Sprd\_2713\_power.c

rootfs\_initcall(**sprd\_battery\_init**);

static int \_\_init sprd\_battery\_init(void)

{

return platform\_driver\_register(&**sprdbat\_driver**);

}

static struct platform\_driver sprdbat\_driver = {

.probe = **sprdbat\_probe**,

.remove = sprdbat\_remove,

.suspend = sprdbat\_suspend,

.resume = sprdbat\_resume,

.driver = {

.name = "sprd-battery",

#ifdef CONFIG\_OF

.of\_match\_table = of\_match\_ptr(battery\_of\_match),

#endif

}

};

static int **sprdbat\_probe**(struct platform\_device \*pdev)

{

int ret = -ENODEV;

struct**sprdbat\_drivier\_data** \*data;

struct sprdbat\_drivier\_data {

struct **sprd\_battery\_platform\_data** \*pdata; 总线

struct **sprdbat\_info** bat\_info; 电池信息

struct mutex lock;

struct device \*dev; ？？？

struct power\_supply battery; 电源

struct power\_supply ac; 充电

struct power\_supply usb; USB供电

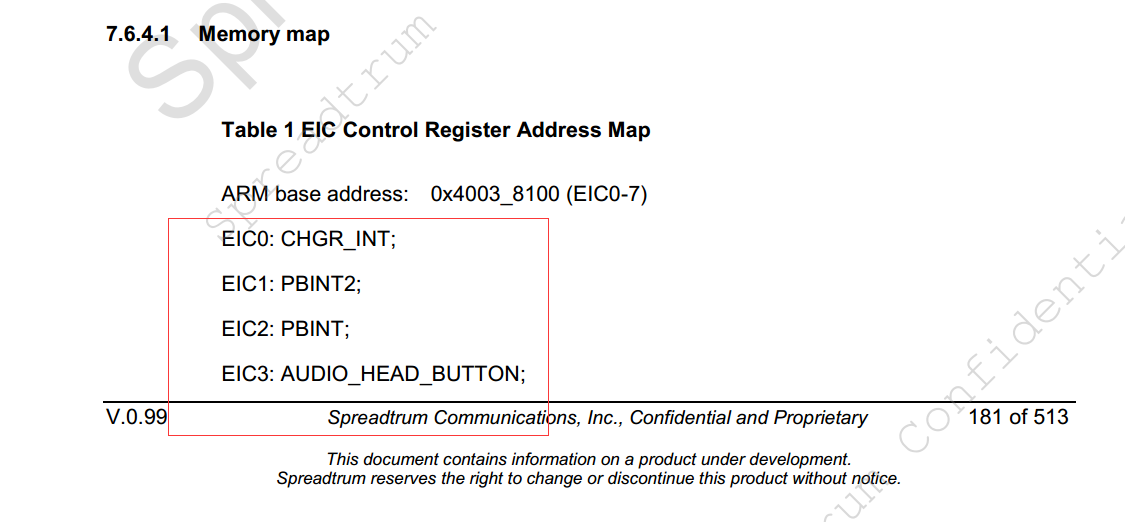
uint32\_t gpio\_charger\_detect; 接入电池 of\_get\_named\_gpio(np, "gpios", 0) 取得

uint32\_t gpio\_chg\_cv\_state; 充电状态 of\_get\_named\_gpio(np, "gpios", 1);

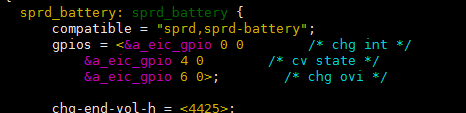
uint32\_t gpio\_vchg\_ovi; 充电过压of\_get\_named\_gpio(np, "gpios", 2);

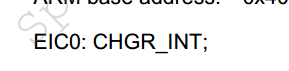
这里是通过读DTS里面的值可通过2723规格书找到对应的功能引脚

－－》7.6 Ext Interrupt Controller(EIC)－－》



这些ＥＩＣ有各自的号



ＧＩＰＯ上定义的是　０４６对应着





uint32\_t irq\_charger\_detect;

gpio\_to\_irq(data->gpio\_charger\_detect); 这个函数是空的

uint32\_t irq\_chg\_cv\_state; 充电中断状态

uint32\_t irq\_vchg\_ovi; 充电过压中断

struct wake\_lock charger\_plug\_out\_lock;

struct workqueue\_struct \*monitor\_wqueue; 创建一个线程

<http://blog.csdn.net/myarrow/article/details/8090504>

struct delayed\_work cv\_irq\_work; 恒流恒压转换进程

struct delayed\_work battery\_work; 电池检查线程

struct delayed\_work battery\_sleep\_work; 电池修眠线程

struct work\_struct ovi\_irq\_work; 电池充电过压线程

struct delayed\_work \*charge\_work; 电池充电线

int (\*start\_charge) (void); 开始充电方法

int (\*stop\_charge) (void); 结束充电方法

#ifdef CONFIG\_LEDS\_TRIGGERS

struct led\_classdev charging\_led;

#endif

};

//电池信息

struct **sprdbat\_info** {

uint32\_t module\_state;

uint32\_t bat\_health;

uint32\_t chging\_current;

int bat\_current;

uint32\_t chg\_current\_type;

uint32\_t adp\_type;

uint32\_t usb\_online;

uint32\_t ac\_online;

uint32\_t vchg\_vol;

uint32\_t cccv\_point;

unsigned long chg\_start\_time;

uint32\_t chg\_stop\_flags;

uint32\_t vbat\_vol;

uint32\_t vbat\_ocv;

int cur\_temp;

uint32\_t capacity;

uint32\_t soc;

uint32\_t chg\_this\_timeout;

};

#ifdef SPRDBAT\_TWO\_CHARGE\_CHANNEL

struct resource \*res = NULL;

#endif

struct device\_node \*np = pdev->dev.of\_node;

详细的设备树说明链接　　　　这里是把pdev里的节点取出来

<http://blog.csdn.net/lichengtongxiazai/article/details/38942033>

相关链接：　<http://blog.csdn.net/jgw2008/article/details/52692616>

SPRDBAT\_DEBUG("sprdbat\_probe\n");

#ifdef CONFIG\_OF

if (!np) {

dev\_err(&pdev->dev, "device node not found\n");

return -EINVAL;

}

#endif

data = kzalloc(sizeof(\*data), GFP\_KERNEL); 申请空间

if (data == NULL) {

ret = -ENOMEM;

goto err\_data\_alloc\_failed;

}

if (np) {

data->pdata = sprdbat\_parse\_dt(pdev); 解析ＤＴ参数

} else {

data->pdata = dev\_get\_platdata(&pdev->dev); 这里取默认值

}

data->dev = &pdev->dev; 这里的data 是申请的设备，Pdev是传进来的

platform\_set\_drvdata(pdev, data); 这里是把生成的data 保存到pdev的私有变量里　dev->p->driver\_data

sprdbat\_data = data; data保存到全局变量中去

print\_pdata(sprdbat\_data->pdata); 打印data中的信息

data->battery.properties = sprdbat\_battery\_props; power\_supply支持

data->battery.num\_properties = ARRAY\_SIZE(sprdbat\_battery\_props);

data->battery.get\_property = sprdbat\_battery\_get\_property;

data->battery.name = "battery";

data->battery.type = POWER\_SUPPLY\_TYPE\_BATTERY;

data->battery.supplied\_to = battery\_supply\_list;

data->battery.num\_supplicants = ARRAY\_SIZE(battery\_supply\_list);

data->ac.properties = sprdbat\_ac\_props;

data->ac.num\_properties = ARRAY\_SIZE(sprdbat\_ac\_props);

data->ac.get\_property = sprdbat\_ac\_get\_property;

data->ac.name = "ac";

data->ac.type = POWER\_SUPPLY\_TYPE\_MAINS;

data->ac.supplied\_to = supply\_list;

data->ac.num\_supplicants = ARRAY\_SIZE(supply\_list);

data->usb.properties = sprdbat\_usb\_props;

data->usb.num\_properties = ARRAY\_SIZE(sprdbat\_usb\_props);

data->usb.get\_property = sprdbat\_usb\_get\_property;

data->usb.name = "usb";

data->usb.type = POWER\_SUPPLY\_TYPE\_USB;

data->usb.supplied\_to = supply\_list;

data->usb.num\_supplicants = ARRAY\_SIZE(supply\_list);

data->start\_charge = sprdbat\_start\_charge; 　　　　　添加开启停止充电功能

data->stop\_charge = sprdbat\_stop\_charge;

ret = power\_supply\_register(&pdev->dev, &data->usb); power\_supply支持

if (ret)

goto err\_usb\_failed;

ret = power\_supply\_register(&pdev->dev, &data->ac);

if (ret)

goto err\_ac\_failed;

ret = power\_supply\_register(&pdev->dev, &data->battery);

if (ret)

goto err\_battery\_failed;

sprdbat\_creat\_caliberate\_attr(data->battery.dev);这个是对属性进行初始化和创建的操作

data->gpio\_chg\_cv\_state = data->pdata->gpio\_cv\_state; 设置电池充电中断

ret = gpio\_request(data->gpio\_chg\_cv\_state, "chg\_cv\_state");

if (ret) {

dev\_err(&pdev->dev, "failed to request gpio: %d\n", ret);

goto err\_io\_cv\_request;

}

gpio\_direction\_input(data->gpio\_chg\_cv\_state); 设置GPIO方向

data->irq\_chg\_cv\_state = gpio\_to\_irq(data->gpio\_chg\_cv\_state); gpio复用成中断

irq\_set\_status\_flags(data->irq\_chg\_cv\_state, IRQ\_NOAUTOEN); 设置中断状态

//irq\_set\_irq\_type(data->irq\_chg\_cv\_state, IRQ\_TYPE\_LEVEL\_HIGH);

ret = request\_irq(data->irq\_chg\_cv\_state, sprdbat\_chg\_cv\_irq,

IRQF\_NO\_SUSPEND, "sprdbat\_chg\_cv\_state", data);

if (ret)

goto err\_request\_irq\_cv\_failed;

gpio\_request :

int gpio\_request(unsigned gpio, const char \*label) GPIO 申请的函数，gpio则为你要申请的哪一个管脚，label则是为其取一个名字。 http://blog.csdn.net/heanyu/article/details/6709571

http://blog.csdn.net/njuitjf/article/details/40622845

gpio\_direction\_input:

设置GPIO方向

http://blog.csdn.net/mirkerson/article/details/8464290

irq\_set\_status\_flags:

设置初始中断状态

request\_irq：

申请中断

data->gpio\_vchg\_ovi = data->pdata->gpio\_vchg\_ovi; 设置电池过压中断

ret = gpio\_request(data->gpio\_vchg\_ovi, "vchg\_ovi");

if (ret) {

dev\_err(&pdev->dev, "failed to request gpio: %d\n", ret);

goto err\_io\_ovi\_request;

}

gpio\_direction\_input(data->gpio\_vchg\_ovi);

data->irq\_vchg\_ovi = gpio\_to\_irq(data->gpio\_vchg\_ovi);

irq\_set\_status\_flags(data->irq\_vchg\_ovi, IRQ\_NOAUTOEN);

ret = request\_irq(data->irq\_vchg\_ovi, sprdbat\_vchg\_ovi\_irq,

IRQF\_NO\_SUSPEND, "sprdbat\_vchg\_ovi", data);

if (ret)

goto err\_request\_irq\_ovi\_failed;

data->gpio\_vbat\_detect = data->pdata->gpio\_vbat\_detect; 电池检测

if (data->gpio\_vbat\_detect > 0) {

gpio\_request(data->gpio\_vbat\_detect, "vbat\_detect");

gpio\_direction\_input(data->gpio\_vbat\_detect);

data->irq\_vbat\_detect = gpio\_to\_irq(data->gpio\_vbat\_detect);

//set\_irq\_flags(data->irq\_vbat\_detect,

// IRQF\_VALID | IRQF\_NOAUTOEN);

irq\_set\_status\_flags(data->irq\_vbat\_detect, IRQ\_NOAUTOEN);

ret =

request\_irq(data->irq\_vbat\_detect, sprdbat\_vbat\_detect\_irq,

IRQ\_TYPE\_LEVEL\_LOW | IRQF\_NO\_SUSPEND,

"sprdbat\_vbat\_detect", data);

if (ret)

dev\_err(&pdev->dev, "failed to use vbat gpio: %d\n",

ret);

}

mutex\_init(&data->lock); 初始化互斥体

wake\_lock\_init(&(data->charger\_wake\_lock), WAKE\_LOCK\_SUSPEND,

"charger\_wake\_lock"); 超时锁　休眠唤醒机制

初始化wakelock: charger\_wake\_lock, 同时将其加入到非活动锁链表中

<http://blog.sina.com.cn/s/blog_a266ea0c0101cbif.html>

INIT\_DELAYED\_WORK(&data->cv\_irq\_work, sprdbat\_cv\_irq\_works);

INIT\_DELAYED\_WORK(&data->battery\_work, sprdbat\_battery\_works);

INIT\_DELAYED\_WORK(&data->battery\_sleep\_work,

sprdbat\_battery\_sleep\_works);

INIT\_WORK(&data->ovi\_irq\_work, sprdbat\_ovi\_irq\_works);

INIT\_WORK(&data->vbat\_detect\_irq\_work, sprdbat\_vbat\_detect\_irq\_works);

INIT\_DELAYED\_WORK(&sprdbat\_charge\_work, sprdbat\_charge\_works);

INIT\_DELAYED\_WORK和INIT\_WORK定时器中断函数　的初始始化工作

<http://blog.csdn.net/daniel80110_1020/article/details/54380453>

对应的函数才是开始工作

INIT\_DELAYED\_WORK()对schedule\_delayed\_work()

INIT\_WORK()对schedule\_work().

data->charge\_work = &sprdbat\_charge\_work;

data->monitor\_wqueue = create\_singlethread\_workqueue("sprdbat\_monitor");

<http://blog.csdn.net/lhl161123/article/details/53447851>

没看懂　因该是开启线程

sprdchg\_timer\_init(sprdbat\_timer\_handler, data);

没看懂 里面都是设置寄存器的操作 然后回调 sprdbat\_timer\_handler方法

sprdchg\_set\_chg\_ovp(data->pdata->ovp\_stop);

操作寄存器CHGR\_CTRL2 和CHGR\_CTRL1　来写入过压保护值

sprdchg\_init(data->pdata); 充电初始化　　信息量不少

sprdfgu\_init(data->pdata); 库伦计初始化 这个里面信息量庞大，

#ifdef CONFIG\_LEDS\_TRIGGERS 宏没开

data->charging\_led.name = "sprdbat\_charging\_led";

data->charging\_led.default\_trigger = "battery-charging";

data->charging\_led.brightness\_set = sprdchg\_led\_brightness\_set;

led\_classdev\_register(&pdev->dev, &data->charging\_led);

#endif

#ifdef SPRDBAT\_TWO\_CHARGE\_CHANNEL 宏没开

sprdchg\_charge\_init\_ext(pdev);

INIT\_DELAYED\_WORK(&sprdbat\_charge\_work\_ext, sprdbat\_charge\_works\_ext);

#ifndef CONFIG\_OF

res = platform\_get\_resource(pdev, IORESOURCE\_IO, 0); 取资源

data->gpio\_charger\_detect = res->start;

#else

data->gpio\_charger\_detect = of\_get\_named\_gpio(np, "gpios", 0);

#endif

gpio\_request(data->gpio\_charger\_detect, "sprd\_charger\_detect");

gpio\_direction\_input(data->gpio\_charger\_detect);

data->irq\_charger\_detect = gpio\_to\_irq(data->gpio\_charger\_detect);

ret = request\_irq(data->irq\_charger\_detect, sprdbat\_chg\_detect\_int,

IRQF\_SHARED | IRQF\_TRIGGER\_HIGH,

"sprdbat\_charger\_detect", data);

INIT\_WORK(&sprdbat\_chg\_detect\_work, sprdbat\_chg\_detect\_works);