
# The Seven Year Leap

## Updating a product from Linux 2.6 to 4.15, a real-world project case study

ELC-E October 2018, Edinburgh UK

Ed Langley, Kobilon
elangley@kobilon.com

# Not the first talk related to working with older kernels

- Stuck in 2009 — How I Survived
  Will Sheppard, Embedded Bits Limited
  ELCE 2016
  https://elinux.org/ELC_Europe_2016_Presentations

- How I survived to a SoC with a terrible Linux BSP
  Luca Ceresoli, AIM Sportline
  FOSDEM 2016, ELCE 2017
  https://elinux.org/ELC_Europe_2017_Presentations

- Long-Term Maintenance, or How to (Mis-)Manage Embedded Systems for 10+ Years
  Jan Lübbe, Pengutronix e.K.
  ELCE 2016
  https://elinux.org/ELC_Europe_2016_Presentations

# Where are you on the kernel development spectrum?

(Or, where is your project on it?)


1: kernel maintainer
2: semiconductor/IP vendor (Initial support may go into their own repository first)
3: Recently started product/device development
4: bigger Linux distributions, (Extending to right for LTS maintenance)
5: BSP or dev kit vendor for class of devices (E.G. set top boxes)
6: Ongoing product development/maintenance (Or new one with bad BSP)
7: Legacy / heavily regulated / safety critical device maintenance

# What was the device in this project?

- Single purpose handset running Android 4.4
- Boots and loads a dedicated app written by customer app team
- HW design quite old - based on TI DM3730 (OMAP3)
  - Similar design to the BeagleBoard xM
  - Display through OMAP DSS RFBI (MIPI-DBI) -> HX8369 -> ILI9806 TFT panel
  - Other off-chip peripherals: GSM modem, Bluetooth interface

# Older kernel problems resolved

- Initial Android support using a 2.6 TI kernel
  - That kernel pre-dates addition of certain required driver features
- Backported I2C touchscreen driver
- Then had to backport OMAP3 RTC alarm wake up from suspend
  - 12 cherry picks, plus fixup to allow backported init code to run through correctly
  - Not just a case of taking a self contained .c file from a later kernel

# Older kernel problems NOT resolved

## ...and becoming blockers

- Battery life too short
  - Current draw in suspend too high: 0.09mA, needed to be 0.03mA
  - Need the SYS_OFF_MODE support for TWL4030 PMIC, added after 2.6
    - Suspend/resume scripts loaded to PMIC, initialisation, OMAP HWMOD framework
    - Would have needed to take functionality and rewrite to fit older kernel

# Older kernel problems NOT resolved

- SD card eject and re-insertion detection
  - SD card only detected and read correctly if inserted at boot
  - Expected to be trivial, was not
  - More integration with OMAP HWMOD framework
  - Again, would need significant rewrite for the very different OMAP hwmod framework in older kernel

# Older kernel problems NOT resolved

- The big one: Suspend lock up
  - When system goes into suspend, intermittently does not resume, not responsive to power button
  - Debugging the issue:
    - Extra kernel printks, function tracing
    - Observing current draw measurement on power supply helped characterise the issue
  - Which established:
    - System will recover when serial port activity (Required opening handset case and soldering in wires to debug UART)
    - More likely to occur if no touchscreen input generated after reset
    - Very likely I2C related
  - Could have continued investigating (Had more ideas to try since)
    - But with the other issues to deal with in the old kernel, is it worth it?

# Making the case for the upgrade

- I proposed a new kernel may solve all 3 problems at once
- Q
  - "Will it fix everything?"
  - "How long will it take? / Cost?"
- A
  - Can't promise lower power consumption, or problems fixed
  - But could try it and see...
    - I.e. Propose least amount of effort needed to establish requirements are met / problems are fixed by a newer kernel version

# Initial plan / estimates

- To check if new kernel will resolve problems, only need to estimate work needed to check that
  - Get Linux booting to a shell prompt, kernel only needs:
    - UART driver
    - Existing SoC support for on-chip peripherals - to test problem areas
  - Use a plain rootfs (Not Android)
- In other projects, depending on what the problems are, getting to this point might not be so easy

# Finding the starting point

- This is probably the main area where YMMV

- For rootfs, just used buildroot
  - No BeagleBoard Yocto BSP
  - Eventual goal is Android anyway
- Newer toolchain
  - Ended up keeping older toolchain for AOSP userspace stuff
- TI DM37xx support all in mainline kernel by now
  - Should have started with that ….

# Getting to a bash prompt

```
U-Boot# bootz ${loadaddr} - ${fdtaddr}
Kernel image @ 0x80200000 [ 0x000000 - 0x50abb8 ]
## Flattened Device Tree blob at 80f80000
   Booting using the fdt blob at 0x80f80000
   Using Device Tree in place at 80f80000, end 80f9810c

Starting kernel ...

<nothing happens>
```

- First difference – where has default kernel config gone?
  - In 2.6, for a BeagleBoard-xM built kernel with omap3_beagle_defconfig – long since removed
- Had several false starts trying methods to build kernel/rootfs images for a BB-xM
  - Buildroot version with extra patches for BeagleBoard
  - Kernel on BeagleBoard github
  - These had custom kernel configs or menu defaults (bb.org_defconfig, omap2plus_defconfig)
    - Distracted from the correctly booting options available in mainline
- Eventual optimum setup which worked correctly:
  - Mainline kernel tree, latest stable release
  - Build with **multi_v7_defconfig**, then load **omap3-beagle-xm.dtb**

# Newer kernel, newer boot loader

- U-Boot support for loading a DTB
- Could have managed with older U-Boot, but path of least resistance was to upgrade U-Boot+kernel as a pair
- MLO in NAND will load U-Boot from SD if found
  - Needed to be able to load old U-Boot from newer MLO, for re-flashing old OS without opening case to press boot select button:

    Default: NAND → USB → UART3 → MMC1
    Boot select held: USB → UART3 → MMC1 → NAND

  - After consideration, this only matters when MLO in NAND loads U-Boot from SD
- Solution was to ensure
  - U-Boot file on SD always named u-boot.bin (Same as old version)
  - New U-Boot linked to run from 0x80008000 (Again, same as old version)

# New kernel running

- Stuff that just worked out of the box:
  - Serial port
  - Power button
  - Suspend/resume
    - Current draw in suspend now at 0.03mA
  - RTC alarm waking up from suspend
  - SD card insertion/removal detection and ability to mount

# Check the new kernel has fixed the blocking issues

- First step - ensure the point of upgrading has been achieved
- Only then move on to adding support for platform customisations/bespoke peripherals

# Check the new kernel has fixed the blocking issues

```
#!/bin/sh

LOOP_COUNT=0

echo "Suspend wake cycle test script"
echo ""

while [ true ]; do
        echo "suspending for 10 seconds"
        echo +10 > /sys/class/rtc/rtc0/wakealarm
        echo mem > /sys/power/state

        echo "woken by RTC, waiting 10 seconds"
        sleep 10

        echo ""
        echo "Iterations so far: $LOOP_COUNT"
        echo ""
        LOOP_COUNT=`expr $LOOP_COUNT + 1`
done
```

# Re-adding support for peripherals to new kernel

- First thing to remember when jumping to a present day kernel:
  - Look closely at what drivers/subsystems are about in the kernel source
  - Not everything is where you would expect it
    - Check the `drivers/staging/` directory
  - Check DTSs of similar platforms for examples

# Example of almost doing it wrong: Backlight

- In the old kernel, the platform used a copy of backported driver:

  `drivers/video/backlight/pandora_bl.c`

  - With PWM channel changed
- That driver still there in 4.x
  - Started adding an altered version again, then realised that driver is leftover legacy

# Example of almost doing it wrong: Backlight

```
backlight {
    compatible = "pwm-backlight";
    pwms = <&twl_pwm 1 12000000 0>;
    pwm-names = "backlight";
    Brightness-levels = <0 5 10 15 20 25 30 35 40 45 50
                         55 60 65 70 75 80 85 90 95 100>;
    default-brightness-level = <19>; /* => 90 */
    pinctrl-names = "default";
};
```

- drivers/video/backlight/pwm_bl.c
- drivers/pwm/pwm-twl.c

# Working with your DTS

- Just take a few moments to stop and.... learn device tree
  - As in learn how to mine through the bindings doc and identify the obscure attributes available in any given node used on your platform and so on
- https://events.static.linuxfound.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf

# Touchscreen

- Old kernel: `atmel_mxt_ts.c` backported
- New kernel: Atmel Maxtouch driver now in mainline
- Still needed to forward port a few alterations, E.g. for co-ordinate axis reversal
  - Initially tried adding `touchscreen-inverted-x/y` property to the DTS node to do this, but use of those properties are down to the driver
  - DTS bindings doc is your friend, but so is reading the source still

# Display: A bigger challenge

From drivers/video/fbdev/omap2/omapfb/dss/Kconfig:

```
config FB_OMAP2_DSS_RFBI
        bool "RFBI support"
        depends on BROKEN
        default n
        help
          MIPI DBI support (RFBI, Remote Framebuffer Interface, in Texas
          Instrument's terminology).
```

- Uh oh!
- First step: unflag as broken - got compile errors
  - Time to roll up sleeves do some proper dev work

# Re-enabled a driver... in the wrong subsystem?

- Correct long term approach would probably be to put RFBI support into:
  - `drivers/gpu/drm/omapdrm/dss/`
- For this port it was quicker to re-enable RFBI in existing location:
  - `drivers/video/fbdev/omap2/omapfb/dss/`
- Nagging awareness during this work that I was making updates in the wrong place

# Re-enabled a driver... in the wrong subsystem?

- Forward ported panel driver using HX8369+ILI9806 into:
  - `drivers/video/fbdev/omap2/omapfb/displays` (For now)
- HX83xx + ILI9xxx sounds like panel drivers which have gone into drivers/staging/fbtft/
  - Similar result – some of the chips variants share same interfaces (MIPI-DBI serial for example),
  - But driver plumbing in SW for TI MIPI-DBI implementation is very different to hanging a display off SPI

# RFBI being stubborn

- Driver running happily, but nothing on the screen. Checked:
  - Clocks (RFBI, DISPC, DSS)
  - Test frame output in driver, FB manual update test app
  - DISPC register values set correctly (After probe, after frame output)
  - IRQs (FRAMEDONE not triggering….)
  - Clocks again
  - Pinmuxing
  - Reset GPIO (Manually toggled, checked regs directly, check GPIO pinmux….)
  - Clocks again (Oh wait, PLL2 isn’t enabled!….Still doesn’t work though)
  - Reset GPIO again (Switched to using reset-gpios dts property, seemed to get that working)
  - Clocks again (Yes, really)
  - And then...

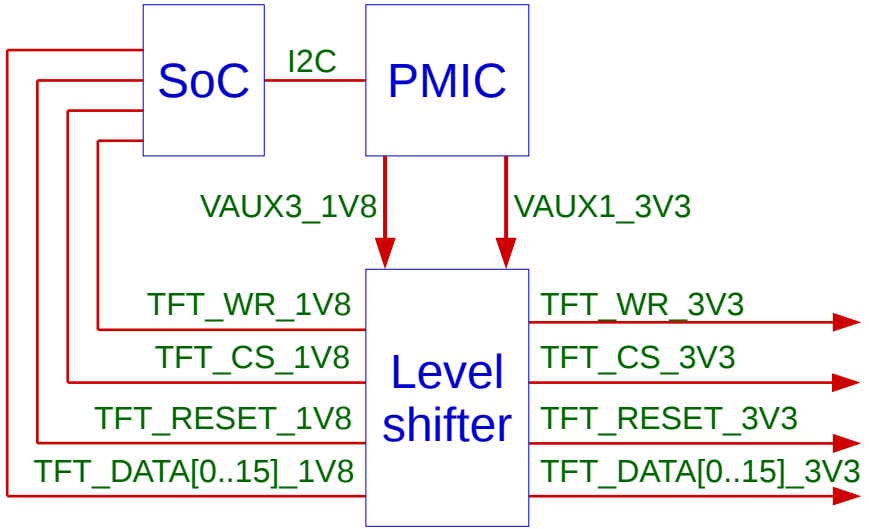
# Regulators

- Example of an item largely ignored by me in the previous kernel port

```
static struct regulator_consumer_supply beagle_vaux1_supply = {
        .supply         = "cam_3v0",
};

static struct regulator_init_data beagle_vaux1 = {
        .constraints = {
….
        },
        .num_consumer_supplies  = 1,
        .consumer_supplies      = &beagle_vaux1_supply,
};

static struct twl4030_platform_data beagle_twldata = {
….
        .vaux1          = &beagle_vaux1,
….
};
```



# Regulators

- Which must now be enabled in a very different way

DTS:

```
lcd0: display {
        compatible = "ilitek,ili9806";
        label = "lcd";
        vcc1-supply = <&vaux1>;
        vcc2-supply = <&vaux3>;
```

Driver:

```
struct panel_drv_data {
...
        struct regulator *vcc1;
};
```

In probe():

```
        ddata->vcc1 =
                devm_regulator_get(&pdev->dev, "vcc1");
```

In panel_enable():

```
        r = regulator_enable(ddata->vcc1);
```

# Putting Android back on again

- “Simply just replace the buildroot test rootfs with the Android filesystem images from before, right?”
- Oh yes and switch to SW rendering (Easier to ignore the GPU for now)

BoardConfig.mk:

```
USE_OPENGL_RENDERER := false
```

(Depending on Android version)

# Staying with Kitkat on 4.x kernel

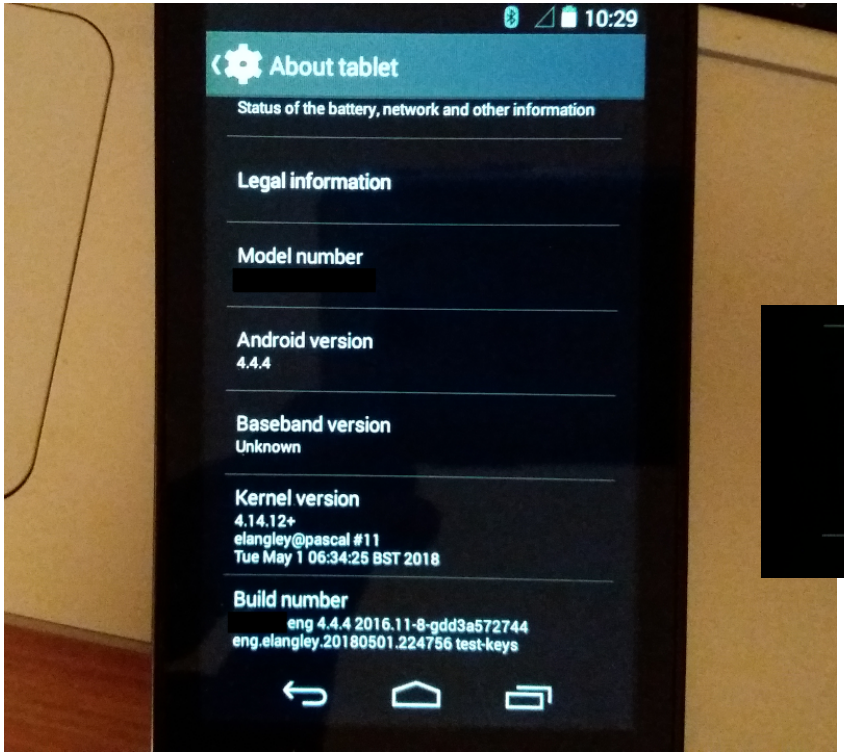
- Problems:
  - ashmem driver removed
  - Android logger driver removed
  - ADB not working

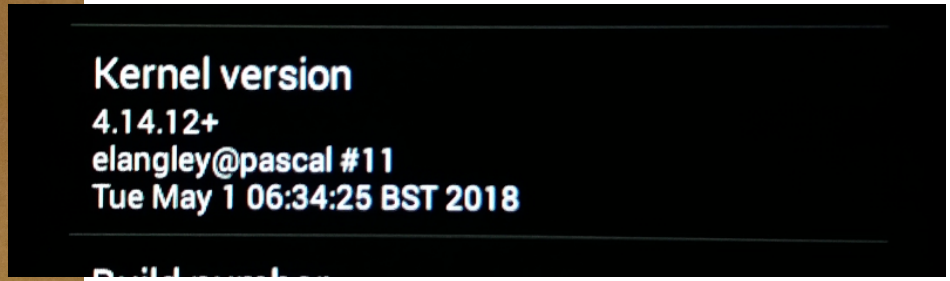
# ADB support

- Android composite USB driver had been taken out
- Soon found that last available version of Android composite USB gagdet driver is now very incompatible with latest kernel
- Hang on though, how does newer Android do it, after the driver was removed?
- With UBS gadget ConfigFS
  - adbd had support for that added in Android version... Kitkat (Phew)

- Re-affirmation of the lesson: Don't blindly forward port stuff, understand and leverage what is new

# First usable release

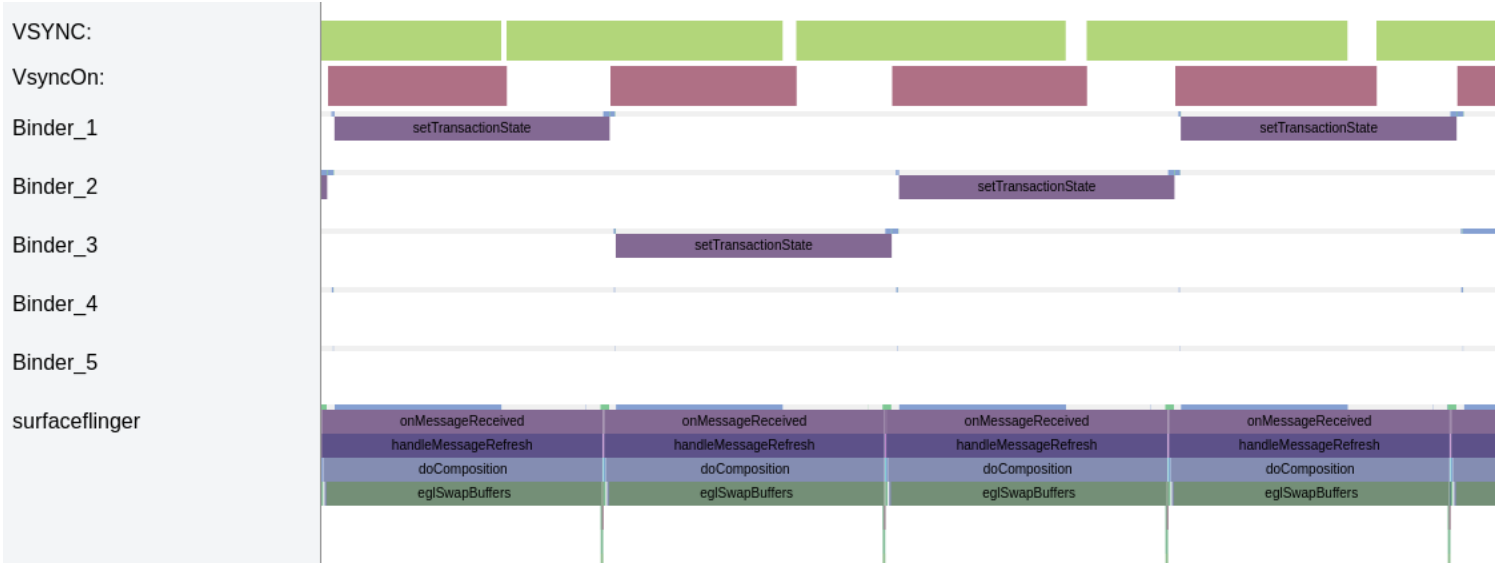



Yay

# Initial feedback from test team

- "Performance is slow"
- First suspicion was the software rendering I had enabled
  - Experience of this with the platform after working on the RFBI in 2.6



# Re-adding PowerVR SGX rendering support

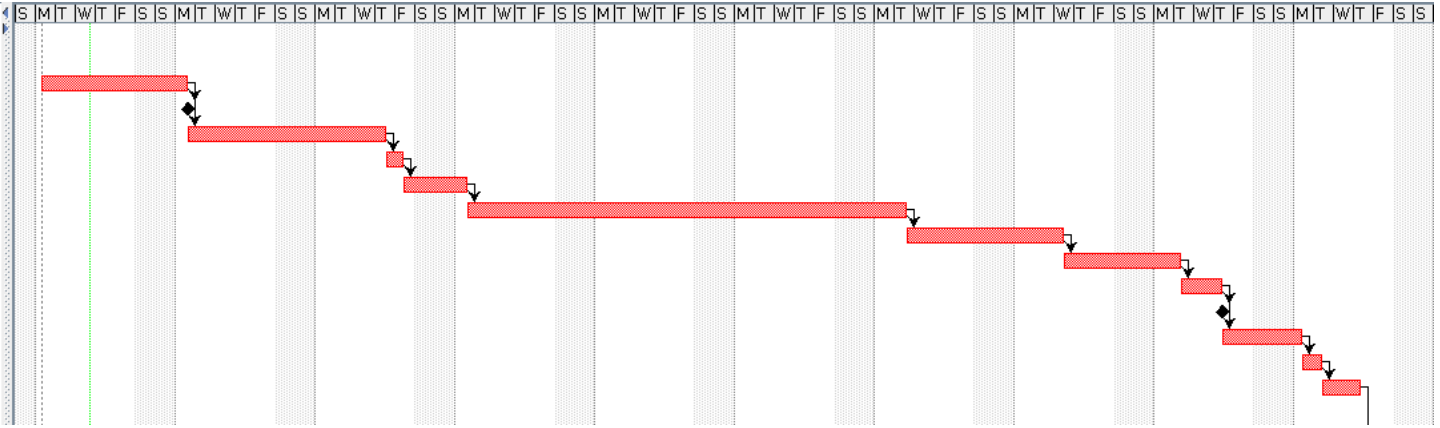
- Latest version of TI Graphics SDK (5.01) only supports kernel up to 3.8
  - Forward ported pvrsrvkm module to 4.14: Missing symbols on module load
  - Applied patches found online: Reset and clocking issues
  - Made further changes to enable clocks according to clock tree configuration on this platform
- That got 5.01 PVR modules + 5.01 regular SDK + 4.14 kernel working
  - Android version of TI Graphics SDK, with different EGL lib, only up to version 4.06
    - 5.01 PVR modules + 4.06 Android SDK: PVR userspace daemon version mismatch error (against kernel driver) at init
- Eventually found a working approach by forward porting the older TI Android graphics SDK 4.06 kernel modules
  - And applying my patches

# A whole bunch of other bits

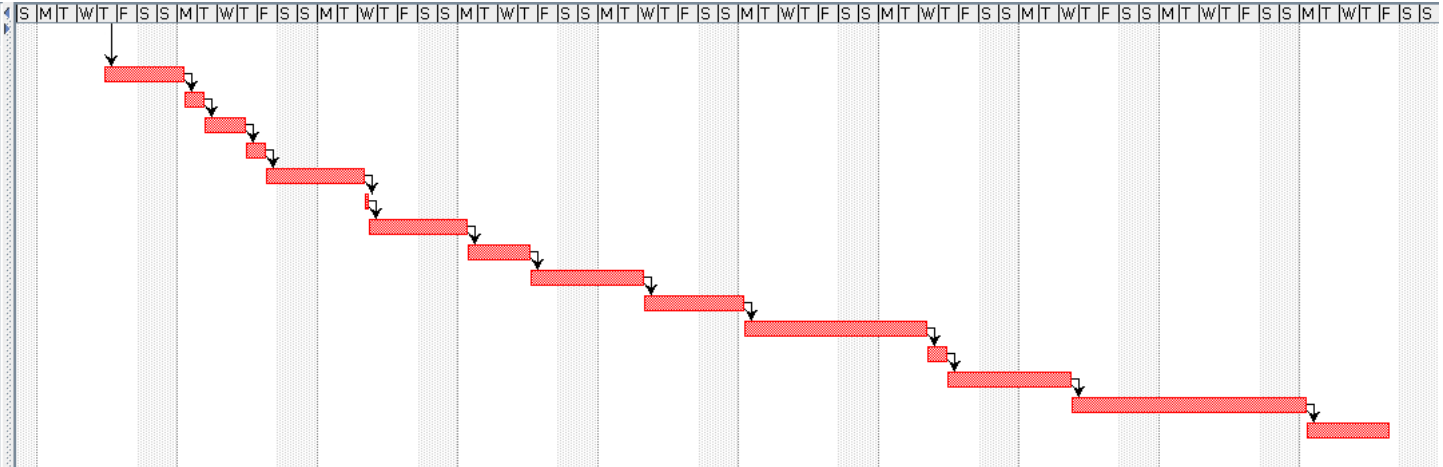
- Battery charging status / level reporting PMIC driver – update to be usable again
  - Add support for DTS properties
- Secondary battery enable and voltage reading PMIC driver
- Android OTA updater application mods for different UBI layout in NAND
- Arbitrary GPIO numbering due to enumerating each bank in random order at boot
  - Patched gpio-omap.c driver to set ordering in DTS
- ADB support take two - not working on Windows workstations (But was on Linux PC)
  - Serial number argument to gadget driver needed
  - First attempt used functionFS, Windows ADB only registers composite USB devices, made switch to configFS
  - Had to wade through Windows adb.exe source to figure those out
- GPU driver take two - VSYNC interrupt not firing

# How long did it take, in retrospect?

| Name | Work |
|---|---|
| Booting 4.x kernel, bootloader, rc | 47.5 hours |
| Pre Android eval release | 0 hours |
| NAND support, unlock, UBI rootfs | 62 hours |
| Backlight | 5 hours |
| Touchscreen | 12 hours |
| RFBI+TFT panel driver | 128 hours |
| Android boot | 45.5 hours |
| Touchscreen continued | 29 hours |
| Android ADB driver setup | 18 hours |
| Alpha-0.1 release | 0 hours |
| Battery charging interface | 15.5 hours |
| AOSP build changes | 8 hours |
| GSM modem power control driver | 14.5 hours |



| Name | Work |
|---|---|
| Clean up and config adjustments | 16 hours |
| Audio | 9 hours |
| Bootloader backwards compatibil | 17 hours |
| AOSP changes integration | 8 hours |
| GPIO numbering | 23 hours |
| Backup battery PMIC driver | 3 hours |
| Off-chip peripheral power up pinm | 23 hours |
| NAND Android boot, unlock | 26.5 hours |
| Android update application UBI la | 26 hours |
| Wakelock issue | 24 hours |
| GPU | 58 hours |
| ADB take two | 8 hours |
| PPPD config, RIL issues | 37 hours |
| GPU continued | 58 hours |
| GPU VSYNC interrupt issue | 35 hours |



# How long did it take, in retrospect?

| Activity | Hours | Days if 8 hours |
|---|---|---|
| Booting 4.x kernel, bootloader, rootfs on SD | 47.5 | 5.9375 |
| NAND support, unlock, UBI rootfs | 62 | 7.75 |
| Backlight | 5 | 0.625 |
| Touchscreen | 12 | 1.5 |
| RFBI+TFT panel driver | 128 | 16 |
| Android boot | 45.5 | 5.6875 |
| Touchscreen continued | 29 | 3.625 |
| Android ADB driver setup | 18 | 2.25 |
| Battery charging interface | 15.5 | 1.9375 |

# How long did it take, in retrospect?

| Activity | Hours | Days if 8 hours |
|---|---|---|
| AOSP build changes (kernel config, repo branches, toolchain, makefiles) | 8 | 1 |
| GSM modem power control driver | 14.5 | 1.8125 |
| Clean up and config adjustments | 16 | 2 |
| Audio | 9 | 1.125 |
| Bootloader backwards compatibility | 17 | 2.125 |
| AOSP changes (Init script changes etc) | 8 | 1 |
| GPIO numbering | 23 | 2.875 |
| Backup battery PMIC driver | 6 | 0.75 |
| Off-chip peripheral power up pinmuxing | 23 | 2.875 |

# How long did it take, in retrospect?

| Activity | Hours | Days if 8 hours |
|---|---|---|
| NAND Android boot, unlock | 26.5 | 3.3125 |
| Android update application UBI layout | 26 | 3.25 |
| Wakelock issue | 24 | 3 |
| GPU | 58 | 7.25 |
| ADB take two | 8 | 1 |
| PPPD config, RIL issues | 37 | 4.625 |
| GPU continued | 58 | 7.25 |
| GPU VSYNC interrupt issue | 35 | 4.375 |
| | | |
| Totals | 759.5 | 94.9375 |

# Points to remember when jumping major kernel versions

- Best starting point is to plan work needed to establish if key requirements of upgrade are met
  - Then decide to continue and plan / estimate further work after that
- Don't blindly forward port, inspect code carefully to assess:
  - What to port from old kernel
  - What to leave behind
  - What to extend in new kernel
- Check all your platform data!
  - Re-apply in DTS as appropriate
- Upstream useful changes during the project, not afterwards (Depending on certain factors)
- Every project/platform has different challenges
  - But universally something(s) always takes longer than planned



# Questions?

Get in touch: elangley@kobilon.com