全志科技
Allwinner Technology

R18 设置 PL8 唤醒问题

---经验总结

Author : ranchao

Mail : flyranchao@allwinnertech.com

Created Time:20180307

Copyright © 2013 by Allwinner. All rights reserved

文档履历

版本	日期	修改人	修改内容
V1.0	2018.03.07	冉超	创建初始版本

目录

1. 引言.....3
1.1. 编写目的.....3
1.2. 适用范围.....3
1.3. 术语与缩略语.....3
2. 客户描述.....4
2.1. 问题描述.....4
2.2. 具体操作.....4
2.3. 操作结果.....4
3. 问题分析.....5
3.1. 排除法缩小问题范围.....5
3.2. 定位问题.....5
3.3. 验证猜想.....7
3.4. 验证结果.....7
4. 学习心得.....8
4.1. 中断流程.....8
4.2. Wifi 中断流程.....8
4.3. GPIO 配置信息.....8
4.4. 读写寄存器值.....8
4.5. 相关操作说明.....9

1. 引言

1.1. 编写目的

此文档详细记录本人在解决科通 R18 配置 PL8 为唤醒源时无法唤醒的问题的一些总结，以便大家日后参考。

1.2. 适用范围

基于 ubuntu14.04 演示

1.3. 术语与缩略语

术语/缩略语	解释说明
CPUS	小 cpu
CPUX	大 cpu(cpu0,cpu1...)
GIC	中断控制器
PL8	Cpus 端的 pin8 引脚

2. 客户描述

2.1. 问题描述

科通在 R18 板子上配置 PL8 引脚为中断唤醒源，当系统进入休眠后，尝试拉高 PL8 唤醒系统。结果无法唤醒。

2.2. 具体操作

echo 0x800000 0x168 1 > /sys/power/sunxi/wake_src （设置 PL8 为唤醒源）

echo mem > /sys/power/state （系统进入休眠）

拉高 PL8 引脚尝试唤醒系统

2.3. 操作结果

PL8 无法唤醒，power 按键可以唤醒。

3. 问题分析

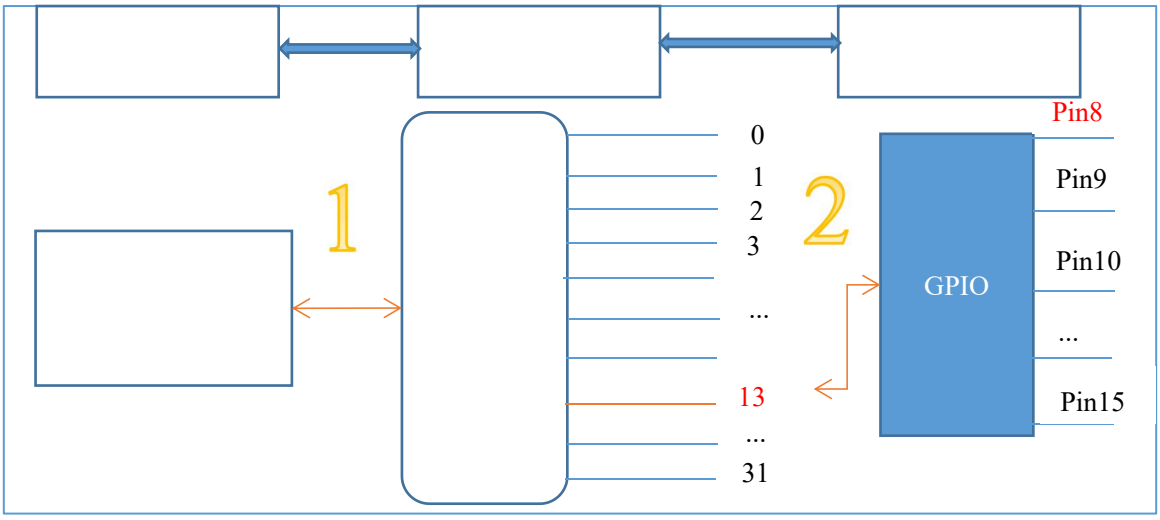


图 1 中断流程

从图中可以明显看出，PL8-PL15 共享一个中断号 13，中断控制器共 32 个中断号，配置 PL8 为唤醒源，即使用 PL8 为中断源，图中橙色线路必须是通的。

3.1. 排除法缩小问题范围

1.尝试其他唤醒源：power 按键唤醒

结果--->power 键能够唤醒。

结论--->说明 1 号线路和 GIC(中断控制器)肯定是没问题的，那么问题就出在 2 号线路，以及 GPIO 组，Pin8 引脚。

2.尝试其他 PL 脚唤醒：配置 PL9 为唤醒源

结果--->系统进入休眠后，拉高 PL9 引脚能够唤醒系统。

结论--->说明 2 号线路，以及 GPIO 组是没有问题。

3.2. 定位问题

1.经过上面两个尝试，将问题定位在对 PL8 引脚的配置上。

借助 wifi 中对 PL 脚的配置流程(PL3)：

GPIO 配置：

Name = port: PLxx <功能> <触发方式> <驱动力> <输出电平>

wlan_regon = port:PL02<1><default><default><0>

wlan_hostwake = port:PL03<6><default><default><0>

结果--->ifconfig wlan0 up 或者 ifconfig wlan0 down 时，能够打开或是关闭 wifi。通过在小机端

echo 0x01f02c00 > /sys/class/sunxi-dump/dump ;cat dump

确实可以发现对应 PL3 寄存的值发生了变化。

结论--->结合 wifi 的配置，并查看 spec 文档

说明 PL8 的配置是没有问题的，那么需要考虑的就是配置是否生效。

2.所以进一步追踪 wifi 驱动调用流程，通过添加打印的方式确定具体是哪一个函数调用时修改了寄存器 PL3 的值。

- 在 drivers/net/wireless/bcmdhd/dhd_linux.c 中

dhd_ioctl_entry()--->dhd_ioctl_process()--->dhd_bus_start()--->dhd_bus_oob_intr_register()

- 在 drivers/net/wireless/bcmdhd/dhd_sdio.c 中两处调用到 bcmsdh_oob_intr_register()

dhd_bus_oob_intr_register()--->bcmsdh_oob_intr_register()

- 在/drivers/net/wireless/bcmdhd/bcmsdh_linux.c 中

bcmsdh_oob_intr_register()--->request_irq(...,wlan_oob_irq,...)--->wlan_oob_irq()--->
bcmsdh_oob_intr_set(){设置中断}--->enable_irq()

最终发现在调用 request_irq(...,wlan_oob_irq,...)函数时对 PL3 寄存器的值进行了修改。具体修改分析如下：

执行 ifconfig wlan0 up 时，寄存器 0x01f02c00 的值由 0x77127222--->修改为 0x77126222，刚好是 PL03 这一位由 7（disable）--->修改为 6（intc 即中断模式）

drivers/net/wireless/bcmdhd/bcmsdh_linux.c 中调用 request_irq()

```
request_irq(unsigned int irq, irq_handler_t handler, unsigned long flags,
                const char *name, void *dev)
{
        return request_threaded_irq(irq, handler, NULL, flags, name, dev);
}
```

分析得出在配置 PL8 为唤醒源时同样也是会调用到这个函数，通过比较差异发现，函数在实现上都是一样的，或者说该函数就是公共的，相同的，不同的中断源都会最终调用到这里，只是传递的参数不同而已。

接下来仔细研究一下该函数的参数：

unsigned int irq ：要申请的硬件中断号

irq_handler_t handler ：中断服务函数

unsigned long flags： 中断标志

const char *name ：中断名称（设备驱动的名称）

void *dev ：共享中断时的中断区别参数

devname 设置中断名称，通常是设备驱动程序的名称 在 cat /proc/interrupts 中可以看到此名称。

3.于是进一步猜想可能是 irq_handler_t handler，unsigned long flags 两个参数的影响。先尝试证明 unsigned long flags 参数的影响。

发现在 wifi 中调用 request_irq()时，传进来的

oob_irq_flags = (IRQF_TRIGGER_HIGH | IRQF_SHARED | IRQF_NO_SUSPEND)

而在配置 PL8 时调用 request_irq()时，传进来的

flags =(IRQF_TRIGGER_HIGH | IRQF_SHARED)

于是研究一下这些常量的含义：

IRQF_TRIGGER_HIGH

设置中断触发模式，高电平触发。

IRQF_SHARED

这是 flag 用来描述一个 interrupt line 是否允许在多个设备中共享。

IRQF_NO_SUSPEND

这个 flag 比较好理解，在系统 suspend 的时候，不用 disable 这个中断，如果 disable，可能会导致系统不能正常 resume。

IRQF_ONESHOT

one_shot 为保证中断在底半部 threaded_handler 执行完之后才会继续接受中断，作用是保证 thread_handler 函数执行完整，才会接受下一个中断信号。

4.最终猜想是由 IRQF_NO_SUSPEND 标志引起的。

3.3. 验证猜想

配置 PL8 为唤醒源时，在驱动中修改

flags = (IRQF_TRIGGER_HIGH | IRQF_SHARED | IRQF_NO_SUSPEND)

即加上 IRQF_NO_SUSPEND 标志。

3.4. 验证结果

通过执行：

echo 0x800000 0x168 1 > /sys/power/sunxi/wake_src （设置 PL8 为唤醒源）

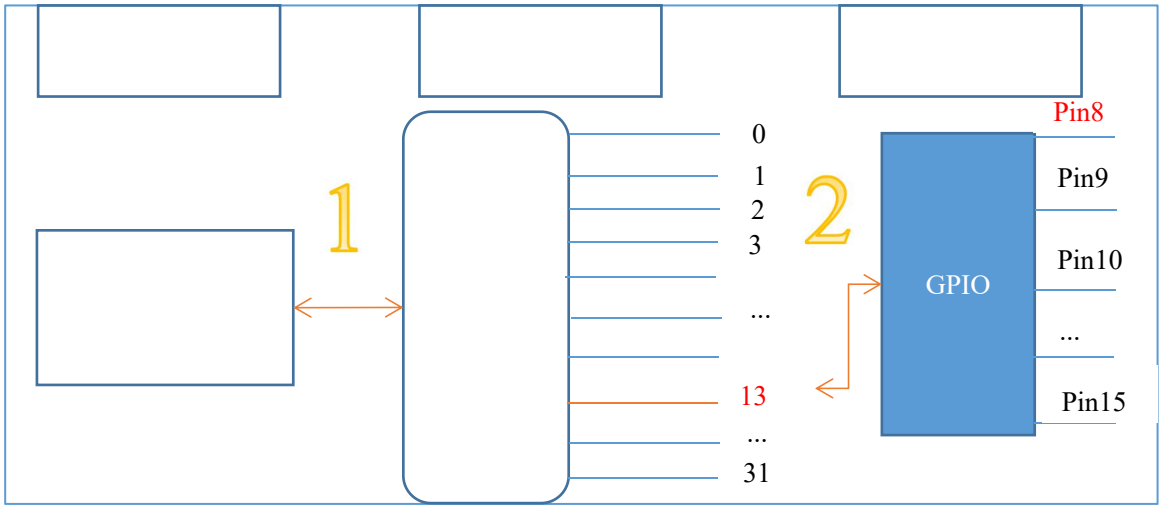
echo mem > /sys/power/state （系统进入休眠）

拉高 PL8 引脚能够唤醒系统。

4. 学习心得

4.1. 中断流程

熟悉了中断流程，了解了中断控制器。



4.2. Wifi 中断流程

熟悉了 wifi 驱动，了解了 request_irq()函数，清楚了相关中断标志的含义。

4.3. GPIO 配置信息

了解了 GPIO 各配置项的具体含义：

```
[主键]
子键    = port: PLxx <功能分配> <触发方式> <驱动能力> <输出电平>
```

Port: 端口分组，如 porta,portb 等等；

PLxx: 组内序号，如 PL03，PL08 等等；

<功能>: 000:input 001:output 010:S_JTAG_XX 011:Reserved 100:Reserved 101:Reserved 110:S_PLEINTX 111: IO Disable

<触发方式>: 0x0:Positive Edge(上升沿) 0x1:Negative Edge(下降沿) 0x2:High level(高电平) 0x3:Low level(低电平) ox4:Double Edge(边缘触发) others:Reserved

<驱动力>: 0：level 0 1:level 1 2:level 2 3:level 3 数值代表驱动等级，一般 0 表示 10mA,依次类推。

<输出电平>: 0：低电平 1：高电平

4.4. 读写寄存器值

熟悉了在内核中读写寄存器的值，调用 readl()和 writel()函数。

```
//ranchao test
  static unsigned int *reg_value;
  reg_value = ioremap(0x01f02c00,0);
  printk(KERN_ERR "1===reg_value_change_start : 0x%x===\n",readl(reg_value));
//ranchao test
```

```
 err = request_irq(bcmsdh_osinfo->oob_irq_num, wlan_oob_irq,
                   bcmsdh_osinfo->oob_irq_flags, "bcmsdh_sdmmc", bcmsdh);
//ranchao test
 reg_value = ioremap(0x01f02c00,0);
 printk(KERN_ERR "2===reg_value_change_end : 0x%x===\n",readl(reg_value));
//ranchao test
```

了解内存映射关系，以及 dump 节点的实现原理。

内核中有内存地址保护机制，无法直接使用 readl()函数去读写物理地址，于是用 ioremap()函数对物理地址做了个映射处理，即虚拟地址。readl()和 writel()只能读写虚拟地址，即 ioremap()函数处理过的地址。

```
ioremap(addr,offset)
将 addr 到 addr+offset 这段内存地址进行映射转换。
```

4.5. 相关操作说明

动态配置唤醒源：

echo wakeup_src para (1:enable)/(0:disable) > /sys/power/sunxi/wakeup_src

例如添加 PL5 为唤醒源:

echo 0x800000 0x165 1 > /sys/power/sunxi/wakeup_src

参数说明：0x800000 表示 CPUS_WAKEUP_GPIO 唤醒源　0x165 指定 PL5　1 表示 enable

系统进入休眠：

echo mem > /sys/power/state

定位某寄存器地址：

echo 0x01f02c00　> /sys/class/sunxi_dump/dump；一个地址

echo 0x01f02c00,0x01f02cff　> /sys/class/sunxi_dump/dump；一片地址

读取该（片）寄存器地址当前值：

cat dump

向该寄存器地址写入新的值：

echo 0x01f02c00 0x77126222　> /sys/class/sunxi_dump/write；

读取刚刚写入的值：

cat write

查看某一引脚的配置：

echo r_pio > /sys/kernel/debug/sunxi_pinctrl/dev_name

echo PL7 > /sys/kernel/debug/sunxi_pinctrl/sunxi_pin

cat　/sys/kernel/debug/sunxi_pinctrl/sunxi_pin_configure