HAL_V2 RTC

开发指南

版本号: 1.0

发布日期: 2025.7.15

版本历史

版本号	日期	制/修订人	内容描述
1.0	2025.7.15	AWA2351	初始版本

目　录

1 前言

1.1 文档简介

介绍 HAL_V2 中 RTC 驱动的接口及使用方法，为 RTC 使用者提供参考。

1.2 目标读者

RTC 驱动的开发/维护人员。

1.3 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
T536	FreeRTOS	hal_rtc.c
T153	FreeRTOS	hal_rtc.c

1.4 文档约定

1 标志说明

⚠ 注意

- **提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。**

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.5 相关术语介绍

1.5.1 硬件术语

表 1-2: 硬件术语

术语	解释说明
RTC	Real-Time Clock, 实时时钟

1.5.2 软件术语

表 1-3: 软件术语

术语	解释说明
HAL	Hardware Abstraction Layer，硬件抽象层

1.6 模块功能

实时时钟 RTC 用于实现时钟计数和定时唤醒功能，能够实时显示日，时，分，秒功能。该模块电源独立，可以在系统掉电后工作，RTC 具有以下特点。

- 内部具有一个 16bit 的日计数器，5bit 的小时计数器，6bit 的分计数器，6bit 的秒计数器
- 可外接 32768Hz 低频晶振作为计数时钟
- 可随时软件配置初始值
- 具有定时闹钟功能，可产生中断及唤醒外围设备
- 8 个用户寄存器可存放掉电信息
- 多个特殊寄存器记录 BROM 相关信息

模块配置介绍

1.7.1 menuconfig 配置项

```
DRIVERS_V2_RTC //打开驱动
HAL_TEST_RTC  //打开测试，可不开
```

1.7.2 RTC 时钟源配置

RTC 模块支持配置使用内部 32K 还是外部 32K 时钟源。如果对时间精度比较敏感，建议选择外部
钟源。

1.8 模块源码结构

1. 驱动源码

```
hal_v2/hal/source/RTC/
├── hal_rtc.c
├── rtc_lib.c
├── common_rtc.h
├── Kconfig
├── Makefile
├── platform
│   ├── rtc_sun8iw22.h
│   └── rtc_sun55iw6.h
└── platform_rtc.h
```

2. 驱动 APIs 声明头文件

```
hal_v2/include/hal/
└── hal_rtc.h
```

动 APIs 测试代码

```
hal_v2/hal/examples/RTC/
├── rtc_get_time
│   ├── rtc_get_time.c
│   └── Makefile
└── rtc_set_alarm
    ├── rtc_set_alarm.c
    └── Makefile
```

2 模块接口说明

表 2-1: 模块接口列表

API	解释说明
hal_rtc_init　模块初始化	RTC
hal_rtc_deinit	RTC 模块反初始化
hal_rtc_get_time	获取 RTC 时间
hal_rtc_set_time	设置 RTC 时间
hal_rtc_get_alarm	获取 RTC 闹钟
hal_rtc_set_alarm	设置 RTC 闹钟
hal_rtc_set_fel_flag	设置 FEL 标志位
hal_rtc_probe_fel_flag	探测 FEL 标志位并返回其值
hal_rtc_clear_fel_flag	清除 FEL 标志位
hal_rtc_write_data	在指定索引位置向 RTC 写入数据
hal_rtc_read_data	从指定索引位置从 RTC 读取数据
hal_rtc_get_bootmode_flag	获取当前的启动模式标志
hal_rtc_set_bootmode_flag	设置启动模式标志为指定的值

hal_rtc_init

- 作用：初始化 RTC 模块
- 参数：
  - rtc_init 代表 rtc 初始化配置结构体
- 返回值：
  - 0 代表成功
  - -1 代表失败

hal_rtc_deinit

- 作用：反初始化 RTC 模块
- 参数：无
- 返回值：
  - 0 代表成功
  - -1 代表失败

2.3 hal_rtc_get_time

- 作用：获取 rtc 日历时间值
- 参数：
  - rtc_tm 代表 rtc 日历时钟结构体
- 返回值：
  - 0 代表成功
  - -1 代表失败

2.4 hal_rtc_set_time

- 作用：设置 rtc 日历时间值
- 参数：
  - rtc_tm 代表 rtc 日历时钟结构体
- 返回值：
  - 0 代表成功
  - -1 代表失败

hal_rtc_get_alarm

- 作用：获取 rtc 闹钟时间值
- 参数：
  - alarm 代表 rtc 闹钟结构体
- 返回值：
  - 0 代表成功
  - -1 代表失败

hal_rtc_set_alarm

- 作用：设置 rtc 闹钟时间值
- 参数：
  - alarm 代表 rtc 闹钟结构体
- 返回值：
  - 0 代表成功
  - -1 代表失败

hal_rtc_set_fel_flag

- 功能：设置 FEL 标志位
- 参数：无
- 返回值：无

2.8 hal_rtc_probe_fel_flag

- 功能：探测 FEL 标志位并返回其值
- 参数：无
- 返回值：FEL 标志位的值

2.9 hal_rtc_clear_fel_flag

- 功能：清除 FEL 标志位
- 参数：无
- 返回值：无

2.10 hal_rtc_get_bootmode_flag

- 功能：获取当前的启动模式标志
- 参数：无
- 返回值：启动模式标志的值

1 hal_rtc_set_bootmode_flag

- 功能：设置启动模式标志为指定的值
- 参数：
  - flag: 要设置的启动模式标志值
- 返回值：
  - 0 代表成功
  - -1 代表失败

2 hal_rtc_write_data

- 功能：在指定索引位置向 RTC 写入数据
- 参数：
  - index: 写入数据的索引位置
  - val: 要写入的数据
- 返回值：无

3 hal_rtc_read_data

- 功能：从指定索引位置从 RTC 读取数据
- 参数：
  - index: 读取数据的索引位置
- 返回值：读取到的数据

3 模块使用范例

可参考驱动 APIs 测试代码（hal_v2/hal/examples/rtc/rtc_set_alarm/rtc_set_alarm.c），具体实现如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

de <hal_clk.h>

#include <hal_reset.h>
#include <hal_interrupt.h>
#include <hal_rtc.h>

#if defined(CONFIG_KERNEL_FREERTOS)
#include <hal_log.h>
#include <hal_cmd.h>
#include <hal_thread.h>
#include <hal_timer.h>
#elif defined(CONFIG_KERNEL_BAREMETAL)
#include "shell.h"
#endif


static int rtc_clk_init(void)

    int ret;
    hal_clk_t clk_rc;
    hal_clk_t clk_32k;
    hal_clk_t clk_spi;
    clk_number_t clk_rtc_rc_num;
    clk_number_t clk_rtc_32k_num;
    clk_number_t clk_rtc_spi_num;

    /*RC16M ENABLE*/
    clk_rtc_rc_num = MAKE_CLKn(AW_RTC_CCU, CLK_RTC_RC_GATE);

    /*RTC LOSC EXTERNAL CLK ENABLE*/
    clk_rtc_32k_num = MAKE_CLKn(AW_RTC_CCU, CLK_RTC_CLK32K_MUX);

    /*CLK RTC SPI ENABLE*/
_rtc_spi_num = MAKE_CLKn(AW_RTC_CCU, CLK_RTC_SPI);

    printf("clk info, clk_num: %u, cc_id: %u, clk_id: %u\n", clk_rtc_rc_num, CC_ID(clk_rtc_rc_num), CLK_ID(
      clk_rtc_rc_num));
    printf("clk info, clk_num: %u, cc_id: %u, clk_id: %u\n", clk_rtc_32k_num, CC_ID(clk_rtc_32k_num), CLK_ID(
      clk_rtc_32k_num));
    printf("clk info, clk_num: %u, cc_id: %u, clk_id: %u\n", clk_rtc_spi_num, CC_ID(clk_rtc_spi_num), CLK_ID(
      clk_rtc_spi_num));

    ret = hal_n_clk_get_by_clk_num(clk_rtc_rc_num, &clk_rc);
    if (ret) {
        printf("get clk(%u) failed, ret: %d\n", clk_rtc_rc_num, ret);
```

```
        return -1;
    }

    ret = hal_n_clk_get_by_clk_num(clk_rtc_32k_num, &clk_32k);
    if (ret) {
        printf("get clk(%u) failed, ret: %d\n", clk_rtc_32k_num, ret);
        return -1;
    }

    ret = hal_n_clk_get_by_clk_num(clk_rtc_spi_num, &clk_spi);
    if (ret) {
        printf("get clk(%u) failed, ret: %d\n", clk_rtc_spi_num, ret);
        return -1;
    }

    ret = hal_n_clk_enable(clk_rc);
    if (ret) {
printf("%d\n", clk_rtc_rc_num, ret);
        ret = -2;
        goto exit_with_put_rc_clk;
    }

    ret = hal_n_clk_enable(clk_32k);
    if (ret) {
        printf("enable clk(%u) failed, ret: %d\n", clk_rtc_32k_num, ret);
        ret = -2;
        goto exit_with_put_32k_clk;
    }

    ret = hal_n_clk_enable(clk_spi);
    if (ret) {
        printf("enable clk(%u) failed, ret: %d\n", clk_rtc_spi_num, ret);
        ret = -2;
goto exit_with_put_spi_clk;
    }

    printf("enable clk(%u) success\n", clk_rtc_rc_num);
    printf("enable clk(%u) success\n", clk_rtc_32k_num);
    printf("enable clk(%u) success\n", clk_rtc_spi_num);

    ret = 0;

exit_with_put_spi_clk:
    hal_n_clk_put(clk_spi);

exit_with_put_32k_clk:
    hal_n_clk_put(clk_32k);

exit_with_put_rc_clk:
l_n_clk_put(clk_rc);


    return ret;
}

static hal_irqreturn_t alarm_callback(void *data)
{
    hal_v2_interrupt_clear_pending(SUXNI_IRQ_RTC);
    writel(SUNXI_ALRM_IRQ_STA_CNT_IRQ_PEND, SUNXI_RTC_BASE+SUNXI_ALRM_IRQ_STA);
    printf("alarm interrupt!!!\ n");
    return 0;
```

```
}


void hal_v2_rtc_set_alarm(void)
{
    int ret = 0;
    int count = 10;
    int delay = 1 * 1000 * 1000;
    struct rtc_time time;
    ret = rtc_clk_init () ;
    if (ret) {
        printf("rtc_clk_init failed, ret: %d\n", ret);
        return;
    }
    rtc_init_t rtc_initstruct ;
    /*rt real time set 2025-5-17 10:30:00*/
    rtc_initstruct . time.years = 125;
_initstruct.time.months=5;
    rtc_initstruct . time.days = 17;
    rtc_initstruct . time.hours = 10;
    rtc_initstruct . time.minutes = 30;
    rtc_initstruct . time.seconds = 00;

    /*rt alarm time set 2025-5-17 10:30:09*/
    rtc_initstruct . alarm.time.years = 125;
    rtc_initstruct . alarm.time.months = 5;
    rtc_initstruct . alarm.time.days = 17;
    rtc_initstruct . alarm.time.hours = 10;
    rtc_initstruct . alarm.time.minutes = 30;
    rtc_initstruct . alarm.time.seconds = 5;
    rtc_initstruct . alarm.enabled = 1;

    ret = hal_rtc_init(&rtc_initstruct ) ;
if (ret){
        printf("hal_rtc_init failed, ret: %d\n", ret);
        return;
    }

    hal_irq_init_t intc_initstruct ;
    intc_initstruct . irqn = SUXNI_IRQ_RTC;
    intc_initstruct . preemptionpriority = 0;
    intc_initstruct . subpriority = 0;
    intc_initstruct . cmd = HAL_INIT_CMD_ENABLE;
    intc_initstruct . isr_info . func = alarm_callback;
    hal_v2_intc_irq_config(&intc_initstruct);

    struct rtc_alarm wkalrm;
    hal_rtc_get_alarm(&wkalrm);
    printf("get alarm time :%04d-%02d-%02d %02d:%02d:%02d\n",
    wkalrm.time.years + 1900, wkalrm.time.months, wkalrm.time.days,
    wkalrm.time.hours, wkalrm.time.minutes, wkalrm.time.seconds);

    while(count > 0) {
        hal_rtc_get_time(&time);
        printf("rtc time :%04d-%02d-%02d %02d:%02d:%02d\n",
            time.years + 1900, time.months, time.days,
            time.hours, time.minutes, time.seconds);
        udelay(delay);
        count --;
    }
```

```
}

#if  defined(CONFIG_KERNEL_FREERTOS)
FINSH_FUNCTION_EXPORT_CMD(hal_v2_rtc_set_alarm, rtc_set_alarm_test, rtc hal_v2 APIs tests)
#elif  defined(CONFIG_KERNEL_BAREMETAL)
SHELL_EXPORT_CMD(
SHELL_CMD_PERMISSION(0)|SHELL_CMD_TYPE(SHELL_TYPE_CMD_FUNC)|SHELL_CMD_DISABLE_RETURN,
rtc_set_alarm_test, hal_v2_rtc_set_alarm, test for rtc  set  alarm);
#endif
```

4 FAQ

无

声明

版权所有 ©2025 珠海全志科技股份有限公司。保留一切权利。

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。