全志科技
Allwinner Technology

V40 项目

standby 使用说明书 V1.0

文档履历

版本号	日期	制/修订人	内容描述
V1.0	2016-5-17		正式版本

目 录

V40 项目 1
standby 使用说明书 V1.0 1
目 录 3
1. 概述 4
1.1. 编写目的 4
1.2. 适用范围 4
1.3. 相关人员 4
2. 模块介绍 5
2.1. 模块功能介绍 5
2.2. 模块配置介绍 5
2.2.1. Menuconfig 配置 5
2.2.2. Extended_standby 下对唤醒源的配置 6
2.2.3. 调试打印信息配置 6
2.3. 内核对 standby 支持 7
2.4. 普通模块对 super standby 的支持 7
2.5. late_resume 部分的特别处理 7
2.6. 源码结构介绍 8
3. 接口描述 9
3.1. 用户与内核交互接口 9
3.1.1. Linux 9
3.1.2. Android 9
4. Debug 调试 10
4.1. 调试节点 10
4.2. 唤醒源分析 10
4.2.1. 抓取休眠打印 10
4.2.2. 开启调试选择，对唤醒源进行解析 11
4.2.3. 抓取唤醒的打印 11
4.2.4. 开启调试选项，对唤醒源进行解析 11
4.3. Bug 的定位与解决 11
4.3.1. 系统不能进入睡眠 11
Declaration 13

1. 概述

1.1. 编写目的

该文档的目的在于，简要介绍，V40 sdk 所支持的 standby 行为，以及为了支持 standby，各驱动模块所需工作及其注意的事项。

1.2. 适用范围

硬件平台：V40
软件平台：V40 sdk v1.0 及其后续版本。

1.3. 相关人员

V40 平台下的 bsp 开发/维护人员。

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。

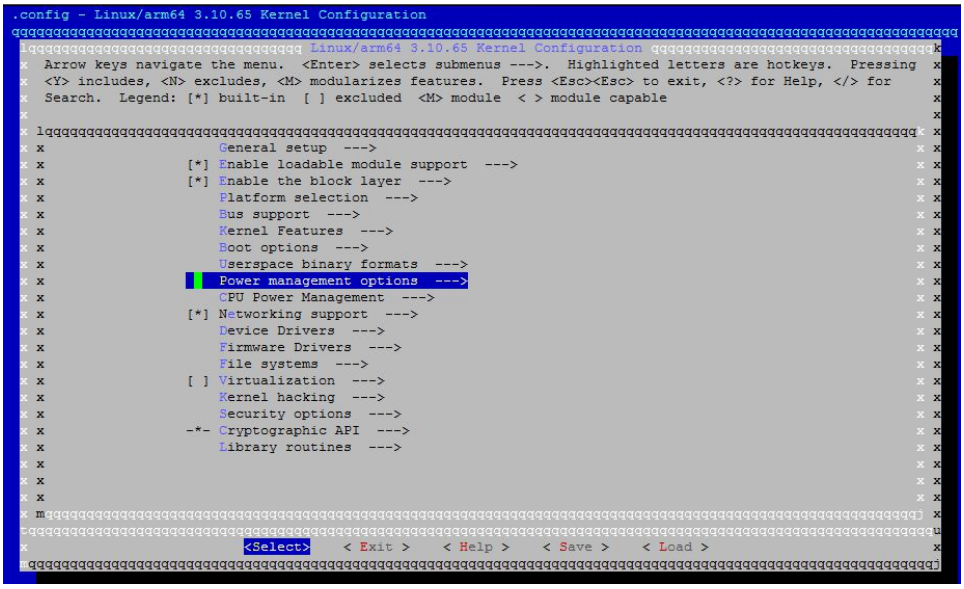
standby 模块，具有以下特点：

1. 支持 super standby + extended_standby；
2. 支持 standby 模式下：唤醒源的增加；
3. 在 extended_standby 的支持下，支持在特定场景下，进入 standby 的可能（如：usb_standby，normal standby， misc_standby）。

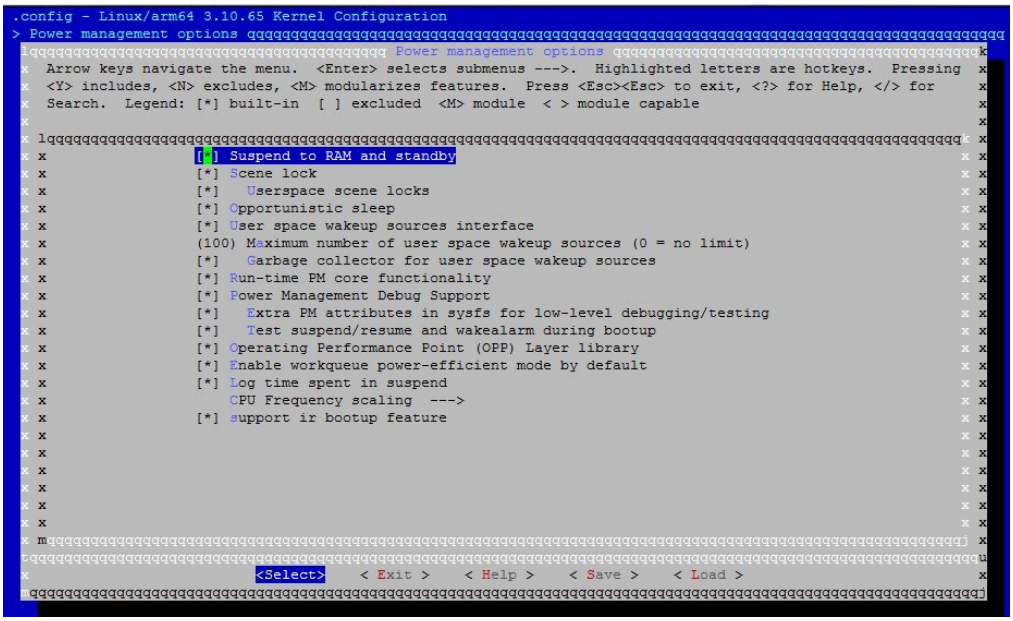
2.2. 模块配置介绍

2.2.1. Menuconfig 配置

1. 运行 make ARCH=arm menuconfig，见下图界面;



2. 选择 Power Management options，按下图所示配置即可：



2.2.2. Extended_standby 下对唤醒源的配置

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。

extended_standby 将参与动态运行时，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。

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. 唤醒源；

2.2.3. 调试打印信息配置

通过以下命令：echo 0xff > /sys/power/aw_pm/debug_mask
更改 debug_mask 的配置，可控制 standby 的打印。
每一个 bit 控制不同内容的打印，如下：

```
typedef enum{
    PM_STANDBY_PRINT_STANDBY          = (1U << 0),
    PM_STANDBY_PRINT_RESUME           = (1U << 1),
    PM_STANDBY_ENABLE_JTAG            = (1U << 2),
    PM_STANDBY_PRINT_PORT             = (1U << 3),
    PM_STANDBY_PRINT_IO_STATUS        = (1U << 4),
    PM_STANDBY_PRINT_CACHE_TLB_MISS   = (1U << 5),
    PM_STANDBY_PRINT_CCU_STATUS       = (1U << 6),
    PM_STANDBY_PRINT_PWR_STATUS       = (1U << 7),
    PM_STANDBY_PRINT_CPUS_IO_STATUS   = (1U << 8),
    PM_STANDBY_PRINT_CCI400_REG       = (1U << 9),
    PM_STANDBY_PRINT_GTBUS_REG        = (1U << 10),
    PM_STANDBY_TEST                   = (1U << 11),
```

```
    PM_STANDBY_PRINT_RESUME_IO_STATUS     = (1U << 12)
}debug_mask_flag;
```

2.3. 内核对 standby 支持

内核导出了两个符号：standby_type, standby_level, 同时提供了一个场景接口，以利于各模块实现对 standby 的支持：

根据目标： 区分 normal standby 和 super standby.

根据掉电情况：区别对待设备；

根据场景：区分系统当前处在的场景，关心该场景的模块，需要对其进行特殊处理；

变量介绍：

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；

check_scene_locked： 获取系统当前处在的场景，talking, usb, bootfast, or etc...

2.4. 普通模块对 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；流程如下：

1. 包含头文件 linux/pm.h；
2. 判断 standby 类型,并进行相应处理；
3. 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.
}
```

2.5. late_resume 部分的特别处理

late_resume 部分的特别处理：

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
    }
}
```

2.6. 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\driver\soc\allwinner\pm

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

通过 echo mem > /sys/power/state，控制系统进入休眠状态
通过 echo super standby > /sys/power/scene_lock 控制进入 super standby 状态；
通过 echo normal standby > /sys/power/scene_lock 控制进入 super standby 状态；

3.1.2. Android

1. 按 power 键，执行 check the wake_count and wake_lock 部分；
2. 若无用户申请 wake_lock, 则进入 kernel 的 suspend 流程；
3. 平台实现的 standby 处理，直至掉电。

4. Debug 调试

4.1.调试节点

节点路径	default	debug	简介
/sys/module/printk/parameters/console_suspend	Y	N	console_suspend： 该值为 Y，表明 suspend 控制台,此后，所有的打印，都被放入 buffer 空间，不再经控制台输出
/sys/module/kernel/parameters/initcall_debug	N	Y	该值为 Y，会打印各个 device 的名称，调用的开始和结束时刻；linux-3.10 及之后的版本,打印的是 syscore 对应的设备
/proc/sys/kernel/printk	4	8	调整内核打印级别
/sys/power/pm_print_times	0	1	该值为 1，会打印各个 device 的名称，调用的开始和结束时刻；
/sys/module/pm_tmp/parameters/parse_bitmap_en	0	0xf	打开 bitmap 的解析开关，确保 bitmap 被解析后，以更直观的方式表示唤醒源的配置信息；
/sys/power/aw_pm/debug_mask	0	0x8f3	查看 suspend 之前，系统资源的状态；输入：cat /sys/power/aw_pm/debug_mask， 可以查看每一个 bit 的意义；
/sys/module/pm_tmp/parameters/aw_ex_standby_debug_mask	0	1	开启 extend_standby 调试开关，查看 extended_standby 的配置过程和信息
echo mem > /sys/power/state			触发 kernel 进入休眠，在 android 环境下，不建议使用
input keyevent 26			触发 Android 进入休眠

4.2.唤醒源分析

4.2.1. 抓取休眠打印

输入命令：

```
echo N > /sys/module/printk/parameters/console_suspend
```

抓取打印：

```
[  203.780931] dynamic config wakeup_src: 0x0        //此处提示：动态配置的唤醒源信息，由设备驱动设置；
[  203.780931] total dynamic config standby wakeup src config: 0x4.  //此处提示：系统结合管理需求（调试：超时唤醒；场景管理：bootfast）生成的动态配置的唤醒源信息
[  203.780931] final standby wakeup src config = 0x15.  //此处提示：系统默认配置(powerkey，低电提示等) + 动态配置  的唤醒源信息
```

备注：不同的 sdk，打印内容有略微差异，但唤醒源的信息，都是由三个部分组成：设备驱动配置，系统动态管理需求，系统默认配置；

4.2.2. 开启调试选择，对唤醒源进行解析

输入命令：
echo 0xf > /sys/module/pm_tmp/parameters/parse_bitmap_en
再次抓取打印：
[81.475395] enable CPU0_WAKEUP_MSGBOX.//此处提示：通过字符串，描述了允许唤醒的唤醒源；
[81.475395] enable CPU0_WAKEUP_EXINT.
[81.475395] enable CPU0_WAKEUP_ALARM.

4.2.3. 抓取唤醒的打印

[81.475395] [pm]platform wakeup, standby wakesource is:0x10 //此处提示：唤醒源是 0x10 对应的事件，
//wakesource 是一个 32bit 的整形，最多支持 32 个唤醒源信息的描述；
//若要得到每个 bit 对应的唤醒源，须对这个数据进行解析；
备注：对于无 arisc 架构的 soc，当进入 super standby 时， 该提示信息是无效的；
理由是：此时，soc 受 pmic 控制，唤醒源有限（nmi, irq, gpio）,交由 pmic driver 管理即可；

4.2.4. 开启调试选项，对唤醒源进行解析

输入命令：
echo 0xf > /sys/module/pm_tmp/parameters/parse_bitmap_en
再次抓取打印：
WAKEUP_SRC is as follow:
'''CPU0_WAKEUP_ALARM''' bit 0x10 //此处提示：通过字符串，描述了 0x10 对应的事件；

4.3. Bug 的定位与解决

4.3.1. 系统不能进入睡眠

不能进入休眠，有以下原因：
1. 有设备驱动或应用程序，持有 wakelock, 导致系统不能进入休眠；
2. 有设备驱动，在休眠过程中，通过 wake_lock(wakeup_sources)申请中断休眠，导致系统唤醒；
3. 其它情形；

4.3.1.1. 查看 android wake_lock 状态：dumpsys power

查看与电源相关的信息，主要用于定位因为持有 wakelock 而不能进入 suspend 的 bug。
Eg: 插入 usb 而不能进入 suspend 的时候，dumpsys power 的输出如下：

mLocks.size=3:
PARTIAL_WAKE_LOCK 'UsbDeviceManager' activated (minState=0, uid=1000, pid=1884)
SCREEN_DIM_WAKE_LOCK 'StayOnWhilePluggedIn Screen Dim' activated (minState=1, uid=1000, pid=1884)
PARTIAL_WAKE_LOCK 'StayOnWhilePluggedIn Partial' activated (minState=0, uid=1000, pid=1884)

4.3.1.2. 查看 kernel wake_lock 状态：wake_lock + wakeup_sources

wake_lock
查看内核节点：/sys/power/wake_lock，确认内核持有的 wake_lock 状态
实例如下：
cat /sys/power/wake_lock
PowerManagerService.Display PowerManagerService.WakeLocks

通过添加 wake_lock, 可以阻止系统进入休眠
实例如下：
echo test_wake_lock > /sys/power/wake_lock
wakeup_sources

在 linux-3.10 上，wake_lock 的本质，是 wakeup_sources；
查看内核节点：/sys/kernel/debug/wakeup_sources， 可以确认内核持有的 wakeup_sources 状态
确定 active_since 的值 value，是否为 0； 若为非 0， 表明该 wakeup_source 已经阻止系统进入休眠 value ms；
定位之后，联系对应模块开发人员，解决即可；

实例如下：
cat /sys/kernel/debug/wakeup_sources

name	active_count	event_count	wakeup_count	expire_count	active_since	total_time	max_time	last_change	prevent_suspend_time
PowerManagerService.WakeLocks	1	1	0	0	97046	97046	97046	18204	0

Declaration