全 志 科 技
Allwinner Technology

A64

A64 Standby 开发调试说明文档

/V1.2

全 志 科 技
Allwinner Technology

Confidential

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2012-10-24	Hx	建立初始版本
V1.1	2012-11-04	Gq.yang	建立 v1.1 版本
V1.2	2013-06-28	Gq.yang	适用于 sunxi 平台
V1.2	2014-01-28	Gq.yang	加入唤醒源的描述

目　　录

1. 概述

1.1. 编写目的

该设计文档的目的在于，简要介绍，为了支持超级 standby，各模块负责人需要完成的工作。

1.2. 适用范围

1.3. 相关人员

预期读者为模块开发者。

2. 模块介绍

2.1. 模块功能介绍

对于嵌入式设备尤其是移动设备来说，功耗是系统的重要指标，系统设计的重要目标之一就是要尽可能地降低功耗。
Standby 模块，在特定策略控制下，通过与 CPU，内存、显示屏，gpu 等相协调，支持对一系列电压和频率的快速调节，从而降低嵌入式系统的功耗。
Standby 模块，支持以下 standby 模式：Normal standby; super standby; extended_standby; bootfast;

2.2. 相关术语介绍

1. Normal Standby （standby）: CPU and RAM are powered but not executed.

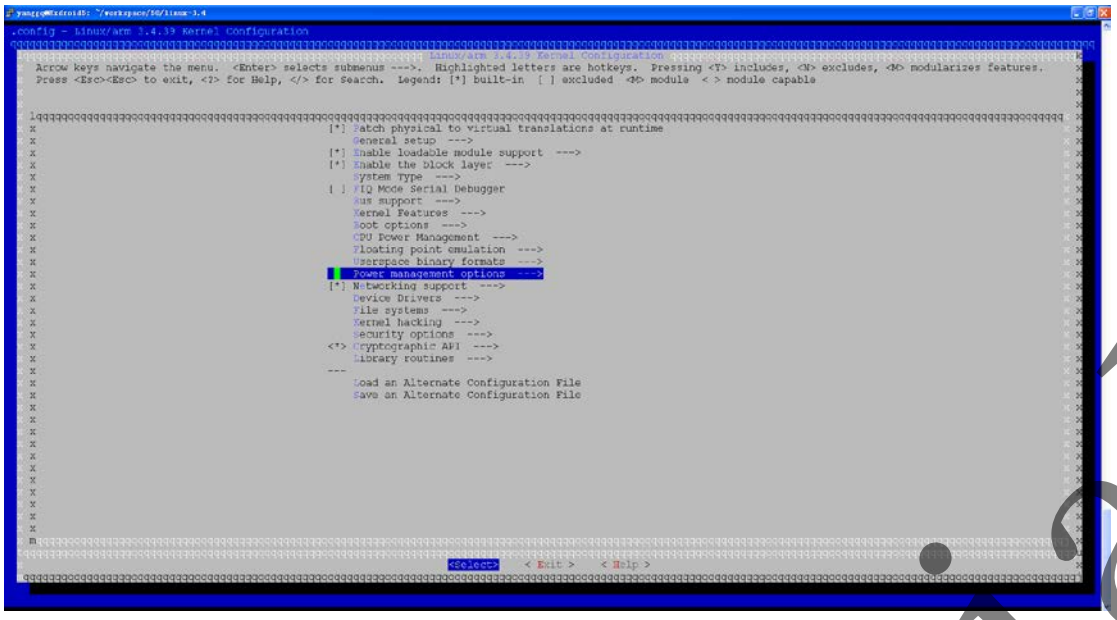
2. super standby: mean Suspend to RAM（mem）, cpu is powed off, while RAM is powered and the running content is saved to RAM。
   超级 standby：等同于 suspend to ram。
   Super standby 或者 normal standby 是根据 cpu 是否断电来定义的。在 super standby 或 者 normal standby， 默认定义了 vcc-3.0 及其它 power-domain 的带电状态;

3. extended_standby： 对 cpu 和各 power domain，clk tree 在 standby 下的带电状态进行了定义。
4. Bootfast：通过唤醒替代开机，对终端用户而言，就是加速开机过程。可以是 super 或者 normal standby 的一种特殊类型。

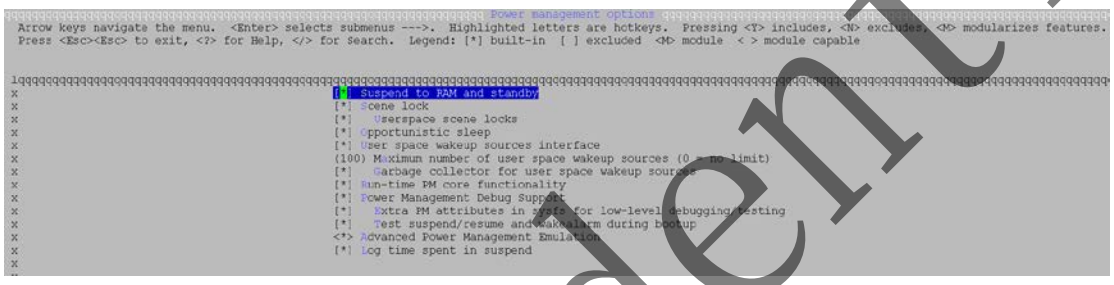
2.3. 模块配置介绍

2.3.1. Menuconfig 配置

运行 make ARCH=arm menuconfig，见下图界面;



选择 Power Management options，按下图所示配置即可。



2.3.2. Menuconfig 配置介绍

CONFIG_ARCH_SUSPEND_POSSIBLE ->
CONFIG_SUSPEND ->
CONFIG_PM_SLEEP->
CONFIG_PM

驱动对 standby 的支持，主要关心 CONFIG_PM_SLEEP 即可。通过 CONFIG_PM_SLEEP 判断，是否支持 standby.

注：CONFIG_PM_SLEEP 生效，则 CONFIG_PM 生效；但 CONFIG_PM 生效，并不意味着 CONFIG_PM_SLEEP 生效，这两个宏的使用，是有所区别的；在 sunxi 平台，这两个宏是等效的；但为了跨平台的兼容，建议使用 CONFIG_PM_SLEEP。

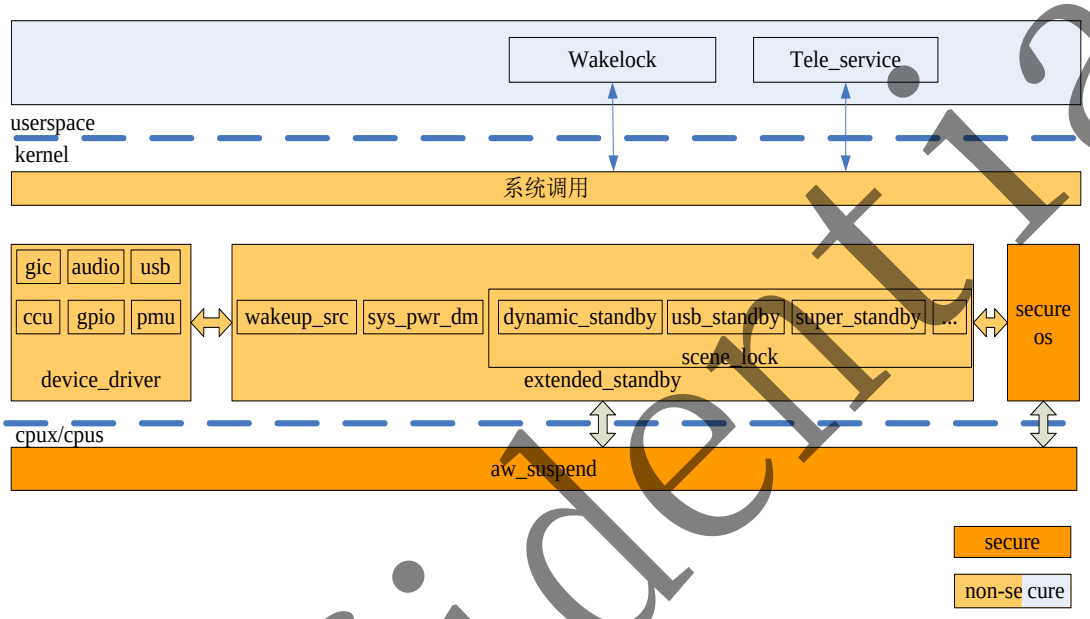
驱动对 scene_manager 的支持，主要关心 CONFIG_SCENELOCK 即可。

2.3.3. 开启 super standby 的支持

要开启 super standby 的支持，请确认在 sysconfig 进行了配置，正确的配置如下：

;--
; if 1 == standby_mode, then support super standby;
; else, support normal standby.
;--
[pm_para]
standby_mode = 1

2.3.4. Extended_standby 参数的配置



架 构

上,wakeup_src, sys_pwr_dm, scene_lock 的配置，共同组成 extended_standby 的配置，这些信息在 cpus 的解析之后，能保证用户所需的场景资源得到保持。

2.3.4.1. Default 状态下唤醒源的配置

对 gpio 唤醒源的配置说明如下:

;--
; wakeup_src_para:
; 注：考虑到 extended_standby 支持唤醒源的动态配置，不再支持唤醒源的静态配置，
; 此配置不再生效；
;--

2.3.4.2. sys_pwr_dm 的配置

根据 soc 的特性，开关后，会影响系统稳定性的 power_domain 交由系统管理，属于 sys_pwr_dm.
在 soc 整合完毕后，sys_pwr_dm 即固定了。归为 sys_pwr_dm 的供电，会由系统进行管理，进行掉电或开电操作；
若出于方案的差异化需求，不希望系统对特定的 power_domain 进行操作，需将特定的 power_domain 从 sys_pwr_dm 拿掉；

以下范例：将 vcc-io 的供电管理，从 arisc 拿掉，从而，进入 suspend 状态时，系统不会关闭 vcc-io 对应的 regulator: dcdc1;
支持哪些 pwr_domain 属性的更改，需参考 pmu_regu 的描述，或咨询系统研发人员；

```
+;----------------------------------------------------------------------------
+;sys_pwr_dm_para
+;this para is used to change default sys_pwr_dm config when necessary.
+;     allowed sys_pwr_dm is such as follow:
+;                     vdd-cpua
+;                     vdd-cpub
+;                     vdd-gpu
+;                     vcc-dram
+;                     vdd-sys
+;                     which is compatible with pmu regu config. see: [pmu1_regu] for more info.
+;     value: 0: del the pwr_dm from sys_pwr_dm_mask;
+;               1: add the pwr_dm into sys_pwr_dm_mask;
+;----------------------------------------------------------------------------
+[sys_pwr_dm_para]
+vcc-io = 0
```

2.3.4.3. 配置 extended_standby

Extended_standby 的配置，是支持动态更改的，因而，对其唤醒源的配置，也是支持动态更改的。因而，extended_standby 的配置应包含两个方面：

1. 供电依赖关系（能够支持对唤醒源的检测）；
2. 支持的唤醒源；

2.3.4.3.1. 供电依赖关系的配置

为方便 extended_standby 的使用，在 extended_standby 基础上，封装了 scene_manager, Scene_lock 的使用，隐藏了系统的供电依赖关系的复杂性；其使用方法，参见 extended_standby 使用文档； 使用 scene_lock 接口进行操作即可；

2.3.4.3.2. Dynamic_standby 的配置

为了应对方案级应用中，对供电依赖关系的更改需求。在 sys_config 中增加了 dynamic_standby 的配置信息；dynamic_standby 与 scene_manager 管理的场景，是平级关系，在 scene_manager 的管理下，进行供电依赖关系的合并，从而满足方案级应用中对 extended_standby 供电依赖关系进行调整的需求。

目前，只支持特定方案中，由于 dram 品质原因，dram 不能进入 selfresh 时，阻止 dram 进入 selfresh 的配置需求。

```
+;----------------------------------------------------------------------------
+;dynamic_standby_para
+;     enable:
+;          value: 0: all config is ignored.
+;                      1: all config is effective.
+;     dram_selfresh_flag:
+;          value: 0: dram will not enter selfresh,
+;                          this config is used for stop dram entering selfresh, in case of dram memory have bug.
+;                      1: dram will enter slefresh.
+;
+;----------------------------------------------------------------------------
+[dynamic_standby_para]
+enable = 0
+dram_selfresh_flag = 0
```

2.3.4.3.3. Extended_standby 下对唤醒源的描述

Standby 总是和唤醒源联系在一起的，唤醒源描述的是使得系统退出 standby 状态的条件。要支持某些唤醒源，总是与 standby 状态下，器件的工作状态联系在一起的。

wakeup_src_para，并不适用于 extended_standby. 因为 extended_standby 将更多的参与动态运行时，各种场景下的功耗管理。在这些场景下的供电依赖关系与唤醒源之间需要保持一致性，才能满足系统的工作需求。

2.3.4.4. Extended_standby 配置的生效原则

wakeup_src, sys_pwr_dm, scene_lock 的配置，共同组成 extended_standby 的配置。这些配置的组合原则是，保证所有的场景都能 work。

因而：可能出现其他场景影响当前场景的情形：例如功耗升高，出现不期望的唤醒事件。若只需要一个场景，可以将其他场景 unlock;

2.3.5. 内核对 super standby 支持

内核导出了两个符号：standby_type, standby_level 以利于各模块：
根据目标区分 normal standby 和 super standby.
根据掉电情况，区别对待设备；
standby_type：表目标，支持 NORMAL_STANDBY，SUPER_STANDBY；
standby_level：表结果，支持 STANDBY_WITH_POWER_OFF，STANDBY_WITH_POWER；

2.3.6. 模块对 super standby 的支持

各模块需要实现 suspend+resume 接口，以支持超级 standby, 确保系统唤醒后，能正常稳定的工作；

1. 包含头文件 linux/pm.h；
2. 判断 standby 类型,并进行相应处理；

示例：

```
if(NORMAL_STANDBY== standby_type){
    //process for normal standby
}else if(SUPER_STANDBY == standby_type){
    //process for super standby
}
```

2.3.7. 模块对 earlysuspend 的支持

各模块需要实现 suspend+resume 接口，以支持超级 standby, 确保系统唤醒后，能正常稳定的工作；

1. 根据内核配置，包含头文件 linux/earlysuspend.h：

```
#ifdef CONFIG_HAS_EARLYSUSPEND
#include <linux/earlysuspend.h>
#endif
```

2. 实现 earlysuspend 相关接口
3. 分配 EARLY_SUSPEND_LEVEL, level 的分配，决定了 late resume 的调用顺序，其分配遵循两个原则：
    1. 调用顺序满足模块之间的依赖关系；
    2. 唤醒时，late_resume 在 display, tp 之后调用；
4. 向内核注册

```
early_suspend.level = EARLY_SUSPEND_LEVEL_** + 3;
early_suspend.suspend = sunxi_**_early_suspend;
early_suspend.resume        = sunxi_**_late_resume;
register_early_suspend(&early_suspend);
```

2.3.8. Late_resume 部分的特别处理

Early_suspend + late_resume 部分的特别处理：

对于处于 early_suspend 部分的模块，在唤醒时，模块可能处于两种状态：

1. 系统并没有进入 super standby 状态，就因为某种原因唤醒了。
2. 系统进入了 super standby 状态，由用户或其他信号唤醒。

取决于系统是否进入了 super standby 状态，模块在唤醒时就有两种可能的状态：带电或掉电；为了让模块清楚的知道，模块唤醒时，系统是否真正进入 super standby, 从而影响了模块的带电状态，内核导出了另一个全局变量：standby_level；若模块认为有需要对带电和不带电两种状态进行区分处理， 从而加速唤醒过程，可借助此变量的帮助。

示例：

```
if(NORMAL_STANDBY== standby_type){
    //process for normal standby
}else if(SUPER_STANDBY == standby_type){
    //process for super standby
    if(STANDBY_WITH_POWER_OFF == standby_level){
        //处理模块经过掉电后的恢复
    }else if(STANDBY_WITH_POWER == standby_level){
        //处理模块带电状态的恢复
    }

}
```

2.4. 源码结构介绍

内核提供的公用代码：

kernel_src_dir\kernel\power\

kernel_src_dir\drivers\base\power\

kernel_src_dir\drivers\base\

平台相关代码：

kernel_src_dir\arch\arm\mach-sunxi\pm

kernel_src_dir\arch\arm\mach-sunxi\pm\”*.[hcS]”: normal standby, super standby 公用代码；

kernel_src_dir\arch\arm\mach-sunxi\pm\standby\” *.[hcS]”: normal standby 相关代码；

kernel_src_dir\arch\arm\mach-sunxi\pm\standby\super\”*.[hcS]”: super standby 相关代码；

各驱动模块源码: ….;

3. 接口描述

3. 1. 用户与内核交互接口

通过 sys 文件系统的节点： /sys/power/state 与上层应用交互；

3.1.1. Linux 层

通过 echo mem > /sys/power/state ，控制系统进入 super standby 状态；
通过 echo standby > /sys/power/state ，控制系统进入 normal standby 状态；

3.1.2. Android

按 power 键，执行 early_suspend 部分；若无用户申请 wake_lock，则进入 kernel 的 suspend，直至掉电；

3. 2. 模块与内核交互接口

3.2.1. 实现 suspend, resume 接口；

功能说明：
1. Suspend： 将所属模块带入低功耗状态，受此影响，该模块可能不能正常工作，其对应的软件服务单元，也应该被禁止运行；
2. Resume: 恢复所属模块至进入 suspend 之前的状态。同时，该模块对应的软件服务单元，也应根据进入 suspend 前的状态，决定是否处于服务状态。

接口说明：

```
static struct bus_driver l3gd20_driver = {
    .class = I2C_CLASS_HWMON,
    .suspend = XXXX;----------------------------------------------→ (2)
    .resume = XXXXX;
    .driver = {
            .owner = THIS_MODULE,
            .name = L3GD20_GYR_DEV_NAME,
            .pm = &l3gd20_pm,----------------------------------→ (1)
    },
    .probe = l3gd20_probe,
    .remove = __devexit_p(l3gd20_remove),
    .id_table = l3gd20_id,
    .address_list    = normal_i2c,

};
```

各模块需初始化(1)或(2)处的 suspend, resume 接口，
若（1）被初始化，则调用（1），否则
若（2）被初始化，则调用（2），否则不执行 suspend, resume 操作；

3.2.2. 实现 earlysuspend 接口

参见：模块对 earlysuspend 的支持

4. Standby 支持范例

步骤

1. 包含头文件 linux/pm.h；
2. 判断 standby 类型,并进行相应处理；
1. 关心场景的模块，还需判断场景，进行处理；

建议： super standby 和 normal standby 使用相同的代码，不对 normal standby 进行特殊处理，可减少代码的维护量，且对系统性能影响不大。

示例：

```
if (check_scene_locked(SCENE_XXXX_STANDBY) == 0) {
    //process for the scene you care.
} else {
    //process    for super standby and normal standby.
}
```

4.1. 普通模块对 super standby 的支持

参考 sunxi-keyboard 对 suspend 操作的处理：

```
                    if (check_scene_locked(SCENE_TALKING_STANDBY) == 0) {
185                     printk("lradc-key: talking standby, enable wakeup source lradc!!\n");
186                     enable_wakeup_src(CPUS_LRADC_SRC, 0);
187                 } else {
188                     sunxi_keyboard_ctrl_set(0, 0);
189                 }
```

4.2. Late_resume 部分的特别处理

参考 gt818_ts 对 resume 操作的处理：

```
    if(STANDBY_WITH_POWER_OFF == standby_level){
        //reset
        ctp_ops.ts_reset();
        //wakeup
        ctp_ops.ts_wakeup();

        //set to input floating
        gpio_set_one_pin_io_status(gpio_wakeup_hdle, 0, "ctp_wakeup");

        ret=goodix_init_panel(ts);

    }else if(STANDBY_WITH_POWER == standby_level){
```

```
        if(ts->use_irq) {
            ret = ctp_ops.set_irq_mode("ctp_para", "ctp_int_port", CTP_IRQ_MODE);
            if(0 != ret){
                printk("%s:ctp_ops.set_irq_mode err. \n", __func__);
                return ret;
            }
        }
        else {
            //。。。。。。。。
        }
    }
```

因为 wakeup, reset 这类操作需要较长的延时,仅在掉电的条件下，才进行 ctp 的 reset, wakeup 操作，可有效加快未进入 super standby 状态时的唤醒速度.

5. 唤醒源介绍

Standby 总是和唤醒源联系在一起的，唤醒源描述的是使得系统退出 standby 状态的条件。唤醒源在保持小机的人机交互功能方面，发挥了重要作用，是实现 standby 时，必须要考虑的部分。要支持某些唤醒源，总是与 standby 状态下，器件的工作状态联系在一起的。因而对唤醒源的描述需要包含两个属性：

1. 支持的唤醒源；
2. 供电依赖关系，能够支持对唤醒源的检测。

不同的 standby, 对应不同的唤醒源需求 + 不同的供电依赖关系。下表，列出了 sunxi 方案，不同的 standby 能支持的唤醒源。

唤醒源	sunxi-evb 支持情况（power_domain）	是否计划支持	是否已支持	super standby	Extended_standby	bootfast
VBUS/ACIN insert/remove	Axp_xx	Y	Y	Y	Y	N
PWRON key(下降沿)		Y	Y	Y	Y	N
PWRON key(长按键)		Y	Y	N	N	Y
Low Power		Y	Y	Y	Y	Y
pmu wakeup gpio		Y	Y	Y	Y	N
ir	cpus	Y	Y	N	方案决定	N
rtc		Y	Y	Y	Y	Y
3g		Y	Y	N	方案决定	N
other cpus wakeup gpio		Y	Y	Y	方案决定	N
wifi	cpu0	Y	Y	N	方案决定	N
蓝牙		Y	Y	N	方案决定	N
other cpu0 wakeup gpio		Y	Y	N	方案决定	N
usb	Soc 决定	Y	Y	N	方案决定	N

为了支持多样的唤醒源需求，引入了 extended_standby. 为了在产品端简化 extended_standby 的使用，在 extended_standby 的基础上，封装了 talking_standby, usb_standby, gpiox_standby 等。 如何支持各种 **_standby, 及对应的唤醒源，请查看 extended_standby 使用文档。

6. 总结

6.1. 模块与内核交互接口的改变

模块与内核交互接口较多，本文档，只描述了最通常的交互接口，后期会根据需要及对休眠唤醒理解的加深，启用某些接口，相应的，本文档会修订。