Tina

电源管理说明文档 v1.2

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

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

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

目　录

1. 概述........4
1.1. 编写目的........4
1.2. 适用范围........4
2. 电源监控........5
2.1. 代码文件........5
2.2. 框架流程........5
2.3. Epoll 机制........5
2.3.1. epoll_create........6
2.3.2. epoll_ctl........6
2.3.3. epoll_wait........7
2.4. 电源状态........8
2.4.1. 关键数据结构........8
2.4.2. uevent 监听 power_supply 事件........8
3. 关机充电........10
3.1. 框架流程........10
3.1.1. 代码文件........10
3.1.2. 流程框架........11
3.2. 流程分析........12
3.2.1. 数据结构........12
3.2.2. 初始化........12
3.2.3. 处理按键........14
4. Declaration........15

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

1. 概述

文档主要描述 Tina 电源管理，包含电源监控、关机充电等内容。

1.1. 编写目的

简要介绍 Tina 电源管理机制，

1.2. 适用范围

适用于 Tina SDK

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

2. 电源监控

实时监控电源状态，包括电池充放电、电源状态变化，利用消息通知机制，从底层上报给应用，保护系统用电安全以及监控电源状态变化。

2.1. 代码文件

代码路径：

```
tina/package/allwinner/healthd/src/healthd.cpp
tina/package/allwinner/healthd/src/BatteryMonitor.cpp
tina/package/allwinner/healthd/src/healthd_mode_tina.cpp
```

2.2. 框架流程

在 healthd.cpp 的 main 函数里，首先会根据系统进入哪一种模式设置不同的 healthd_mode_ops。在 healthd_init 会添加各个 epoll 事件，最后在死循环 healthd_mainloop 里面等待 epoll 事件的发生，并作出相应的处理，基本流程如下：

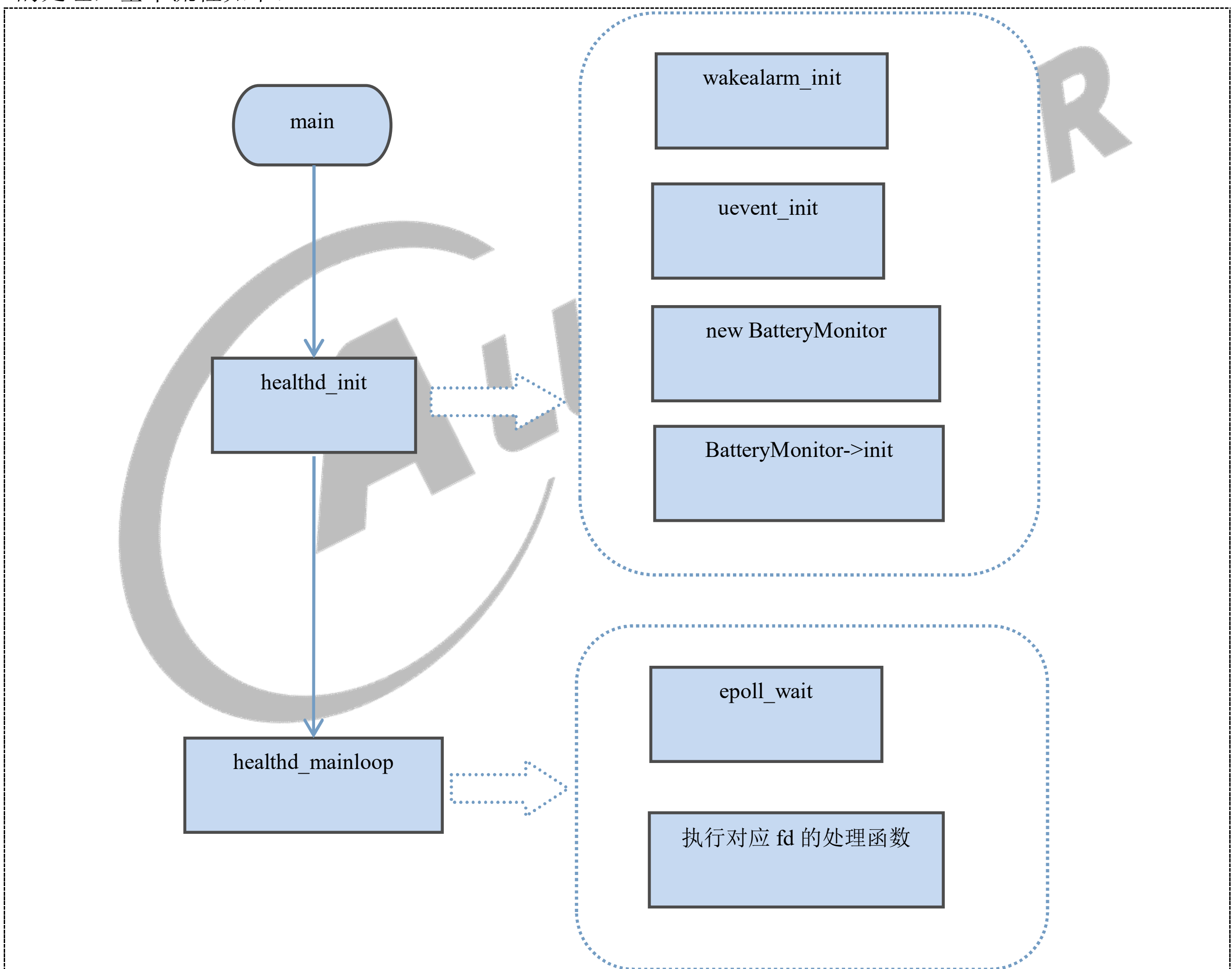

图 2-1 healthd 流程

2.3. Epoll 机制

Healthd 中使用 epoll 进行 IO 复用。使用 epoll_create 创建 epoll 专用的文件描述符。接下来 epoll_ctrl 和 epoll_wait 会与之配套使用。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

2.3.1. epoll_create

```
epollfd = epoll_create(MAX_EPOLL_EVENTS);
if (epollfd == -1) {
    printf( "epoll_create failed; errno=%d\n",errno);
    return -1;
}
```

创建一个 epoll 的句柄，告诉内核监听的数据一共有 MAX_EPOLL_EVENTS 个。生成一个 epoll 专用的的文件描述符，它其实是在内核申请一个空间，用来存放关注的 fd 上是否发生以及发生了什么事件。MAX_EPOLL_EVENTS 就是在这个 epollfd 上能关注的最大 socket fd 数。

2.3.2. epoll_ctl

Uevent_init 使用 uevent_open_socket 创建了 socket 句柄（用于读取内核上报的事件），添加到 epoll 句柄中，并设置相应的回调函数。

```
int healthd_register_event(int fd, void (*handler) (uint32_tt))
{
    struct epoll_event ev;

    ev.events = EPOLLIN;
    if (is_charger)
        ev.events |= EPOLLWAKEUP;
    ev.data.ptr = (void *)handler;
    if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
        TLOGE("epoll_ctl failed; errno=%d\n", errno);
        return -1;
    }

    eventct++;
    return 0;
}

static void uevent_init(void)
{
    uevent_fd = uevent_open_socket(64 * 1024, true);

    if (uevent_fd < 0) {
        TLOGE("uevent_init: uevent_open_socket failed\n");
        return;
    }
    fcntl(uevent_fd, F_SETFL, O_NONBLOCK);
    if (healthd_register_event(uevent_fd, uevent_event))
        TLOGE("register for uevent events failed\n");

}
```

wakealarm_init 使用 timerfd_create 创建了定时器文件句柄，保存在一个全局变量 wakealarm_fd 中。接着，wakealarm_set_interval 设置了唤醒时间间隔。

```
static void wakealarm_init(void)
{
    /* change CLOCK_BOOTTIME_ALARM to CLOCK_MONOTONIC, avoid wakeup frequently! */
    wakealarm_fd = timerfd_create(CLOCK_MONOTONIC, TFD_NONBLOCK);
    if (wakealarm_fd == -1) {
        TLOGE("wakealarm_init: timerfd_create failed\n");
        return;
    }
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        if (healthd_register_event(wakealarm_fd, wakealarm_event))
            TLOGE("Registration of wakealarm event failed\n");

        wakealarm_set_interval(healthd_config.periodic_chores_interval_fast);
    }
```

将被监听的描述符 uevent_fd、wakealarm_fd 添加到 epoll 句柄中。

2.3.3. epoll_wait

```
static void healthd_mainloop(void)
{
    while (1) {
        struct epoll_event events[eventct];
        int nevents;
        int timeout = awake_poll_interval;
        int mode_timeout;
        mode_timeout = healthd_mode_ops->preparetowait();
        DLOG("mode_timeout=%d\n", mode_timeout);
        if (timeout < 0 || (mode_timeout > 0 && mode_timeout < timeout))
            timeout = mode_timeout;
        nevents = epoll_wait(epollfd, events, eventct, timeout);
        if (nevents == -1) {
            if (errno == EINTR)
                continue;
            TLOGE("healthd_mainloop: epoll_wait failed\n");
            break;
        }

        for (int n = 0; n < nevents; ++n) {
            if (events[n].data.ptr)
                (*(void (*)(int))events[n].data.ptr) (events[n].events);
        }
        if (!nevents) {
            periodic_chores();
        }
        healthd_mode_ops->heartbeat();

    }
    return;
}
```

epoll_wait 等待事件的发生，当超过 timeout 还没有事件触发时，就超时。参数 events 用来从内核得到事件的集合，eventct 告知内核这个 events 有多大（数组成员的个数），这个 eventct 不能大于创建 epoll_create 时的 size。参数 timeout 是超时事件。epoll_wait 返回需要处理的事件数目，如返回 0 表示已经超时。当有事件发生时就调用相应的回调函数：uevent_event、wakealarm_event。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

2.4. 电源状态

2.4.1. 关键数据结构

```
struct BatteryProperties {
        bool chargerAcOnline;
        bool chargerUsbOnline;
        bool batteryPresent;
        int batteryStatus;
        int batteryHealth;
        int batteryLevel;
        int batteryVoltage;
        int batteryCurrentNow;
        int batteryTemperature;
        bool batteryLowCapWarn;
        bool batteryOverTempWarn;
};
struct healthd_config{
        int periodic_chores_interval_fast;
        int periodic_chores_interval_slow;
        int batteryPresent;
        int batteryCapacity;
        int batteryVoltage;
        int batteryTemperature;
        int batteryCurrentNow;
        int batteryStatus;
        int batteryHealth;
        bool (*screen_on)(BatteryProperties *props);
};
```

2.4.2. uevent 监听 power_supply 事件

基本流程如下所示：

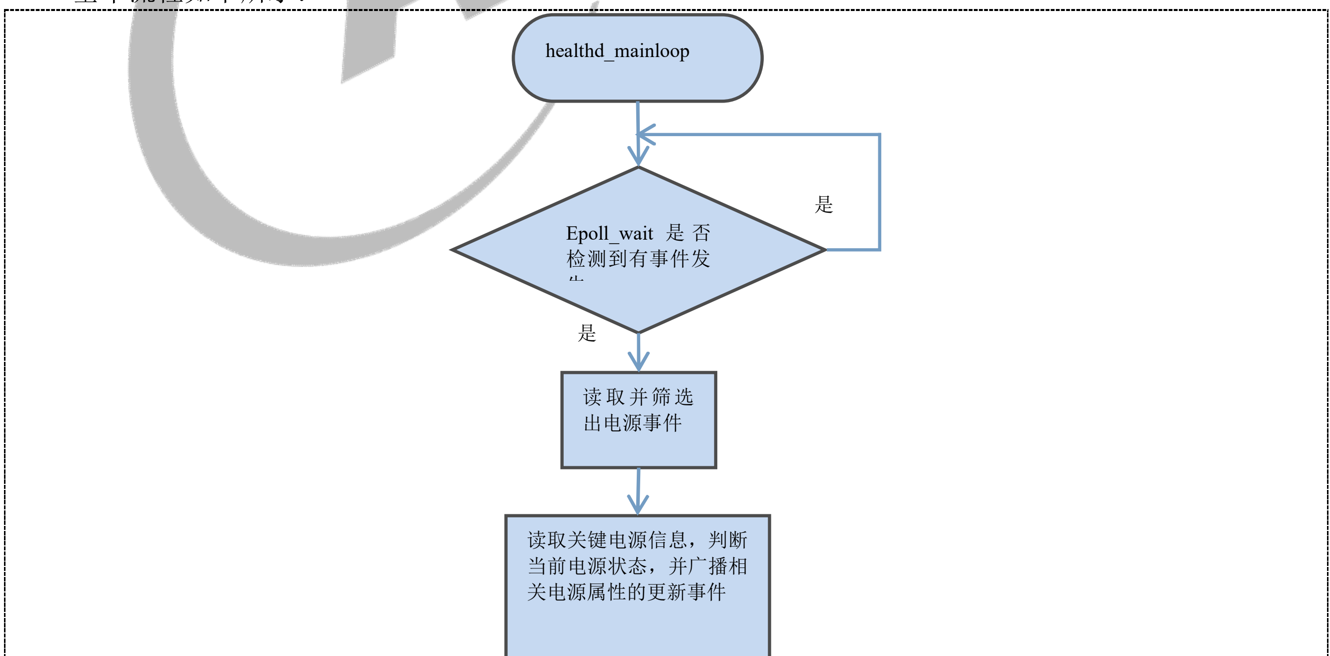

图 2-2 uevent 监听事件流程

Copyright © 2018 by Allwinner. All rights reserved

从 uevent 监听到底层上报的事件中，筛选出电源上报事件，然后进入 BatteryMonitor 的 update 接口，在 update 里去读取/sys/class/power_supply/中关键电源信息,并通过 dbus 发送广播相关电源属性的更新事件。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

3. 关机充电

和正常开机启动不同的是，关机充电会通过 cmdline 的 boot.mode 传入 charger，在 healthd.cpp 中根据该字段设置 healthd_mode_ops。关机充电时 healthd_mode_ops = &charger_ops;

```
static struct healthd_mode_ops charger_ops = {
#ifdef SHUTDOWN_CHARGER
        .init = healthd_mode_charger_init,
        .preparetowait = healthd_mode_charger_preparetowait,
        .heartbeat = healthd_mode_charger_heartbeat,
        .battery_update = healthd_mode_charger_battery_update,
#else
        .init = NULL,
        .preparetowait = NULL,
        .heartbeat = NULL,
        .battery_update = NULL,
#endif
};
```

3.1. 框架流程

3.1.1. 代码文件

代码路径：

```
tina/package/allwinner/healthd/src/healthd.cpp
tina/package/allwinner/healthd/src/healthd_mode_charger.cpp
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

3.1.2. 流程框架

关机状态下，DC/USB 插入机器以后，uboot 会判断出机器是由于插电引起的开机，传递 charger 字符串给 cmdline，在 healthd.cpp 中会去解析 cmdline，如果解析得到 charger，则判断是关机充电，然后就会设置 healthd_mode_ops = &charger_ops。在 charger_ops 的 init 接口会去初始化电源按键的检测，在 charger_ops 的 heartbeat 接口里会去处理按键输入事件、会去处理电源状态事件，根据相应的输入事件作出处理，流程如下：

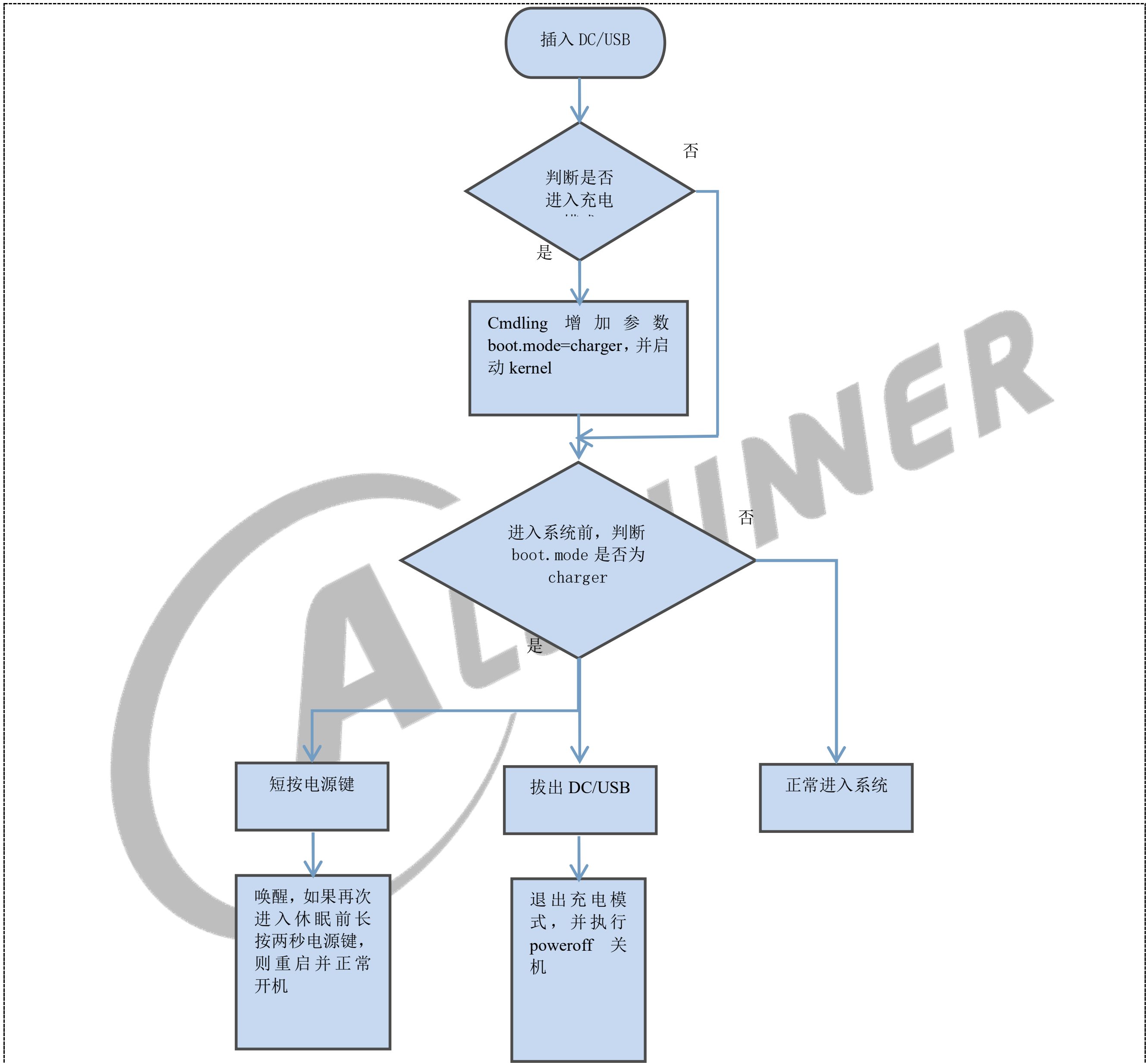

图 3-1 关机充电流程

Copyright © 2018 by Allwinner. All rights reserved

3.2. 流程分析

3.2.1. 数据结构

```
struct charger {
    bool have_battery_state;
    bool charger_connected;
    int64_t next_screen_transition;
    int64_t next_key_check;         //下一次按键检测的时间
    int64_t next_pwr_check;        //下一次 power 检测的时间
    struct key_state keys[KEY_MAX + 1];        //保存按键的状态
    struct animation *batt_anim;
    gr_surface surf_unknown;
};
```

3.2.2. 初始化

```
ret = ev_init(input_callback, charger);
if (!ret) {
        epollfd = ev_get_epollfd();
        healthd_register_event(epollfd, charger_event_handler);
}
......
ev_sync_key_state(set_key_callback, charger);
//检测按键的状态，如果有按键事件，则调用回调函数 set_key_callback()处理
```

set_key_callback()主要用于记录按键的状态，接收三个参数：code、value、data。Code 表示键值，value 表示按键的状态，data 是 charger 的指针。set_key_callback()只记录按下时间与相对应的时间，不作响应处理。

```
static int set_key_callback(int code, int value, void *data)
{
    struct charger *charger = (struct charger *)data;
    int64_t now = curr_time_ms();
    int down = ! !value;

    if (code > KEY_MAX)
        return -1;

    /* ignore events that don't modify our state */
    if (charger->keys[code].down == down)
        return 0;

    /* only record the down even timestamp, as the amount
     * of time the key spent not being pressed is not useful */
    if (down)
        charger->keys[code].timestamp = now; //记录下按下事件的时间
    charger->keys[code].down = down; //记录按键的状态
    charger->keys[code].pending = true;
    if (down) {
        DLOG("[%" PRId64 "] key[%d] down\n", now, code);
    } else {
        int64_t duration = now - charger->keys[code].timestamp;
        int64_t secs = duration / 1000;
        int64_t msecs = duration - secs * 1000;
        DLOG("[%" PRId64 "] key[%d] up (was down for %" PRId64 ".%" PRId64 "sec)\n", now, code,
secs, msecs);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        }

        return 0;
}}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

3.2.3. 处理按键

process_key（）是真正用来处理按键事件的函数

```
static void process_key(struct charger *charger, int code, int64_t now)
{
	struct key_state *key = &charger->keys[code];
	int64_t reboot_timeout = key->timestamp + POWER_ON_KEY_TIME;
	int64_t display_timeout = key->timestamp + (POWER_ON_KEY_TIME / 4);

	DLOG("timestamp=%lld, now=%lld, reboot_timeout=%lld, display_timeout=%lld\n",
			key->timestamp, now, reboot_timeout, display_timeout);
	if (code == KEY_POWER) {
		if (key->down) {
			if (now >= reboot_timeout) {
				reset_animation(charger->batt_anim);
				clear_screen();
				gr_flip();
				DLOG("[%" PRId64 "] rebooting\n", now);
				reboot(RB_AUTOBOOT);
			} else if (now >= display_timeout) {
				kick_animation(charger->batt_anim);
				DLOG("[%" PRId64 "] batt_anim\n", now);
				request_suspend(false);
				set_next_key_check(charger, key, (POWER_ON_KEY_TIME * 3 / 4));
			} else {
				/* if the key is pressed but timeout hasn't expired,
				 * make sure we wake up at the right-ish time to check
				 */
				DLOG("power_key pressed but timeout hasn't expired\n");
				//request_suspend(false);
				set_next_key_check(charger, key, (POWER_ON_KEY_TIME / 4));
			}
		} else {
			/* if the power key got released, force screen state cycle */
			if (key->pending) {
				if (!charger->batt_anim->run) {
					DLOG("re-run animation!\n");
					kick_animation(charger->batt_anim);
					request_suspend(false);
				} else {
					reset_animation(charger->batt_anim);
					charger->next_screen_transition = -1;
					clear_screen();
					gr_flip();
					if (charger->charger_connected) {
						DLOG("enter super standby!\n");
						request_suspend(true);
					}
				}
			}
		}
	}

	key->pending = false;
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

4. 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.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved