Device Driver Gardening - transplant Linux drivers fast but gently

Stefan Kalkowski

<Brussels FOSDEM 2023>

Outline

1. Motivation
2. Linux kernel ports revisited
3. Short Demo
4. Results
5. Q & A

Why to re-use Linux drivers

- Increasing complexity of hardware

Why to re-use Linux drivers

- Increasing complexity of hardware
- Poor hardware documentation

Why to re-use Linux drivers

- Increasing complexity of hardware
- Poor hardware documentation
- Hardware bugs and necessary quirks

Why to re-use Linux drivers

- Increasing complexity of hardware
- Poor hardware documentation
- Hardware bugs and necessary quirks
- Linux is open & runs everywhere

Why to re-use Linux drivers

- Increasing complexity of hardware
- Poor hardware documentation
- Hardware bugs and necessary quirks
- Linux is open & runs everywhere

Simply an economic decision

Opposed approaches

Purely the driver code

- Emulate needed semantic only
- Less sharing of emulation code
- Low complex
- Lots of manual efforts
- Deep knowledge of driver needs

Opposed approaches

Purely the driver code

- Emulate needed semantic only
- Less sharing of emulation code
- Low complex
- Lots of manual efforts
- Deep knowledge of driver needs

Maximum re-usage

- Likewise original runtime
- Sharing of emulation code possible
- Bigger codebase
- Less manual efforts
- Deep knowledge of Linux internals

Opposed approaches

Purely the driver code

- Emulate needed semantic only
- Less sharing of emulation code
- Low complex
- Lots of manual efforts
- Deep knowledge of driver needs

Maximum re-usage

- Likewise original runtime
- Sharing of emulation code possible
- Bigger codebase
- Less manual efforts
- Deep knowledge of Linux internals

High efforts $\Rightarrow$ Tendency to keep code

Still too much effort

Display controller & connectors i.MX 8MQ

- HDMI for EVK board: **3 PM**

Still too much effort

Display controller & connectors i.MX 8MQ

- HDMI for EVK board: **3 PM**
- MIPI DSI with Touchscreen for EVK board: **3 PM**

Still too much effort

Display controller & connectors i.MX 8MQ

- HDMI for EVK board: **3 PM**
- MIPI DSI with Touchscreen for EVK board: **3 PM**
- MIPI DSI with eDP bridge and Panel for MNT Reform2: **wrong version**

Still too much effort

Display controller & connectors i.MX 8MQ

- HDMI for EVK board: **3 PM**
- MIPI DSI with Touchscreen for EVK board: **3 PM**
- MIPI DSI with eDP bridge and Panel for MNT Reform2: **wrong version**

Turning point $\Rightarrow$ Need for change

Goals for new approach

- Reduce manual work for driver-specific environment

Goals for new approach

- Reduce manual work for driver-specific environment
- Meet original semantic as close as possible

Goals for new approach

- Reduce manual work for driver-specific environment
- Meet original semantic as close as possible
- Simplify correlation to original driver

Goals for new approach

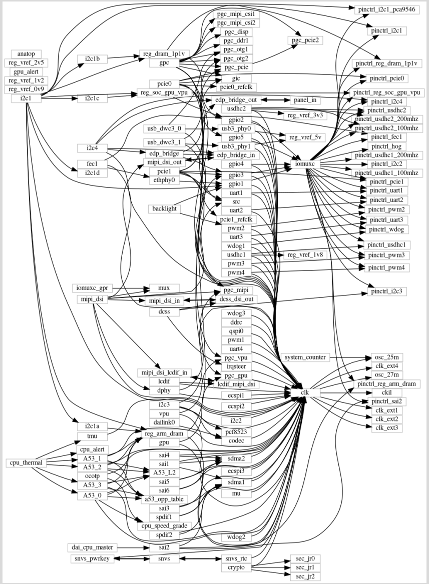
- Reduce manual work for driver-specific environment
- Meet original semantic as close as possible
- Simplify correlation to original driver
- Consolidate commonly used emulation parts

Outline

Minimal, executable Linux kernel

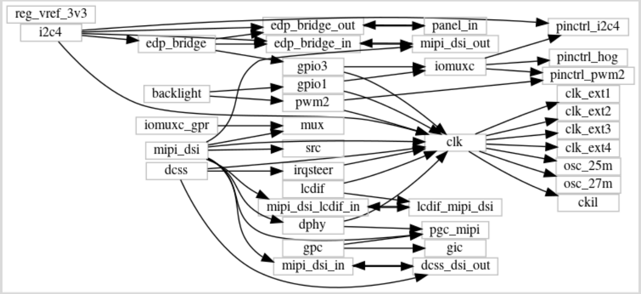
```
make tinyconfig

LX_ENABLE   = PCI PCI_MSI
LX_ENABLE  += WLAN CFG80211 MAC80211 RFKILL
LX_ENABLE  += WLAN_VENDOR_ATH ATH_COMMON ATH9K ATH9K_PCI ATH9K_DEBUGFS
LX_DISABLE  = CC_HAS_ASM_GOTO

scripts/config --file .config $(addprefix --enable ,$(LX_ENABLE))
scripts/config --file .config $(addprefix --disable ,$(LX_DISABLE))

make olddefconfig
```

Device Tree of MNT Reform 2



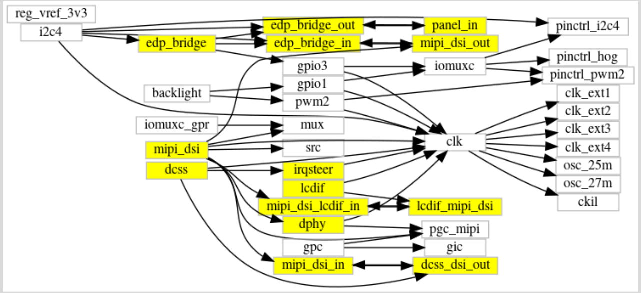
Minimal Device Tree

```
tool/dts/extract --select dcss --select edp_bridge --select lcdif
```



Minimal Device Tree

```
tool/dts/extract --select dcss --select edp_bridge --select lcdif
```



Initial set of compilation units

```
compatible = "fsl,imx8mq-lcdif", "fsl,imx28-lcdif";
compatible = "fsl,imx8m-irqsteer", "fsl,imx-irqsteer";
compatible = "fsl,imx8mq-nwl-dsi";
compatible = "fsl,imx8mq-mipi-dphy";
compatible = "nxp,imx8mq-dcss";
compatible = "ti,sn65dsi86";
compatible = "innolux,n125hce-gn1", "simple-panel";

grep -r "fsl,imx8mq-lcdif" drivers   # delivered no hit
grep -r "fsl,imx28-lcdif"  drivers   # then try the second one
```

Initial set of compilation units

```
drivers/gpu/drm/bridge/nwl-dsi.c
drivers/gpu/drm/bridge/ti-sn65dsi86.c
drivers/gpu/drm/imx/dcss/dcss-drv.c
drivers/gpu/drm/mxsfb/mxsfb_drv.c
drivers/gpu/drm/panel/panel-simple.c
drivers/irqchip/irq-imx-irqsteer.c
drivers/phy/freescale/phy-fsl-imx8-mipi-dphy.c
```

Linux headers included unmodified

- Include unmodified Linux kernel header

Linux headers included unmodified

- Include unmodified Linux kernel header
- No manual definition rewriting anymore

Linux headers included unmodified

- Include unmodified Linux kernel header
- No manual definition rewriting anymore
- Exception proves the rule, example initcalls:

```
#include_next <linux/init.h>
#include      <lx_emul/init.h>

#undef __define_initcall
#define __define_initcall ...
```

Find further compilation units

- Lots of undefined references!

Find further compilation units

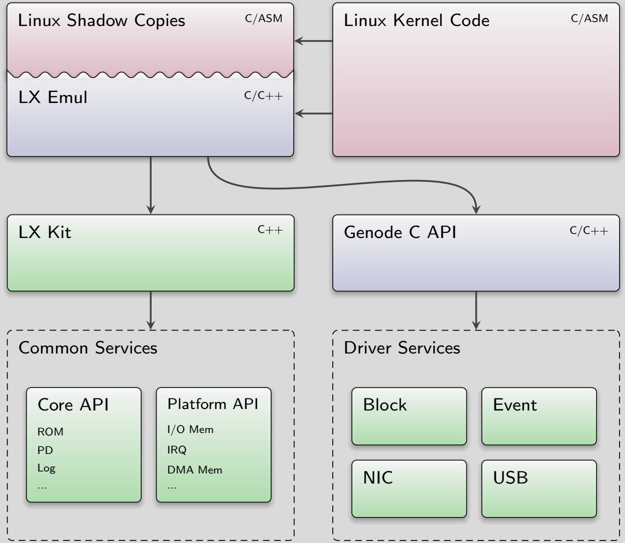
- Lots of undefined references!
- Tooling for identification & generation

Find further compilation units

- Lots of undefined references!
- Tooling for identification & generation

```
create_dummies <command> [VARIABLES]

--- available commands ---
show         - shows missing symbols of given TARGET
generate     - generates DUMMY_FILE for given TARGET
```

Strict API discipline



Outline

Driver ports within one year

PC universe

- USB *HCI Controllers
- Intel Display Engine
- Intel HD Audio
- Intel Touchpad
- WIFI (Intel, Realtek, Atheros)

+ Architecture independent port of WireGuard

ARM SoC landscape

- (e)MMC (Zynq 7000, A64, i.MX8)
- Ethernet (A64, i.MX 5/6/7/8)
- USB Host Controller (A64, i.MX8)
- Mali GPU (A64)
- Vivante GPU (i.MX8)
- Display Engine (A64, i.MX8)
- Camera over CSI (A64)

Outcome

- Initial driver port time: ~15%

Outcome

- Initial driver port time: ~15%
  - Tooling reduces manual work

Outcome

- Initial driver port time: ~15%
  - Tooling reduces manual work
  - Debugging aid: tinykernel correlation

Outcome

- Initial driver port time: ~15%
  - Tooling reduces manual work
  - Debugging aid: tinykernel correlation
  - Driver update resp. version change faster

Outcome

- Initial driver port time: ~15%
  - Tooling reduces manual work
  - Debugging aid: tinykernel correlation
  - Driver update resp. version change faster
- Drivers better meet all-purpose

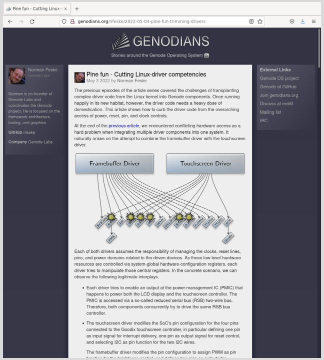
Outcome

- Initial driver port time: ~15%
  - Tooling reduces manual work
  - Debugging aid: tinykernel correlation
  - Driver update resp. version change faster
- Drivers better meet all-purpose
- Compiled codebase: ~200-300%

Outcome

- Initial driver port time: ~15%
  - Tooling reduces manual work
  - Debugging aid: tinykernel correlation
  - Driver update resp. version change faster
- Drivers better meet all-purpose
- Compiled codebase: ~200-300%
- Manually code to maintain: ~20%

Further reading



Pine fun - Cutting Linux-d

genodians.org/nfeske/2022-05-03-pine-fun-trimming-drivers

GENODIANS

Stories around the Genode Operating System

Norman Feske
Genode Labs

Norman is co-founder of Genode Labs and coordinates the Genode project. He is focused on the framework architecture, tooling, and graphics.

GitHub nfeske

Company Genode Labs

Pine fun - Cutting Linux-driver competencies
May 3 2022 by Norman Feske

The previous episodes of the article series covered the challenges of transplanting complex driver code from the Linux kernel into Genode components. Once running happily in its new habitat, however, the driver code needs a heavy dose of domestication. This article shows how to curb the driver code from the overarching access of power, reset, pin, and clock controls.

At the end of the previous article, we encountered conflicting hardware access as a hard problem when integrating multiple driver components into one system. It naturally arises on the attempt to combine the framebuffer driver with the touchscreen driver.

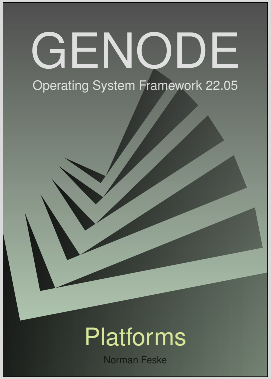
Each of both drivers assumes the responsibility of managing the clocks, reset lines, pins, and power domains related to the driven devices. As those low-level hardware resources are controlled via system-global hardware-configuration registers, each driver tries to manipulate those central registers. In the concrete scenario, we can observe the following legitimate interplays.

- Each driver tries to enable an output at the power-management IC (PMIC) that happens to power both the LCD display and the touchscreen controller. The PMIC is accessed via a so-called reduced serial bus (RSB) two-wire bus. Therefore, both components concurrently try to drive the same RSB bus controller.
- The touchscreen driver modifies the SoC's pin configuration for the four pins connected to the Goodix touchscreen controller, in particular defining one pin as input signal for interrupt delivery, one pin as output signal for reset control, and selecting I2C as pin function for the two I2C wires.

  The framebuffer driver modifies the pin configuration to assign PWM as pin

External Links
Genode OS project
Genode at GitHub
Join genodians.org
Discuss at reddit
Mailing list
IRC

Genodians.org



GENODE
Operating System Framework 22.05
Platforms
Norman Feske

Genode Platforms 22.05