StarFive 赛昉科技

JH-7110 RTC Developing and Porting Guide

JH-7110 DevKit

Version: 1.0

Date: 2023/05/04

Doc ID: JH7110-DGEN-001

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Information in this document is provided "as is," with all faults. Contents may be periodically updated or revised due to product development. Shanghai StarFive Technology Co., Ltd. (hereinafter "StarFive") reserves the right to make changes without further notice to any products herein.

StarFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose, and non-infringement.

StarFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: http://www.starfivetech.com

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

Contents

List of Tables

List of Figures

Preface

About this guide and technical support information.

About this document

This document mainly provides the SDK developers with the programing basics and debugging know-how for the RTC of the StarFive next generation SoC platform - JH-7110.

Audience

This document mainly serves the RTC relevant driver developers. If you are developing other modules, place a request to your sales or support consultant for our complete documentation set on JH-7110.

Revision History

Table 0-1 Revision History

Version	Released	Revision
1.0	2023/05/04	The First Official Release.

Notes and notices

The following notes and notices might appear in this guide:

- **Tip:**
  Suggests how to apply the information in a topic or step.
- **Note:**
  Explains a special case or expands on an important point.
- **Important:**
  Points out critical information concerning a topic or step.
- **CAUTION:**
  Indicates that an action or step can cause loss of data, security problems, or performance issues.
- **Warning:**
  Indicates that an action or step can result in physical harm or cause damage to hardware.

1. Introduction

Real Time Clock (RTC) usually refers to the module that tracks wall clock time so that it works even with system power off. Such clocks will normally not track the local time zone or daylight savings time -- unless they dual boot with MS-Windows -- but will instead be set to Coordinated Universal Time (UTC, formerly "Greenwich Mean Time").

is equipped with the AT8563 RTC chip, which is a classic industrial grade real-time clock chip (RTC) with an I2C bus interface. It has the characteristics of low power consumption and high accuracy, and is widely used in products such as electricity meters, water meters, gas meters, telephones, etc.

1.1. Functional Introduction

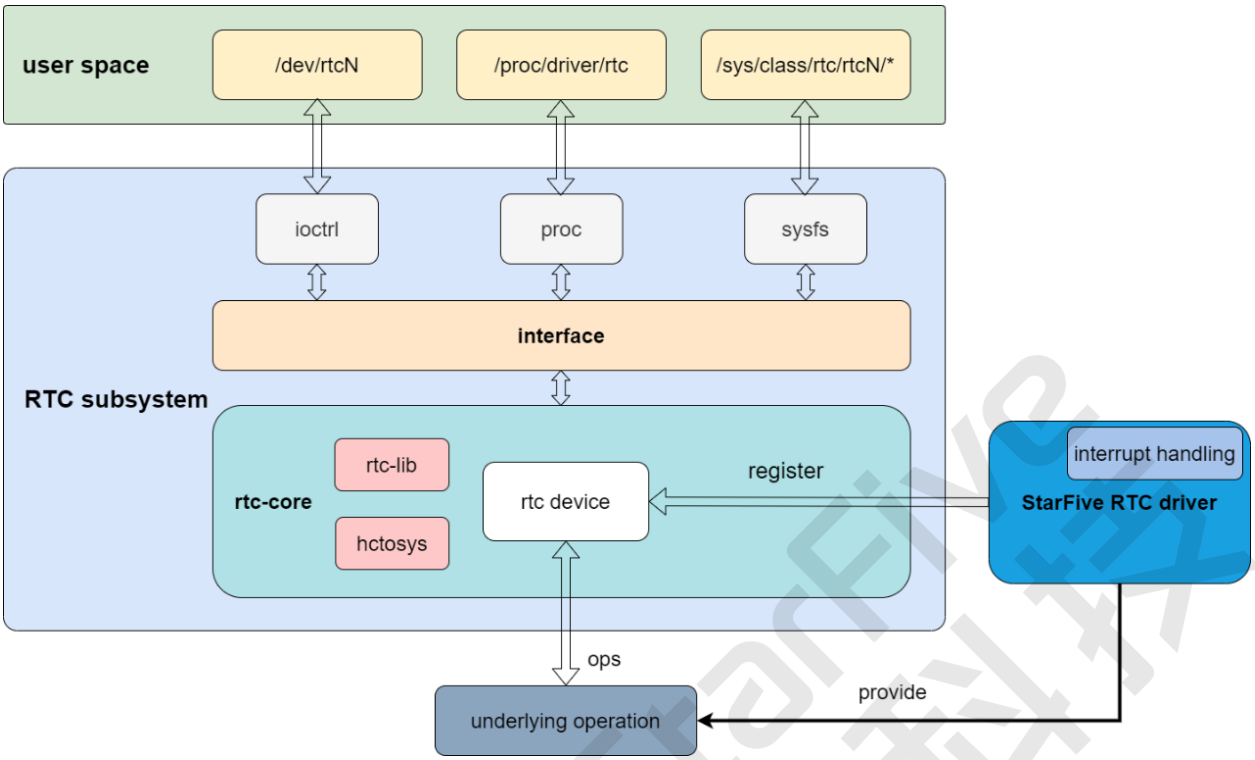
The AT8563 supports the following features and specifications on the *Real Time Clock (RTC)* module.

- Wide working voltage range: 1.0 V to 5.5 V
- The typical value of low sleep current is 0.25 μ A (VDD = 3.0 V, Tamb = 25 ℃)
- With a century symbol, it can work from year 1900 to 2099 (years are related to the century, mainly used to calculate the number of days in February during leap years)
- I2C bus interface supports the frequency up to 400k Hz (VDD = 1.8 V to 5.5V); Read address is 0A3H and write address is 0A2H
- Programmable clock output frequency is 2.768k Hz/1024 Hz/32 Hz/1 Hz
- Contains alarms and timers
- Supports low voltage detection function and power-on reset failure function
- Open drain interrupt output pin

1.2. Block Diagram

The following image shows the block diagram of the RTC driver.

Figure 1-1 Block Diagram



Besides the general user space layer, the following layers are specific to the RTC module.

- **RTC subsystem**: The RTC driver framework provided by the Linux kernel.
  - **rtc-core**: Provide API for RTC drivers to register the RTC devices and drivers.
  - **interface**: Provide interface to interact with the RTC devices (by calling ops functions). So `ioctrl/proc/sysfs` can either be used to set or read RTC time and alarm using API interface.
- **StarFive(AT8563) RTC driver**: Define all underlying operations (read/write register) and register RTC devices to **rtc-core**.

1.3. Device Tree Overview

Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

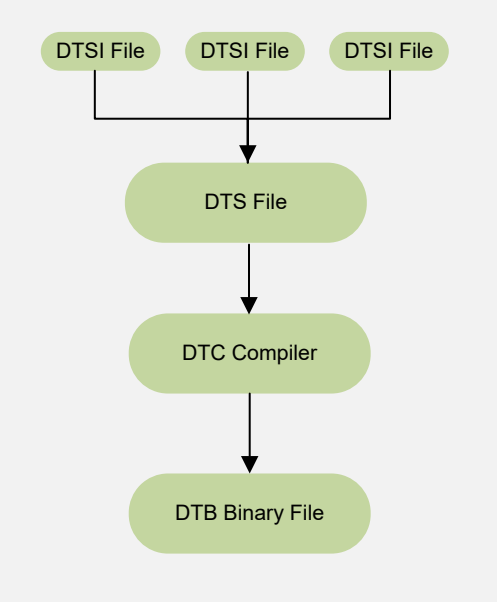
A device tree is primarily represented in the following forms.

- *Device Tree Compiler (DTC)*: The tool used to compile device tree into system-readable binaries.
- *Device Tree Source (DTS)*: The human-readable device tree description file. You can locate the target parameters and modify hardware configuration in this file.

- *Device Tree Source Information (DTSI)*: The human-readable header file which you can include in device tree description. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Blob (DTB)*: The system-readable device tree binary blob files which is burned in system for execution.

The following diagram shows the relationship (workflow) of the above forms.

Figure 1-2 Device Tree Workflow



1.4. Source Code Structure

The source code structure of RTC is listed as follows.

```
linux-5.15.0
├── drivers
| ├── rtc
| | ├── rtc-core.h
| | ├── lib.c
| | ├── interface.c
| | ├── dev.c
| | ├── class.c
| | ├── proc.c
| | ├── sysfs.c
| | ├── rtc-pcf8563.c
```

2. Configuration

2.1. Kernel Menu Configuration

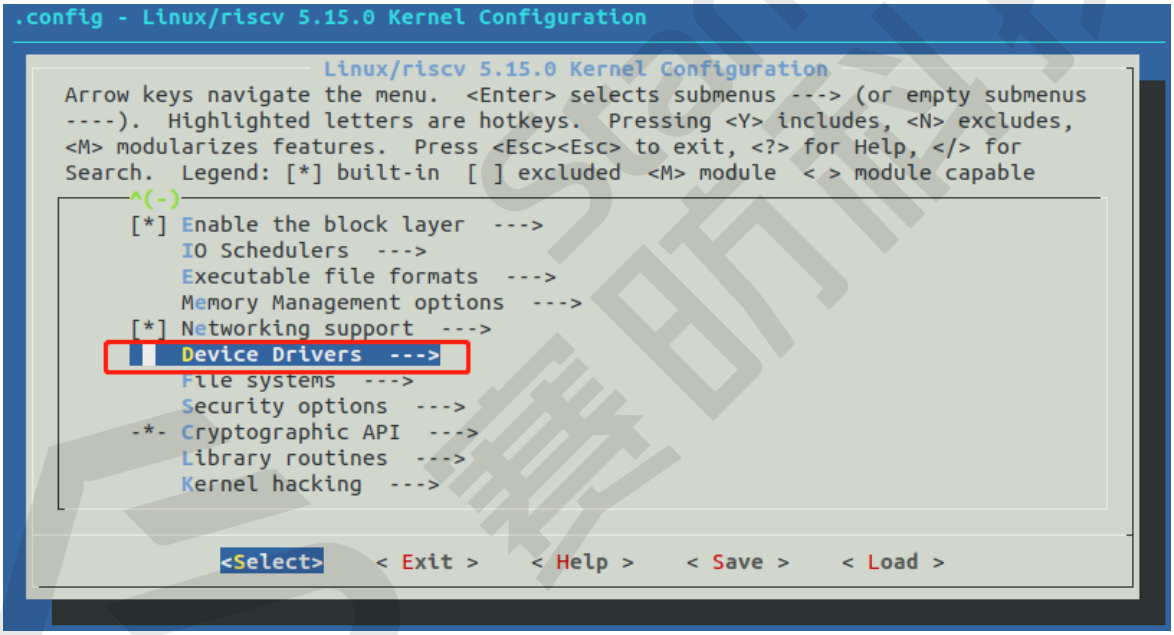
Follow the steps below to enter the kernel menu to enable the kernel configuration for RTC.

1. Under the root directory of `freelight-u-sdk`, type the following command to enter the kernel menu configuration GUI.

```
make linux-menuconfig
```

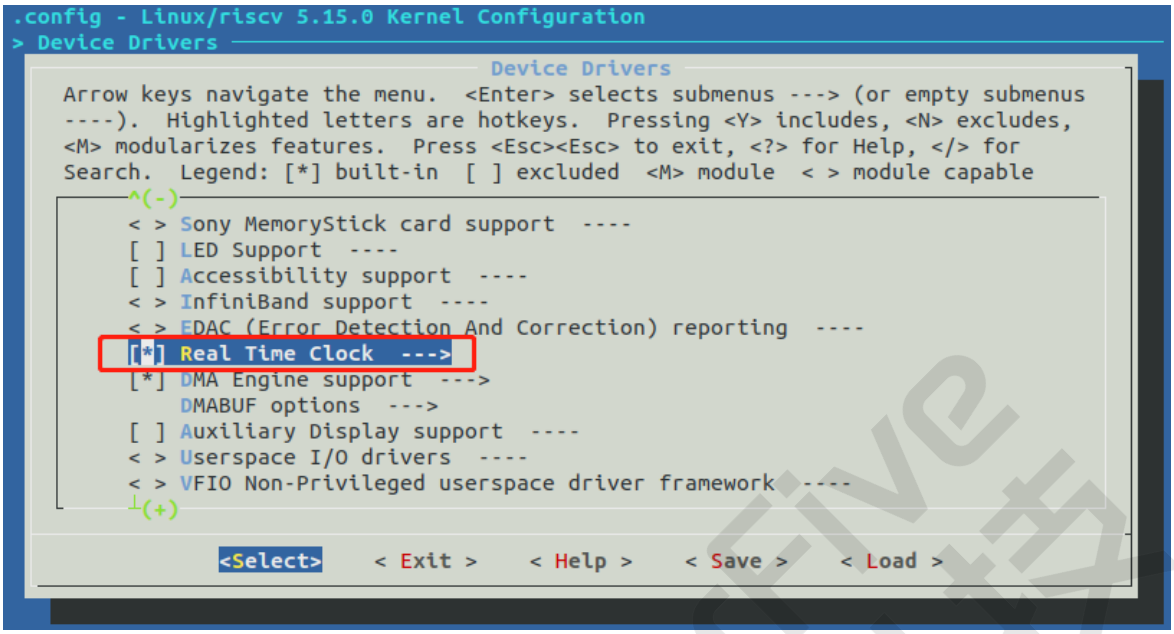
2. Enter the **Device Drivers** menu option.

Figure 2-1 Device Drivers



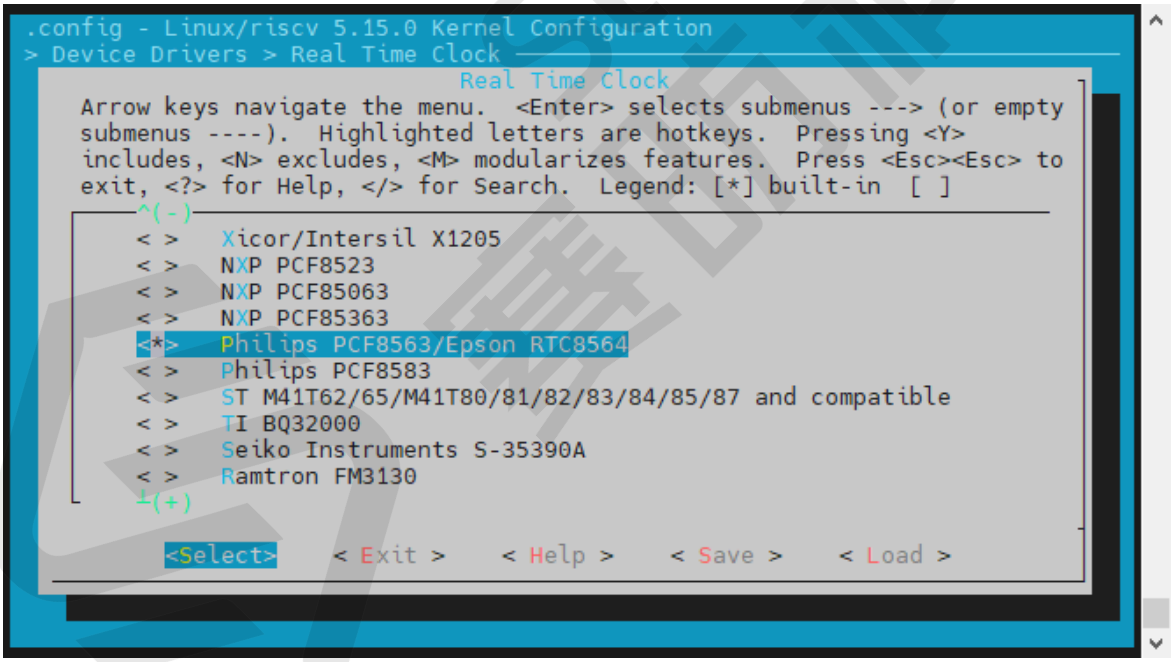
3. Enter the **Real Time Clock** menu option.

Figure 2-2 Real Time Clock



4. Select the **Philips PCF8563/Epson RTC 8564** menu option to enable the RTC driver.

Figure 2-3 Philips PCF8563/Epson RTC 8564



5. Save your change before you exit the kernel configuration dialog.

2.2. Device Tree Source Code

Overview Structure

The device tree source code of JH-7110 is listed as follows:

```
linux
├── arch
| ├── riscv
| | ├── boot
| | | ├── dts
| | | | └── starfive
| | | | ├── codecs
| | | | | ├── sf_ac108.dtsi
| | | | | ├── sf_es8316.dtsi
| | | | | ├── sf_hdmi.dtsi
| | | | | ├── sf_pdm.dtsi
| | | | | ├── sf_pwmdac.dtsi
| | | | | ├── sf_spdif.dtsi
| | | | | ├── sf_tdm.dtsi
| | | | | └── sf_wm8960.dtsi
| | | | ├── evb-overlay
| | | | | ├── jh7110-evb-overlay-can.dts
| | | | | ├── jh7110-evb-overlay-rgb2hdmi.dts
| | | | | ├── jh7110-evb-overlay-sdio.dts
| | | | | ├── jh7110-evb-overlay-spi.dts
| | | | | ├── jh7110-evb-overlay-uart4-emmc.dts
| | | | | ├── jh7110-evb-overlay-uart5-pwm.dts
| | | | | └── Makefile
| | | | ├── jh7110-clk.dtsi
| | | | ├── jh7110-common.dtsi
| | | | ├── jh7110.dtsi
| | | | ├── jh7110-evb-can-pdm-pwmdac.dts
| | | | ├── jh7110-evb.dts
| | | | ├── jh7110-evb.dtsi
| | | | ├── jh7110-evb-dvp-rgb2hdmi.dts
| | | | ├── jh7110-evb-pcie-i2s-sd.dts
| | | | ├── jh7110-evb-pinctrl.dtsi
| | | | ├── jh7110-evb-spi-uart2.dts
| | | | ├── jh7110-evb-uart1-rgb2hdmi.dts
| | | | ├── jh7110-evb-uart4-emmc-spdif.dts
| | | | ├── jh7110-evb-uart5-pwm-i2c-tdm.dts
| | | | ├── jh7110-fpga.dts
| | | | ├── jh7110-visionfive-v2.dts
| | | | ├── jh7110-devkits-wifi.dts
| | | | ├── jh7110-devkits.dts
| | | | ├── Makefile
```

SoC Platform

The device tree source code of the JH-7110 SoC platform is in the following path:

```
devkits/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

Devkits

The device tree source code of the devkits is in the following path:

```
devkits/linux/arch/riscv/boot/dts/starfive/jh7110-devkits.dts
-- devkits/linux/arch/riscv/boot/dts/starfive/jh7110-devkits-pinctrl.dtsi
-- devkits/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

2.3. Device Tree Configuration

Due to the RTC device being mounted on the I2C bus, the RTC device tree node should be added to the I2C device node. The I2C device address of AT8563 RTC is `0×51`. The following is the RTC device tree source code:

```
    rtc@51 {
        compatible = "nxp,pcf8563";
        reg = <0x51>;
    };
```

The following list provides explanations for the parameters included in the above code block.

- **compatible**: Compatibility information, used to associate the driver and its target device.
- **reg**: The I2C device address of RTC is "`0×51`".

3. Interface Description

3.1. Reading or Changing Device Status

The RTC module of JH-7110 Devkit provides the following interfaces to read or change device status.

3.1.1. pcf8563_read_block_data

The interface has the following parameters.

- **Syntax**:

```
static int pcf8563_read_block_data(struct i2c_client
 *client, unsigned char reg,unsigned char length, unsigned char *buf)
```

- **Description**: This function is used to communicate between AT8563 RTC and I2C.
- **Parameter**:
  - **client**: I2C device of the target RTC device.
  - **reg**: Data address.
  - **length**: Data length.
  - **buf**: Read data.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.1.2. pcf8563_write_block_data

The interface has the following parameters.

- **Syntax**:

```
static int pcf8563_write_block_data(struct i2c_client *client,
                  unsigned char reg, unsigned char length,
                  unsigned char *buf)
```

- **Description**: This function is used to communicate between AT8563 RTC and I2C.
- **Parameter**:

  - **client**: I2C device of the target RTC device.
  - **reg**: Data address.
  - **length**: Data length.
  - **buf**: Write data.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.1.3. pcf8563_rtc_read_time

The interface has the following parameters.

- **Syntax**:

```
static int pcf8563_rtc_read_time(struct device *dev, struct rtc_time
 *tm)
```

- **Description**: The function is used to read RTC time information from AT8563 RTC devices.
- **Parameter**:
  - **dev**: The target RTC device.
  - **tem**: The location where RTC time information is stored.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.1.4. pcf8563_rtc_set_time

The interface has the following parameters.

- **Syntax**:

```
static int pcf8563_rtc_set_time(struct device *dev, struct rtc_time
 *tm)
```

- **Description**: The function is used to set the RTC time information on AT8563 RTC device.
- **Parameter**:
  - **dev**: The target RTC device.
  - **tem**: The location where RTC time information is stored.

- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.1.5. pcf8563_rtc_read_alarm

The interface has the following parameters.

- **Syntax**:

```
Static int pcf8563_rtc_read_alarm(struct device *dev, struct rtc_wkalrm
 *alarm)
```

- **Description**: The function is used to read alarm time from AT8563 RTC device.
- **Parameter**:
  - **dev**: The target RTC device.
  - **alarm**: The location where RTC time is stored.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.1.6. pcf8563_rtc_set_alarm

The interface has the following parameters.

- **Syntax**:

```
Static int pcf8563_rtc_set_alarm(struct device *dev, struct rtc_wkalrm
 *alarm)
```

- **Description**: The function is used to set alarm time on AT8563 RTC device.
- **Parameter**:
  - **dev**: The target RTC device.
  - **alarm**: The location where RTC time is stored.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.1.7. pcf8563_irq_enable

The interface has the following parameters.

- **Syntax**:

```
static int pcf8563_irq_enable(struct device *dev, unsigned int enabled)
```

- **Description**: The function is used to set interrupt mode of RTC alarm.
- **Parameter**:
  - **dev**: The target RTC device.
  - **enable**: Interrupt mode.
- **Return**: None.

3.2. Registering Devices

The RTC module of JH-7110 Devkit provides the following interfaces to register an RTC device.

3.2.1. rtc_allocate_device

The interface has the following parameters.

- **Syntax**:

```
static struct rtc_device *rtc_allocate_device(void)
```

- **Description**: The function is used to allocate an RTC device and then initialize it.
- **Parameter**: None
- **Return**:
  - **Success**: The pointer of the allocated RTC device.
  - **Failure**: NULL.

3.2.2. dvem_rtc_allocate_device

The interface has the following parameters.

- **Syntax**:

```
struct rtc_device *devm_rtc_allocate_device(struct device *dev)
```

- **Description**: The function is used to allocate an RTC device with the `devm` method and then initialize it.

- **Parameter**:
  - **dev**: The parent device to which the RTC device belongs.
- **Return**:
  - **Success**: The pointer of the allocated RTC device.
  - **Failure**: NULL.

3.2.3. dvem_rtc_register_device

The interface has the following parameters.

- **Syntax**:

```
#define devm_rtc_register_device(device)
 __devm_rtc_register_device(THIS_MODULE, device)
int __devm_rtc_register_device(struct module *owner, struct rtc_device
 *rtc)
```

- **Description**: The function is used to register an RTC device.
- **Parameter**:
  - **owner**: The module to which the RTC device belongs.
  - **rtc**: The target RTC device to register.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.2.4. dvem_rtc_device_register

The interface has the following parameters.

- **Syntax**:

```
struct rtc_device *devm_rtc_device_register(struct device
 *dev, const char *name, const struct rtc_class_ops *ops, struct module
 *owner)
```

- **Description**: The function is used to re-allocate an RTC device and then register it.
- **Parameter**:
  - **dev**: The parent device to which the RTC device belongs.
  - **name**: Not used.

  - **ops**: The operation function of the RTC device.
  - **owner**: The module to which the RTC device belongs.

- **Return**:
  - **Success**: The pointer of the allocated RTC device.
  - **Failure**: Error code.

4. General Use Case

4.1. `ioctl` Interface

The `ioctl` interface can be used to:

- Obtain an RTC device handle
- Configure RTC time

The following demo program provides an example of using the interface.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/stat.h>
6 #include <fcntl.h>
7 #include <linux/rtc.h>  /* Where the RTC ioctl commands is stored */
8 #include <errno.h>
9 #include <string.h>
10
11 #define RTC_DEVICE_NAME "/dev/rtc0"  /* The device name of the RTC device
 */
12
13 /* Set RTC time */
14 int set_rtc_time(int fd)
15 {
16  struct rtc_time rtc_tm;
17
18  rtc_tm.tm_year = 2022 - 1900; /* year, range: 101-199, which matches
 2001-2099 */
19  rtc_tm.tm_mon = 7 - 1; /* month, range: 0-11, which matches 1-12 */
20  rtc_tm.tm_mday = 15; /* day */
21  rtc_tm.tm_hour = 14; /* hour */
22  rtc_tm.tm_min = 10; /* minute */
23  rtc_tm.tm_sec = 20; /* second */
24
25  if (ioctl(fd, RTC_SET_TIME, &rtc_tm) < 0) {
26      printf("RTC set time failed\n");
27      return -1;
28 }
29
30  return 0;
31 }
32
33 /* Read RTC time */
```

```
34 int read_rtc_time(int fd)
35 {
36  struct rtc_time rtc_tm;
37
38  if (ioctl(fd, RTC_RD_TIME, &rtc_tm) < 0) {
39      printf("RTC read time failed\n");
40      return -1;
41 }
42  printf("RTC time: %04d-%02d-%02d %02d:%02d:%02d\n",
43  rtc_tm_temp.tm_year + 1900, rtc_tm_temp.tm_mon + 1, rtc_tm_temp.tm_mday,
44  rtc_tm_temp.tm_hour, rtc_tm_temp.tm_min, rtc_tm_temp.tm_sec);
45
46  return 0;
47 }
48
49 int main(int argc, char *argv[])
50 {
51  int fd, ret;
52
53  /* Open RTC device */
54  fd = open(RTC_DEVICE_NAME, O_RDWR);
55  if (fd < 0) {
56      printf("Open rtc device %s failed\n", RTC_DEVICE_NAME);
57      return -ENODEV;
58 }
59
60 /* Set time */
61  ret = set_rtc_time(fd);
62  if (ret < 0)
63      return -EINVAL;
64
65 /* Read time */
66  ret = read_rtc_time(fd);
67  if (ret < 0)
68      return -EINVAL;
69
70  close(fd);
71  return 0;
72 }
```

4.2. `proc` Interface

The `proc` interface is only used to read the RTC status.

You can run the following command to execute the read process:

```
cat /proc/driver/rtc
```

The following code block shows an example of using the interface.

```
# cat /proc/driver/rtc
rtc time         : 00:06:19
rtc date         : 2001-02-01
alrm_time        : 00:00:10
alrm_date        : 2001-02-01
alarm_IRQ        : no
alrm_pending     : no
update IRQ enabled     : no
periodic IRQ enabled   : no
periodic IRQ frequency : 1
max user IRQ frequency : 64
24hr             : yes
```

4.3. `sysfs` Interface

The `sysfs` interface can be used to read and set the status of RTC.

The interface only supports setting on the following two parameters:

- **wakealarm**: Run the following command to set a wakeup alarm in X seconds since the current RTC time.

  ```
  echo '+X' > /sys/class/rtc/rtc0/wakealarm
  ```

  Note:

  - The interface is Linux original and the alarm is set in seconds. For example, to set an alarm which rings after 2 hours, use the following command:

    ```
    echo '+7200' > /sys/class/rtc/rtc0/wakealarm
    ```

  - The minimum alarms unit supported by the RTC alarm register is in minutes. If it is less than 1 minute, the next minute is used as the alarm time. For example, if the current time is 09 : 00 : 45, the example command should be:

    ```
    echo '+30' > /sys/class/rtc/rtc0/wakealarm
    ```

    So, the time to trigger alarm should be 09 : 02 : 00.

- **offset**: Used to calibrate the time offset caused by temperature and oscillator. Not recommended to use in the current product release.

The following code block shows a simple example.

```
# cd /sys/class/rtc/rtc0/
# ls
```

```
alarmtimer.0.auto       hctosys         range           uevent
date                    max_user_freq   since_epoch     wakealarm
dev                     name            subsystem
device                  offset          time
# cat date
2001-02-01
# cat time
00:23:57
```