

Unisoc Confidential For hiar

Kernel 4.14 RTC ALARM 客制化指导手册

文档版本
发布日期

V1.1
2020-08-11

版权所有 © 紫光展锐科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

Unisoc Confidential For hiar

紫光展锐科技有限公司



前言

概述

本文档主要介绍了基于 **kernel4.14** 设置时间和闹钟的流程，以及一些常见问题的定位方法。

读者对象


本文档主要适用于需要设置时间和闹钟的客户。

缩略语

缩略语	英文全名	中文解释
RTC	Real Time Clock	实时时钟

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2019-08-09	初稿。
V1.1	2020-08-11	<ul style="list-style-type: none">文档名由《UNISOC RTC_ALARM Customization Guide For Kernel4.14》修改为《Kernel 4.14 RTC ALARM 客制化指导手册》。内容优化、格式更新

关键字

RTC、ALARM、DTS、CONFIG、DRIVER。

Unisoc Confidential For hiar

目 录

1 概述	1
2 RTC/ALARM 的使用	2
2.1 相关代码及流程介绍	2
2.1.1 相关代码位置	2
2.1.2 流程介绍	2
2.2 客制化配置清单	5
2.2.1 编译宏配置	5
2.2.2 DTS 配置	5
3 常见问题	6
3.1 关机充电闹钟失效	6
3.2 设置的闹钟类型错误	7
4 客制化实例	9

Unisoc Confidential For hiar

图目录

图 2-1 设置时间流程	2
图 2-2 开机状态下设置闹钟流程	3
图 2-3 设置关机闹钟流程	4
图 2-4 编译宏配置 1	5
图 2-5 编译宏配置 2	5
图 2-6 DTS 配置	5
图 3-1 关机时串口 log	6
图 3-2 关机充电 log	7
图 3-3 闹钟到期后 log	7
图 3-4 clock 类型	8
图 3-5 关机闹钟失效 log 分析	8
图 4-1 清除 RTC 接口	9
图 4-2 RTC 复位	9

1 概述

RTC 模块的主要功能如下：

- RTC 为系统提供可靠的时间，即使系统关机后，RTC 也可以通过电池供电一直运行下去。
- 通过 RTC 设备读写 RTC 时间。
- 通过 `alarmtimer` 来设置 RTC ALARM。
- `timerfd` 提供一些系统调用来设置 `alarmtimer`。

Unisoc Confidential For hiar

2 RTC/ALARM 的使用

2.1 相关代码及流程介绍

2.1.1 相关代码位置

- drivers rtc/rtc-dev.c
- kernel/time/alarmtimer.c
- fs/timerfd.c
- drivers rtc/rtc-sc27xx.c

2.1.2 流程介绍

2.1.2.1 设置时间

设置时间的流程如图 2-1 所示。

图2-1 设置时间流程

```
System_call(/dev/rtc0)
    rtc_dev_ioctl
        rtc_set_time(rtc, &tm);
            err = rtc->ops->set_time(rtc->dev.parent, tm);
                sprd_rtc_set_time; // 最终调用到平台的 rtc 设置时间函数
```

2.1.2.2 设置闹钟

Google 最早提供 /dev/alarm 设备节点来设置 alarm，现已废除。目前的实现方式是用 timerfd 来实现，具体的设置流程如下：

- 开机状态下设置闹钟

图2-2 开机状态下设置闹钟流程

timerfd_settime

```
ret = do_timerfd_settime(ufd, flags, &new, &old);  
  
ret = timerfd_setup(ctx, flags, new);  
  
alarm_start(&ctx->t.alarm, texp);  
  
alarmtimer_enqueue(base, alarm);  
  
hrtimer_start(&alarm->timer, alarm->node.expires, HRTIMER_MODE_ABS);
```

- 设置关机闹钟

关机的时候，最终会调用 shutdown 的接口，具体如图 2-3。

Unisoc Confidential For hiar

图2-3 设置关机闹钟流程

```
void alarmtimer_shutdown(struct platform_device *pdev)
{
    ktime_t min = 0, early = ktime_set(120, 0), now;
    int ret, alarm_type = ALARM_NUMTYPE, i;
    struct rtc_device *rtc;
    struct rtc_wkalrm alarm;
    unsigned long flags;
    struct rtc_time tm;

    rtc = alarmtimer_get_rtcddev();
    /* If we have no rtcddev, just return */
    if (!rtc)
        return;

    /* Find the soonest timer to expire */
    for (i = ALARM_POWERON; i < ALARM_NUMTYPE; i++) { /* 遍历关机闹钟及定时开机闹钟 */
        struct alarm_base *base = &alarm_bases[i];
        struct timerqueue_node *next;
        ktime_t delta;

        spin_lock_irqsave(&base->lock, flags);
        next = timerqueue_getnext(&base->timerqueue);
        spin_unlock_irqrestore(&base->lock, flags);
        if (!next)
            continue;

        delta = ktime_sub(next->expires, base->gettime());
        if (i == ALARM_POWEROFF_ALARM &&
            ktime_compare(delta, early) <= 0) { /* 设置的闹钟要大于2min, 否则不会设置到rtc */
            pr_info("Poweroff alarm is less than two minutes.\n");
        } else if (!min || ktime_compare(delta, min) <= 0) { /* 记录最早到期的闹钟 */
            min = delta;
            alarm_type = i;
        }
    }

    if (min == 0) {
        pr_info("No poweroff alarm found\n");
        return;
    } else if (ktime_to_ms(min) < 10 * MSEC_PER_SEC) {
        pr_warn("Don't need to set poweroff alarm due to less than 10s\n");
        return;
    }

    /* Setup an rtc timer to fire that far in the future */
    rtc_read_time(rtc, &tm);
    now = rtc_tm_to_ktime(tm);
    now = ktime_add(now, min);

    if (alarm_type == ALARM_POWEROFF_ALARM)
        now = ktime_sub(now, early);

    tm = rtc_ktime_to_tm(now);
    pr_info("Poweroff alarm: %d-%d-%d %d:%d:%d\n", tm.tm_year + 1900,
        tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec);

    alarm.time = tm;
    alarm.enabled = 1;

    ret = rtc_set_alarm(rtc, &alarm); /* 设置到rtc中 */
    if (ret < 0)
        pr_err("Power off alarm setting error %d\n", ret);
}
```

2.2 客制化配置清单

2.2.1 编译宏配置

- 编译宏在 driver/rtc/Kconfig 中定义，如图 2-4。

图2-4 编译宏配置 1

```
config RTC_DRV_SC27XX
    tristate "Spreadtrum SC27xx RTC"
    depends on MFD_SC27XX_PMIC || COMPILE_TEST
    help
        If you say Y here you will get support for the RTC subsystem
        of the Spreadtrum SC27xx series PMICs. The SC27xx series PMICs
        includes the SC2720, SC2721, SC2723, SC2730 and SC2731 chips.

        This driver can also be built as a module. If so, the module
        will be called rtc-sc27xx.
```

- 在相应的 defconfig 文件中配置这个宏的开关，配置为 y 表示“开”，不配置表示“关”，如图 2-5。

图2-5 编译宏配置 2

```
# CONFIG_RTC_DRV_V3020 is not set
CONFIG_RTC_DRV_SC27XX=y
# CONFIG_RTC_DRV_ZYNQMP is not set
```

2.2.2 DTS 配置

在 PMIC 的 dtsi 文件中配置，具体如图 2-6。

图2-6 DTS 配置

```
&adi_bus {
    sc2721_pmic: pmic@0 {
        .....
        .....
        rtc@200 {
            compatible = "sprd,sc27xx-rtc", "sprd,sc2721-rtc";
            reg = <0x200>;
            interrupt-parent = <&sc2721_pmic>;
            interrupts = <1 IRQ_TYPE_LEVEL_HIGH>;
        };
        .....
        .....
    }
}
```

3 常见问题

3.1 关机充电闹钟失效

这种问题底层出问题的概率很小，上层出问题的概率比较大，但是一般都是从底层查起。

现象

设置一个闹钟（一定是大于 2min），然后关机，插上 USB 线，闹钟到期后不会重启系统。

分析思路

分 3 步：

1. 关机时有没有设到 RTC
2. 手机充电的时候系统有没有重新开机
3. 闹钟到期后，手机有没有重启

相关 log 及分析

1. 关机的时候，串口会有如图 3-1log，说明有闹钟设置到 RTC 了。

图3-1 关机时串口 log

```
[ 158.523860] c0 init: powerctl_shutdown_time_ms:9084:2
[ 158.528856] c0 init: Reboot ending, jumping to kernel
[ 158.535716] c0 mmc0: clock 0Hz busmode 2 powermode 0 cs 0 Vdd 0 width 1 timing 0
[ 158.543156] c0 sprd_signal_voltage_on_off(sdio_emmc) there is no signal voltage!
[ 158.552434] c0 Poweroff alarm: 2019-8-14 2:16:0 ==> 说明有闹钟设置到rtc了
[ 158.629795] c2 CPU2: update max cpu_capacity 1024
[ 158.649819] c0 Retrying again to check for CPU kill
```

2. 插入充电器的时候，手机会从关机状态开机，并进入关机充电模式，如图 3-2。

图3-2 关机充电 log

```
[ 158.990645] c0 CPU1 killed.
[ 158.993997] c0 Disabling non-boot CPUs ...
[ 158.998031] c0 reboot: Power down
[ 159.009015] c0 Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000000
[ 159.UUUUUUMCI
ERS
ECS
ROM
RIO
DDR init start
Before swap
CS0_MR8:0x0000001F
Swap later
CS0_MR8:0x0000001F
CS1_MR8:0x0000001F
DDR init OK

chip id : 0x32000000 0x50696B65 0x1
pmic id : 0xA001 0x2720NOTICE: SP_MIN: v1.4-79e1baf(debug)
NOTICE: SP_MIN: Built : 00:16:00, Aug 7 2019

U-Boot 2015.07-01223-gada92bb (Aug 07 2019 - 00:15:19 +0800)

U-Boot code: 9F000000 -> 9F0A1000 BSS: -> 9F24E000
I2C: i2c0, sprd_i2c_init() freq=100000
ready
monitor len: 0024E000
ramsize: 7FE00000
TLB table from ffd00000 to ffd04000
```

3. 闹钟到期后，手机会重启，有如图 3-3 关键 log。

图3-3 闹钟到期后 log

```
[ 54.059484] c0 CPU0: update max cpu_capacity 1024
[ 54.119556] c0 Retrying again to check for CPU kill
[ 54.124369] c0 CPU1 killed.
[ 54.179481] c0 CPU0: update max cpu_capacity 1024
[ 54.210066] c0 CPU2 killed.
[ 54.320106] c0 CPU3 killed.
[ 54.323294] c0 reboot: Restarting system with command 'alarm' ==> 说明alarm到期，重启了
```

如果以上三点都满足的话，可以基本说明底层 RTC 没有问题，RTC 闹钟配置正常并能够正确触发。需要继续分析其他方面。

3.2 设置的闹钟类型错误

内核中的 clock 类型主要有以下几种：

图3-4 clock 类型

```

/*
 * The IDs of the various system clocks (for POSIX.1b interval timers):
 */
#define CLOCK_REALTIME 0
#define CLOCK_MONOTONIC 1
#define CLOCK_PROCESS_CPUTIME_ID 2
#define CLOCK_THREAD_CPUTIME_ID 3
#define CLOCK_MONOTONIC_RAW 4
#define CLOCK_REALTIME_COARSE 5
#define CLOCK_MONOTONIC_COARSE 6
#define CLOCK_BOOTTIME 7
#define CLOCK_REALTIME_ALARM 8
#define CLOCK_BOOTTIME_ALARM 9
/*
 * The driver implementing this got removed. The clock ID is kept as a
 * place holder. Do not reuse!
 */
#define CLOCK_SGI_CYCLE 10
#define CLOCK_TAI 11

#define CLOCK_POWEROFF_WAKE 12
#define CLOCK_POWERON_WAKE 13
#define CLOCK_POWEROFF_ALARM 14

```

- 设置唤醒的就用 CLOCK_POWERON_WAKE
- 设置关机闹钟就用 CLOCK_POWEROFF_ALARM

在设置闹钟时不能随便设置同一个类型的闹钟，会导致其中的一些闹钟失效。举例说明：由于上层设置了两个同类型的闹钟（其中一个应该是做定时用的），导致真正的关机闹钟失效。如图 3-5 所示。

图3-5 关机闹钟失效 log 分析

```

[ 88.756420] c3 CPU3: Booted secondary processor
[ 88.756442] c0 [is_chip_true]chip does not need to disable/enable pmu irq
[ 88.772770] c2 timerfd_setup: clockid = 14 //上层通过系统调用先设了一个14:6:0 的闹钟, 闹钟类型是
poweroff_alarm,当前系统时间是14:0:0左右(看第三段log获取系统时间来反推的)
[ 88.772780] c2 timerfd_setup: alarm_timer_time: 2018-12-13 14:6:0
[ 88.789146] c1 [SPRD_KPLED_INFO]sprd_kpled_enable
[ 88.794138] c1 [SPRD_KPLED_INFO]sprd_kpled_set_brightness:led->run_mode = 0

[ 99.426398] c2 CPU2: Booted secondary processor
[ 99.426418] c0 [is_chip_true]chip does not need to disable/enable pmu irq
[ 99.450420] c0 timerfd_setup: clockid = 14 //上层通过系统调用又设了一个14:0:31的闹钟, 闹钟类型是
poweroff_alarm
[ 99.450429] c0 timerfd_setup: alarm_timer_time: 2018-12-13 14:0:31
[ 99.546020] c0 !! lit:we gonna plugin cpu3 !!
[ 99.550712] c3 CPU3: Booted secondary processor

[ 100.816058] c1 regulator regulator.1: regu 0xce9f60a8 (vddcore) turn on, return 1
[ 100.824777] c1 [sprd-adj] panel_suspend panel suspend OK
[ 100.830790] c1 list next->expires: 2018-12-13 14:0:31
[ 100.835767] c1 system time: 2018-12-13 14:0:27 //当前的系统时间
[ 100.840224] c1 Poweroff alarm is less than two minutes. //由于系统时间和当前最早到期的闹钟时间小于2min, 所以设到rtc中失败
[ 100.845409] c1 disable rtc alarm.

```

4 客制化实例

- 刷机的时候，RTC 会被复位，RTC 里面的所有信息会被清除。清除 RTC 的接口在 uboot 里面，路径是 u-boot15/drivers/rtc/sprd-rtc.c，具体如图 4-1。

图4-1 清除 RTC 接口

```
int sprd_clean_rtc(void)
{
    int err;

    sprd_rtc_init();

    ANA_REG_AND(ANA_RTC_INT_EN, ~(RTC_INT_ALL_MSK)); // disable all interrupt

    CLEAR_RTC_INT(RTC_INT_ALL_MSK);
    sprd_rtc_set_sec(0);
    sprd_rtc_set_alarm_sec(0);
    printf("now time sec %lu\n", sprd_rtc_get_sec());
    printf("now alarm sec %lu\n", sprd_rtc_get_alarm_sec());
    return 0;
}
```

- 刷机的 RTC 复位，需要在 u-boot15/common/cmd_download.c 中配置，如图 4-2。

图4-2 RTC 复位

```
int do_download(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
    .....
    sprd_clean_rtc();
    .....
}
```

在没有 android 上层的 linux 系统上，可通过系统调用来设置一个关机闹钟，本文仅提供 sample code 供参考，代码如下：

```

void printTime()
{
    struct timeval tv;
    gettimeofday(&tv, NULL);
    printf("printTime: current time:%ld.%ld ", tv.tv_sec, tv.tv_usec);
}

#define CLOCK_REALTIME          0
#define CLOCK_BOOTTIME          7
#define CLOCK_REALTIME_ALARM    8
#define CLOCK_BOOTTIME_ALARM    9
#define CLOCK_POWEROFF_WAKE     12
#define CLOCK_POWERON_WAKE      13
#define CLOCK_POWEROFF_ALARM    14
// ./xxxxx_timer_test 30 0 1 &
int main(int argc, char *argv[])
{
    struct timespec now;

    int tot_exp;
    if (clock_gettime(CLOCK_REALTIME, &now) == -1)
        handle_error("clock_gettime");

    struct itimerspec new_value;
    new_value.it_value.tv_sec = now.tv_sec + atoi(argv[1]);
    new_value.it_value.tv_nsec = now.tv_nsec;
    new_value.it_interval.tv_sec = atoi(argv[2]);
    new_value.it_interval.tv_nsec = 0;

    // int fd = timerfd_create(CLOCK_REALTIME, 0);
    // int fd = timerfd_create(CLOCK_POWEROFF_ALARM, 0);

    int fd = timerfd_create(CLOCK_BOOTTIME, 0);
    if (fd == -1)
        handle_error("timerfd_create");

    if (timerfd_settime(fd, TFD_TIMER_ABSTIME, &new_value, NULL) == -1)
        handle_error("timerfd_settime");

    printTime();
    printf("timer started\n");

    for (tot_exp = 0; tot_exp < atoi(argv[3]);)
    {
        uint64_t exp;
        ssize_t s = read(fd, &exp, sizeof(uint64_t));
        if (s != sizeof(uint64_t))
            handle_error("read");

        tot_exp += exp;
        printTime();
        printf("read: %llu; total=%llu\n", exp, tot_exp);
    }

    exit(EXIT_SUCCESS);
}

```