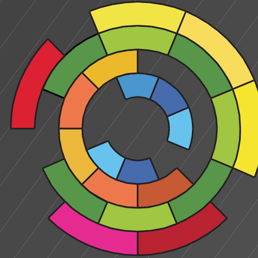
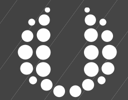


designwest

center of the engineering universe

ESC 2012: Rationalizing The Platform Perimeter

Linus Walleij



esc android NEW blackhat NEW designmed LEDs multicore sensors NEW

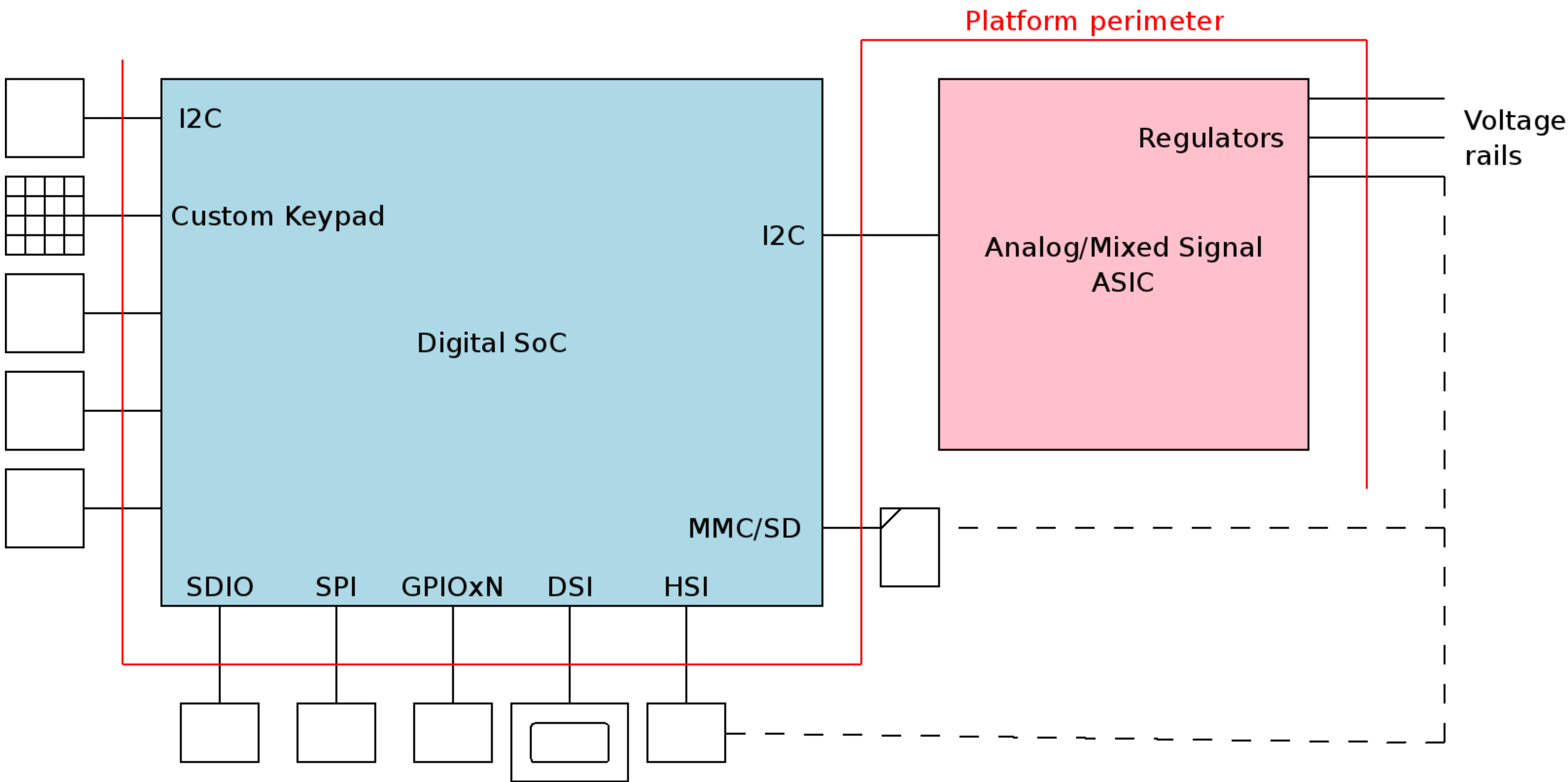
Background

Worked for some years with people trying to make the best use of the Linux kernel

- Noticed a lot of discrepancies between model and reality, long lead time from initiatives in the community to trickle down to embedded vendors
- Created the pin control subsystem in the Linux kernel to complement GPIO
- So I want to take this opportunity to try to present the state of relevant subsystems in the Linux kernel when working with the platform perimeter – what has been done so far and what is coming next
- The kernel changes relevant to *users* of SoC:s as opposed to the *producers* manufacturing the silicon
- See it as a ”perimeter report” as compared to Jon Corbets ”kernel reports” given at regular intervals about the state of the kernel at large

Recap: Linux device model

- Devices in Linux are defined by struct device, which may be subclassed by being a .dev entry of a sub-class device such as struct i2c_device
- Devices are usually registered to a bus
- Push to get rid of “class” and “type” device containers
- Introduction of reference-counting “power domain” concept in drivers/base/power/domain.c
- Push to stop using any statically declared struct device or derivates (struct platform_device, struct amba_device)
- Push to use Device Tree for registering all devices rather than board files – especially for the overpopulated ARM arch tree
- Push to use devm_* prefixed allocators etc to propely reference count and free especially memory allocated by device drivers
- Push to try to solve probe order problems currently often solved by pushing devices to different initlevels by using deferred probe which is basically a “probe me again, later” mechanism that will iterate the probe until dependencies are met


Platform perimeter
I2C
Custom Keypad
I2C
Digital SoC
MMC/SD
SDIO
SPI
GPIOxN
DSI
HSI
Regulators
Analog/Mixed Signal
ASIC
Voltage
rails

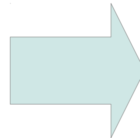
Device Tree Push (part 1)

- When the arch/arm/* tree was small, there were a few machines supported, mainly the RISC PC and then the StrongARM machines
- Then a lot of machines appeared and code was grouped into mach-foo/* directories with plat-foo/* directories to consolidate machine families
- Then things got out of hand:
  - No proper review of code going to machines
  - No active push to consolidate and refactor code across machines
  - All were "necessarily different" and thinking in their own silo
  - Merge conflicts and churn in the tree upsets Torvalds
- Quick fix: strip down defconfigs to the bare minimum deleting 194000 LOC
- ARM subarchitecture maintainers get together and discuss the problem
- One part of the problem is the "board files" – mainly static defines of the platform devices and their configuration

Device Tree Push (part 2)

- Device Tree presented as part of the solution – if we can get the data out of the kernel, the complexity can be reduced and/or externalized
- This should be easy – because experience shows it was easy for Power PC
- Strategy:
  - New platforms: only allow Device Tree probing
  - Legacy platforms: create new board files with the suffix -dt, move everything over then eventually delete the old board files
- Not as easy as it seems – Power PC was taken as model, but the PPC business was pretty closed and the producers had control over the software implementation process for their devices
- ARM was more chaotic and less keen on software standardization, also lack of volunteers and assignees to do the transition hampered progress
- So developers are still struggling with this
- However now it is happening due to the ARM SoC tree gatekeepers strong push and the backing of Linaro behind the Device Tree push
- Rely on Device Tree on a per-platform basis, not “one size fits all” for now

```
/* mach-foo/board-foo.c */

unsigned int mmc_status(struct device *dev)
{
        return !!readl(FOO_MMC_DETECT_REGISTER);
}

static struct mmci_platform_data mmc_plat_data = {
        .ocr_mask       = MMC_VDD_32_33|MMC_VDD_33_34,
        .status         = mmc_status,
        .gpio_wp        = -1,
        .gpio_cd        = -1,
};

static struct amba_device uart_device = {
        .res = {
                .start = 0x1000,
                .end   = 0x1000 + SZ_4K - 1,
                .flags = IORESOURCE_MEM,
        },
        .irq = { 1 },

};

static struct amba_device mmc_device = {
        .dev = {
                .platform_data = &mmc_plat_data,
        },
        .res = {
                .start = 0x2000,
                .end   = 0x2000 + SZ_4K - 1,
                .flags = IORESOURCE_MEM,
        },
        .irq = { 2, 3 },

};

static struct amba_device *amba_devs[] __initdata = {
        &uart_device,
        &mmc_device,
};

static void __init foo_init(void)
{
        int i;

        for (i = 0; i < ARRAY_SIZE(amba_devs); i++) {
                struct amba_device *d = amba_devs[i];
                amba_device_register(d, &iomem_resource);
        }
}

MACHINE_START(FOO, "Foo Machine")
        ...
        .init_machine   = foo_init,
MACHINE_END
```



```
/* mach-foo/board-foo-dt.c */

unsigned int mmc_status(struct device *dev)
{
        return !!readl(FOO_MMC_DETECT_REGISTER);
}

static struct mmci_platform_data mmc_plat_data = {
        .ocr_mask       = MMC_VDD_32_33|MMC_VDD_33_34,
        .status         = mmc_status,
        .gpio_wp        = -1,
        .gpio_cd        = -1,
};

struct of_dev_auxdata foo_auxdata_lookup[] __initdata = {
        OF_DEV_AUXDATA("arm,primecell", 0x2000, "mmci",
                 &mmc_plat_data),
};

static void __init foo_dt_init(void)
{
        of_platform_populate(NULL, of_default_bus_match_table,
                             foo_auxdata_lookup, NULL);
}

static const char *foo_dt_match[] __initconst = {
        "arm,foo",
        NULL,
};

DT_MACHINE_START(FOO, "Foo Machine")
        ...
        .init_machine   = foo_dt_init,
        .dt_compat      = foo_dt_match,
MACHINE_END
```

The below is supplied in binary compiled form to the kernel from the boot loader or attached to the kernel image:

```
/include/ "foo.dts"

/ {
        model = "Foo";
        compatible = "arm,foo";

        amba {
                uart@1000 {
                        compatible = "arm,primecell";
                        reg = <0x1000 0x1000>;
                        interrupts = <1>;
                };

                mmc@2000 {
                        compatible = "arm,primecell";
                        reg = <0x2000 0x1000>;
                        interrupts = <2, 3>;
                };
        };
};
```

Device Tree Push (part 3)

- SoC vendor should provide the basic SoC Device Tree in a file named "foo-soc.dtsi, then board files are created named "board-foo.dts"
- Example:
  - arch/arm/boot/dts/tegra20.dtsi – defines a SoC
  - arch/arm/boot/dts/tegra-ventana.dts – defines a specific board
- Through the .dts file you will one day be able to configure the entire platform perimeter – if everything goes well
- For example the SoC .dtsi file defines all the I2C buses, whereas your foo-board.dts file define all the devices that sit on the I2C bus
- You will still have to write drivers for all devices and compile them into your kernel or as modules ...
- Device Tree does not remove hard work, all it does is help a little bit with structuring the board files and keeping their configuration outside of the Linux kernel

I2C [drivers/i2c/*]

- Very mature subsystem, mainly seeing maintenance of bus drivers
- Regmap support in drivers/base/regmap/regmap-i2c.c (more on regmap soon!)
- Runtime PM
- All bus drivers needs to be augmented for Device Tree support
- Chip drivers removed from the subsystem and into respective driver subsystem – only core I2C business live here now (completed by Wolfram Sang in 2010)

SPI [drivers/spi/*]

- Quite mature subsystem, missing common infrastructure
- Proposed patch to create a central message queue mechanism
- Regmap support in drivers/base/regmap/regmap-spi.c
  (more on regmap soon!)
- Runtime PM
- All bus drivers needs to be augmented for Device Tree support

MFD [drivers/mfd/*]

- Multifunction Devices loosely defined as central child device spawning and arbitration hub
- Natural nexus for Mixed Signal circuits such as PMICs exposing various analog electronic controls as an automaton
- Many of these devices are I2C or SPI devices or both
- The Mixed Signal circuits then spawn devices in regulator, ALSA SoC, LED, backlight, PWM, GPIO ...

Regmap [drivers/base/regmap/*]

- ALSA System-on-Chip (ASOC) engineers noted that their register access and caching mechanism was generally useful
- In theory suitable for any register range not memory-mapped but accessed by other means, primarily I2C and SPI
- Handles marshalling register accesses to say 16 or 32 bit registers using consecutive 8bit writes/reads on an I2C bus
- Handles registering expected default values to a large register map at boot so these do not need to be read up from hardware at all
- Handles caching of registers declared non-volatile
- Writes write through and updates the cache
- Reads-through on volatile registers
- Cache is stored in a rbtree
- End result is a speed boost on anything register-based that is handled over a slow peripheral bus that can benefit from caching and simplification and centralization of code for marshalling register access

GPIO [drivers/gpio/*]

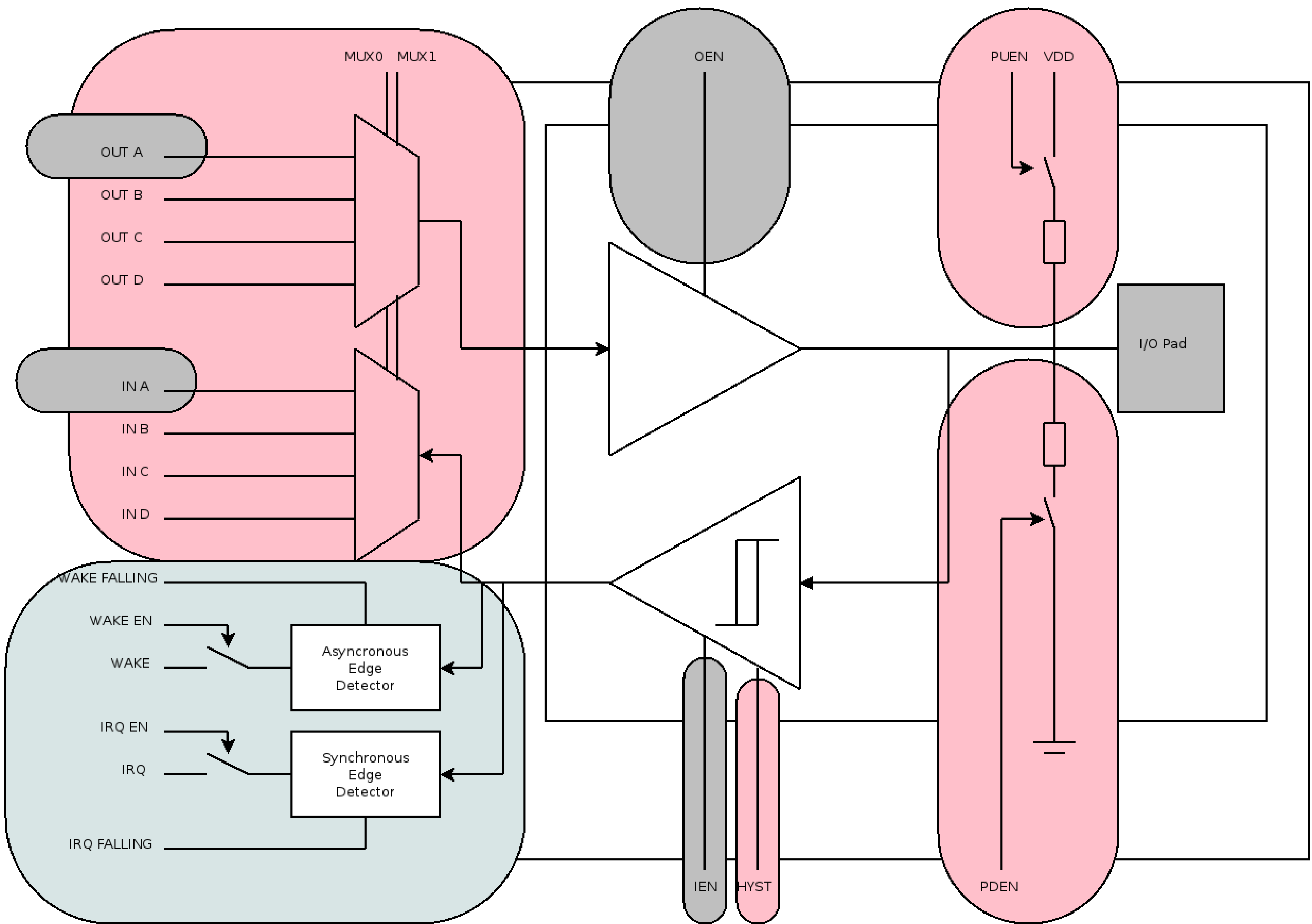
- Big push to move any GPIO drivers out of the arch/arm/* hierarchy and down to the GPIO subsystem, some may remain
- We want to remove generic GPIO once designed for performance bottlenecks and have all drivers use gpiolib
- ARM users split in simple cases (just gpiolib) and "__ARM_GPIOLIB_COMPLEX" which provide custom access macros for example to be used early. Very limited number of users.
- One goal is to get rid of <mach/gpio.h> to compile a multi-platform kernel
- Generic Device Tree bindings in place for GPIO keys, leds
- Everything else will need Device Tree bindings

Things TODO on a longer term:

- We want to get rid of the global GPIO numberspace
- We want to rewrite the questionable userspace interface in sysfs and replace with /dev/gpio0 etc for the gpio_chip:s

Pin Control [drivers/pinctrl/*]

- New subsystem for handling pin multiplexing, pin configuration (push, biasing, drive mode, schmitt-trigger etc) and anything else that is not already handled by GPIOlib or struct irq_chip
- Has caused endless pain for system designers using off-the-shelf SoC:s over the years
- I just had enough of custom pin configuration and created a new subsystem for pin control
- Illustration on following page on what is pin control and what is GPIO or irq_chip for a certain IO Pad
- Migrated a number of platforms to use this instead of custom pin control implementations
- Device driver interface similar to regulators or clocks, drivers can get handles to pins (both individual pins and groups of pins) and put these into different states at runtime
- Working on generic pin states and Device Tree bindings as we speak



MUX0
MUX1
OEN
PUEN
VDD
OUT A
OUT B
OUT C
OUT D
IN A
IN B
IN C
IN D
I/O Pad
WAKE FALLING
WAKE EN
WAKE
Asyncronous
Edge
Detector
IRQ EN
IRQ
Synchronous
Edge
Detector
IRQ FALLING
IEN
HYST
PDEN

HSI [will be drivers/hsi/*]

- Reads out High Speed Synchronous Serial Interface
- New subsystem in the works
- Used for connecting especially high-speed modems to SoC:s
- Nokias Carlos Chinea has created a subsystem
- Proposed for inclusion into Linux v3.3 but was not pulled in

Regulators [drivers/regulator/*]

- Well-established and mature subsystem
- Device drivers increasingly using the regulators to get their voltages
- Deep integration into the MMC/SD subsystem
- Preferred *design pattern* for device drivers: if your device has any kind of voltage supplies, retrieve them with regulator_get(dev, ”FOO”); and define a simple fixed-voltage regulator even if they happen to be wired to correct always-available supplies on your system. Further down the road someone will inevitably connect the same input to a software-controlled regulator
- Finalized Device Tree bindings for core regulator descriptors

MMC/SD/SDIO [drivers/mmc/*]

- Well-established and maturing as we speak
- Several large patchsets for supporting new eMMC and SD specs progressing in an incremental manner – if you need any one particular feature you need to worry else business as usual
- Numerous SDIO fixes the last kernel releases to get e.g. WLAN adapters attached to SDIO to work properly with odd packet sizes
- Several host drivers may still have severe SDIO problems – beware!
- All host drivers need to be augmented with Device Tree bindings, but that's not a perimeter problem!

Input subsystem [drivers/input/*]

- Well-established and mature subsystem
- External connectors moving out of the input subsystem and into the Android-derived "extcon" subsystem in drivers/extcon/* initiative driven by MyungJoo Ham from Samsung
- Extcon will be used for plug-in events of USB cables (i.e. for classless USB charging "chinese charger", audio-video-HDMI jacks
- Major embedded touchscreen vendors especially Synaptics increasingly dedicated to supporting Linux, also in the mainline kernel (maturation)
- GPIO keypads have Device Tree bindings, everything else especially anything custom needs new bindings

On the other side of the wire

- Industrial I/O (IIO) subsystem is gaining traction in staging, need to move into the drivers/* proper – especially crucial feature: timestamped measurements such as needed for industrial control but also for say augmented reality
- All drivers everywhere need to be augmented with Device Tree bindings

THANK YOU