Running Mainline Linux on the Jolla C2

FOSS on Mobile - FOSDEM 2026

Affe Null

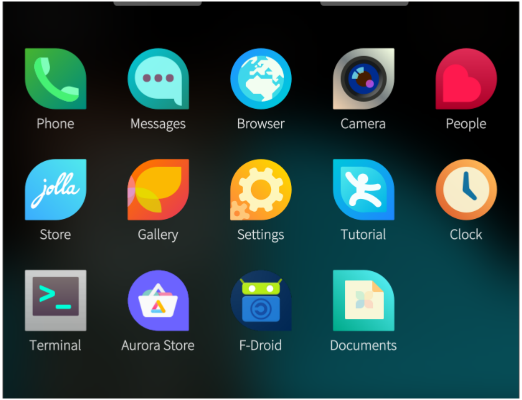
January 31, 2026

What is the Jolla C2?

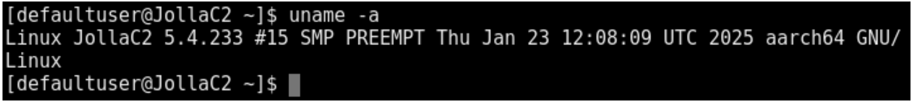
- Jolla smartphone released in 2024
- “Reference device” for Sailfish OS
- Partnership with Reeder (Turkish company)
- Mostly identical to Reeder S19 Max Pro S
- Based on Unisoc Tiger T606 (UMS9230) SoC

How does the phone run Linux?

- Shell access available in developer mode

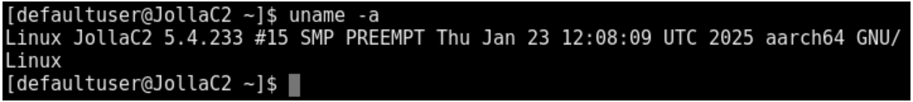


How does the phone run Linux?



```
[defaultuser@JollaC2 ~]$ uname -a
Linux JollaC2 5.4.233 #15 SMP PREEMPT Thu Jan 23 12:08:09 UTC 2025 aarch64 GNU/
Linux
[defaultuser@JollaC2 ~]$ 
```

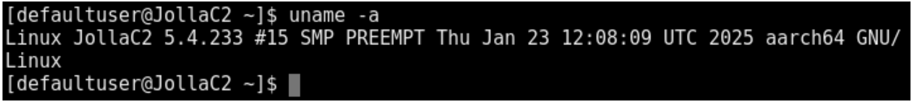
How does the phone run Linux?



```
[defaultuser@JollaC2 ~]$ uname -a
Linux JollaC2 5.4.233 #15 SMP PREEMPT Thu Jan 23 12:08:09 UTC 2025 aarch64 GNU/
Linux
[defaultuser@JollaC2 ~]$ 
```

- typical situation with downstream drivers made for Android

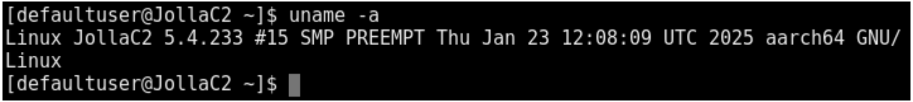
How does the phone run Linux?



```
[defaultuser@JollaC2 ~]$ uname -a
Linux JollaC2 5.4.233 #15 SMP PREEMPT Thu Jan 23 12:08:09 UTC 2025 aarch64 GNU/
Linux
[defaultuser@JollaC2 ~]$ 
```

- typical situation with downstream drivers made for Android
- Sailfish OS uses Android vendor HAL via libhybris (as expected)

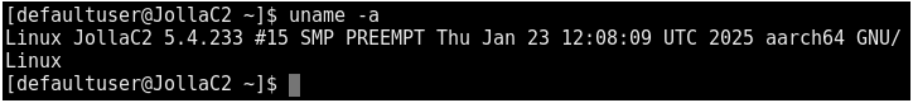
How does the phone run Linux?



```
[defaultuser@JollaC2 ~]$ uname -a
Linux JollaC2 5.4.233 #15 SMP PREEMPT Thu Jan 23 12:08:09 UTC 2025 aarch64 GNU/
Linux
[defaultuser@JollaC2 ~]$ 
```

- typical situation with downstream drivers made for Android
- Sailfish OS uses Android vendor HAL via libhybris (as expected)
- However, it does not strictly depend on libhybris or Android
  - roots in Nokia's Meego project
  - closer to traditional GNU/Linux userspace

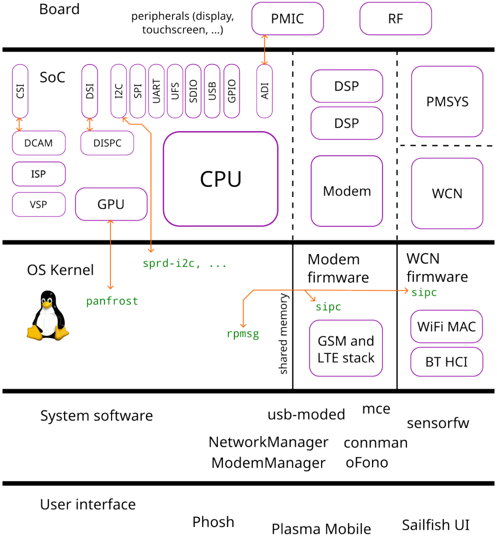
How does the phone run Linux?



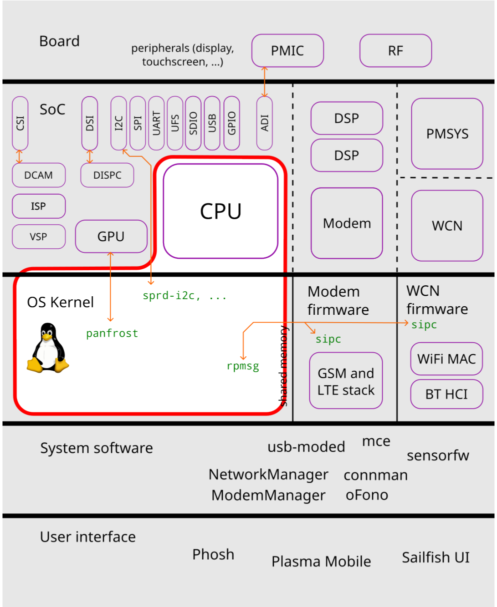
```
[defaultuser@JollaC2 ~]$ uname -a
Linux JollaC2 5.4.233 #15 SMP PREEMPT Thu Jan 23 12:08:09 UTC 2025 aarch64 GNU/
Linux
[defaultuser@JollaC2 ~]$ 
```

- typical situation with downstream drivers made for Android
- Sailfish OS uses Android vendor HAL via libhybris (as expected)
- However, it does not strictly depend on libhybris or Android
  - roots in Nokia's Meego project
  - closer to traditional GNU/Linux userspace
- Why not a mainline kernel?
  - a few drivers from Unisoc are already upstream
  - try something new

Overview


Board
peripherals (display, touchscreen, ...)
PMIC
RF
SoC
CSI
DSI
I2C
SPI
UART
UFS
SDIO
USB
GPIO
ADI
DSP
DSP
PMSYS
DCAM
DISPC
ISP
VSP
GPU
CPU
Modem
WCN
OS Kernel
sprd-i2c, ...
panfrost
rpmsg
shared memory
Modem firmware
sipc
GSM and LTE stack
WCN firmware
sipc
WiFi MAC
BT HCI
System software
usb-moded
mce
sensorfw
NetworkManager
connman
ModemManager
oFono
User interface
Phosh
Plasma Mobile
Sailfish UI

Overview



Board

peripherals (display, touchscreen, ...)

PMIC

RF

SoC

CSI

DSI

I2C

SPI

UART

UFS

SDIO

USB

GPIO

ADI

DCAM

DISPC

ISP

VSP

GPU

CPU

DSP

DSP

Modem

PMSYS

WCN

OS Kernel

sprd-i2c, ...

panfrost

rpmsg

shared memory

Modem firmware

sipc

GSM and LTE stack

WCN firmware

sipc

WiFi MAC

BT HCI

System software

usb-moded

mce

sensorfw

NetworkManager

connman

ModemManager

oFono

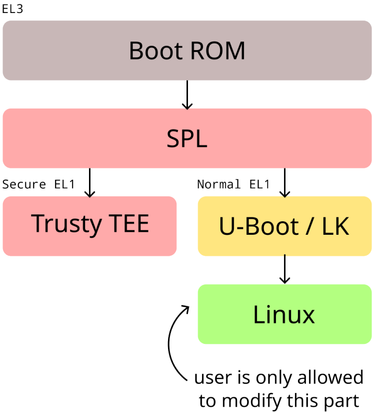
User interface

Phosh

Plasma Mobile

Sailfish UI

Simplified boot chain



EL3

Boot ROM

SPL

Secure EL1

Normal EL1

Trusty TEE

U-Boot / LK

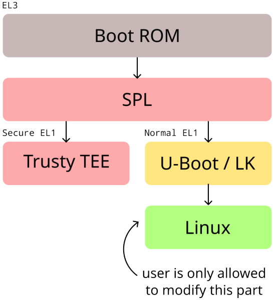
Linux

user is only allowed to modify this part

Mainlining - first steps

- Phone arrived in November 2024
- Step 1: flashing
  - `spd_dump` tool with CVE-2022-38694 exploit

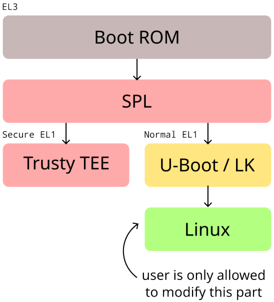
Simplified boot chain



Mainlining - first steps

- Phone arrived in November 2024
- Step 1: flashing
  - `spd_dump` tool with CVE-2022-38694 exploit
- Step 2: get kernel running

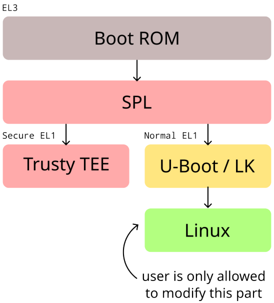
Simplified boot chain



Mainlining - first steps

- Phone arrived in November 2024
- Step 1: flashing
  - spd_dump tool with CVE-2022-38694 exploit
- Step 2: get kernel running
- Android bootloader does initial setup
  - starts display output from memory
  - powers up internal storage
  - unfortunately requires dtbo and messes with device tree

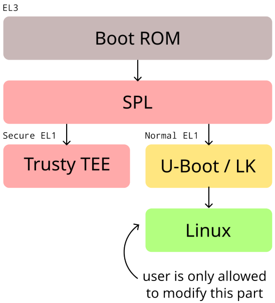
Simplified boot chain



Mainlining - first steps

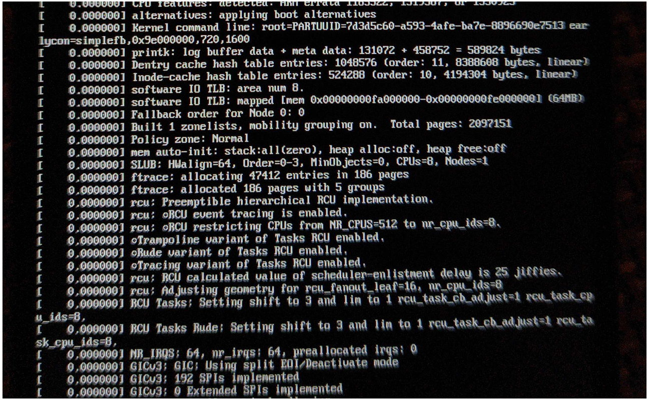
- Phone arrived in November 2024
- Step 1: flashing
  - spd_dump tool with CVE-2022-38694 exploit
- Step 2: get kernel running
- Android bootloader does initial setup
  - starts display output from memory
  - powers up internal storage
  - unfortunately requires dtbo and messes with device tree
- Chainloading U-Boot to simplify things
  - ignores Android device tree and provides its own
  - standard boot process
  - implement just enough drivers to load kernel

Simplified boot chain



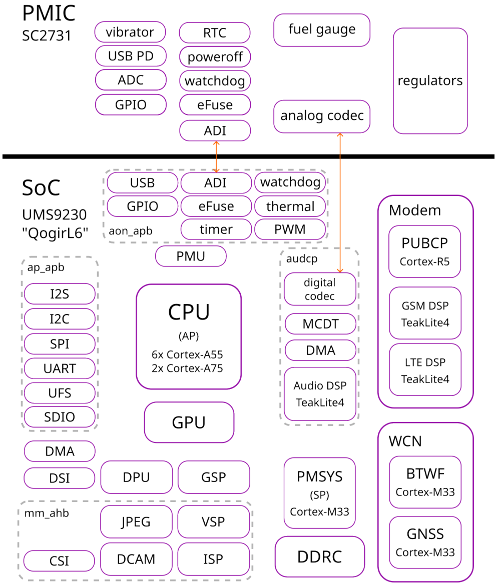
Debugging

- using UART would require disassembling phone
- easier to set display pixels in memory
- simple-framebuffer and simplefb earlycon patch



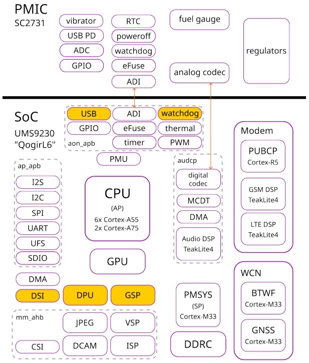
```
[    0.000000] CPU features: detected: ARM errata 1165522, 1319367, or 1530923
[    0.000000] alternatives: applying boot alternatives
[    0.000000] Kernel command line: root=PARTUUID=7d3d5c60-a593-4afe-ba7e-8896690e7513 ear
lycon=simplefb,0x9e000000,720,1600
[    0.000000] printk: log buffer data + meta data: 131072 + 458752 = 589824 bytes
[    0.000000] Dentry cache hash table entries: 1048576 (order: 11, 8388608 bytes, linear)
[    0.000000] Inode-cache hash table entries: 524288 (order: 10, 4194304 bytes, linear)
[    0.000000] software IO TLB: area num 8.
[    0.000000] software IO TLB: mapped [mem 0x00000000fa000000-0x00000000fe000000] (64MB)
[    0.000000] Fallback order for Node 0: 0
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages: 2097151
[    0.000000] Policy zone: Normal
[    0.000000] mem auto-init: stack:all(zero), heap alloc:off, heap free:off
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=8, Nodes=1
[    0.000000] ftrace: allocating 47412 entries in 186 pages
[    0.000000] ftrace: allocated 186 pages with 5 groups
[    0.000000] rcu: Preemptible hierarchical RCU implementation.
[    0.000000] rcu: oRCU event tracing is enabled.
[    0.000000] rcu: oRCU restricting CPUs from NR_CPUS=512 to nr_cpu_ids=8.
[    0.000000] oTrampoline variant of Tasks RCU enabled.
[    0.000000] oRude variant of Tasks RCU enabled.
[    0.000000] oTracing variant of Tasks RCU enabled.
[    0.000000] rcu: RCU calculated value of scheduler-enlistment delay is 25 jiffies.
[    0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=8
[    0.000000] RCU Tasks: Setting shift to 3 and lim to 1 rcu_task_cb_adjust=1 rcu_task_cp
u_ids=8.
[    0.000000] RCU Tasks Rude: Setting shift to 3 and lim to 1 rcu_task_cb_adjust=1 rcu_ta
sk_cpu_ids=8.
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[    0.000000] GICv3: GIC: Using split EOI/Deactivate mode
[    0.000000] GICv3: 192 SPIs implemented
[    0.000000] GICv3: 0 Extended SPIs implemented
```

The UMS9230 chipset



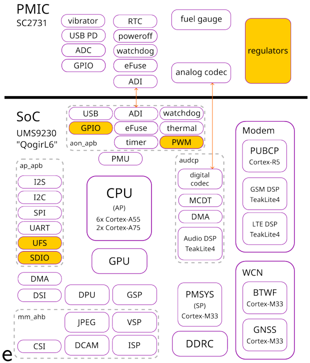
Mainlining – initial bringup

- Clock driver
  - already in mainline for older Unisoc SoCs
  - very similar to downstream
- USB is hard, but simplifies debugging a lot
  - PHY init sequence, obscure registers
  - old MUSB controller with additional DMA quirks
  - messy code both upstream and downstream
  - transfers sometimes get stuck in host mode
- Watchdog
  - phone reboots, seemingly no effect from driver (more on this later)
- Display driver
  - replaces simple-framebuffer
  - usually complicated
  - existing `drivers/gpu/drm/sprd` from 2021
  - many fixes and UMS9230 support added

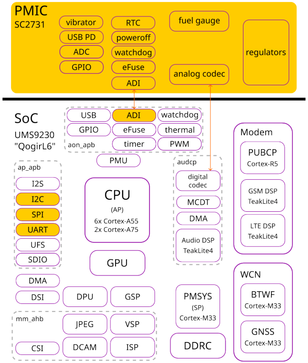


Mainlining - peripherals (1)

- SC2730 PMIC regulators
  - required for powering components including storage
  - MFD driver in mainline since 2021
  - merged downstream driver and existing SC2731 driver
- External Storage: SD card
  - existing driver (mainline since 2018) mostly usable
  - later fixes for high-speed operation
- Internal Storage: UFS
  - existing driver (mainline since 2022) for different SoC is incompatible
  - bootloader initializes hardware with low clock speed
  - restarting after clock change requires PHY init sequence
- GPIO driver (mainline since 2018) worked with almost no changes
  - some buttons or LEDs, depending on the device
  - various control pins for peripherals
- PWM backlight



Mainlining – peripherals (2)



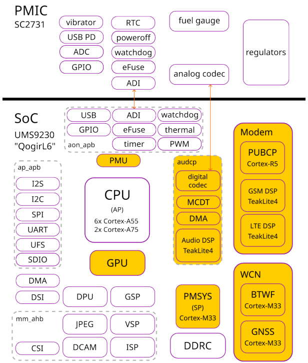
- SC2730 PMIC
  - already in mainline: vibrator, fuel gauge, RTC, ADC, ...
  - some drivers for older SC2731 PMIC adapted
  - bug fixes for fuel gauge driver
  - Type-C PD driver adapted from downstream, cleanups needed
  - separate watchdog in PMIC enabled on startup
- UART, I2C, SPI
  - copy-paste device tree nodes
  - match clocks with clock driver
- Charger (device-specific)
  - `sgm41511` (`bq25601` clone, existing driver works)
- Touchscreen driver (device-specific, usually I2C)
  - existing mainline drivers
  - downstream driver as reference
  - `icnl9916` driver for Jolla C2 was quite easy to write

Minimal example

https://storage.abscue.de/private/zImage/jolla-c2-simple-boot.mp4

Mainlining – SoC subsystems

- Power domains: new driver
- GPU supported by panfrost driver
- Remoteprocs
  - using firmware from Android
  - WCN (BTWF, GNSS) – Cortex-M33
  - Audio – Ceva TeakLite4 DSP (undocumented architecture)
  - Modem – Cortex-R5 and two TeakLite4 DSPs
  - PMSYS aka SP (sensor hub, GNSS, ...) – Cortex-M33
- SIPC protocol for communication with remoteprocs
  - shared memory ring buffer, character and packet-based variants
  - new rpmsg driver
- PMSYS also responsible for managing PMIC watchdog
  - `watchdog on` command needed
  - intercepts AP watchdog to trigger some kind of crash dump
- Bluetooth: mostly standard HCI

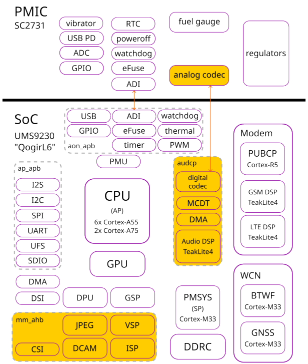


New WiFi driver

- first version took about one month to develop
- cfg80211 driver using original firmware
- downstream driver as reference
- features implemented:
  - scanning
  - regulatory domains
  - connecting to networks
  - WPA key management
  - power saving
- still not fully working
- features missing: AP mode, QoS, IBSS, P2P, ...
- only supports integrated WiFi
  - some phones use external module via SDIO

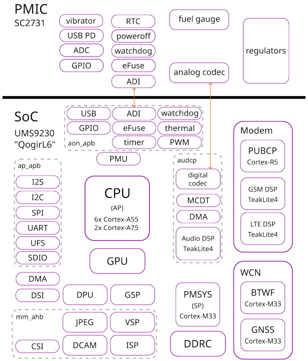
Mainlining steps - multimedia features

- New drivers needed for almost everything
- Audio
  - bootloader starts DSP for some reason
  - AP ↔ DSP communication
  - SoC ↔ PMIC link
  - PMIC analog codec
  - ALSA machine driver
  - call audio via codec2codec link
- Camera
  - CSI PHY
  - DCAM: raw capture with some processing
  - separate ISP: memory-to-memory, debayering, additional processing
  - open-source driver with list of registers
  - device-specific sensor drivers
- Hardware video encoding/decoding
  - not implemented yet
  - proprietary



Mainlining steps - more features

- GNSS: new driver
  - baseband is part of WCN subsystem
  - NMEA data generated by PMSYS remoteproc firmware
  - custom commands for A-GNSS not implemented yet
- Sensors: new IIO driver
  - I2C controlled by PMSYS sensor hub
  - firmware obviously designed with Android in mind
  - sends floating-point numbers requiring conversion in kernel
- CPU frequency scaling
  - implemented in proprietary firmware that runs on the AP
  - new driver for querying and setting frequencies
- Other power management features
  - PSCI suspend worked right away
  - DVFS implemented in various remoteprocs, not supported yet



PMIC
SC2731
vibrator
USB PD
ADC
GPIO
RTC
poweroff
watchdog
eFuse
ADI
fuel gauge
analog codec
regulators

SoC
UMS9230
"QogirL6"
USB
ADI
watchdog
GPIO
eFuse
thermal
aon_apb
timer
PWM
PMU
ap_apb
I2S
I2C
SPI
UART
UFS
SDIO
DMA
DSI
CPU
(AP)
6x Cortex-A55
2x Cortex-A75
GPU
audcp
digital codec
MCDT
DMA
Audio DSP
TeakLite4
DPU
GSP
mm_ahb
JPEG
VSP
CSI
DCAM
ISP
PMSYS
(SP)
Cortex-M33
DDRC
Modem
PUBCP
Cortex-R5
GSM DSP
TeakLite4
LTE DSP
TeakLite4
WCN
BTWF
Cortex-M33
GNSS
Cortex-M33

Common problems

- Clock and power domain management
  - hard to verify without SoC documentation
  - determining when they are safe to turn off requires testing
  - some drivers (e.g., IOMMU) keep clocks on all the time
  - common clock framework can keep power domains on
- Proprietary remoteproc firmware
  - undocumented protocols
  - buggy or Android-oriented behavior
  - reverse-engineering the binaries is sometimes required
  - open-source alternatives should be developed, but need a lot of work
- Upstreaming is slow
  - should have started earlier

Userspace

- Power management (autosleep)
  - mostly works well in Sailfish OS with mce
  - not really supported in other mobile Linux environments
  - important events must keep device awake until processed
  - processing often split across different services
  - userspace vs kernel-side autosleep
- Modem
  - uses AT commands with some quirks
  - basic oFono driver for Sailfish OS
  - ModemManager not quite working yet
- Camera
  - libcamera fork
  - will be upstreamed after kernel driver
  - most camera apps are still very basic
  - new abstraction library for cameras?

Demo

https://storage.abscue.de/private/zImage/jolla-c2-kernel.mp4

Results

- Mainlining a new SoC is possible!
  - not everything needs existing upstream support
  - more opportunities to create simple drivers where none exist yet
  - downstream was usable as reference
- Somewhat usable as daily driver
- Did not take much longer than a year
  - can be done faster with more time investment
- Sailfish OS can work without libhybris

Further opportunities

- Mainline more devices!
  - new Jolla Phone (2026) with MediaTek SoC
- upstream more changes
- fix issues with WiFi
  - work on new open-source firmware
- for Sailfish OS:
  - most userspace components already adapted
  - Jolla is generally open to contributions as long as they don't break things
  - reach feature parity
- for other distributions (e.g., postmarketOS):
  - implement ModemManager support
  - improve userspace power management

Thank you for listening!

Here are some links:

- Kernel fork: `https://codeberg.org/ums9230-mainline/linux`
- remoteproc firmware prototype: `https://codeberg.org/affenull2345/opencp`
- Sailfish OS port: `https://forum.sailfishos.org/t/mainline-linux-kernel-for-the-jolla-c2/21382/19`