Supporting multi-function devices in the Linux kernel: a tour of the mfd, regmap and syscon APIs

Alexandre Belloni
free electrons
alexandre.belloni@free-electrons.com

Alexandre Belloni

- Embedded Linux engineer at ***free electrons***
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
- Open-source contributor
  - Maintainer for the Linux kernel **RTC susbsystem**
  - Co-Maintainer of the **kernel support for Atmel ARM processors**
  - Contributing the **kernel support for Marvell ARM (Berlin) processors**

What is a multi-function device ?

- An external peripheral or a hardware block exposing more than a single functionality
- Examples:
  - PMICs
    - da9063: regulators, led controller, watchdog, rtc, temperature sensor, vibration motor driver, ON key
    - max77843: regulators, charger, fuel gauge, haptic feedback, LED controller, micro USB interface controller
    - wm831x: regulator, clocks, rtc, watchdog, touch controller, temperature sensor, backlight controller, status LED controller, GPIOs, ON key, ADC
    - some even include a codec
  - atmel-hlcdc: display controller and backlight pwm
  - Diolan DLN2: USB to I2C, SPI and GPIO controllers
  - Realtek PCI-E card reader: SD/MMC and memory stick reader
- The main issue is to register those in different kernel subsystems. In particular the external peripheral are represented by only one `struct device` (or the specialized `i2c_client` or `spi_device`)

MFD subsystem

- The MFD subsystem has been created to handle those devices
- Allows to register the same device in multiple subsystems
- The MFD driver has to multiplex accesses on the bus (mainly takes care of locking) and handle IRQs
- may handle clocks
- may also need to configure the IP
- may do variant or functions detection
- Other benefit: allows driver reuse, multiple MFD can reuse drivers from other subsystems.

MFD API

- Defined in `include/linux/mfd/core.h`
- Implemented in `drivers/mfd/mfd-core.c`

```
int mfd_add_devices(struct device *parent, int id,
                    const struct mfd_cell *cells, int n_devs,
                    struct resource *mem_base,
                    int irq_base, struct irq_domain *irq_domain);
```

```
extern void mfd_remove_devices(struct device *parent);
```

- Also `mfd_add_hotplug_devices`, `mfd_clone_cell`, `mfd_cell_enable`, `mfd_cell_disable` but they are seldom used.

struct mfd_cell

```
struct mfd_cell {
        const char                *name;
        int                        id;
--*--*--*--
        /* platform data passed to the sub devices drivers */
        void                       *platform_data;
        size_t                       pdata_size;
        /*
         * Device Tree compatible string
         * See: Documentation/devicetree/usage-model.txt Chapter 2.2 for details
         */
        const char                *of_compatible;
--*--*--*--
        /*
         * These resources can be specified relative to the parent device.
         * For accessing hardware you should use resources from the platform dev
         */
        int                        num_resources;
        const struct resource        *resources;
--*--*--*--
};
```

Example: tps6507x - registration

```
static const struct i2c_device_id tps6507x_i2c_id[] = {
        { "tps6507x", 0 },
        { }
};
MODULE_DEVICE_TABLE(i2c, tps6507x_i2c_id);

#ifdef CONFIG_OF
static const struct of_device_id tps6507x_of_match[] = {
        {.compatible = "ti,tps6507x", },
        {},
};
MODULE_DEVICE_TABLE(of, tps6507x_of_match);
#endif

static struct i2c_driver tps6507x_i2c_driver = {
        .driver = {
                .name = "tps6507x",
                .of_match_table =
                        of_match_ptr(tps6507x_of_match),
        },
        .probe = tps6507x_i2c_probe,
        .remove = tps6507x_i2c_remove,
        .id_table = tps6507x_i2c_id,
};
```

```
static int __init tps6507x_i2c_init(void)
{
        return i2c_add_driver(&tps6507x_i2c_driver);
}
/* init early so consumer devices can complete system boot */
subsys_initcall(tps6507x_i2c_init);

static void __exit tps6507x_i2c_exit(void)
{
        i2c_del_driver(&tps6507x_i2c_driver);
}
module_exit(tps6507x_i2c_exit);
```

- registers as a simple i2c device
- only oddity `subsys_initcall(tps6507x_i2c_init);` to register early enough

Example: tps6507x - probing

```
static const struct mfd_cell tps6507x_devs[] = {
        {
                .name = "tps6507x-pmic",
        },
        {
                .name = "tps6507x-ts",
        },
};
```

- `tps6507x-pmic` in `drivers/regulator/tps6507x-regulator.c`
- `tps6507x-ts` in `drivers/input/touchscreen/tps6507x-ts.c`

```
static int tps6507x_i2c_probe(struct i2c_client *i2c,
                              const struct i2c_device_id *id)
{
        struct tps6507x_dev *tps6507x;

        tps6507x = devm_kzalloc(&i2c->dev, sizeof(struct tps6507x_dev),
                                GFP_KERNEL);
        if (tps6507x == NULL)
                return -ENOMEM;

        i2c_set_clientdata(i2c, tps6507x);
        tps6507x->dev = &i2c->dev;
        tps6507x->i2c_client = i2c;
        tps6507x->read_dev = tps6507x_i2c_read_device;
        tps6507x->write_dev = tps6507x_i2c_write_device;

        return mfd_add_devices(tps6507x->dev, -1, tps6507x_devs,
                               ARRAY_SIZE(tps6507x_devs), NULL, 0, NULL);
}
```

Example: tps6507x - `struct tps6507x_dev`

```
struct tps6507x_dev {
        struct device *dev;
        struct i2c_client *i2c_client;
        int (*read_dev)(struct tps6507x_dev *tps6507x, char reg, int size,
                        void *dest);
        int (*write_dev)(struct tps6507x_dev *tps6507x, char reg, int size,
                         void *src);
--✂--✂--✂--
};
```

- defined in `include/linux/mfd/tps6507x.h`
- Allows to pass the `i2c_client` and the accessors.
- `tps6507x.h` also contains the register definitions that can be used in the function drivers.

Example: tps6507x - function drivers

```
static int tps6507x_ts_probe(struct platform_device *pdev)
{
        struct tps6507x_dev *tps6507x_dev = dev_get_drvdata(pdev->dev.parent);
--✂--✂--✂--
};
```

```
static int tps6507x_pmic_probe(struct platform_device *pdev)
{
        struct tps6507x_dev *tps6507x_dev = dev_get_drvdata(pdev->dev.parent);
--✂--✂--✂--
};
```

- Easy to get the `struct tps6507x_dev` by using `dev.parent`

Example: da9063 - registering

```
static struct resource da9063_rtc_resources[] = {
        {
                .name         = "ALARM",
                .start          = DA9063_IRQ_ALARM,
                .end          = DA9063_IRQ_ALARM,
                .flags          = IORESOURCE_IRQ,
        },
        {
                .name         = "TICK",
                .start          = DA9063_IRQ_TICK,
                .end          = DA9063_IRQ_TICK,
                .flags          = IORESOURCE_IRQ,
        }
};
static const struct mfd_cell da9063_devs[] = {
--><--><--><--
        {
                .name                 = DA9063_DRVNAME_RTC,
                .num_resources         = ARRAY_SIZE(da9063_rtc_resources),
                .resources          = da9063_rtc_resources,
                .of_compatible         = "dlg,da9063-rtc",
        },
--><--><--><--
};
```

- resources are defined like it was done using `platform_data`
- in that case, they are named for easy retrieval
- when using `.of_compatible`, the function has to be a child of the MFD (see bindings)

Example: da9063 - drivers/rtc/rtc-da9063.c

```
static int da9063_rtc_probe(struct platform_device *pdev)
{
--%--%--%--
        irq_alarm = platform_get_irq_byname(pdev, "ALARM");
        ret = devm_request_threaded_irq(&pdev->dev, irq_alarm, NULL,
                                        da9063_alarm_event,
                                        IRQF_TRIGGER_LOW | IRQF_ONESHOT,
                                        "ALARM", rtc);
        if (ret) {
                dev_err(&pdev->dev, "Failed to request ALARM IRQ %d: %d\n",
                        irq_alarm, ret);
                return ret;
        }
--%--%--%--
};
```

- Use `platform_get_resource`, `platform_get_resource_byname`, `platform_get_irq`, `platform_get_irq_byname` to retrieve the resources
- Doesn't even need `dev.parent`, the same driver could be used for an MFD and a standalone chip.

Example: da9063 - DT bindings

```
pmic0: da9063@58 {
        compatible = "dlg,da9063"
        reg = <0x58>;
        interrupt-parent = <&gpio6>;
        interrupts = <11 IRQ_TYPE_LEVEL_LOW>;
        interrupt-controller;

        rtc {
                compatible = "dlg,da9063-rtc";
        };
--✂--✂--✂--
};
```

MFD: multiplexing register access

- A common way of multiplexing accesses to register sets is to use `regmap`.
- create the `regmap` from the MFD driver and pass it down to the children

regmap

- has its roots in ALSA
- can use I2C, SPI and MMIO (also SPMI)
- can handle locking when necessary
- can cache registers
- can handle endianness conversion
- can handle IRQ chips and IRQs
- can check register ranges
- handles read only, write only, volatile, precious registers
- handles register pages
- API is defined in `include/linux/regmap.h`
- implemented in `drivers/base/regmap/`

regmap: creation

```
#define regmap_init(dev, bus, bus_context, config)                      \
        __regmap_lockdep_wrapper(__regmap_init, #config,                \
                                 dev, bus, bus_context, config)
```

```
#define regmap_init_i2c(i2c, config)                                    \
        __regmap_lockdep_wrapper(__regmap_init_i2c, #config,            \
                                 i2c, config)
```

```
#define regmap_init_spi(dev, config)                                    \
        __regmap_lockdep_wrapper(__regmap_init_spi, #config,            \
                                 dev, config)
```

- Also `devm_` versions
- and `_clk` versions

regmap: accesses

- ```
  int regmap_read(struct regmap *map, unsigned int reg, unsigned int *val);
  ```
- ```
  int regmap_write(struct regmap *map, unsigned int reg, unsigned int val);
  ```
- ```
  int regmap_update_bits(struct regmap *map, unsigned int reg,
                         unsigned int mask, unsigned int val);
  ```
  ```

regmap: cache management

- `int regcache_sync(struct regmap *map);`
- `int regcache_sync_region(struct regmap *map, unsigned int min, unsigned int max);`
- `int regcache_drop_region(struct regmap *map, unsigned int min, unsigned int max);`
- `void regcache_cache_only(struct regmap *map, bool enable);`
- `void regcache_cache_bypass(struct regmap *map, bool enable);`
- `void regcache_mark_dirty(struct regmap *map);`

Example: atmel-hlcdc

include/linux/mfd/atmel-hlcdc.h

```
struct atmel_hlcdc {
        struct regmap *regmap;
        struct clk *periph_clk;
        struct clk *sys_clk;
        struct clk *slow_clk;
        int irq;
};
```

driver/mfd/atmel-hlcdc.c

```
static const struct regmap_config atmel_hlcdc_regmap_config = {
        .reg_bits = 32,
        .val_bits = 32,
        .reg_stride = 4,
        .max_register = ATMEL_HLCDC_REG_MAX,
        .reg_write = regmap_atmel_hlcdc_reg_write,
        .reg_read = regmap_atmel_hlcdc_reg_read,
        .fast_io = true,
};

static int atmel_hlcdc_probe(struct platform_device *pdev)
{
        struct atmel_hlcdc_regmap *hregmap;
        struct device *dev = &pdev->dev;
        struct atmel_hlcdc *hlcdc;
        struct resource *res;
--><--><--
        hlcdc->regmap = devm_regmap_init(dev, NULL, hregmap,
                                         &atmel_hlcdc_regmap_config);
        if (IS_ERR(hlcdc->regmap))
                return PTR_ERR(hlcdc->regmap);

        dev_set_drvdata(dev, hlcdc);
--><--><--
}
```

Example: pwm-atmel-hlcdc

```
static int atmel_hlcdc_pwm_probe(struct platform_device *pdev)
{
        const struct of_device_id *match;
        struct device *dev = &pdev->dev;
        struct atmel_hlcdc_pwm *chip;
        struct atmel_hlcdc *hlcdc;
        int ret;

        hlcdc = dev_get_drvdata(dev->parent);
--x---x---x--
        chip->hlcdc = hlcdc;
--x---x---x--
}
```

```
static int atmel_hlcdc_pwm_set_polarity(struct pwm_chip *c,
                                        struct pwm_device *pwm,
                                        enum pwm_polarity polarity)
{
        struct atmel_hlcdc_pwm *chip = to_atmel_hlcdc_pwm(c);
        struct atmel_hlcdc *hlcdc = chip->hlcdc;
        u32 cfg = 0;

        if (polarity == PWM_POLARITY_NORMAL)
                cfg = ATMEL_HLCDC_PWMPOL;

        return regmap_update_bits(hlcdc->regmap, ATMEL_HLCDC_CFG(6),
                                  ATMEL_HLCDC_PWMPOL, cfg);
}
```

Example: atmel-flexcom

- Sometimes an MFD only supports one simultaneous function.
- The MFD driver only configures the function.

```
static int atmel_flexcom_probe(struct platform_device *pdev)
{
        struct device_node *np = pdev->dev.of_node;
--✂--✂--✂--
        err = of_property_read_u32(np, "atmel,flexcom-mode", &opmode);
        if (err)
                return err;

        if (opmode < ATMEL_FLEXCOM_MODE_USART ||
            opmode > ATMEL_FLEXCOM_MODE_TWI)
                return -EINVAL;
--✂--✂--✂--
        writel(FLEX_MR_OPMODE(opmode), base + FLEX_MR);
--✂--✂--✂--
        return of_platform_populate(np, NULL, NULL, &pdev->dev);
}
```

Example: atmel-flexcom - DT bindings

```
flexcom@f8034000 {
        compatible = "atmel,sama5d2-flexcom";
        reg = <0xf8034000 0x200>;
        clocks = <&flx0_clk>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0x0 0xf8034000 0x800>;
        atmel,flexcom-mode = <2>;

        spi@400 {
                compatible = "atmel,at91rm9200-spi";
                reg = <0x400 0x200>;
                interrupts = <19 IRQ_TYPE_LEVEL_HIGH 7>;
                pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_flx0_default>;
--✂--✂--✂--
                };
        };
};
```

- The SPI driver from 2007 is reused and has not been modified to handle the MFD specifics.

syscon

- Sometimes, a set of registers is used to configure miscellaneous features from otherwise well separated IPs
- automatically creates a `regmap` when accessed
- defined in `include/linux/mfd/syscon.h`
- implemented in `drivers/mfd/syscon.c`

syscon: API

- ```
  extern struct regmap *syscon_node_to_regmap(struct device_node *np);
  ```
- ```
  extern struct regmap *syscon_regmap_lookup_by_compatible(const char *s);
  ```
- ```
  extern struct regmap *syscon_regmap_lookup_by_pdevname(const char *s);
  ```
- ```
  extern struct regmap *syscon_regmap_lookup_by_phandle(
                                          struct device_node *np,
                                          const char *property);
  ```

Example: pinctrl-dove.c

```
static int dove_pinctrl_probe(struct platform_device *pdev)
{
        struct resource *res, *mpp_res;
        struct resource fb_res;
        const struct of_device_id *match =
                of_match_device(dove_pinctrl_of_match, &pdev->dev);
        pdev->dev.platform_data = (void *)match->data;
--><--><--><--
        res = platform_get_resource(pdev, IORESOURCE_MEM, 1);
        if (!res) {
                dev_warn(&pdev->dev, "falling back to hardcoded MPP4 resource\n");
                adjust_resource(&fb_res,
                        (mpp_res->start & INT_REGS_MASK) + MPP4_REGS_OFFS, 0x4);
                res = &fb_res;
        }

        mpp4_base = devm_ioremap_resource(&pdev->dev, res);
        if (IS_ERR(mpp4_base))
                return PTR_ERR(mpp4_base);

        res = platform_get_resource(pdev, IORESOURCE_MEM, 2);
        if (!res) {
                dev_warn(&pdev->dev, "falling back to hardcoded PMU resource\n");
                adjust_resource(&fb_res,
                        (mpp_res->start & INT_REGS_MASK) + PMU_REGS_OFFS, 0x8);
                res = &fb_res;
        }

        pmu_base = devm_ioremap_resource(&pdev->dev, res);
        if (IS_ERR(pmu_base))
                return PTR_ERR(pmu_base);

        gconfmap = syscon_regmap_lookup_by_compatible("marvell,dove-global-config");
--><--><--><--
```

simple-mfd

- Simple DT binding
- Documented in `Documentation/devicetree/bindings/mfd/mfd.txt`
- Implemented in `drivers/of/platform.c`
- It is actually an alias to `simple-bus`
- Used in conjunction with `syscon` to create the `regmap`, it allows to avoid writing an MFD driver.

Example: system-timer

arch/arm/boot/dts/at91rm9200.dtsi

```
        st: timer@fffffd00 {
                compatible = "atmel,at91rm9200-st", "syscon", "simple-mfd";
                reg = <0xfffffd00 0x100>;
                interrupts = <1 IRQ_TYPE_LEVEL_HIGH 7>;
                clocks = <&slow_xtal>;

                watchdog {
                        compatible = "atmel,at91rm9200-wdt";
                };
        };
```

Example: system-timer

drivers/clocksource/timer-atmel-st.c

```
static struct regmap *regmap_st;
--✂--✂--✂--
static void __init atmel_st_timer_init(struct device_node *node)
{
        unsigned int val;
        int irq, ret;

        regmap_st = syscon_node_to_regmap(node);
        if (IS_ERR(regmap_st))
                panic(pr_fmt("Unable to get regmap\n"));

        /* Disable all timer interrupts, and clear any pending ones */
        regmap_write(regmap_st, AT91_ST_IDR,
                AT91_ST_PITS | AT91_ST_WDOVF | AT91_ST_RTTINC | AT91_ST_ALMS);
        regmap_read(regmap_st, AT91_ST_SR, &val);
--✂--✂--✂--
}
CLOCKSOURCE_OF_DECLARE(atmel_st_timer, "atmel,at91rm9200-st",
                       atmel_st_timer_init);
```

Example: system-timer

drivers/watchdog/at91rm9200_wdt.c

```
static struct regmap *regmap_st;
--✂--✂--✂--
static int at91wdt_probe(struct platform_device *pdev)
{
        struct device *dev = &pdev->dev;
        struct device *parent;
--✂--✂--✂--
        parent = dev->parent;
        if (!parent) {
                dev_err(dev, "no parent\n");
                return -ENODEV;
        }

        regmap_st = syscon_node_to_regmap(parent->of_node);
        if (IS_ERR(regmap_st))
                return -ENODEV;
--✂--✂--✂--
}
```

Questions?

Alexandre Belloni
alexandre.belloni@free-electrons.com