全志科技
Allwinner Technology

Tina

PMU 模块使用文档 v1.2

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2017/10/25		初始版本
V1.1	2018/1/16		适配 R30
V1.2	2018/4/11		更新

目　录

1. 概述

1.1. 编写目的

介绍 R30 项目的 Tina PMU 部分的 Linux 驱动使用。

1.2. 适用范围

硬件：AXP803是高度集成的电源系统管理芯片，针对单芯锂电池(锂离子或锂聚合物)且需要多路电源转换输出的应用，提供简单易用而又可以灵活配置的完整电源解决方案，充分满足目前日益复杂的应用处理器系统对于电源精确控制的要求。它可以搭配市面上出现的各款主控IC。

软件：Linux-4.4 内核

1.3. 相关人员

适用 Tina 平台的广大客户和相关技术人员。

2. AXP mfd 设计

2.1. mfd

mfd 多功能设备中，有可能存在多种设备（功能），比如触摸屏、键盘、led、电源管理芯片、rtc、音频 codec 等等。这些设备都通过一个或多个 IRQ 线和低速数据总线（SPI,I2C,RSB）跟主控进行交互。linux 为了统一管理多功能设备，提出了一种 mfd 设备驱动模型。

每一个子功能设备称之为 mfd 设备的一个 cell。

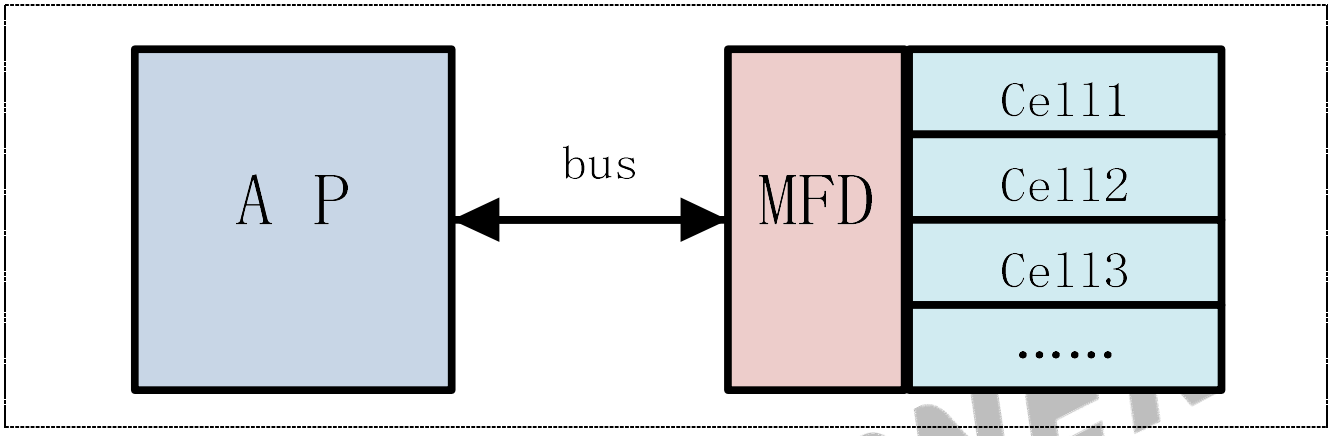

图 2-1 mfd 驱动模型

mfd core 是内核提供的 mfd 设备框架，在父设备 driver 中描述资源及子设备信息，然后向 mfd core 注册 mfd 设备，由 mfd core 负责创建子设备。子设备使用父设备的资源，当移除父设备时 mfd core 移除所有子设备。

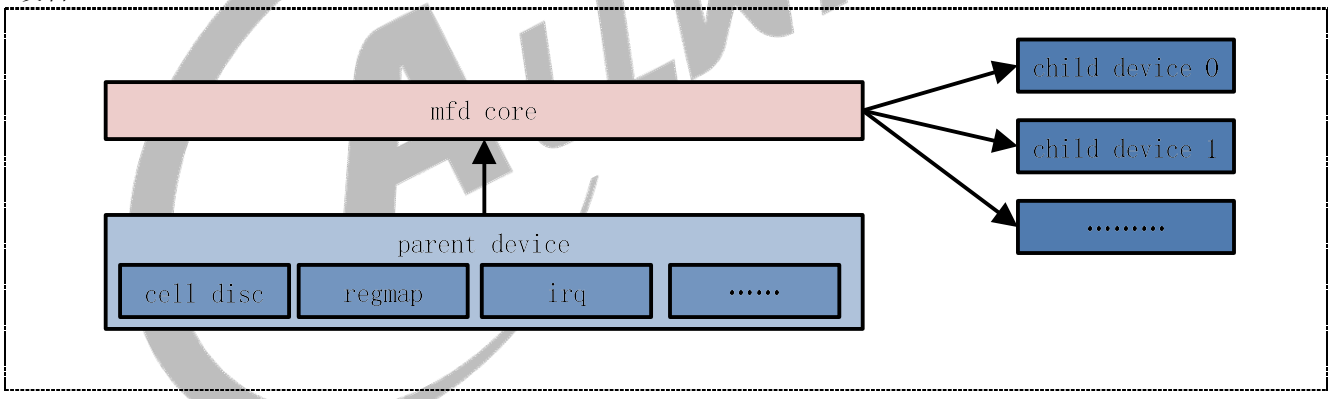

图 2-2 mfd core 模型

mfd_cell - 描述 mfd 设备的一个 cell 子设备，重点成员是子设备的名字及资源。

```
struct mfd_cell {
        const char              *name;    //子设备名
        int                     id;       //子设备 id
        /* refcounting for multiple drivers to use a single cell */
        atomic_t        *usage_count;
        int                     (*enable)(struct platform_device *dev);
        int                     (*disable)(struct platform_device *dev);

        int                     (*suspend)(struct platform_device *dev);
        int                     (*resume)(struct platform_device *dev);

        /* 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;

        /* Matches ACPI */
        const struct mfd_cell_acpi_match  *acpi_match;

        /*
         * These resources can be specified relative to the parent device.
         * For accessing hardware you should use resources from the platform dev
         */
        int                     num_resources; //子设备资源个数
        const struct resource       *resources;   //子设备资源列表

        /* don't check for resource conflicts */
        bool                    ignore_resource_conflicts;

        /*
         * Disable runtime PM callbacks for this subdevice - see
         * pm_runtime_no_callbacks().
         */
        bool                    pm_runtime_no_callbacks;

        /* A list of regulator supplies that should be mapped to the MFD
         * device rather than the child device when requested
         */
        const char * const   *parent_supplies;
        int                     num_parent_supplies;
};
```

2.2. regmap

regmap 是 linux3.1 加入进来的特性，提供了一种通用的接口来操作底层硬件上的寄存器，目的是提取出一些总线相关的注册、使能及读写公共部分，以提高代码的可重用性，并且使得在使用上变得简单易用。

注册 remap 后，设备可以使用 remap 框架提供的通用读写接口访问设备的寄存器，可以屏蔽总线级的读写操作。通常配合 mfd 框架使用，在父设备中根据不同的 bus 接法，注册 remap，子设备只需要获取到 remap 句柄则可以进行设备读写。

regmap 访问方式一般和 mfd 方式配置使用，降低父子设备之间的耦合性。axp 驱动中借鉴了内核的 regmap，建立了 axp regmap 访问方式。

3. AXP 驱动层次

3.1. 模块结构

AXP 的多功能设备驱动采用 i2c 或者 rsb 总线跟主控进行交互，使用 regmap 方式注册访问接口。将 AXP 按照功能抽象出数个子设备模块，共同使用父设备 axp mfd 的资源（bus、irq）。基本的软件结构图如图所示。

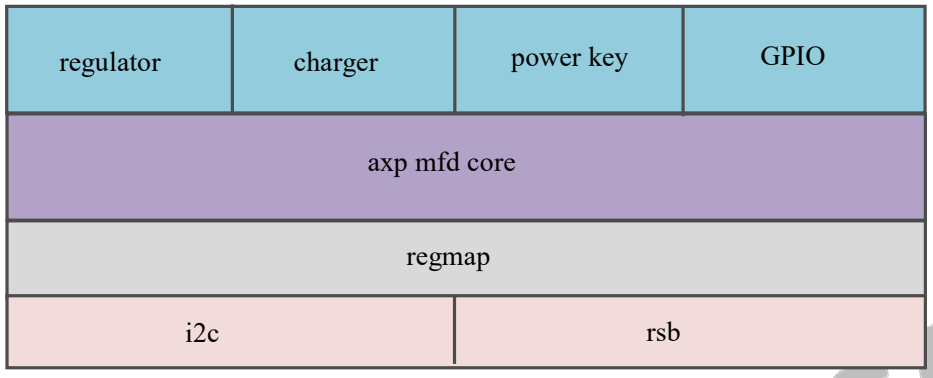

图 3-1 AXP 软件结构

将 axp 按照功能划分为几个子设备，分别是 regulator、charger、power key、gpio。每个子设备作为一个 cell，使用父设备的资源（bus，irq），与不同的内核子系统交互，实现完整的电源管理功能。

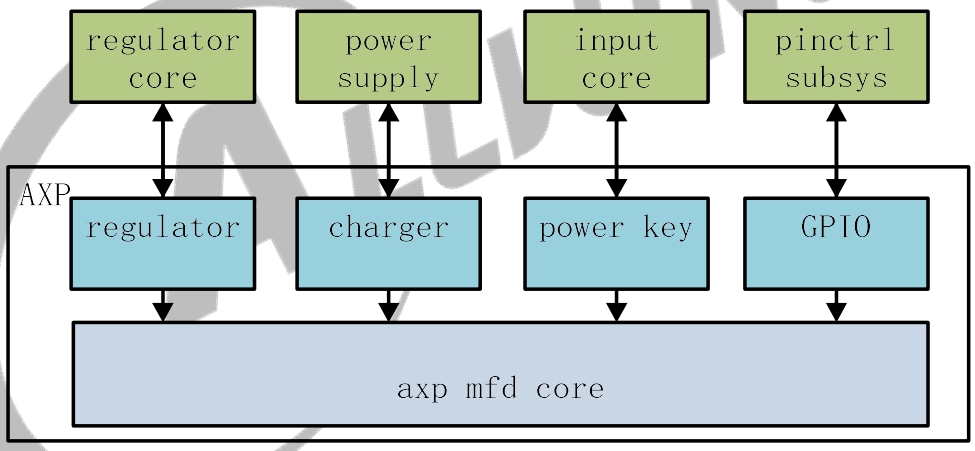

图 3-2 AXP 子设备结构

代码层次如下：

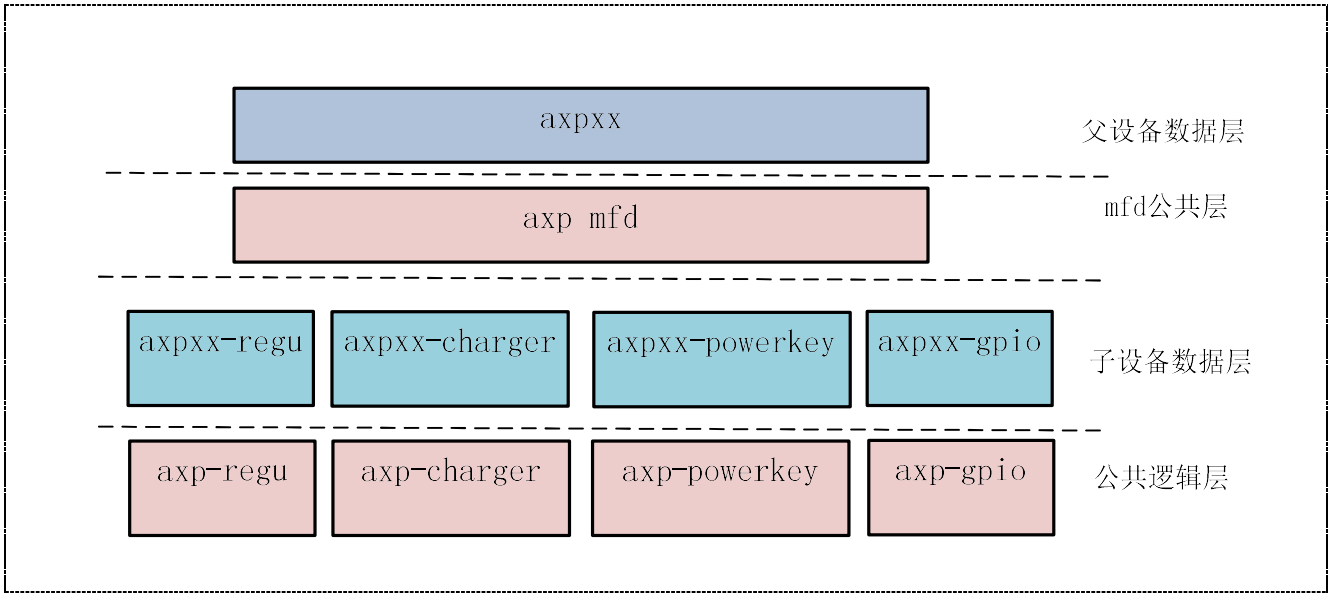

图 3-3 代码层次

以 axp152 为例，需要实现

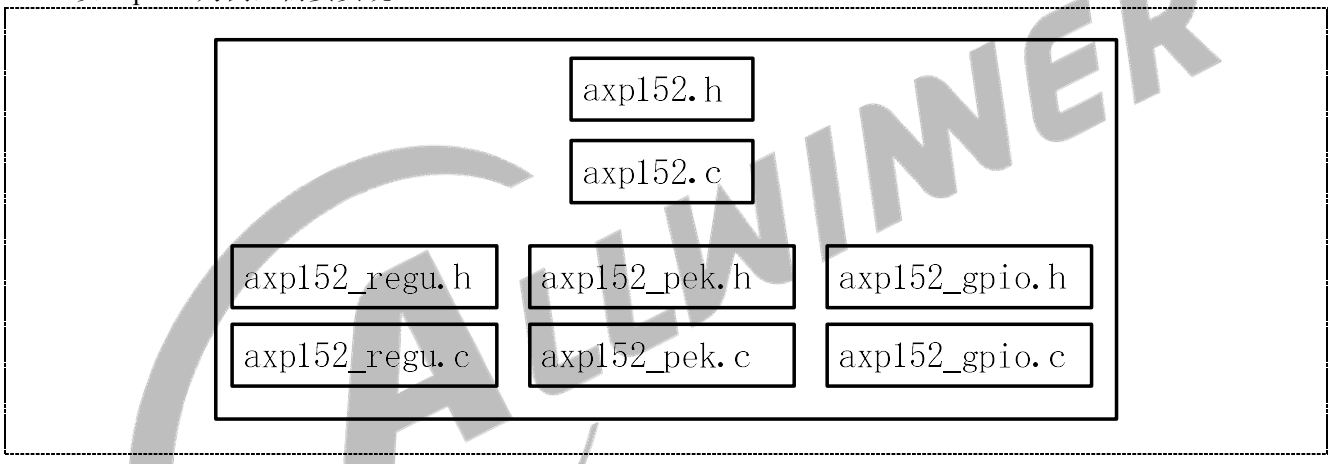

图 3-4 代码层次示例

3.2. irq domain

AXP 内部支持多个中断事件，但是作为外挂设备只能占用 cpu 的一路 irq，通常采用中断后查询的方式。这里在 AXP 父设备初始化时，向 axp core 层注册一个 irq domain，作为 MFD 设备的资源，子设备可以像申请一般 irq 一样申请 AXP 内部的 irq，由系统分发。

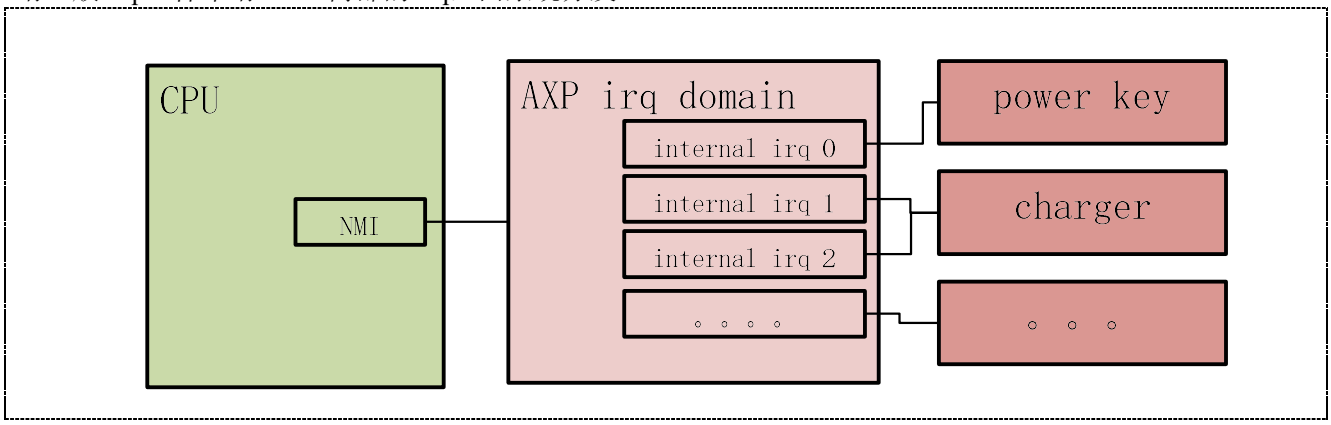

图 3-5 AXP IRQ domain

irq 由 axp irq domain 分发，类似于 gpio 的 irq

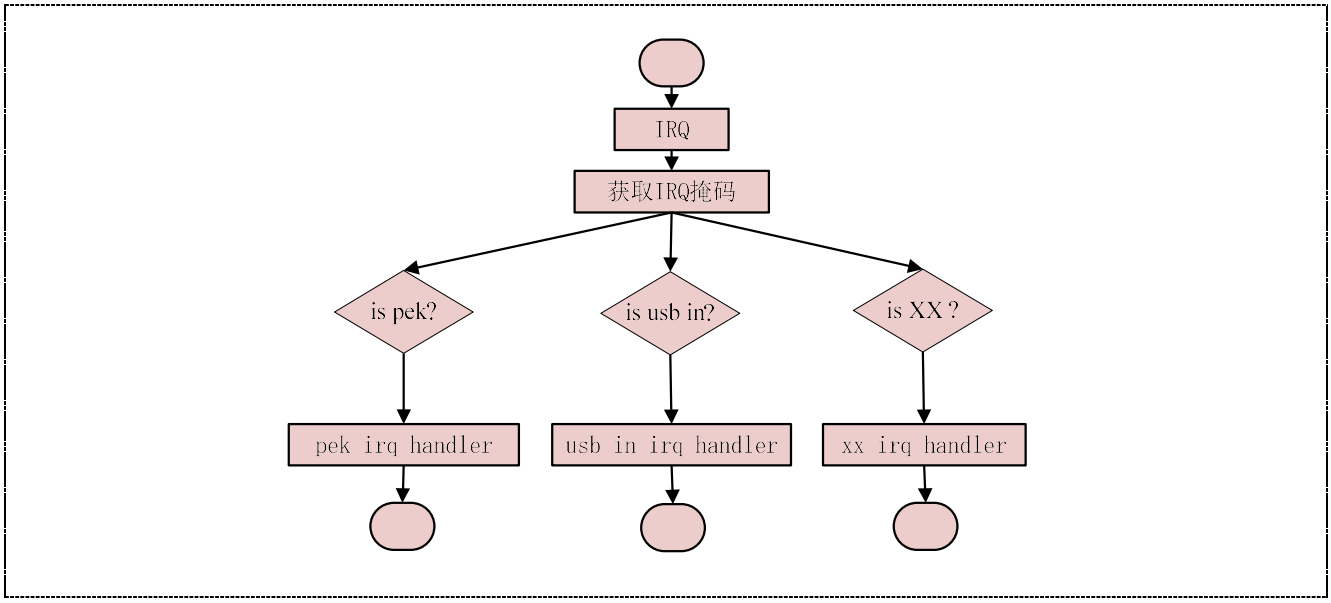

图 3-6 AXP IRQ 处理流程

3.3. axp regulator 层次

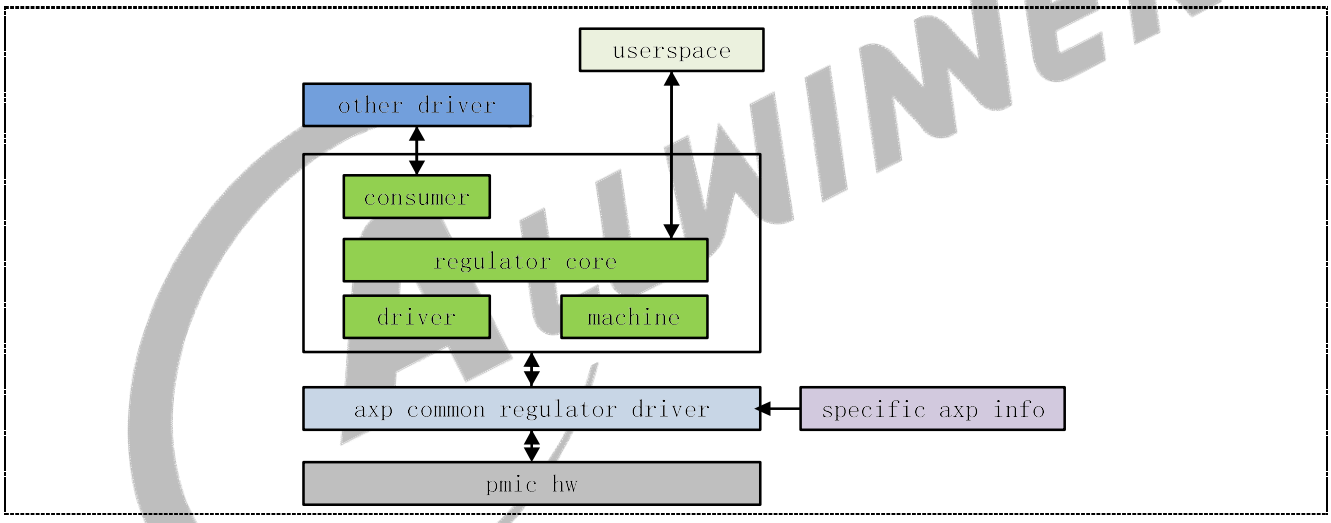

图 3-7 axp regulator 层次

3.4. axp charger

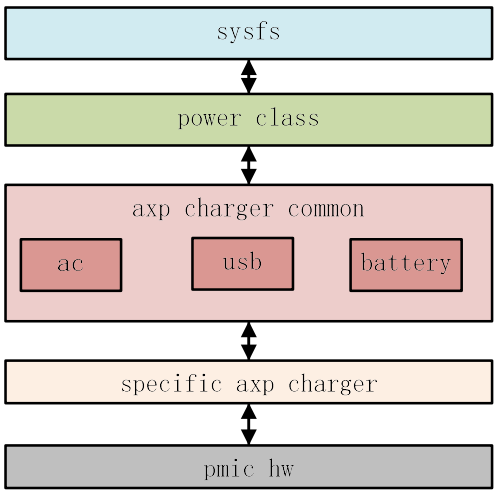

图 3-8 axp charger 层次

3.5. axp gpio 层次

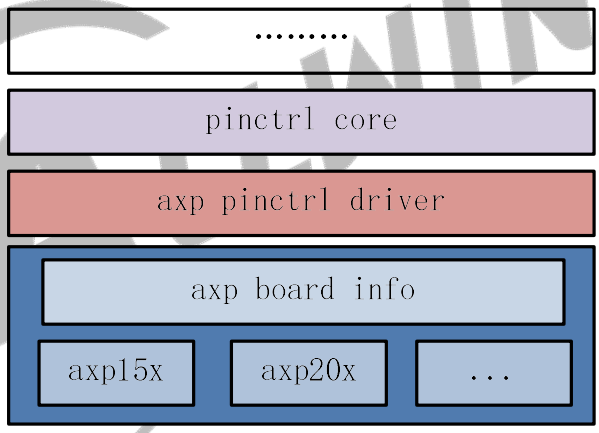

图 3-9 axp gpio 层次

4. 接口设计

4.1. axp_core 接口：

4.1.1. 数据结构

4.1.1.1.axp_regmap

regmap 结构体，供驱动访问设备。

```
struct axp_regmap {
    enum AXP_REGMAP_TYPE type;   //AXP_REGMAP_I2C or AXP_REGMAP_RSB
    struct i2c_client      *client;   //i2c client 句柄
    struct mutex           lock;
#ifndef CONFIG_AXP_TWI_USED
    spinlock_t             spinlock;
#endif
    u8                     rsbaddr;   //rsb 地址
};
```

4.1.1.2.axp_regmap_irq_chip

axp 中断硬件描述

```
struct axp_regmap_irq_chip {
    const char *name;
    unsigned int status_base;   //中断状态寄存器地址
    unsigned int mask_base;   //中断使能寄存器
    int num_regs;                 //中断寄存器个数
};
```

4.1.1.3.axp_irq_chip_data

axp 中断分发信息描述，一般不用关心

```
struct axp_irq_chip_data {
    struct mutex lock;
    struct axp_regmap *map;
    struct axp_regmap_irq_chip *chip;
    struct axp_regmap_irq *irqs;
    int num_irqs;
    u64 irqs_enabled;
    void (*wakeup_event)(void);
};
```

4.1.1.4.axp_dev

axp mfd 描述结构

```
struct axp_dev {
```

```
    struct device                   *dev;
    struct axp_regmap               *regmap;
    int                             nr_cells;   //子设备个数
    struct mfd_cell                 *cells;     //子设备 mfd cell
    struct axp_irq_chip_data        *irq_data;
    int                             irq;
    bool                            is_dummy;
    bool                            is_slave;
    struct list_head                list;
    int                             pmu_num;
};
```

4.1.2. axp_regmap_init_i2c

函数原型	struct axp_regmap *axp_regmap_init_i2c(struct device *dev)
参数说明	dev：mfd 父设备的 i2c 设备的 device 指针
返回说明	返回 axp_regmap 句柄
功能描述	初始化并获取 axp 设备的 axp_regmap 句柄以供访问硬件，用于 i2c 连接方式

4.1.3. axp_irq_chip_register

函数原型	struct axp_irq_chip_data *axp_irq_chip_register(struct axp_regmap *map, int irq_no, int irq_flags, struct axp_regmap_irq_chip *irq_chip, void (*wakeup_event)(void))
参数说明	map：是 axp_regmap 句柄， irq_no：是对主控的 irq 号， irq_flags：axp_regmap_irq_chip 句柄 irq_chip：中断信息描述，决断分发哪些中断、使能哪些中断 wakeup_event：上报唤醒事件的函数
返回说明	成功返回 0.否则返回相应错误号
功能描述	注册 axp irq chip，创建 axp irq domain

4.1.4. axp_irq_chip_unregister

函数原型	void axp_irq_chip_unregister(int irq, struct axp_irq_chip_data *irq_data)
参数说明	irq：axp 对主控的 irq 号， irq_data：axp_irq_chip_data 句柄
返回说明	无
功能描述	注销 axp irq chip

4.1.5. axp_mfd_add_devices

函数原型	int axp_mfd_add_devices(struct axp_dev *axp_dev)
参数说明	axp_dev：axp_dev 句柄。
返回说明	正确写入则返回 0，否则返回相应错误号。
功能描述	注册 axp mfd 设备，接口内部是对 mfd_add_devices 的封装

4.1.6. axp_mfd_remove_devices

函数原型	int axp_mfd_remove_devices(struct axp_dev *axp_dev)
参数说明	axp_dev：axp dev 句柄

返回说明	正确写入则返回 0，否则返回相应错误号
功能描述	注销 axp mfd 设备

4.1.7. axp_request_irq

函数原型	int axp_request_irq(struct axp_dev *adev, int irq_no, irq_handler_t handler, void *data)
参数说明	adev：为 axp dev 句柄， irq_no：为内部 irq 号， handler：为中断回调函数 data：传入中断回调函数的参数
返回说明	正确写入则返回 0，否则返回相应错误号
功能描述	子设备向父设备的 irq domain 申请 irq

4.1.8. axp_free_irq

函数原型	int axp_free_irq(struct axp_dev *adev, int irq_no)
参数说明	adev：为 axp dev 句柄， irq_no：为内部 irq 号
返回说明	正确写入则返回 0，否则返回相应错误号
功能描述	子设备向父设备的 irq domain 释放 irq

4.1.9. axp_regmap_write

函数原型	s32 axp_regmap_write(struct axp_regmap *map, s32 reg, u8 val)
参数说明	map：axp_regmap 句柄， reg：寄存器地址， val：写入的值
返回说明	正确写入则返回 0，否则返回相应错误号
功能描述	往寄存器 reg 写入 val

4.1.10. axp_regmap_writes

函数原型	s32 axp_regmap_writes(struct axp_regmap *map, s32 reg, s32 len, u8 *val)
参数说明	map：axp_regmap 句柄 reg：寄存器地址， len：要写入寄存器的数量 val：指针指向的数组是要写入的值
返回说明	正确写入则返回 0，否则返回相应错误号
功能描述	往寄存器及其他的寄存器连续写入 val 指针指向的数组的值

4.1.11. axp_regmap_read

函数原型	s32 axp_regmap_read(struct axp_regmap *map, s32 reg, u8 *val)
参数说明	map：axp_regmap 句柄 reg：寄存器地址， val：指针指向的数组是读入的值
返回说明	正确读出则返回 0，否则返回相应错误号
功能描述	读取寄存器 reg 的值并写入 val

4.1.12. axp_regmap_reads

函数原型	s32 axp_regmap_reads(struct axp_regmap *map, s32 reg, s32 len, u8 *val)
参数说明	map：axp_regmap 句柄 reg：寄存器地址， len：要读取的寄存器数量， val：指针指向的数组是读入的值
返回说明	正确读出则返回 0，否则返回相应错误号
功能描述	读取连续寄存器 reg 的值并写入 val 指向的地址中

4.1.13. axp_regmap_update

函数原型	s32 axp_regmap_update(struct axp_regmap *map, s32 reg, u8 val, u8 mask)
参数说明	map：axp_regmap 句柄 reg：寄存器地址， val：要存入 reg 的值， mask：要更新的位
返回说明	正确写入则返回 0，否则返回相应错误号
功能描述	更新 reg 寄存器里面的 mask 位的值

4.1.14. axp_regmap_set_bits

函数原型	s32 axp_regmap_set_bits(struct axp_regmap *map, s32 reg, u8 bit_mask)
参数说明	map：axp_regmap 句柄 reg：寄存器地址， bit_mask：是把相应的位置 1
返回说明	正确写入则返回 0，否则返回相应错误号
功能描述	把 reg 寄存器的 bit_mask 位置 1

4.1.15. axp_regmap_clr_bits

函数原型	s32 axp_regmap_clr_bits(struct axp_regmap *map, s32 reg, u8 bit_mask)
参数说明	map：axp_regmap 句柄 reg：寄存器地址， bit_mask：是把相应的位清 0
返回说明	正确写入则返回 0，否则返回相应错误号
功能描述	把 reg 寄存器的 bit_mask 位清 0

4.2. axp_gpio 接口

4.2.1. 数据结构

4.2.1.1.axp_gpio_ops

axp gpio 操作集合

```
struct axp_gpio_ops {
    int (*gpio_get_data)(struct axp_dev *axp_dev, int gpio);   // 状态获取
    int (*gpio_set_data)(struct axp_dev *axp_dev, int gpio, int value); //状态设置
    int (*pmx_set)(struct axp_dev *axp_dev, int gpio, int mux); // IO 状态设置
    int (*pmx_get)(struct axp_dev *axp_dev, int gpio);          //IO 状态获取
};
```

4.2.1.2.axp_desc_function、axp_desc_pin

axp pin 功能描述

```
struct axp_desc_function {
        const char *name;
        u8          muxval;
        u8          irq_valid;
};

struct axp_desc_pin {
        struct pinctrl_pin_desc     pin;
        struct axp_desc_function *functions;
};
```

4.2.1.3.axp_pinctrl

axp pinctrl 描述符

```
struct axp_pinctrl {
        struct device                   *dev;
        struct pinctrl_dev              *pctl_dev;
        struct gpio_chip                gpio_chip;
        struct axp_dev                  *axp_dev;
        struct axp_gpio_ops             *ops;
        struct axp_pinctrl_desc         *desc;
        struct axp_pinctrl_function     *functions;
        unsigned                        nfunctions;
        struct axp_pinctrl_group        *groups;
        unsigned                        ngroups;
};
```

4.2.2. axp_pinctrl_register

函数原型	struct axp_pinctrl *axp_pinctrl_register(struct device *dev, struct axp_dev *axp_dev, struct axp_pinctrl_desc *desc, struct axp_gpio_ops *ops)
参数说明	dev：device 句柄， axp_dev：axp_dev 句柄， desc：axp pinctrl 描述符句柄 ops：axp 的 gpio 操作集合句柄
返回说明	axp_pinctrl 句柄
功能描述	注册 axp pinctrl，向外部驱动提供 axp 的 pinctrl 功能，内部是 pinctrl 接口封装

4.2.3. axp_pinctrl_unregister

函数原型	int axp_pinctrl_unregister(struct axp_pinctrl *pctl)
参数说明	ptcl：axp pinctrl 描述符句柄
返回说明	成功返回 0，否则返回错误号
功能描述	**注销** axp pinctrl

4.3. axp_charger 接口

4.3.1. 数据结构

4.3.1.1.axp_config_info

axp 充电配置项信息

```
struct axp_config_info {
        u32 pmu_used;
        u32 pmu_id;
        u32 pmu_battery_rdc;
        u32 pmu_battery_cap;
        u32 pmu_batdeten;
        u32 pmu_chg_ic_temp;
        u32 pmu_runtime_chgcur;
        u32 pmu_suspend_chgcur;
        u32 pmu_shutdown_chgcur;
        u32 pmu_init_chgvol;
        u32 pmu_init_chgend_rate;
        u32 pmu_init_chg_enabled;
        u32 pmu_init_bc_en;
        u32 pmu_init_adc_freq;
        u32 pmu_init_adcts_freq;
        u32 pmu_init_chg_pretime;
        u32 pmu_init_chg_csttime;
        u32 pmu_batt_cap_correct;
        u32 pmu_chg_end_on_en;
        u32 ocv_coulumb_100;

        u32 pmu_bat_para1;
        u32 pmu_bat_para2;
        u32 pmu_bat_para3;
        u32 pmu_bat_para4;
        u32 pmu_bat_para5;
        u32 pmu_bat_para6;
        u32 pmu_bat_para7;
        u32 pmu_bat_para8;
        u32 pmu_bat_para9;
        u32 pmu_bat_para10;
        u32 pmu_bat_para11;
        u32 pmu_bat_para12;
        u32 pmu_bat_para13;
        u32 pmu_bat_para14;
        u32 pmu_bat_para15;
        u32 pmu_bat_para16;
        u32 pmu_bat_para17;
        u32 pmu_bat_para18;
        u32 pmu_bat_para19;
        u32 pmu_bat_para20;
        u32 pmu_bat_para21;
        u32 pmu_bat_para22;
        u32 pmu_bat_para23;
        u32 pmu_bat_para24;
        u32 pmu_bat_para25;
        u32 pmu_bat_para26;
```

```
u32 pmu_bat_para27;
u32 pmu_bat_para28;
u32 pmu_bat_para29;
u32 pmu_bat_para30;
u32 pmu_bat_para31;
u32 pmu_bat_para32;

u32 pmu_ac_vol;
u32 pmu_ac_cur;
u32 pmu_usbpc_vol;
u32 pmu_usbpc_cur;
u32 pmu_pwroff_vol;
u32 pmu_pwron_vol;
u32 pmu_powkey_off_time;
u32 pmu_powkey_off_en;
u32 pmu_powkey_off_delay_time;
u32 pmu_powkey_off_func;
u32 pmu_powkey_long_time;
u32 pmu_powkey_on_time;
u32 pmu_pwrok_time;
u32 pmu_pwrnoe_time;
u32 pmu_reset_shutdown_en;
u32 pmu_battery_warning_level1;
u32 pmu_battery_warning_level2;
u32 pmu_restvol_adjust_time;
u32 pmu_ocv_cou_adjust_time;
u32 pmu_chgled_func;
u32 pmu_chgled_type;
u32 pmu_vbusen_func;
u32 pmu_reset;
u32 pmu_irq_wakeup;
u32 pmu_hot_shutdown;
u32 pmu_inshort;
u32 power_start;
u32 pmu_as_slave;
u32 pmu_bat_unused;
u32 pmu_ocv_en;
u32 pmu_cou_en;
u32 pmu_update_min_time;

u32 pmu_bat_temp_enable;
u32 pmu_bat_charge_ltf;
u32 pmu_bat_charge_htf;
u32 pmu_bat_shutdown_ltf;
u32 pmu_bat_shutdown_htf;
u32 pmu_bat_temp_para1;
u32 pmu_bat_temp_para2;
u32 pmu_bat_temp_para3;
u32 pmu_bat_temp_para4;
u32 pmu_bat_temp_para5;
u32 pmu_bat_temp_para6;
u32 pmu_bat_temp_para7;
u32 pmu_bat_temp_para8;
u32 pmu_bat_temp_para9;
u32 pmu_bat_temp_para10;
u32 pmu_bat_temp_para11;
u32 pmu_bat_temp_para12;
```

```
        u32 pmu_bat_temp_para13;
        u32 pmu_bat_temp_para14;
        u32 pmu_bat_temp_para15;
        u32 pmu_bat_temp_para16;
};
```

4.3.1.2.axp_ac_info

ac 充电描述信息

```
struct axp_ac_info{
    int   det_bit;          //ac detect
    int   det_offset;       //ac detect 寄存器地址
    int   valid_bit;//ac valid  位
    int   valid_offset;
    int   in_short_bit;   //短接状态位
    int   in_short_offset;      //短接状态寄存器地址
    int   ac_vol;               //ac 充电电压
    int   ac_cur;               //ac  充电电流
    int (*get_ac_voltage)(struct axp_charger_dev *cdev);        //获取 ac  电压
    int (*get_ac_current)(struct axp_charger_dev *cdev);        //获取 ac  电流
    int (*set_ac_vhold)(struct axp_charger_dev *cdev, int vol); //设置 ac vhold
    int (*get_ac_vhold)(struct axp_charger_dev *cdev);                  //读取 ac vhold
    int (*set_ac_ihold)(struct axp_charger_dev *cdev, int cur);         //  设置 ac ihold
    int (*get_ac_ihold)(struct axp_charger_dev *cdev);                  //  读取 ac ihold
};
```

4.3.1.3.axp_usb_info

usb 充电描述信息，与 ac 描述类似

```
struct axp_usb_info{
    int det_bit;
    int det_offset;
    int valid_bit;
    int valid_offset;
    int det_unused;
    int usb_pc_vol;      //usb pc 充电电压
    int usb_pc_cur;      //usb pc  充电电流
    int usb_ad_vol;      //usb  充电器充电电压
    int usb_ad_cur;      // usb  充电器充电电流
    int (*get_usb_voltage)(struct axp_charger_dev *cdev);
    int (*get_usb_current)(struct axp_charger_dev *cdev);
    int (*set_usb_vhold)(struct axp_charger_dev *cdev, int vol);
    int (*get_usb_vhold)(struct axp_charger_dev *cdev);
    int (*set_usb_ihold)(struct axp_charger_dev *cdev, int cur);
    int (*get_usb_ihold)(struct axp_charger_dev *cdev);
};
```

4.3.1.4.axp_battery_info

电池充电信息

```
struct axp_battery_info{
        int acpresent_bit;
        int vbuspresent_bit;
        int pwrsrc_offset;
        int chgstat_bit;            //充电状态位
        int chgstat_offset;         //充电状态寄存器地址
        int bat_temp_offset;
        int det_bit;
        int det_offset;
        int det_valid_bit;
        int det_valid;
        int det_unused;
        int cur_direction_bit;      //电池电流方向位
        int cur_direction_offset; //电池电流方向寄存器地址
        int polling_delay;
        int runtime_chgcur;         //开机充电电流
        int suspend_chgcur;         //休眠充电电流
        int shutdown_chgcur         //关机充电电流
        int (*get_rest_cap)(struct axp_charger_dev *cdev);          //获取剩余电量
        int (*get_bat_health)(struct axp_charger_dev *cdev);   //获取电池健康状态
        int (*get_vbat)(struct axp_charger_dev *cdev);              //获取电池电压
        int (*get_ibat)(struct axp_charger_dev *cdev);              //获取电池电流
        int (*set_chg_cur)(struct axp_charger_dev *cdev, int cur);   //设置电池充电电流
        int (*set_chg_vol)(struct axp_charger_dev *cdev, int vol);   //设置电池充电电压
        int (*pre_time_set)(struct axp_charger_dev *cdev,int min);
        int (*pos_time_set)(struct axp_charger_dev *cdev,int min);
};
```

4.3.1.5.axp_supply_info

包含 axp_ac_info、axp_usb_info、axp_battery_info 句柄

```
struct axp_supply_info {
        struct axp_ac_info *ac;
        struct axp_usb_info *usb;
        struct axp_battery_info *batt;
};
```

4.3.1.6.axp_charger_dev

axp charger 设备描述

```
struct axp_charger_dev {
        struct power_supply *batt;
        struct power_supply *ac;
        struct power_supply *usb;
        struct power_supply_info *battery_info;
        struct axp_supply_info *spy_info;
        struct device *dev;
        struct axp_dev *chip;
        struct timer_list usb_status_timer;
        struct delayed_work work;
        struct delayed_work usbwork;
        unsigned int interval;
        struct mutex charger_lock;

        int rest_vol;
        int usb_vol;
        int usb_cur;
        int ac_vol;
        int ac_cur;
        int bat_vol;
        int bat_cur;
        int bat_discur;

        bool bat_det;
        bool ac_det;
        bool usb_det;
        bool ac_valid;
        bool usb_valid;
        bool ext_valid;
        bool in_short;
        bool charging;
        bool ac_charging;
        bool usb_pc_charging;
        bool usb_adapter_charging;
        bool bat_current_direction;

        int pmic_temp_offset;

        /*ic temperature*/
        s32 ic_temp;
        s32 bat_temp;

        struct axp_adc_res *adc;

        void (*private_debug)(struct axp_charger_dev *cdev);
};
```

4.3.2. axp_power_supply_register

函数原型	struct axp_charger_dev *axp_power_supply_register(struct device *dev, struct axp_dev *axp_dev, struct power_supply_info *battery_info, struct axp_supply_info *info)
参数说明	dev：设备句柄， axp_dev：axp_dev 句柄， battery_info：电池信息句柄， info：axp_supply_info 句柄
返回说明	成功返回 axp_charger_dev 句柄
功能描述	通过 axp charger 公共层向系统注册 power supply

4.3.3. axp_power_supply_unregister

函数原型	void axp_power_supply_unregister(struct axp_charger_dev *chg_dev)
参数说明	chg_dev：axp_charger_dev 句柄
返回说明	无
功能描述	通过 axp charger 公共层向系统注销 power supply

4.3.4. axp_change

函数原型	void axp_change(struct axp_charger_dev *chg_dev)
参数说明	chg_dev：axp_charger_dev 句柄
返回说明	无
功能描述	更新充电状态

4.3.5. axp_capchange

函数原型	void axp_capchange(struct axp_charger_dev *chg_dev)
参数说明	chg_dev：axp_charger_dev 句柄
返回说明	无
功能描述	更新电池电量状态

4.3.6. axp_usbac_in

函数原型	void axp_usbac_in(struct axp_charger_dev *chg_dev)
参数说明	chg_dev：axp_charger_dev 句柄
返回说明	无
功能描述	通知系统外部供电接入

4.3.7. axp_usbac_out

函数原型	void axp_usbac_out(struct axp_charger_dev *chg_dev)
参数说明	chg_dev：axp_charger_dev 句柄
返回说明	无
功能描述	通知系统外部供电移除

4.3.8. axp_charger_suspend

函数原型	void axp_charger_suspend(struct axp_charger_dev *chg_dev)
参数说明	chg_dev：axp_charger_dev 句柄
返回说明	无
功能描述	通知系统进入休眠

4.3.9. axp_charger_resume

函数原型	void axp_charger_resume(struct axp_charger_dev *chg_dev)
参数说明	chg_dev：axp_charger_dev 句柄
返回说明	无
功能描述	通知系统进入唤醒

4.3.10. axp_charger_shutdown

函数原型	void axp_charger_shutdown(struct axp_charger_dev *chg_dev)
参数说明	chg_dev：axp_charger_dev 句柄
返回说明	无
功能描述	通知系统进入关机

4.3.11. axp_charger_dt_parse

函数原型	int axp_charger_dt_parse(struct device_node *node, struct axp_config_info *axp_config)
参数说明	node：device tress 节点 axp_config：axp 配置信息
返回说明	成功返回 0
功能描述	解析 dts 节点属性

4.4. axp_regulator 接口

4.4.1. 数据结构

4.4.1.1.axp_regulator_info

axp regulator 配置描述符

```
struct axp_regulator_info {
        struct regulator_desc desc;
        struct axp_regmap *regmap;
        s32 min_uv;
        s32 max_uv;
        s32 enable_val;
        s32 disable_val;
        s32 step1_uv;
        s32 vol_reg;
        s32 vol_shift;
        s32 vol_nbits;
        s32 switch_uv;
        s32 step2_uv;
        s32 new_level_uv;
        s32 mode_reg;
        s32 mode_mask;
        s32 freq_reg;
        s32 dvm_enable_reg;
        s32 dvm_enable_bit;
        s32 dvm_finish_flag;
        s32 *vtable;
        s32 pmu_num;
};
```

4.4.2. axp_regulator_register

函数原型	struct regulator_dev *axp_regulator_register(struct device *dev, struct axp_regmap *regmap, struct regulator_init_data *init_data, struct axp_regulator_info *info)
参数说明	dev：device 句柄， regmap：axp regmap 句柄， init_data：regulator init data 句柄， info：axp regulator info 句柄
返回说明	regulator 句柄
功能描述	注册 axp regulator，内部封装了 regulator_register

4.4.3. axp_regulator_unregister

函数原型	void axp_regulator_unregister(struct regulator_dev *rdev)
参数说明	rdev：regulator 句柄
返回说明	无
功能描述	注销 axp regulator

4.4.4. regulator_get

函数原型	struct regulator *regulator_get(struct device *dev, const char *id)
参数说明	dev：device 句柄， id：为各路输出对应的设备 id 号
返回说明	regulator 句柄
功能描述	获得 id 号相应的 regulator 结构体

4.4.5. Regulator 使能操作函数

函数原型	int regulator_enable(struct regulator *regulator) [功能见①] int regulator_disable(struct regulator *regulator) [功能见②] int regulator_force_disable(struct regulator *regulator) [功能见③] int regulator_is_enabled(struct regulator *regulator) [功能见④]
参数说明	regulator：是获得的结构体指针
返回说明	若正确打开或者关闭则返回 0，否则返回相应错误号
功能描述	① 打开，每调用一次 use_count 加 1 ② 关闭，每调用一次 use_count 减 1，直至 use_count 为 0 才真正关闭 ③ 强制关闭,同时将 use_count 清 0 ④ 查询

4.4.6. regulator_set_voltage

函数原型	int regulator_set_voltage(struct regulator *regulator, int min_uV, int max_uV)
参数说明	regulator：就是获得的结构体指针， min_uV：regulator_set_voltage 里面 min_uV 是需要的最小电压， max_uV：可接受的最大电压(min_uV=<max_uV，调电压时，是调到>=min_uV 的最小电压上)
返回说明	若正确则返回 0，否则返回相应错误号
功能描述	设置 regulator 输出电压

4.4.7. regulator_get_voltage

函数原型	int regulator_get_voltage(struct regulator *regulator)
参数说明	regulator：是获得的结构体指针
返回说明	若正确则返回当前 regulator 电压值，单位 uV，否则返回相应错误号
功能描述	获取 regulator 输出电压

4.4.8. regulator_put

函数原型	void regulator_put(struct regulator *regulator)
参数说明	regulator：是获得的结构体指针
返回说明	无
功能描述	释放 regulator，退出相应输出控制

4.5. 外部 sysfs 节点

4.5.1. Regulator(标准)

AXP803 电源对应表：

原理图名称	节点名称	原理图名称	节点名称
DCDC1	regulator.1	DCDC2	regulator.2
DCDC3	regulator.3	DCDC4	regulator.4
DCDC5	regulator.5	DCDC6	regulator.6
DCDC7	regulator.7	RTC-VCC	regulator.8
ALDO1	regulator.9	ALDO2	regulator.10
ALDO3	regulator.11	DLDO1	regulator.12
DLDO2	regulator.13	DLDO3	regulator.14
DLDO4	regulator.15	ELDO1	regulator.16
ELDO2	regulator.17	ELDO3	regulator.18
FLDO1	regulator.19	FLDO2	regulator.20
GPIO0/LDO	regulator.21	GPIO1/LDO	regulator.22
SWOUT	regulator.23		

功能	属性	文件路径	设置值
各路输出名字	r	/sys/class/regulator/regulator.X/name	对应的输出的名字
各路输出最大电压	r	/sys/class/regulator/regulator.X/max_microvolts	对应的最大电压值，单位 uV
各路输出最小电压	r	/sys/class/regulator/regulator.X/min_microvolts	对应的最小电压值，单位 uV
各路输出实际电压	r	/sys/class/regulator/regulator.X/microvolts	对应的输出电压值，单位 uV
各路输出状态	r	/sys/class/regulator/regulator.X/state	对应的输出状态，enabled/disabled:开启\关闭
各路输出对应的设备个数	r	/sys/class/regulator/regulator.X/num_users	0\1\2\……

4.5.2. Virtual consumer(非标准)

功能	属性	文件路径	设置值
最小电压	rw	/sys/devices/platform/reg-803-cs-X/min_microvolts	对应输出的电压设定值，单位 uV
最大电压	rw	/sys/devices/platform/reg-803-cs-X/max_microvolts	X 对应输出的电压最高值，单位 uV

4.5.3. Power_supply(标准)

功能	属性	文件路径	设置值
电池剩余电量	r	/sys/class/power_supply/battery/capacity	百分比，0\1\2\……\100
电池电流大小	r	/sys/class/power_supply/battery/current_now	单位 uA
电池状况	r	/sys/class/power_supply/battery/health	Unknown 未知, "Good" 好, "Overheat"过温, "Dead"坏掉, "Over voltage"过压,"Unspecified failure" 错误, "Cold"冷
供电状态	r	/sys/class/power_supply/battery/online	0\1：未在供电\正在供电
电池存在	r	/sys/class/power_supply/battery/present	0\1：存在\不存在
电池当前状态	r	/sys/class/power_supply/battery/status	Unknown 未知, "Charging"正在充电, "Discharging"放电, "Not charging"未在充电, "Full"满
电池技术	r	/sys/class/power_supply/battery/technology	Unknown, "NiMH", "Li-ion", "Li-poly", "LiFe", "NiCd","LiMn"
电池供电剩余时间	r	/sys/class/power_supply/battery/time_to_empty_now	单位 min
电池充满所需时间	r	/sys/class/power_supply/battery/time_to_full_now	单位 min
设备类别	r	/sys/class/power_supply/battery/type	battery 电池,"Mains"火牛,"USB"USB
电池最大设计电压	r	/sys/class/power_supply/battery/voltage_max_design	单位 uV
电池最小设计电压	r	/sys/class/power_supply/battery/voltage_min_design	单位 uV
电池电压大小	r	/sys/class/power_supply/battery/voltage_now	单位 uV
电池容量	r	/sys/class/power_supply/battery/charge_full_design	单位 mAh
DC 是否接上	r	/sys/class/power_supply/ac/present	0\1：未插上\插上
DC 是否在使用	r	/sys/class/power_supply/ac/online	0\1：未使用\正在使用
设备类别	r	/sys/class/power_supply/ac/type	"battery"电池,"Mains"火牛,"USB"USB
DC 供电时的电流大小	r	/sys/class/power_supply/ac/current_now	单位 uA
DC 供电时的电压大小	r	/sys/class/power_supply/ac/voltage_now	单位 uV
USB 是否接上	r	/sys/class/power_supply/usb/present	0\1：插上\没插上
USB 是否在使用	r	/sys/class/power_supply/usb/online	0\1：未使用\正在使用
设备类别	r	/sys/class/power_supply/usb/type	"battery"电池,"Mains"火牛,"USB"USB
USB 供电时的电流大小	r	/sys/class/power_supply/usb/current_now	单位 uA
USB 供电时的电压大小	r	/sys/class/power_supply/usb/voltage_now	单位 uV

5. Regulator 使用示例

5.1. regulator 使用 demo

5.1.1. LDO/DCDC 电源对应表

AXP803 电源对应表：

输出名称	Regulator ID	节点名称
DCDC1	axp803_dcdc1	regulator.1
DCDC2	axp803_dcdc2	regulator.2
DCDC3	axp803_dcdc3	regulator.3
DCDC4	axp803_dcdc4	regulator.4
DCDC5	axp803_dcdc5	regulator.5
DCDC6	axp803_dcdc6	regulator.6
DCDC7	axp803_dcdc7	regulator.7
RTC-VCC	axp803_rtc	regulator.8
ALDO1	axp803_aldo1	regulator.9
ALDO2	axp803_aldo2	regulator.10
ALDO3	axp803_aldo3	regulator.11
DLDO1	axp803_dldo1	regulator.12
DLDO2	axp803_dldo2	regulator.13
DLDO3	axp803_dldo3	regulator.14
DLDO4	axp803_dldo4	regulator.15
ELDO1	axp803_eldo1	regulator.16
ELDO2	axp803_eldo2	regulator.17
ELDO3	axp803_eldo3	regulator.18
FLDO1	axp803_fldo1	Regulator.19
FLDO2	axp803_fldo2	Regulator.20
GPIO0/LDO	axp803_ldoio0	Regulator.21
GPIO1/LDO	axp803_ldoio1	Regulator.22
SWOUT	axp803_dc1sw	Regulator.23

5.1.2. 使用示例

以 DCDC1 为例，需要设置 DCDC1 最大输出电压值为 3.4V，需要设置目标电压值为 3V。

```
#include <linux/regulator/consumer.h>

struct regulator *regu= NULL;
    int ret = 0;
    regu= regulator_get(NULL, "axp803_dcdc1");
    if (IS_ERR(regu)) {
        pr_err("%s: some error happen, fail to get regulator \n", __func__);
        goto exit;
    }

    //set output voltage to 3V
    ret = regulator_set_voltage(regu, 3000000, 3400000);
    if (0 != ret) {
        pr_err("%s: some error happen, fail to set regulator voltage!\n", __func__);
        goto exit;
    }

    //enalbe regulator
    ret = regulator_enable(regu);
    if (0 != ret) {
        pr_err("%s: some error happen, fail to enable regulator!\n", __func__);
        goto exit;
    }

    //disalbe regulator
    ret = regulator_disable(regu);
    if (0 != ret) {
        pr_err("%s: some error happen, fail to disable regulator!\n", __func__);
        goto exit;
    }

    //put regulater, when module exit
    regulator_put(regu);
```

5.1.3. use_count 值查看

根据 3.1.1 节中找到对应 ALDO1 的节点名称，这里以 axp803 的 aldo1 为例，其节点名称为 regulator.9，则在/sys/class/regulator 目录下就有个 regulator.9 目录，regulator.9 目录下有个 num_users 节点，cat 此节点就可以获得当前 use_count 值。

```
cat /sys/class/regulator/regulator.9/num_users
```

5.1.4. dump 节点使用方法

在串口中，输入如下命令。

```
cat /sys/class/regulator/dump
```

会将系统所有 regulator 的使用信息打印出来。

```
[详见①]          [详见②]      [详见③]      [详见④]          [详见⑤]：...
 axp803_dc1sw :   disabled   0   1600000     supply_name:
 axp803_ldoio1 :  disabled   0   2800000     supply_name:
 axp803_ldoio0 :  disabled   0   3000000     supply_name:
 axp803_fldo2 :    enabled   0   1100000     supply_name:
 axp803_fldo1 :    disabled  0   1200000     supply_name:
 axp803_eldo3 :   disabled   0   1800000     supply_name:
 axp803_eldo2 :   disabled   0   1800000     supply_name:
 axp803_eldo1 :   enabled    0   1800000     supply_name:
 axp803_dldo4 :   disabled   0   3000000     supply_name:
 axp803_dldo3 :   disabled   0   2500000     supply_name:
 axp803_dldo2 :   disabled   0   3200000     supply_name:
 axp803_dldo1 :   enabled    0   3300000     supply_name:
 axp803_aldo3 :   enabled    0   3000000     supply_name:
 axp803_aldo2 :   enabled    0   1800000     supply_name:
 axp803_aldo1 :   disabled   0   3000000     supply_name:
 axp803_rtc :     enabled    0   3000000     supply_name:
 axp803_dcdc7 :   disabled   0   1000000     supply_name:
 axp803_dcdc6 :   enabled    0   1100000     supply_name:
 axp803_dcdc5 :   enabled    0   1500000     supply_name:
 axp803_dcdc4 :   disabled   0   1100000     supply_name:
 axp803_dcdc3 :   enabled    0   1100000     supply_name:
 axp803_dcdc2 :   enabled    0   1100000     supply_name:
 axp803_dcdc1 :   enabled    2   3000000     supply_name:   vcc-pc   vcc-emmc
```

详注：

① 第一列为 regulator 的 name。

② 第二列为 regulator 的 state，代表使能或不使能。

③ 第三列为 regulator 的 use_count，代表了此路 regulator 被 enable 的次数。

④ 第四列为 regulator 的电压值。

⑤ 第五列的 supply_name,会将系统中调用了此路 regulator 的所有 regulator id 打印出来。

5.2. GPIO 驱动应用

5.2.1. AXP GPIO 对应表

原理图名称	配置表名称	GPIO 组名称	AXP803 PIN No.	IO status
Axp803 GPIO0	power0	GPIO_AXP(0)	35	IO
Axp803 GPIO1	power1	GPIO_AXP(1)	34	IO
Axp803 CHGLED	power2	GPIO_AXP(2)	53	O
Axp803 N_VBUSEN	power3	GPIO_AXP(3)	51	O

5.2.2. AXP GPIO 使用示例

以 power2 CHGLED 作为 moto 的控制脚为例，电路如下图。

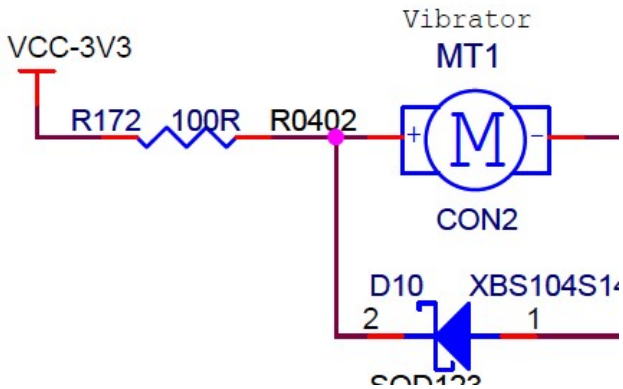

图 5-1 power3 电路图

MT-DRV-N 即连接到了 power2 CHGLED 脚，拉低则马达震动，拉高则关闭。

首先，在 sysconfig 中 moto 的配置如下：

```
;------------------------------------------------------------------------------
;motor configuration
;------------------------------------------------------------------------------
[motor_para]
motor_used                  = 1
motor_shake         = port:     power2<1><default><default><1>
                                (详见①)(详见②)          (详见③)
```

详注：

① 第一列，power2 为 CHGLED 脚的配置名称。

② 第二列，1 代表设置为输出，0 代表输入。

③ 第五列，1 代表默认输出高，0 代表默认输出低。

在 moto 驱动中，代码入下。

```
struct gpio_config motor_gpio;
int vibe_off;
script_item_u  val;
script_item_value_type_e    type;

/* 脚本解析 */
type = script_get_item("motor_para", "motor_shake", &val);
if(SCIRPT_ITEM_VALUE_TYPE_PIO != type) {
        printk(KERN_ERR "no motor_shake, ignore it!");
} else {
        motor_gpio = val.gpio;
        vibe_off = val.gpio.data;
}

/* 申请 GPIO，并设置为默认值 */
if (0 != motor_gpio.gpio) {
        if(0 != gpio_request(motor_gpio.gpio, "vibe")) {
                printk(KERN_ERR "ERROR: vibe Gpio_request is failed\n");
        }
        gpio_direction_output(motor_gpio.gpio, vibe_off);
}

/* 根据变量 on，进行输出高，输出低设置 */
if (0 != motor_gpio.gpio) {
        if(on) {
                __gpio_set_value(motor_gpio.gpio, !vibe_off);
        } else {
                __gpio_set_value(motor_gpio.gpio, vibe_off);
}

/* 模块退出时，释放 GPIO */
if (0 != motor_gpio.gpio) {
        gpio_free(motor_gpio.gpio);
}
```

5.3. Regulator shell 命令使用示例

AXP regulator 可以通过 shell 命令控制和设置其开关以及输出电压，各路文件节点建立在在 /sys/devices/platform 目录下，AXP803 分别为：

reg-803-cs-aldo1、reg-803-cs-aldo、reg-803-cs-aldo3、reg-803-cs-dc1sw、reg-803-cs-dcdc1、reg-803-cs-dcdc2、reg-803-cs-dcdc3、reg-803-cs-dcdc4、reg-803-cs-dcdc5、reg-803-cs-dcdc6、reg-803-cs-dcdc7、reg-803-cs-dldo1、reg-803-cs-dldo2、reg-803-cs-dldo3、reg-803-cs-dldo4、reg-803-cs-eldo1、reg-803-cs-eldo2、reg-803-cs-eldo3、reg-803-cs-fldo1、reg-803-cs-fldo2、reg-803-cs-gpio0ldo、reg-803-cs-gpio1ldo、reg-803-cs-rtc

以设置 AXP803 ALDO1 输出最大电压为 3.3V，设置目标电压为 3.0V 为例做说明。

```
//设置输出电压为 3.0V
echo    3300000 > /sys/devices/platform/reg-803-cs-aldo1/max_microvolts
echo    3000000 > /sys/devices/platform/reg-803-cs-aldo1/min_microvolts

//关闭输出
echo    3300000 > /sys/devices/platform/reg-803-cs-aldo1/max_microvolts
echo    3000000 > /sys/devices/platform/reg-803-cs-aldo1/min_microvolts
echo    0 > /sys/devices/platform/reg-803-cs-aldo1/min_microvolts
```

关闭输出，在设置完电压后，在将电压设成 0，就可以关闭此路 ldo/dcdc 的输出

5.4. AXP 读写接口 shell 命令使用示例

往 AXP803 寄存器 0f 写入值 0x55：

```
echo 0f55 > /sys/class/axp/axp_reg
```

读出 AXP803 寄存器 0f 的值：

```
echo 0f > /sys/class/axp/axp_reg
cat   /sys/class/axp/axp_reg
```

5.5. 打开/关闭 AXP driver debug 信息

```
echo 8 > /sys/class/axp/debug_mask   //打开充电 debug 信息
echo 0 > /sys/class/axp/debug_mask   //关闭 debug 信息
```

此开关，可打开关闭 AXP803 的充电控制信息的打印。

```
1：SPLY; 2:REGU; 4:INT; 8:CHG
```

6. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgment to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.