# RTC Device Driver User Guide

*UG7407; Rev 1; 12/21*


## Abstract

This document provides an overview to the Linux real-time clock (RTC) Device Drivers and information on how to program and evaluate the MAX31343 RTC on the MAX31343 shield board (MAX31343SHLD) in a Linux environment.

# Table of Contents

## List of Figures

## List of Tables



# 1. Linux Device Driver

This user guide explains Linux Device Driver concepts, especially the RTC drivers.

## 1.1 What is a Device Driver?

Device drivers are part of the Linux kernel. They make user space commands independent from system hardware designs so that there is no need for knowledge on hardware. There are generally two groups of users: board manufacturers and end customers.

## 1.2 Ways to Implement a Driver as Kernel Module

The Maxim RTC driver is implemented in two different ways:

1. Driver is released to kernel.org that is the main distribution point of source code for the Linux kernel.
    a. The driver is included in the kernel that comes with the future Linux distribution. To compile the driver, KConfig (which is a Linux compilation feature extraction interface) must be configured to include the driver. See the *Kernel Compilation Procedure* section for more information.
2. Download driver from the Maxim website.
    a. Add the driver into the downloaded kernel. Then, compile with the whole kernel. For more details, see the *Kernel Compilation Procedure* section.

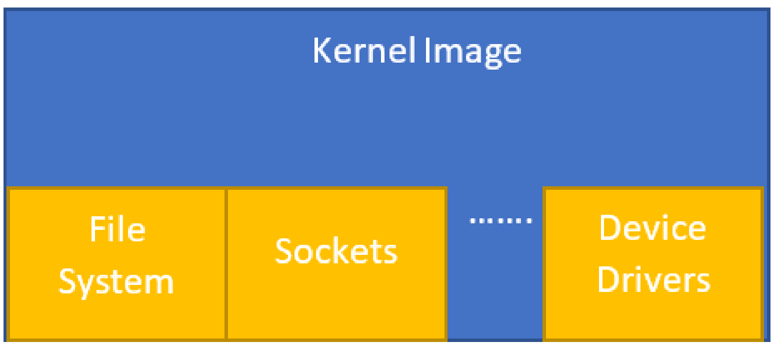



*Figure 1. Kernel image w/out modules.*

    b. Linux has a proper way to inject a driver into the running Linux distribution or Linux kernel module (LKM). The LKMs are not part of the main kernel. They are injected by users into the running kernel. The user can determine which LKM driver to include for kernel compilation. This helps to reduce the size of the kernel. For example, the ethernet LKM can be eliminated in a kernel compilation if an ethernet interface is not needed in a system. So, the bootloader and kernel module loader do not try to load the LKM into the RAM.

The LKM files are usually kept in the /lib/modules folder. They are loaded according to the distribution configuration such as device tree, scripts, etc.

### 1.3 Device Driver Structure

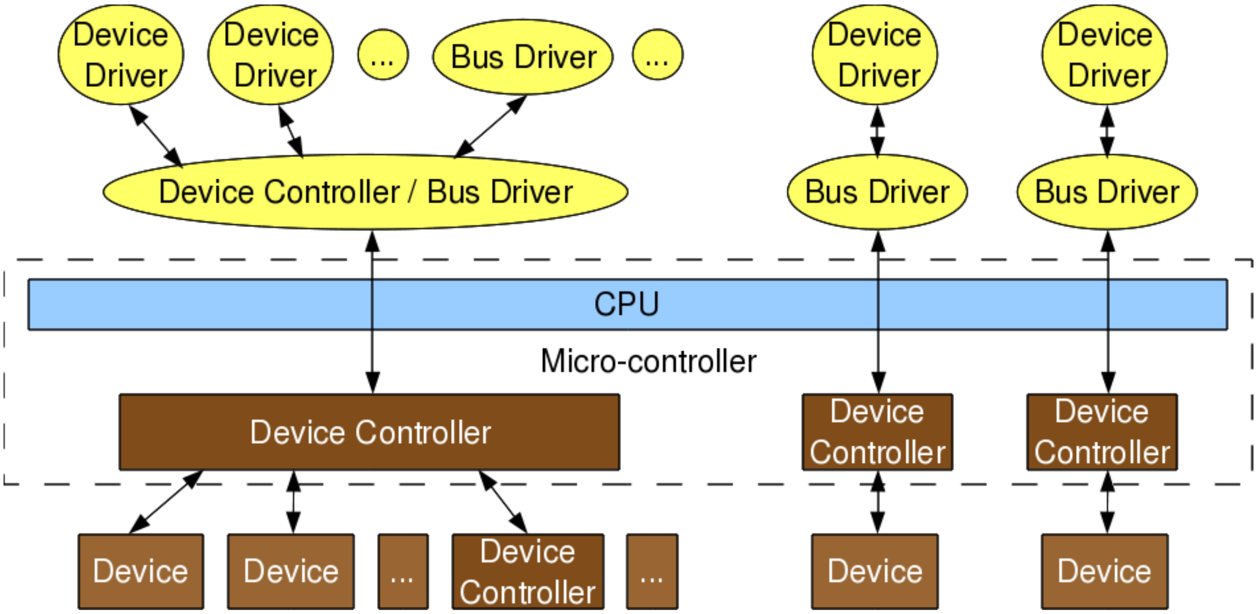



*Figure 2. Linux device management [1].*

Kernel developers generate Linux's generic device driver. The device driver must support all the required features and functions. The driver code should report it in the device driver structures if optional features are not implemented.

## 2. Device Tree Parameters

Device drivers must learn hardware parameters due to the nature of the hardware environment. For example, the CPU's model, active core number, clock frequency, board's memory inputs, memory over bus drivers, which device is connected to which bus, almost everything is described in the device tree of the system. This feature supports Arm after Linux Kernel Version 3.7. The device tree is the hardware description for the kernel usually provided by the board manufacturer/provider.

### 2.1 RTC Device Trees

The RTC device trees usually contain information on the bus and signals connected to the RTC. Rest of the settings are device-specific parameters dependent on the RTC IC model and on-board design.

The bus number, interrupt lines, and device addresses are interpreted within the kernel. So, the driver only gets necessary bus structures from high-level application program interfaces (APIs). However, the device driver must have parser and default values for custom parameters like trickle charger settings or power management mode.

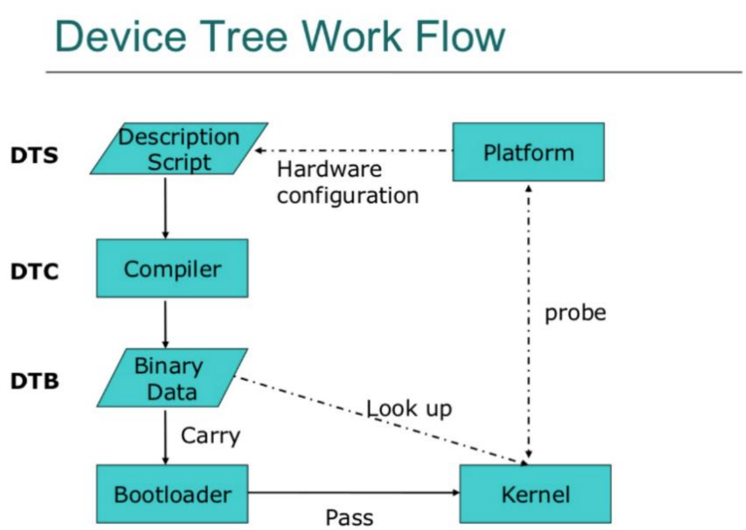



*Figure 3. Device tree workflow [2].*

### 2.2 Device Tree Compilation

The device tree has its own compiler to generate output. More information can be found here: [device-tree-compiler/manual.txt at master · vagrantc/device-tree-compiler · GitHub](device-tree-compiler/manual.txt at master · vagrantc/device-tree-compiler · GitHub).

## 3. Linux Generic RTC Device Drivers

The RTC device driver is a generic device driver for RTC ICs from manufacturers. System calls must be used in user space programs to access driver features from the user space. The IOCTL call is used to access the RTC features for the RTC driver.

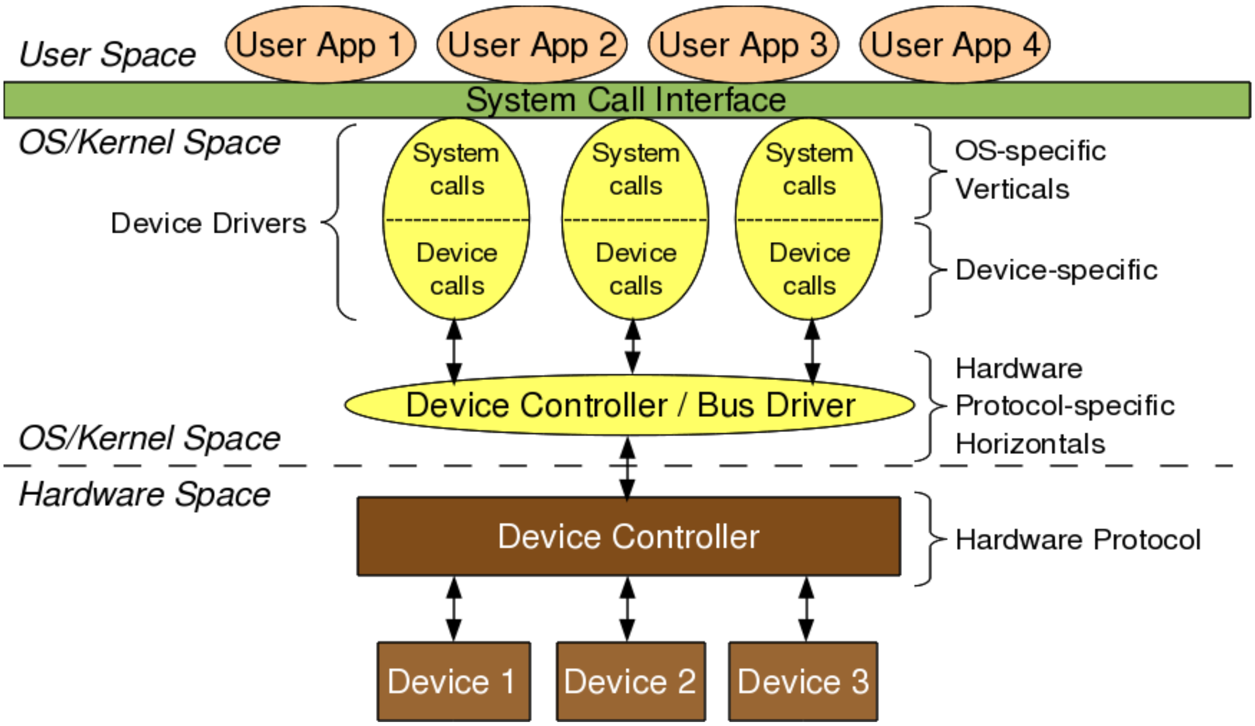



*Figure 4. System calls to device drivers [1].*

### 3.1 RTC-Specific Default Features

M: Must have, O: Optional, N: Nice to have.

An invalid argument system code is returned to the user space if the must-have and optional features are not implemented. The user space programs can read the return code and respond to it if needed when the optional features are not implemented. Everything must be implemented for the must-have options.

Nice-to-have features may not be implemented in the kernel space.

## Table 1. Default RTC Features.

| LINUX OPTIONS | DESCRIPTION |
|---|---|
| Set/Read Time (M) | RTC read/set time support |
| Set/Read Alarm (O) | Set/read alarm per user request |
| Periodic Interrupt (N) | Periodic interrupt support (2Hz to 8192Hz) |
| Alarm Interrupt (O) | Alarm interrupt generation based on user’s alarm request |
| Update Interrupt (O) | On-demand update interrupt, usually 1 Hz |
| EPOCH (N) | RTC set/get time EPOCH format |
| NVMEM (N) | Battery-supported RAM or EEPROM-based memory within the RTC IC |
| Linux Power Management (N) | Registering a device as the wakeup of the system. Developer must set the CONFIG_PM_SLEEP option in the compilation parameters. |

An RTC driver is ready to release on kernel.org and work in a Linux system with the features mentioned in Table 1.

### 3.2 Device-Specific Custom Features

The RTC ICs may have more features than Linux requires. These features can be utilized through the IOCTL system calls and/or SysFs file interface.

Using the file APIs can change the device attribute or configuration using sysfs. A single sysfs file usually maps to a single attribute and is usually readable (and/or writable) using a simple text string. For example, the use of **cat** to read the state of the power management configuration and the **echo** shell command to change it.

**Sysfs** is a pseudo file system that can be used by the end/mid user. Those files are usually read-only files. But, some can take parameters through the command line. For example,

- $ echo out >/sys/class/gpio/gpio24/direction
- $ cat /sys/class/gpio/gpio24/direction

  out
- $ echo 1 >/sys/class/gpio/gpio24/value

The **cat** program reads data from a file in the Linux environment. Likewise, the **echo** program writes data to a file.

Every IOCTL option has its own function-specific code. The user and kernel spaces must have the same code numbers. So, the user space programs can use IOCTL functions through system calls.

Again, the user can use the command line or Linux’s file API to access the sysfs as files.

The end/mid user must know the parameters to give and read in both the options. Figure 5 shows an example.

```
#define MXC_RTC_REG_READ             _IOWR('p', 0x20, int)
#define MXC_RTC_REG_WRITE            _IOW('p',  0x21, struct reg_data_s)

#define MXC_RTC_PWR_MGMT_READ        _IOR('p', 0x22, int)
#define MXC_RTC_PWR_MGMT_WRITE       _IOW('p',  0x23, int)

#define MXC_RTC_ALARM2_CONF_WRITE    _IOW('p', 0x24, struct alarm2_conf_s)
#define MXC_RTC_ALARM2_CONF_READ     _IOR('p', 0x25, struct alarm2_conf_s)

#define MXC_RTC_EXT_CLK_READ         _IOR('p',  0x26, int)
#define MXC_RTC_EXT_CLK_WRITE        _IOW('p',  0x27, int)

#define MXC_RTC_DATA_RET_READ        _IOR('p',  0x28, int)
#define MXC_RTC_DATA_RET_WRITE       _IOW('p',  0x29, int)
```

*Figure 5. IOCTL parameters for MAX31343.*

## 4. MAX31343 Device Driver Package

### 4.1 Supported Features

**Table 2. MAX31343 Kernel Version's Features.**

| STANDARD LINUX RTC CONFIGURATION OPTIONS | STATUS |
|---|---|
| Set/Read Time | Supported |
| Set/Read Alarm | Supported |
| Periodic Interrupt | Not Supported |
| Alarm Interrupt | Supported |
| Update Interrupt | Supported |
| EPOCH | N/A |
| Linux Power Management | Supported |
| **NVMEM** | Supported |

Here is the driver for Linux Kernel 4.x.x.

rtc-max31343.c

**Table 3. MAX31343 Pin Configuration.**

| MAX31343 | RPi 3 B+ |
|---|---|
| $V_{CC}$ | Pin1 - 3V3 |
| GND | Pin6 - Ground |
| SDA | Pin3 - GPIO2 |
| SCL | Pin5 - GPIO3 |
| INTA | Pin36 - GPIO16 |

a. The driver must be compiled from the source code for portability. The Raspberry Pi kernel must be updated to the latest version before the compilation as shown in Figure 6.

- sudo apt-get update && sudo apt-get install --reinstall raspberrypi-bootloader raspberrypi-kernel

  The RPi must then reboot with the **reboot** command.

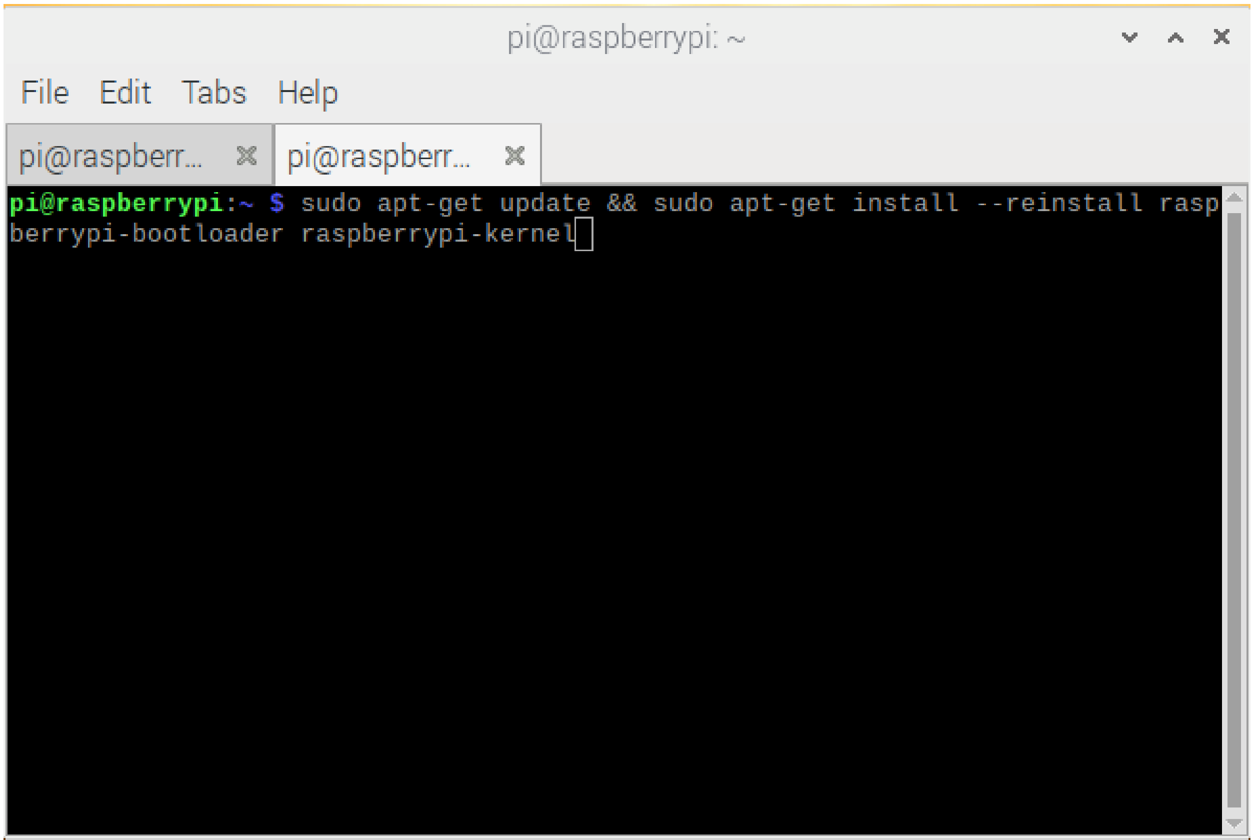



*Figure 6. Commands to prepare the Raspberry Pi for device driver compilation.*

b. Kernel headers must be installed to compile the MAX31343 LKM.

- sudo apt-get install raspberrypi-kernel-headers

  The command installs the necessary kernel headers for development.

c. RPi 3 B+ does not use $i^2c$ automatically. So, enable the $i^2c$ bus and RTC in /boot/config.txt.

- sudo nano /boot/config.txt

  Use the command to open the file and add the following lines:

  - dtparam=i2c_arm=on
  - dtoverlay=i2c-rtc,MAX31343

d. A Makefile is required to create the LKM from the driver file. A Makefile is provided to work with Raspberry Pi 3B+. The Makefile can be used to compile the driver, device tree, and install the LKM to run the Linux system.

- Go to the folder with the Makefile.
- The **make clean** command clears all outputs of the driver.
- The **make all** command compiles the driver and generates the LKM (.ko) file as shown in Figure 7.

```
pi@raspberrypi:~/Desktop/max31343_linux_driver $ make all
make -C /lib/modules/4.19.97-v7+/build M=/home/pi/Desktop/max31343_linux_driver modules
make[1]: Entering directory '/usr/src/linux-headers-4.19.97-v7+'
  CC [M]  /home/pi/Desktop/max31343_linux_driver/rtc-max31343.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/pi/Desktop/max31343_linux_driver/rtc-max31343.mod.o
  LD [M]  /home/pi/Desktop/max31343_linux_driver/rtc-max31343.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.19.97-v7+'
sudo cp rtc-max31343.ko /lib/modules/4.19.97-v7+/kernel/drivers/rtc
pi@raspberrypi:~/Desktop/max31343_linux_driver $
```

*Figure 7. Command for compiling the device driver.*

- The **sudo make install** command installs the driver into the LKM folder as shown in Figure 8. Most system folders are not accessible to users. Use the **sudo** command to allow **make** command to make changes to system directories. This step is required for installing the driver to the system.

```
pi@raspberrypi:~/Desktop/max31343_linux_driver $ sudo make install
make -C /lib/modules/4.19.97-v7+/build M=/home/pi/Desktop/max31343_linux_driver INSTALL_MOD_PATH= modules_install
make[1]: Entering directory '/usr/src/linux-headers-4.19.97-v7+'
  INSTALL /home/pi/Desktop/max31343_linux_driver/rtc-max31343.ko
  DEPMOD  4.19.97-v7+
Warning: modules_install: missing 'System.map' file. Skipping depmod.
make[1]: Leaving directory '/usr/src/linux-headers-4.19.97-v7+'
depmod -A
pi@raspberrypi:~/Desktop/max31343_linux_driver $
```

*Figure 8. Command for installing the device driver to the system.*

- The **make dtbs** command compiles the device tree overlay (i$^2$c-rtc-overlay.dts) as shown in Figure 9 and generates the i$^2$c-rtc.dtbo output. It copies the output to RPi's /boot/overlays folder if the compilation is successful.

```
pi@raspberrypi:~/Desktop/max31343_linux_driver $ make -q dtbs
pi@raspberrypi:~/Desktop/max31343_linux_driver $
```

*Figure 9. Command for compiling and installing the device tree overlay.*

Restart the Raspberry Pi after these steps. The driver is ready for testing.

### *4.1.1 Basic Tests*

The **timedatectl** and **hwclock** commands can be used for basic tests such as setting/reading time. These commands are provided by the Raspberry Pi OS. So, they do not require any installation.

a. Manually disable the Network Time Protocol (NTP) before any test. The NTP updates the system time periodically if not disabled.

- sudo timedatectl set-ntp no

  The command disables the NTP as shown in Figure 10.

```
pi@raspberrypi:~/Desktop $ timedatectl
               Local time: Mon 2020-01-27 11:49:04 GMT
           Universal time: Mon 2020-01-27 11:49:04 UTC
                 RTC time: Mon 2020-01-27 11:49:04
                Time zone: Europe/London (GMT, +0000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
pi@raspberrypi:~/Desktop $ sudo timedatectl set-ntp no
pi@raspberrypi:~/Desktop $ timedatectl
               Local time: Mon 2020-01-27 11:49:18 GMT
           Universal time: Mon 2020-01-27 11:49:18 UTC
                 RTC time: Mon 2020-01-27 11:49:18
                Time zone: Europe/London (GMT, +0000)
System clock synchronized: yes
              NTP service: inactive
          RTC in local TZ: no
pi@raspberrypi:~/Desktop $
```

*Figure 10. Commands for disabling NTP.*

b. The **hwclock** command can be used to set the time in the RTC.

- Set RTC time: “sudo hwclock  -- set --date “1/27/2020 14:50:00”

```
pi@raspberrypi:~/Desktop $ sudo hwclock --set --date "1/27/2020 14:50:00"
```

*Figure 11. Command for changing the RTC time.*

- Read RTC time: “timedatectl” or “sudo hwclock -r”

```
pi@raspberrypi:~/Desktop $ timedatectl
               Local time: Mon 2020-01-27 11:50:20 GMT
           Universal time: Mon 2020-01-27 11:50:20 UTC
                 RTC time: Mon 2020-01-27 14:50:03
                Time zone: Europe/London (GMT, +0000)
System clock synchronized: yes
              NTP service: inactive
          RTC in local TZ: no
```

*Figure 12. Reading RTC time using systemctl.*

```
pi@raspberrypi:~/Desktop $ sudo hwclock -r 2020-01-27 11:47:05.120164+00:00
```

*Figure 13. Reading RTC time using hwclock.*

- Synchronizing system clock with RTC: “sudo  hwclock --hctosys”

```
pi@raspberrypi:~/Desktop $ sudo hwclock --hctosys

pi@raspberrypi:~/Desktop $ timedatectl
               Local time: Mon 2020-01-27 14:50:09 GMT
           Universal time: Mon 2020-01-27 14:50:09 UTC
                 RTC time: Mon 2020-01-27 14:50:09
                Time zone: Europe/London (GMT, +0000)
System clock synchronized: no
              NTP service: inactive
          RTC in local TZ: no
```

*Figure 14. Sycnhronizing system clock with RTC.*

#### *4.1.2 SysFs Entry Test of the Maxim Version Driver*

There is only one writable sysfs entry (userram) on this version of the driver. This interface is a nice-to-have feature from the Linux kernel, which is defined as the sysfs. Other sysfs entries are there to read some of the important configurations. Sysfs entries can be found in the “/sys/class/rtc/rtc<device_no>/device” folder.

a. The **cat power_mgmt** command reads the current power-management mode and backup battery threshold from the RTC as shown in Figure 15. Change these settings using the IOCTL or device tree.

```
pi@raspberrypi: ~/Desktop
File  Edit  Tabs  Help
pi@raspberrypi:~ $
pi@raspberrypi:~ $
pi@raspberrypi:~ $
pi@raspberrypi:~ $ cd /sys/class/rtc/rtc0/device
pi@raspberrypi:/sys/class/rtc/rtc0/device $ cat power_mgmt
Reg Value = 0x01: Power Management Auto and Trickle Charger
Backup Battery Threshold = 2.2V
pi@raspberrypi:/sys/class/rtc/rtc0/device $
pi@raspberrypi:/sys/class/rtc/rtc0/device $
pi@raspberrypi:/sys/class/rtc/rtc0/device $
```

*Figure 15. Reading power management mode.*

b. The **cat trickle_charger** command reads the trickle charger settings as shown in Figure 16. Change the settings in the device tree.

```
root@raspberrypi:/sys/class/rtc/rtc0/device# cat trickle_charger
3k Ohm in series with a Schottky diode
```

*Figure 16. Reading the trickle charger settings.*

c. A sysfs entry allows only a root user to write data.

- sudo -s

Enter the command mentioned above to change the user mode.

```
pi@raspberrypi:/sys/class/rtc/rtc0/device $ sudo -s
```

*Figure 17. Command for changing the user mode.*

The userram (nvmem) is a binary interface unlike other sysfs entries. So, the **cat** command does not show meaningful data.

- hexdump -C userram

  Use this command to read as shown in Figure 18.

```
root@raspberrypi:/sys/devices/platform/soc/3f804000.i2c/i2c-1/1-0069# hexdump -C userram
00000000  c7 cc f7 81 5b 39 49 ab  08 07 c8 fa 53 ad d6 74  |....[9I.....S..t|
00000010  0c 3a 65 ef 0d 13 01 06  3f 19 12 fb 7c 5c 86 44  |.:e.....?...|\.D|
00000020  28 7d c5 83 b6 0e 2e bf  15 f7 b9 68 a4 90 dc b1  |(}.........h....|
00000030  ca 42 a0 d7 55 a1 dd 94  ba ef 90 36 4b 16 7a 73  |.B..U......6K.zs|
00000040
```

*Figure 18. Reading RTC userram with hexdump.*

The root user access is a must to write data to userram.

- echo “<data>” >> userram

  The command works after getting access.

- echo -n -e “<data in binary format>” >> userram

  The command works to write data in binary.

```
pi@raspberrypi:/sys/devices/platform/soc/3f804000.i2c/i2c-1/1-0069 $ sudo su
root@raspberrypi:/sys/devices/platform/soc/3f804000.i2c/i2c-1/1-0069# echo "Maxim Integrated" >> userram
root@raspberrypi:/sys/devices/platform/soc/3f804000.i2c/i2c-1/1-0069# hexdump -C userram
00000000  4d 61 78 69 6d 20 49 6e  74 65 67 72 61 74 65 64  |Maxim Integrated|
00000010  0a 98 c0 dd 20 44 2f ba  8d 1f 89 c0 40 56 a8 10  |.... D/.....@V..|
00000020  38 ec b4 16 55 59 17 61  08 08 38 7d 5f a1 6a 26  |8...UY.a..8}_.j&|
00000030  13 b8 24 4d bc b8 ac 00  b8 8a ad ca 78 51 95 34  |..$M........xQ.4|
00000040
root@raspberrypi:/sys/devices/platform/soc/3f804000.i2c/i2c-1/1-0069# echo -n -e '\x01\x02\x03\x04\x05\x06\x07' >> userram
root@raspberrypi:/sys/devices/platform/soc/3f804000.i2c/i2c-1/1-0069# hexdump -C userram
00000000  01 02 03 04 05 06 07 6e  74 65 67 72 61 74 65 64  |.......ntegrated|
00000010  0a 98 c0 dd 20 44 2f ba  8d 1f 89 c0 40 56 a8 10  |.... D/.....@V..|
00000020  38 ec b4 16 55 59 17 61  08 08 38 7d 5f a1 6a 26  |8...UY.a..8}_.j&|
00000030  13 b8 24 4d bc b8 ac 00  b8 8a ad ca 78 51 95 34  |..$M........xQ.4|
00000040
root@raspberrypi:/sys/devices/platform/soc/3f804000.i2c/i2c-1/1-0069#
```

*Figure 19. Writing data to RTC userram.*

d. Use the **cat additional_interrupt** command to see the status of the remaining interrupts not used by the kernel. These are analog power fail, D1 input, loss of external clock, and oscillator failed interrupts.

If no interrupt is asserted:

```
pi@raspberrypi:/sys/class/rtc/rtc0/device $ cat additional_interrupt
This section shows EIF, ANA_IF, OSF and LOS interrupts.
NONE
```

*Figure 20. Reading the additional interrupt status.*

If interrupts are asserted:

```
This section shows EIF, ANA_IF, OSF and LOS interrupts.
        Last Interrupts
LOS Int.: LOS of signal.
Analog Int.: Analog interrupt flag/Power fail flag.
External Int.: External interrupt flag for D1
```

*Figure 21. Command output when the interrupts are asserted.*

e. A test program, rtc_ctest, is provided to test the IOCTL options including the userram, update interrupt, alarm interrupt, and sysfs read values. Call the program with security privilege.

- sudo chmod 777 rtc_ctest

  Run the command mentioned above to ensure the file is readable, writable, and executable.

- sudo ./rtc_ctest

The command calls the program.

```
pi@raspberrypi:~ $ sudo ./Desktop/latest_driver/rtc_test/rtc_ctest/bin/Debug/rtc_ctest

                        RTC Driver Test Example.


***********IOCTL register read test***********
Address: 0x59 Value = 0x10
IOCTL register write test Address: 0x55 Value = 0xAA
IOCTL register write/read test SUCCESS!!!
***********IOCTL Data Retention Access Test***********
IOCTL Data Retention test SUCCESS!!!!
***********IOCTL External Clock Access Test***********
IOCTL External Clock test SUCCESS!!!!
***********IOCTL Power Management Access Test***********
IOCTL Power Management test SUCCESS!!!!
***********IOCTL Alarm2 Test***********
IOCTL Alarm2 set for one minute Min:14
IOCTL Waiting alarm2
IOCTL  Alarm2 test SUCCESS!!!!

                Update IRQ Test Started

Counting 5 update (1/sec) interrupts from reading /dev/rtc0: 1 2 3 4 5
Again, from using select(2) on /dev/rtc: 1 2 3 4 5

Current RTC date/time is 27-1-2020, 18:14:10.
Alarm time now set to 18:14:15.
Waiting 5 seconds for alarm... okay. Alarm rang.

Periodic IRQ rate is 64Hz.
Counting 20 interrupts at:
2Hz:     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
4Hz:     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
8Hz:     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
16Hz:    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
32Hz:    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
64Hz:    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

                 User Ram Write Test Values:
 0x6B 0x62 0xAF 0xAE 0x89 0xF4 0x71 0xDC 0x04 0x5A 0x84 0x75 0xBD 0xEF 0x5F 0x35 0xA0 0x1C 0x77 0xC9 0x9F 0x60 0x9A 0x2F 0x9A 0x99 0x3D 0x
20 0xBC 0xC1 0x91 0x27 0x23 0x41 0xD5 0xAD 0x35 0x46 0x89 0x39 0xA0 0x0E 0xAE 0x5D 0xFD 0x0E 0x93 0x9E 0x2A 0x0A 0x67 0xC9 0x6A 0x02 0xF8
0x05 0x9B 0x35 0x25 0x57 0xF7 0xB7 0x7E 0x1A

                 User Ram Read Test Values:
 0x6B 0x62 0xAF 0xAE 0x89 0xF4 0x71 0xDC 0x04 0x5A 0x84 0x75 0xBD 0xEF 0x5F 0x35 0xA0 0x1C 0x77 0xC9 0x9F 0x60 0x9A 0x2F 0x9A 0x99 0x3D 0x
20 0xBC 0xC1 0x91 0x27 0x23 0x41 0xD5 0xAD 0x35 0x46 0x89 0x39 0xA0 0x0E 0xAE 0x5D 0xFD 0x0E 0x93 0x9E 0x2A 0x0A 0x67 0xC9 0x6A 0x02 0xF8
0x05 0x9B 0x35 0x25 0x57 0xF7 0xB7 0x7E 0x1A

                User Ram Test PASSED


                 Trickle Charger Value From Device Tree
3k Ohm in series with a Schottky diode



                 Power Management Mode & Threshold
Reg Value = 0x01: Power Management Auto and Trickle Charger
Backup Battery Threshold = 2.2V



                         *** Test complete ***
```

*Figure 22. Command output of RTC test script.*

f. Install the necessary packages first to use the test program in Python. Install them with the following commands:

- sudo pip3 install ioctl-opt

```
pi@raspberrypi:~/Desktop $ sudo python3 rtc_python3Test.py
You need ioctl_opt!
                install it from https://pypi.org/project/ioctl-opt/
                or run pip install ioctl-opt
pi@raspberrypi:~/Desktop $ ^C
pi@raspberrypi:~/Desktop $ sudo pip3 install ioctl-opt
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting ioctl-opt
  Downloading https://www.piwheels.org/simple/ioctl-opt/ioctl_opt-1.2.2-py3-none-any.whl
Installing collected packages: ioctl-opt
Successfully installed ioctl-opt-1.2.2
```

*Figure 23. Installing dependencies.*

- sudo pip3 install pytz

```
pi@raspberrypi:~/Desktop $ sudo pip3 install pytz
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting pytz
  Downloading https://files.pythonhosted.org/packages/e7/f9/f0b53f88060247251bf481fa6ea62cd0d25bf1b11a87888e53ce5b7c8ad2/pytz-2019.3-py2.py3-none-any.whl (509kB)
    100% |████████████████████████████████| 512kB 429kB/s
Installing collected packages: pytz
Successfully installed pytz-2019.3
```

*Figure 24. Installing dependencies.*

After installing the necessary packages, run the program with the following command:

- sudo python3 rtc_python3Test.py "/dev/rtc0"

```
pi@raspberrypi:~/Desktop $ sudo python3 rtc_python3Test.py '/dev/rtc0'
                        RTC Driver Test Example.


***********IOCTL register read test***********
Address: 0x55 Value = 0xAA

IOCTL register write test
Address: 0x55 Value = 0xAA

IOCTL register write/read test SUCCESS!!!

***********IOCTL Data Retention Access Test***********

IOCTL Data Retention test SUCCESS!!!!

***********IOCTL External Clock Access Test***********

IOCTL External Clock test SUCCESS!!!!

***********IOCTL Power Management Access Test***********

IOCTL Power Management test SUCCESS!!!!

***********IOCTL Alarm2 Test***********

IOCTL Alarm2 set for one minute Min: 17

IOCTL Waiting alarm2

IOCTL  Alarm2 test SUCCESS!!!!


                Update IRQ Test Started


Counting 5 update (1/sec) interrupts from reading /dev/rtc0:
1 2 3 4 5

Current RTC date/time is  27-02-2020, 12:17:05.

Alarm time now set to  12:17:10.

Waiting 5 seconds for alarm...
Okay. Alarm rang.


Periodic IRQ rate is 64Hz.

Counting 20 interrupts at:

2Hz:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
4Hz:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
8Hz:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
16Hz:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
32Hz:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
64Hz:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

                 User Ram Write Test Values:

0x22 0x29 0x50 0x09 0x3F 0xB5 0x38 0x19 0x6C 0x2A 0x39 0xD3 0x4C 0xC5 0xA3 0x1A 0xFD 0x90 0x75 0x5E 0xA1 0x95 0x34 0xA0 0xD2 0x93 0xE8 0x7D 0x93 0x37
AB 0x69 0x24 0x7B 0x13 0x19 0x98 0xA6 0x23 0x4E 0xD6 0x71 0x31 0x63 0x72 0xFF 0xB5

                 User Ram Read Test Values:

0x22 0x29 0x50 0x09 0x3F 0xB5 0x38 0x19 0x6C 0x2A 0x39 0xD3 0x4C 0xC5 0xA3 0x1A 0xFD 0x90 0x75 0x5E 0xA1 0x95 0x34 0xA0 0xD2 0x93 0xE8 0x7D 0x93 0x37
AB 0x69 0x24 0x7B 0x13 0x19 0x98 0xA6 0x23 0x4E 0xD6 0x71 0x31 0x63 0x72 0xFF 0xB5

                User Ram Test PASSED



         Trickle Charger Value From Device Tree
         3k Ohm in series with a Schottky diode



         Power Management Mode & Threshold
         Reg Value = 0x01: Power Management Auto and Trickle Charger



                         *** Test complete ***

pi@raspberrypi:~/Desktop $
```

*Figure 25. Command output of RTC Python test script.*

## 4.2 Kernel Compilation Procedure

Another way to compile and run the driver is to compile it with the whole kernel. Follow these instructions to build the kernel in the computer: (Ubuntu is used here to build the kernel. Use **this link** for another system.)

First, download and install toolchain with the following commands:

- git clone https://github.com/raspberrypi/tools ~/tools
- echo PATH=\$PATH:~/tools/arm-bcm2708/arm-linux-gnueabihf/bin >> ~/.bashrc
- source ~/.bashrc


*Figure 26. Downloading the required tools for compiling the Raspberry Pi kernel.*

Install an additional set of libraries for a 32-bit operating system (for example, Raspberry Pi Desktop for the PC):

- sudo apt install zlib1g-dev:amd64

Download the kernel source after these steps:

- git clone --depth=1 https://github.com/raspberrypi/linux

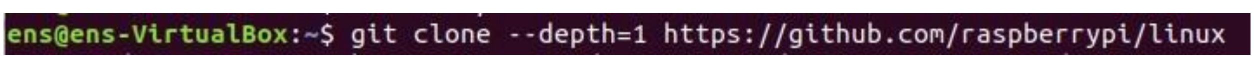



*Figure 27. Downloading the Linux kernel for Raspberry Pi.*

Ensure the needed dependencies are there on the machine to build the sources for cross-compilation. Execute the following command:

- sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev

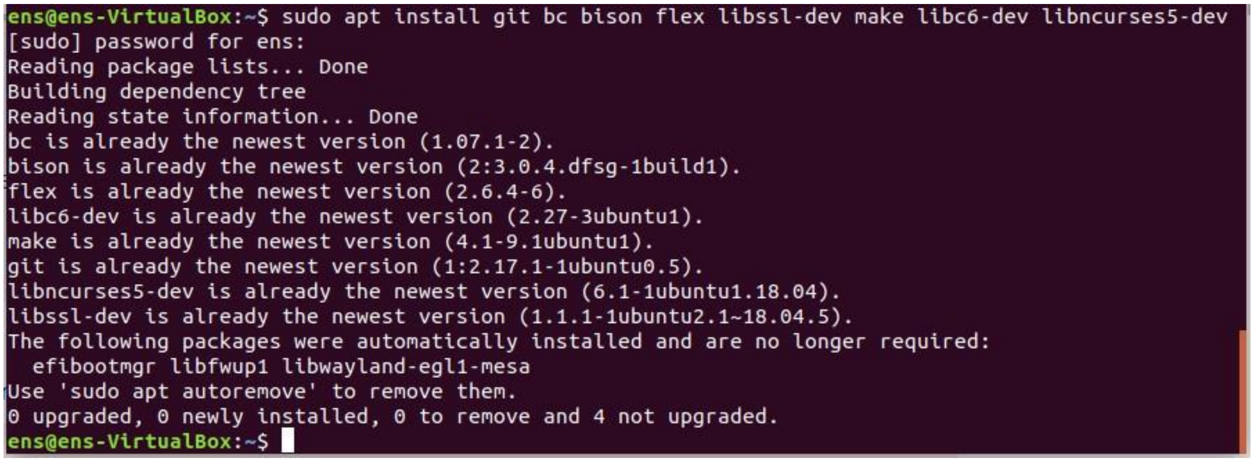



*Figure 28. Installing the system dependencies for kernel compiling.*

Copy rtc-MAX31343.c in ./linux/drivers/rtc after these steps.

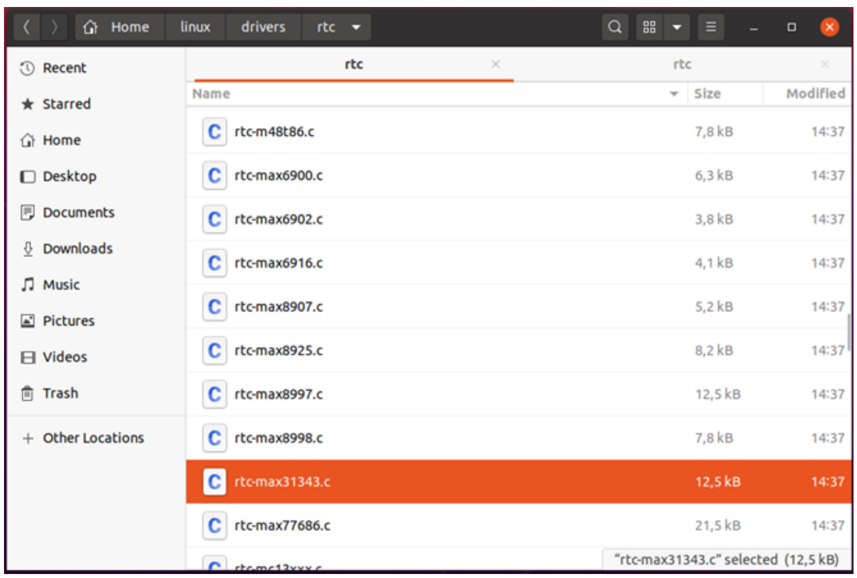



*Figure 29. Device driver file located in kernel.*

Add the following lines to the drivers/rtc/Kconfig file:

config RTC_DRV_MAX31343

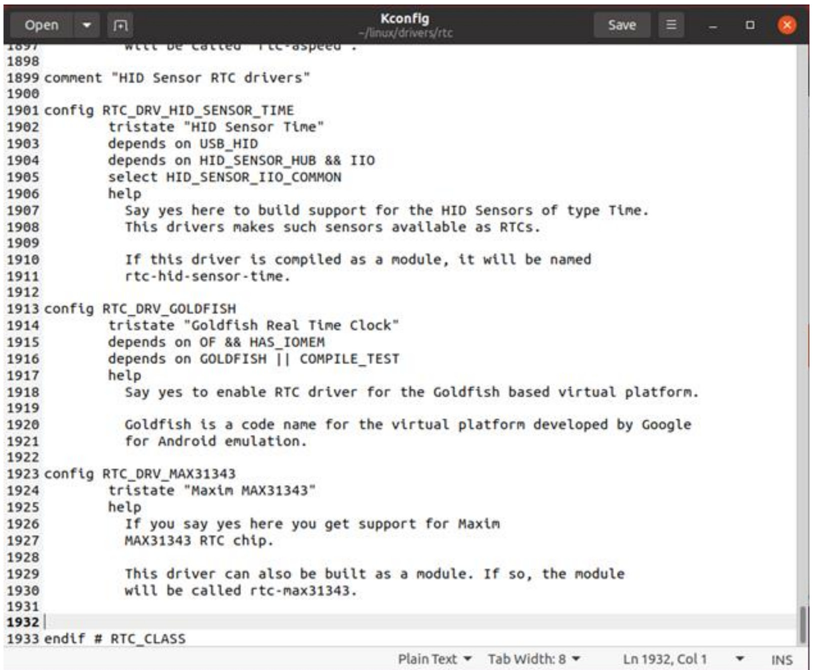
tristate “Maxim MAX31343”

help

If you say yes here you get support for Maxim

MAX31343 RTC chip.

This driver can also be built as a module. If so, the module is called rtc-max31343.




```
comment "HID Sensor RTC drivers"

config RTC_DRV_HID_SENSOR_TIME
        tristate "HID Sensor Time"
        depends on USB_HID
        depends on HID_SENSOR_HUB && IIO
        select HID_SENSOR_IIO_COMMON
        help
          Say yes here to build support for the HID Sensors of type Time.
          This drivers makes such sensors available as RTCs.

          If this driver is compiled as a module, it will be named
          rtc-hid-sensor-time.

config RTC_DRV_GOLDFISH
        tristate "Goldfish Real Time Clock"
        depends on OF && HAS_IOMEM
        depends on GOLDFISH || COMPILE_TEST
        help
          Say yes to enable RTC driver for the Goldfish based virtual platform.

          Goldfish is a code name for the virtual platform developed by Google
          for Android emulation.

config RTC_DRV_MAX31343
        tristate "Maxim MAX31343"
        help
          If you say yes here you get support for Maxim
          MAX31343 RTC chip.

          This driver can also be built as a module. If so, the module
          will be called rtc-max31343.


endif # RTC_CLASS
```

*Figure 30. Kconfig file after modifications.*

Add the following line to the drivers/rtc/Makefile:

obj-$(CONFIG_RTC_DRV_MAX31343) += rtc-max31343.o

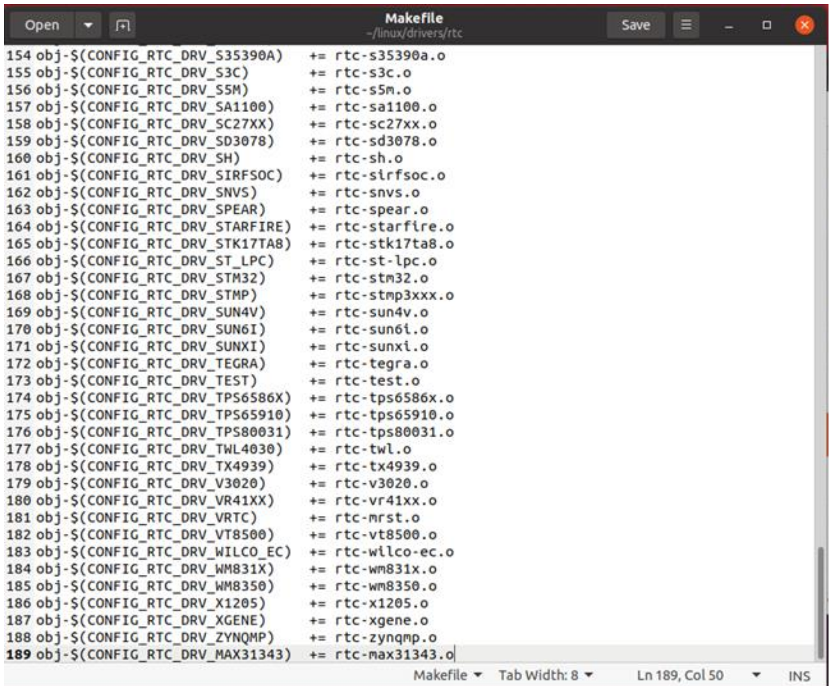



*Figure 31. Adding the device driver to Makefile of RTC drivers.*

Ensure the Maxim MAX31343 option is selected within the kernel configuration before building the kernel. Use menuconfig to configure the kernel. (Ubuntu is used here to build the kernel for Raspberry Pi 3B+. Use **this link** for another system.)

- KERNEL=kernel7
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig

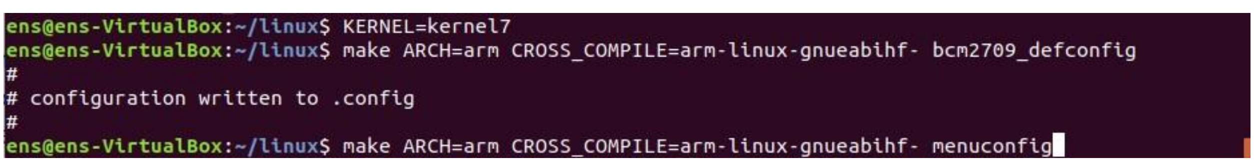



*Figure 32. Configuring the kernel before compiling.*

Follow these figures:

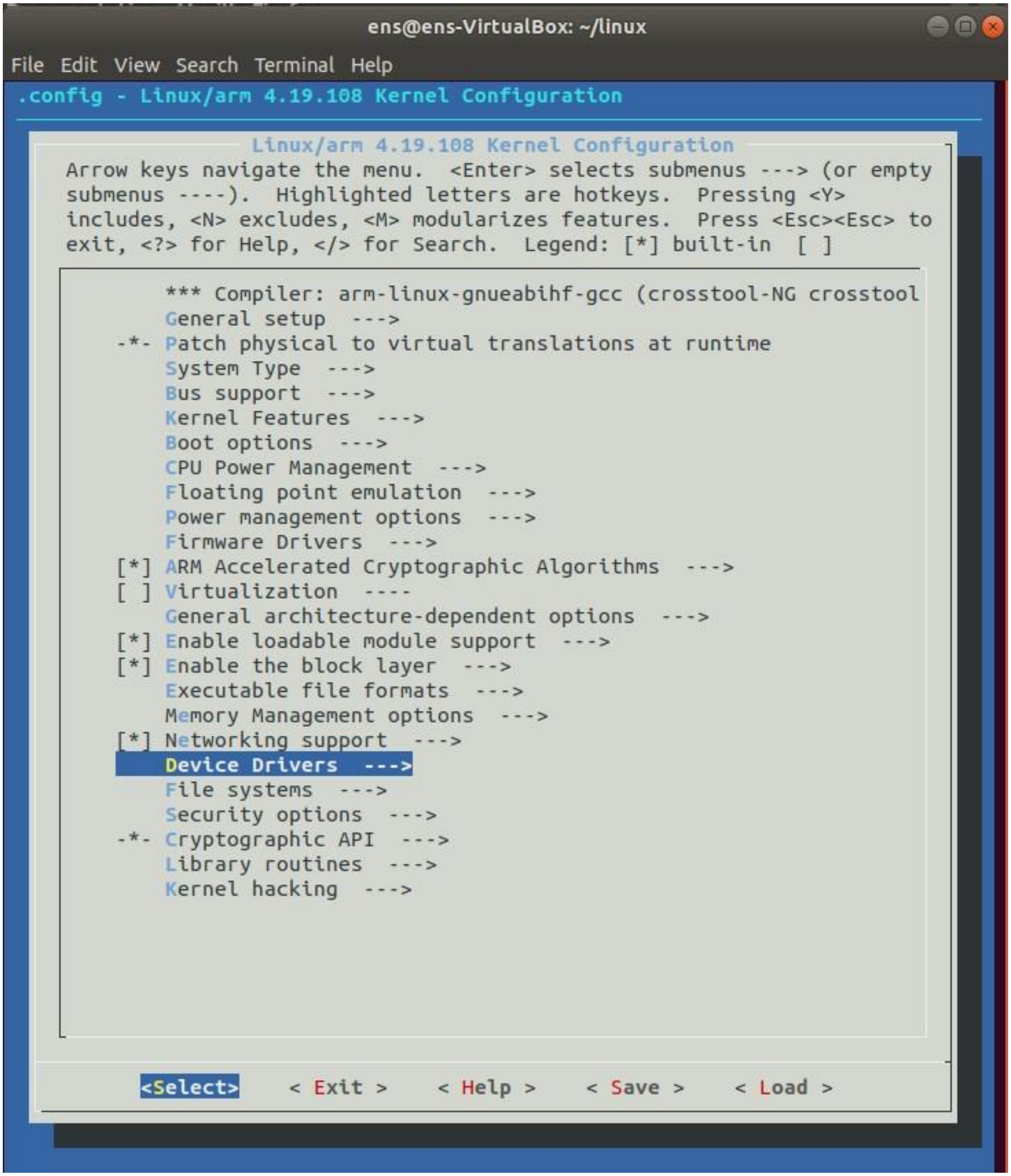



*Figure 33. Selecting Device Drivers menu.*

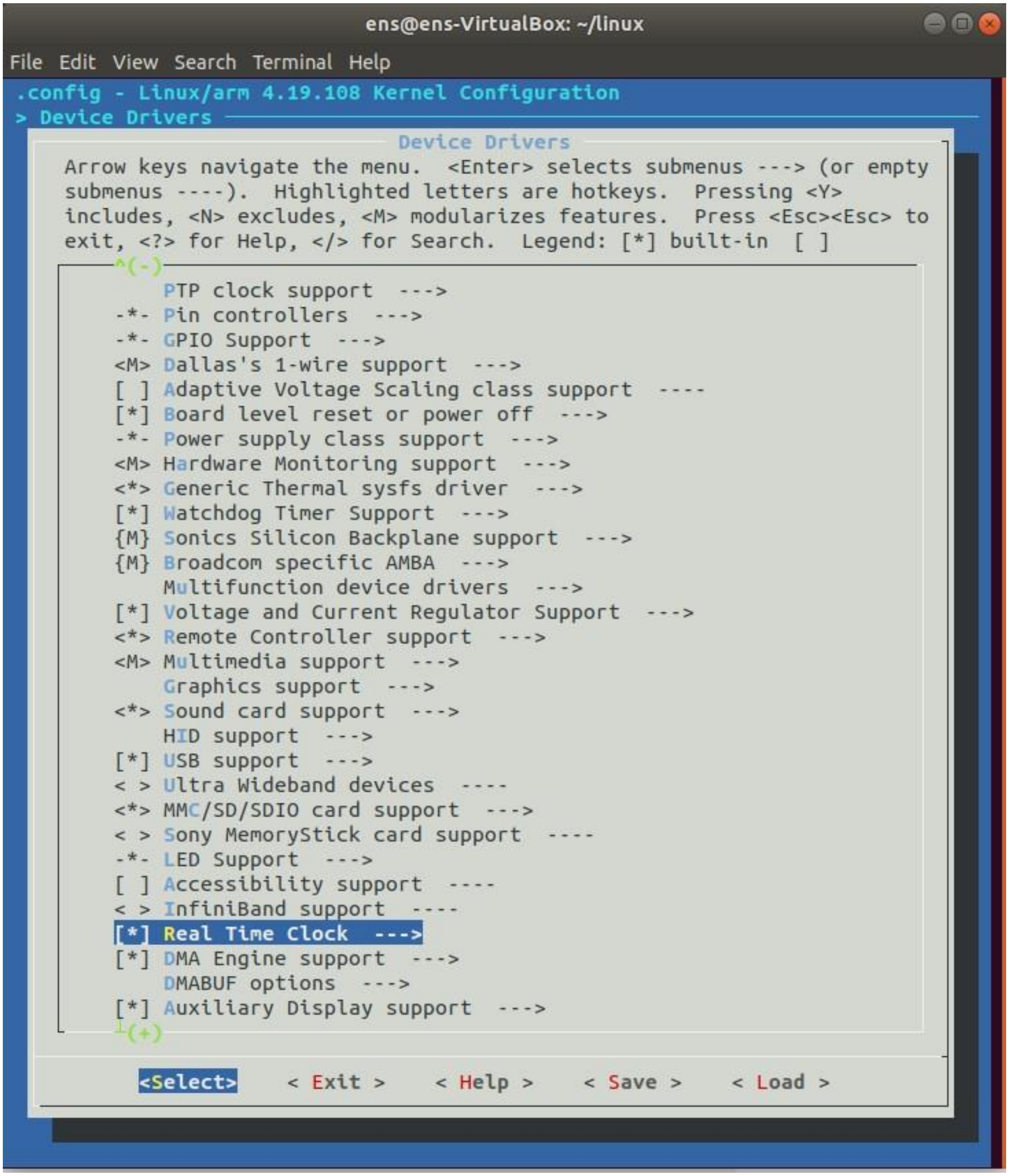



*Figure 34. Selecting Real Time Clock menu.*

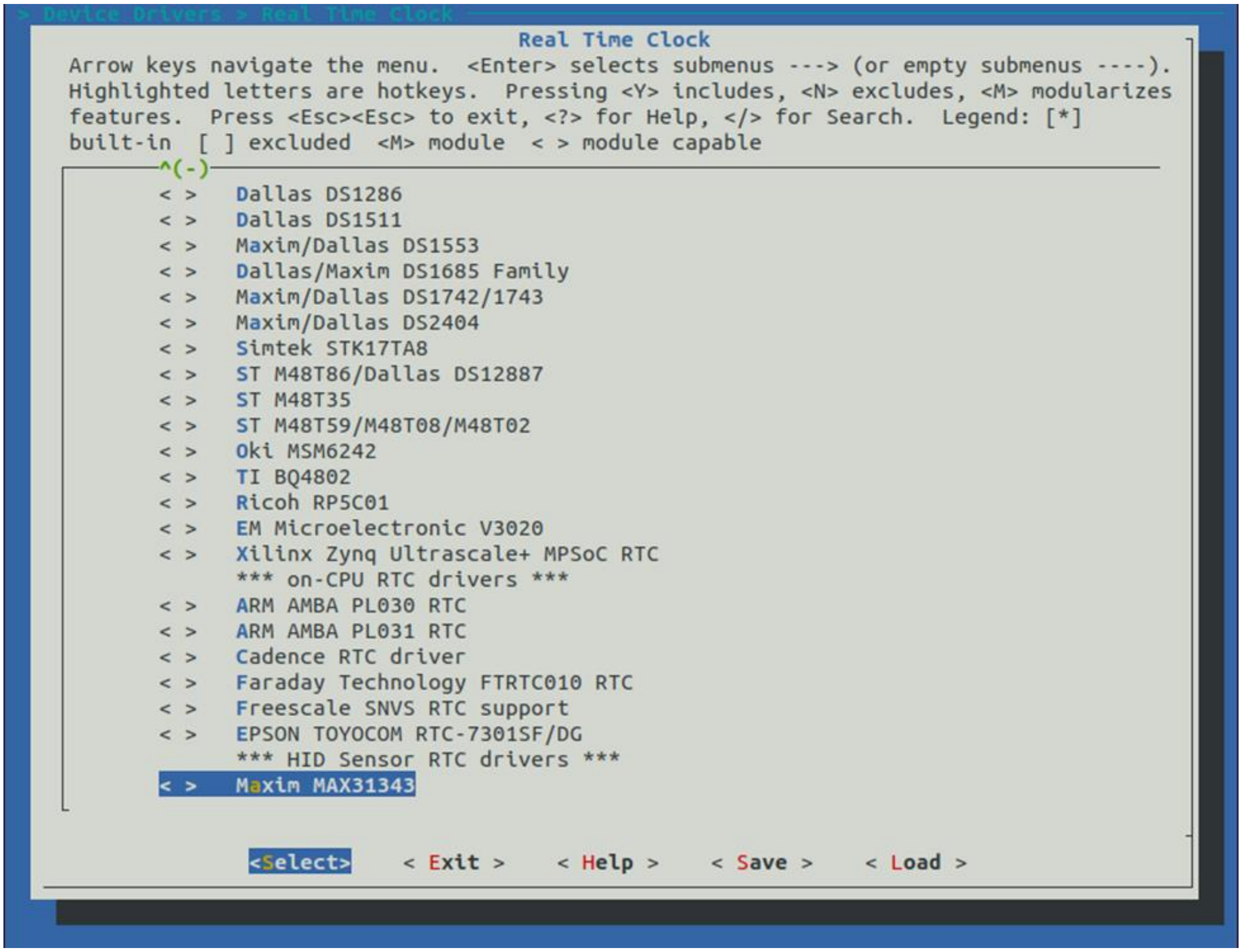



*Figure 35. Enabling the Maxim MAX31343 device driver for compilation.*

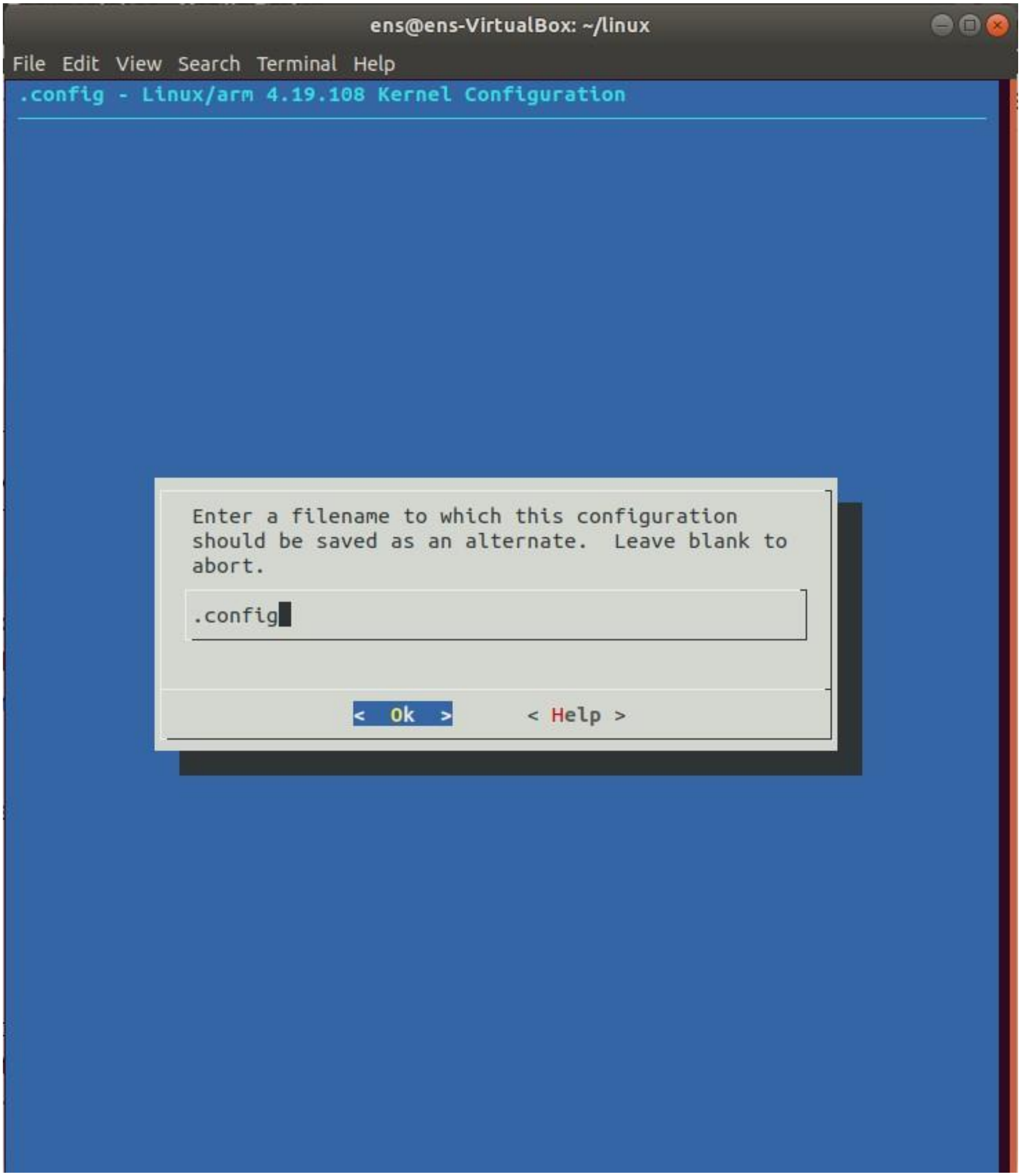



*Figure 36. Saving the configuration file.*

After the steps mentioned above, compile the kernel with the following commands:

- cd linux
- KERNEL=kernel7
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
- make ARCH

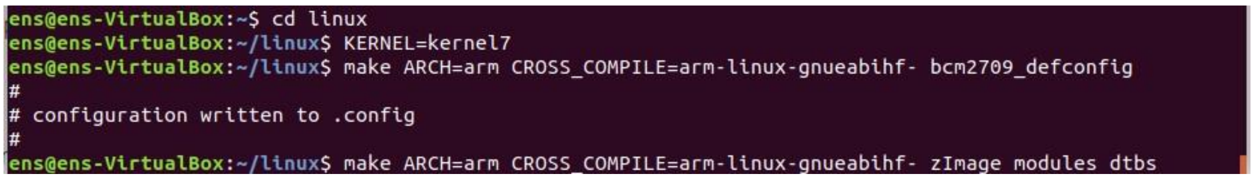


*Figure 37. Compiling the kernel*

## References


- https://sysplay.github.io/books/LinuxDrivers/book/
- http://robbie-cao.github.io/2016/09/device-tree
- https://www.jameco.com/Jameco/workshop/circuitnotes/raspberry-pi-circuit-note.html
- https://stackoverflow.com/questions/40529308/linux-driver-ioctl-or-sysfs
- https://www.raspberrypi.org/documentation/linux/kernel/building.md
- https://www.man7.org/linux/man-pages/man4/rtc.4.html
- https://www.kernel.org/doc/html/latest/admin-guide/rtc.html#new-portable-rtc-class-drivers-dev-rtcn

## Revision History

| REVISION NUMBER | REVISION DATE | DESCRIPTION | PAGES CHANGED |
|---|---|---|---|
| 0 | 06/21 | Initial release | — |
| 1 | 12/21 | Updated the link for the rtc-max31343.c, removal of the i2c-rtc-overlay.dts link | 11,12 |
| | | | |
| | | | |
| | | | |