zarządzania wybudowanym obrazem firwmare
- Możliwość manipulacji na wygenerowanym obrazie (przeglądanie, modyfikacja)
- Uruchomienie dla wybudowanego obrazu QEMU:

```
$ ./cbfstool ./coreboot.rom print
FMAP REGION: COREBOOT
Name                           Offset     Type           Size   Comp
cbfs_master_header             0x0        cbfs header         32 none
fallback/romstage              0x80       stage            17368 none
fallback/ramstage              0x4500     stage            59901 LZMA (122824 decompressed)
config                         0x12f80    raw               1879 LZMA (5507 decompressed)
revision                       0x13740    raw                700 none
build_info                     0x13a40    raw                 96 none
fallback/dsdt.aml              0x13b00    raw               4044 none
cmos_layout.bin                0x14b00    cmos_layout        640 none
fallback/postcar               0x14dc0    stage            20720 none
fallback/payload               0x19f00    simple elf       43075 none
(empty)                        0x24780    null           4033124 none
bootblock                      0x3fd200   bootblock         11200 none
```

Uruchamianie na sprzęcie

- W niektórych przypadkach możemy użyć wewnętrzego programatora
  - uruchomić jakiś OS na docelowym systemie
  - `flashrom -p internal`
- W pozostałych przypadkach musimy użyć zewnętrznego programatora



Gdzie jeszcze działa coreboot?

- Chromebooki od Google
- Inne laptopy
  - https://configurelaptop.eu/
  - https://system76.com/
  - https://starlabs.systems/
  - https://www.puri.sm
- Urządzenia sieciowe
  - https://eu.protectli.com/
  - https://pcengines.ch/
- Maszyny CNC (e.g. Siemens SINUMERIK)
  - https://www.youtube.com/watch?v=tq4xSipCWEU
- Dedykowane rozwiązania dla przemysłu, wojska
- Repozytorium Tesli
  - https://github.com/teslamotors/coreboot/tree/tesla-4.12-amd

Dystrybucje

- coreboot to firmware framework
  - daje wiele możliwości konfiguracji
  - daje możliwość uruchamiania wielu różnych payloadów
- Może być wiele dystrybucji firmware opartych o coreboot
  - podobnie jak jest wiele dystrybucji OS opartych o jądro Linux
- Przykłady
  - Dasharo: https://docs.dasharo.com/
  - heads: https://github.com/osresearch/heads
  - Libreboot: https://libreboot.org/
  - skulls: https://github.com/merge/skulls
  - lista w dokumentacji: https://doc.coreboot.org/distributions.html

Co dalej?

- Szkolenia OpenSecurityTraining2: https://ost2.fyi/
  - Architecture 2001: x86-64 OS Internals
  - Architecture 4021: Introductory UEFI
  - Architecture 4031: x86-64 Reset Vector: coreboot
- whitepaper `Minimal Intel Architecture Boot Loader`
  - http://www.cs.cmu.edu/~410/doc/minimal_boot.pdf
- A Hardware enthusiast view on the usefulness of open source Firmwares
  - https://zirblazer.github.io/htmlfiles/coreboot.html?ver=123

Jak się z nami skontaktować

Jak się z nami skontaktować:

- contact@3mdeb.com
- facebook.com/3mdeb
- @3mdeb_com
- linkedin.com/company/3mdeb
- https://3mdeb.com
- Book a call
- Sign up for the newsletter
- Events
- career@3mdeb.com

Q&A