SC9820E 充电介绍

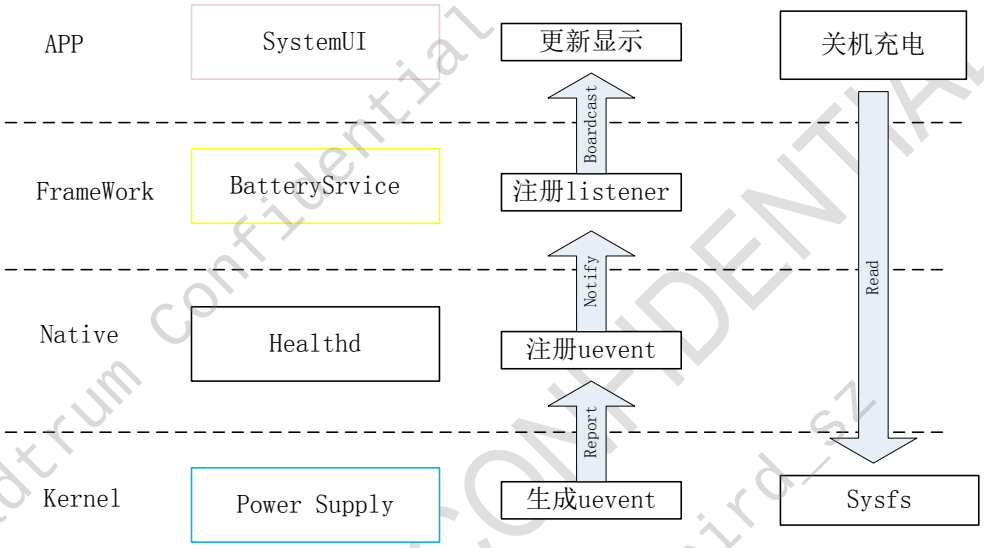
Document Number:		**Document Version:**	
Owner:		**Date:**	
Document Type:			
NOTE:	ALL MATERIALS INCLUDED HEREIN ARE COPYRIGHTED AND CONFIDENTIAL UNLESS OTHERWISE INDICATED. The information is intended only for the person or entity to which it is addressed and may contain confidential and/or privileged material. Any review, retransmission, dissemination, or other use of or taking of any action in reliance upon this information by persons or entities other than the intended recipient is prohibited. This document is subject to change without notice. Please verify that your company has the most recent specification. 		

www.spreadtrum.com

目 录

第1章 充电驱动代码框架

1.1 Android 电量整体框架



- Kernel 层

本层属于电池的驱动部分，负责与硬件进行交互，当电池电量信息发生变化时，生成相应的 uevent，上报给用户层。

- Healthd 守护进程

在 Android 中属于 Native 层，healthd 中运行一个系统服务，负责接收 Kernel 中上报的 uevent，对电池状态进行实时监控。

- BatteryService 系统服务

本层的系统服务 battery 使用 Java 代码写成，运行在 framework 的中 SystemServer 进程。该系统服务的主要作用是：监听电池信息变化消息，并将该消息以系统广播的形式转发至 Android 系统中各处。

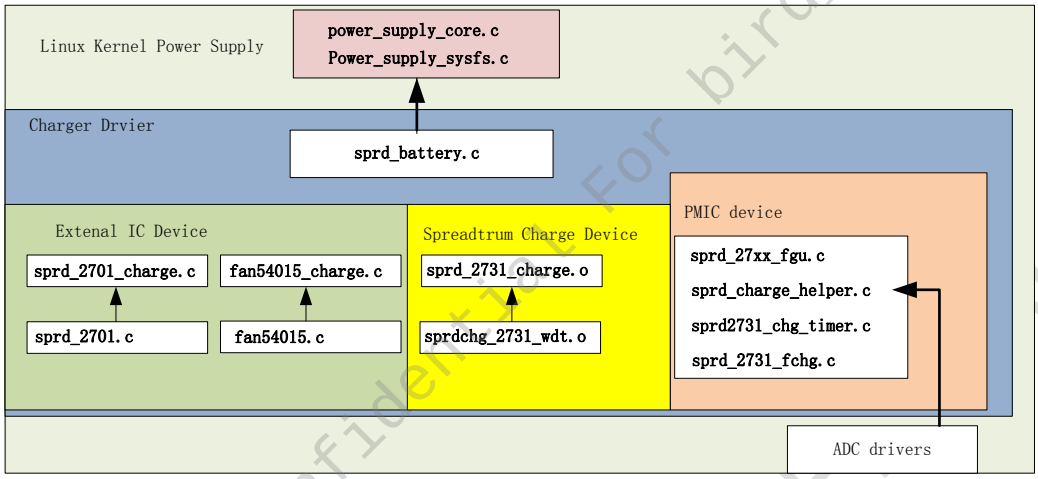
- SystemUI 应用

该部分属于电量上报的最后的环节。其主要工作是：监听系统广播并对 UI 作出相应更新。

- 关机充电

关机充电展讯单独启动的一个 linux 应用，通过系统调用直接读取 sysfs 来获取电池信息，init 进程会根据启动模式来启动 charge 服务，不会启动 android 相关进程。

1.2 驱动代码结构：



Sprd 充电方案使用的标准 linux power supply 框架。将外接充电、平台充电、电源管理作为单独设备驱动。

充电驱不再区分充电方案剥离了硬件细节仅实现电量跟跟状态管理，这样结构交之前的代码相比结构更加清晰。

Extenal IC device:

外置充电 IC 驱动

Fan54015.c、Bq25896.c、Sc2701.c等文件是以外置充电IC型号命名，文件实现了设备驱动注册及硬件层控制接口；

Fan54015_charge.c、Bq25896_charge.c、Sc2701_charge.c为充电驱动调用提供逻辑接口，相当于hal层，将充电功能注册，主要功能

```
struct sprd_ext_ic_operations {
    void (*ic_init)(struct sprdbat_drivier_data *);   //芯片初始化
    void (*charge_start_ext)(void);                   //开始充电
    void (*charge_stop_ext)(unsigned int);            //结束充电
    void (*set_charge_cur)(unsigned int);             //设置充电电流
    void (*set_input_cur_limit)(unsigned int);        //设置充电输出电流上线
    int (*get_charging_status)(void);                 //获取充电状态
    int (*get_charging_fault)(void);                  //获取 error 状态
    void (*timer_callback_ext)(void);                 //喂狗
    unsigned int (*get_charge_cur_ext)(void);         //获取当前充电电流
    void (*set_termina_cur_ext)(unsigned int);        //设置截止充电电流
    void (*set_termina_vol_ext)(unsigned int);        //设置截止充电电压
    void (*otg_charge_ext)(int);                      //OTG 输出控制
    int (*get_chgcur_step)(int, int);                 //获取充电电流步进
    void (*ext_register_notifier)(struct notifier_block *);
    void (*ext_unregster_notifier)(struct notifier_block *);
};
```

Spreadtrum Charge device

平台默认 pmic 线性充电

sprd_2723_charge.c、sprd_2721_charge.c 命名以 pmic 型号+charge 为规则，实现平台默认线性充电方案，文件将硬件实现和逻辑接口注册放在同一个文件中。

平台默认 pmic 开关充电

sprd_2731_charge.c 实现了平台的开关充电方案；

sprdchg_2731_wdt.c 实现了看门狗功能并作为驱动，为 2731 开关充电提供喂狗功能，防止芯片挂死后充电事故

Pmic Device

Pmic 驱动

sprd2723_charge_ helper.c、sprd2721_charge_ helper.c、sprd_charge_helper.c，这些文件以 pmic 型号命名，后缀添加 helper 主要是体现对充电驱动的辅助，没带型号的是默认 pmic 是 2731，文件提供跟充电相关的 adie 操作，比如读取 auxadc 电压、读取校准参数、充电器类型识别等；

sprd2723_chg_timer.c、sprd2721_chg_timer.c、sprd2731_chg_timer.c，充电需要唤醒查询充电状态，这几个以 timer 为结尾的文件是定时器驱动，用于设置定期唤醒系统查询充电状态，注册相关控制函数如下

NormalText Code

```
struct sprd_chg_timer_operations {
    void (*timer_enable)(unsigned int);
    void (*timer_disable)(void);
    int (*timer_request)(irq_handler_t, void *);
};
```

sprd_2731_fchg.c：快充驱动文件，软件仅 sc2731 支持快充，文件提供快充初始化及相关功能。

NormalText Code

```
struct sprd_fchg_operations {
    int (*fchg_init)(unsigned int);
    void (*fchg_deinit)(void);
};
```

FGU 驱动

Sprd_27xx_fgu.c：库仑计驱动，提供 FGU 相关操作接口，如电量统计，容量自学习等功能。其命名规则是 27xx 代表任何一个 pmic 都适用。

Charge Driver

充电驱动

sprd_battery.c 是充电驱动，这个是充电功能的核心内容，电量显示策略、温度检测策略、充电保护机制等功能在这里实现，功能实现与硬件细节剥离，调用通用接口实现逻辑控制。

1.3 编译配置说明：

Makefile：

```
obj-$(CONFIG_BATTERY_SPRD)  += sprd_battery.o
obj-$(CONFIG_CHARGER_SC2705)    += sc2705-charger.o sprd-sc2705-charger.o
obj-$(CONFIG_CHARGER_SC2723)            += sprd_2723_charge.o
obj-$(CONFIG_CHARGER_SC2721)            += sprd_2721_charge.o
obj-$(CONFIG_CHARGER_SC2731)    += sprd_2731_charge.o sprdchg_2731_wdt.o
obj-$(CONFIG_CHARGER_FAN54015)  += fan54015.o fan54015_charge.o
obj-$(CONFIG_CHARGER_SPRD2701)  += sprd_2701.o sprd_2701_charge.o
obj-$(CONFIG_FGU_SC27XX)                += sprd_27xx_fgu.o
obj-$(CONFIG_PMIC_SC2723)       += sprd2723_chg_timer.o sprd_charge_2723_helper.o
obj-$(CONFIG_PMIC_SC2731)       += sprd2731_chg_timer.o sprd_2731_fchg.o sprd_charge_helper.o
obj-$(CONFIG_PMIC_SC2721)       += sprd2721_chg_timer.o sprd_charge_2721_helper.o
obj-$(CONFIG_CHARGER_BQ25896)   += bq25896.o bq25896_charge.o
obj-$(CONFIG_CHARGER_SPRD2700) += sprd_2700.o sprd_2700_charge.o
```

编译开关控制分为 4 个部分

充电驱动、fgu 驱动、PMIC 驱动、充电 IC 驱动，用户配置需要打开四个宏即可；

1、无论适用哪种充电方案都需要打开

CONFIG_BATTERY_SPRD ：充电驱动

CONFIG_FGU_SC27XX：fgu 驱动

2、根据使用平台搭配的 pmic 选择使用：

CONFIG_PMIC_SC2731：搭配 2731

CONFIG_PMIC_SC2721：搭配 2721

CONFIG_PMIC_SC2723：搭配 2723

3、根据具体充电方案选择开关

CONFIG_CHARGER_FAN54015:搭配充电芯片 fan54015

CONFIG_CHARGER_SPRD2701: 搭配充电芯片 sc2701

CONFIG_CHARGER_SC2723: 搭配 SC2723 线性充电方案

CONFIG_CHARGER_SC2721: 搭配 SC2723 线性充电方案

CONFIG_CHARGER_SC2731: 搭配 SC2731 开关充电方案

举例：如果我们使用的是 SC2721 搭配 fan54015 的充电方案需要保证如下开关

CONFIG_BATTERY_SPRD = y

CONFIG_FGU_SC27XX = y

CONFIG_PMIC_SC2721 = y

CONFIG_CHARGER_FAN54015 = y

1.4 DTS 设备文件注册

1、 spxxxx_xxxx_native.dts

项目 board.dts，用于设备注册，充电驱动在这里进行注册。

如果使用外置 IC 开关充电：根据使用芯片驱动中配置的 name 在 dts 中加载。

NormalText Code

```
&i2c1 {             //对应 I2C 总线
        fan54015_chg: charger@6a {
                //设备名称
                compatible = "fairchild,fan54015_chg";
                reg = <0x6a>;
                chg-fault-gpios = <&ap_gpio 142 0>;
        };
};
```

如果使用 PMIC 线性充电或者开关充电：

NormalText Code

```
&pmic_charger {
        status = "okay";
};
```

2、 sc27xx.dtsi

电源管理配置文件，这里默认会将 pmic-charger 设置为关闭状态，使用的时候需要在 board.dts 中打开。

3、 sprd-battery.dtsi

用户参数配置，这里配置一般是通用参数无需修改，客制话的参数在 board.dts 中重新修正。

1.5 U-boot 充电

代码路径：u-boot/driver/power/battery/

1、使用 PMIC 充电默认开关

根据使用的 PMIC 型号进行区分选择编译如下文件

sprd_battery_2713.o

sprd_battery_2731.o

sprd_battery.o

2、 使用外置 IC 需要添加编译

如果用的外部充电 IC，uboot 里面需要实现一些简单的驱动，需要在 board.h 里面定义宏 CONFIG_SPRD_EXT_IC_POWER，当然 i2c 和控制外部 IC 代码编译的宏也是需要定义的，如：

#define CONFIG_SPRD_EXT_IC_POWER

#define CONFIG_SYS_I2C

#define CONFIG_SPRD_I2C

#define CONFIG_SPRD2701_CHARGE_IC

1.6 关机充电

代码路径：vendor/sprd/proprietories-source/charge/

第2章 主要功能介绍

2.1 低电量充电

电池电压低于 3V 软件是跑不起来的，硬件控制充电，我们这里仅介绍软件跑起来后的低电量充电。

为了保证系统稳定工作，软件不允许在 VBAT 电压过低情况下开机，uboot 中会等待 VBAT 电压高于 3.3V 以上才允许进入 kernel，如果充电过程中还不能达到开机电压则等待开机过程我们可以选择使用一张低电图片作为用户提示，图片读取 charge_logo 分区的 png 格式图片作为提示，下载的时候可以根据需要更换相应分区的图片资源。

Uboot15/common/cmd_cboot.c

```
    while(is_bat_low()) {
      if(charger_connected()) {
           debugf("cboot:low battery,charging...\n");
           sprdbat_show_chglogo();
           sprdbat_lowbat_chg();
           mdelay(SPRDBAT_CHG_POLLING_T);
      }else{
           debugf("cboot:low battery and shutdown\n");
           return CMD_POWER_DOWN_DEVICE;
```

Uboot 阶段为了提升开机速度，可以设置充电电流：

Uboot15/drivers/power/battery/sprd_battery.c

```
if (charger_connected()) {
    enum sprd_adapter_type adp_type = sprdchg_charger_is_adapter();
    if (adp_type == ADP_TYPE_CDP || adp_type == ADP_TYPE_DCP) {
        debugf("uboot adp AC\n");
        sprdchg_set_chg_cur(700);
    } else {
        debugf("uboot adp USB\n");
        sprdchg_set_chg_cur(450);
    }
}
```

一般充电激活时间不会超过 3 分钟，具体情况需要以硬件环境为准。

2.2 充电器类型识别

充电器插入检测：

充电器插入检测以 VCHG 电压上升到 3.5V 以上为准，VCHG 下降到 3.3V 以下充电器认为被拔出。

BC1.2 部分识别充电器类型：

PIMC 内置 BC1.2 完成充电器类型识别并将识别结果写入寄存器中，软件根据硬件识别结果进行类型区分，可以识别的充电器类型如下

NormalText Code

```
enum usb_charger_type {
  UNKNOWN_TYPE,
  SDP_TYPE,       //USB 充电
  DCP_TYPE,       //标准 AC 充电
  CDP_TYPE,       //标准 CDP 充电
  ACA_TYPE,
};
```

快充识别：

快充识别建立在 BC1.2 完成之后，软件需要打开快充功能，如果能与充电器握手成功则启动快充功能；

快充功能是可配置的，即使支持快充功能也可以通过应用层操作 sys/class/power_supply/ac/voltage_max 节点来修改为普通充电模式，如果小于等于 5V 则为普通充电。

设置充电电流

线性充电方案充电电流设置下限为 300mA，电流上限为 2300mA，300mA – 1400mA 设置最小步进单位为 50mA，1400mA – 2300mA 设置最小步进单位为 100mA。

```
void sprdchg_set_chg_cur(uint32_t chg_current)
{
        uint32_t temp;
        if (chg_current > SPRDBAT_CHG_CUR_LEVEL_MAX) {
                chg_current = SPRDBAT_CHG_CUR_LEVEL_MAX;
        }
        if (chg_current < SPRDBAT_CHG_CUR_LEVEL_MIN) {
                chg_current = SPRDBAT_CHG_CUR_LEVEL_MIN;
        }
        if (chg_current < 1400) {
                temp = ((chg_current - 300) / 50);
        } else {
                temp = ((chg_current - 1400) / 100);
                temp += 0x16;
        }
        sci_adi_clr(ANA_REG_GLB_CHGR_CTRL2, BIT_CHGR_CC_EN);
        sci_adi_write(ANA_REG_GLB_CHGR_CTRL1,
                        BITS_CHGR_CC_I(temp), BITS_CHGR_CC_I(~0));
        sci_adi_set(ANA_REG_GLB_CHGR_CTRL2, BIT_CHGR_CC_EN);
}
```

开关充电充电电流需要根据使用的 IC 不同区分设置。

2.3 快充功能

展讯快充支持高压快充，电压档位有三挡，9V、12V、20V。最大支持输入功率 15W，快充效率以实际使用的快充芯片为准，一般可以高于 80%；

SC2721\SC2731 均支持快充协议检测，如果需要快充功能要使用特定充电器，充电器需要支持 SPRD 快充检测，检测协议参考 spec。硬件需要搭配支持快充的开关充电芯片，平台默认支持 FAN54015、SC2701 开关充电芯片。

1、快充检测软件流程

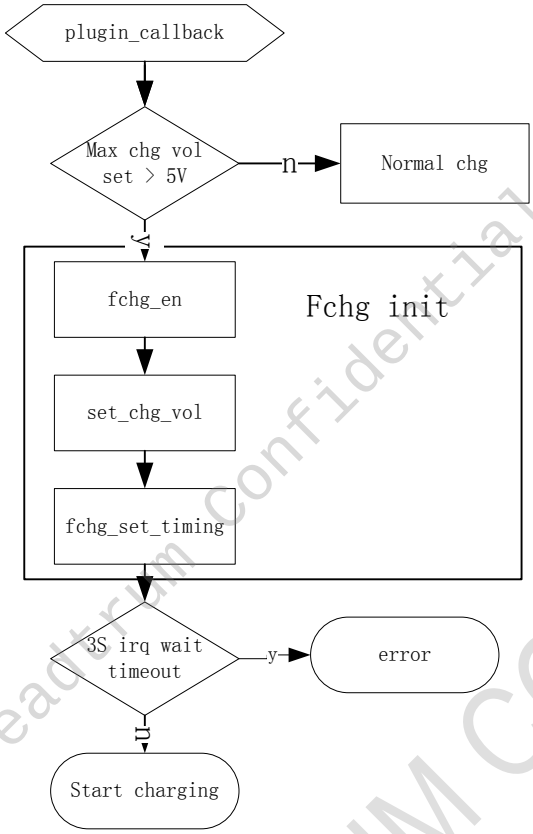
1、优先判断用户配置，如果用户允许快充需要将最大充电电压设置高于 5V

2、如果允许快充则打开快充并设置充电器的检测时序和充电电压，硬件会根据设置的时序与充电器握手并反馈握手状态。

3、初始化函数设置 3S 的超时时间，如果在这 3S 内充电器检测完成会上报中断，中断处理函数会将当前快充状态反馈并退出超时等待；

4、 设置充电电流，如果快充检测成功则按照用户设定的快充电流充电，否则按照普通充电电流充电；

2、流程图如下



2.4 电池容量自适应

1、功能介绍

当电池工作电流足够小的时候电池工作电压近似等于电池开路电压，如果有准确的电量表我们就能准确的定位电池容量我们记录为 C1，将这时库仑计计数值容量记录为 Q1。充电满后电池真实容量接近 100%，这里我们将库仑计计数值记录为 Q2。根据以上几个数据我们就可以得到电池真实有效容量:

Qmax = (Q2 – Q1) * 100/(100 –C1)

我们一般采用低电量开始作为初始采集点，并严格规定了环境温度即电流、电压条件，只有符合规定的容量自学习我们才认为是有效的自学习。自学习功能被抽象为 qmax，结构如下，各种限制条件需要在初始化的时候写入。

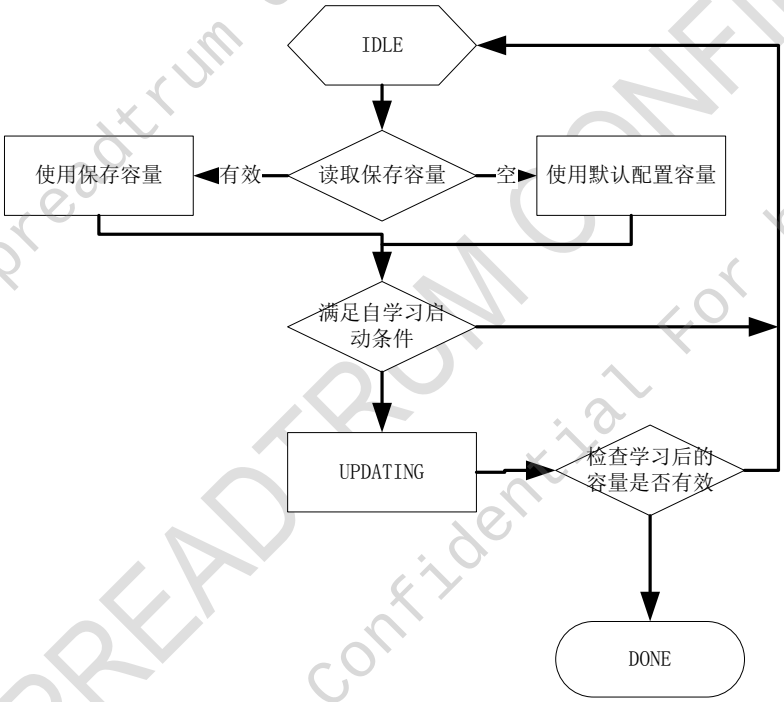
```
struct sprdfgu_qmax {
    int state;                  //自学习状态
    int cur_cnom;               //当前容量
    int s_clbcnt;               //开始自学习的库仑计计数
    int s_soc;                  //自学习初始点百分比
    int s_vol;                  //自学习初始电压限制
    int s_cur;                  //自学习初始电流限制
    int e_vol;                  //自学习结束电压条件
    int e_cur;                  //自学习结束电流条件
```

```
    int force_s_flag;               //应用层控制启动标志位
    int force_s_soc;                //应用层直接写入容量值
    uint32_t timeout;               //设置的超时门限
    __s64 s_time;                   //开始自学习的时间点
};
```

2、电量保存与回读

自学习状态 State 有 5 种，DONE 状态是学习完成状态，在 DONE 状态会将学习完成的容量值写入/productinfo/.battery_file 这个文件。productinfo 分区掉电不擦除，重启后再 INIT 状态会回读这个数据作为容量基数进行电量积分，并将自学习状态切换到 IDLE 状态。如果满足了自学习条件就进入了 QMAX_UPDATING，UPDATING 状态下会检查自学习后容量，如果容量检查失效则恢复 IDLE 状态，需要重新开始学习。如果手机单体没有完成过自学习则使用默认的容量配置。

```
enum QMAX_STATE {
    QMAX_INIT,      //开机默认状态
    QMAX_IDLE,      //自学习准备状态
    QMAX_UPDATING,  //电量需要更新
    QMAX_DONE,      //自学习完成
    QMAX_ERR,       //异常
};
```



3、用户配置

自学习方案采用轮询方式检测，用户也可以通过应用强制开始自学习，同时也可以修改自学习的限定条件，如下节点是呈献给用户层的信息。

```
sys/class/power_supply/sprdfgu/ qmax_force_start
应用层主动开启自学习功能
sys/class/power_supply/sprdfgu/ qmax_force_s_soc
应用层设置开启自学习时的容量
sys/class/power_supply/sprdfgu/ qmax_state
读取当前自学习状态
sys/class/power_supply/sprdfgu/ qmax_s_vol
设置开启自学习的电压条件
sys/class/power_supply/sprdfgu/ qmax_s_cur
```

```
设置开启自学习的电流条件
sys/class/power_supply/sprdfgu/ qmax_e_vol
设置自学习停止电压条件
sys/class/power_supply/sprdfgu/ qmax_e_cur
设置自学习停止电流条件
sys/class/power_supply/sprdfgu/ cnom
读取当前积分容量基数
sys/class/power_supply/sprdfgu/ saved_cnom
应用层直接写数保存容量
```

2.5 电池内阻自适应

电池开路电压 OCV 是表征电池容量的最关键因素，在电量显示、电池容量自学习的时候都可能用到电池开路电压来定位百分比。因此我们必须找到一个方法定位电池开路电压。

FGU 可以获取到电池工作状态下的电流 I 和电压 VBAT，根据欧姆定律只要我们得到电池内阻 R 就可以计算出电池开路电压了：OCV = VBAT – I *R。电池内阻一般都是动态变化的，不同容量不同温度都会有不同的电池内阻，所以需要一个动态调整内阻的算法来保证内阻的准确性，平台提供了一套电池内阻自适应的算法：提取硬件 buff 中的一组电流电压值（共 8 个）并将这 8 个数据按大小排序，找到最大的压降 DeltaV 和电流变化 DeltaI，根据欧姆定律 R =DeltaV/DeltaI；使用计算出来的 R 就能得到实时 OCV 了。

电池内阻自学习通过 sprdfgu_tracking 来记录学习状态，成员包括自学习时的温度、平均电流、平均电压、电流 buff、电压 buff、最小电流、最小电压等等。Chargework 会调用 static void sprdfgu_track(void)来更新 sprdfgu_data.track，使用的时候通过接口 static struct sprdfgu_tracking sprdfgu_get_track_data(void);获取到实时的电池工作状态信息 sprdfgu_data.track。

```
struct sprdfgu_tracking {
    __s64 r_time;
    __s64 query_time;           //结构更新时间间隔
    int rint;                   //电池内阻自适应出来的内阻
    int r_temp;                  //获取电池内阻时候判断采样温度
    int c_temp;                  //容量自适应的时候判断采样的温度
    int r_vol;                  //
    int relax_vol;              //最接近电池开路电压
    int relax_cur;              //最接近零点的电流
    int ave_vol;                //平均电压
    int ave_cur;                //平均电流
    int *cur_buff;              //电流 buff
    int *vol_buff;              //电压 buff
};
```

2.6 温度检测

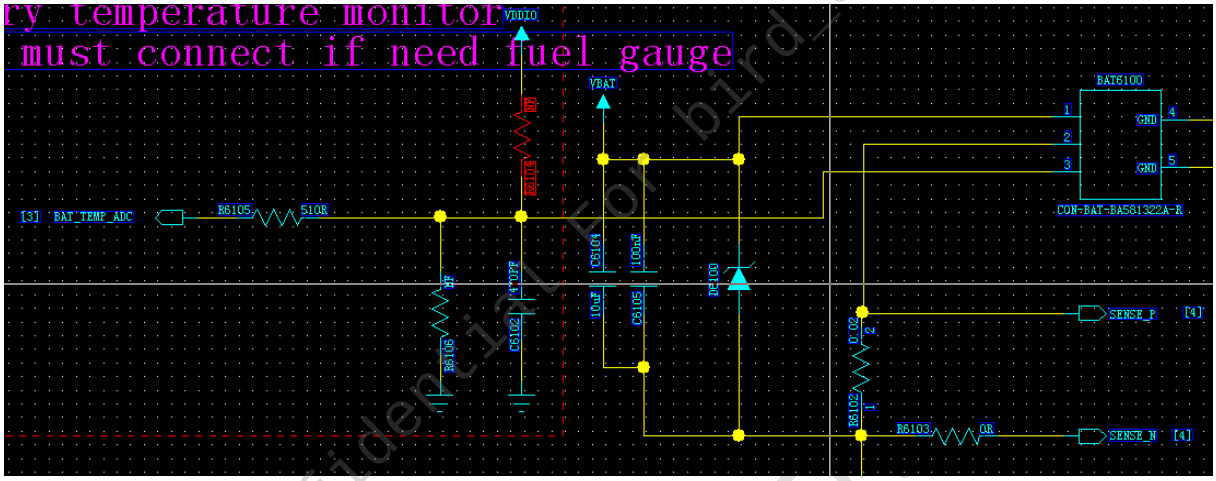
温度检测目的

高温和低温情况下对电池充电是有一定安全隐患，需要在安全温区进行充电操作。电池性能及寿命受温度影响很大，需要在异常温区停止电池放电。

static void sprdbat_temp_monitor(void)接口完成不同温度下充电控制逻辑

温度值读取方式

不同温度 NTC 的分压值查表的方式读取温度，硬件设计如下：



VDDIO 电压为 1800mv，通过 R6104 电阻可以计算出不同温度的分压值填入下表

NormalText Code

```
temp-tab-val = <1084 1072 1055 1033 1008 977  938 897 850 798 742 686
                                628 571 513 459 410 362 318>;
 temp-tab-temp = <750 800 850 900 950 1000 1050 1100 1150 1200 1250 1300
                                1350 1400 1450 1500 1550 1600 1650>;
```

精度补偿

一般来说现在电池都是自带 NTC 的。温度 ADC 读值是参考 GND 的，但是 NTC 连接的是电池负极，由于硬件设计上存在 FGU 电阻（电池负极对地有 20mohm），因此在有功耗或者充电的时候 20mohm 产生的压差会对温度产生影响。

芯片在读值的时候对 ADC 做了处理，内部直接将电池负极（sense_p）与 GND 连接，剥离了由功耗导致的读值跳变，代码如下在读取温度的时候需要设置下 AUXAD_CTL0;

NormalText Code

```
int sprdchg_read_temp_adc(void)
{
    int cnt_adc_swp = 2, sum_adc = 0, adc_value;

    if (pbat_data->temp_support) {
        while (cnt_adc_swp--) {
            glb_adi_write(AUXAD_CTL0,
                (cnt_adc_swp & 0x01) ? AUXAD_NTCAMP_EN : 0,
                AUXAD_NTCAMP_EN);
            /* udealy(50) to ensure write register AUXAD_CTL0 success */
            udelay(50);
            adc_value = sprdchg_read_ntc_vol(pbat_data->channel_temp);
            sum_adc += adc_value;
        }
        return sum_adc >> 1;
    }
    return 3000;
}
```

Sense_p 与电池连接器之间的阻抗是另外一个影响温度读值的因素，这里我们沿用以前温度补偿算法将其对 ADC 读值的影响做补偿以达到不同功耗切换的时候温度读取正常；

Temp_vol = vol - vol_comp + ((vol_comp *(vol - vol_comp))/(1800 - vol_comp));

Vol : Temp_adc_channel read value

vol_comp: 20mohm * current

1800: VDDIO

温度精度确认

完成温度表及补偿之后需要验证精度，不建议使用温箱直接验证，因为温箱精度不能保证，且电池工作状态下 NTC 处温度不一定和环境温度一样。

建议验证方法：使用滑动变阻器模拟 NTC 电阻，设置对应 NTC 温度的阻抗读取 log 中温度信息，看是否准确。

JEITA 功能

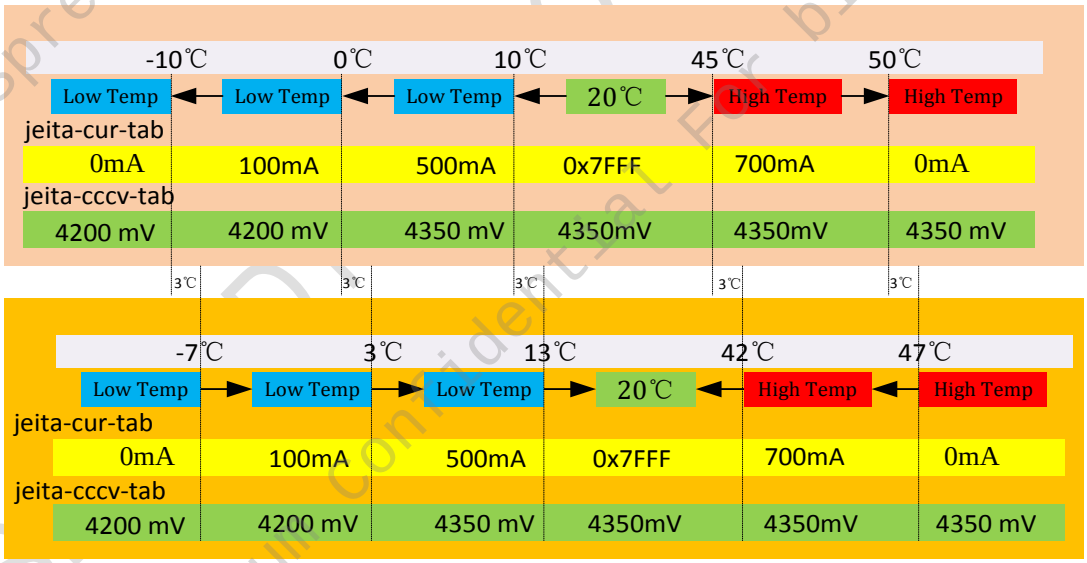
平台提供动态修正充电流及恒压电压的功能，可以根据当前温度来调整充电流及满电电压。以 NORMAL_TEMP 对应温度值（默认是 20℃）为原点 jeita-temp-tab 设置的对应温度点为调整点对充电电流及恒压电压做调整；jeita-temp-recovery-tab 设置的温度点为温区恢复点；温度变化到更高或者更低区间后如果恢复回来有 3℃的缓冲区，具体流程可参见下图。

Sprd-battery.dtsi

```
jeita-temp-tab = <900 1000 1100 1450 1500>;
jeita-temp-recovery-tab = <930 1030 1130 1420 1470>;
jeita-cur-tab = <0 100 500 0x7FFF 700 0>;
jeita-cccv-tab = <4200 4200 4350 4350 4350 4350>;
```

对应 jeit_cur-tab 中电流就是 0x7fff 即为最大值，软件中对最大值的处理就是默认充电电流；

以常温 20 度作为原点，温度上升和下降认为是温度从常温扩散对应的温度表就是 jeita-temp-tab，反之温度恢复为常温使用 jeita-temp-recovery-tab，每个区间对应的电流值和电压值在 jeita-cur-tab 和 jeita-cccv-tab；



注意：

温度区间因为软件中设置了默认零点 20 度所以 jeita-temp-tab 和 jeita-temp-recovery-tab 实际成员会增加一个常温参数以匹配 jeita 电流和电压表

2.7 电池在位检测

AUXADC_CH0 复用检测电池在位信息，软件规定电池不在位情况下不允许充电的。Uboot 启动过程如果电池不在位停止充电后 VBAT 会被迅速拉低到关机电压，硬件掉电。

Uboot15/driver/power/sprd_battery.c

```
if (!sprdbat_is_battery_connected()) {
                printf("battery unconnected shutdown charge!!!!!\n");
                sprdchg_stop_charge();
}
```

Kernel 部分电池在位处理

电池在位检测以中断形式注册，对应中断控制器 eic 9；

battery-det-gpios = <&pmic_eic 9 0>;

在不支持路径管理的方案中充电器在位状态下电池拔出后是不允许系统继续工作的，通过停止充电来实现系统高负载状态下硬件下电，BATTERY_DET 触发门限是 1.2V，软件不可配置。

Kernel/driver/power/sprd_battery.c

```
static void sprdbat_vbat_detect_irq_works(struct work_struct *work)
{
    int value;

    value = gpio_get_value(sprdbat_data->gpio_vbat_detect);
    SPRDBAT_DEBUG("bat_detect value:0x%x\n", value);
    mutex_lock(&sprdbat_data->lock);
    if (value) {
        if (!sprdbat_data->bat_info.bat_present) {
            sprdbat_data->bat_info.bat_present = 1;
            sprdbat_change_module_state
                (SPRDBAT_CHG_UNSPEC_RESTART_E);
            if (POWER_SUPPLY_STATUS_DISCHARGING !=
                sprdbat_data->bat_info.module_state)
                sprdbat_data->start_charge();
            SPRDBAT_DEBUG("vbat_detect-start_charge!!!!\n");
        }
        irq_set_irq_type(sprdbat_data->irq_vbat_detect,
            IRQ_TYPE_LEVEL_LOW);
    } else {
        if (sprdbat_data->bat_info.bat_present) {
            sprdbat_data->bat_info.bat_present = 0;
            sprdbat_change_module_state(SPRDBAT_CHG_UNSPEC_E);
            sprdbat_data->stop_charge();
            SPRDBAT_DEBUG("vbat_detect-stop_charge!!!!\n");
        }
        irq_set_irq_type(sprdbat_data->irq_vbat_detect,
            IRQ_TYPE_LEVEL_HIGH);
    }
    enable_irq(sprdbat_data->irq_vbat_detect);
    mutex_unlock(&sprdbat_data->lock);
}
```

2.8 电池兼容

电池 ID 识别：默认方案兼容两种电池，一种是有 NTC 的，一种是无 NTC 的，通过 sprdbat_get_bat_id 方法读取 NTC 分压值实现。

NormalText Code

```
int __weak sprdbat_get_bat_id(struct platform_device *pdev)
```

```
{
    struct iio_channel *channel_temp;
    int ntc_vol, ret = 0;

    channel_temp = iio_channel_get(&pdev->dev, "adc_temp");
    if (IS_ERR(channel_temp))
        return PTR_ERR(channel_temp);
    ntc_vol = sprdchg_read_ntc_vol(channel_temp);
    SPRDBAT_DEBUG("%s ntc_vol:%d\n", __func__, ntc_vol);

    /* if ntc voltage value is over 50mv, it is battery 1 */
    if (ntc_vol < BAT_NTC_THRESHOLD)
        ret = 1;
    else
        ret = 0;
    iio_channel_release(channel_temp);
    SPRDBAT_DEBUG("%s bat id:%d\n", __func__, ret);
    return ret;
}

static int (*sprdbat_bat_fun[4])(struct platform_device *pdev) = {
    sprdbat_get_bat_id,
};
```

设备注册的时候解析 sprd-battery.dtsi 中的子节点 battery 作为当前的电池参数，如果需要添加新的电池参数就要添加新的子节点名称，参考如下

```
if (!of_property_read_u32(np, "battery-adapt-fun",
        &fun_num) &&
        !of_property_read_u32(np, "battery-adapt-support",
        &bat_adapt)) {
        if (bat_adapt)
            bat_id = sprdbat_bat_fun[fun_num](pdev);
    }

    if (bat_id) {
        struct device_node *child;

        for_each_child_of_node(np, child) {
            u32 r;

            if (!child->name || of_node_cmp(child->name, "battery"))  //轮询子节点名称
                continue;

            if (of_property_read_u32(child, "reg", &r) < 0)
                continue;

            if (r == bat_id) {
                np = child;          //np 指向子节点
                break;
            }
        }
        SPRDBAT_DEBUG("Sub-nodes: battery@%d %s.\n", bat_id,
                np == child ? "enabled" : "failed");
    }
```

2.9 充电状态切换

充电状态控制接口：sprdbat_change_module_state

充电状态集合如下，每当充电状态发生变化跟随着设置相应的标志位，并修改电池状态跟随

uevent 上报给应用，应用根据电池 health 来选择 ui 显示相关内容。

NormalText Code

```
enum sprdbat_event {
    SPRDBAT_ADP_PLUGIN_E,               //充电器插入
    SPRDBAT_ADP_PLUGOUT_E,              //充电器拔出
    SPRDBAT_OVI_STOP_E,                 //充电器电压过高
    SPRDBAT_OVI_RESTART_E,              //充电器过压后恢复
    SPRDBAT_OTP_COLD_STOP_E,            //电池温度过低
    SPRDBAT_OTP_OVERHEAT_STOP_E,        //电池温度过高
    SPRDBAT_OTP_COLD_RESTART_E,         //电池温度从低温恢复
    SPRDBAT_OTP_OVERHEAT_RESTART_E,     //电池温度从高温恢复
    SPRDBAT_CHG_FULL_E,                 //充满电
    SPRDBAT_RECHARGE_E,                 //满电后复充
    SPRDBAT_CHG_TIMEOUT_E,              //充电超时
    SPRDBAT_CHG_TIMEOUT_RESTART_E,      //超时后重新启动充电
    SPRDBAT_VBAT_OVP_E,                 //电池过压
    SPRDBAT_VBAT_OVP_RESTART_E,         //电池过压恢复
    SPRDBAT_CHG_UNSPEC_E,               //电池拔出
    SPRDBAT_CHG_UNSPEC_RESTART_E,       //电池拔出后重新插入
    SPRDBAT_FULL_TO_CHARGING_E,         //满电强制启动充电
    SPRDBAT_CHARGING_TO_FULL_E,         //充电强制显示 100
    SPRDBAT_CHG_FORCE_STOP_E,           //强制启动充电
    SPRDBAT_CHG_FORCE_START_E,          //强制关闭充电
};
```

2.10 充电器过压保护

static void sprdbat_chg_ovp_monitor(void)

充电器过压采用轮询方式轮询用户配置电压参数修改充电状态，快充电压如果下降 2000mv 则退出快充，重新设定充电电流。

```
ovp-stop = <6500>;          //充电过压保护
ovp-restart = <5800>;       //过压恢复电压
fchg-ovp-stop = <11000>;    //快充过压电压
fchg-ovp-restart = <10000>; //快充过压恢复电压
```

2.11 复充检测

static void sprdbat_chg_rechg_monitor(void)

充电满后充电截止，但是系统持续耗电，如果判断开路电压降低到 rechg-vol = <4131>; 以下需要重新启动充电。

2.12 充电满电检测

软件通过读取硬件状态来判断当前是什么充电阶段。读取有两种模式：

1、 通过电流电压组合判断

满足用户配置截止充电电流和电压后可以结束充电，这种方式是通用的，如果使用充电方案是线性充电方案则只能用这个方法判断是否满电。如果使用 I2C 控制的外接充电 IC 需要对芯片作适当的配置，将其自身停充判断 disable 掉。

2、 通过状态寄存器状态判断

外接 IC 需要设置截止条件，并通过读取状态寄存器来判断是否满足满电条件。

NormalText Code

```
int sprdchg_fan54015_get_charge_status(void)
{
    unsigned char chg_status = fan54015_get_chg_status();

    switch (chg_status) {
    case CHG_READY:
        SPRDCHG_DEBUG("fan54015 charge ready\n");
        return POWER_SUPPLY_STATUS_NOT_CHARGING;
    case CHG_CHGING:
        SPRDCHG_DEBUG("fan54015 is charging\n");
        return POWER_SUPPLY_STATUS_CHARGING;
    case CHG_DONE:
        SPRDCHG_DEBUG("fan54015 charge full\n");
        return POWER_SUPPLY_STATUS_FULL;
    default:
        SPRDCHG_DEBUG("fan54015 charge fault\n");
        return POWER_SUPPLY_STATUS_DISCHARGING;
    }
}
```

2.13 充电超时检测

充电超时后如果满足电量充足则状态设置为满电状态，如果电池电压过低则需要重新启动充电

NormalText Code

```
static void sprdbat_chg_timeout_monitor(void)
{
    SPRDBAT_DEBUG("sprdbat_chg_timeout_monitor enter\n");
    if (sprdbat_data->bat_info.chg_stop_flags &
        SPRDBAT_CHG_END_TIMEOUT_BIT) {
        SPRDBAT_DEBUG("sprdbat_chg_timeout_monitor recharge\n");
        sprdbat_change_module_state(SPRDBAT_CHG_TIMEOUT_RESTART_E);
        sprdbat_data->start_charge();
    }
    if (sprdbat_data->bat_info.chg_stop_flags == SPRDBAT_CHG_END_NONE_BIT) {
        if (sprdbat_is_chg_timeout()) {
            SPRDBAT_DEBUG
                ("sprdbat_chg_timeout_monitor chg timeout\n");
            if (sprdbat_data->bat_info.vbat_ocv >
                sprdbat_data->pdata->rechg_vol) {
                sprdbat_change_module_state(SPRDBAT_CHG_FULL_E);
                sprdbat_data->stop_charge();
            } else {
                sprdbat_data->bat_info.chg_this_timeout =
                    sprdbat_data->pdata->chg_rechg_timeout;
                sprdbat_change_module_state
                    (SPRDBAT_CHG_TIMEOUT_E);
                sprdbat_data->stop_charge();
            }
        }
    }
}
```

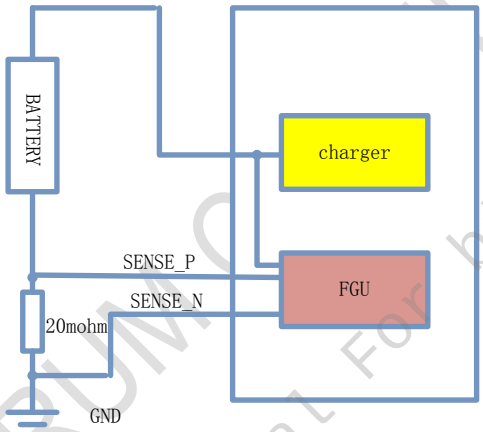
第3章 电量显示

3.1 电量统计原理

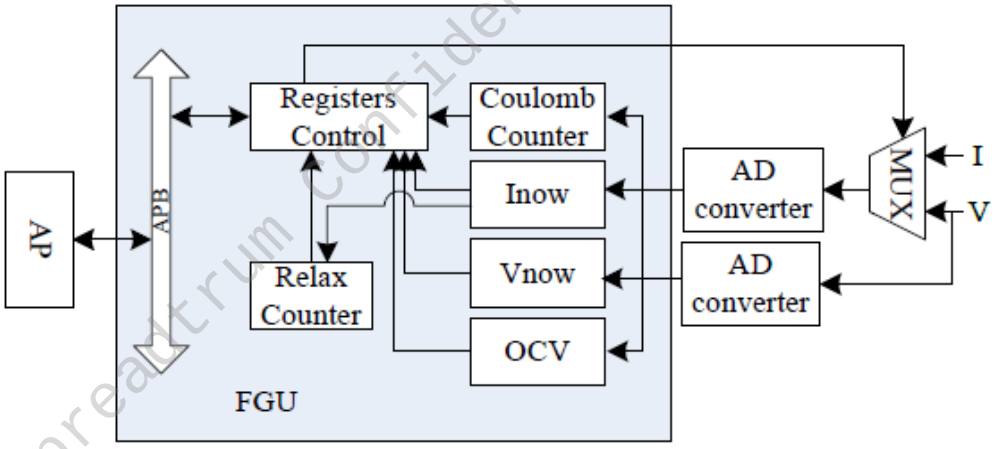
电量统计采用库仑积分形式对使用或者充入电量进行计算，并通过电压模式在百分比开始点和结束点对电池老化造成的积分不准确进行辅助校准，这样既能保证充放电百分比绝大部分的线性度又可以保证充放电的安全性，我们称为混合电量统计模式，电流经过积分可以计算出使用到库仑量，与初始设置的库仑总量比较即可得到变化的百分比。

3.2 FGU 介绍

FGU 依靠采样电池负极对地 20mhom 流过的电流积分来统计电量，电流采样精度为 500ms。



模块设计框图



- 电压检测：Vnow 寄存器可以对电压进行采样，采样频率为 500ms

sprdfgu_reg_get(REG_FGU_ OCV _VAL);

- 电流检测：Inow 寄存器可以对电流进行采样，采样频率为 500ms

sprdfgu_reg_get(REG_FGU_ CURT _VAL);

- 电流积分：Coulomb Counter 用于记录电流积分数据

sprdfgu_clbcnt_get(); 获取当前 Coulomb Counter 计数量下次轮询调用的时候可以计算出两次 Coulomb Counter 统计的差值从而得到库仑量变化 cc_delta。
DIV_ROUND_CLOSEST(cc_delta, (3600 * 2));
电流采样频率为 500ms，由此计算公式转化为单位 mah。
DIV_ROUND_CLOSEST(mah * 100, sprdfgu_data.pdata->cnom);
通过变化的 mah 计算出与总容量 cnom 的比值，从而得到变化百分比。

- Relax 模式：OCV 用于在 Relax 模式下采集电压，Relax 模式是硬件认为的低功耗模式低功耗门限电流值可配置，一般在硬件初始化的时候就配置好了，标准为 50ma。

sci_adi_write(REG_FGU_RELAX_CURT_THRE,
BITS_RELAX_CUR_THRE(sprdfgu_cur2adc_ma(sprdfgu_data.pdata->relax_current)),
BITS_RELAX_CUR_THRE(~0));

3.3 电量显示策略

初始电量

定义的初始容量有两个，一种是断电开机，另外一种是 reboot，这两种启动方式初始容量的获取方式也是不同的。

- 断电开机：

断电开机是客户端拿到单体后第一次开机或者使用过程中重新插入电池开机，在此之前 FGU 是没有统计过电量的，因此手机并不知道当前电池容量是多少，我们需要通过电池的电压来辅助定位电池初始容量。硬件提供一个 ADC 来采样电池开路电压，这一路 ADC 比较特殊，在系统启动很早的时候就可以工作了，系统功耗很小的时候采样的电压值近似等于电池开路电压，通过已经制作好的电池开路电压和容量关系对照表即可定位电池初始容量。

- Reboot

reboot 是指客户通过 watchdog 重启，和断电重启不一样芯片 PMU 不会断电。启动前电池电量是通过库仑计来计算的，手机启动的时候已经知道电池实际容量是多少了不需要重新定位初始容量，我们仅需要延续重启前的电量显示即可。平台提供一组 RTC 专门记录电池容量和启动模式，如果不下电 RTC 不会被清掉，判断是此类启动模式则直接读取 RTC 中保存的容量作为初始容量显示。

容量统计模式

FGU 通过统计流经电池的电流来计算电池剩余容量，内置库仑计实时统计流量，电量更新的时候需要读取统计的流量差值可以得到单位时间内的电流积分，通过设置好的总容量计算出变化的百分比并与初始容量相加刻算出当前电池实际容量并显示。

电量更新

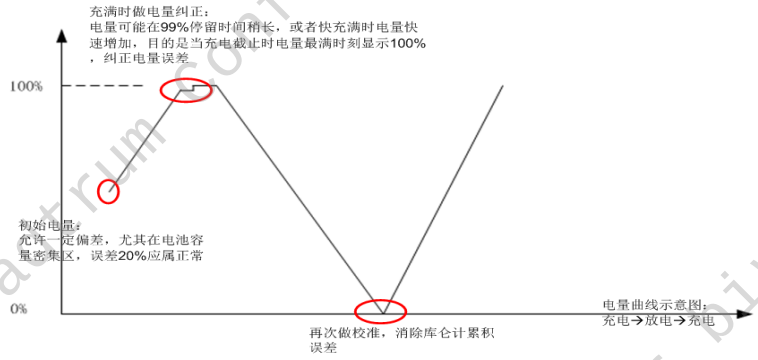
设置一个硬件 timer，在充电过程中定时唤醒系统来查询电量状态，放电过程中允许系统休眠，休眠过程中 FGU 模块会持续统计容量，电量更新仅在系统唤醒的瞬间执行。

电量加工

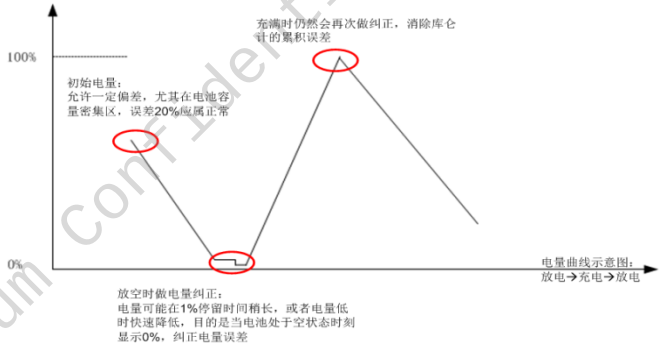
由于存在 ADC 精度、环境温度、硬件准确性等因素可能影响到电量显示，软件加入一些机制来保证电量显示与实际容量匹配。

- 电量显示不允许跳变：更新最短时间可配置，默认是 30S；
- 充电电量允许下降：如果充电入不敷出则电量显示会减少；

- 放电电量不允许上升：放电时电量无论如何不允许上涨；
- 满电策略：充电 99%维持时间可配置，默认 20 分钟，如果超时后显示满电后台会继续充电；
- 低电关机：为了保护硬件，软件设置最低工作门限电压，如果低于配置值会通知上层主动发起关机流程，默认配置 3100mv；
- 电量精度：电量统计精度为 0.1%；
- 通过两个 RTC 来记录电量：一个为实际容量，另外一个为显示容量，显示容量可能和实际容量不同，如果出现不同情况则显示的容量会以固定速度（30S）每 1%的速度追赶实际容量。
- 满电校准：充电的时候在高电量区比对电池实际容量和显示容量，如果两者不同则触发软件校准，显示容量会去匹配实际容量。
- 低电校准：放电的时候在低电量区比对电池实际容量和显示容量，如果两者不同则触发软件校准，显示容量会去匹配实际容量。



- 充电电流下降到指定值后充电结束，也就是这个是电池真正的满电点，如果充电结前电量统计已经超过 100，则在 99%等待，等待如果超时（时间可配置）则显示满电，后台继续充电。



- 如果充电完成了电量统计还没有到 100，则电量直接显示 100%。
- 当电量显示 100%后拔出充电器且电池容量还没有下降到 99%重新插入充电器，充电重新启动但是电量会维持 100%。

第4章 关机充电

4.1 关机充电介绍

关机状态下的手机插入 vbus 线会启动并根据这个启动模式来判断进入了关机充电模式，init 进程会根据 charge 模式的启动来区分启动相关的服务，关机充电模式下不启动 android，平台提供一个服务来维护关机充电，代码位置在 vendor/sprd/proprietories-source/charge。

4.2 关机充电服务主要进程：

void *charge_thread(void *cookie);
实时获取电池电量、电池健康度、电池温度等电池相关信息

void *power_thread(void *cookie);
实时监控充电器在位信息，如果充电器拔出执行掉电操作

void *input_thread(void *write_fd);
实时监控 input 事件，维护背光及 LCD 显示

4.3 关机充电显示

关机充电显示是标准的 miniui 接口，显示资源可以根据我们指定的图片格式更换。

第5章 用户配置

5.1 Uboot 配置

打开编译相关开关

#define CONFIG_SPRD_EXT_IC_POWER
#define CONFIG_FAN54015_CHARGE_IC
#define CONFIG_SYS_I2C
#define CONFIG_SPRD_I2C
#define CONFIG_SPRDCHG_I2C_BUS 1

配置 I2C 地址

Fan54015.c
#define SLAVE_ADDR (0x6a)

开机门限配置

充电器不在位开机电压
#define LOW_BAT_VOL 3400
充电器在位开机电压
#define LOW_BAT_VOL_CHG 3300

5.2 Kernel 配置

编译说明

公共开关：
CONFIG_BATTERY_SPRD
CONFIG_FGU_SC27XX

电源管理芯片开关

SC2723：CONFIG_PMIC_SC2723
SC2731：CONFIG_PMIC_SC2731
SC2721：CONFIG_PMIC_SC2721

PMIC 自带开关充电

SC2721 线性充电方案：CONFIG_CHARGER_SC2721
SC2723 线性充电方案：CONFIG_CHARGER_SC2723
SC2731 开关充电方案：CONFIG_CHARGER_SC2731

外接充电芯片

BQ25896: CONFIG_CHARGER_BQ25896

FAN54015：CONFIG_CHARGER_FAN54015

SP2701：CONFIG_CHARGER_SPRD2701

以上开关是并列关系，可以根据硬件设计选择组合使用

用户基本参数配置

驱动配置

使用外接充电 IC

```
&i2c4 {
    status = "okay";
    clock-frequency = <400000>;
    fan54015_chg: charger@6a {
        compatible = "fairchild,fan54015_chg";
        reg = <0x6a>;
        chg-fault-gpios = <&ap_gpio 8 0>;
        vbus-det-gpios = <&pmic_eic 0 0>;
    };
};
```

使用 PMIC 自带充电方案

```
&pmic_charger{
    status = "okay";
};
```

其他用户配置项说明

```
{
    battery: battery {
        compatible = "sprd,sprd-battery";
        status = "disabled";
        battery-adapt-support = <0>;                    //电池兼容开关
        battery-adapt-fun = <0>;                        //电池兼容函数索引 0 即为电池 ID 识别
        chg-end-vol = <4200>;                           //恒流恒压点电压值
        chg-end-vol-check = <4190>;                     //截止充电电压条件
        chg-bat-safety-vol = <4280>;                    //电池过压保护电压值
        fchg-vol = <9000>;                              //快充电压
        rechg-vol = <4131>;                                     //复充 OCV
        adp-cdp-cur = <700>;                                    //cdp 类型充电器充电电流
        adp-dcp-cur = <700>;                                    //标准 AC 充电器充电电流
        adp-sdp-cur = <450>;                                    //标准 USB 充电电流
        adp-unknown-cur = <450>;                            //其他充电器充电电流
        adp-fchg-cur = <3000>;                              //快充充电电流
        adp-cdp-cur-limit = <1500>;                     //CDP 电流限流
        adp-dcp-cur-limit = <3000>;                     //DPC 电流限流
        adp-sdp-cur-limit = <500>;                      //SDP 电流限流
        adp-fchg-cur-limit = <2000>;                    //fchg 电流限流
        adp-unknown-cur-limit = <500>;                //其他充电电流
        ovp-stop = <6500>;                                      //充电器过压保护电压
        ovp-restart = <5800>;                                   //过压保护恢复电压
        fchg-ovp-stop = <11000>;                            //快充过压保护电压
        fchg-ovp-restart = <10000>;                     //快充过压保护恢复电压
        chg-timeout = <21600>;                              //充电超时时间
        chg-rechg-timeout = <5400>;                     //满电或者充电超时后修正超时时间
        chg-end-cur = <120>;                                    //充电满电流条件
        chg-polling-time = <15>;                            //充电 timer3 唤醒系统时间
        chg-polling-time-fast = <1>;                    //
        cap-one-per-time = <30>;                        //每个百分比变化最短时间
        /*0 vol and cur,1 status and cur,2 ext ic*/
        chg-full-condition = <0>;                       //满电判断条件电流电压或者外置 IC
        temp-support = <0>;                                     //温度检测开关
        /*30mohm,if temp-table-mode = <1>, use it*/
        temp-comp-res = <6>;
        temp-tab-val = <1052 953 858 762 671 584
```

```
                504 434 372 318 271 234
                198 165 140 119 104 88>;
        /* temperature + 1000,750 = 1000 + (-250)*/
        temp-tab-temp = <750 800 850 900 950 1000
                 1050 1100 1150 1200 1250 1300
                 1350 1400 1450 1500 1550 1600>;
        jeita-temp-tab = <900 1000 1100 1450 1500>;
        jeita-temp-recovery-tab = <930 1030 1130 1420 1470>;
        jeita-cur-tab = <0 100 500 0x7FFF 700 0>;
        jeita-cccv-tab = <4200 4200 4350 4350 4350 4350>;
        fgu-mode = <0>;                                          //库仑计模式
        alm-soc = <5>;
        alm-vol = <3450>;                                        //低电量校准电压门限
        soft-vbat-uvlo = <3050>;                                //软件最低工作电压
        rint = <250>;                                            //电池内阻
        cnom = <1900>;                                           //电池容量配置单位 mah
        rsense-real = <215>;                                     //fgu 对地电阻真实阻抗
        rsense-spec = <200>;                                     //fug 对地电阻理论值
        relax-current = <50>;                                    //进入 relax 模式下的电流值
        fgu-cal-ajust = <0>;
        ocv-tab-vol = <4185 4113 4066 4022 3983 3949 3917
                3889 3864 3835 3805 3787 3777 3773
                3770 3765 3752 3724 3680 3605 3400>;
        ocv-tab-cap = <100  95   90   85   80   75   70
                65   60   55   50   45   40   35
                 30   25   20   15   10   5    0>;
        cnom-temp-tab = <1020 1800  /*value = 1000 + temp*/
                1010 1300
                1000 1070
                990  1000>;
        rint-temp-tab = <1020 200   /*value = 1000 + temp*/
                1010 450
                1000 650
                990  1100>;
        };
};
```

第6章 Battery 相关 Log

Battery driver 会周期打印一些 debug 信息，这些 debug 信息对分析 battery 相关问题很有帮助，battery 相关的 log 通常是 sprdbat，sprdfgu 等开始：

- **状态查询： sprdbat: chg_log:|state**

[12164.621443] c0 sprdbat: chg_log:health:1,state:1,chg_s_time:1933

- **停止充电标志： stopflags:**

[12164.621462] c0 sprdbat: stopflags0x:0,temp:200

stopflags：充电停止标志，充电最重要的 log 信息，充电器插着时，只有为 0 才会充电，每一个 bit 位代表一种停止标志，详见代码中的宏定义，有很多种异常都是通过这个关键字发现的。

- **充电器类型： chg_log|adp_type:**

[12164.623316] c0 sprdbat: chg_log:chgcur_type:450,vchg:4959,adp_type:4

adp_type：当前充电器的类型，具体意味着哪种类型的充电器详见枚举定义。

- **当前时间： cur time:**

[44726.156763] c4 28104 sprdbat: cur time:44728

- **开始充电的时间：chg_s_time：**

[12164.621443] c0 sprdbat: chg_log:health:1,state:1,chg_s_time:1933

chg_s_time：开始充电的时间，用当前的时间（cur time）减去开始充电的时间可以得到本次充电持续时间。

- **当前充电电流的设置值:chgcur_type**

[12164.623316] c0 sprdbat: chg_log:chgcur_type:450,vchg:4959,adp_type:4

- **充电器电压:chgcur_type**

[12164.623316] c0 sprdbat: chg_log:chgcur_type:450,vchg:4959,adp_type:4

- **库仑计统计的电量值：fgu_cap**

[12187.040135] c2 sprdbat: fgu_cap: = 36,flush: = 301,period:=16

- **电量变化：sprdfgu:mAh**

[12187.039500] c2 sprdfgu: d cap -16,cnom 2780,g -436mAh(0.1mA),init_cc:10229170

显示百分比：sprdbat: vbat: | cap:

[12187.040177] c2 sprdbat: vbat:3793,ocv:3817,current:65,cap:36

cap：最终显示给 UI 的电量，大部分时间与 fgu_cap 是相等的，但是如果出现电量跳变的可能，两者很可能不相等。

- **AUX_ADC 检测的电压**

[44730.259972] c0 28216 sprdbat: state:4,auxbat:4350,temp:200,present:1

- **FGU 读取的 vbat 电压：sprdbat: vbat:**

[12187.040177] c2 sprdbat: vbat:3793,ocv:3817,current:65,cap:36

- **FGU 统计的流过电池电流：sprdbat: vbat:| current:**

[12187.040177] c2 sprdbat: vbat:3793,ocv:3817,current:65,cap:36

- **电池开路电压：sprdbat: vbat:|ocv**

[12187.040177] c2 sprdbat: vbat:3793,ocv:3817,current:65,cap:36

- **距离上次更新 UI 电量持续的时间：flush：**

[44730.258460] c0 28216 sprdbat: fgu_cap: = 98,flush: = 15,period:=15

period：电量轮询的周期，本例为 16S

- **开始充电：sprdbat_start_charge**

[45116.145241] c0　3458 sprdbat: sprdbat_start_charge,cur_temp=200

- **结束充电：sprdbat_stop_charge**

[46916.048402] c1 31027 sprdbat: sprdbat_stop_charge

- **Thermal 调整充电电流：set_cur_state stae**

[46916.048402] c1 31027 sprdbat: set_cur_state stae=1