Order at Last

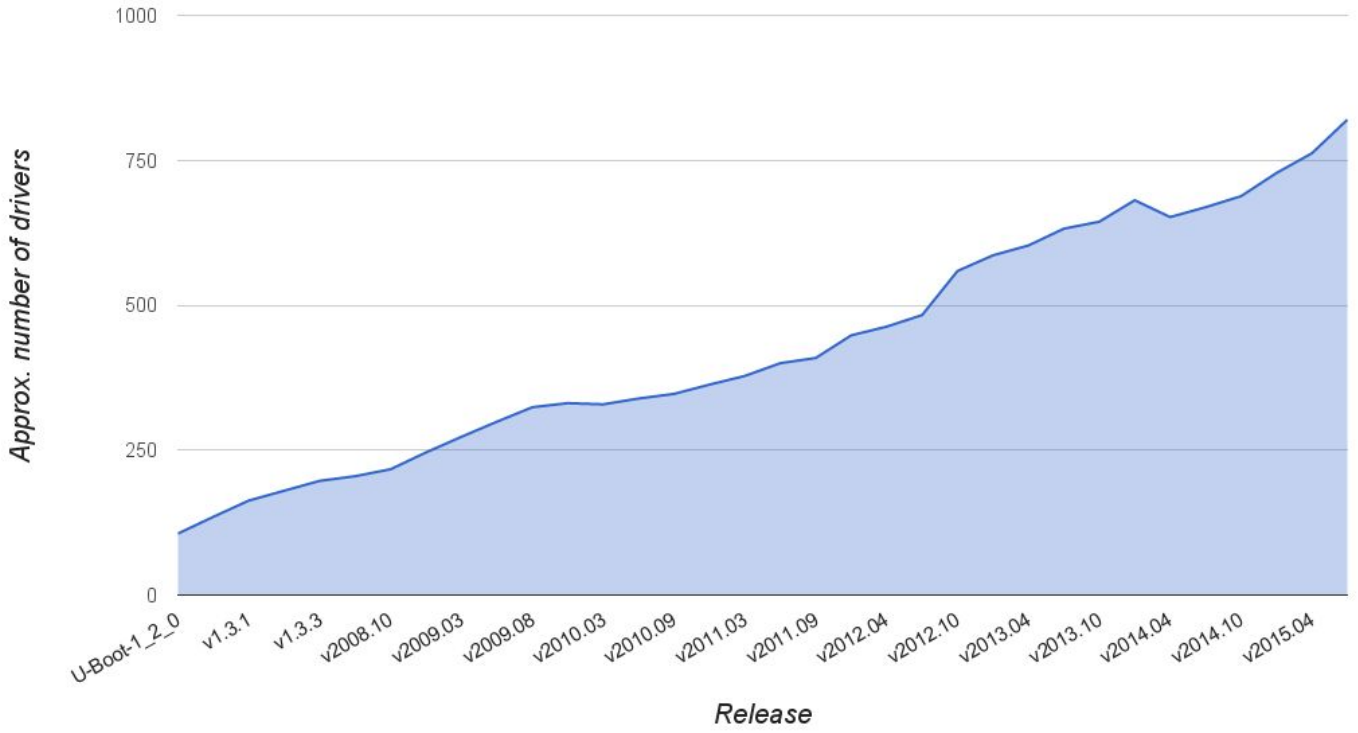
The New U-Boot Driver Model Architecture

Simon Glass, Google Inc, ELCE 2015, Dublin

Agenda

- Before driver model
- Design goals
- Architecture
- Benefits and limitations
- Test methodology
- Comparisons with Linux
- Performance
- A few examples
- Status update

More and more drivers



Before driver model

- U-Boot has 10 useful design principles (e.g. small, fast, simple, portable)
  - Huge community, over 1000 boards supported by the end of 2011
  - But Ad-hoc driver model started to bite
- Drivers were invoked through direct C calls
  - i2c_read() is implemented by whichever driver is compiled in
  - CONFIG option select which I2C driver to use, clock speed, bus number, etc.
- Hard to scale
  - Multiple I2C drivers must be munged into a single driver
  - Or an ad-hoc framework created to handle this requirements
- Configuration becoming unwieldy
  - 6000 CONFIG options at its peak
  - Kconfig conversion helps, but that's still a lot of options

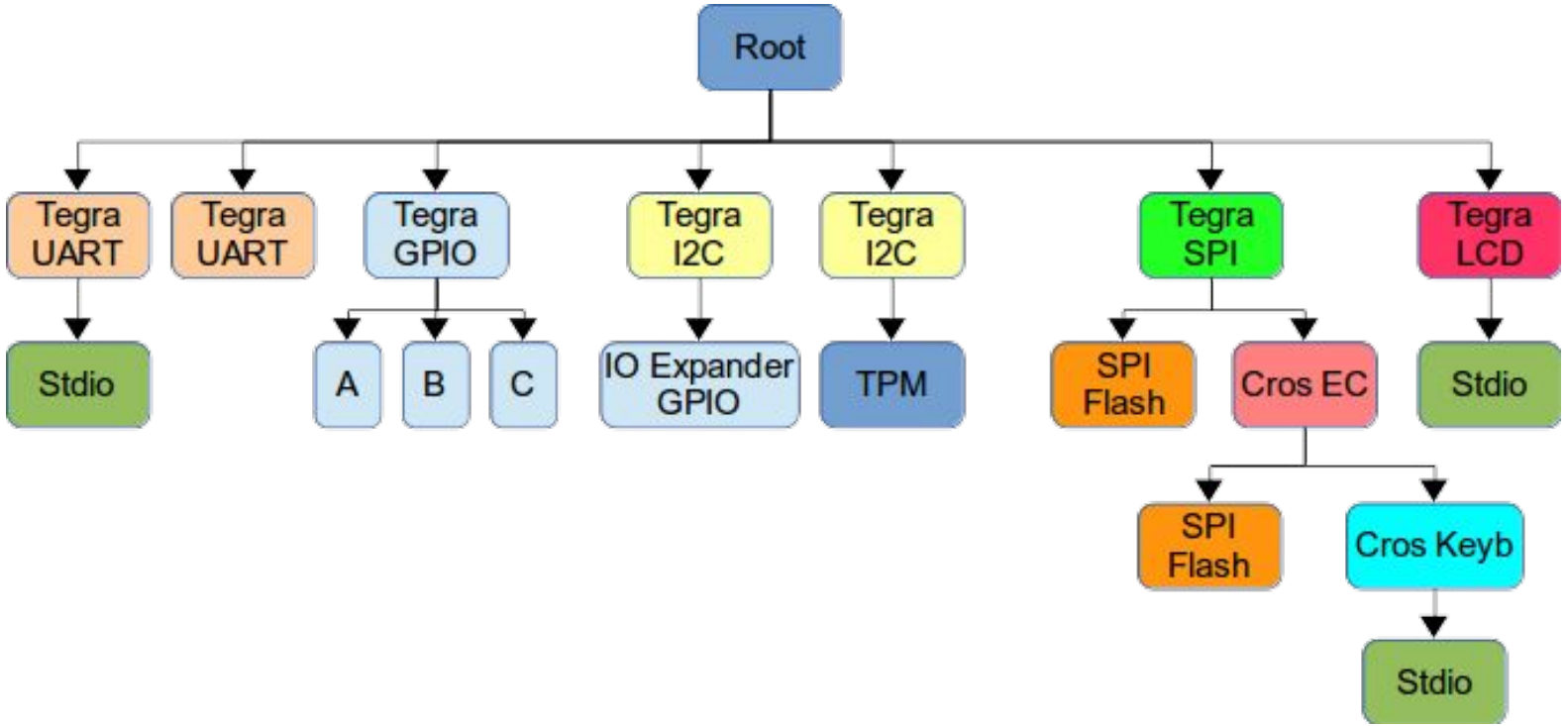
Driver model design goals

- Simple
- Lazy initialisation
- Small memory overhead
- Small execution overhead
- Support 'required' features
  - Device numbering, etc.
- History
  - Conceived in 2010 by Marek Vasut
  - University project in 2012 led by Marek with 3 collaborators
  - RFC in April 2013
  - v9 series merged in 2014.04

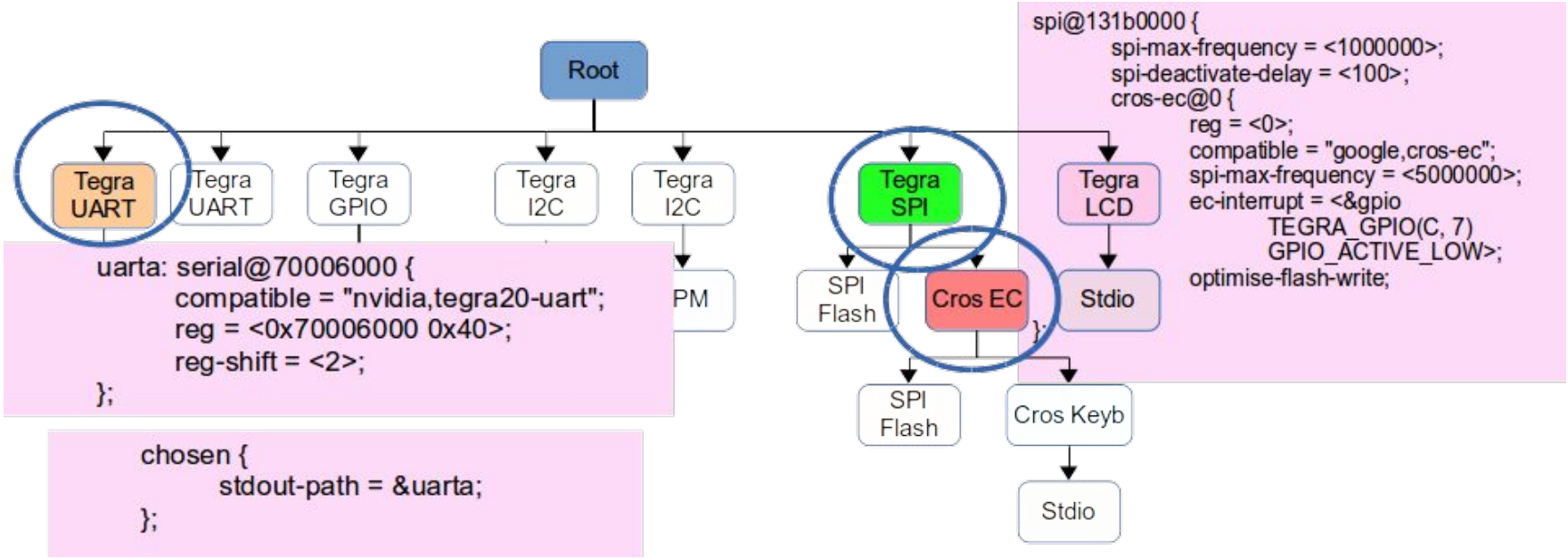
Architecture

- Uclass
  - A way of grouping devices which operate the same way
- Driver
  - Code to talk to a peripheral type (e.g. Ethernet switch, I2C controller, LCD)
- Device
  - Instance of a driver
  - Created from some platform-specific information bound to a driver
- Device tree and hierarchy
- Memory allocation
- Sequence numbers

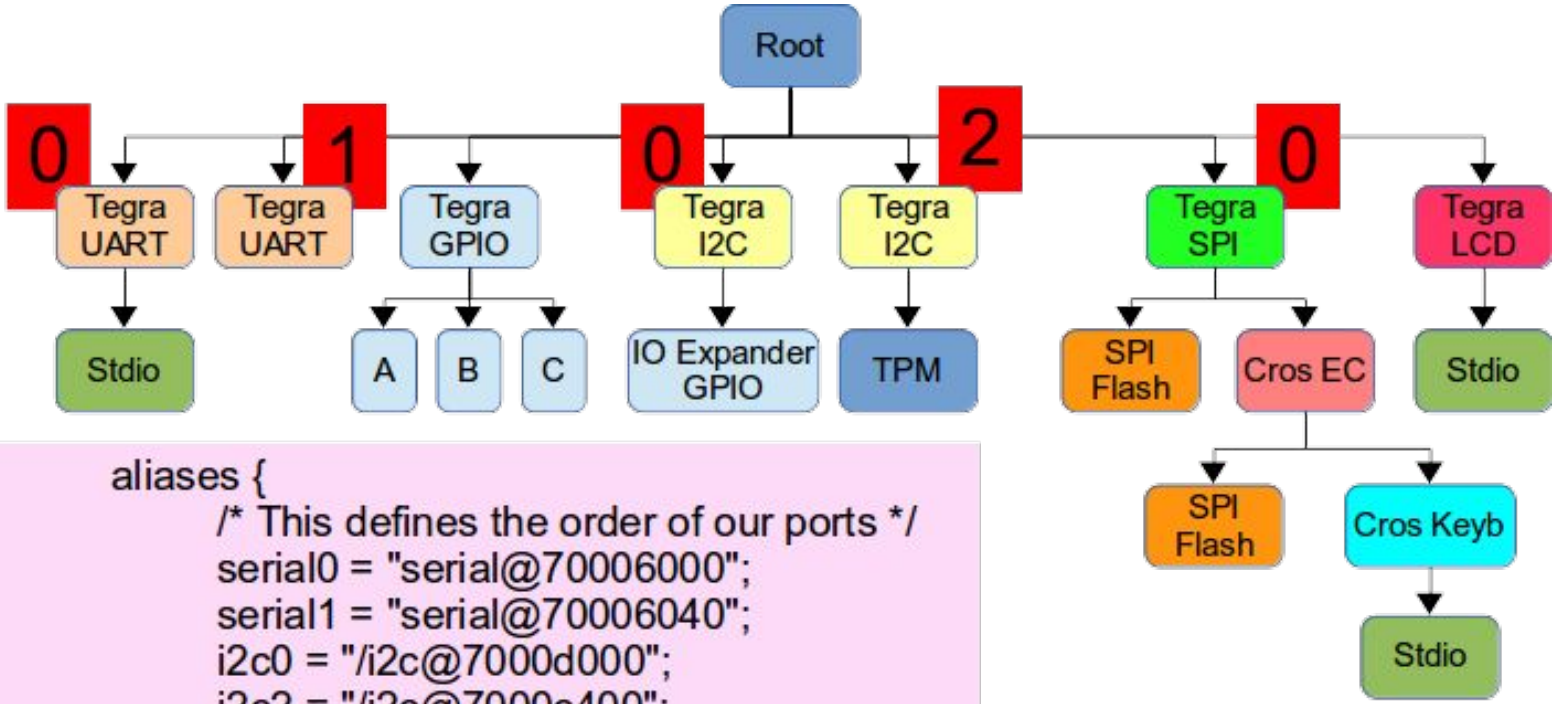
Device hierarchy



Device tree configuration

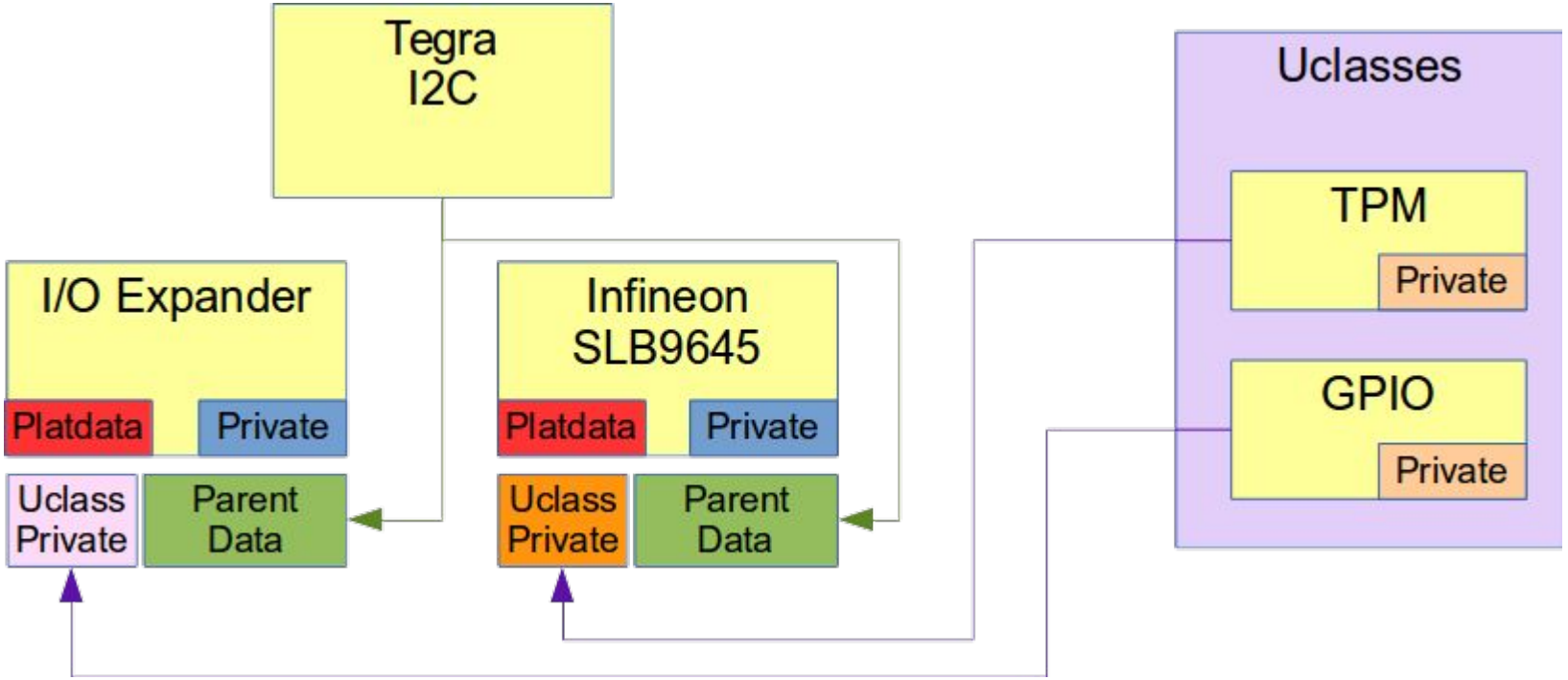


Device sequence




```
aliases {
        /* This defines the order of our ports */
        serial0 = "serial@70006000";
        serial1 = "serial@70006040";
        i2c0 = "/i2c@7000d000";
        i2c2 = "/i2c@7000c400";
};
```

Automatic memory allocation



Architecture 2

- Binding and probing
  - Binding creates the device but does not touch hardware
  - Probing activates the device ready for use
- Avoid private data structures
  - Everything out in the open
- SPL support
  - fdtgrep
  - Simple malloc()
  - Drop device removal code, warnings, etc.

Start-up sequence

- dm_init_and_scan():
- dm_init()
  - Creates an empty list of devices and uclasses
  - Binds and probes a root device
- dm_scan_platdata()
  - Scans available platform data looking for devices to be created
  - Platform data may only be used when memory constraints prohibit device tree
- dm_scan_fdt()
  - Scan device tree and bind drivers to nodes to create devices

Some useful features

- Bind only a subset of devices before relocation
- Debug UART for early debugging
- A few commands

```
=> dm tree
 Class         Probed   Name
-----------------------------------------
 root          [ + ]    root_driver
 serial        [ + ]    |-- serial
 rtc           [   ]    |-- rtc
 gpio          [   ]    |-- gpioa
 gpio          [   ]    |-- gpiob
 gpio          [   ]    |-- gpioc
 gpio          [   ]    |-- gpiod
 gpio          [   ]    |-- gpioe
 gpio          [   ]    |-- gpiof
 pci           [ + ]    |-- pci
 pci_generic   [   ]    |   |-- pci_0:0.0
 pci_generic   [   ]    |   |-- pci_0:2.0
 pci_generic   [   ]    |   |-- pci_0:11.0
```

Test methodology

- Automated tests cover core features and all uclasses
  - Current test suites is about 85 tests
  - Runs in a few seconds
- Tests use sandbox
  - Do not need to run on real hardware
  - Emulation drivers are provided for each uclass
- Mostly unit tests
  - A few functional tests (e.g USB flash stick)
- "What is not automatically tested does not work"

Driver model benefits

- Consistent view of devices regardless of their type
- Multiple drivers can be used with the same subsystem
  - Drivers can be created which use others drivers for their transport layer
- The lifecycle of a device is clear and consistent
- Devices can be bound automatically
  - Then probed automatically when used
- Supports device tree for configuration
  - Thus sharing this with Linux and potentially other projects
  - Avoids recreating the same information again in a different format

Limitations

- A driver can be in only one uclass
  - Multi-function devices must use separate child devices
- Uses flattened device tree
  - Driver model uses the device tree offset
  - Overlays and other mutations are not supported

Code size

- Individual control over feature set
  - Pinctrl, simple-bus, device removal, warnings
  - E.g. MMC uclass is linked in only if CONFIG_DM_MMC is enabled
- Device tree code is optional (CONFIG_[SPL_]OF_CONTROL)
- Totals for drivers/core/

Architecture	Code size *	Data size
ARM	9051	280
PowerPC	10379	336
Thumb 2	5745	280
x86 (32-bit)	11970	280

* includes command-line code in dump.o

Code size example *

Features	Code size	Minimal total size	Incremental Overhead	Total Overhead
Driver model, device tree, serial, printf()	8816	9501	3319	8125
Driver model, device tree, serial	6182	7069	2814	4806
Driver model, serial	3368	3648	1992	1992
Serial (without driver model)	1376	1376	-	-

* Firefly based on Rockchip RK3288, using gcc 4.9 with rodata bug fixed: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=54303

Data size

- Core structure sizes are moderate
- Device tree can be cut down with fdtgrep
  - E.g. Rockchip RK3288: 33KB -> 3757 bytes

U-Boot structure	Size (32-bit)	Size (64-bit)
struct udevice	84	176
struct uclass	24	48
struct driver	68	120

CPU overhead

- Must run efficiently at slow clock speed
  - Various features help with this
- Small amount of core code!
- Simple malloc()
- Only bind devices that are needed
  - Mark those needed in SPL and before relocation
  - Device tree property / driver flag
- Only probe devices when needed
- Cut-down device tree (fdtgrep)

CPU overhead examples

Board	CPU details	SPL Time (µs)	Pre-relocation time (µs)	Post-relocation time (µs)
Beaglebone Black	800MHz TI OMAP4 (Cortex-A8)	-	<1,000, 8,000	<1,000 / <1,000
Firefly RK3288	1.8GHz Rockchip RK3288 (Cortex-A17)	90 / 1,098	75 / 3,474	54 / 1,932
Link (Pixel 2013)	1.8GHz Intel Core i5	-	1,052 / 339	5 / 34
Minnowmax	1.33GHz Intel Atom E3825	-	3,443 / 2,027	7 / 30
Nyan (Chromebook)	2.3GHz Tegra K1 (Cortex-A15)	527 / 517	14 / 65	516 / 508
Snapper 9260	180MHz Atmel AT91SAM9260 (ARM926Ej-S)	-	53 / 148	111 / 258
Snow (Chromebook)	1.7GHz Samsung Exynos 5250 (Cortex-A15)	-	32 / 2,485	5 / 172

Time to execute dm_init_and_scan() / time to start serial driver

Comparing to Linux

- Classes
- Buses
- Binding and probing
- Memory allocation
- Relocation and SPL
- Device visibility
- Locking

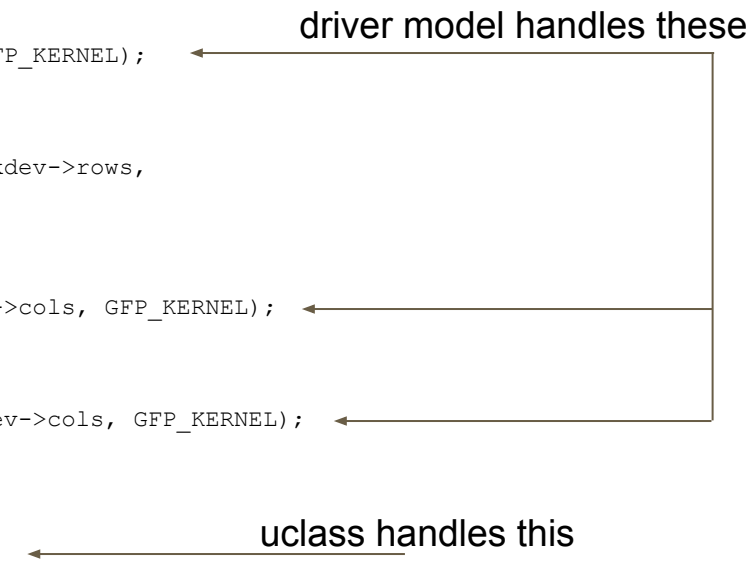
Comparing to Linux: data structure size

Structure	U-Boot structure	Linux size	U-Boot size	Relative
struct device	struct udevice	480	84	18%
struct class	struct uclass	284	24	8%
struct device_driver	struct driver	144	68	47%

Comparing to Linux: automatic memory allocation

- Linux code

```
struct cros_ec_keyb *ckdev;

ckdev = devm_kzalloc(&pdev->dev, sizeof(*ckdev), GFP_KERNEL);

if (!ckdev)
       return -ENOMEM;
err = matrix_keypad_parse_of_params(&pdev->dev, &ckdev->rows,
                                   &ckdev->cols);
if (err)
       return err;

ckdev->valid_keys = devm_kzalloc(&pdev->dev, ckdev->cols, GFP_KERNEL);
if (!ckdev->valid_keys)
       return -ENOMEM;

ckdev->old_kb_state = devm_kzalloc(&pdev->dev, ckdev->cols, GFP_KERNEL);
if (!ckdev->old_kb_state)
       return -ENOMEM;

idev = devm_input_allocate_device(&pdev->dev);
if (!idev)
       return -ENOMEM;
```



Using driver model with your board

- Examples
  - Exynos boards (e.g. snow)
  - Allwinner (sunxi) board (e.g. A20-OLinuXino_MICRO)
  - Any tegra board (e.g. jetson_tk1)
  - Any x86 board (e.g. qemu-x86)
  - glacier_ramboot (PowerPC)
  - firefly-rk3288

Example 1: Requesting GPIOs

board_config.h:

```
#define CONFIG_EC_INTERRUPT       97     /* GPX11, active low! */
```

code:

```
#ifdef CONFIG_EC_INTERRUPT
       ret = gpio_request(CONFIG_EC_INTERRUPT, "cros-ec");
       if (ret)
              return ret;
       ret = gpio_direction_in(CONFIG_EC_INTERRUPT);
       if (ret)
              return ret;
       val = !gpio_get_value(CONFIG_EC_INTERRUPT)
#endif
```

device tree fragment:

```
       cros-ec {
              ec-interrupt-gpios = <&gpx1 6 GPIO_ACTIVE_LOW>;
       };
```

code (within cros_ec device probe() method):

```
       struct gpio_desc ec_int;

       gpio_request_by_name(dev, "ec-interrupt-gpios", 0, &ec_int,
                     GPIOD_IS_IN);
       if (dm_gpio_is_valid(&ec_int))
              val = gpio_get_value(&ec_int);
```

Example 2: Enabling power

device tree fragments:

```
ldo6_reg: LDO6 {
    regulator-name = "vdd_mydp";
    regulator-min-microvolt = <1200000>;
    regulator-max-microvolt = <1200000>;
    regulator-always-on;
    op_mode = <3>;
};
ps8622-bridge@8 {
    power-supply = <&ldo6_reg>;
};
```

code (in ps8622 driver attach() method):

```
ret = uclass_get_device_by_phandle(UCLASS_REGULATOR, dev,
                "power-supply", &reg);
if (!ret)
    ret = regulator_autoset(reg);
```

Porting a driver to driver model

- Good opportunity to clean up the code
- Global/static variables move into a struct - dev_get_priv()
- API changes
  - At minimum, adds a struct udevice *
  - But a minor rethink is desirable
- Often need to continue to support non-driver-model boards
- Convert to use Kconfig
- Overall fairly invasive
- See example patches in the notes

Status update

- Work started in 2014.04
- 32 uclasses supported so far
- Major remaining work:
  - block devices
  - environment
  - display (LCD and video)
  - filesystems
  - stdio
  - usb gadget

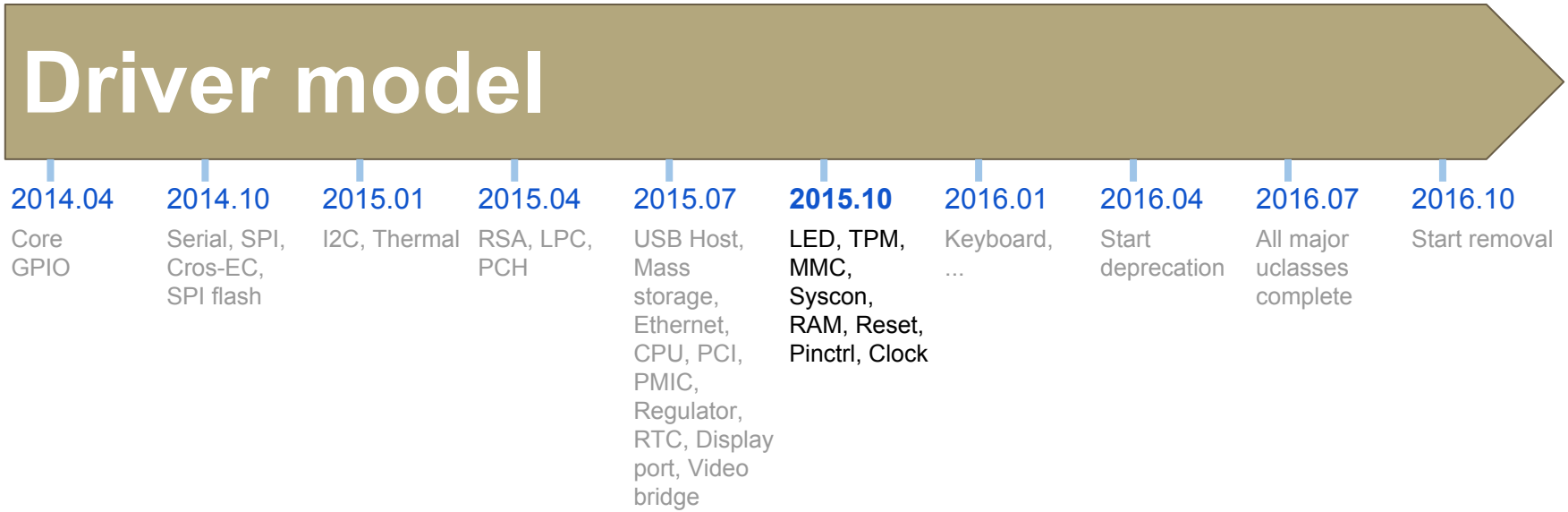
When will we be done?

$$\frac{\text{650 drivers remaining to convert}}{\text{400 contributors to U-Boot in the last 12 months}} \cong \text{2 years}$$

$$\frac{\text{830 boards remaining to convert *}}{\text{400 contributors to U-Boot in the last 12 months}} \cong \text{2 years}$$

* With 2015.10, 234 out of 1064 boards enable CONFIG_DM. So far there are 708 commits tagged with 'dm:'

Timeline


Driver model
2014.04
Core GPIO
2014.10
Serial, SPI, Cros-EC, SPI flash
2015.01
I2C, Thermal
2015.04
RSA, LPC, PCH
2015.07
USB Host, Mass storage, Ethernet, CPU, PCI, PMIC, Regulator, RTC, Display port, Video bridge
2015.10
LED, TPM, MMC, Syscon, RAM, Reset, Pinctrl, Clock
2016.01
Keyboard, ...
2016.04
Start deprecation
2016.07
All major uclasses complete
2016.10
Start removal

Conclusion

- Driver model is ready for prime time
- Substantial improvement in organisation and use of drivers
- Active and ongoing effort to convert the code base
  - Majority of subsystems are supported in U-Boot 2015.10
  - Much work remains to convert all drivers and boards
- Overhead is moderate, even in SPL
- Can support SoCs and boards of arbitrary complexity

Questions

- Full ELCE paper at https://goo.gl/F75qIQ
- See the README:
  - http://git.denx.de/?p=u-boot.git;a=blob;f=doc/driver-model/README.txt
- Contact details:
  - Simon Glass <sjg@chromium.org>
  - **Cc: U-Boot Mailing List <u-boot@lists.denx.de>**
  - IRC sjg1
  - https://plus.google.com/+SimonGlass