全 志 科 技
Allwinner Technology

A20 红外遥控器与休眠唤醒配置

V1.0

2014-06-18

Revision History

Version	Date	Changes compared to previous issue
V1.0	2014-06-18	建立初始版本

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 Homlet A20 平台的红外配置与休眠唤醒的关系，以及休眠的功耗问题等。

1.2. 适用范围

介绍本模块设计适用于基于 A20 的平台。

1.3. 相关人员

预期读者为 standby 客户案支持人员和相关开发人员。

2. 红外遥控配置

2.1. 相关配置项

与红外遥控器相关的配置项如下：

1. 遥控器地址码

   默认情况下，红外驱动（drivers/input/keyboard/sun7i-ir.c）是开启了地址码验证功能，只有通过地址校验的红外信号才会发送到应用层（android），否则将丢弃该数据。地址码是通过 sys_config.fex 进行配置的，如：

```
[ir_para]

ir_used                 = 1

ir_rx                   = port:PB04<2><default><default><default>

ir_wakeup               = 0

power_key               = 0x57

ir_addr_code            = 0x9f00
```

   其中 ir_addr_code 配置的就是遥控器地址码，公版默认的地址是 0x9f00，请根据实际情况修改该值。

   其他相关的配置项：
   ir_used：是否启用红外遥控驱动，如果要使用红外遥控器，请设为 1，否则设为 0；
   ir_rx：　红外信号输入 pin，默认不需要修改；
   ir_wakeup：该项只有在无 MCU 方案中，使用红外唤醒功能需要配置为 1，否则一律为 0；
   power_key：红外遥控器 power key 对应的按键码，休眠后，红外唤醒需要使用。

2. 按键映射表；

   按键映射表是供 android 层进行识别按键用的，对于 A20 平台，映射表的位置在：

   /system/usr/keylayout/sun7i-ir.kl（盒子上的路径）

   在源码中的路径为：android/device/softwinner/sugar-ref001/sun7i-ir.kl

文件格式如下：

```
key 79    BACK                WAKE_DROPPED
key 22    MENU                WAKE_DROPPED
key 13    SEARCH              WAKE_DROPPED
key 2     DPAD_CENTER         WAKE_DROPPED
key 10    DPAD_DOWN           WAKE_DROPPED
key 67    DPAD_UP             WAKE_DROPPED
key 71    HOME                WAKE
```

对于每一个按键码（keycode）的描述的格式都固定：

key keycode FUNC WAKE/WAKE_DROPPED

其中 keycode 可以通过 getevent 获得；FUNC 为 android 定义好的的字符串，用于描述按键功能；WAKE/WAKE_DROPPED，选其一，其区别是：如果设备当前是休眠状态，如果此时接受到的按键是 WAKE 类型的，那么在唤醒机器之后，这个按键事件会同时通知到应用 app，如果是 WAKE_DROPPED 类型的按键，则仅仅是把机器唤醒，按键信息不会发送到 app。

2.2. 如何获取遥控器的地址码、按键值

2.2.1. 获取地址码

■ **方法一（适用 v2.0 之前的 sdk）：**

对应 v2.0 之前（包括 v2.0）的 sdk，只能够将 ir 驱动解码得到的按键值打印到控制台，截取遥控器地址码：

红外驱动的实现是在：drivers/input/keyboard/sun7i-ir.c 文件中，其中的 ir_packet_handler 函数是解码处理，可以通过在函数的最后将红外遥控器地址打印出来，具体处理示例如下：

```
static unsigned long ir_packet_handler(unsigned char *buf, unsigned long dcnt)

{

    ....

    ....

    // 添加地址码打印

    printk("addr code 0x%04x\n", code&0xffff);

    return code;

}
```

■ **方法二（适用 v2.1 之后的 sdk）：**

sdk v2.1 之后，加入了内核调试节点，可以方便获取到红外遥控器的地址，无需改动 ir 驱动代码，具体方法如下：

1. mount -t debugfs none /sys/kernel/debug（v2.1 sdk 默认 debugfs 已经挂载，可忽略此步骤）；
2. 随意按下遥控器按键，可以反复多次，确定 ir 已经接收到；
3. cat /sys/kernel/debug/sun7i_ir/ir_addr，将得到遥控器的地址，如：0xdf00；如果显示的地址码是 0x0000，表示地址码无效，请再次重复步骤 2、3。

注意：如果系统已经默认挂载了 debugfs，则无需执行第一步。

2.2.2. 获取按键值

在 sys_config.fex 文件中配置完成红外遥控器地址（如：ir_addr_code = 0x9f00）后，在 linux 命令行运行 getevent 命令，可以读取到按键对应的按键码；示例，按下 power key，getevent 的输出如下：


```
/dev/input/event3: 0001 0057 00000001

/dev/input/event3: 0000 0000 00000000
```

其中 0057 就是 power key 按键对应的遥控器按键码，需要注意的是：getevent 得到的按键码是 16 进制的，而 android/device/softwinner/sugar-ref001/sun7i-ir.kl 映射文件使用的值是 10 进制的，需要进行转换，对应该 power key 按键，配置如下：

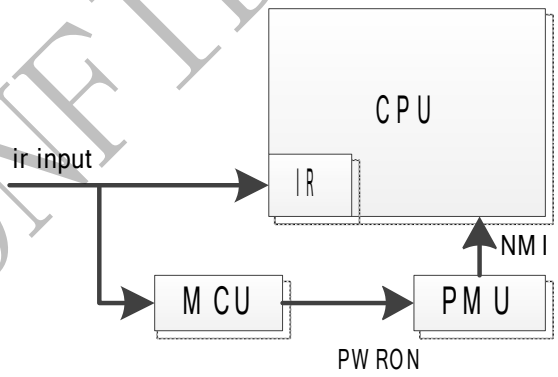
```
key 93    VOLUME_DOWN          WAKE

key 87    POWER                WAKE
```

按照相同的方法，获取其他按键并设置其映射。

2.3. 有 MCU 方案和无 MCU 方案区别

有无 MCU 的区别只是体现在 power key 的处理上：

1. 对应无 MCU 的方案，所有的按键解码都是由 CPU 的 ir 驱动进行解码并上报 input 信息；
2. 而对应有 MCU 方案，电源键进行了特殊处理，其输入信号如下图所示：



有 MCU 方案红外输入示意图

从示意图中可以看到，红外输入信号同时连接到 CPU 和 MCU，MCU 通过 1 个 IO 连接到 PMU 的 PWRON 上（这样设计的目的是为了做遥控开机的功能，关机情况下，通过拉低 PWRON 引脚 2s，可将 PMU 的输出唤醒，启动 CPU）。

处理如下：

1. 红外输入信号（ir input）同时输入到 A20 和 MCU，当有信号输入是，CPU 和 MCU 同时对红外信号进行解码；
2. 如果此时的按键非 power key，MCU 不做任何处理；CPU 则将解码得到的 key code 上报到

应用层；

3. 如果此时是 power key，MCU 将拉低 PWRON，PMU 检测得到 PWRON 变低，将产生中断，这个中断经由 NMI 输入到 CPU；PMU 驱动将向应用层上报长按键信息或者短按键信息，取决于红外按键的输入；这个由 PMU 驱动上报的按键信息，由 axp20-supplyer.kl 映射为 POWER 按键；
4. 如果此时是 power key，CPU 同样进行解码处理，上报信息。

由上述描述可知，当遥控器的 power key 被按下，PMU 驱动也上报了 POWER 按键信息，如果不对 ir 输入的按键信息进行处理，应用层将接受到两个 POWER 按键信息。

因此，对于有 MCU 方案，在 android/device/softwinner/sugar-ref001/sun7i-ir.kl 中将 POWER 按键映射进行了屏蔽：

```
key 93    VOLUME_DOWN          WAKE

#key 87   POWER                WAKE
```

因此，在配置按键映射的时候，请务注意区分有无 MCU 的 sun7i-ir.kl 配置。

2.4. 多遥控器支持

2.4.1. 多遥控器实现简介

多遥控器是在单遥控器的基础上扩展而来的。假设原有遥控器为 A，现在需要增加遥控器 B 的支持：处理如下，根据根据遥控器的地址码对遥控器进行区分，如果是 B 遥控器的地址码，则将按键值映射为 A 遥控器对应功能的按键值，接下来由 input_report_key 上报按键事件；即无论是 A 还是 B 遥控器，按键操作是一样的。

2.4.2. 数据结构描述

```
struct remap_array_t {
    int addr;
    const unsigned int *array;
};

struct multi_ir_remap_t {
    int count;
    int current_code;
    struct remap_array_t map[8];
};
```

- struct remap_array_t
  remap_array_t 用以描述一个遥控器，addr 保留遥控器的地址码，指针 array 指向该遥控器的按键映射表；
- struct multi_ir_remap_t
  multi_ir_remap_t 用以维护所有的遥控器映射，count 保留了当前支持的遥控器的个数；current_code 保留当前按键经过映射后的键值；数组 map，最多支持 8 个遥控器。

2.4.3. 映射表建立

映射表的目的是为了将 B 遥控器的按键键值映射到 A 遥控器对应按键的键值，这样对于上层而言都是同一个按键。映射表定义在 lichee/linux-3.4/drivers/input/keyboard/ir-keymap.h，如下所示：

```
static const unsigned int multi_ir_map0[]=

{

    [0x00] = 0xff,

    [0x01] = 0xff,

    [0x02] = 0xff,

    [0x03] = 0xff,

    [0x04] = 0xff,

    [0x05] = 0xff,

    [0x06] = 0xff,

     ....

     .....

    [0xFA] = 0xff,

    [0xFB] = 0xff,

    [0xFC] = 0xff,

    [0xFD] = 0xff,

    [0xFE] = 0xff,

    [0xFF] = 0xff

};
```


映射关系：假设 A 遥控器的向右方向键的键值为 0x43，B 遥控器的向右方向键是 0xCA，那么其映射就是[0xCA] = 0x43；其他按键按照同样的方法建立映射即可；其他无用的键值，则都填写为 0xFF。

2.4.4. 接收校验

为保证机器不受干扰（干扰可能来自非配对的遥控器、自然关等），驱动内部是对遥控器的地址码和按键值进行了校验，如果需要支持多遥控器，则需要修改其校验过程：

```
dummy_addr = code & 0xffff;

for(i=0; i<remap.count; i++){

    if(dummy_addr == remap.map[i].addr)

        break;

}



if(i>remap.count)

    return 0;
```

即遍历所有的遥控器，如果匹配到地址，则认为地址校验通过；否则返回 0，丢弃该按键。

2.4.5. 按键重映射

就是按照功能划分，将后续遥控器的按键码映射为第一个遥控器对应的按键码；重映射过程，通过按键映射表实现：

```
static int ir_code_remap(unsigned long code)

{

    int i;

    int rev_addr = (code & 0xffff);

    int rev_code = (code>>16)&0xff;
```



```
    for(i=0; i<remap.count; i++){
        if(rev_addr == remap.map[i].addr) {
            remap.current_code = *((remap.map[i].array)+rev_code);
            break;
        }
    }
    return 0;
}
```

2.4.6. 使用配置

如果需要支持多遥控器，需要按照以下步骤进行配置：

1. 在 lichee/linux-3.4/drivers/input/keyboard/ir-keymap.h 中添加新遥控器的映射表，映射表映射关系按照 2.4.3 所述方法操作；
2. 初始化数据结构 struct multi_ir_remap_t，下面是 2 个遥控器的示例：

```
static int multi_ir_init(void)
{
    // 遥控器个数
    remap.count = 2;

    // 第一个遥控器配置
    remap.map[0].addr = ir_addr_code;
```



```
    remap.map[0].array = ir_keycodes;



    // 第二个遥控器配置

    remap.map[1].addr = 0xfd01;

    remap.map[1].array = multi_ir_map0;

    return 0;

}
```

3. 支持多遥控器的 demo：

2.4.7. 注意事项

1. 无 MCU 方案：
   无 MCU 方案可以直接按照 2.4.1 所述方法进行配置，无其他要求。

2. 有 MCU 方案：
   有 MCU 方案，需要要求 MCU 固件能够识别第二个遥控器的地址码和 power 按键，否则无法使用遥控器开机唤醒。

3. 休眠唤醒配置

目前 A20 sdk 是支持 super standby 和 normal standby 两种休眠模式，两种休眠模式对比如下表：

super standby/normal standby 对比

standy_mode	CPU 是否断电	sys_config.fex	PMU 要求	MCU 要求
super	断电	standby_mode = 1	axp209	必须
normal	不断电	standby_mode = 0	axp152/axp209	可选

super standby 与 normal standby 最明显的区别是 CPU 是否断电；

1. 对于 super standby，休眠时 CPU 电源是关掉的，CPU 不耗电；
2. 对于 normal standby，休眠时 CPU 电源并没有关闭，CPU 进入 WFI（wait for interrupt）模式。

需要注意的是，super standby 是需要硬件支持的：

第一：PMU 芯片必须使用 axp209，axp152 不支持 super standby；

第二：硬件上必须配置 MCU，如果是无 MCU 方案，不能使用 super standby。

3.1. 休眠模式配置

目前代码是同时支持 super standby 和 normal standby 的，通过 sys_config.fex 区分 super standby 和 normalstandby，与休眠模式相关的配置示例如下：

```
[pm_para]

standby_mode            = 0

usbhid_wakeup_enable = 0
```

上述例子表示使用的是 normal standby（standby_mode = 0）。

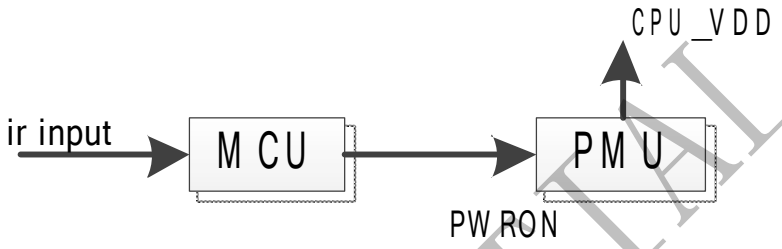
usbhid_wakeup_enable 是配置是否使能 usb 唤醒功能；usb 唤醒功能是指通过点击连接到盒子上的 usb 鼠标或者 usb 键盘，可以将系统唤醒。usb 唤醒功能要求当前的休眠模式是 normal standby，否则无法实现 usb 唤醒。

3.2. 唤醒实现

super standby 情况下可以通过红外电源按键、板上电源按键唤醒系统；normal standby 的唤醒源则较多，包括外部中断、usb 插入、rtc 中断等，下面分别介绍两种不同模式下的唤醒实现。

3.2.1. super standby 唤醒

super standby 进入休眠后，CPU 是关闭的，PMU 处于待机模式；唤醒系统的实现是通过将 PMU 唤醒，恢复电源输出，给 CPU 供电，进而恢复 CPU 关闭前的状态，从而达到唤醒系统的目的。因此 super standby 下的唤醒，首先需要将 PMU 唤醒，恢复电源输出。因为休眠时 CPU 是关闭的，为了实现红外遥控唤醒，只能通过借助外部 MCU 进行红外解码，判断红外遥控器按键，进而唤醒 PMU 输出（这也是为什么 super standby 必须配上 MCU 的原因，否则休眠之后无法通过红外遥控器唤醒），其唤醒过程示意图：



红外唤醒示意图

唤醒过程如下：

1. 按下遥控器；
2. MCU 接收到红外接收管发来的脉冲信号，MCU 进行解码；
3. 如果解码得到的是电源键，MCU 拉低 PMU 的 PWRON 脚，唤醒 PMU 输出；
4. 如果解码得到的不是电源键，MCU 什么也不做。

注意事项：

1. MCU 解码的时候是会匹配红外遥控器地址已经电源键键值，即 MCU 和红外遥控器是配套的，否则无法唤醒 PMU；因为 MCU 里面的代码是固化的，不能更改，在量产前必须进行确认；

3.2.2. normal standby 唤醒

normal standby 有两种情况：有 MCU 和无 MCU；有 MCU 的方案无需特别配置，无 MCU 方案需要配置 sys_config.fex 的 ir_wakeup 选项，示例如下：

```
[ir_para]
ir_used                 = 1
ir_rx                   = port:PB04<2><default><default><default>
```



```
ir_wakeup           = 0

power_key           = 0x57

ir_addr_code        = 0x9f00
```

其中 ir_wakeup = 0，表示是有 MCU 的方案，ir_wakeup = 1，表示为无 MCU 的方案，请根据实际情况进行配置，否则无法通过红外遥控器进行唤醒。

4. standby 模式及其功耗

4.1. 功耗对比

A20 SDK 支持两种休眠模式：normal standby 和 super standby；这两种模式最根本的区别在于 CPU 是否断电；normal standby 模式下，CPU 进入 WFI（wait for interrupt）模式，供电保持；super standby 模式下，直接将 CPU 的电源关掉。

normal/super standby 对比

standy_mode	CPU 是否断电	最低功耗
super	断电	10mA~15mA @ 5V
normal	不断电	25mA @ 5V

注意：上表的最低功耗数据是有条件的：

1. 休眠的时候外围器件都是关闭电源的，包括 wifi/emac/usb hub/音频功放芯片等；
2. usb 唤醒功能关闭；
3. dram 进入自刷新模式；
4. 上述数据只是盒子的功耗，未考虑火牛的静态功耗以及转换效率。

4.2. 功耗分析步骤

如果功耗出现异常，可根据以下步骤进行定位：

1. 测量休眠状态下 PMU 各路（DCDC/LDO）输出电压，观察输出是否在合理范围内；
2. 测量外围模块供电情况，如：wifi、emac、音频功放、usb hub 芯片；
3. 如果是 super standby 模式，确认 CPUVDD 是否关闭；
4. 是否配置了 usb 唤醒功能，确定是否需要支持；
5. 确定 dram 是否进入自刷新模式。

4.3. 建议

A20 支持 AXP209 和 AXP152 两种 PMU 芯片，但是 super standby 模式只有 AXP209 方案才支持；因此在原理图设计初期就要明确整机功耗的规格，是否有必要支持 super standby；如果要支持 super standby，电源芯片必须使用 AXP209，避免后续因功耗问题导致改板。

5. 休眠唤醒 Debug

如果出现休眠后不能唤醒的情况，请按照如下步骤进行处理：

1. 检查红外遥控器相关配置是否正确：
   a) sys_config.fex 的 ir_addr_code 是否正确配置；
   b) sys_config.fex 的 power_key 是否正确配置；
   c) 检查盒子方案是 super standby 还是 normal standby，配置是否正确；
2. 检查休眠后，红外接收是否正确：使用示波器测量红外接收管的输出管脚，当按下电源键时，是否有脉冲序列输出；
3. 如果在确认上述配置都正确后，仍然无法解决问题，则需要将 bug 反馈，有效的反馈信息应该包含如下内容：
   a) 出现问题的 sdk 版本、休眠模式（super or normal）、sys_config.fex 文件
   b) 打印等级为 8 的串口 log；

6. Declaration

