An overview of Replicant development

Replicant

Paul Kocialkowski
paulk@replicant.us

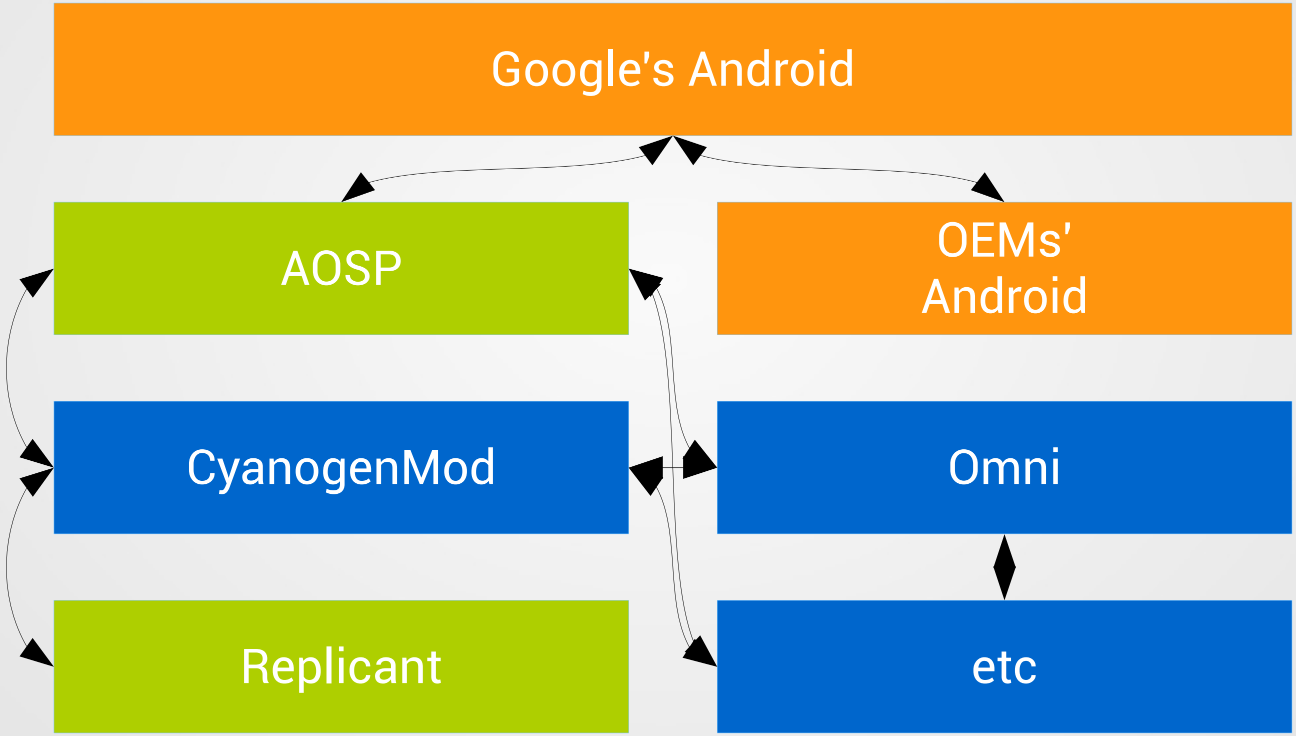
Wednesday 9th July 2014

RMLL
MONTPELLIER 2014

Le libre et vous !
15èmes Rencontres Mondiales
du Logiciel Libre
Du 5 au 11 juillet 2014

Taking a closer look at Android

Android is actually a family of operating systems:



Proprietary Android versions
Open source Android versions
Fully free Android versions

Android and Replicant

Replicant is a **fully free** Android version running on several mobile devices

Regarding Android and freedom:

- AOSP is close to fully free but doesn't run on devices
- Community versions use proprietary parts
- Proprietary parts used for communication with the hardware

Well, I see that people have figured out why I'm quitting AOSP.

There's no point being the maintainer of an Operating System that can't boot to the home screen on its flagship device for lack of GPU support, especially when I'm getting the blame for something that I don't have authority to fix myself and that I had anticipated and escalated more than 6 months ahead.

Jean-Baptiste Quéru, August 2013

Android and Replicant

Proprietary components:

- Never shipped with the system
- Replaced when possible, avoided when not

Replicant aims to be usable daily:

- Basic functionalities must work (**graphics**, **audio**, **telephony**)
- When proprietary: replace modules with free software

Biggest part of the work on Replicant: reverse engineering

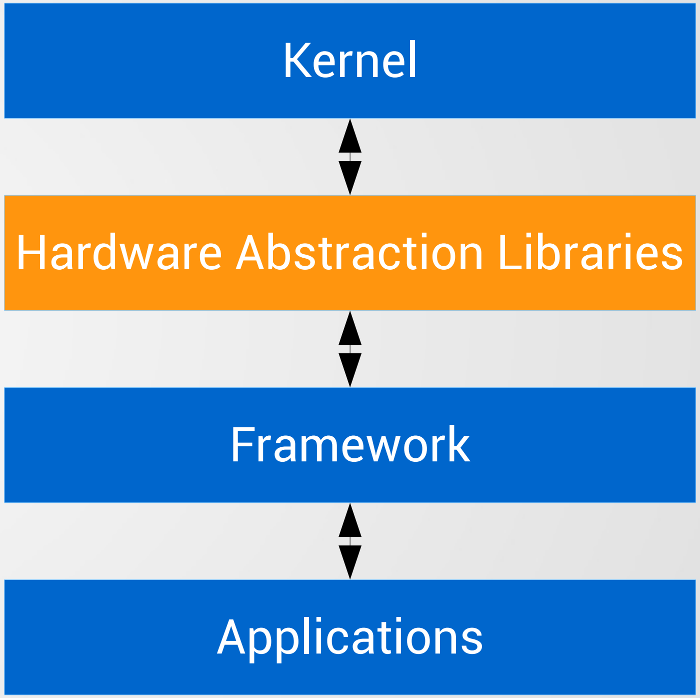
- Understanding how the proprietary components work
- Writing free software replacements

Replicant doesn't deal with:

- Graphics acceleration (Freedreno, Lima)
- Firmwares
- Modem operating system

Proprietary HALs

- Kernel drivers
- libc syscalls
  bionic
- Hardware modules
  struct sensors_module_t HAL_MODULE_INFO_SYM;
- Modules interface (dlopen)
  hardware/libhardware/hardware.c
- Modules headers
  hardware/libhardware/include/hardware/
- Framework Java JNI
  frameworks/base/services/jni/
- Framework
  frameworks/base/
- Applications



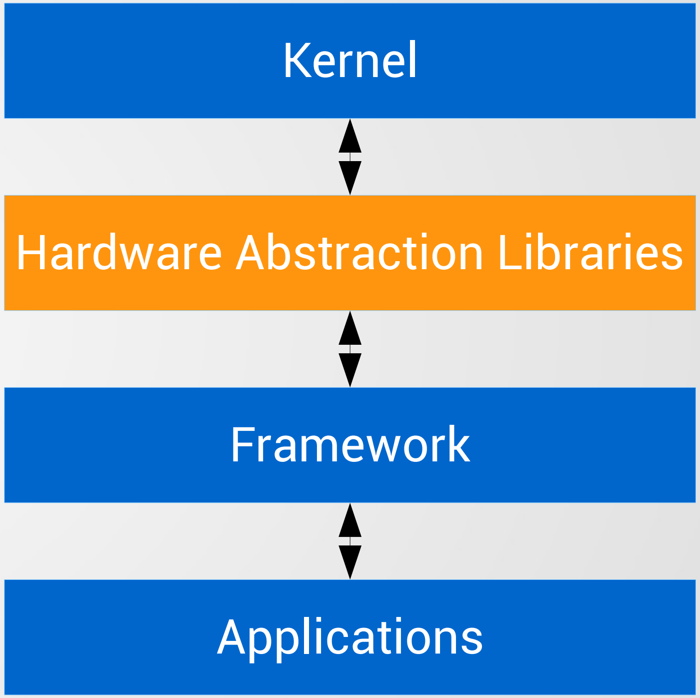
Free components
Proprietary component

Proprietary HALs

- Other specific interfaces: Wi-Fi, vibrator, EGL
- Modules with dedicated libraries: Bluetooth, NFC
- Proprietary kernel drivers are rare

Not all the modules are proprietary:

- Google Nexus devices
- Chips manufacturers: CodeAurora, Omapzoom, AOSP
- Upstream community work



Free components
Proprietary component

Proprietary HALs

Always proprietary modules:

- EGL (graphics acceleration)
- RIL (telephony)
- GPS

Still a considerable amount (I9300):

sbin/cbd
system/bin/bintvoutservice
system/bin/gps.cer
system/bin/lpmkey
system/bin/playlpm
system/lib/egl/libEGL_mali.so
system/lib/egl/libGLESv1_CM_mali.so
system/lib/egl/libGLESv2_mali.so
system/lib/libMali.so
system/lib/libMcClient.so
system/lib/libMcRegistry.so
system/lib/libMcVersion.so
system/lib/libQmageDecoder.so
system/lib/libTVOut.so
system/lib/libUMP.so
system/lib/libcec.so
system/lib/libddc.so
system/lib/libedid.so
system/lib/libfimc.so
system/lib/libfimg.so
system/lib/libhdmi.so
system/lib/libhdmiclient.so
system/lib/libhwconverter.so
system/lib/libhwjpeg.so
system/lib/libquramimagecodec.so
system/lib/libsecnativefeature.so
system/lib/libtvout_jni.so
system/lib/libtvoutinterface.so
system/lib/libtvoutservice.so
system/lib/libvdis.so
system/vendor/lib/drm/libdrmwvmplugin.so
system/vendor/lib/libWVStreamControlAPI_L1.so
system/vendor/lib/libwvdrm_L1.so
system/vendor/lib/libwvm.so
system/bin/gpsd
system/lib/hw/gps.exynos4.so
system/lib/hw/vendor-camera.exynos4.so
system/lib/hw/sensors.smdk4x12.so
system/lib/libakm.so
system/lib/libsec-ril.so

About this talk

Aim of this talk:

- Not so much about the stakes or the result
- All about the process
- Reverse engineering looks hard
- Reverse engineering can be hard
- Reverse engineering is not so hard on Replicant

What **you** need to get involved:

- Read/write C code, makefiles, git
- Ability to keep going, handle failure and frustration
- Time

Skills: not so much...

RE 101: Warming up

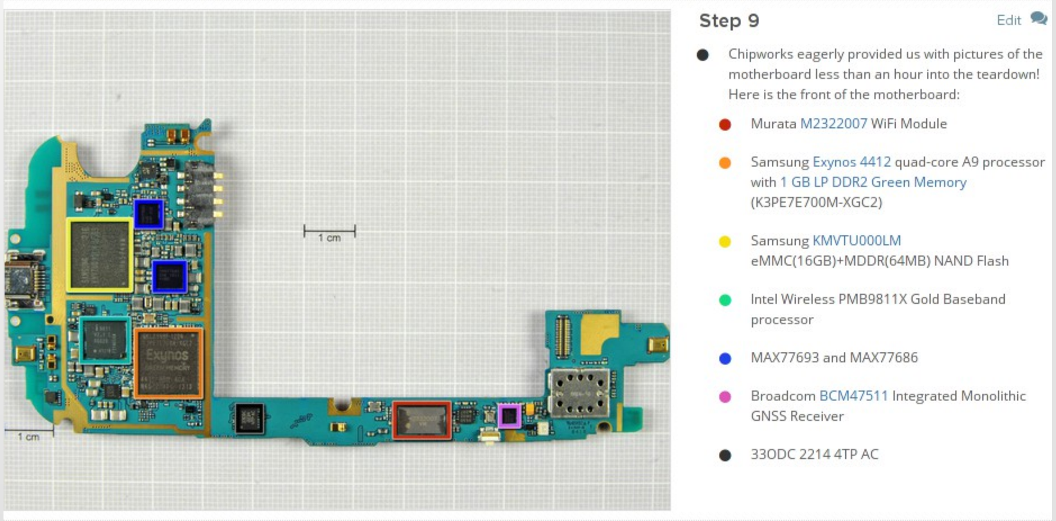
So we've got a proprietary module: let's get to the bottom of this!

Working together:

- Find other motivated **people**!
- Contact other developers who worked on similar things

Find out more about the **hardware**: chip **manufacturer** and **name:**

- Kernel sources have that information:
  arch/arm/configs/cyanogenmod_i9300_defconfig: CONFIG_SENSORS_AK8975C=y
- **Teardowns** can help
  iFixit
- Anything else:
  Wikipedia, manufacturer website



Step 9

- Chipworks eagerly provided us with pictures of the motherboard less than an hour into the teardown! Here is the front of the motherboard:
  - Murata M2322007 WiFi Module
  - Samsung Exynos 4412 quad-core A9 processor with 1 GB LP DDR2 Green Memory (K3PE7E700M-XGC2)
  - Samsung KMVTU000LM eMMC(16GB)+MDDR(64MB) NAND Flash
  - Intel Wireless PMB9811X Gold Baseband processor
  - MAX77693 and MAX77686
  - Broadcom BCM47511 Integrated Monolithic GNSS Receiver
  - 33ODC 2214 4TP AC

RE 101: Warming up

Once the relevant chip is identified, time to look for documentation:

- Technical literature on the subject
  learn about the basic concepts involved
- Search for **datasheets**, **manuals**
  look for websites selling the chip (DigiKey, SparkFun)
- Look up the chip manufacturer's website, see what you find
- Search for a reference software implementation
  just in case you're feeling lucky
- Look for any other available resource and code about the chip
- If there is a dedicated kernel driver, read it, headers too
  figure out what it does and doesn't do

In case of advanced desperation:

- Politely ask either the phone or the chip manufacturer

For the question you are asking, foo is not in a position to provide details of the lux formula we addressed with bar phone team.

RE 101: Warming up

At this point, with some luck, you may have an idea of what the proprietary software might be doing:

- I/O with the kernel (camera, audio)
- Communication protocol (GPS, RIL)
- Algorithms and maths (sensors)

Still have to figure out the magic that makes it work!

Time to look at the proprietary binary:

- Make sure this is legal in your case
  Europe: article 6 of the 1991 EU Computer Programs Directive
- Always start by looking at logs of the binary:
  adb logcat
- Try to make logs as verbose as possible:
  command line arguments, configuration files

RE 101: Static analysis

Static look at the binary:

- Always start with *strings*:
  spot debug strings, function names, error messages
- Decompile the program: *objdump -Dslx*
  helps understanding the structure of the program
- More advanced techniques: *radare2*

Android binaries are usually not obfuscated:

- Function names are preserved
- Often not stripped

Static analysis:

- Will help understand how things roll
- Can help figure out static data
- Will not tell much about the overall magic
  … unless you're very good at reading assembly!

RE 101: Dynamic analysis

Let's run the proprietary binary:

- Trace the I/O (syscalls) with *strace*
  that can be enough to figure it all out!
- Module loaded by the framework: don't trace the framework
  write a wrapper, trace the wrapper
- If any, make the kernel driver very verbose:
  trace every relevant function and I/O (ioctl, read/write, transport)
  hexdump any relevant data that comes through
- If maths are involved, trace in and out values and parameters
- Force specific values (kernel driver, wrapper)
- More advanced techniques: *gdb*

If there is no dedicated kernel driver (e.g. UART)

- *strace* shall be enough

RE 101: Figuring out the magic

Hopefully, the analysis should provide enough material:

- To understand what the program is expected to do
- To understand each step of the program
- To understand the physical meaning of things
- To figure out the relation between in and out values use spreadsheet software and guess equations!

It usually doesn't work at first try:

- Try the various techniques during the process
- Try another binary for the same chip
- Take some sleep
- Take a step back

It doesn't always work:

- Not enough material is available
- Manufacturers aren't friendly

Samsung IPC, Nexus S boot

Samsung IPC protocol

- Logs from the device:

```
E/RIL     (  131): ===== HDLC DUMP =====
E/RIL     (  131): 12 00 FF FF 08 05 03 FF 02
E/RIL     (  131): 01 00 00 00 00 00 00 00 00
E/RIL     (  131): ===================
[…]
E/RIL     (  131): RX: (M)IPC_NET_CMD  (S)IPC_NET_REGIST  (T)IPC_CMD_NOTI  len:12 mseq:ff aseq:ff
E/RIL     (  131): RX: ---- DATA BEGIN ----
E/RIL     (  131): RX: FF 02 01 00 00 00 00 00 00 00 00
E/RIL     (  131): RX: ---- DATA  END ----
```

We can already figure out part of the data!

Looking at different messages prefixed with IPC_NET:

```
E/RIL     (  131): ===== HDLC DUMP =====
E/RIL     (  131): 12 00 FF FF 08 05 03 FF 02
E/RIL     (  131): 01 00 00 00 00 00 00 00 00
E/RIL     (  131): ===================
E/RIL     (  131): RX: (M)IPC_NET_CMD  (S)IPC_NET_REGIST  (T)IPC_CMD_NOTI  len:12 mseq:ff aseq:ff
E/RIL     (  131): ===== HDLC DUMP =====
E/RIL     (  131): 12 00 00 04 08 03 02 03 04
E/RIL     (  131): FF 40 36 35 40 35 23 00 00
E/RIL     (  131): ===================
E/RIL     (  131): RX: (M)IPC_NET_CMD  (S)IPC_NET_SERVING_NETWORK  (T)IPC_CMD_RESP  len:12 mseq:0 aseq:4
```

IPC_NET is 0x08! The next byte is specific to the command.

Samsung IPC, Nexus S boot

Samsung IPC protocol

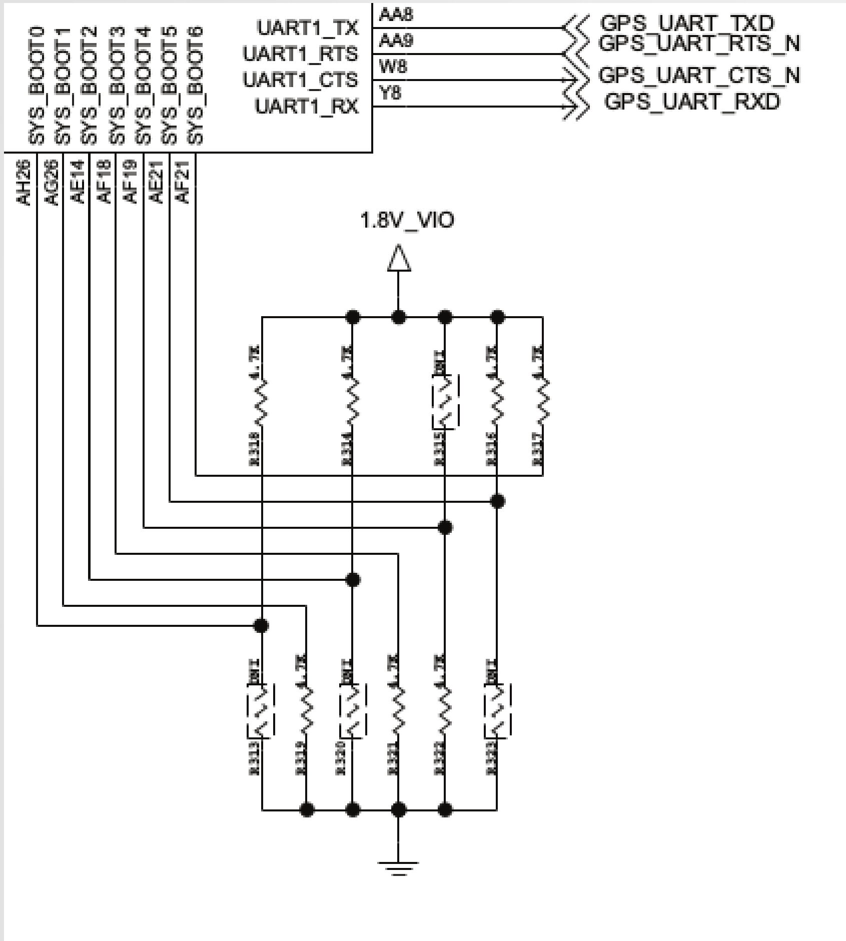
- IPC header implementation:

```
struct ipc_fmt_header {
    unsigned short length;
    unsigned char mseq;
    unsigned char aseq;
    unsigned char group;
    unsigned char index;
    unsigned char type;
} __attribute__((__packed__));

E/RIL     (  131): ===== HDLC DUMP =====
E/RIL     (  131): 12 00 FF FF 08 05 03 FF 02
E/RIL     (  131): 01 00 00 00 00 00 00 00 00
E/RIL     (  131): ===================
```

Rest of the packet: specific to each message, logs can help!

OMAP3 boot mode



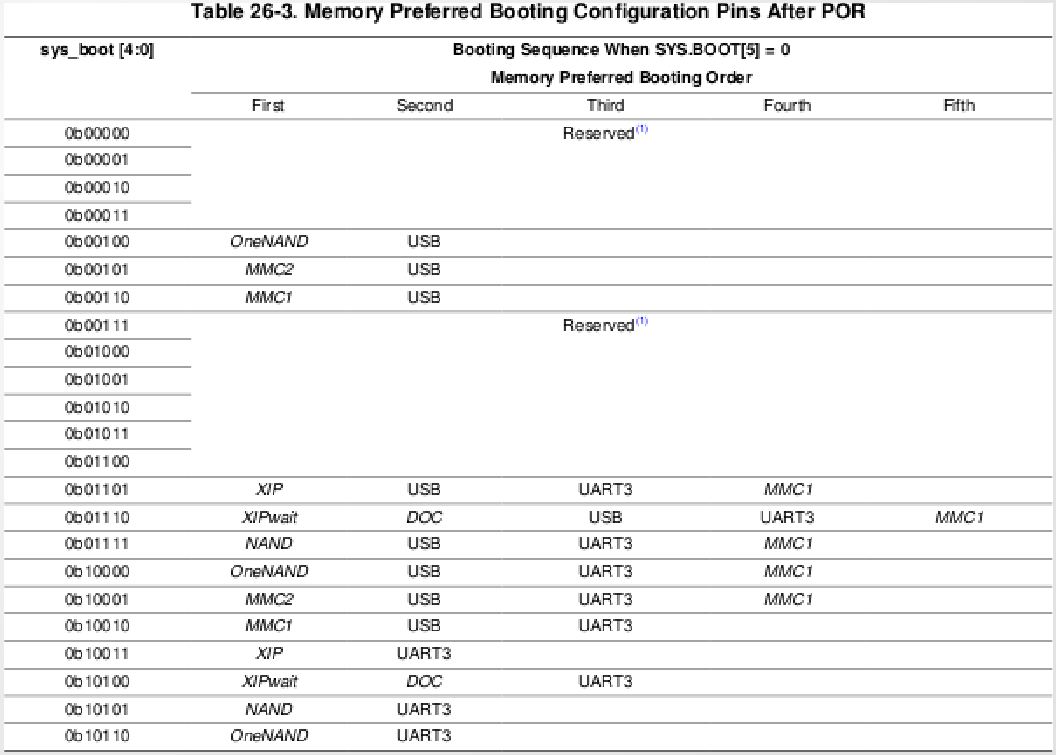
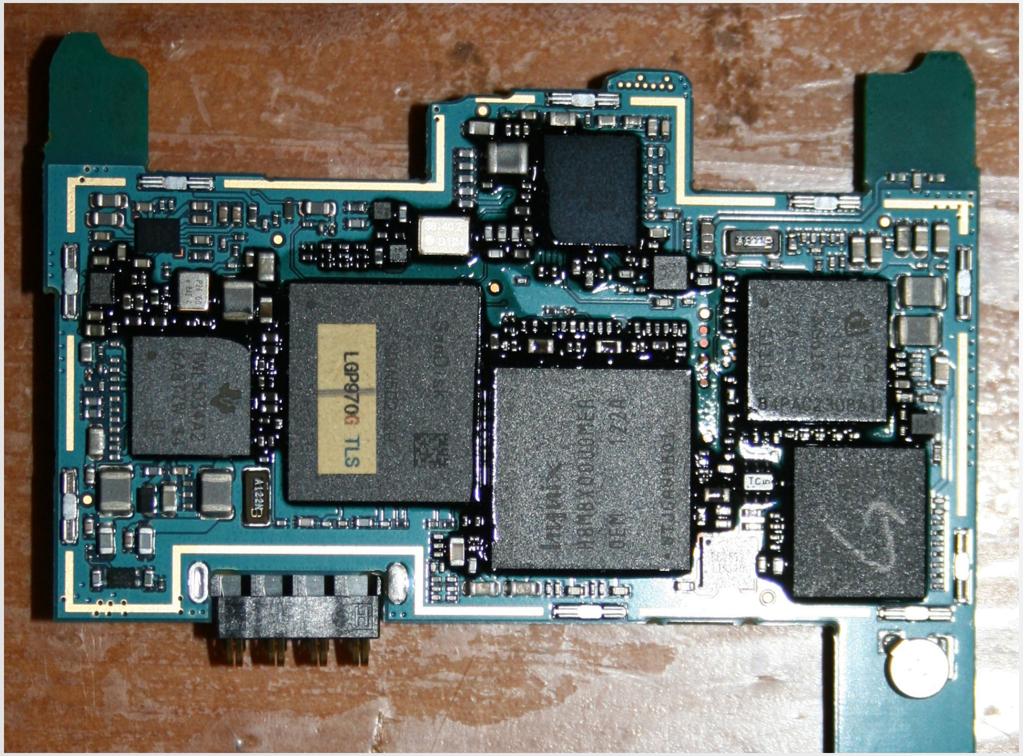
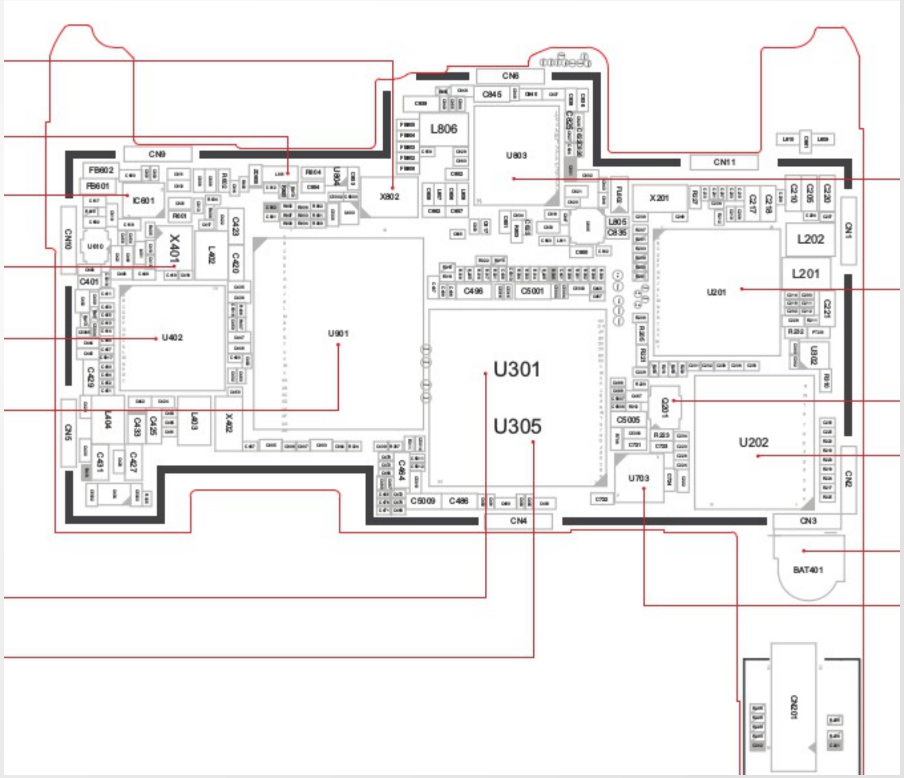


Table 26-3. Memory Preferred Booting Configuration Pins After POR

sys_boot [4:0]	Booting Sequence When SYS.BOOT[5] = 0 Memory Preferred Booting Order				
	First	Second	Third	Fourth	Fifth
0b00000			Reserved[1]		
0b00001					
0b00010					
0b00011					
0b00100	*OneNAND*	USB			
0b00101	*MMC2*	USB			
0b00110	*MMC1*	USB			
0b00111			Reserved[1]		
0b01000					
0b01001					
0b01010					
0b01011					
0b01100					
0b01101	*XIP*	USB	UART3	*MMC1*	
0b01110	*XIPwait*	*DOC*	USB	UART3	*MMC1*
0b01111	*NAND*	USB	UART3	*MMC1*	
0b10000	*OneNAND*	USB	UART3	*MMC1*	
0b10001	*MMC2*	USB	UART3	*MMC1*	
0b10010	*MMC1*	USB	UART3		
0b10011	*XIP*	UART3			
0b10100	*XIPwait*	*DOC*	UART3		
0b10101	*NAND*	UART3			
0b10110	*OneNAND*	UART3			

OMAP3 boot mode





OMAP3 boot mode

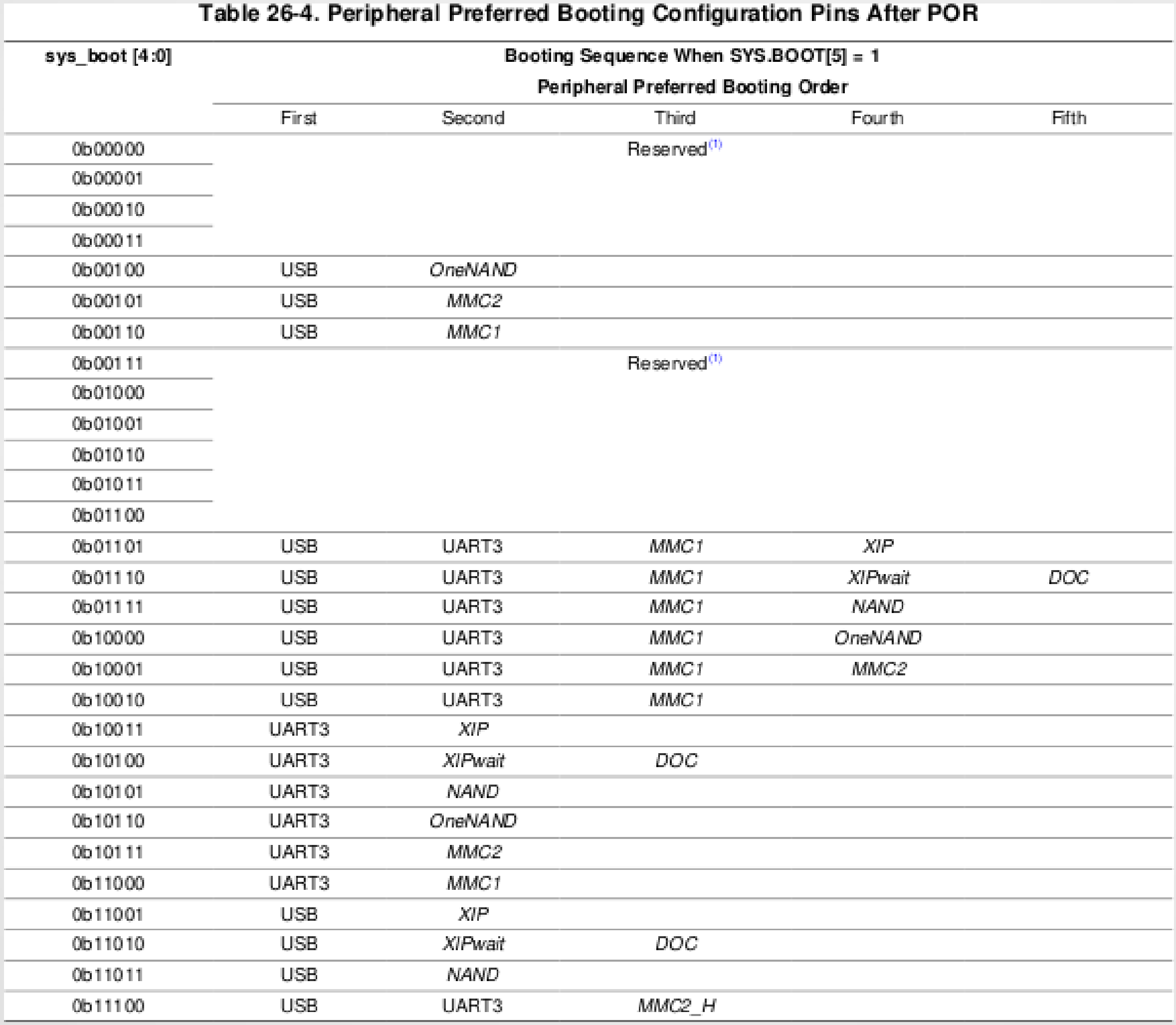


Table 26-4. Peripheral Preferred Booting Configuration Pins After POR

sys_boot [4:0]	**Booting Sequence When SYS.BOOT[5] = 1**				
	Peripheral Preferred Booting Order				
	First	Second	Third	Fourth	Fifth
0b00000	Reserved[1]				
0b00001					
0b00010					
0b00011					
0b00100	USB	*OneNAND*			
0b00101	USB	*MMC2*			
0b00110	USB	*MMC1*			
0b00111	Reserved[1]				
0b01000					
0b01001					
0b01010					
0b01011					
0b01100					
0b01101	USB	UART3	*MMC1*	*XIP*	
0b01110	USB	UART3	*MMC1*	*XIPwait*	*DOC*
0b01111	USB	UART3	*MMC1*	*NAND*	
0b10000	USB	UART3	*MMC1*	*OneNAND*	
0b10001	USB	UART3	*MMC1*	*MMC2*	
0b10010	USB	UART3	*MMC1*		
0b10011	UART3	*XIP*			
0b10100	UART3	*XIPwait*	*DOC*		
0b10101	UART3	*NAND*			
0b10110	UART3	*OneNAND*			
0b10111	UART3	*MMC2*			
0b11000	UART3	*MMC1*			
0b11001	USB	*XIP*			
0b11010	USB	*XIPwait*	*DOC*		
0b11011	USB	*NAND*			
0b11100	USB	UART3	*MMC2_H*		

Getting started on Replicant hacking

Replicant is currently driven by one developer!

- Financial contributions are fine
- We need brains!
- We need new functionalities
- We need new devices supported
- Lots of different tasks:
  http://redmine.replicant.us/projects/replicant/wiki/Tasks
- Easy ports exist:
  Galaxy Tab (P1000), Galaxy Note 10.1 (N8000)

Not every port or task requires reverse engineering!

Getting started on Replicant hacking

Steps to get started:

- Get a supported device, install Replicant on it
- Grab the Replicant source code
- Build Replicant for your device
  Add small modifications, play around with the source
- Get familiar with it
- Learn about the Replicant development process:
- Complete a task or work on anything else you want to improve!

Once you're familiar, start a new port:

- Evaluate the device carefully
- If the port is doable, follow the guide:
- Once the port is usable, push it to Replicant

Getting started on Replicant hacking

Resources at the Replicant wiki:

- Replicant status
  http://redmine.replicant.us/projects/replicant/wiki/ReplicantStatus
- Installation guides
  http://redmine.replicant.us/projects/replicant/wiki#Installing-Replicant
- Build guides
  http://redmine.replicant.us/projects/replicant/wiki#Building-Replicant
- Developer guide
  http://redmine.replicant.us/projects/replicant/wiki/DeveloperGuide
- Porting guide
  http://redmine.replicant.us/projects/replicant/wiki/Replicant40PortingGuide
- List of tasks
  http://redmine.replicant.us/projects/replicant/wiki/Tasks

Replicant

Learn more about Replicant:

- Website: http://www.replicant.us/
- Wiki/tracker: http://redmine.replicant.us/
- Source code: http://gitorious.org/replicant

Get in touch with us:

- Forums
- Mailing list
- IRC channel: #replicant at freenode

During the LSM/RMLL:

- Free Your Android Workshop (TD011, Polytech building)
- ARM devices and your freedom (Wednesday 11:40)

That's all Folks!