RTC Device Driver
User Guide

Table of Contents

List of Figures

List of Tables

1. Linux Device Driver

This document explains Linux Device Driver concepts and specifically 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 released to kernel.org.
    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. More details follow in 4.4.
2. Download driver from Maxim website.
    a. Add the driver into the downloaded kernel. Then compile with the whole kernel. More details follow in 4.4.

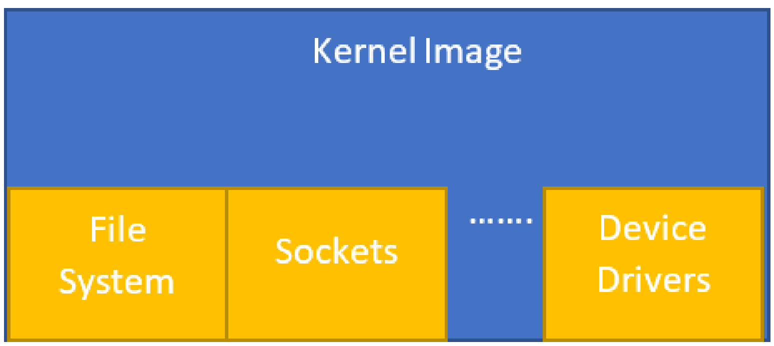


Figure 1. Kernel image w/out modules.

    b. Linux has a proper way to inject a driver into the running Linux distribution or Linux Kernel Module (LKM). LKMs are not part of the main kernel. They are injected by users into the running Kernel. The user can determine which LKM module driver to include for kernel compilation. This helps to reduce the size of the kernel. For example, the ethernet LKM module 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

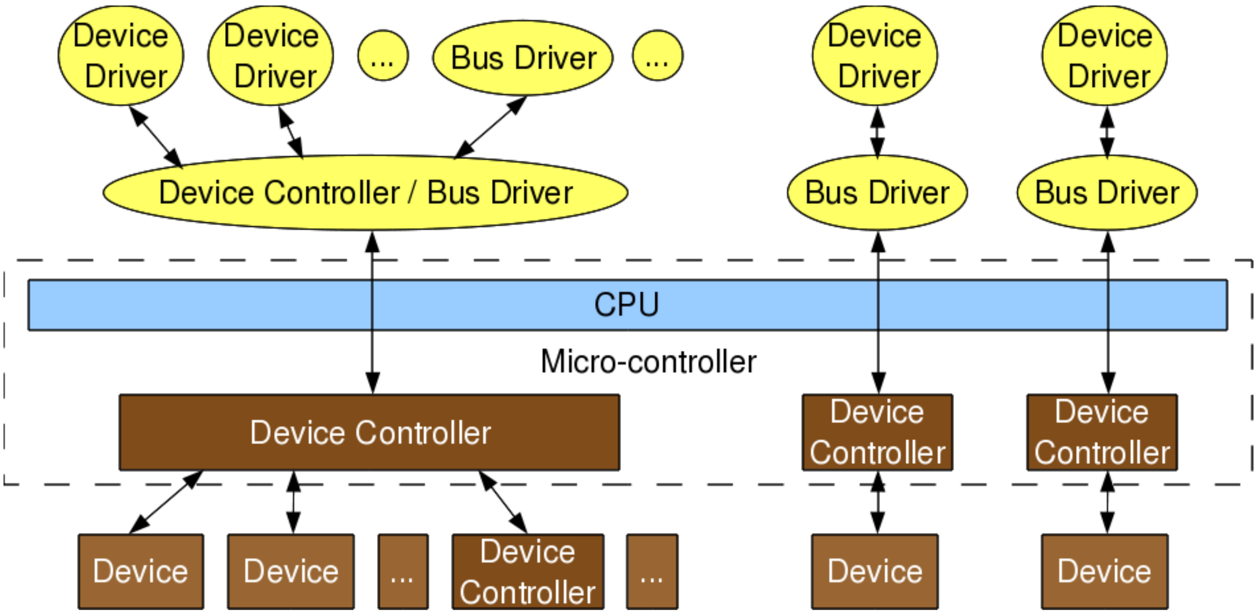


Figure 2. Linux device management [1].

Kernel developers generate Linux's generic device driver. The device driver must support all the required feature 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 APIs. However, the device driver must have parser and default values for custom parameters like trickle charger settings or power-management mode.

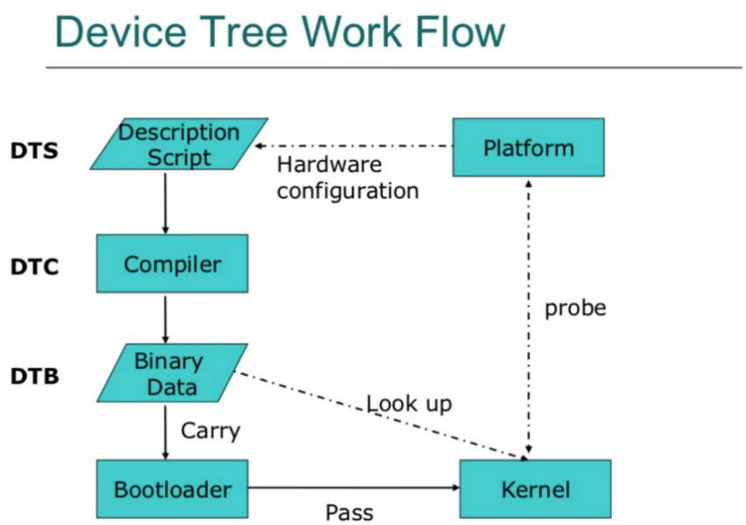


Figure 3. Device tree workflow [2].

2.2. Device Tree Compilation

The device tree has its own compiler to generate output. More details can be found on **this link**.

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.

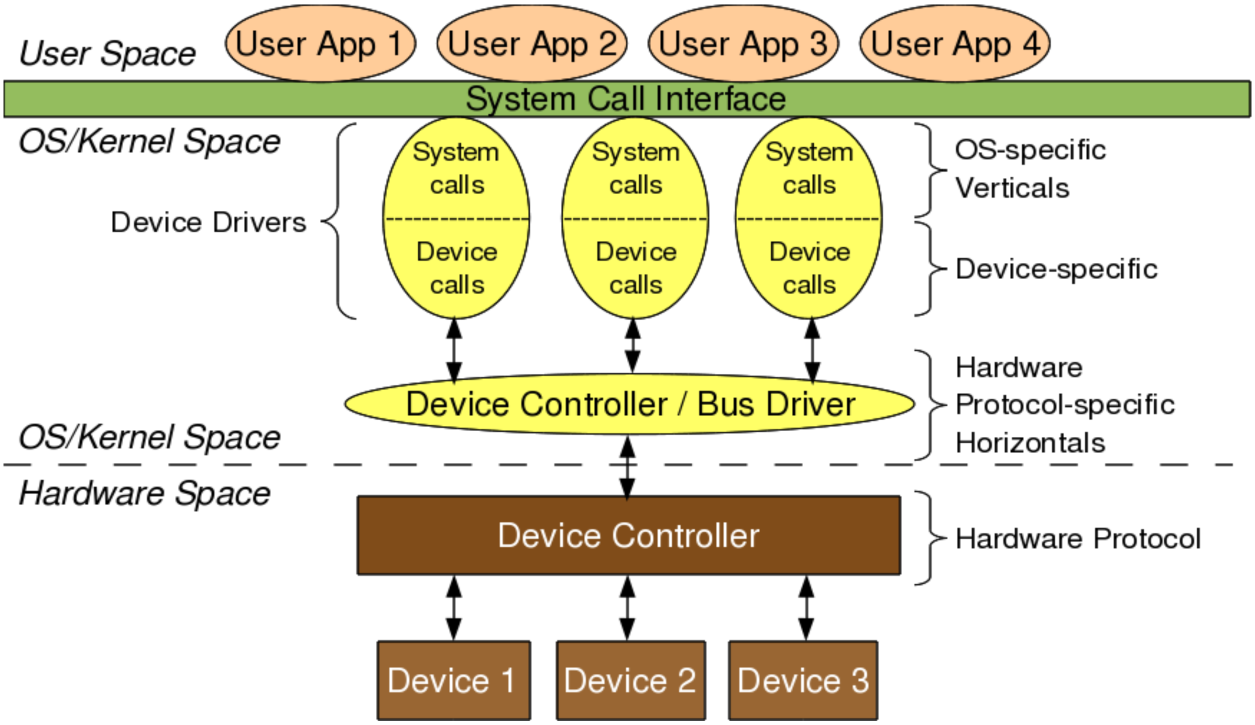


Figure 4. System calls to device drivers [1].

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 if 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 above.

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 via 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 MAX31341B.

4. MAX31341B Device Driver Package

4.1. Supported Features

4.1.1. Kernel Version

Table 2. MAX31341B Kernel Version's Features.

STANDARD LINUX RTC CONFIGURATION OPTIONS	**STATUS**	**NOTES**
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-max31341b.c

4.1.2. Maxim Version

The Maxim Website version has a couple of extra features (Table 3) unlike the kernel version of the driver. The main difference between them is the IOCTL option to demonstrate the IC features. Also, an external interrupt pin is supported by the driver. The external interrupt pin must be defined in the device tree to make the device driver use INTA and INTB simultaneously. More details follow in section 4.2.

Table 3. MAX31341B Maxim Version's Features.

MAXIM DRIVER CUSTOM IOCTL OPTIONS	DESCRIPTION	KERNEL VERSION IMPLEMENTATION
Power Management	Selects the current power mode of the IC. Changes default power-management mode (mode 1). The possible values are: • 0 for comparator mode. • 1 for power management auto and trickle charger is on. • 2 for power management manual and trickle charger is on, V_{CC} is active supply. • 3 for power management manual and trickle charger is on, AIN is active supply, if AIN > V_{CC}.	The module is accessible via sysfs.
Alarm 2	Supported.	Not implemented. Kernel-generic RTC driver does not have this option.
Register Read/Write Access	Supported.	Not implemented. A driver's flow must not be interrupted by external access within the Linux kernel. The register access is not implemented as it can break flow.
External Clock	Supported.	Not implemented. Kernel-generic RTC driver does not have this option.
Data Retention	Supported.	Not implemented. Kernel-generic RTC driver does not have this option.

4.2. Device-Specific Device Tree Parameters

There are two types of parameters for MAX31341B drivers.

1. Kernel parameters.
    a. Main interrupt line
    b. I^2C bus identifier
    c. I^2C device address

2. Driver parameters.

   a. Trickle resistor ohm value

   e.g. Trickle-resistor-ohms=MAX31341B_TC_3K_OHM_SINGLE_DIODE;

   Options for trickle resistor:

   MAX31341B_TC_NO_CONNECT
   MAX31341B_TC_3K_OHM_SINGLE_DIODE
   MAX31341B_TC_6K_OHM_SINGLE_DIODE
   MAX31341B_TC_11K_OHM_SINGLE_DIODE
   MAX31341B_TC_3K_OHM_DOUBLE_DIODE
   MAX31341B_TC_6K_OHM_DOUBLE_DIODE
   MAX31341B_TC_11K_OHM_DOUBLE_DIODE

   b. Backup threshold voltage

   e.g. backup-threshold=MAX31341B_BACKUP_THRES_2_2V;

   Options for backup threshold voltage:

   MAX31341B_BACKUP_THRES_1_3V

   MAX31341B_BACKUP_THRES_1_7V

   MAX31341B_BACKUP_THRES_2_0V

   MAX31341B_BACKUP_THRES_2_2V

   c. Power-management mode

   e.g. power-management=<0>

   Options for power-management setting:

   0 - comparator mode

   1 - power management auto and trickle charger is on

   2 - power management manual and trickle charger is on, V_{CC} is active supply

   3 - power management manual and trickle charger is on, AIN is active supply, if AIN > V_{CC}

   d. Trickle diode on/off selection

   e.g. trickle-diode-enable=”yes”;

   e. INTB pin’s complement pin on board (Maxim version only)

   e.g.

   fragment@12{

   target = <&gpio>;

```
__overlay__{
        rtc_intpins: rtc_intpins {
            brcm, pins = <20>;
            brcm, function = <0>; /* in */
            brcm, pull = <2>; /* up */
        };
    };
};
```

The device-specific parameters must be documented for kernel.org. An example is uploaded to the mainline kernel. Here is the documentation/example for kernel.org.

rtc-max31341b.txt

4.3. Testing the Maxim Driver on Raspberry Pi 3B+

A project example was prepared for RPi 3 B+. The i^{2}c-rtc overlay file must be updated to comprise the MAX31341B driver. The driver device tree file overlays on the Raspberry pi kernel device tree.

Table 4. MAX31341B Pin Configuration.

MAX31341B	**RPi 3 B+**
V_{CC}	Pin1 - 3V3
GND	Pin6 - Ground
SDA	Pin3 - GPIO2
SCL	Pin5 - GPIO3
INTA	Pin36 - GPIO16
INTB	Pin38 - GPIO20

Here is the kernel.org version of the device tree overlay for Raspberry Pi.

i2c-rtc-overlay.dts

a. The driver must be compiled from the source code for portability. The Raspberry Pi kernel must be updated to the latest version before that.

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

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

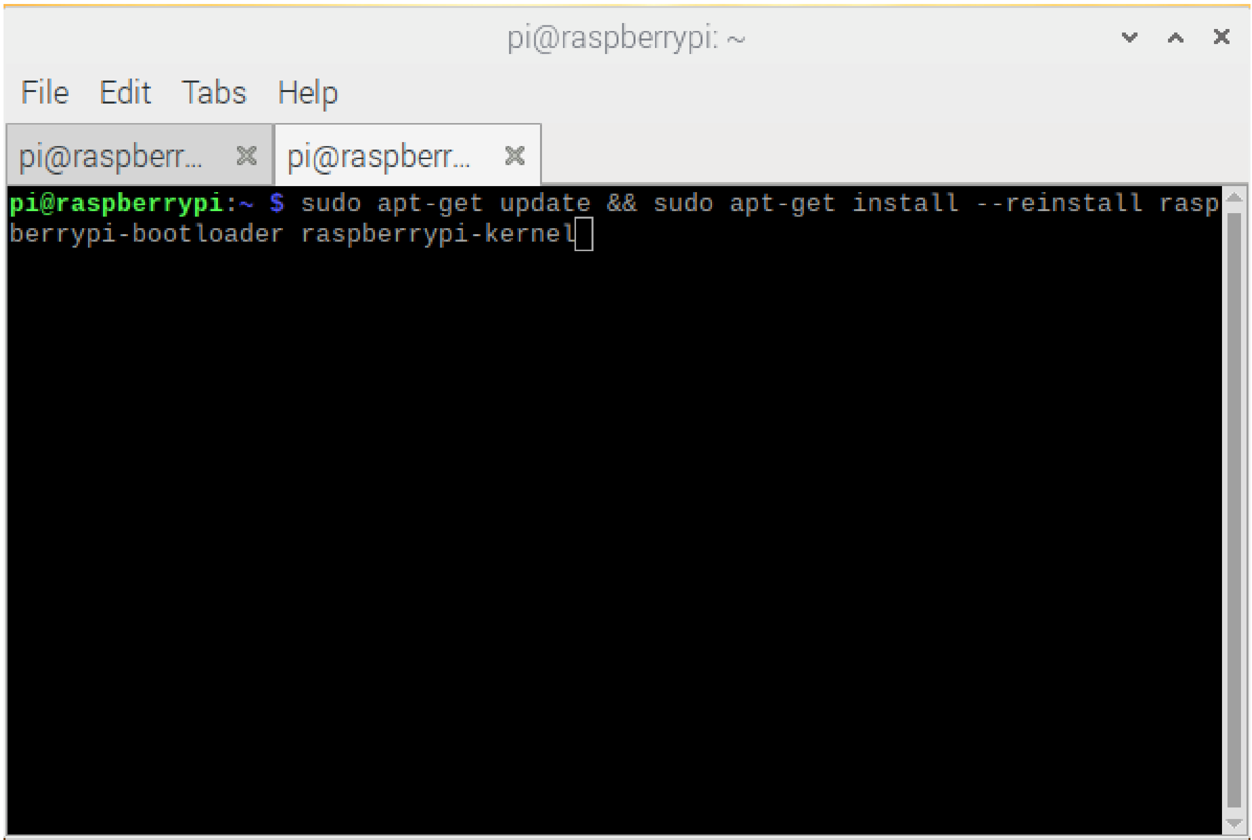


Figure 6

b. Kernel headers must be installed to compile the MAX31341B 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^{2}c automatically. So, enable the i^{2}c 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,max31341b

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.

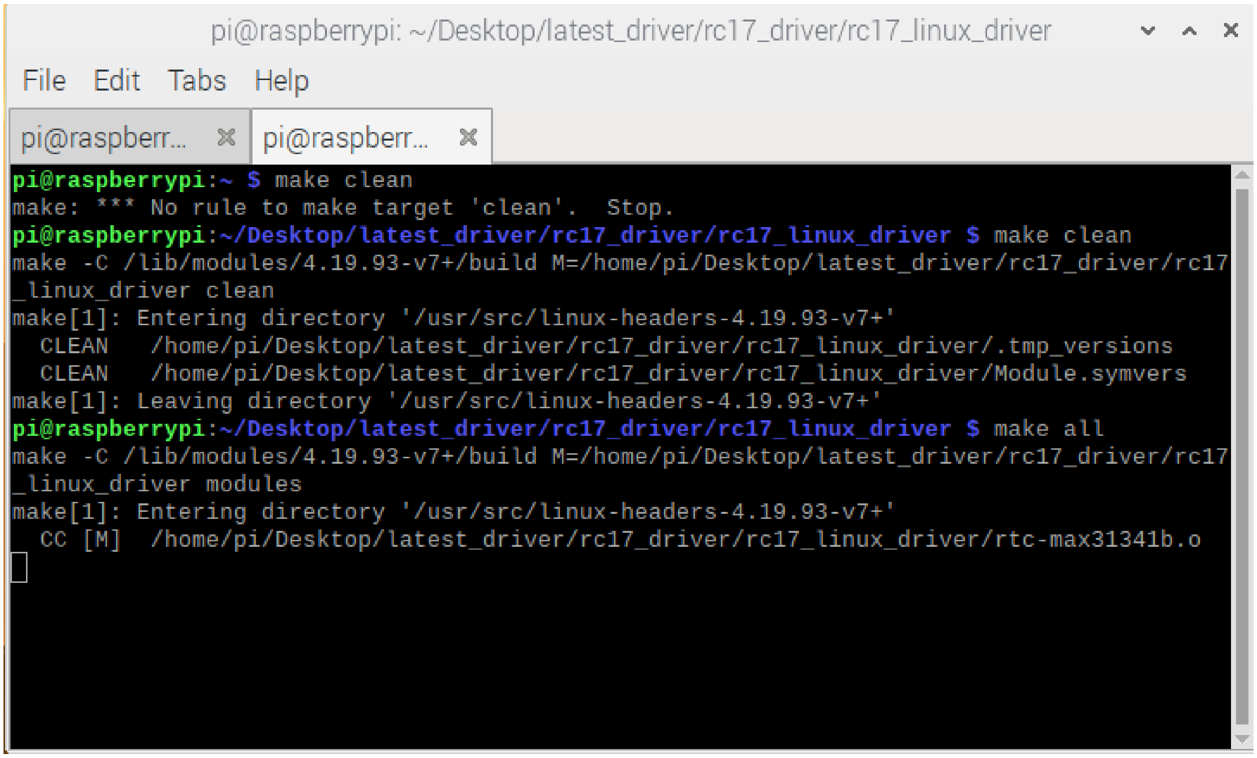


Figure 7

- The **sudo make install** command installs the driver into the LKM folder. Most system folders are not accessible to users. Use the sudo extension for the commands for access. This extension provides privileged access.

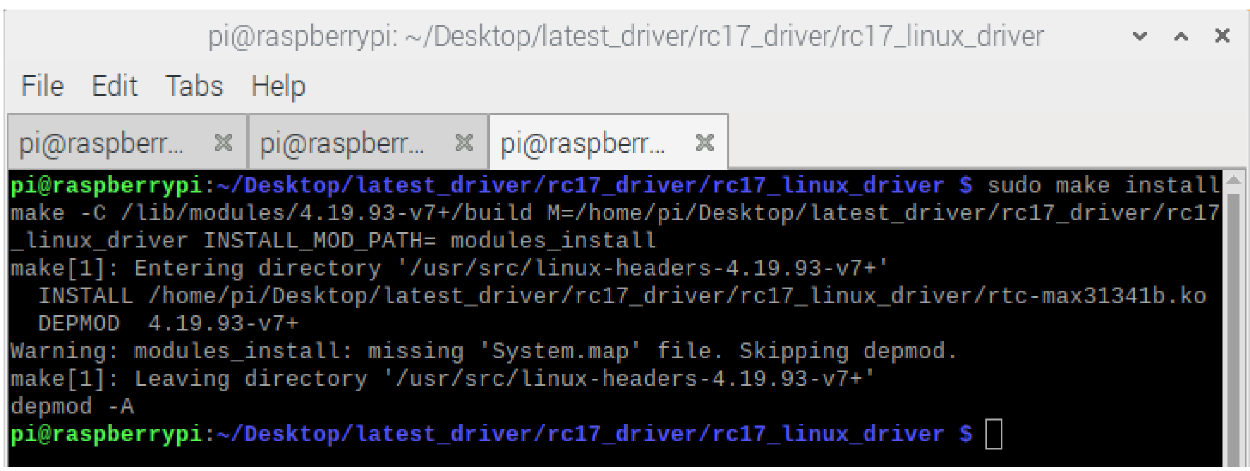


Figure 8

- The **make dtbs** command compiles the device tree overlay (i^{2}c-rtc-overlay.dts) 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/latest_driver/rc17_driver/rc17_linux_driver $ make dtbs
cpp -nostdinc -I include -I arch  -undef -x assembler-with-cpp i2c-rtc-overlay.dts i2c-r
tc-overlay.dts.preprocessed
dtc -I dts -O dtb -q -o i2c-rtc.dtbo i2c-rtc-overlay.dts.preprocessed
rm i2c-rtc-overlay.dts.preprocessed
sudo cp i2c-rtc.dtbo /boot/overlays
```

Figure 9

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

4.1.3. Basic Tests

The **timedatectl** and **hwclock** commands can be used for basic tests such as setting/reading time. These commands are provided by the Raspbian 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.

```
pi@raspberrypi: ~/Desktop/latest_driver/rc17_driver/rc17_linux_driver
File  Edit  Tabs  Help
pi@raspberr...  pi@raspberr...  pi@raspberr...  pi@raspberr...
pi@raspberrypi:~/Desktop/latest_driver/rc17_driver/rc17_linux_driver $ 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/latest_driver/rc17_driver/rc17_linux_driver $ sudo timedatectl set-
ntp no
pi@raspberrypi:~/Desktop/latest_driver/rc17_driver/rc17_linux_driver $ 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/latest_driver/rc17_driver/rc17_linux_driver $
```

Figure 10

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/latest_driver/rc17_driver/rc17_linux_driver $ sudo hwclock --set --
date "1/27/2020 14:50:00"
```

Figure 11

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

```
pi@raspberrypi:~/Desktop/latest_driver/rc17_driver/rc17_linux_driver $ 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

```
pi@raspberrypi:~/Desktop/latest_driver/rc17_driver/rc17_linux_driver $ sudo hwclock -r
2020-01-27 11:47:05.120164+00:00
```

Figure 13

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

```
pi@raspberrypi:~/Desktop/latest_driver/rc17_driver/rc17_linux_driver $ sudo hwclock --hctosy
s
pi@raspberrypi:~/Desktop/latest_driver/rc17_driver/rc17_linux_driver $ 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

4.3.2. 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. Change these settings using 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

b. The **cat trickle_charger** command reads the trickle charger settings. 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

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

- sudo -s

Enter the above command to change the user mode.

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

Figure 17

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.

```
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

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

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

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

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 above command 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

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

- 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

Run the program with the following command after installing them:

- 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

4.4. 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.

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

Figure 26

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

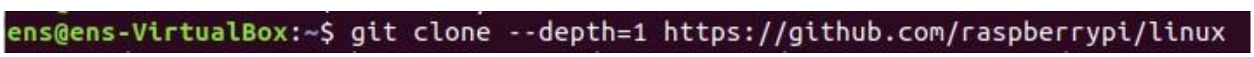
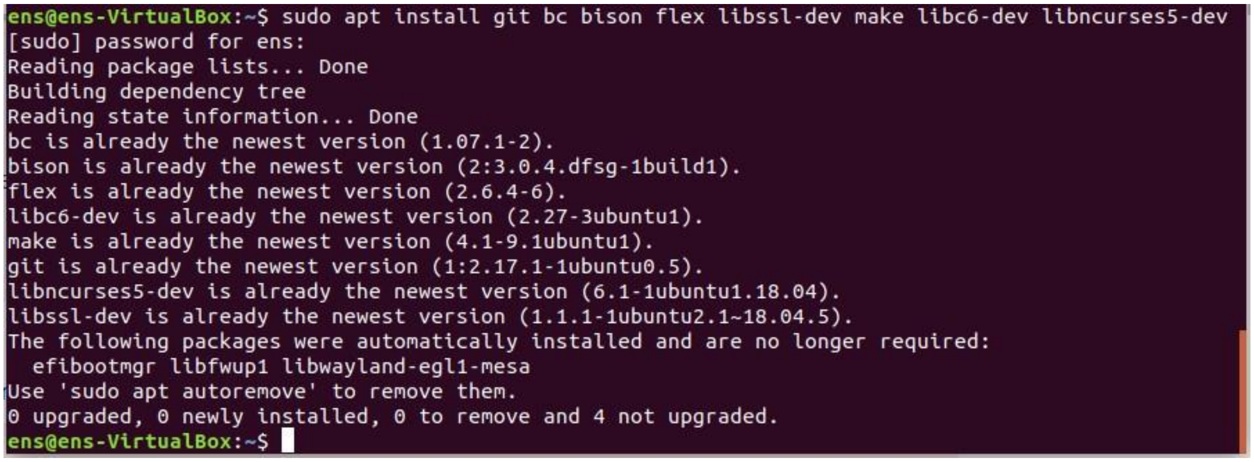


Figure 27

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



```
ens@ens-VirtualBox:~$ sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev
[sudo] password for ens:
Reading package lists... Done
Building dependency tree
Reading state information... Done
bc is already the newest version (1.07.1-2).
bison is already the newest version (2:3.0.4.dfsg-1build1).
flex is already the newest version (2.6.4-6).
libc6-dev is already the newest version (2.27-3ubuntu1).
make is already the newest version (4.1-9.1ubuntu1).
git is already the newest version (1:2.17.1-1ubuntu0.5).
libncurses5-dev is already the newest version (6.1-1ubuntu1.18.04).
libssl-dev is already the newest version (1.1.1-1ubuntu2.1~18.04.5).
The following packages were automatically installed and are no longer required:
  efibootmgr libfwup1 libwayland-egl1-mesa
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
ens@ens-VirtualBox:~$
```

Figure 28

Copy rtc-max31341b.c in ./linux/drivers/rtc after these steps.

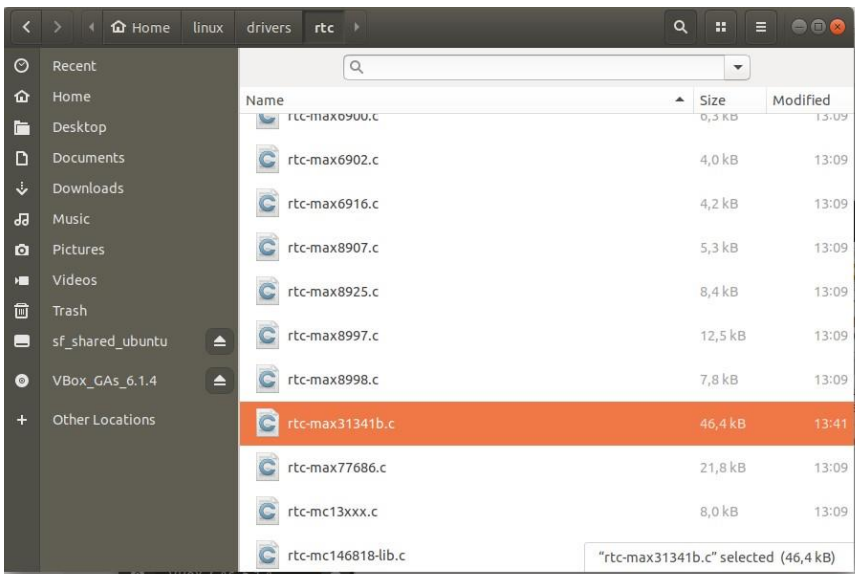


Figure 29

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

config RTC_DRV_MAX31341B

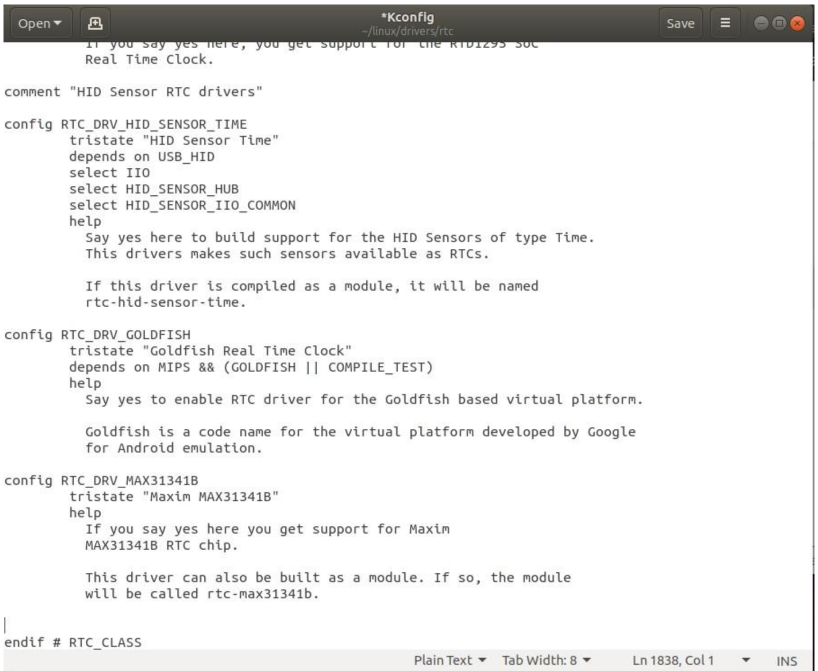
tristate “Maxim MAX31341B”

help

If you say yes here you get support for Maxim

MAX31341B RTC chip.

This driver can also be built as a module. If so, the module is called rtc-max31341b.




```
          Real Time Clock.

comment "HID Sensor RTC drivers"

config RTC_DRV_HID_SENSOR_TIME
        tristate "HID Sensor Time"
        depends on USB_HID
        select IIO
        select HID_SENSOR_HUB
        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 MIPS && (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_MAX31341B
        tristate "Maxim MAX31341B"
        help
          If you say yes here you get support for Maxim
          MAX31341B RTC chip.

          This driver can also be built as a module. If so, the module
          will be called rtc-max31341b.


endif # RTC_CLASS
```


Figure 30

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

obj-$(CONFIG_RTC_DRV_MAX31341B) += rtc-max31341b.o

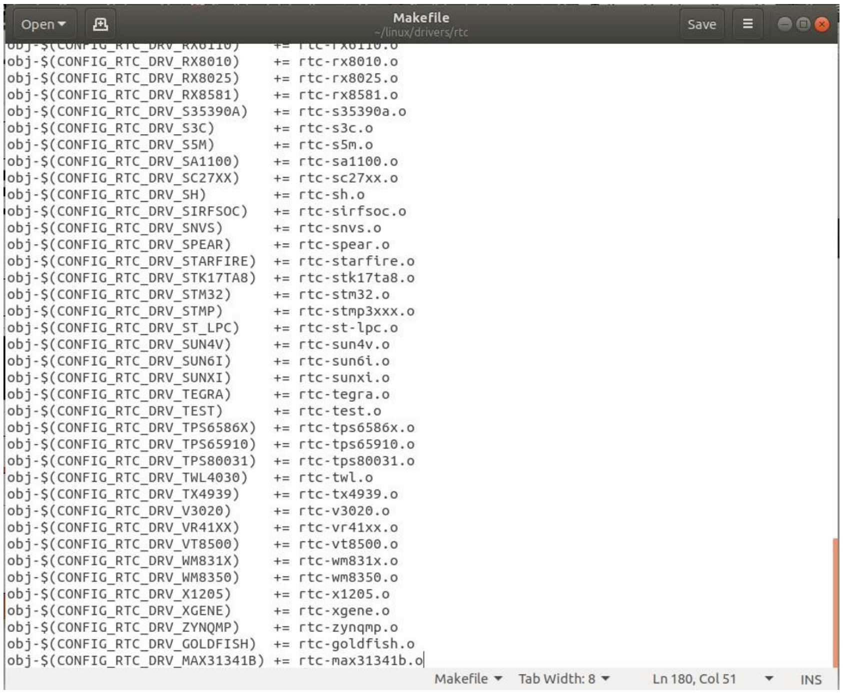


Figure 31

Ensure the Maxim MAX31341B 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

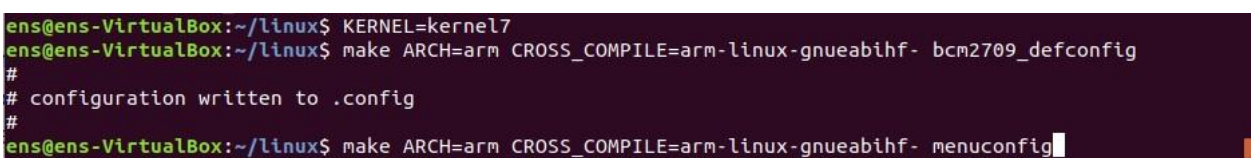


Figure 32

Follow these figures:

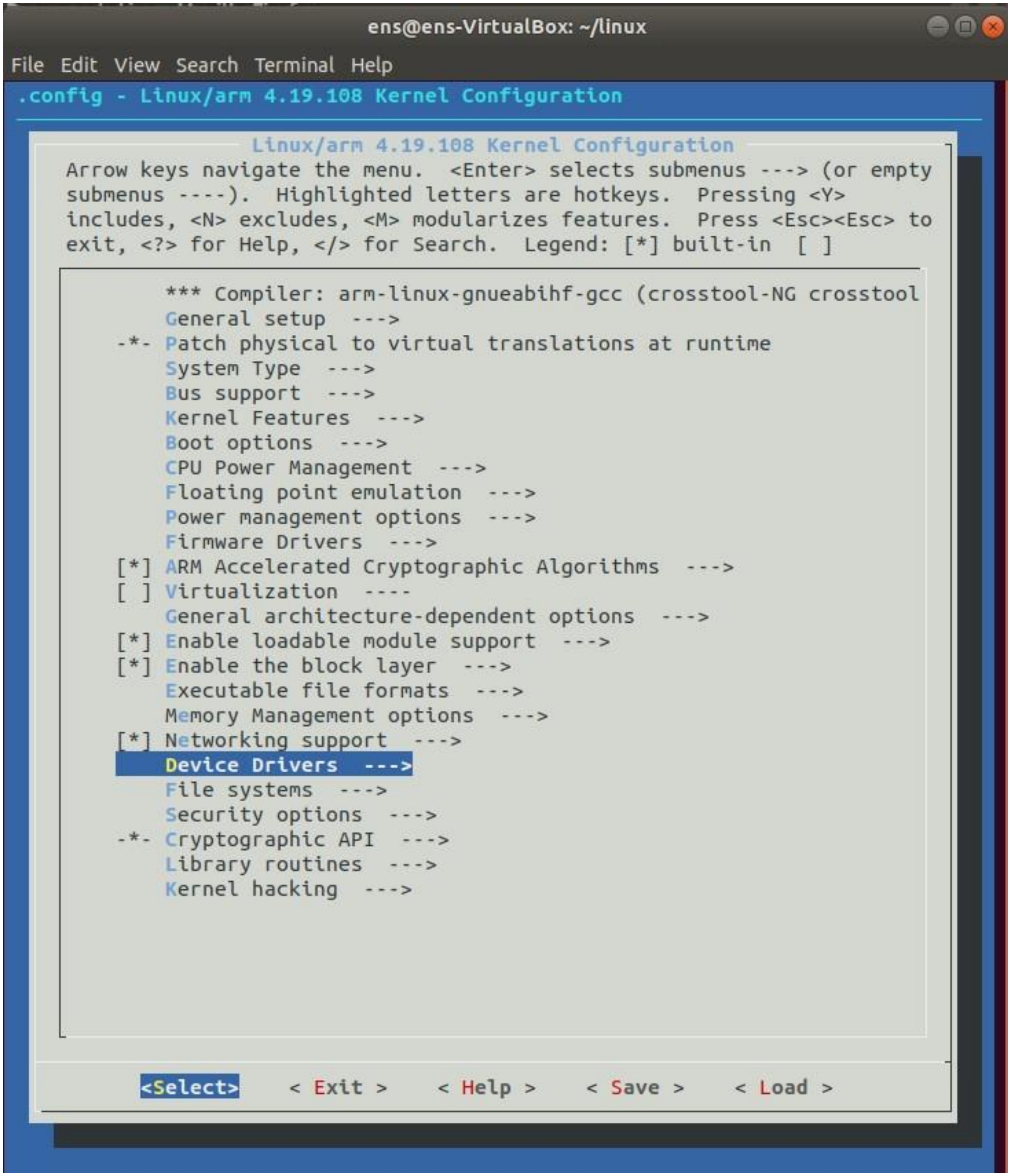
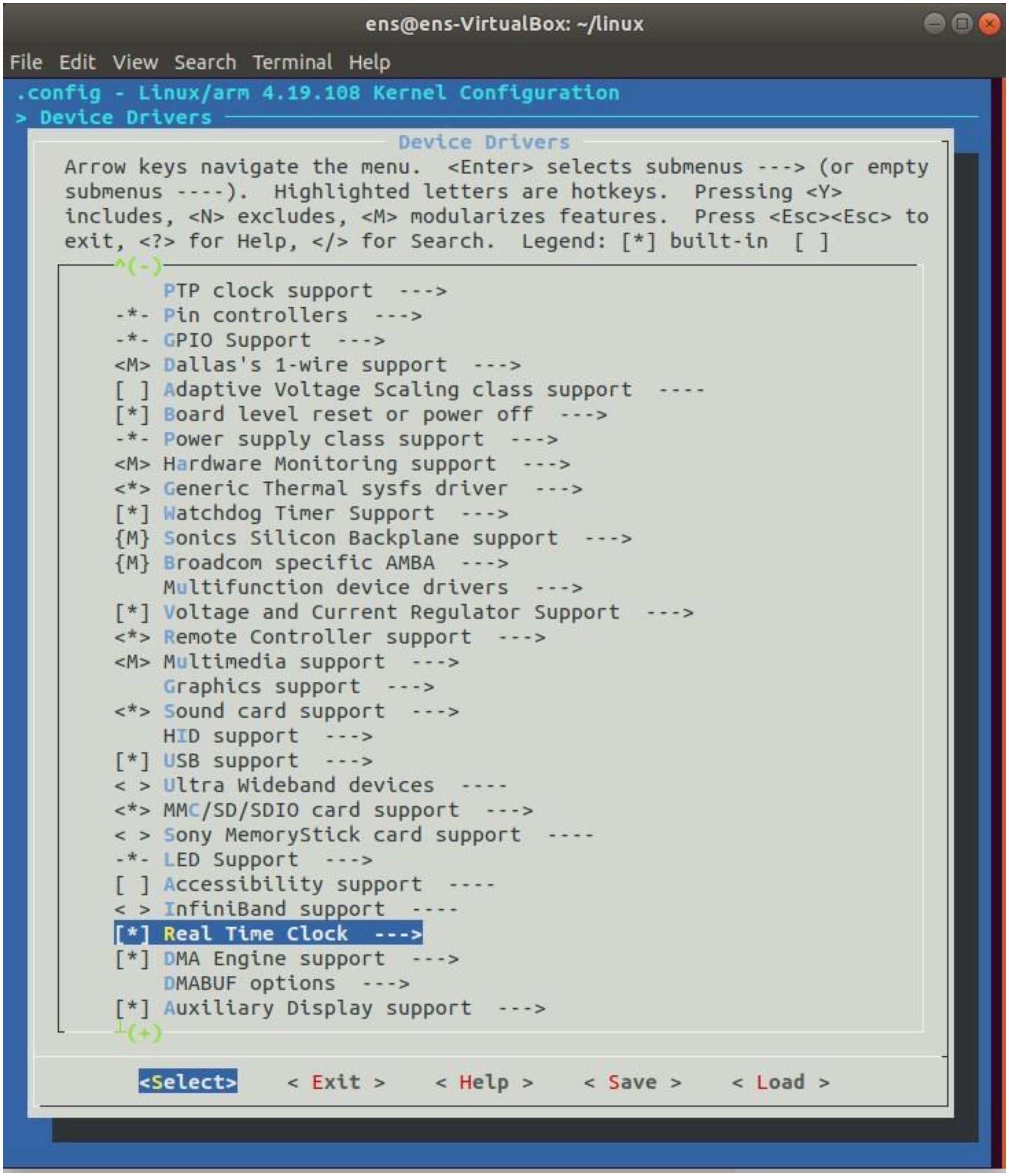


Figure 33


ens@ens-VirtualBox: ~/linux
File Edit View Search Terminal Help
.config - Linux/arm 4.19.108 Kernel Configuration
> Device Drivers
Device Drivers
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
^(-)
PTP clock support --->
-*- Pin controllers --->
-*- GPIO Support --->
<M> Dallas's 1-wire support --->
[ ] Adaptive Voltage Scaling class support ----
[*] Board level reset or power off --->
-*- Power supply class support --->
<M> Hardware Monitoring support --->
<*> Generic Thermal sysfs driver --->
[*] Watchdog Timer Support --->
{M} Sonics Silicon Backplane support --->
{M} Broadcom specific AMBA --->
Multifunction device drivers --->
[*] Voltage and Current Regulator Support --->
<*> Remote Controller support --->
<M> Multimedia support --->
Graphics support --->
<*> Sound card support --->
HID support --->
[*] USB support --->
< > Ultra Wideband devices ----
<*> MMC/SD/SDIO card support --->
< > Sony MemoryStick card support ----
-*- LED Support --->
[ ] Accessibility support ----
< > InfiniBand support ----
[*] Real Time Clock --->
[*] DMA Engine support --->
DMABUF options --->
[*] Auxiliary Display support --->
(+)
<Select> < Exit > < Help > < Save > < Load >
Figure 34

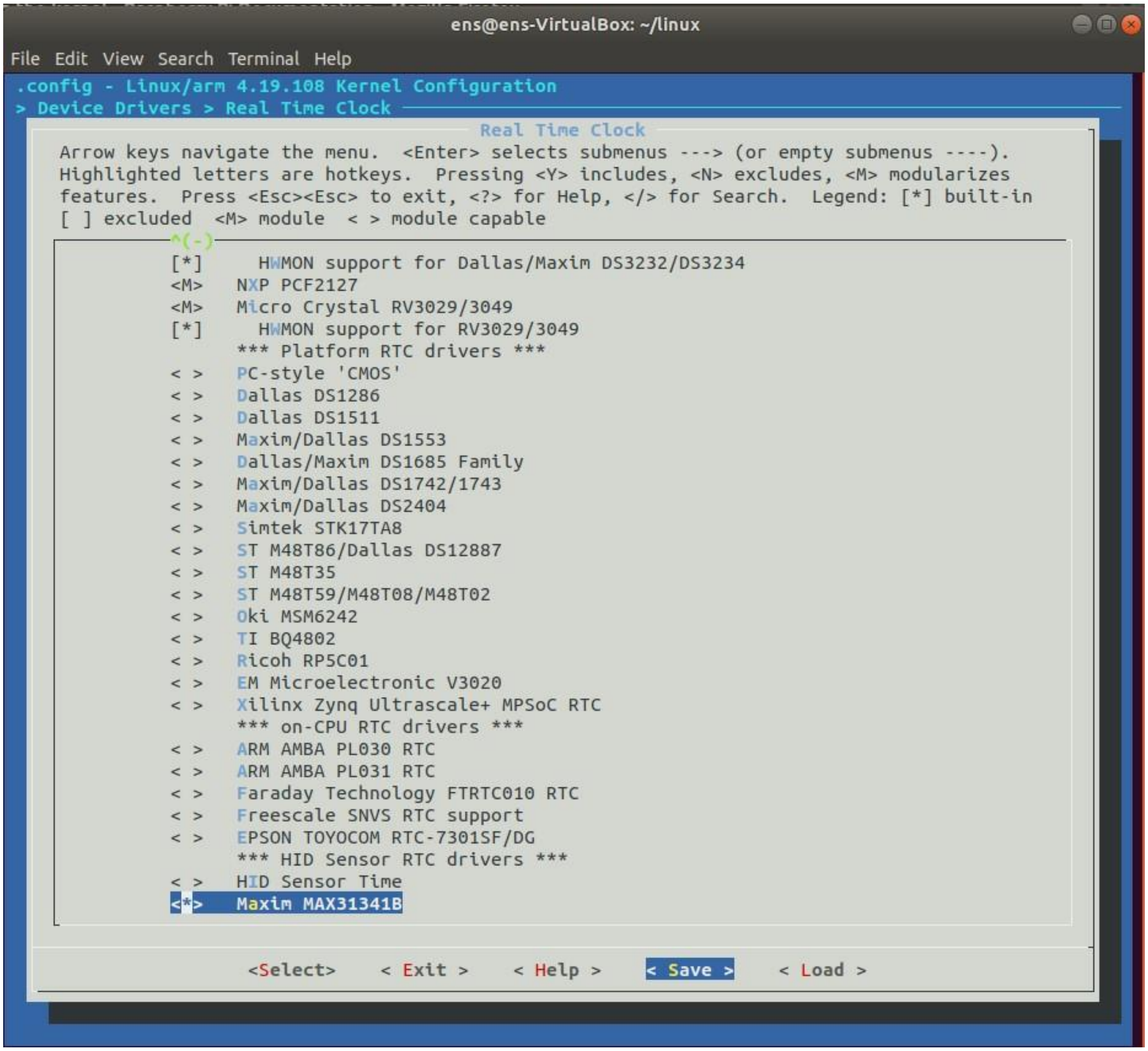
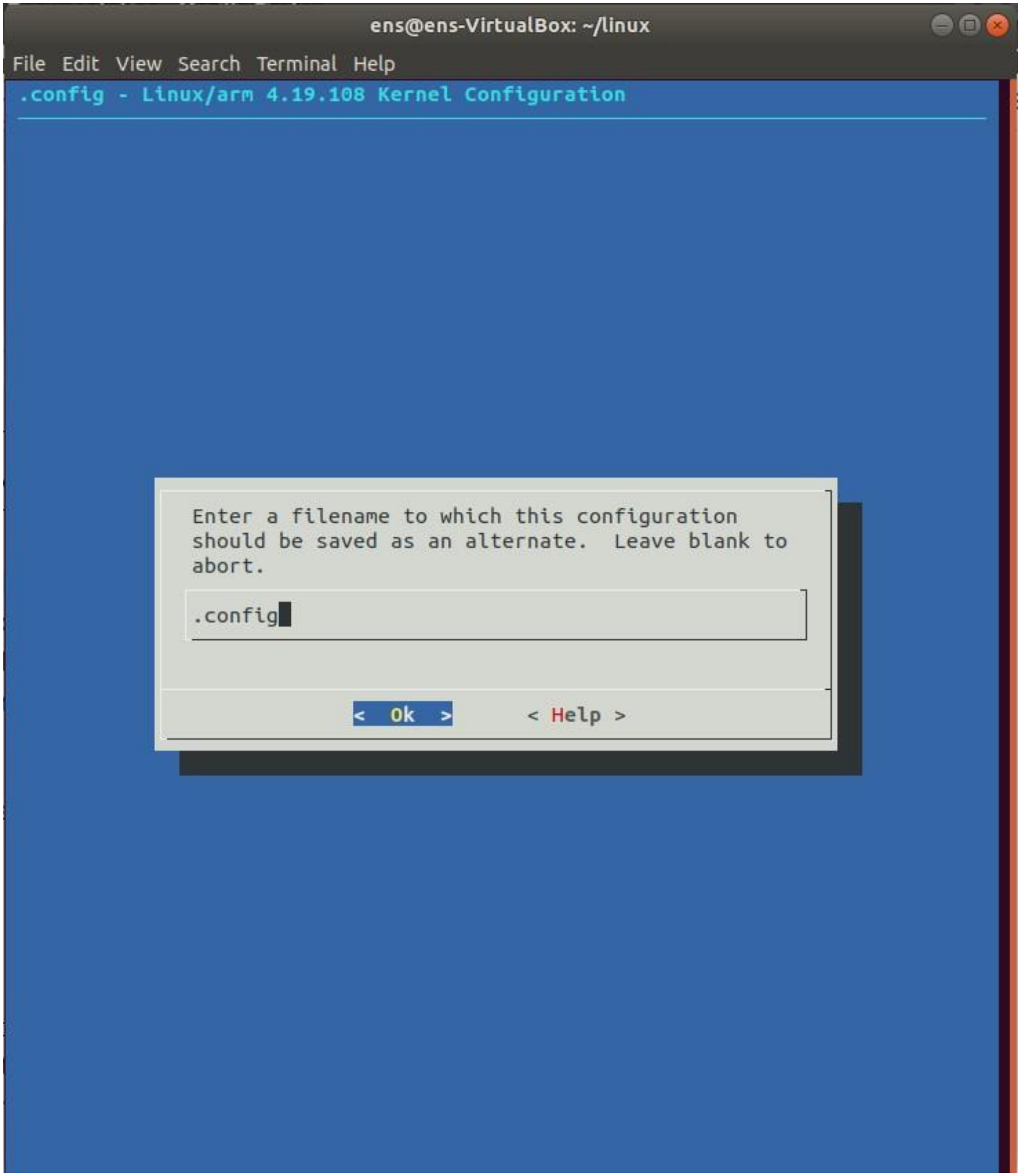

ens@ens-VirtualBox: ~/linux
File Edit View Search Terminal Help
.config - Linux/arm 4.19.108 Kernel Configuration
> Device Drivers > Real Time Clock
Real Time Clock
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in
[ ] excluded <M> module < > module capable
^(-)
[*] HWMON support for Dallas/Maxim DS3232/DS3234
<M> NXP PCF2127
<M> Micro Crystal RV3029/3049
[*] HWMON support for RV3029/3049
*** Platform RTC drivers ***
< > PC-style 'CMOS'
< > Dallas DS1286
< > Dallas DS1511
< > Maxim/Dallas DS1553
< > Dallas/Maxim DS1685 Family
< > Maxim/Dallas DS1742/1743
< > Maxim/Dallas DS2404
< > Simtek STK17TA8
< > ST M48T86/Dallas DS12887
< > ST M48T35
< > ST M48T59/M48T08/M48T02
< > Oki MSM6242
< > TI BQ4802
< > Ricoh RP5C01
< > EM Microelectronic V3020
< > Xilinx Zynq Ultrascale+ MPSoC RTC
*** on-CPU RTC drivers ***
< > ARM AMBA PL030 RTC
< > ARM AMBA PL031 RTC
< > Faraday Technology FTRTC010 RTC
< > Freescale SNVS RTC support
< > EPSON TOYOCOM RTC-7301SF/DG
*** HID Sensor RTC drivers ***
< > HID Sensor Time
<*> Maxim MAX31341B
<Select> < Exit > < Help > < Save > < Load >

Figure 35


ens@ens-VirtualBox: ~/linux
File Edit View Search Terminal Help
.config - Linux/arm 4.19.108 Kernel Configuration
Enter a filename to which this configuration should be saved as an alternate. Leave blank to abort.
.config
< Ok >
< Help >
Figure 36

Compile the kernel with these commands after all these steps:

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

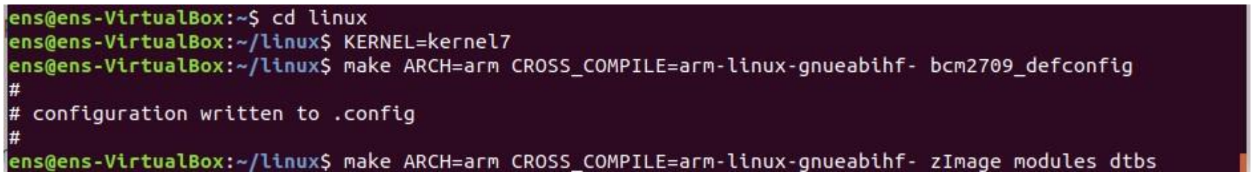


Figure 37

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
1	06/20	Initial release	—