U-Boot Driver Model

Marek Vašut

July 12, 2012

Outline

What?

- What is the driver model?
- Why do we need it?
- How will we implement it?
- When will it be deployed?

What is the driver model?

- An unified framework for device drivers

Is

A way of writing device drivers

Provides

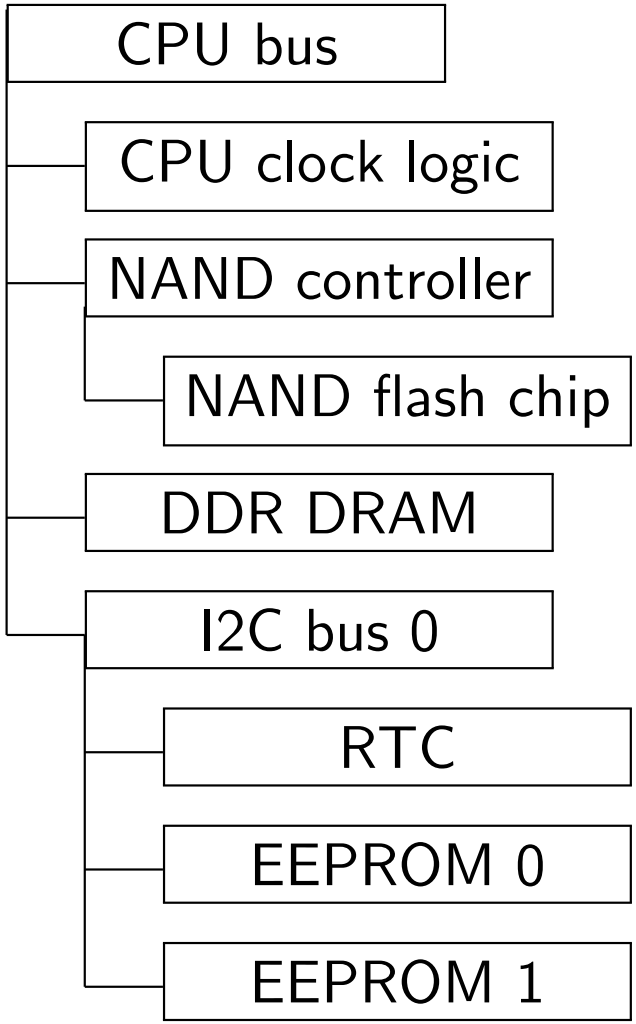
Software representation of how hardware is connected

Organises

The system in tree-like structure

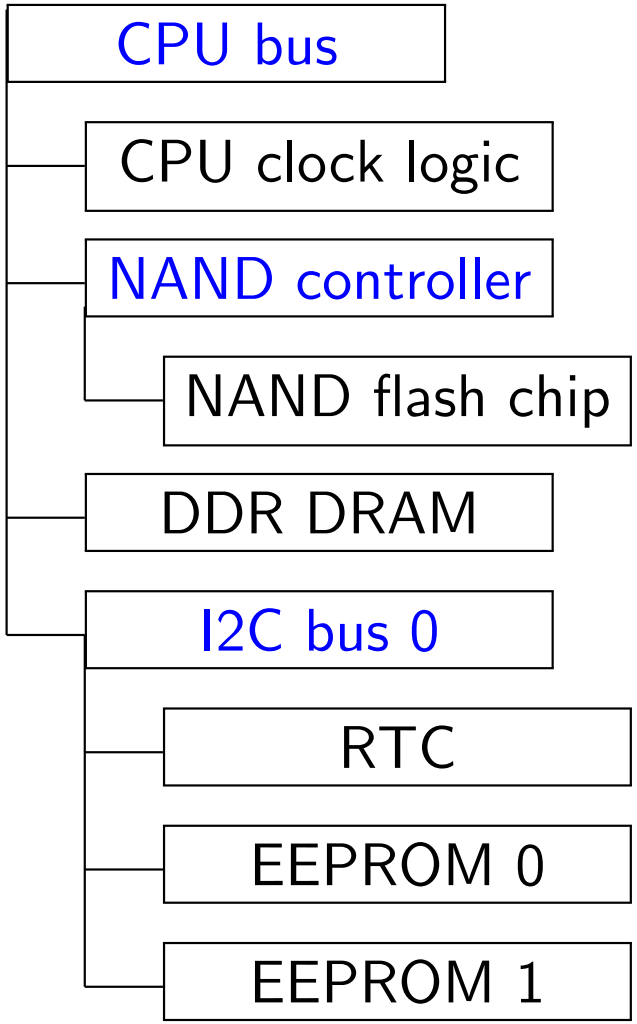
What is the driver tree?

- A tree-like structure
- Nexus nodes represent busses
- Leaf nodes represent devices



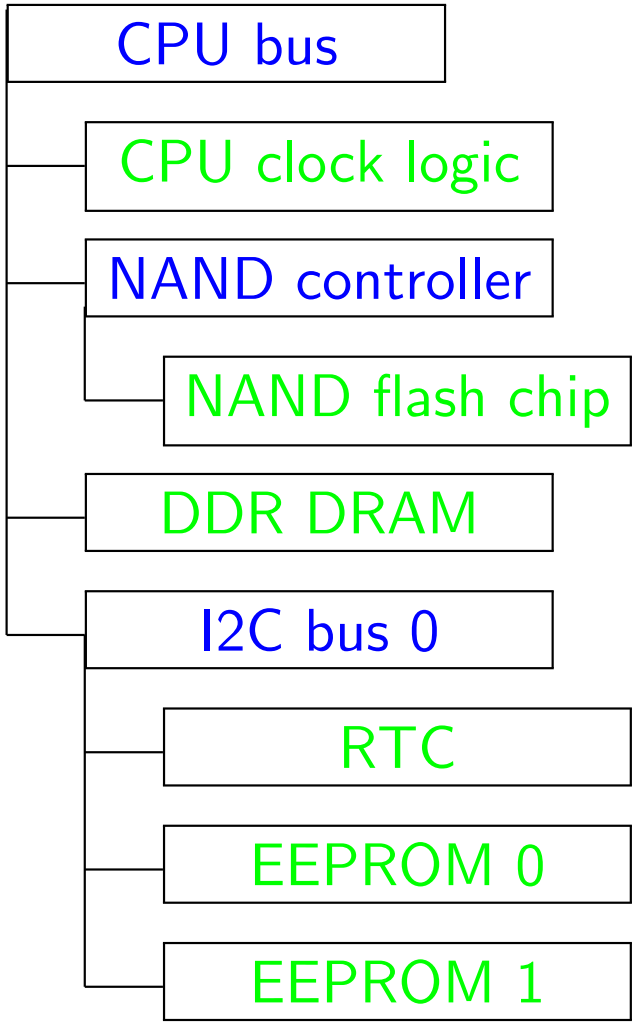
What is the driver tree?

- A tree-like structure
- Nexus nodes represent busses
- Leaf nodes represent devices



What is the driver tree?

- A tree-like structure
- Nexus nodes represent busses
- Leaf nodes represent devices



denx software engineering

Why do we need it?

U-Boot . . .

- is full of `#ifdef` – `#else` – `#endif` constructs
- is mostly configured by changing `#define`-d values
- has hard time supporting multiple devices of the same type
  - We have per-device-type ad-hoc implementations
  - More such ad-hoc hacks are starting to appear
- all in all simply doesn't scale anymore

How will we implement it?

The implementation breaks down into several stages:

- Move drivers from *arch/* to *drivers/*
  - EASY
- Introduce the driver model core mechanisms.
  - HARD 
- Convert drivers onto the new driver model.
  - EXTRA HARD 
- Introduce early mallocator for early drivers.
  - MEDIUM 

Introduce the driver model core mechanisms

Driver's point of view:

- U-Boot must be aware of driver's existence

Introduce the driver model core mechanisms

Driver's point of view:

- U-Boot must be aware of driver's existence

```
struct driver __attribute__((section(driver_list)))
```

Introduce the driver model core mechanisms

Driver's point of view:

- U-Boot must be aware of driver's existence

  ```
  struct driver __attribute__((section(driver_list)))
  ```

- Driver's instances must make the U-Boot aware of them

Introduce the driver model core mechanisms

Driver's point of view:

- U-Boot must be aware of driver's existence

```
struct driver __attribute__((section(driver_list))) {
        char    name[LENGTH];
        int     (*bind)(struct instance *i);
        int     (*probe)(struct instance *i);
        int     (*reloc)(struct instance *i);
        int     (*remove)(struct instance *i);
        int     (*unbind)(struct instance *i);
};
```

- Driver's instances must make the U-Boot aware of them

Introduce the driver model core mechanisms

Driver's point of view:

- U-Boot must be aware of driver's existence
- Driver's instances must make the U-Boot aware of them
- Driver must be able to create multiple independent instances of itself

Introduce the driver model core mechanisms

Driver's point of view:

- U-Boot must be aware of driver's existence
- Driver's instances must make the U-Boot aware of them
- Driver must be able to create multiple independent instances of itself

```
struct driver_instance {
        uint32_t                flags;
        struct instance         i;
};
struct instance {
        const struct driver_info  *info;
        struct instance           *bus;
        void                      *private_data;
        struct successor_block    *succ;
};
```

Introduce the driver model core mechanisms

U-Boot's point of view:

- Must be able to track driver instances
- Must be able to use driver instances
- Concept of cores
  - Special single-instance kind of driver
  - Tracks driver instances of certain class
  - Provides unified access API for class of devices
  - Driver binds with the core using `bind()` function

Introduce the driver model core mechanisms

Programmer's point of view:

- Is presented with a (virtual) root bus
- Must create the bindings between devices and busses:

```
struct driver_info {
        char name[LENGTH];
        void *platform_data;
}

struct instance *
driver_bind(struct instance *parent,
            const struct driver_info *di));
```

Review of the workage

- The programmer calls `driver_bind()`
- Driver is located
- Instance is allocated and initialized
- `bind()` is called

The result:

- The particular core is aware of the driver's instance
- BUT, the driver is not yet running

Starting the driver

- Driver is started by calling `driver_activate()`
- This is handled by the core upon first use of the driver
- This in turn calls it's `.probe()` function
- That's when the hardware is initialized!

Summary

Pros:

- Lazy initialization of devices
- Results in faster boot times
- No ad-hoc hacks to allow multiple devices
- Much less `#define`-d values
- Model close to Linux kernel's one

Cons:

- The bootloader is a bit bigger
- But it's mostly static data in ROM
- Slightly more memory is used

Problems

- Drivers that need to be initialized very early
  - Introduce early stack-based mallocator
  - Allocate only, stub `free()`
- Relocation of driver's internal data
  - Introduce `reloc()` call
- Writing the whole binding is bothersome
  - Per-CPU generic bindings
  - Per-architecture generic binding
  - ...

When?

- During the next 6 months
- That's two releases away

What next?

- Review and discuss the design
- Update the prototype code
- Let it all get mainline
- Enjoy the result

The End

Thanks to:

- You
  - For your attention!
- Pavel Herrmann, Viktor Krivak, Tomas Hlavacek
  - For being a great team on this project
- Graeme Russ
  - For coming up with wild and crazy ideas
- The DENX crew
  - For being good friends, helpful advisors and constructive critics
- Wolfgang Denk
  - For countering my crazy ideas with real-world examples and for tolerating my flubs

Questions?

?

Contact: Marek Vasut <marex@denx.de>
Mailing list: u-boot-dm@lists.denx.de