



Linux RTC 开发指南

版本号: 2.4
发布日期: 2022.10.31

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.06.29	AWA1440	1. 添加初版
2.0	2020.11.12	XAA0192	1. 添加关于 linux-5.4 的内容
2.1	2020.11.16	XAA0175	1. 修正 make menuconfig 操作步骤
2.2	2020.11.18	XAA0175	1. 根据评审意见修正了文档
2.3	2021.04.08	XAA0175	1. 修改 linux5.4 的 device tree 配置
2.4	2022.10.31	XAA0331	1. 修改图片不能显示的问题 2. 修复 pdf 转化失败



目 录

1 概述	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
2 模块介绍	2
2.1 模块功能介绍	2
2.2 相关术语介绍	2
2.3 源码结构介绍	3
3 模块配置介绍	4
3.1 kernel menuconfig 配置	4
3.1.1 linux-4.9 版本下	4
3.1.2 linux-5.4 版本下	6
3.2 device tree 源码结构和路径	8
3.2.1 linux-4.9 版本下	9
3.2.2 linux-5.4 版本下	9
3.3 device tree 对 RTC 控制器的通用配置	9
3.3.1 linux-4.9 版本下	9
3.3.2 linux-5.4 版本下	10
3.4 board.dts 板级配置	10
4 接口描述	11
4.1 打开/关闭 RTC 设备	11
4.2 设置和获取 RTC 时间	11
5 模块使用范例	12
6 FAQ	14
6.1 RTC 时间不准	14
6.2 RTC 时间不走	14

插 图

图 2-1	Linux RTC 体系结构图	2
图 3-1	内核 menuconfig 根菜单	4
图 3-2	内核 menuconfig RTC 菜单	5
图 3-3	内核 menuconfig SUNXI RTC 驱动菜单	5
图 3-4	内核根菜单	6
图 3-5	内核 menuconfig 根菜单	7
图 3-6	内核 menuconfig 根菜单	8
图 6-1	RTC 时钟源	14



1 概述

1.1 编写目的

介绍 Linux 内核中 RTC 驱动的适配和 DEBUG 方法，为 RTC 设备的使用者和维护者提供参考。

1.2 适用范围

表 1-1: 适用产品列表

内核版本	驱动文件
Linux-4.9 及以上	rtc-sunxi.c

1.3 相关人员

RTC 驱动及应用层的开发/维护人员。

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:

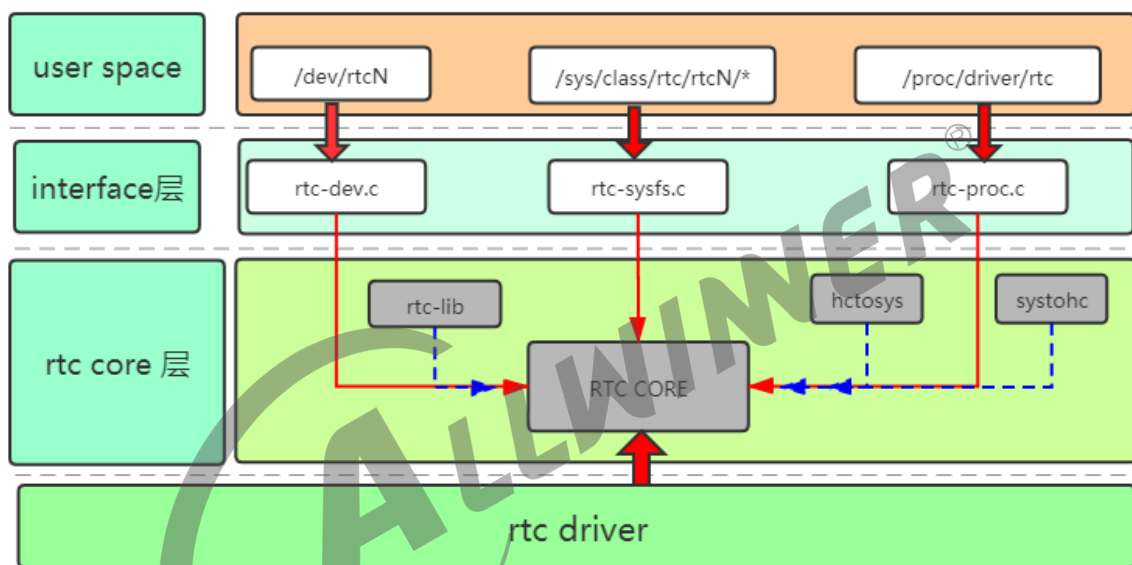


图 2-1: Linux RTC 体系结构图

- 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 相关术语介绍

表 2-1: RTC 模块相关术语介绍

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

2.3 源码结构介绍

```
linux-4.9
|-- drivers
|   |-- rtc
|       |-- class.c
|       |-- hctosys.c
|       |-- interface.c
|       |-- rtc-dev.c
|       |-- rtc-lib.c
|       |-- rtc-proc.c
|       |-- rtc-sysfs.c
|       |-- systohc.c
|       |-- rtc-core.h
|       |-- rtc-sunxi.c
|       |-- rtc-sunxi.h
```

```
linux-5.4
|-- drivers
|   |-- rtc
|       |-- class.c
|       |-- hctosys.c
|       |-- interface.c
|       |-- dev.c
|       |-- lib.c
|       |-- proc.c
|       |-- sysfs.c
|       |-- systohc.c
|       |-- rtc-core.h
|       |-- rtc-sunxi.c
|       |-- rtc-sunxi.h
```

3 模块配置介绍

3.1 kernel menuconfig 配置

3.1.1 linux-4.9 版本下

在命令行中进入内核根目录 (kernel/linux-4.9)，执行 `make ARCH=arm64 arm menuconfig` (32 位系统为 `make ARCH=arm menuconfig`) 进入配置主界面 (linux-5.4 内核版本在 `longan` 目录下执行 `./build.sh menuconfig` 进入配置主界面)，并按以下步骤操作：

首先，选择 `Device Drivers` 选项进入下一级配置，如下图所示：

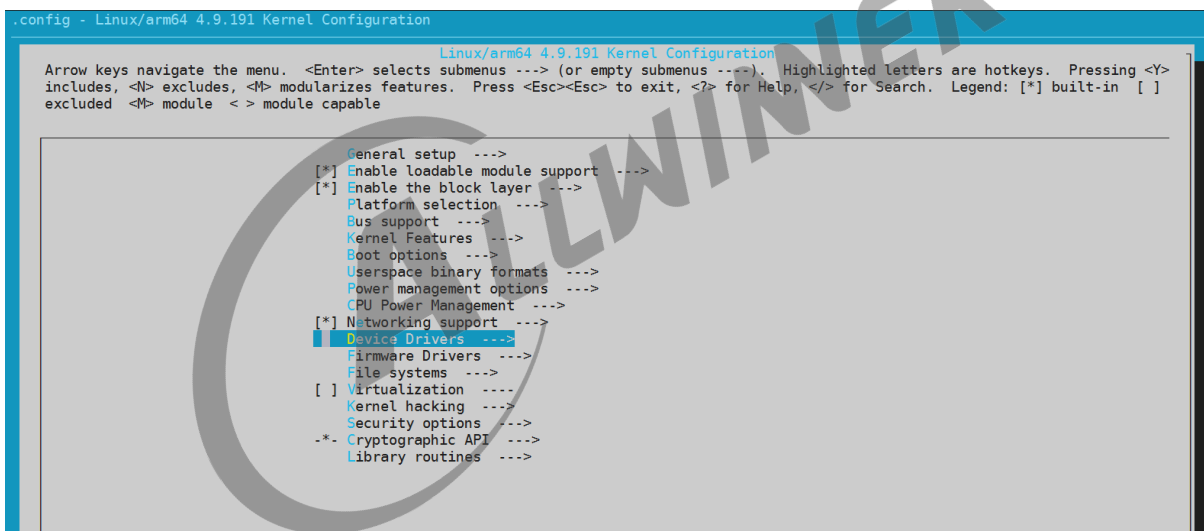


图 3-1: 内核 menuconfig 根菜单

选择 `Real Time Clock`，进入下级配置，如下图所示：

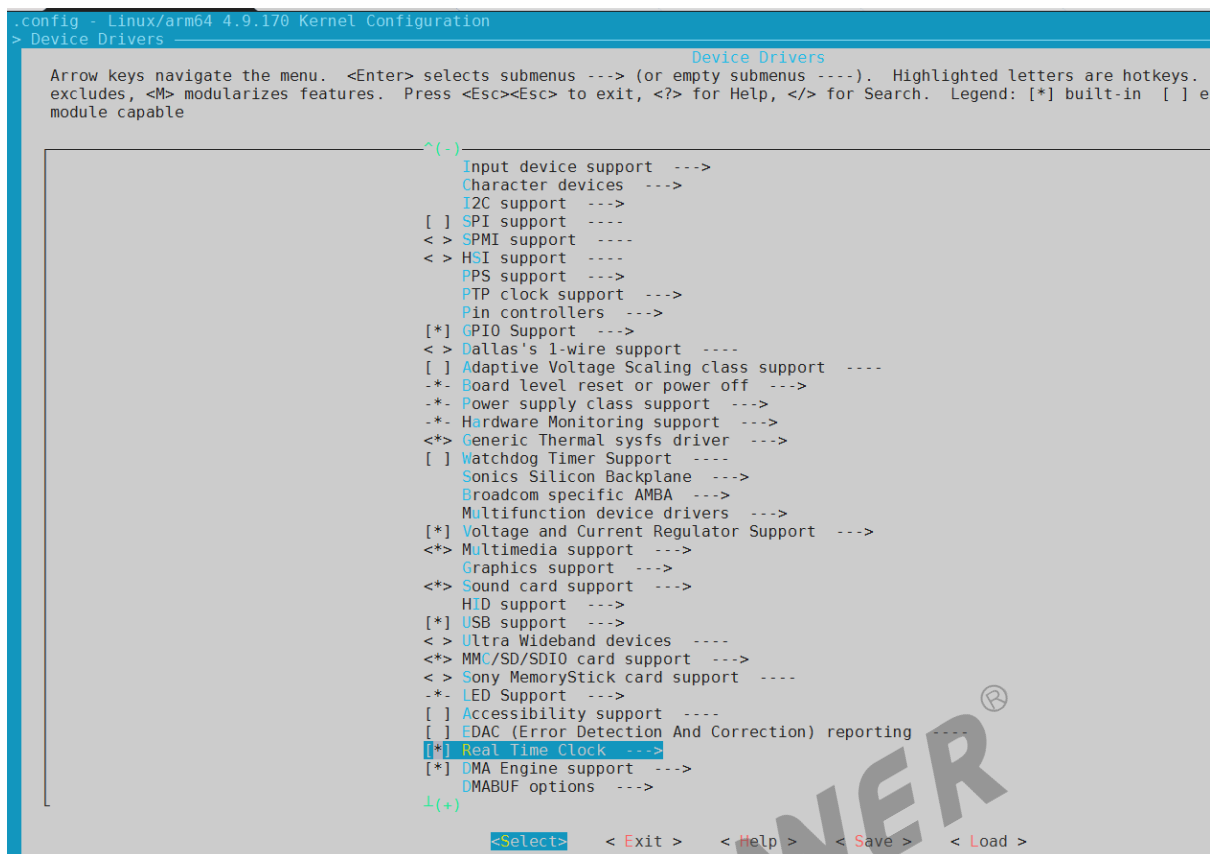


图 3-2: 内核 menuconfig RTC 菜单

选择 Allwinner sunxi RTC，如下图所示：

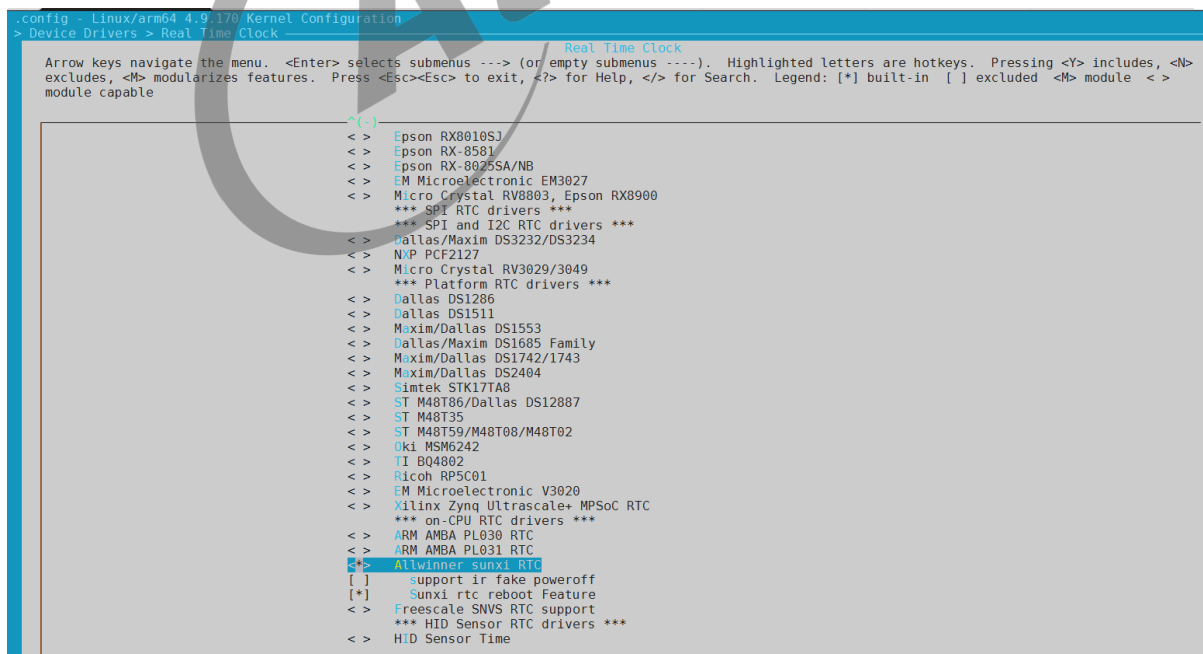


图 3-3: 内核 menuconfig SUNXI RTC 驱动菜单

由于在关机过程中，RTC 一般都是独立供电的，因此在 RTC 电源域中的寄存器不会掉电且 RTC

寄存器的值也不会恢复为默认值。利用此特性，Sunxi 平台支持 reboot 命令的一些扩展功能和假关机功能，但需要打开 support ir fake poweroff 和 Sunxi rtc reboot Feature 选项，RTC 驱动才能支持这些扩展功能。

3.1.2 linux-5.4 版本下

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，进入内核图形化配置界面，并按以下步骤操作：

选择Device Driver选项进入下一级配置，如下图所示：

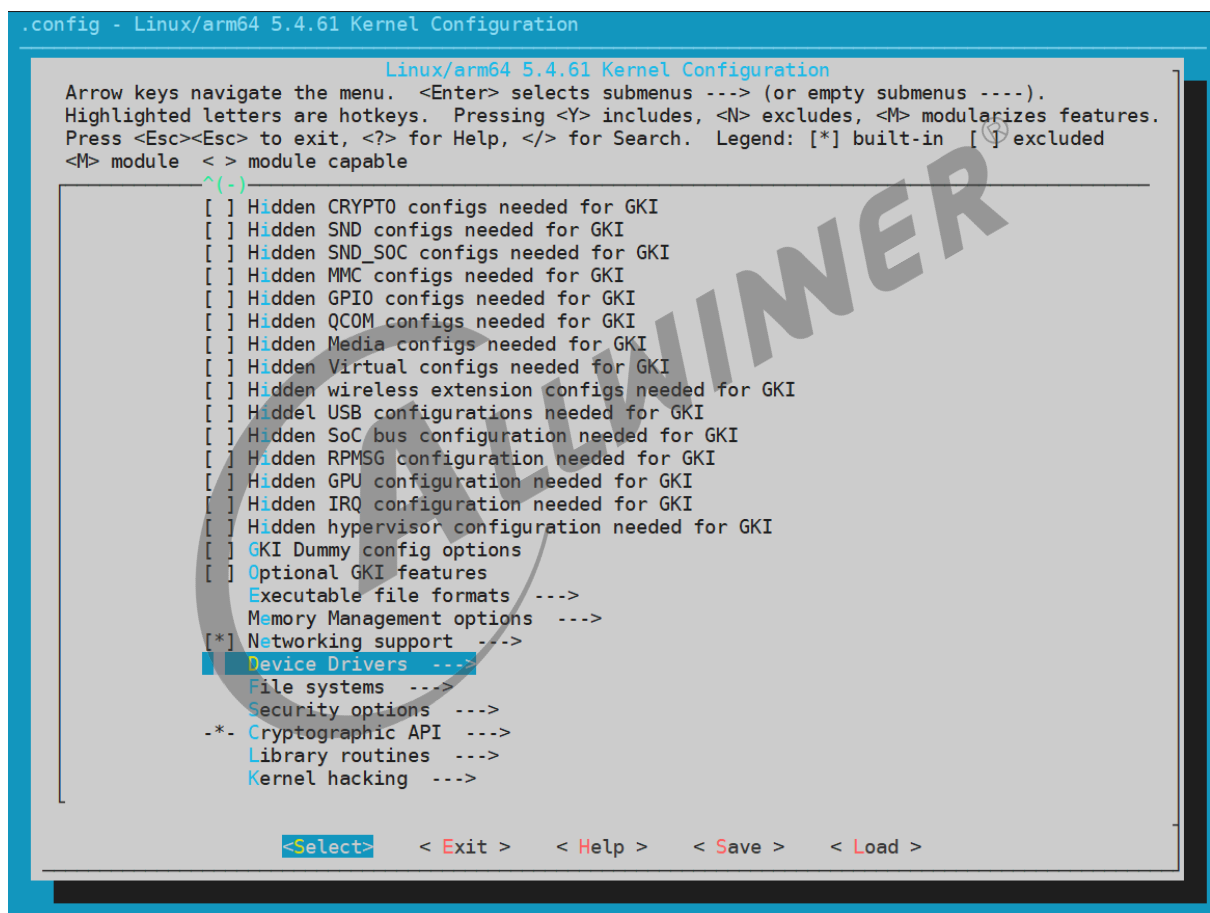


图 3-4: 内核根菜单

选择Real Time Clock进入下一级配置，如下图所示：

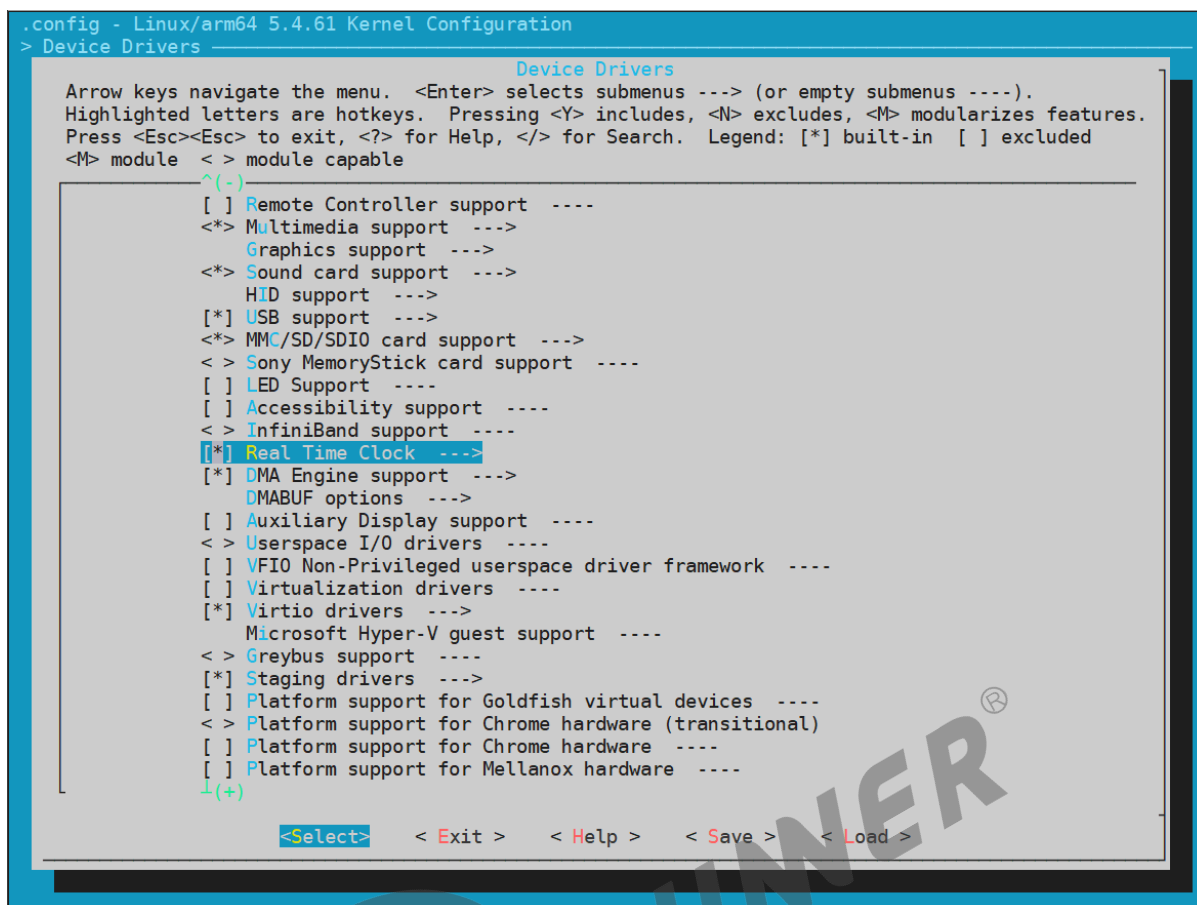


图 3-5: 内核 menuconfig 根菜单

选择Allwinner sunxi RTC配置，如下图所示。

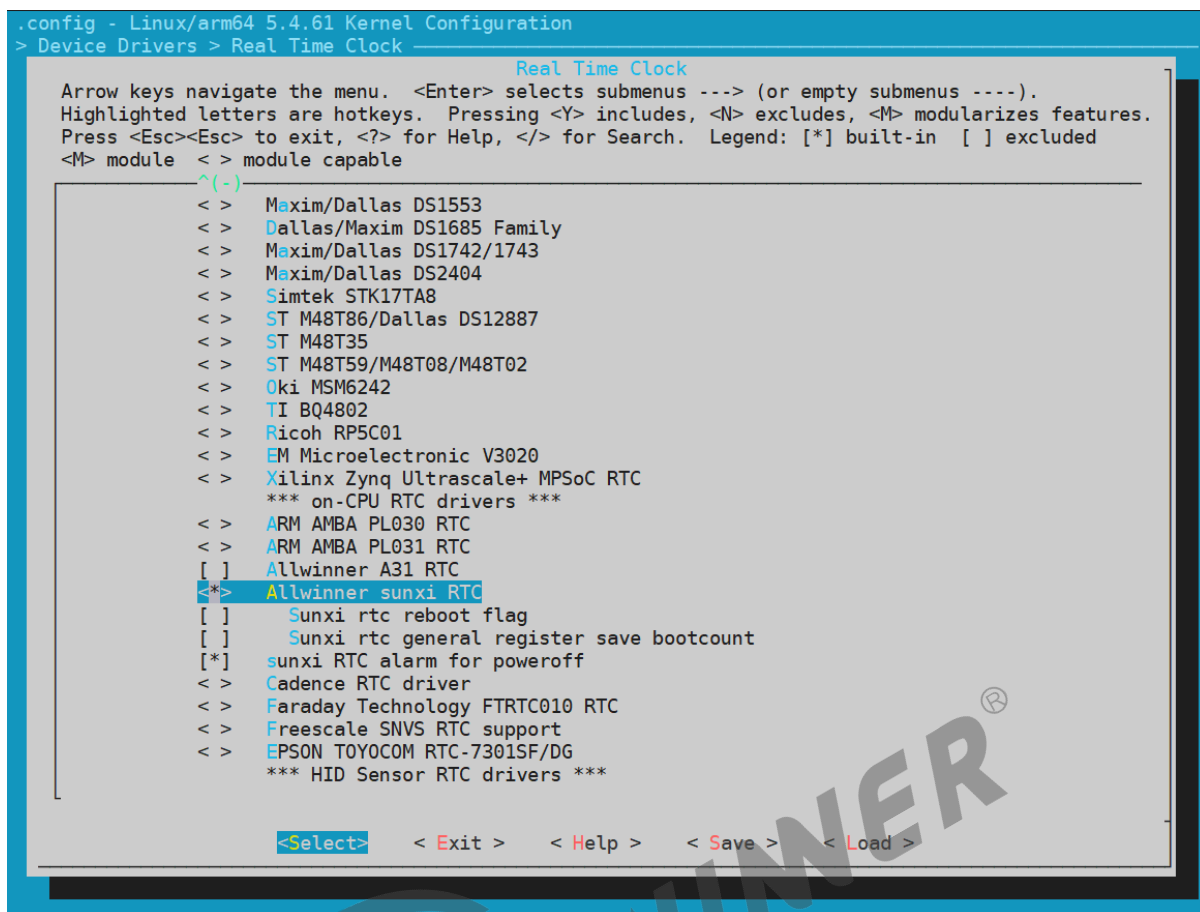


图 3-6: 内核 menuconfig 根菜单

由于在关机过程中，RTC 一般都是独立供电的，因此在 RTC 电源域中的寄存器不会掉电且 RTC 寄存器的值也不会恢复为默认值。利用此特性，Sunxi 平台支持 reboot 命令的一些扩展功能，但需要打开 Sunxi rtc reboot flag 和 Sunxi rtc general register save bootcount 选项，RTC 驱动才能支持这些扩展功能。

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：

- 对于 ARM64 CPU 而言，SoC 级设备树的路径为：arch/arm64/boot/dts/sunxi/sun*.dtsi
- 对于 ARM32 CPU 而言，SoC 级设备树的路径为：arch/arm/boot/dts/sun*.dtsi

板级设备树文件 (board.dts) 是针对该板型的专用配置：

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

3.2.1 linux-4.9 版本下

device tree 的源码结构关系如下：

```
board.dts
├-----sun*.dtsi
│-----sun*-pinctrl.dtsi
└-----sun*-clk.dtsi
```

3.2.2 linux-5.4 版本下

device tree 的源码结构关系如下：

```
board.dts
├-----sun*.dtsi
```

3.3 device tree 对 RTC 控制器的通用配置

3.3.1 linux-4.9 版本下

```
/ {
    rtc: rtc@07000000 {
        compatible = "allwinner,sunxi-rtc"; //用于probe驱动
        device_type = "rtc";
        auto_switch; //支持RTC使用的32k时钟源硬件自动切换
        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 104 IRQ_TYPE_LEVEL_HIGH>; //RTC硬件中断号
        gpr_offset = <0x100>; //RTC通用寄存器的偏移
        gpr_len = <8>; //RTC通用寄存器的个数
        gpr_cur_pos = <6>;
    };
}
```

注意对于 linux-4.9 内核，当 RTC 结点下配置 auto_switch 属性时，RTC 硬件会自动扫描检查外部 32k 晶体振荡器的起振情况。当外部晶体振荡器工作异常时，RTC 硬件会自动切换到内部 RC16M 时钟分频出来的 32k 时钟，从而保证 RTC 工作正常。当没有配置该属性时，驱动代码中直接把 RTC 时钟源设置为外部 32k 晶体的，当外部 32K 晶体工作异常时，RTC 会工作异常。因此建议配置上该属性。

3.3.2 linux-5.4 版本下

```
/ {
    rtc: rtc@7000000 {
        compatible = "allwinner,sun50iw10p1-rtc";    //用于probe驱动
        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
    };
}
```

在 Device Tree 中对每一个 RTC 控制器进行配置, 一个 RTC 控制器对应一个 RTC 节点, 节点属性的含义见注释。

3.4 board.dts 板级配置

board.dts用于保存每个板级平台的设备信息 (如 demo 板、demo2.0 板等等)。board.dts路径如下:

```
device/config/chips/{IC}/configs/{BOARD}/board.dts
```

在board.dts中的配置信息如果在*.dtsi(如sun50iw9p1.dtsi等) 中存在, 则会存在以下覆盖规则:

1. 相同属性和结点, board.dts的配置信息会覆盖*.dtsi中的配置信息
2. 新增加的属性和结点, 会添加到编译生成的 dtb 文件中

4 接口描述

RTC 驱动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 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>
```

4.2 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>
```

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. 按照下图 RTC 时钟源的路径，确认一下 RTC 所使用的时钟源

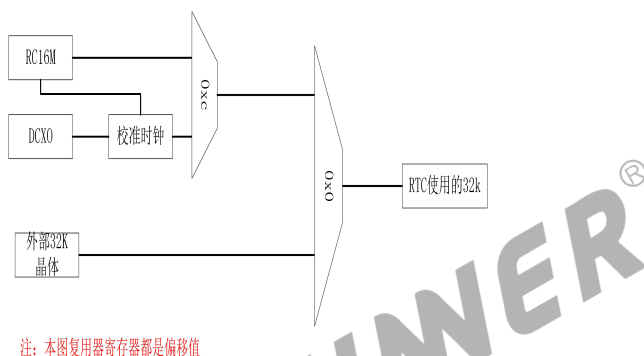


图 6-1: RTC 时钟源

2. 如果确认使用的时钟源为 RC16M，则确认一下有没有启用校准功能，因为 RC16M 有正负 50% 的偏差。
3. 如果使用外部晶体，则确认一下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。

注意：当使用示波器测量外部 32k 晶体起振情况时，有可能会導致 32k 晶体起振。

3. 当排查完时钟源，确认时钟源没有问题后，通过以下命令 dump rtc 相关寄存器，查看偏移 0x0 寄存器的状态位 bit7 和 bit8 是否异常置 1 了，如下所示：

```
/ # 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
```

注意每款 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

版权所有 © 2022 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

、 全志科技、（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。