ESC-400: Rationalizing the Platform Perimeter

Linus Walleij
ST-Ericsson
Linaro

The Platform Perimeter

As a consumer of a System on Chip (SoC) when constructing diverse embedded systems you seldom see what is happening to the community-maintained Linux kernel from your perspective. What is the community doing that I should be aware of? What is happening out there that will help me or could ruin my business or make me look bad if not taken into account.

For this session I assume that the embedded engineer will be interested in getting the grips of the *platform perimeter*. I define this as the sum total of the electric border between your core SoC system (or a large pre-packaged chipset) and the peripherals found in a certain design, i.e. a PCB or similar arrangement.

We will not go into details on what is on either side of the perimeter boundary, i.e. not discuss about DMA in SoC:s on one hand or the best way to implement a LED or gyroscope driver on the other hand – instead we will discuss the perimeter as such.

The scope will be what has happened "recently" in the Linux kernel, with some rather personal selection of important points regarding a few related subsystems.

An illustration can be found in figure 1.

Push for Device Tree

All SoC vendors active in the Linux kernel are currently aware about a major push from the ARM SoC architecture maintainers to migrate their systems to use the Device Tree infrastructure. If you have previously used SPARC or Power PC this concept will be familiar as it is a spin-off from OpenFirmware.

The Device Tree provides an hierarchical representation of the devices in the system and a database to query for any configuration including IRQ lines, GPIO pins, register memory ranges, voltage regulators and any other so-called "platform data" the device may need.

This has come about for practical reasons as the ARM portion of the kernel tree was growing wildly. It can be seen as a way to achive with external means what things like Plug-n-Play BIOS, ACPI DSDT, or the PCI configuration space is providing in other contexts – a means to ask a central registry about the hardware attached to a system. When such a feature has not been engineered into the hardware of the system, we can still provide it in the form of a Device Tree.

For system designers, users of SoC:s, the Device Tree is yet to deliver its promised benefits, but the idea is certainly to describe any hardware attached around the platform perimeter using them.

Thomas Abraham has made an excellent presentation about how to go about enabling Device Tree for a certain system which can be found in the references section. As a consumer of SoC:s, you should normally request your vendor to provide a basic Device Tree implementation for your system.

The implementation of Device Tree provides a *de facto* standardization of the so-called *bindings* that define the structure of any configuration data for a certain device. These bindings are intended to be neutral in character and useful also for other operating systems than Linux.

Device Model

The devices instatiated by the Device Tree or by the more traditional board files should nowadays be all dynamic, not the result of a `static struct platform_device` but rather a `kmalloc()`:ed structure in memory.

Devices should use the `devm_kmalloc()` and similar facilities to let the device core free up memory as drivers are removed from the system for example.

Runtime Power Management or "runtime PM" has been solidifying the last year but may still be lacking in certain spots. Vendors are now migrating custom solutions to hammering off clocks and regulators to the runtime PM framework.[1]

Devices can nowadays be collected into power domains which are used by the runtime power management core to reference count the users of a certain domain and makes it possible to shut entire regions of a system down whenever the domain is unused. This is primarily intended for silicon complexes but can be used for any devices powered off the same power terminal.

Peripheral Buses

Among the peripheral buses we find I^2C and SPI have been pretty silent and mature with only casual refactoring taking place.

The Multifunction Device (MFD) subsystem has been well established as a device nexus responsible for spawning child devices and arbiting and marshalling calls when devices on peripheral buses contain several functionality blocks, each to be handled by a separate subsystem in Linux. The archaic example of an MFD device is a mixed-signal circuit exposing its software interface as a heterogeneous I^2C register range.

A significant change that can improve the use of devices on slow peripheral buses devices is *regmap* in `drivers/base/regmap` that comes from the ALSA SoC part of the kernel.

The regmap provides a unified marshalling and caching mechanism to get away from all custom code providing such facilities for example if the target register is 16 or 32 bit wide yet marshalled over an 8-bit I^2C bus. It can be used for any devices on slow links, including Multifunction Devices (MFDs) that present themselves as remote, large register ranges.

A new peripheral bus named High-Speed Synchronous Serial Interface (HSI) has appeared and will likely soon make its way into the kernel. A patch set has been devised by Carlos Chinea from Nokia and was proposed for inclusion into Linux v3.3, but not pulled in. It will likely be accepted in coming kernel releases.[2]

A new peripheral bus named SLIMbus has also appeared, and in august of 2011 Sagar Dharia from CodeAurora presented a new

[1] See e.g. `http://www.elinux.org/images/1/18/Elc2011_damm.pdf` for an introduction to runtime PM.

[2] See: `https://lkml.org/lkml/2011/6/10/280`

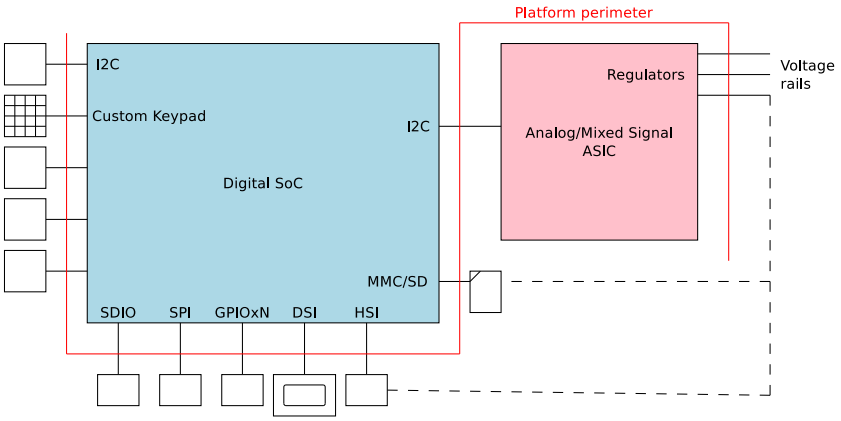

Fig. 1. The Platform Perimeter

subsystem in `drivers/slimbus` for this bus. However nothing has been heard from it since.[3]

GPIO and Pin Control

GPIO and pin control have been pretty busy lately. We have moved many of the ARM GPIO drivers out of their hiding in the `arch/arm` hierarchy and would like to refactor GPIO to use more exclusively the `struct gpio_chip` facilities in the kernel.

The Pin Control subsystem has been created to handle aspects of pins that cannot fit well with either GPIO or `struct irq_chip`.

For example the Pin Control subsystem is intended to standardize first and foremost *pin multiplexing* as is common practice in the resource-constrained world of SoC packaging solutions. However it will also centralize and define how to handle the knobs available in certain SoC pin arrays to bias pins, select drive strength, set schmitt-triggering, configure slew rate or other electrical characteristics not really related to any existing subsystem.

Regulators

The regulator subsystem is pretty mature and has recently been augmented with Device Tree bindings.

It has been established that the regulator subsystem wants us to write drivers so that any power supply lines into the component are requested at proper places in the driver. The basic rule is that if a power pin happen to be wired to an appropriate voltage in your system, assume that the next person using the same component will connect it to a software-controlled regulator, so have your driver issue `regulator_get(dev, ''foo'')` for any voltage terminal on your packe, and atleast provide a fixed regulator for that voltage in your board file or device tree.

MMC/SD/SDIO

The MMC, eMMC, SD card and SDIO subsystem in `drivers/mmc` has seen a lot of stepwise improvement recently. A special point of interest to the platform perimeter are the SDIO portions of the subsystem.

The protocol spec deviation fixes, pure bugs, and gradual improvement that have aggregated in this subsystem over the last year is significant, to the point of vendors back-porting the entire MMC subsystem onto older kernels to get proper support for these devices.

Several of the fixes are performance-related, such as Per Förlin's back-to-back pre- and post- hooks to speed up cache mapping/unmapping of buffers during intense transfers, or Stefan Nilsson's fixes for pushing SDIO transfers into larger block chunks. The former give some improvements for MMC/SD cards whereas the latter will affect large-volume SDIO transfers such as for WLAN devices on SDIO.

On a longer term is work related to aligning the block layer or file systems in Linux to the characteristics of contemporary MMC or SD cards (but this does not really concern the platform perimeter).

Input and Extcon

The input subsystem is mature and has gained a lot of drivers the last year. The main notable trend is that touchscreen vendors (or their customers as proxies) have started to submit drivers for embedded system incarnations of touchscreens following the rise of Android.

Another effect from Android is the creation of the Extcon subsystem by Samsung engineer MyongJoo Ham. This subsystem is still in the works but will handle connection of external connectors such as USB cables (primary example "chinese chargers"), audio/video plugs, portable handsfree and HDMI connectors. These have previously been attempted by the input subsystem but are now forming their own subsystem.[4]

This initiative is a reflection of a larger trend of bringing good ideas from Android into the mainline Linux kernel as vendors struggle not to have to support multiple codebases.

References

Thomas Abrahams excellent presentation of how to enable Device Tree on an ARM board, as well as my own presentation of the pin control subsystem from the Embedded Linux Conference 2012 can be found here:

`http://www.elinux.org/ELC_2012_Presentations`

[3] See: `http://lwn.net/Articles/454945/`

[4] See: `https://lwn.net/Articles/483965/`