# Replicant and bootloaders

Denis 'GNUtoo' Carikli

July 25, 2019

Freedom Privacy and Security:

- The Replicant website has a page (`https://replicant.us/freedom-privacy-security-issues.php`) that has more details on freedom, privacy and security issues commonly found in smartphones.

- That page consists mainly of HTML, its source code is available in `https://git.replicant.us/replicant/website.git`, and patches are reviewed on the Replicant mailing list.

- The README also has some information on a very easy way to deploy the website locally to test changes.
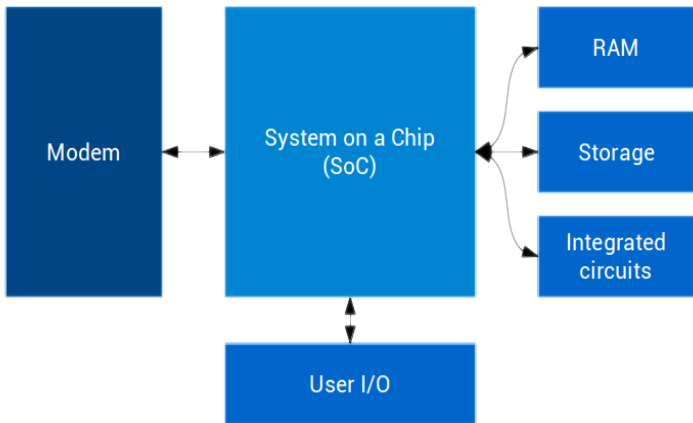
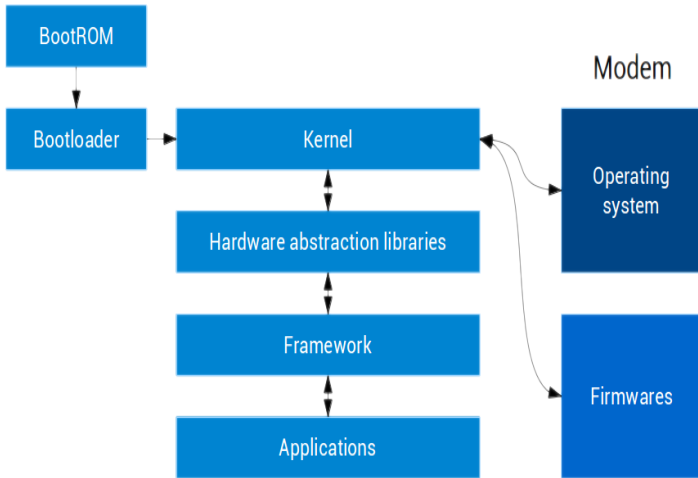Hardware-side overview

Bootloader?

Here it is Equivalent to:

- The BIOS + GRUB
- UEFI: UEFI can load the Linux kernel directly

Why are bootloader needed on smartphones and tablets?

By itself, the hardware is usually capable of loading and running a limited amount of code from the internal storage, however most of the hardware doesn't even work at boot, including:

- The Display
- The RAM
- The buttons

SoC

BootROM

Bootloader

Kernel

Hardware abstraction libraries

Framework

Applications

Modem

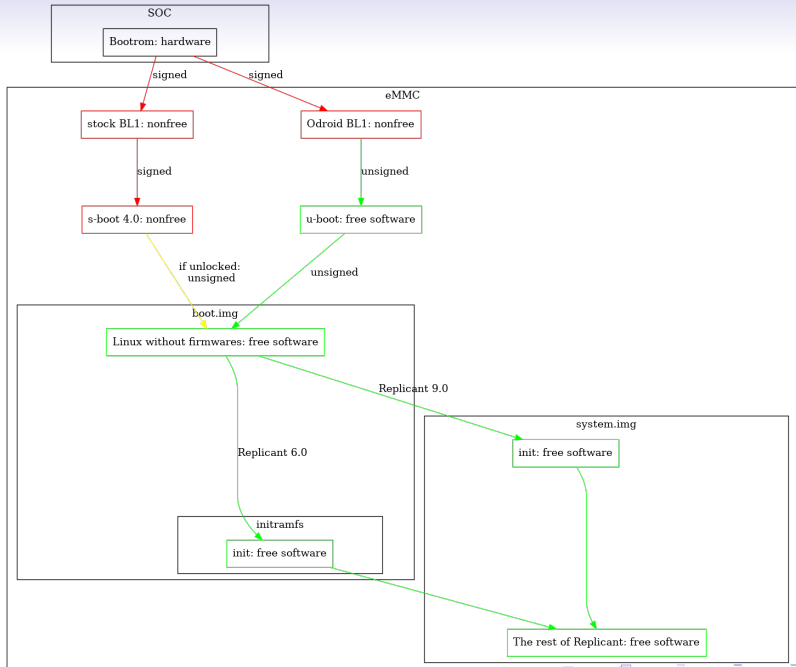Operating system

Firmwares

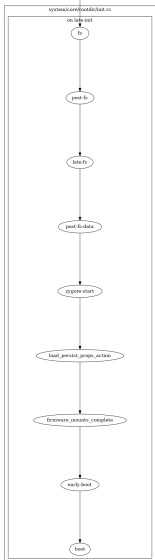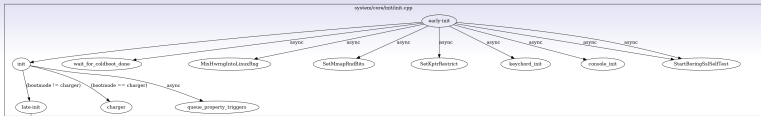Software-side overview

Integrated circuits

Bootrom→Bootloader→ Replicant

Samsung devices Bootrom→BL1→s-boot 4.0→Replicant
Applies at least to the following devices:

- Galaxy S2 (I9100)
- Galaxy SIII (I9300)
- Galaxy SIII 4G (I9305)
- Galaxy Note II (N7100)
- Galaxy Note II 4G (N7105)

General nonfree bootloader issues:

The fact that the bootloader is mostly invisible to the user doesn't magically make its related freedom issues disapear.

As with all software, it is required for it to be free software in order to respect users freedom.

Examples of issues:

- As it is able to modify the operating system it's supposed to load and execute, and that the code is nonfree, we cannot give any guarantees that it doesn't do that. Some BIOS do that with computrace for instance.

- It could refuses to boot if you changed the software or hardware.
    - Apple's ipad and iphones?
    - BIOS/UEFI with the WiFi card.

- Or loads another extra software you don't whish.

- Be able to tinker with boot is also important (ways to boot, security, etc)

Examples of tinkering:

- Adding filesystems (recent ext4).
- Adding dual boot.
- Adding new recovery ways (A/B recovery).
- Having a single bootloader for multiple devices.
- Having a single Android image for multiple devices.
- Fixing hardare security issues when they are found (rowhammer).
- Getting failed boot logs.
- Automatic testing of Replicant (install from the network, boot on it).

"Midas":

- Galaxy SIII (I9300)
- Galaxy SIII 4G (I9305)
- Galaxy Note II (N7100)
- Galaxy Note II 4G (N7105)

Issues found on Midas with BL1→s-boot 4.0→Linux:

- Nonfree BL1 and s-boot
- Doesn't take the commandline arguments from the boot.img
- Doesn't support the devicetree
- Initialize the MMU
- Loads and run a second OS (MobiCore) in TrustZone that is nonfree.

TrustZone

- Has more hardware privileges than the Linux kernel
- Implementing a rootkit in TrustZone is possible[3]
- "the TEE has [..] monotonic clock that ticks in suspend[2]"
- SOC specific and lacks SOC documentation.
- Discussions on weather free software in TrustZone is desirable are planned in this conference.

Upstream Linux bootloader requirements
Documentation/arm/Booting:

The MMU must be off.

Instruction cache may be on or off.

Data cache must be off.

MMU on in practice:

- Requires a patch to boot: "ANDROID: arm: decompressor: Flush tlb before swiching domain 0 to client mode"

- I spent between several days and a week bisecting the commit that broke booting between 5.1 and 5.2-rc1 (merges, rebases, non-compiling commits, etc).

- Not substainable: What happens if in the future new maintainers don't have the time or skills to bisect breakages?

- Most of the work on Android 9 is still re-usable on future devices that would have free software bootloaders.

Sharp Zaurus (SA11x0)

```
$ ls arch/arm/boot/compressed/
atags_to_fdt.c debug.S efi-header.S
head-sa1100.S head-xscale.S
ll_char_wr.S misc.c piggy.S
vmlinux.lds.S big-endian.S decompress.c
head.S head-sharpsl.S libfdt_env.h
Makefile misc.h string.c
```

head-sa1100.S

```
    @ Data  cache  might  be  active.
    @ Be  sure  to  flush  kernel  binary  out  of  the  cache,
    @ whatever  state  it  is,  before  it  is  turned  off.
    @ This  is  done  by  fetching  through  currently  executed
    @ memory  to  be  sure  we  hit  the  same  cache.
    bic       r2,  pc,  #0x1f
    add       r3,  r2,  #0x4000             @ 16  kb  is  quite  enough...
1:  ldr       r0,  [r2],  #32
    teq       r2,  r3
    bne       1b
    mcr       p15,  0,  r0,  c7,  c10,  4   @ drain  WB
    mcr       p15,  0,  r0,  c7,  c7,  0    @ flush  I & D  caches

    @ disabling  MMU  and  caches
    mrc       p15,  0,  r0,  c1,  c0,  0    @ read  control  reg
    bic       r0,  r0,  #0x0d              @ clear  WB,  DC,  MMU
    bic       r0,  r0,  #0x1000            @ clear  Icache
    mcr       p15,  0,  r0,  c1,  c0,  0
```

In practice Russel King (ARM maintainer):

SA11x0 pre-dates the booting document, which came about because of the desire to make the kernel less dependent on the host CPU type. So "sa11x0 does it so we can do it" is really not an argument I ever want to see to justify this kind of stuff.

The booting requirements have been known since at least 2002, some SEVENTEEN years ago, and the problem was identified as buggy back in 2012. As far as I can see, nothing has changed.

Entering the kernel with the MMU on and optionally caches on is an inherently unsafe thing to do. The kernel would have been placed into RAM via the data cache, and then we're trying to execute code - unless the caches have been properly cleaned and invalidated, there is no guarantee that we'd even reach any instructions to do our own cache cleaning and invalidation. So, caches on is utter madness.

MMU on presents a problem: the kernel moves itself around during decompression - if it happens to move itself on top of the in-use page tables, then that would be really bad. There's another issue as well - if the page tables are already setup, and we create a different mapping for the virtual address range, the _only_ way to safely switch to that mapping is via a break-make arrangement, which means we need code to disable the MMU, flush it. It is not as simple as "a few extra instructions to flush TLBs" although that may work in the majority of cases. Architecturally, it is wrong.

Things can get even worse - what if the page tables are located where the kernel writes its own page tables - modifying the live tables and changing the type of the entries. Architecturally unpredictable behaviour may result.

What is written in Documentation/arm/Booting is not for our fun, it is there to spell out what the kernel requires to be able to boot reliably on hardware. If it isn't followed, then booting a kernel will be unreliable.

downstream u-boot port

- No MMU enabled at boot
- Gets the commandline arguments from the boot.img
- Support the devicetree. Even supports modifying it on the fly.
- No display support, but leds are supported

Issues found on Midas with BL1→u-boot→Linux:

- Nonfree and non-redistributable BL1

Licenses:

- Attribution-ShareAlike 3.0 Unported for the two images that comes from the Freedom Privacy and Security page.

- `https://creativecommons.org/licenses/by-sa/4.0/` for the rest.

https://replicant.us/
freedom-privacy-security-issues.php

https://source.android.com/security/
authentication/gatekeeper

https://hackinparis.com/data/slides/2013/
Slidesthomasroth.pdf