全志科技
Allwinner Technology

Tina 功耗管理说明 V1.2

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2016/08/19		初始版本
V1.1	2016/10/27		第一次 review
V1.2	2017/10/24		第二次 review

目　录

1. 概述

文档主要描述 Tina 功耗管理，包含 Standby 管理、场景管理、用户层接口等内容。

1.1. 编写目的

简要介绍 Tina 功耗管理机制，

1.2. 适用范围

适用于 Tina SDK

2. nativepower 全局描述

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

关键代码路径：

```
/tina/package/allwinner/nativepower/deamon/main.c
/tina/package/allwinner/nativepower/libpower
/tina/package/allwinner/nativepower/libsuspend
```

2.2. nativepower package tree

```
/tina/ackage/allwinner/nativepower
        ---daemon          场景管理，服务端监控和处理客户端发来的 dbus 消息
        ---demo            使用说明
        ---files           不同功耗场景的配置
        ---include         具体场景的宏定义和函数定义
        ---libnativepower  应用层接口，具体场景函数的实现以及客户端的处理
        ---libpower        wake_lock 的请求，释放，计数，获取等
        ---libsuspend      监测 wake_lock 的状态，使系统进入不同状态
        ---Makefile        编译规则
```

2.3. 框架流程

在 daemon 的 main.cpp 的 main 函数里：

首先会申请一个 NATIVE_POWER_DISPLAY_LOCK 的锁， 让系统不能进入休眠。

然后通过 autosuspend_enable 启动 libsuspend 中的线程去监测 wake_lock 的状态， 当状态为无锁时，系统自动进入休眠。具体启动的线程与内核支持的 suspend 方式有关。有 earlysuspend_thread_func 和 suspend_thread_func 两种。

接着会启动 ubus_server_thread_func，去响应客户端的请求。

完成这些初始化的动作之后， 会进入 while 循环，循环监测自动休眠时间是否到了， 到了的话就释放 NATIVE_POWER_DISPLAY_LOCK 锁， 让系统进入休眠。

具体框架示意图如下：

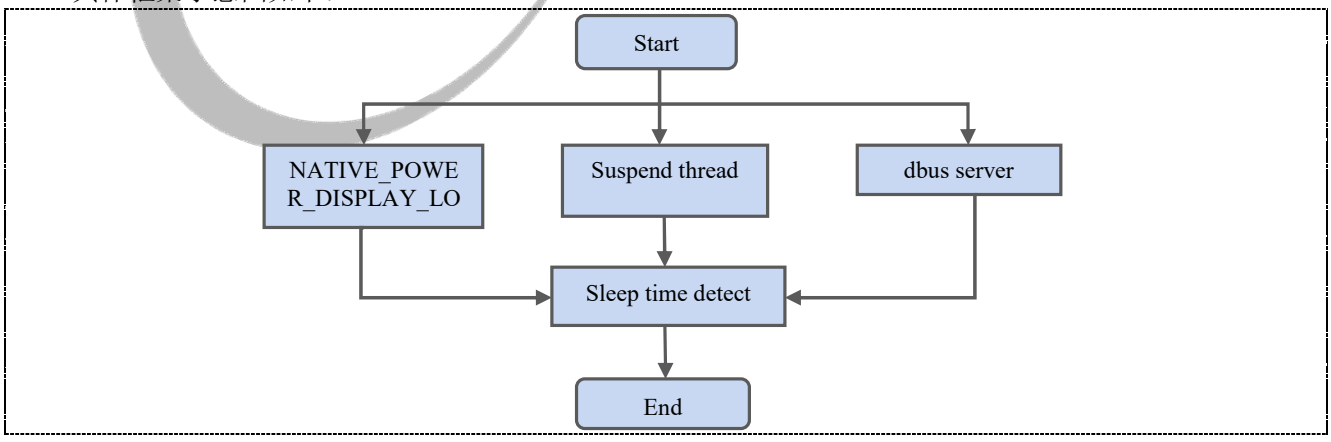

图 2-1 nativepower 框架

2.4. 场景管理

系统启动后，会自动设置一个 boot complete 的场景。其他场景的设置，则由用户程序根据自己的需求去调用应用层接口，通过 bus 来设置。不同的场景，通过配置文件去配置， 详见 3.4 节。

具体到守护进程内，这是调用了 np_scene_change， 然后根据场景名称， 读取/etc/config/nativepower 下的对应配置， 设置到/sys 文件系统的对应节点中。

具体流程示意图如下：

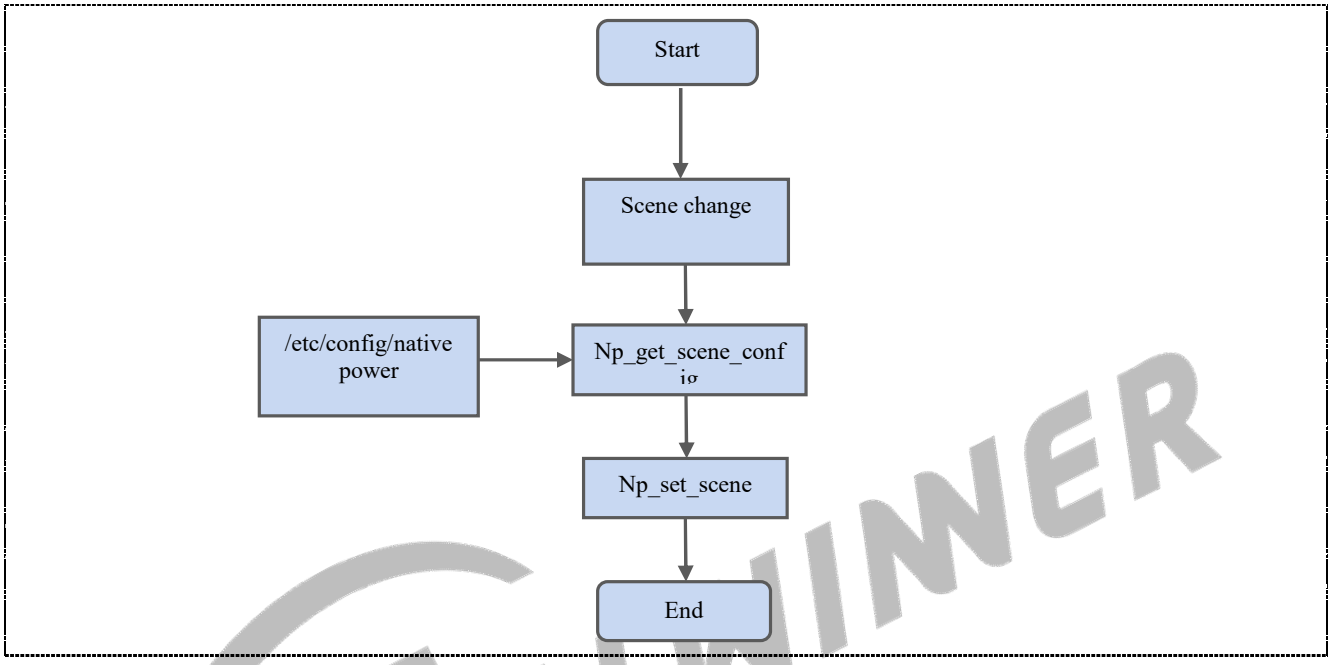

图 2-2 场景管理

2.5. 关键数据结构

```
typedef struct
{
    int (*SetBootLock)(const char *boot_lock);
    int (*SetRoomAge)(const char *room_age);
    int (*SetCpuFreq)(const char *cpu_freq);
    int (*SetCpuGov)(const char *cpu_gov);
    int (*SetCpuHot)(const char *cpu_hot);
    int (*GetCpuFreq)(char *cpu_freq, int len);
    int (*GetCpuOnline)(char *cpu_online, int len);
} CPU_SCENE_OPS;
typedef struct
{
    int (*SetGpuFreq)(const char *gpu_freq);

    int (*GetGpuFreq)(char *gpu_freq, int len);
} GPU_SCENE_OPS;
typedef struct
{
    int (*SetDramScen)(const char *dram_scen);

    int (*GetDramFreq)(char *dram_freq, int len);
} DRAM_SCENE_OPS;
```

以上三个 struct 分别对应 CPU GPU DRAM 对 sys 文件系统节点的操作函数。不同的平台会有不同的操作函数。

```
typedef struct
{
    char bootlock[4];
    char cpu_freq[16];
    char roomage[64];
    char cpu_gov[16];
    char cpu_hot[16];
} CPU_SCENE;

typedef struct
{
    char gpu_freq[16];
} GPU_SCENE;

typedef struct
{
    char dram_freq[16];
    char dram_scene[8];
} DRAM_SCENE;

typedef struct
{
    CPU_SCENE cpu;
    GPU_SCENE gpu;
    DRAM_SCENE dram;
} NP_SCENE;
```

以上结构体对应 uci 配置文件中不同的场景配置， 场景由用户执行设置，然后读入到以上结构体中，设置到系统。

2.6. 使用示例

```
C 语言调用
np_scene_change("boot_complete");

命令行调用
dbus-send --system --type=method_call --print-reply --dest=nativepower.dbus.server /nativepower/service/method nativepower.method.interface.method uint32:7 uint32:0
其中 int32:7 表示 id,固定为 7， uint32:0 与 以下枚举值对应。 可由用户按需添加。

typedef enum
{
    NATIVEPOWER_SCENE_BOOT_COMPLETE,
    NATIVEPOWER_SCENE_MAX,
} NativePowerScene
```

3. 应用层接口

3.1. 代码文件

关键代码路径：

package/allwinner/nativepower/include
package/allwinner/nativepower/libnativepower

3.2. 总体框架

Libnativepower 封装了通过 dbus 发送请求的接口，这些请求会调用守护进程 nativepower_daemon 的功能函数。

3.3. 主要功能函数

```
int PowerManagerSuspend(long event_uptime, SuspendReason reason);      (详见①)
int PowerManagerShutDown(ShutdownReason reason);                       (详见②)
int PowerManagerReboot(RebootReason reason);                           (详见③)
int PowerManagerAcquireWakeLock(const char * id);                      (详见④)
int PowerManagerReleaseWakeLock(const char *id);                       (详见⑤)
int PowerManagerUserActivity(UserActivityReason reason);               (详见⑥)
int PowerManagerSetAwakeTimeout(long timeout_s);                       (详见⑦)
int PowerManagerSetScene(NativePowerScene scene);                      (详见⑧)
```

详注：

① Suspend 该函数提供直接进入 standby 入口，用户调用后系统会直接进入 standby。
② Shutdown 该函数提供系统关机的入口。
③ Reboot 该函数提供重启系统的入口。
④ AcquireWakeLock 该函数提供用户层申请 wake_lock 的入口。
⑤ ReleaseWakeLock 该函数提供用户层释放 wake_lock 的入口。
⑥ UserActivity 该函数提供用户刷新自动待机时间的入口。
⑦ SetAwakeTimeout 该函数提供用户设置自动待机时间的入口。
⑧ SetScene 该函数提供用户设置不同功耗场景的入口。

3.4. 配置文件

用户配置不同功耗场景的配置文件位于

```
package/allwinner/nativepower/files/nativepower
```

具体配置项如下：

```
config scene [场景名称]
        option bootlock
        option cpu_freq         //配置 cpu 频率
        option cpu_gov          //配置 cpu 场景
        option cpu_hot          //配置 cpu 热插拔
        option roomage         //设置 soc 不同的场景
        option gpu_freq         //配置 gpu 频率
        option dram_freq        //配置内存频率
        option dram_scene       //配置内存场景
```

4. 模块分析

4.1. daemon

4.1.1. 概述

开启如下几个线程：

① tinasuspend_thread 自动进入 suspend 的操作。
② thread_key_power 监控按键 power key，并进行处理。
③ dbus_thread 对客户端发来的 dbus 消息进行监控和处理。

4.1.2. 主要功能函数

void wakeup_callback(bool sucess)；	(详见①)
int set_sleep_state(unsigned int state)；	(详见②)
static int open_input(int *fd_array, int *num);	(详见③)
static int open_input(int *fd_array, int *num)；	(详见④)
static void *scanPowerKey(void *arg)；	(详见⑤)
int np_input_init()；	(详见⑥)
int np_scene_change(const char *scene_name)；	(详见⑦)

详注：
①设置 has_sleep 状态，申请 wakelock－NativePower.Display.lock，设置 mLastUserActivityTime。
②获取 sleep_lock 互斥锁，将 state 值设给 has_sleep。说明：0 表示 awake； 1 表示 sleep。
③搜索并打开匹配/dev/input/event*的文件名，将 fd 存放到 fd_array 中，将 num 设置为 fd 个数。
④从系统启动开始计时，获取当前的时间，返回 ms 数。
⑤对 power 按键的监听。
⑥创建执行 scanPowerKey 函数的线程。
⑦调用 np_get_scene_config 依据给定的 scene_name 获取对应的场景信息 scene。

4.2. demo

4.2.1. 概述

打印使用说明。用法为： nativepower_client sel [arg]

4.2.2. 主要功能函数

void usage(char *name);	(详见①)
int main(int argc, char **argv);	(详见②)

详注：
①打印一些使用说明的提示。
②解析第一个参数，依据此参数，分别进行处理：
- SEL_SUSPEND：PowerManagerSuspend
- SEL_SHUTDOWN：PowerManagerShutDown
- SEL_REBOOT：PowerManagerReboot
- SEL_ACQUIRE：PowerManagerAcquireWakeLock
- SEL_RELEASE：PowerManagerReleaseWakeLock
- SEL_USRACTIVITY：PowerManagerUserActivity
- SEL_SETWAKETIME：PowerManagerSetAwakeTimeout
- SEL_SETSCENE：PowerManagerSetScene

4.3. libnativepower

4.3.1. 概述

Client 向 Server 发送一些 dbus 消息。

4.3.2. 主要功能函数

int PowerManagerSuspend(long event_uptime, SuspendReason reason);	(详见①)
int PowerManagerShutDown(ShutdownReason reason) ;	(详见②)
int PowerManagerReboot(RebootReason reason) ;	(详见③)
int PowerManagerAcquireWakeLock(const char * id) ;	(详见④)
int PowerManagerReleaseWakeLock(const char *id) ;	(详见⑤)
int PowerManagerUserActivity(UserActivityReason reason) ;	(详见⑥)
int PowerManagerSetAwakeTimeout(long timeout_s) ;	(详见⑦)
DBusConnection *dbus_client_open() ;	(详见⑧)

详注：
①向 server 发送 dbus 消息，参数包含 NATIVEPOWER_DEAMON_GOTOSLEEP，reason。
②向 server 发送 dbus 消息，参数包含 NATIVEPOWER_DEAMON_SHUTDOWN，reason。
③向 server 发送 dbus 消息，参数包含 NATIVEPOWER_DEAMON_REBOOT，reason。
④向 server 发送 dbus 消息，参数包含 NATIVEPOWER_DEAMON_ACQUIRE_WAKELOCK，wakelock。
⑤向 server 发送 dbus 消息，参数包含 NATIVEPOWER_DEAMON_RELEASE_WAKELOCK，wakelock。
⑥向 server 发送 dbus 消息，参数包含 NATIVEPOWER_DEAMON_USERACTIVITY，reason。
⑦向 server 发送 dbus 消息，参数包括 NATIVEPOWER_DEAMON_SET_AWAKE_TIMEOUT。
⑧调用 dbus_bus_get 连接到系统总线。

4.4. libsuspend

4.4.1. 概述

检测 wake_lock 锁，使系统进入不同状态。

4.4.2. 主要功能函数

static int autosuspend_init(void);	(详见①)
void set_wakeup_callback(void (*func)(bool success));	(详见②)

int wait_for_fb_wake(void);	(详见③)

①autosuspend 的初始化,调用 autosuspend_tinasuspend_init 或者 autosuspend_earlysuspend_init。
②设置全局 wakeup_func 的值
③确保打开/sys/power/wait_for_fb_wake 文件没有错误。

4.5. libpower

4.5.1. 概述

Wake_lock 锁的请求，释放，等待等处理。

4.5.2. 主要功能函数

int acquire_wake_lock(int lock, const char* id);	(详见①)
int release_wake_lock(const char* id)	(详见②)
int get_wake_lock_count()	(详见③)

①调用 initialize_fds 函数，新增了一个 wake_lock。
②调用 initialize_fds 函数，删除一个给定名字的 wake_lock。
③调用 initialize_fds 函数，获取 g_fds[0]对应文件中的所有 wake_lock，统计其个数。

5. 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.