Driver Model, Kconfig and a little Patman

U-Boot Miniconf
Edinburgh Oct-13

Simon Glass
Google, Inc

Agenda

- Driver model
  - Why
  - Current status
- Kconfig
  - Why
  - Current status
- A quick Patman demo

Why a Driver Model?

- Device init and access is ad-hoc
- Many subsystems allow only one driver
- Communication between subsystems is tricky
- Hard to answer simple questions
- Much relies on the board file

History

- University of Prague project
  - Marek Vasut, Pavel Herrmann, Viktor Křivák, Tomas Hlavacek
- Stalled in late 2012
- Picked up again in 2013
  - Now at v5

Changes since 'version 1'

- Simplified and removed boilerplate
- Core → uclass
- Dropped specific relocation support for now
  - Adopt another approach
- Device tree integration
- Added test suite, sandbox GPIO driver, demo

Key Concepts

- Tree structure
- Device – struct device
- Driver – struct driver
  - struct driver_info to attach platform_data
  - Or use Device Tree
- U-Boot umbrella class – struct uclass

A device

```
struct device {
    struct driver *driver;
    const char *name;
    void *platform_data;
    int of_offset;
    struct device *parent;
    void *priv;
    struct uclass *uclass;
    void *uclass_priv;
    struct list_head uclass_node;
    struct list_head child_head;
    struct list_head sibling_node;
    uint32_t flags;
}
```

A driver

```
struct driver {
    char *name;
    enum uclass_id id;
    const struct device_id *of_match;
    int (*bind)(struct device *dev);
    int (*probe)(struct device *dev);
    int (*remove)(struct device *dev);
    int (*unbind)(struct device *dev);
    int priv_auto_alloc_size;
    int platform_data_auto_alloc_size;
    const void *ops; /* driver-specific operations */
};
```

A uclass driver

```
struct uclass_driver {
    const char *name;
    enum uclass_id id;
    int (*post_bind)(struct device *dev);
    int (*pre_unbind)(struct device *dev);
    int (*post_probe)(struct device *dev);
    int (*pre_remove)(struct device *dev);
    int (*init)(struct uclass *class);
    int (*destroy)(struct uclass *class);
    int priv_auto_alloc_size;
    int per_device_auto_alloc_size;
    const void *ops;
};
```

uclass function: gpio_get_value()

```
int gpio_get_value(unsigned gpio)
{
        unsigned int offset;
        struct device *dev;
        int ret;

        ret = gpio_to_device(gpio, &dev, &offset);
        if (ret)
        return ret;

        return gpio_get_ops(dev)->get_value(dev, offset);
}
```

Sandbox's GPIO driver

```
static const struct dm_gpio_ops gpio_sandbox_ops = {
    .request        = sb_gpio_request,
    .free           = sb_gpio_free,
    .direction_input   = sb_gpio_direction_input,
    .direction_output= sb_gpio_direction_output,
    .get_value      = sb_gpio_get_value,
    .set_value      = sb_gpio_set_value,
    .get_state      = sb_gpio_get_state,
};
```

```
U_BOOT_DRIVER(gpio_sandbox) = {
    .name  = "gpio_sandbox",
    .id    = UCLASS_GPIO,
    .of_match = sandbox_gpio_ids,
    .probe  = gpio_sandbox_probe,
    .ops= &gpio_sandbox_ops,
};
```

Platform Data

```
static const struct dm_demo_pdata red_square = {
    .colour = "red",
    .sides = 4.
};

U_BOOT_DEVICE(demo0) = {
    .name = "demo_shape_drv",
    .platform_data = &red_square,
};
```

Device Tree

```
square {
   compatible = "demo-shape";
   colour = "blue";
   sides = <4>;
};
```

```
int demo_parse_dt(struct device *dev)
{
   struct dm_demo_pdata *pdata = dev->platform_data;
   int dn = dev->of_offset;

   if (dev->of_offset < 0)
      return 0;
   pdata->sides = fdtdec_get_int(gd->fdt_blob, dn, "sides", 0);
   pdata->colour = fdt_getprop(gd->fdt_blob, dn, "colour", NULL);
   if (!pdata->sides || !pdata->colour) {
      debug("%s: Invalid device tree data\n", __func__);
      return -EINVAL;
   }

   return 0;
}
```

Demo

- 'dm' command
- 'demo'

Next steps

- Review, discuss, adjust, agree, merge
- Announce
- Encourage conversion of major subsystems
  - I2C, GPIO, block, …
  - Require automated tests for each uclass
- Major work is creating a sensible uclass
  - Adjust drivers to use it is fairly straightforward

Deep Breath...

Why Kconfig?

- Menu-driver configuration
- Configuration dependencies more obvious
- Load/save user configurations
- Better build system?
- Familiar to Linux users

Kconfig Progress

- Initial series in May
  - Build system first
  - Omitted SPL, out-of-tree building, binary output!
- Since them Masahiro has picked it up
  - Approaching from the other size (Makefiles first)
  - Future series will being in Kconfig, build scripts

Another Deep Breath...

Patman demo

- ~/.patmanrc
- Example series
- Make changes
- patman -n
- patman

```
# ~/.patmanrc alias file

[alias]
me: Simon Glass <sjg@chromium.org>

marek: Marek Vasut <marex@denx.de>

u-boot: U-Boot Mailing List <u-boot@lists.denx.de>
mikef: Mike Frysinger <vapier@gentoo.org>
joe: Joe Hershberger <joe.hershberger@ni.com>
mkimage: joe, marek
...
```

Tags

- Series-to, Series-cc
- Series-version, Series-changes
- Series-prefix, Series-notes, Cover-letter
- Series-process-log
- README is here
  - http://git.denx.de/?p=u-boot.git;a=blob;f=tools/patman/README;