



## CV184x RTC 操作指南

Version: 1.0.0

Release date: 2025-03-21

©2025 北京晶视智能科技有限公司  
本文件所含信息归北京晶视智能科技有限公司所有。  
未经授权，严禁全部或部分复制或披露该等信息。

# 目录

<b>1</b>	<b>声明</b>	<b>2</b>
<b>2</b>	<b>Cvitek RTC 具体操作指南</b>	<b>3</b>
2.1	模块介绍 . . . . .	3
2.1.1	计数时钟频率 . . . . .	3
2.2	操作准备 . . . . .	3
2.2.1	使用 RTC（默认） . . . . .	3
2.2.2	移除 RTC . . . . .	3
2.3	应用层使用方式 . . . . .	4
2.3.1	设备节点与头文件 . . . . .	4
2.3.2	常用 ioctl 指令 . . . . .	4
2.3.3	结构体 rtc_time 说明 . . . . .	5
2.4	示例代码 . . . . .	5
2.4.1	用户态操作示例（rtc_example.c） . . . . .	5
2.5	小结 . . . . .	8
2.6	RTC 中断定时重启（软重启） . . . . .	8
2.6.1	执行方式 . . . . .	8
2.6.2	从 RTC 闹钟到 SoC 复位的流程 . . . . .	8
2.6.3	关键点小结 . . . . .	9
<b>3</b>	<b>Linux 系统中 RTC 测试命令</b>	<b>10</b>
3.1	date 和 hwclock . . . . .	10

修订记录

Revision	Date	Description
1.0.0	2025/03/21	初稿
1.0.1	2026/04/28	更新 RTC 操作指南并优化

# 1 声明



## 法律声明

本数据手册包含北京晶视智能科技有限公司（下称“晶视智能”）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致晶视智能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。本数据手册和本文件所含的所有信息均按“原样”提供，无任何明示、暗示、法定或其他形式的保证。晶视智能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，晶视智能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

## 联系我们

**地址** 北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

深圳市宝安区福海街道展城社区会展湾云岸广场 T10 栋

**电话** +86-10-57590723 +86-10-57590724

**邮编** 100094（北京）518100（深圳）

**官方网站** <https://www.sophgo.com/>

**技术论坛** <https://developer.sophgo.com/forum/index.html>

# 2 Cvitek RTC 具体操作指南

## 2.1 模块介绍

RTC(real time clock) 实时时钟，是处理器内一个独立供电的恒电模块，为 Linux 系统提供时间。由于 RTC 由电池供电，当处理器处于下电关机或休眠状态时，RTC 仍会维持工作状态时间不丢失。Linux 内核将 RTC 作为时间与日期维护器。亦可作为内核睡眠时唤醒内核的闹钟。

应用程序可以用 RTC 提供的周期中断做周期性任务。

### 2.1.1 计数时钟频率

RTC 的计数时钟采用 32.768KHz 时钟，运行基于一个 32-bit 加法计数器提供秒计数，计数最大时间为：

$2^{32}$  秒 = 49710 天 = 136 年

## 2.2 操作准备

### 2.2.1 使用 RTC（默认）

- 使用 SDK 发布的 U-boot 与 Kernel 即可，无需修改 DTS。
- 设备节点：/dev/rtc0。

### 2.2.2 移除 RTC

若要在板级禁用 RTC，在板级 DTS（或 overlay）里用 `delete-node` 删除 RTC 节点。

本 SDK 中 RTC 节点名为 `rtc`（在 `cv184x_base.dtsi` / `cv1835_asic.dtsi` 中定义），删除示例如下：

```
/ {  
    /delete-node/ rtc;  
};
```

若文档或其它 BSP 中节点名为 cvitek-rtc@3005000，则使用：

```
/delete-node/ cvitek-rtc@3005000;
```

同文件中删除其它节点示例（仅作参考）：

```
/delete-node/ i2s@04120000;  
/delete-node/ sound_ext1;  
/delete-node/ sound_ext2;  
/delete-node/ sound_PDM;  
/delete-node/ rtc;          /* 删除 RTC */  
  
aliases {  
    /delete-property/ ethernet1;  
};
```

## 2.3 应用层使用方式

### 2.3.1 设备节点与头文件

- 设备节点：/dev/rtc0
- 头文件：#include <linux/rtc.h>、#include <sys/ioctl.h>

### 2.3.2 常用 ioctl 指令

指令	描述
RTC_RD_TIME	读取当前时间
RTC_SET_TIME	设置当前时间
RTC_ALM_READ	读取闹钟时间
RTC_ALM_SET	设置闹钟时间
RTC_AIE_ON	使能闹钟中断
RTC_AIE_OFF	禁止闹钟中断
RTC_UIE_ON	使能更新中断
RTC_UIE_OFF	禁止更新中断
RTC_PIE_ON	开周期中断
RTC_PIE_OFF	关周期中断
RTC_IRQP_SET	设置周期中断频率

### 2.3.3 结构体 rtc\_time 说明

```
struct rtc_time {
    int tm_sec; /* 秒 [0,59] */
    int tm_min; /* 分 [0,59] */
    int tm_hour; /* 时 [0,23] */
    int tm_mday; /* 日 [1,31] */
    int tm_mon; /* 月 [0,11], 0=1月 */
    int tm_year; /* 年, 实际年 = tm_year + 1900 */
    int tm_wday; /* 星期几 [0,6], 0=周日, 1=周一 */
    int tm_yday; /* 一年中第几天 [0,365], 0=1月1日 */
    int tm_isdst; /* 夏令时: 1=是, 0=否 */
};
```

注意：

- tm\_mon: 0 表示 1 月, 11 表示 12 月。
- tm\_year: 存的是“年份 - 1900”，如 2025 年填 125。
- tm\_wday: 0= 周日, 1= 周一, 依此类推。
- tm\_yday: 0=1 月 1 日, 1=1 月 2 日。

## 2.4 示例代码

### 2.4.1 用户态操作示例 (rtc\_example.c)

该示例将“读取时间 / 设置时间 / 设置闹钟”整合为一个程序，通过参数选择具体操作。

编译

```
arm-none-linux-uclibcgnueabi-hf-gcc -o rtc_example rtc_example.c -static
```

运行

```
./rtc_example [read|set|alarm]
```

- read: 读取并打印当前 RTC 时间。
- set: 将 RTC 设置为示例时间 (2025-03-18 12:00:00)，并打印设置后的时间。
- alarm: 将闹钟设为当前时间 + 5 秒，并开启闹钟中断（部分 RTC 可能不支持闹钟中断，会返回错误）。

```
/*
 * Cvitek RTC 使用示例：读取时间、设置时间、设置闹钟
 * 编译: arm-none-linux-uclibcgnueabi-hf-gcc -o rtc_example rtc_example.c -static
 * 运行: ./rtc_example [read|set|alarm]
 */
#include <stdio.h>
#include <stdlib.h>
```

(下页继续)

(续上页)

```

#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <linux/rtc.h>
#include <sys/ioctl.h>
#include <unistd.h>

#define RTC_DEV "/dev/rtc0"

static void print_rtc_time(const struct rtc_time *t)
{
    printf("RTC 时间: %04d-%02d-%02d %02d:%02d:%02d 星期%d\n",
        t->tm_year + 1900, t->tm_mon + 1, t->tm_mday,
        t->tm_hour, t->tm_min, t->tm_sec, t->tm_wday);
}

/* 读取并打印当前 RTC 时间 */
static int do_read(int fd)
{
    struct rtc_time rtc_tm;
    if (ioctl(fd, RTC_RD_TIME, &rtc_tm) < 0) {
        perror("RTC_RD_TIME");
        return -1;
    }
    print_rtc_time(&rtc_tm);
    return 0;
}

/* 设置 RTC 时间 (示例: 2025-03-18 12:00:00) */
static int do_set(int fd)
{
    struct rtc_time rtc_tm = {
        .tm_sec = 0,
        .tm_min = 0,
        .tm_hour = 12,
        .tm_mday = 18,
        .tm_mon = 2, /* 3 月 */
        .tm_year = 125, /* 2025 */
        .tm_wday = 2,
        .tm_isdst = 0
    };
    if (ioctl(fd, RTC_SET_TIME, &rtc_tm) < 0) {
        perror("RTC_SET_TIME");
        return -1;
    }
    printf("已设置 RTC, 当前为: ");
    return do_read(fd);
}

/* 设置闹钟为当前时间 + 5 秒, 并开启闹钟中断 */
static int do_alarm(int fd)
{
    struct rtc_time rtc_tm;
    if (ioctl(fd, RTC_RD_TIME, &rtc_tm) < 0) {

```

(下页继续)



(续上页)

```

    perror("RTC_RD_TIME");
    return -1;
}
rtc_tm.tm_sec += 5;
if (rtc_tm.tm_sec >= 60) {
    rtc_tm.tm_sec %= 60;
    rtc_tm.tm_min++;
}
if (rtc_tm.tm_min >= 60) {
    rtc_tm.tm_min = 0;
    rtc_tm.tm_hour++;
}
if (rtc_tm.tm_hour >= 24)
    rtc_tm.tm_hour = 0;

if (ioctl(fd, RTC_ALM_SET, &rtc_tm) < 0) {
    if (errno == EINVAL)
        fprintf(stderr, "此 RTC 不支持闹钟中断\n");
    else
        perror("RTC_ALM_SET");
    return -1;
}
ioctl(fd, RTC_AIE_ON, 0);
printf("闹钟已设为 5 秒后, 已开启闹钟中断\n");
return 0;
}

int main(int argc, char **argv)
{
    const char *cmd = (argc > 1) ? argv[1] : "read";
    int fd = open(RTC_DEV, O_RDONLY);
    if (fd < 0) {
        perror("open " RTC_DEV);
        return 1;
    }

    if (strcmp(cmd, "read") == 0)
        do_read(fd);
    else if (strcmp(cmd, "set") == 0)
        do_set(fd);
    else if (strcmp(cmd, "alarm") == 0)
        do_alarm(fd);
    else {
        fprintf(stderr, "用法: %s [read|set|alarm]\n", argv[0]);
        close(fd);
        return 1;
    }

    close(fd);
    return 0;
}

```

## 2.5 小结

操作	做法
使用 RTC	不删节点，应用直接操作 <code>/dev/rtc0</code>
移除 RTC	在板级 DTS 中加 <code>/delete-node/ rtc;</code> （本 SDK 节点名为 <code>rtc</code> ）
读时间	<code>ioctl(fd, RTC_RD_TIME, &amp;rtc_tm)</code>
设时间	填充 <code>struct rtc_time</code> ，再 <code>ioctl(fd, RTC_SET_TIME, &amp;rtc_tm)</code>
设闹钟	填充时间后 <code>ioctl(fd, RTC_ALM_SET, &amp;rtc_tm)</code> ，再用 <code>RTC_AIE_ON</code> 使能

## 2.6 RTC 中断定时重启（软重启）

软重启指不断电、不拉闸，仅让 CPU/SoC 重新执行启动流程（内核重新跑一遍），电源保持供电。通过 RTC 闹钟到点触发，可实现定时软重启。

### 2.6.1 执行方式

1. 在设备上查找 `auto_restart_after` 节点路径：

```
find /sys/devices/ -name auto_restart_after
```

2. 向该节点写入秒数即可在指定秒数后触发软重启。下面示例为 30 秒后重启，路径需替换为上一步 find 得到的实际节点（通常形如 `/sys/devices/platform/5026000.rtc/auto_restart_after`）：

```
echo 30 > /sys/devices/platform/5026000.rtc/auto_restart_after
```

### 2.6.2 从 RTC 闹钟到 SoC 复位的流程

1. **设置 RTC 闹钟**：用户通过 `sysfs` 写入属性 `auto_restart_after`，例如命令 `echo 30`，表示约 30 秒后重启。驱动读取 RTC 当前秒数，将「当前秒数 + 设定秒数」写入 RTC 闹钟寄存器并使能闹钟。
2. **RTC 闹钟中断**：到点后 RTC 硬件比较秒计数与闹钟时间，拉高中断。内核 RTC 驱动在中断处理函数中清除中断、通过 `rtc_update_irq()` 通知用户态 `/dev/rtc0`，并提交一个延迟工作项。
3. **工作队列中触发重启**：在工作函数中，若此前已通过 `auto_restart_after` 设定了定时重启，则调用 `orderly_reboot()`；若超时未重启则再调用 `kernel_restart(NULL)`。
4. **orderly\_reboot**：先尝试执行 `/sbin/reboot`，给用户态做收尾（sync、脚本等）；若执行失败则直接调用 `kernel_restart(NULL)`。
5. **kernel\_restart**：进入重启准备阶段（通知链、设备 shutdown、迁移到重启 CPU、系统核心关闭、内核日志 dump），最后调用架构相关的 `machine_restart()`。
6. **machine\_restart (ARM64)**：关中断、停止其它 CPU，再调用 `do_kernel_restart()`。

7. **do\_kernel\_restart**: 调用所有通过 `register_restart_handler()` 注册的重启回调。
8. **Cvitek 重启回调**: 在 `cvi-reboot.c` 中注册的 `cvi_restart_handler()` 写 RTC 域寄存器请求温复位 (`RTC_EN_WARM_RST_REQ`)，等待状态就绪后写 `RTC_CTRL0` 触发 SoC 内部 warm reset。CPU 从 BootROM/固件重新启动，整机不断电，完成软重启。

### 2.6.3 关键点小结

- **RTC 闹钟**: 提供“到点触发”，用 RTC 当前秒数 + N 秒设闹钟，与系统时间解耦。
- **orderly\_reboot**: 先执行 `/sbin/reboot` 做用户态收尾，失败则直接 `kernel_restart`。
- **kernel\_restart**: 关设备、迁 CPU、dump 日志，再交给架构的 `machine_restart`。
- **cvi\_restart\_handler**: Cvitek SoC 的软复位实现，通过写 RTC 域 `WARM_RST_REQ` 由硬件做 CPU/SoC 温复位，电源保持供电。

# 3 Linux 系统中 RTC 测试命令

## 3.1 date 和 hwclock

- **date** 命令用于查询或设置当前 Linux 系统（软件）时钟。Linux 启动时，系统时钟会从硬件时钟（RTC）同步。
- **hwclock** 命令用于查询或设置硬件时钟（RTC）的时间。

示例：设置系统时间

```
date "2020-10-17 9:48:30"  
# Sat Oct 17 09:48:30 CST 2020
```

示例：读取系统时间

```
date  
# Sat Oct 17 09:48:34 CST 2020
```

示例：查询硬件时钟（RTC）时间

```
hwclock  
# Sat Oct 17 09:56:03 2020 0.000000 seconds
```

示例：将系统时间写入硬件时钟

```
hwclock -w
```

示例：将硬件时钟同步到系统时间

```
hwclock -s
```