Linux RTC
开发指南

版本号: 1.8
发布日期: 2025.3.15

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.07.28	XAA0270	初始版本
1.1	2022.11.07	XAA0190	1.2 小节添加适用范围
1.2	2022.12.07	XAA0190	3.1 小节修改 dtsi 名称
1.3	2023.02.17	XAA0190	4.1 小节添加 rtc ioctl 指令及功能描述
1.4	2023.03.16	XAA0190	修改全文字体格式，更新 6.1 小节图片
1.5	2023.04.23	XAA0190	3.1 小节添加 menuconfig 选项信息
1.6	2024.01.17	XAA0190	删除 3.4 小节 board.dts 板级配置更新 3.3 章节内容
1.7	2025.2.18	XAA0339	修改文档格式
1.8	2025.3.15	XAA0331	修改适用范围及 FAQ 描述

目　录

1 概述

1.1 编写目的

介绍 Allwinner RTC 模块的使用方法，为 RTC 设备的使用者和维护者提供参考。

1.2 适用范围

适用于使用 bsp 独立仓库的所有内核及版本，linux-5.4/linux-5.4-ansc 及以上。

1.3 相关术语介绍

术语	解释说明
Sunxi	指 Allwinner 的一系列 SoC 硬件平台
RTC	Real Time Clock，实时时钟

2 模块介绍

2.1 模块功能介绍

Linux 内核中,RTC 驱动的结构图如下所示, 可以分为三个层次:

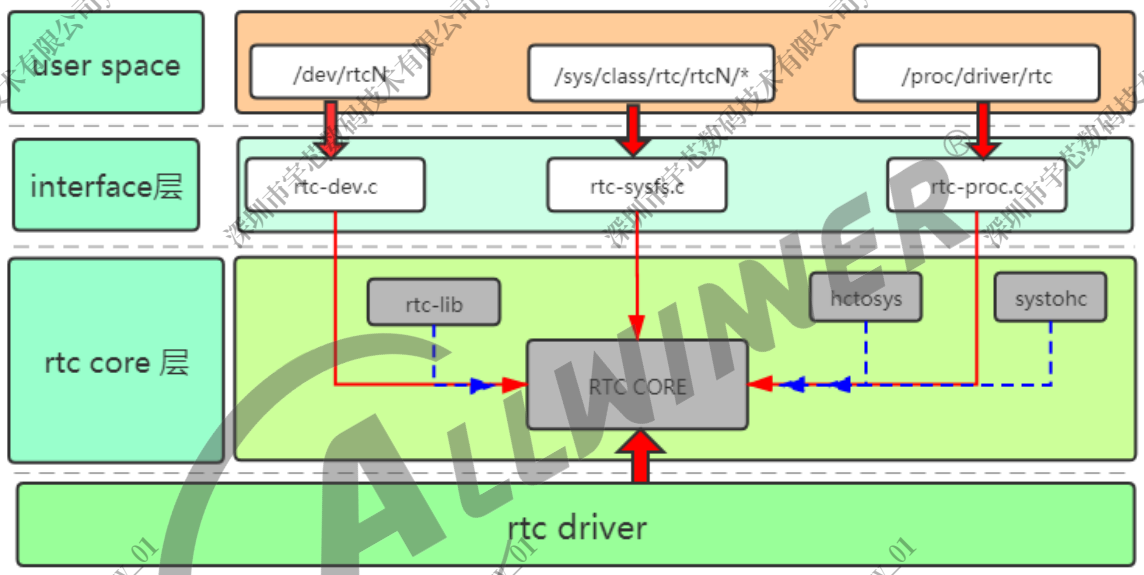

图 2-1

- 接口层，负责向用户空间提供操作的结点以及相关接口。
- RTC Core, 为 RTC 驱动提供了一套 API, 完成设备和驱动的注册等。
- RTC 驱动层，负责具体的 RTC 驱动实现，如设置时间、闹钟等设置寄存器的操作。

2.2 源码结构介绍

```
/drivers/rtc
├── Kconfig
├── Makefile
├── rtc-jxr160.c
├── rtc-sd2059.c
├── rtc-sunxi.c
└── rtc-sunxi.h
```

```
└── sunxi_timer_alarm.c
```

3 模块配置介绍

3.1 menuconfig 配置

在命令行中进入 longan 顶层目录，执行./build.sh config，按照提示配置平台、板型等信息（如果之前已经配置过，可跳过此步骤）。

然后执行./build.sh menuconfig，并按以下步骤操作：

```
Allwinner BSP  --->
  Device Drivers  --->
    RTC Drivers  --->
      <*> RTC Support for Allwinner SoCs
      [*]  Enable RTC General Registers for Reboot Flag
      [*]  Enable RTC Alarm in Power-Off Stat
```

上述 menuconfig 配置时也可直接搜索并打开如下配置项进行配置:

```
CONFIG_AW_RTC
```

3.2 device tree 源码结构和路径

SoC 级设备树文件（sun*.dtsi）是针对该 SoC 所有方案的通用配置，SoC 级设备树路径：bsp/configs/linux-5.10/${chip}.dtsi

板级设备树路径：device/config/chips/{IC}/configs/{BOARD}/board.dts

device tree的源码结构关系如下：

```
board.dts
  └--------${chip}.dtsi
```

3.3 device tree 对 RTC 控制器的通用配置

RTC 作为一个系统资源，不会跟随外设而变化，仅需配置 dtsi 文件，无需适配 board.dts 文件。

在设备树文件（bsp/configs/linux-xxx/sunxxx.dtsi）中里新增对 RTC 的描述。

注意：下述配置使用过程中无需修改，直接采用默认配置即可。

```
/ {
  rtc: rtc@7000000 {
    compatible = "allwinner,sun50iw10p1-rtc"; //用于probe驱动加载
    device_type = "rtc";
    wakeup-source;          //表示RTC是具备休眠唤醒能力的中断唤醒源
    reg = <0x0 0x07000000 0x0 0x200>; //RTC寄存器基地址和映射范围
    interrupts = <GIC_SPI 108 IRQ_TYPE_LEVEL_HIGH>; //RTC硬件中断号
    clocks = <&r_ccu CLK_R_AHB_BUS_RTC>, <&rtc_ccu CLK_RTC_1K>;  //RTC所用到的时钟
    clock-names = "r-ahb-rtc", "rtc-1k";  //上述时钟的名字
    resets = <&r_ccu RST_R_AHB_BUS_RTC>;
    gpr_cur_pos = <6>;          //当前被用作reboot-flag的通用寄存器的序号
    gpr_bootcount_pos = <7>;      //用于存储启动原因，如冷启动，热启动
    gprcm_reg = <0x0 0x4a000000 0x0 0x20> //仅在公共寄存器基地址和rtc基地址不连续时配置，仅适用于部分平台，如
        sun300iw1p1
  };
}
```

其中：

- compatible 属性要与 C 代码里的sunxi_rtc_dt_ids[]里的compatible保持一致。

4 接口描述

RTC 驱动加载成功后会注册生成串口设备/dev/rtcN，应用层的使用只需遵循 Linux 系统中的标准 RTC 编程方法即可。

4.1 确认 RTC 驱动正确加载

小机端执行命令:

```
ls /dev/rtc*
```

能够查询到 RTC 设备，表明内核 RTC 驱动正确加载。

RTC 驱动支持 ioctl 指令功能描述:

表 4-1: RTC 驱动 ioctl 指令功能描述

指令	功能描述
RTC_ALM_READ	读取闹钟时间
RTC_ALM_SET	设置闹钟时间
RTC_SET_TIME	设置时间与日期
RTC_PIE_ON	开 RTC 全局中断
RTC_PIE_OFF	关 RTC 全局中断
RTC_AIE_ON	使能 RTC 闹钟中断
RTC_AIE_OFF	禁止 RTC 闹钟中断
RTC_UIE_ON	使能 RTC 更新中断
RTC_UIE_OFF	禁止 RTC 更新中断
RTC_IRQP_SET	设置中断的频率

4.2 打开/关闭 RTC 设备

使用标准的文件打开函数：

```
int open(const char *pathname, int flags);
int close(int fd);
```

需要引用头文件：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

4.3 设置和获取 RTC 时间

同样使用标准的ioctl函数：

```
int ioctl(int d, int request, ...);
```

需要引用头文件：

```
#include <sys/ioctl.h>
#include <linux/rtc.h>
```

设置 RTC 时间：

```
ioctl(fd, RTC_SET_TIME, &rtc_tm)
```

获取 RTC 时间：

```
ioctl(fd, RTC_RD_TIME, &rtc_tm)
```

5 模块使用范例

此demo程序是打开一个RTC设备，然后设置和获取RTC时间以及设置闹钟功能。

```
#include <stdio.h>    /*标准输入输出定义*/
#include <stdlib.h>   /*标准函数库定义*/
#include <unistd.h>   /*Unix 标准函数定义*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>    /*文件控制定义*/
#include <linux/rtc.h> /*RTC支持的CMD*/
#include <errno.h>    /*错误号定义*/
#include <string.h>

#define RTC_DEVICE_NAME  "/dev/rtc0"

int set_rtc_timer(int fd)
{
 struct rtc_time rtc_tm = {0};
 struct rtc_time rtc_tm_temp = {0};

 rtc_tm.tm_year = 2020 - 1900;  /* 需要设置的年份，需要减1900 */
 rtc_tm.tm_mon = 11 - 1;        /* 需要设置的月份,需要确保在0-11范围 */
 rtc_tm.tm_mday = 21;       /* 需要设置的日期 */
 rtc_tm.tm_hour = 10;       /* 需要设置的时间 */
 rtc_tm.tm_min = 12;        /* 需要设置的分钟时间 */
 rtc_tm.tm_sec = 30;        /* 需要设置的秒数 */

 /* 设置RTC时间 */
 if (ioctl(fd, RTC_SET_TIME, &rtc_tm) < 0) {
  printf("RTC_SET_TIME failed\n");
  return -1;
 }

 /* 获取RTC时间 */
 if (ioctl(fd, RTC_RD_TIME, &rtc_tm_temp) < 0) {
  printf("RTC_RD_TIME failed\n");
  return -1;
 }
 printf("RTC_RD_TIME return %04d-%02d-%02d %02d:%02d:%02d\n",
       rtc_tm_temp.tm_year + 1900, rtc_tm_temp.tm_mon + 1, rtc_tm_temp.tm_mday,
       rtc_tm_temp.tm_hour, rtc_tm_temp.tm_min, rtc_tm_temp.tm_sec);
 return 0;
}

int set_rtc_alarm(int fd)
{
 struct rtc_time rtc_tm = {0};
 struct rtc_time rtc_tm_temp = {0};

 rtc_tm.tm_year = 0;  /* 闹钟忽略年设置 */
 rtc_tm.tm_mon = 0;   /* 闹钟忽略月设置 */
 rtc_tm.tm_mday = 0;  /* 闹钟忽略日期设置 */
```

```
  rtc_tm.tm_hour = 10;  /* 需要设置的时间 */
  rtc_tm.tm_min = 12;   /* 需要设置的分钟时间 */
  rtc_tm.tm_sec = 30;   /* 需要设置的秒数 */

  /* set alarm time */
  if (ioctl(fd, RTC_ALM_SET, &rtc_tm) < 0) {
    printf("RTC_ALM_SET failed\n");
    return -1;
  }

  if (ioctl(fd, RTC_AIE_ON) < 0) {
    printf("RTC_AIE_ON failed!\n");
    return -1;
  }

  if (ioctl(fd, RTC_ALM_READ, &rtc_tm_temp) < 0) {
    printf("RTC_ALM_READ failed\n");
    return -1;
  }

  printf("RTC_ALM_READ return %04d-%02d-%02d %02d:%02d:%02d\n",
          rtc_tm_temp.tm_year + 1900, rtc_tm_temp.tm_mon + 1, rtc_tm_temp.tm_mday,
          rtc_tm_temp.tm_hour, rtc_tm_temp.tm_min, rtc_tm_temp.tm_sec);
  return 0;
}

int main(int argc, char *argv[])
{
    int fd;
    int ret;

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

  /* 设置RTC时间 */
  ret = set_rtc_timer(fd);
  if (ret < 0) {
    printf("set rtc timer error\n");
    return -EINVAL;
  }

  /* 设置闹钟 */
  ret = set_rtc_alarm(fd);
  if (ret < 0) {
    printf("set rtc alarm error\n");
    return -EINVAL;
  }

  close(fd);
  return 0;
}
```

6 FAQ

6.1 RTC 时间不准

1. 请确认 RTC 所使用的时钟源是否准确。

```
ABAP    Warning: Using internal RC 16M clock source. Time may be inaccurate!
```

如存在上述打印，则表明 RTC 时钟源为 RC16M，因为 RC16M 有正负 50% 的偏差，可能会导致 RTC 时间不准。

2. 如果使用外部晶体，则请确认外部晶体的震荡频率是否正确。

6.2 RTC 时间不走

1. 请查看 RTC 时钟源图，确认一下使用的时钟源。
2. 当 RTC 时钟源为外部 32K 时，请确认一下外部 32k 晶体的起振情况。

说明

当使用示波器测量外部 32k 晶体起振情况时，有可能会导致 32k 晶体起振。

3. 当 RTC 时钟源为外部 32K 且确认时钟源没有问题后，通过以下命令dump rtc相关寄存器，查看与外部 32K 晶振相关的寄存器配置是否正常。

```
/ # echo 0x07000000,0x07000200 > /sys/class/sunxi_dump/dump; cat /sys/class/sunxi_dump/dump
0x0000000007000000: 0x00004010 0x00000004 0x0000000f 0x7a000000
0x0000000007000010: 0x00000001 0x00000023 0x00000000 0x00000000
0x0000000007000020: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000030: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000040: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000050: 0x00000001 0x00000000 0x00000000 0x00000000
0x0000000007000060: 0x00000004 0x00000000 0x00000000 0x00000000
0x0000000007000070: 0x00010003 0x00000000 0x00000000 0x00000000
0x0000000007000080: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000090: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000070000a0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000070000b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000070000c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000070000d0: 0x00000000 0x00000000 0x00000000 0x00000000
```

```
0x00000000070000e0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000070000f0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000100: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000110: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000120: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000130: 0x00000000 0x000030ea 0x04001000 0x00006061
0x0000000007000140: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000150: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000160: 0x083f10f7 0x00000043 0x00000000 0x00000000
0x0000000007000170: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000007000180: 0x00000000 0x00000000 0x00010001 0x00000000
0x0000000007000190: 0x00000004 0x00000000 0x00000000 0x00000000
0x00000000070001a0: 0x000090ff 0x00000000 0x00000000 0x00000000
0x00000000070001b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000070001c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000070001d0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000070001e0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000070001f0: 0x00000000 0x00000001 0x00000000 0x00000000
0x0000000007000200: 0x10000000
```

note：每款 SoC 的模块首地址是不一样的，具体根据spec或data sheet确认模块首地址。

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明





（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的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所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。