StarFive 赛昉科技

JH7110 RTC Developing Guide

VisionFive 2

Version: 1.0

Date: 2022/11/10

Doc ID: JH7110-DGEN-006

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

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 - JH7110.

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 JH7110.

Revision History

Table 0-1 Revision History

Version	Released	Revision
1.0		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.

Contents

List of Tables....5
List of Figures....6
Legal Statements....ii
Preface....iii
1. Introduction....7
1.1. Functional Introduction....7
1.2. Block Diagram....7
1.3. Device Tree Overview....8
1.4. Source Code Structure....8
2. Configuration....9
2.1. Kernel Menu Configuration....9
2.2. Device Tree Source Code....10
2.3. Device Tree Configuration....11
3. Interface Description....13
3.1. Reading or Changing Device Status....13
3.1.1. rtc_read_time....13
3.1.2. rtc_set_time....13
3.1.3. rtc_read_alarm....13
3.1.4. rtc_set_alarm....14
3.1.5. rtc_read_offset....14
3.1.6. rtc_set_offset....14
3.1.7. rtc_update_irq....15
3.2. Registering Devices....15
3.2.1. rtc_allocate_device....15
3.2.2. dvem_rtc_allocate_device....15
3.2.3. dvem_rtc_register_device....16
3.2.4. dvem_rtc_device_register....16
4. General Use Case....17
4.1. Ioctl Interface....17
4.2. Proc Interface....18
4.3. Sysfs Interface....18

List of Tables

Table 0-1 Revision History.....iii

List of Figures

Figure 1-1 Block Diagram....7
Figure 1-2 Device Tree Workflow....8
Figure 2-1 Device Drivers....9
Figure 2-2 Real Time Clock....10
Figure 2-3 StarFive RTC....10

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").

1.1. Functional Introduction

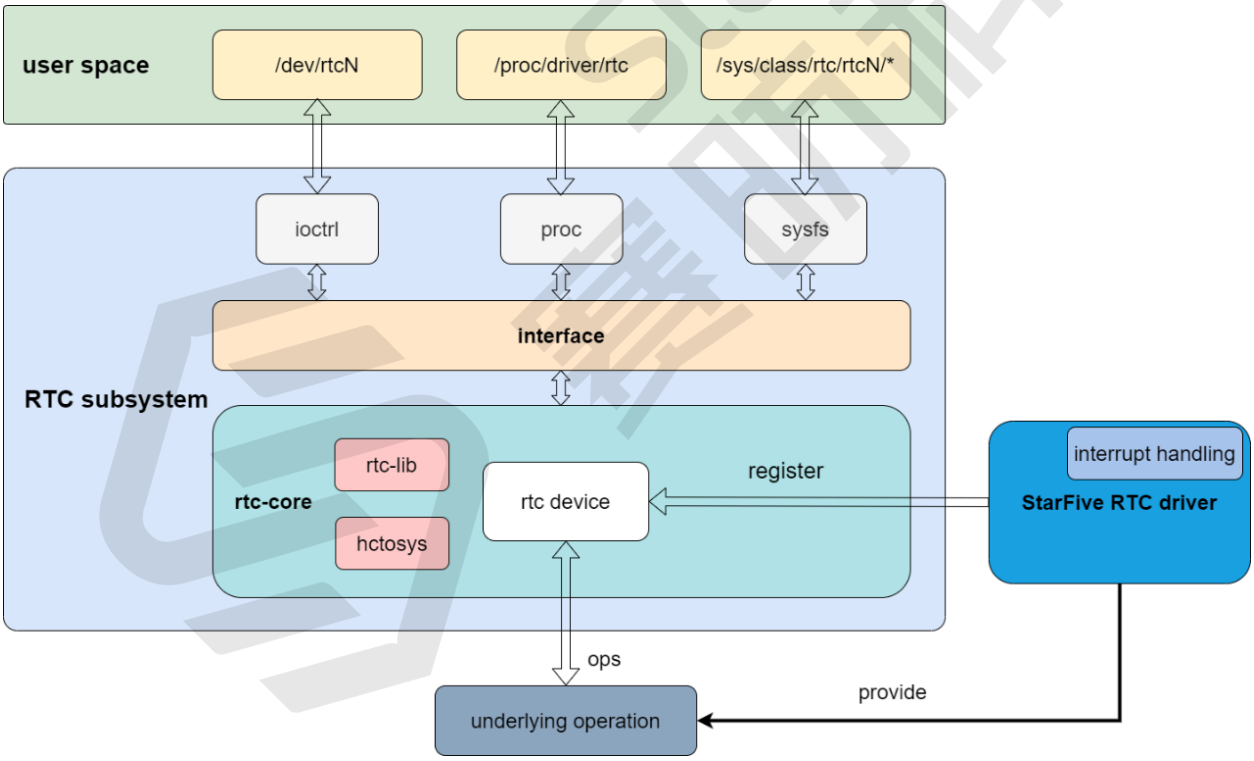
The JH7110 SoC Platform supports the following features and specifications on the *Real Time Clock (RTC)* module.

- Support time range from Jan. 1st 2001 to Dec. 31st 2099
- Support both 24-h (full day) or 12-h (half day) modes
- Allow setting alarms within valid time range
- Hardware calibration

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 RTC driver**: Define all underlying operation (read/write register) and register RTC devices to rtc-core.

1.3. Device Tree Overview

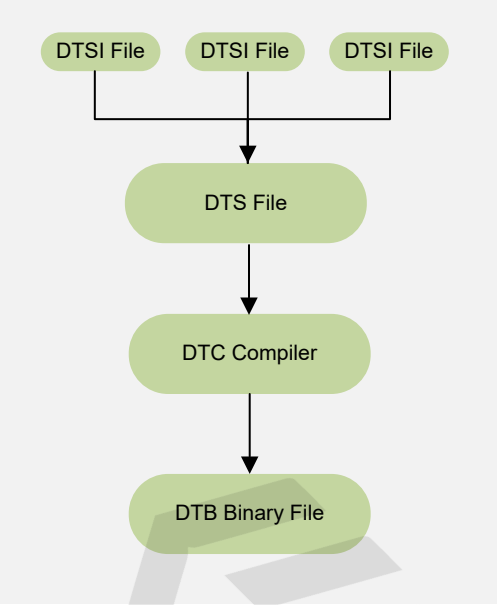
Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

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-starfive.c
```

2. Configuration

2.1. Kernel Menu Configuration

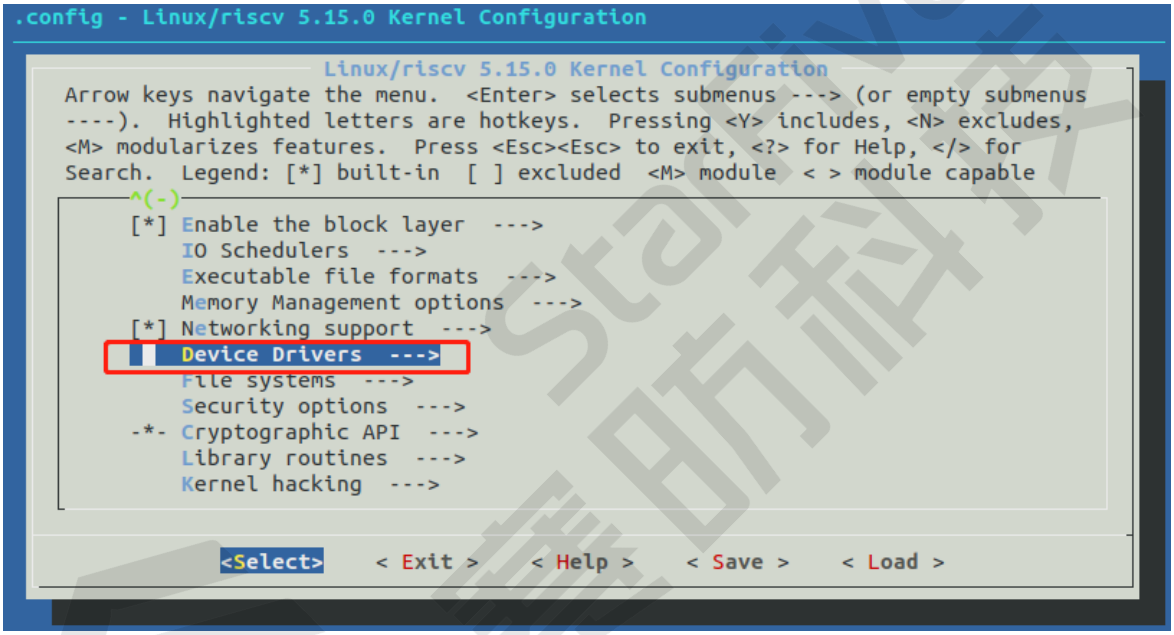
Follow the steps below to enter the kernel menu to enable the kernel configuration for RTC.

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

```
make linux-menuconfig
```

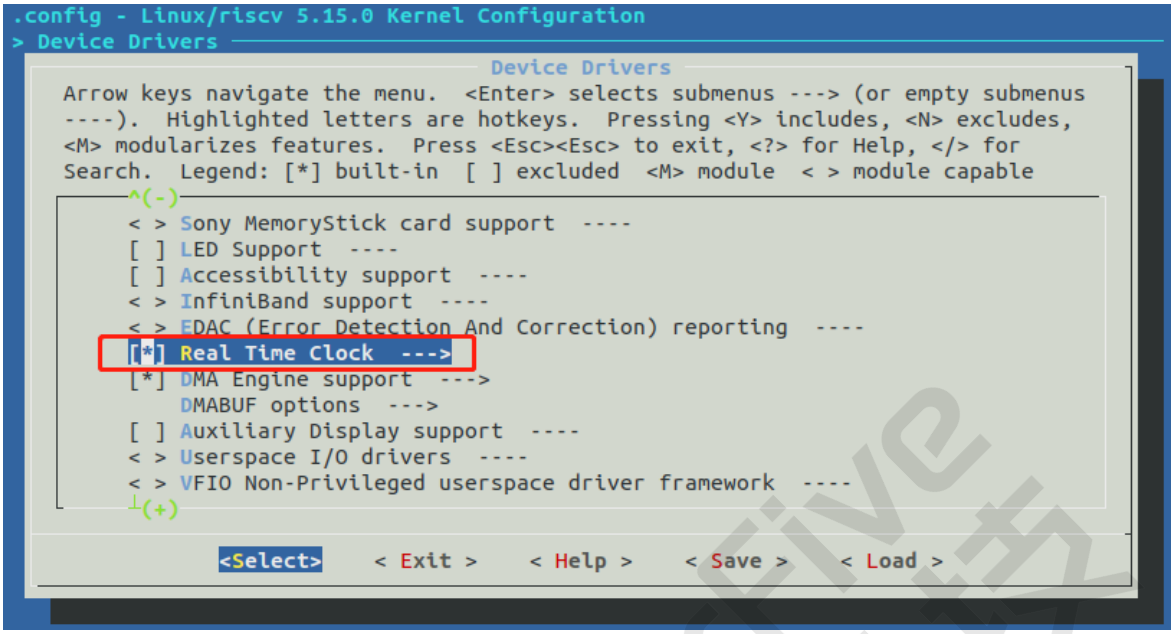
2. Enter the **Device Drivers** menu option.

Figure 2-1 Device Drivers



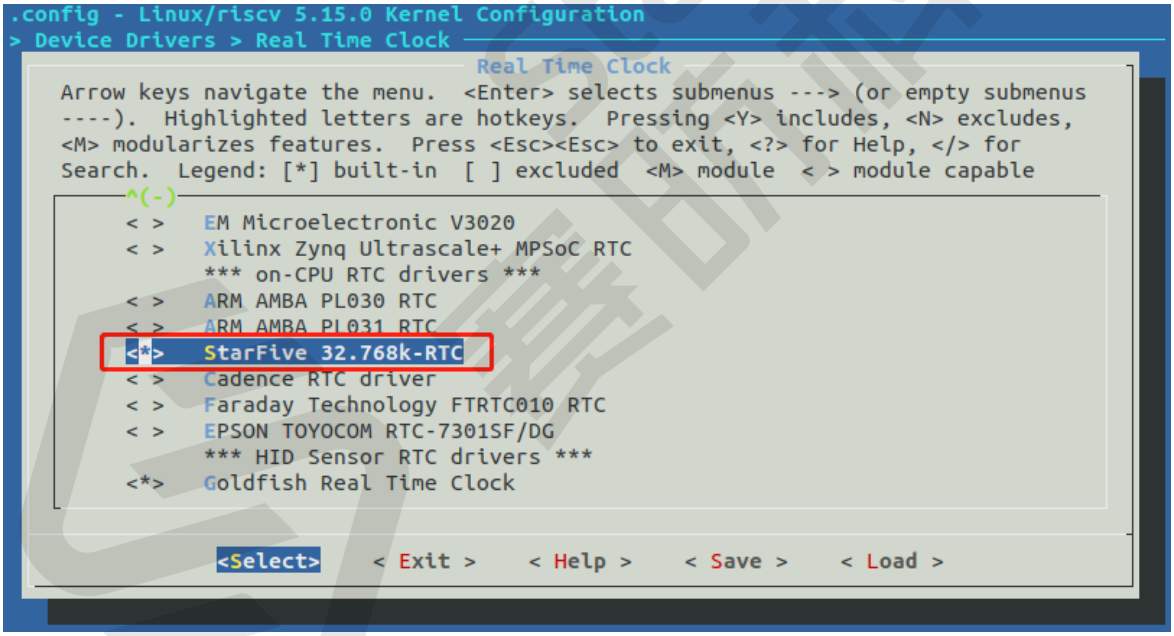
3. Enter the **Real Time Clock** menu option.

Figure 2-2 Real Time Clock



4. Select the **StarFive 32.768k-RTC** menu option to enable the RTC driver.

Figure 2-3 StarFive RTC



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 JH7110 is listed as follows:

```
linux
├── arch
|   ├── riscv
|   |   ├── boot
|   |   |   ├── dts
```

```
|   |   |   |   └── starfive
|   |   |   |       ├── codecs
|   |   |   |       |   ├── 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
|   |   |   |       ├── Makefile
|   |   |   |       └── vf2-overlay
|   |   |   |           ├── Makefile
|   |   |   |           └── vf2-overlay-uart3-i2c.dts
```

SoC Platform

The device tree source code of the JH7110 SoC platform is in the following path:

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

VisionFive 2

The device tree source code of the VisionFive 2 *Single Board Computer (SBC)* is in the following path:

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dts
-- freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-common.dtsi
-- freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

2.3. Device Tree Configuration

All the RTC device tree source code is listed under the `rtc` node of `jh7110.dtsi`.

The following shows more details.

```
        rtc: rtc@17040000 {
            compatible = "starfive,rtc_hms";
            reg = <0x0 0x17040000 0x0 0x10000>;
            interrupts = <10>, <11>, <12>;
            interrupt-names = "rtc_ms_pulse", "rtc_sec_pulse", "rtc";
            clocks = <&clkgen JH7110_RTC_HMS_CLK_APB>,
                     <&clkgen JH7110_RTC_HMS_CLK_CAL>;
            clock-names = "pclk", "cal_clk";
            resets = <&rstgen RSTN_U0_RTC_HMS_APB>,
                     <&rstgen RSTN_U0_RTC_HMS_CAL>,
                     <&rstgen RSTN_U0_RTC_HMS_OSC32K>;
            reset-names = "rst_apb", "rst_cal", "rst_osc";
            rtc,cal-clock-freq = <1000000>;
```

```
            status = "okay";
        };
```

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**: Register base address "`0x17040000`" and range "`0x10000`".
- **interrupts**: Hardware interrupt ID.
- **interrupt-names**: Hardware interrupt names.
- **clocks**: The clocks used by the RTC module.
- **clock-names**: The names of the above clocks.
- **resets**: The reset signals used by the RTC module.
- **reset-names**: The names of the above reset signals.
- **cal-clock-freq**: The work frequency of the clock calibration.
- **status**: The work status of the RTC module. To enable the module, set this bit as "`okay`" or to disable the module, set this bit as "`disabled`".

3. Interface Description

3.1. Reading or Changing Device Status

The RTC module of JH7110 provides the following interfaces to read or change device status.

3.1.1. rtc_read_time

The interface has the following parameters.

- **Syntax**:

```
int rtc_read_time(struct rtc_device *rtc, struct rtc_time *tm)
```

- **Description**: The function is used to read RTC time information from a specific RTC device.
- **Parameter**:
  - **rtc**: The target RTC device.
  - **tm**: The location where RTC time information is stored.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.1.2. rtc_set_time

The interface has the following parameters.

- **Syntax**:

```
int rtc_set_time(struct rtc_device *rtc, struct rtc_time *tm)
```

- **Description**: The function is used to set RTC time information on a specific RTC device.
- **Parameter**:
  - **rtc**: The target RTC device.
  - **tm**: The location where RTC time information is stored.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.1.3. rtc_read_alarm

The interface has the following parameters.

- **Syntax**:

```
int rtc_read_alarm(struct rtc_device *rtc, struct rtc_wkalrm *alarm)
```

- **Description**: The function is used to read alarm time from a specific RTC device.
- **Parameter**:

- **rtc**: The target RTC device.
- **alarm**: The location where alarm time is stored.

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

3.1.4. rtc_set_alarm

The interface has the following parameters.

- **Syntax**:

```
int rtc_set_alarm(struct rtc_device *rtc, struct rtc_wkalrm *alarm)
```

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

3.1.5. rtc_read_offset

The interface has the following parameters.

- **Syntax**:

```
int rtc_read_offset(struct rtc_device *rtc, long *offset)
```

- **Description**: The function is used to read the count of RTC offset in parts per billion.
- **Parameter**:
  - **rtc**: The target RTC device.
  - **offset**: The offset in parts per billion.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.1.6. rtc_set_offset

The interface has the following parameters.

- **Syntax**:

```
int rtc_set_offset(struct rtc_device *rtc, long offset)
```

- **Description**: The function is used to adjust the duration of the average second.
- **Parameter**:
  - **rtc**: The target RTC device.
  - **offset**: The offset in parts per billion.

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

3.1.7. rtc_update_irq

The interface has the following parameters.

- **Syntax**:

```
void rtc_update_irq(struct rtc_device *rtc, unsigned long num, unsigned long events)
```

- **Description**: The function is used to clear the outdated alarms when an RTC interrupt occurs.
- **Parameter**:
  - **rtc**: The target RTC device.
  - **num**: The number of the IRQ events reported. (Usually only 1 event is reported.)
  - **events**: The mask of RTC_IRQF with one or more of RTC_PF, RTC_AF, RTC_UF.
- **Return**: None

3.2. Registering Devices

The RTC module of JH7110 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 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
  ```

- **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
```