The road to liberating software at the lower levels

Replicant

Paul Kocialkowski
paulk@replicant.us

Saturday January 30th 2015

FOSDEM 16 .org
Brussels 30 & 31 January

Devices and hardware components

Scope of devices:

- Traditional, full computers *(x86)*
- Embedded and mobile devices *(ARM, MIPS, etc)*

Devices and hardware components

Scope of devices:

- Traditional, full computers *(x86)*
- Embedded and mobile devices *(ARM, MIPS, etc)*

Different kinds of hardware, chips:

- Main processor
- Auxiliary processors *(modem, VPU, DSP, GPU)*
- Controllers *(xHCI, EC)*
- Peripherals *(Wi-Fi, bluetooth, USB input devices, etc)*

Lower levels of software

Software at the lower levels ?

- Communicating directly with the hardware (registers)
- Hardware access via PIO, MMIO
- Direct access or through controllers

Lower levels of software

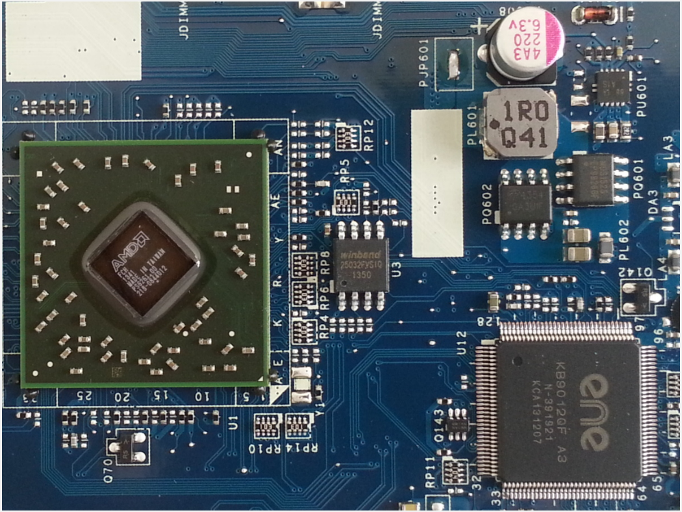
Software at the lower levels ?

- Communicating directly with the hardware (registers)
- Hardware access via PIO, MMIO
- Direct access or through controllers

Low-level software:

- Drivers
- Bootup software *(BIOS, hardware initialization, bootloader)*
- Firmwares

Lower levels of software



Close to the hardware!

Liberating software at the lower levels

Why bother liberating the lower levels?

- Distant from the UI and users
- Not likely to evolve *'it just works'*
- Proprietary software gets the job done
- Also often allows running a free system
  (drivers, bootup, firmwares)

Liberating software at the lower levels

Why bother liberating the lower levels?

- Distant from the UI and users
- Not likely to evolve *'it just works'*
- Proprietary software gets the job done
- Also often allows running a free system
  (drivers, bootup, firmwares)

Because **free software matters**!

Liberating software at the lower levels

Because **free software matters**!

- Knowledge of how the hardware works
- Being in control instead of being controlled
- Ability to adapt to one's needs
- Matter of trust, privacy and security

Liberating software at the lower levels

Because **free software matters**!

- Knowledge of how the hardware works
- Being in control instead of being controlled
- Ability to adapt to one's needs
- Matter of trust, privacy and security

Technical reasons:

- Changes in APIs, interfaces
- Bug fixes, improvements
- Flexibility, hacking, unintended uses

Liberating the software

Liberating the software:

- Manufacturer's positions
  - Economical interest
  - Copyright *(IP blocks, patents)*
  - Copyleft *(kernel, bootloaders)*
  - Quality, maintainability *(reference)*
- Reverse engineering
- Ressources and time needed
- Long-term interest, obsolescence
- Technical possibilities, recurrent limitations

Recurrent limitation when liberating software

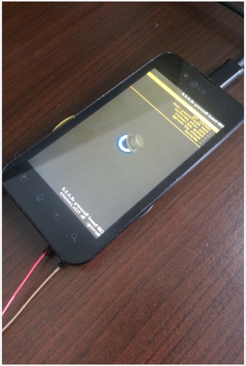
Recurrent limitations:

- Technical knowledge, adapted tools
- Legal constraints (reverse engineering)
- Hardware documentation, schematics, etc
- Ability to replace software:
  Read-only memory, secret interfaces, external access
- Ability to run our own code: signatures
- Ability to debug code execution

Example: Optimus Black

Optimus Black: overview

- Mainstream LG smartphone from 2011
- OMAP3630 platform
- Technical documentation *(schematics)*: EN_LG-P970_SVC_ENG_110415.pdf
- U-Boot and X-Loader bootloaders reference source code released by LG
- Community Android support *(CyanogenMod)*



Optimus Black: signature checks

- HS and GP versions of OMAP platforms
- CONTROL_STATUS (0x480022f0) register:

Bits	Field Name	Description	Type	Reset
31:11	RESERVED	Reserved field	R	0x-
10:8	DEVICETYPE	Device type value sampled at power_on reset 0b011 : GP device Other values : Reserved	R	0x-
7:6	RESERVED	Reserved field	R	0x-
5:0	SYSBOOT	Sys.Boot pin values sampled at power_on reset	R	0x-

```
$ devmem 0x480022f0 16
0x0325
```

- OMAP GP version: no signature checks

Possible to port a **free bootloader** (U-Boot)!

Optimus Black: code execution

Loading code to the device:

- Boot order: SYS_BOOT pins and resistors
- Memory or peripheral priority: SYS_BOOT[5]
- Default: SYS_BOOT[5]=0 *(MMC2 over USB)*

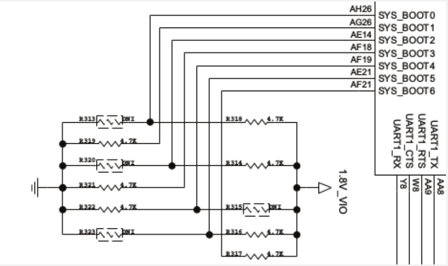


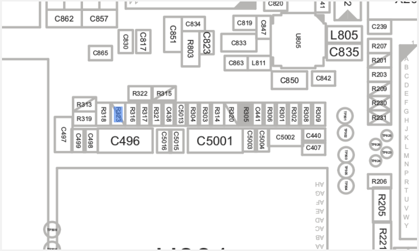
Table 26-3. Memory Preferred Booting Configuration Pins After POR

sys_boot [4:0]	Booting Sequence When SYS.BOOT[5] = 0 Memory Preferred Booting Order				
	First	Second	Third	Fourth	Fifth
0b00101	*MMC2*	USB			

Optimus Black: code execution

Loading code to the device:

- Pull-down on SYS_BOOT[5]: R323
- Removed R323: SYS_BOOT[5]=1 *(USB over MMC2)*



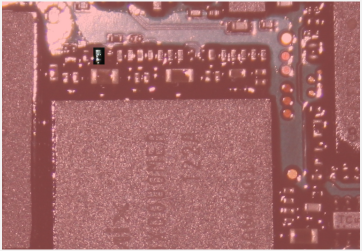


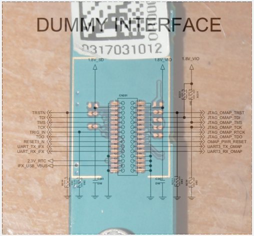
Table 26-4. Peripheral Preferred Booting Configuration Pins After POR

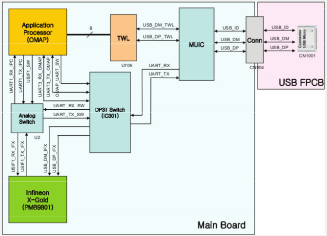
sys_boot [4:0]	Booting Sequence When SYS.BOOT[5] = 1 Peripheral Preferred Booting Order				
	First	Second	Third	Fourth	Fifth
0b00101	USB	*MMC2*			

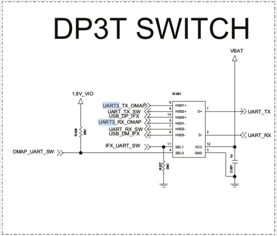
Optimus Black: debugging

Basic debugging feedback:

- Serial console: UART3
- Exposed from: dummy interface, dp3t switch, USB







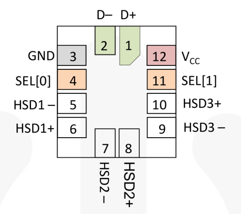
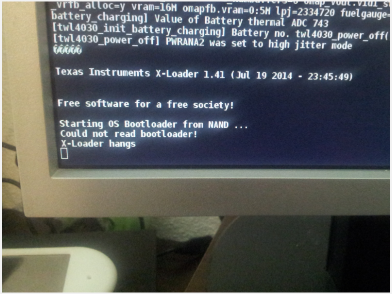


Figure 2. Pin Assignments (Top Through View)

Optimus Black: debugging

- UART Tx exposed from DP3T switch
- Connectors on the device



Upstream U-Boot support!

Example: Chromebook C201

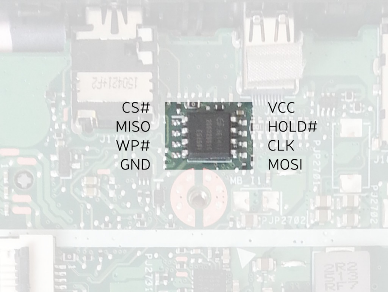
Chromebook C201: overview

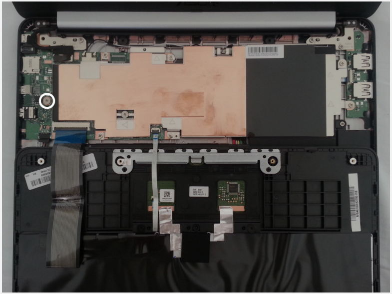
- Asus Chromebook laptop from 2015
- RK3288 platform
- No documentation or schematics
- No signature checks
- Coreboot support *(upstream)*
- Linux support *(downstream)*
- Free Embedded Controller firmware

Reflash all the things!

Chromebook C201: code execution (SoC)

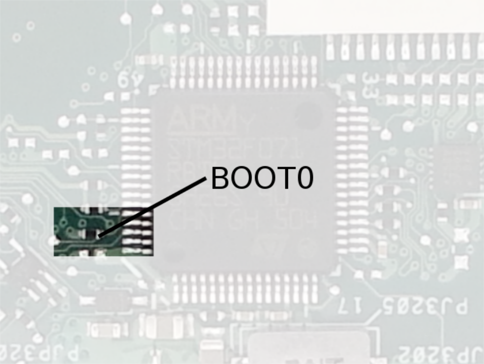
- Bootup software on SPI flash
- Hardware-protected part of the flash
- The screw!





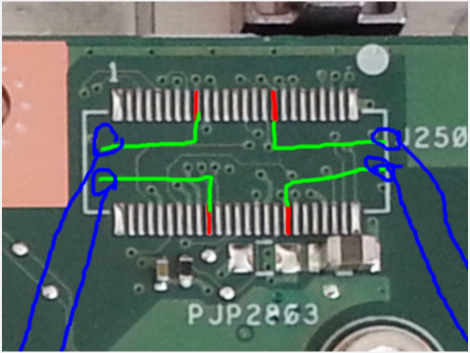
Chromebook C201: code execution (EC)

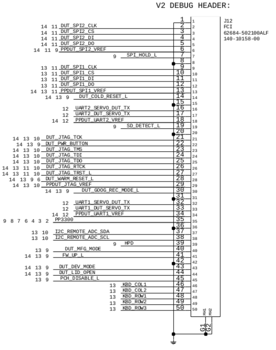
- BOOT0 pin (pull-down to reflash from UART)
- Finding the pull-up resistor



Chromebook C201: debugging

- Serial console: UART (both SoC and EC)
- Exported on Servo header
- Documented pinout





Example: G505s KB9012 Embedded Controller

G505s KB9012 Embedded Controller: overview

- Lenovo laptop from 2013
- AMD fam15h platform
- Technical documentation *(schematics)*
- Coreboot support

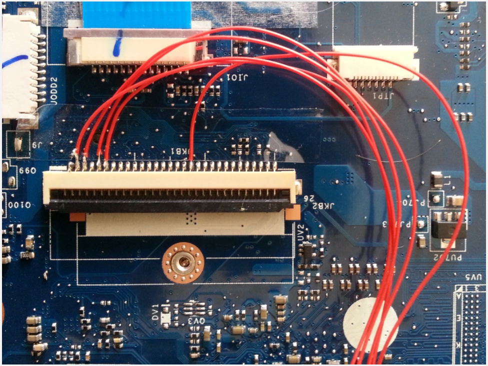
Embedded Controller:

- KB9012 EC from ENE
- Technical documentation *(datasheet)*
- 8051 CPU with controllers
- Internal storage

G505s KB9012 Embedded Controller: code execution

According to the datasheet:

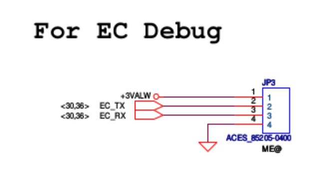
- LPC interface for reflashing
- External EDI (SPI-like) interface, exported on keyboard pins

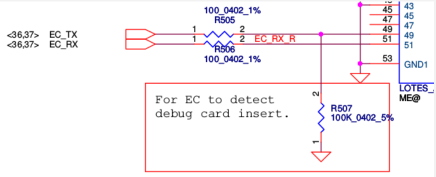


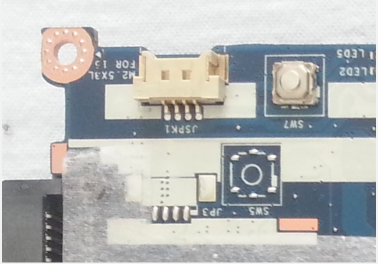
Flashrom support (pending review)!

G505s KB9012 Embedded Controller: debugging

- Serial console: UART
- EC debug interface, PCI-e pins







Replacements installation for end users

Once a working free software replacement is ready:

- Easiness of the installation process
- Required skills for the operation
- Risk of bricking the device

What we can do to reduce the pain:

- Providing clear and complete documentation
- Clearly mentioning the required skills
- Encouraging local organizations: Free software user groups, Hackerspaces