SUNXI RTC

使用说明文档

文档版本号: SDK-V1.0

发布日期: 2019-03-30

商标声明

、全志和其他全志商标均为珠海全志科技股份有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受全志公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，全志公司对本文档内容不做任何明示或默示的声明或保证。
由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

文档履历

版本号	日期	制/修订人	内容描述
V1.0	2019-03-30	Allwinner	V316 初始化版本

目 录

表目录

图目录

1 概述

1.1 编写目的

该使用文档介绍了 RTC 的使用方法

1.2 适用范围

本文档适用于 Linux3.4 及以上版本内核的平台

1.3 相关人员

本文档适用于使用 RTC 的人员

2 模块介绍

2.1 模块功能介绍

RTC(real time clock) 实时时钟，主要作用是给 Linux 系统提供时间。RTC 因为是电池供电的，所以掉电后时间不丢失。Linux 内核把 RTC 用作“离线”的时间与日期维护器。当 Linux 内核启动时，它从 RTC 中读取时间与日期，作为基准值。在运行期间内核完全抛开 RTC，以软件的形式维护系统的当前时间与日期，并在需要时将时间回写 RTC 芯片。另外，如果 RTC 提供了 IRQ 中断并且可以定时，那么 RTC 还可以作为内核睡眠时唤醒内核的闹钟。应用程序可以用 RTC 提供的周期中断做一些周期的任务。

RTC 模块体系结构

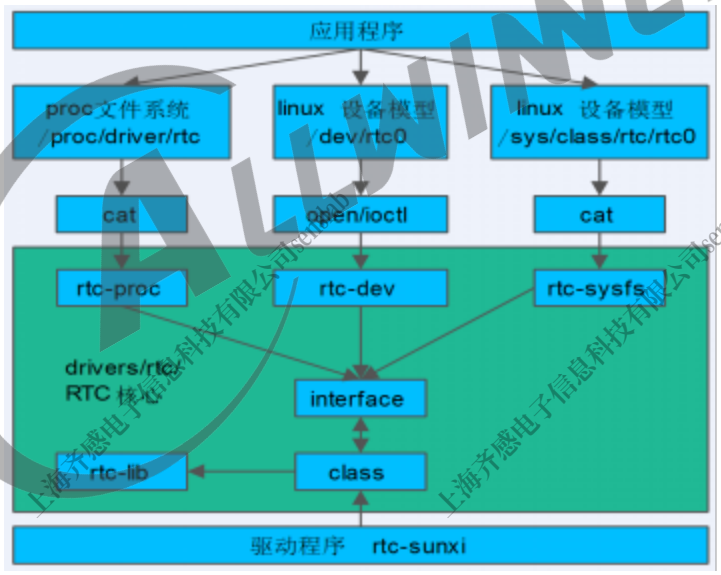

图 2-1: figure1.png

2.2 相关术语介绍

无

2.3 模块配置介绍

进入内核源码目录，执行 make ARCH=arm(arm64) menuconfig 进入配置主界面，并按以下步骤操作：

首先，选择 Real Time Clock，如下图所示：

```
    Broadcom specific AMBA  --->
    Multifunction device drivers  --->
[ ] Voltage and Current Regulator Support  ----
< > Multimedia support  ----
    Graphics support  --->
< > Sound card support  ----
    HID support  --->
[ ] USB support  ----
< > Ultra Wideband devices  ----
< > MMC/SD/SDIO card support  ----
< > Sony MemoryStick card support  ----
[ ] LED Support  ----
[ ] Accessibility support  ----
[ ] EDAC (Error Detection And Correction) reporting  ----
[*] Real Time Clock  --->
[*] DMA Engine support  --->
```

图 2-2: figure2.png

然后选择 Allwinner sunxi RTC

```
< >   Maxim/Dallas DS1553
< >   Dallas/Maxim DS1685 Family
< >   Maxim/Dallas DS1742/1743
< >   Maxim/Dallas DS2404
< >   Simtek STK17TA8
< >   ST M48T86/Dallas DS12887
< >   ST M48T35
< >   ST M48T59/M48T08/M48T02
< >   Oki MSM6242
< >   TI BQ4802
< >   Ricoh RP5C01
< >   EM Microelectronic V3020
< >   Xilinx Zynq Ultrascale+ MPSoC RTC
      *** on-CPU RTC drivers ***
<*>   Allwinner sunxi RTC
< >   Freescale SNVS RTC support
```

图 2-3: figure3.png

3 数据结构

rtc_time
struct rtc_time {
int tm_sec;
int tm_min;
int tm_hour;
int tm_mday;
int tm_mon;
int tm_year;
int tm_wday;
int tm_yday;
int tm_isdst;
};：某天是一个星期的第几天，星期日为
tm_wday0：某天是一年中的第几天
tm_yday 第一天为0：判断是否是夏令时
tm_isdst
1 是夏令时
0 不是夏令时
-1 由（）当前系统是否设置为夏令时kmtime

rtc_wkalrm
struct rtc_wkalrm{
unsigned char enabled;
unsigned char pending;
struct rtc_time time;
};:
enabled0 = alarm disable 1 = alarm enabled:
pending0 = alarm not pending 1 = alarm pending

4 接口介绍

4.1 ioctl 控制 RTC

应用层可以通过 ioctl 控制 RTC，设备节点是/dev/rtc0

```
int ioctl(int fd, ind cmd, …)
```

RTC 中的 ioctl 控制命令有:
RTC_ALM_SET：设置闹钟时间
RTC_ALM_READ：读取闹钟时间
RTC_RD_TIME：读取时间
RTC_SET_TIME：设置时间
RTC_PIE_ON：使能 2^N HZ 的周期中断
RTC_PIE_OFF：禁止 2^N HZ 的周期中断
RTC_AIE_ON：使能闹钟中断
RTC_AIE_OFF：禁止闹钟中断
RTC_UIE_ON：使能中断更新
RTC_UIE_OFF：禁止中断更新
RTC_IRQP_SET：设置 2^N HZ 的周期中断的频率
RTC_IRQP_READ：读取 2^N HZ 的周期中断的频率
RTC_WKALM_SET: 设置闹钟时间
RTC_WKALM_READ: 读取闹钟时间，等同于 RTC_ALM_READ

其中，ioctl 第三个参数
当进行设置或者读取 RTC 时间和闹钟时间，struct rtc_time 或 struct rtc_wkalrm 作为 ioctl 的第三个参数，用来传进设置的时间和返回读取的时间

4.2 RTC 节点信息

在 sys 文件系统的 sys/class/rtc/rtc0 和 proc 文件系统的/proc/driver/rtc 中记录了 RTC 的信息，可以使用 cat 命令获取相关信息。
name：获取 RTC 的名字

date：获取日期
time：获取时间
since_epoch：获取从 1970/1/1 00:00:00:00 到现在的秒数
max_user_freq：获取 2^N HZ 的周期中断的最大频率
hctosys：获取系统时间
wakealarm：获取闹钟唤醒时间

5 使用范例

5.1 设置和获取 RTC 时间

```
int case_rtc_time(char *str)
{
    int iret = 0;
    int     rtc = -1;
    char    time_str[32] = {0};
    struct rtc_time rtc_tm = {0};
    struct rtc_time rtc_tm_temp = {0};

    strncpy(time_str, str, 32);

    /* parse para to rtc_time */
    if (0 != str_to_rtc_time(time_str, &rtc_tm)) {
        DBG("str_to_rtc_time failed\n");
        return -EINVAL;
    }

    /* open rtc device */
    rtc = open(RTC_DEVICE_NAME, O_RDWR);
    if (rtc < 0) {
        ERR("open rtc device %s failed\n", RTC_DEVICE_NAME);
        return -ENODEV;
    }

    /* set rtc time */
    //rtc_tm.tm_isdst = 0;
    rtc_tm.tm_year -= 1900; /* must sub 1900 */
    rtc_tm.tm_mon  -= 1;    /* must be in 0 ~ 11 */
    if(ioctl(rtc, RTC_SET_TIME, &rtc_tm) < 0) {
        ERR("RTC_SET_TIME failed\n");
        iret = -EPERM;
        goto end;
    }

    /* get rtc time */
    if(ioctl(rtc, RTC_RD_TIME, &rtc_tm_temp) < 0) {
        ERR("RTC_RD_TIME failed\n");
```

```
        iret = -EPERM;
        goto end;
    }

    /*
     * Only the values that we set to the RTC are checked. We leave
     * tm_wday, tm_yday and tm_isdst untouched. Because the values
     * of these fields depend on the RTC hardware and driver, they
     * may be read from hardware, or calculated by the software,
     * or left undefined.
     */
    if ((rtc_tm.tm_sec != rtc_tm_temp.tm_sec)
        | (rtc_tm.tm_min != rtc_tm_temp.tm_min)
        | (rtc_tm.tm_hour != rtc_tm_temp.tm_hour)
        | (rtc_tm.tm_mday != rtc_tm_temp.tm_mday)
        | (rtc_tm.tm_mon != rtc_tm_temp.tm_mon)
        | (rtc_tm.tm_year != rtc_tm_temp.tm_year)) {
        ERR("Failed to set target time.\n");
        iret = -1;
    }

end:
    /* close rtc device */
    if ((rtc >= 0) & (close(rtc) < 0))
        ERR("close %s failed\n", RTC_DEVICE_NAME);

    return iret;
}
```

5.2 设置和获取闹钟

```
int case_rtc_alarm(char *str)
{
    int iret = 0;
    int     rtc = -1;
    char    time_str[260] = {0};
    struct rtc_time rtc_tm, rtc_tm_temp;

    strcpy(time_str, str);

    /* parse para to rtc_time */
    if(0 != str_to_alarm_time(time_str, &rtc_tm)) {
```

```
        ERR("str_to_alarm_time failed\n");
        return -EINVAL;
    }

    /* open rtc device */
    rtc = open(RTC_DEVICE_NAME, O_RDWR);
    if(rtc < 0) {
        ERR("open rtc device %s failed\n", RTC_DEVICE_NAME);
        return -ENODEV;
    }

    /* set alarm time */
    if(ioctl(rtc, RTC_ALM_SET, &rtc_tm) < 0) {
        ERR("RTC_ALM_SET failed\n");
        iret = -EPERM;
        goto end;
    }

    if (ioctl(rtc, RTC_AIE_ON) < 0) {
        ERR("RTC_AIE_ON failed!\n");
        iret = -EPERM;
        goto end;
    }

    /* get alarm time */
    if(ioctl(rtc, RTC_ALM_READ, &rtc_tm_temp) < 0) {
        ERR("RTC_ALM_READ failed\n");
        iret = -EPERM;
        goto end;
    }

    if ((rtc_tm.tm_hour != rtc_tm_temp.tm_hour)
        | (rtc_tm.tm_min != rtc_tm_temp.tm_min)
        | (rtc_tm.tm_sec != rtc_tm_temp.tm_sec)) {
        ERR("Failed to set target alarm.\n");
        iret = -1;
    }

end:
    /* close rtc device */
    if(rtc >= 0) {
        if(close(rtc) < 0)
            ERR("close %s failed\n", RTC_DEVICE_NAME);
    }
    return iret;
```

```
}
```

6 Declaration

This document is the original work and copyrighted property of Allwinner Technology (‘“Allwinner”‘). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.