



RTC Device Driver User Guide

UGxxxx; Rev 2.0; 09/01

Table of Contents

1	Linux Device Driver.....	5
1.1	What is A Device Driver?	5
1.2	Ways to Implement a Driver as Kernel Module	5
1.3	Device Driver Structure.....	6
2	Device Tree Parameters	7
2.1	RTC Device Trees	7
2.2	Device Tree Compilation.....	7
3	Linux Generic RTC Device Drivers.....	8
3.1	RTC Specific Default Features.....	8
3.2	Device Specific Custom Features	9
4	MAX31341B Device Driver Package.....	11
4.1	Supported Features	11
4.1.1	Kernel version.....	11
4.1.2	Maxim Version	11
4.2	Device Specific Device Tree Parameters	12
4.3	Testing Maxim Driver on Raspberry Pi 3B+	14
4.3.1	Basic Tests.....	17
4.3.2	SysFs Entry Test of Maxim Version Driver	19
4.4	Kernel Compilation Procedure	24
	References	33
	Revision History	34

List of Figures

Figure 1	Kernel Image W/ out Modules	5
Figure 2	Linux Device Management [1].....	6
Figure 3	Device Tree Work Flow [2].....	7
Figure 4	System calls to Device Drivers [1]	8
Figure 5	IOCTL parameters for MAX31341B	10
Figure 6	15
Figure 7	16

Figure 8.....	16
Figure 9.....	17
Figure 10.....	18
Figure 11.....	18
Figure 12.....	18
Figure 13.....	18
Figure 14.....	18
Figure 15.....	19
Figure 16.....	19
Figure 17.....	19
Figure 18.....	20
Figure 19.....	20
Figure 20.....	20
Figure 21.....	20
Figure 22.....	21
Figure 23.....	22
Figure 24.....	22
Figure 25.....	23
Figure 26.....	24
Figure 27.....	24
Figure 28.....	25
Figure 29.....	25
Figure 30.....	26
Figure 31.....	27
Figure 32.....	27
Figure 33.....	28
Figure 34.....	29
Figure 35.....	30
Figure 36.....	31
Figure 37.....	32

List of Tables

Table 1 Default RTC Features 9

Table 2 MAX31341B Kernel Version’s Features..... 11

Table 3 MAX31341B Maxim Version’s Features 12

Table 4 MAX31341B Pin Configuration 15

Table 5 MAX31342 Features 32

Table 6 MAX31342 Pin Configuration..... 32

Table 7 MAX31343 Features 32

Table 8 MAX31343 Pin Configuration..... 32

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 Linux Kernel to make user space commands independent from system hardware design, so that, end user doesn't need to have 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

Maxim RTC driver is implemented in two different ways.

1. Driver released to kernel.org.
 - a. By doing this, The driver is included in the Kernel that comes with 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 on 4.4.
2. Download driver from Maxim website
 - a. Customer can add the driver into downloaded Kernel then compile with whole Kernel. More details on 4.4.

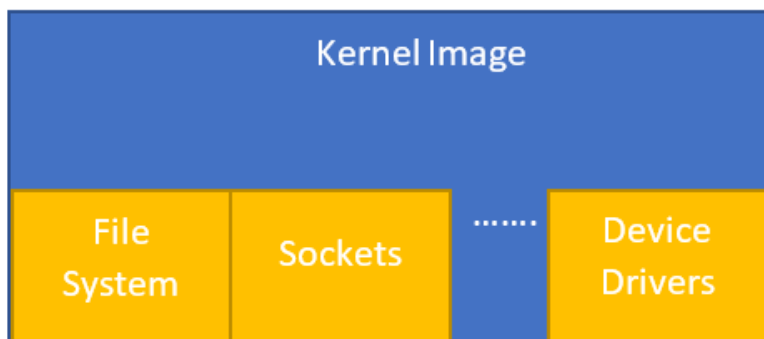


Figure 1 Kernel Image W/ out Modules

- b. Linux has a proper way to inject a driver onto running Linux distribution which is called as Linux Kernel Module (LKM). LKMs are not part of the main Kernel, they are injected by users onto running Kernel. User can determine which LKM module driver to including for kernel compilation. That helps to reduces the size of the Kernel. For example, if ethernet interface is not needed for a system, user can eliminate this LKM module in kernel compilation, so that the bootloader and Kernel module loader won't try to load that LKM into RAM.

LKM files are usually kept under /lib/modules folder and they are loaded according to distribution's configuration such as device tree, scripts etc.

1.3 Device Driver Structure

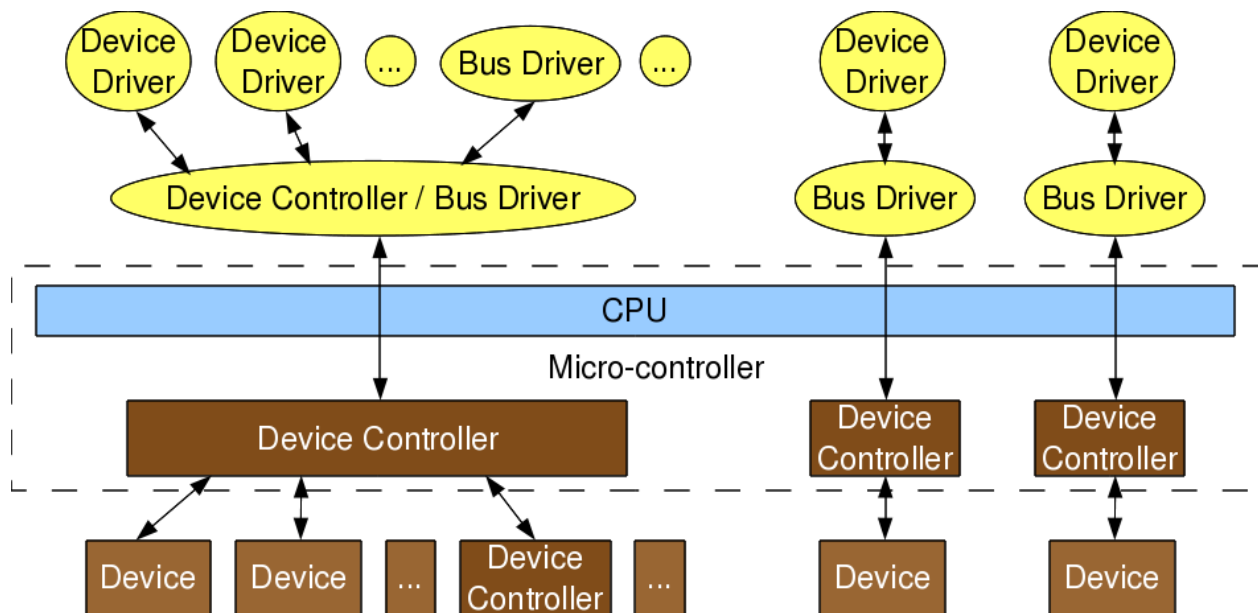


Figure 2 Linux Device Management [1]

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

2 Device Tree Parameters

Due to nature of HW environment, device drivers need to learn HW parameters. For example, 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 device tree of the system. This feature is supported for ARM after Linux Kernel Version 3.7. Device tree is HW description for kernel which is usually provided by board manufacturer/provider.

2.1 RTC Device Trees

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

Bus number, interrupt lines and device addresses are interpreted within kernel, so the driver only gets necessary bus structures from high level APIs. However, for custom parameters like trickle charger settings or power management mode, the device driver should have parser and have default values for those types of parameters.

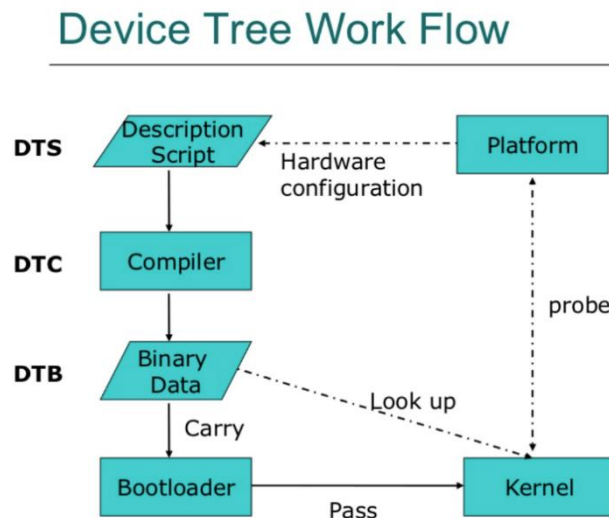


Figure 3 Device Tree Work Flow [2]

2.2 Device Tree Compilation

Device tree has its own compiler to generate output. More details can be found on [this link](#).

3 Linux Generic RTC Device Drivers

RTC device driver is a generic device driver for RTC ICs from manufacturers. To access driver features from user space, system calls must be used in user space programs. For RTC driver, IOCTL call is being used to access RTC features.

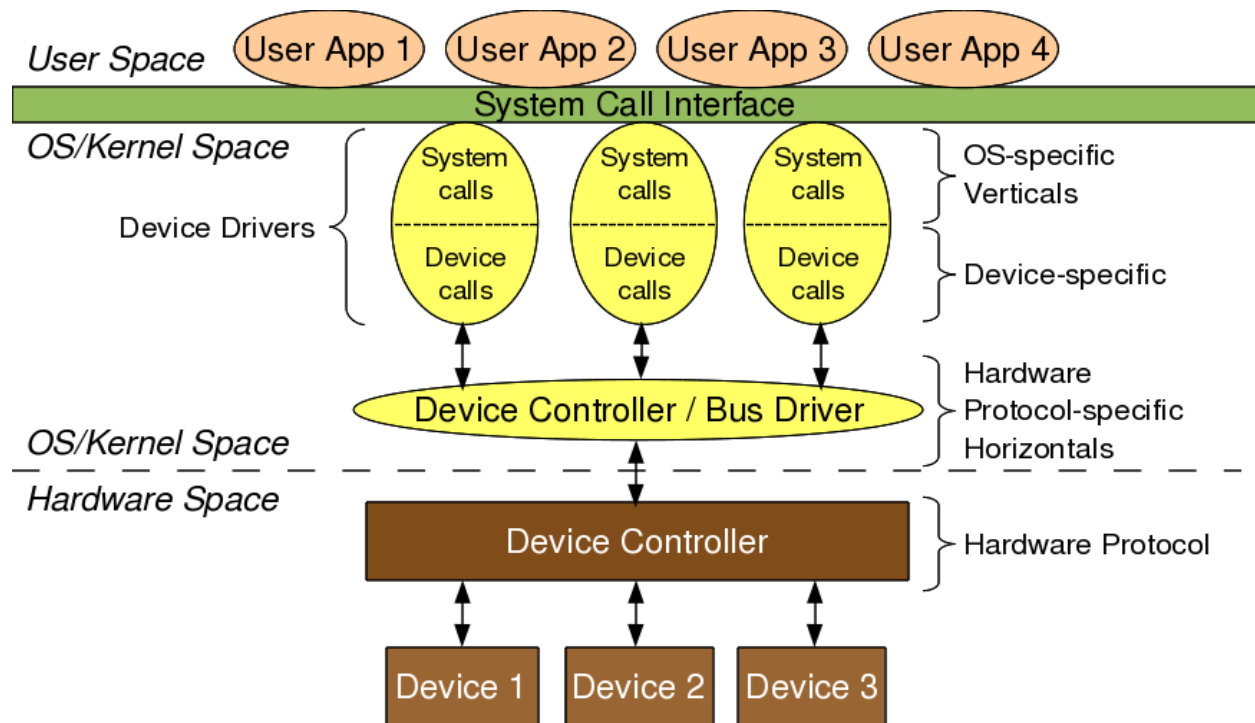


Figure 4 System calls to Device Drivers [1]

3.1 RTC Specific Default Features

M: must have O: optional N: Nice to have

For must have features and optional features, if not implemented, an invalid argument system code will be returned to user space. For optional features, if they are not implemented, user space programs can read the return code and respond to it if needed. For must have options, everything must be implemented.

Nice to have features may not be implemented in 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 RTC IC.
Linux Power Management (N)	Registering a device as wakeup of the system. Developer needs to set CONFIG_PM_SLEEP option in compilation parameters.

With those features above, an RTC driver will be ready to release on kernel.org and work in a Linux system.

3.2 Device Specific Custom Features

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

User who uses the file APIs may 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 simple text string. For example, user may use “cat” to read the state of the power management configuration, and use “echo” shell command to change it.

Sysfs is a pseudo file system that can be used by end user and mid user. Those files are usually read-only files, but some of them can take parameters via 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`

In Linux environment, cat program is designed to read data from a file. Likewise, echo program is designed to write data to a file.

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

On the other hand, user can use command line or Linux's file API to access the sysfs as files.

On both options, end/mid user must know parameters to give and to read. An example can be seen in figure 5.

```
#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	

4.1.2 *Maxim Version*

Unlike Kernel version of the driver, Maxim Website version has couple of extra features as described below. Main difference between them is IOCTL option will be used to demonstrate IC features. And also, external interrupt pin is supported by the driver. In order to make device driver to use INTA and INTB at the same time, external interrupt pin needs to be defined in device tree. More details on section 4.2.

Table 3 MAX31341B Maxim Version's Features

Maxim Driver	Description	Kernel Version Implementation
Custom IOCTL Options		
Power Management	Selecting current power mode of IC. - power-management: Changes default power management mode (mode 1) 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, Vcc is active supply 3 for power management manual and trickle charger is on, AIN is active supply, if $AIN > V_{cc}$	The module can be accessible via sysfs.
Alarm2	Supported.	Not implemented. Kernel generic RTC driver doesn't have this kind of option.
Register Read/Write Access	Supported.	Not implemented. Within Linux Kernel, a driver's flow mustn't be interrupted by external access. Since register access can break flow, it is not implemented.
External Clock	Supported.	Not implemented. Kernel generic RTC driver doesn't have this kind of option.
Data Retention	Supported	Not implemented. Kernel generic RTC driver doesn't have this kind of option.

4.2 Device Specific Device Tree Parameters

For MAX31341B drivers, there are 2 types of parameters.

1. Kernel parameters.
 - a. Main interrupt line
 - b. I2C bus identifier
 - c. I2C 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 are:

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, Vcc is active supply

3 - power management manual and trickle charger is on, AIN is active supply, if AIN > Vcc

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 */
        };
    };
};
```

For kernel.org, device specific parameters must be documented, and an example is uploaded to mainline kernel.

4.3 Testing Maxim Driver on Raspberry Pi 3B+

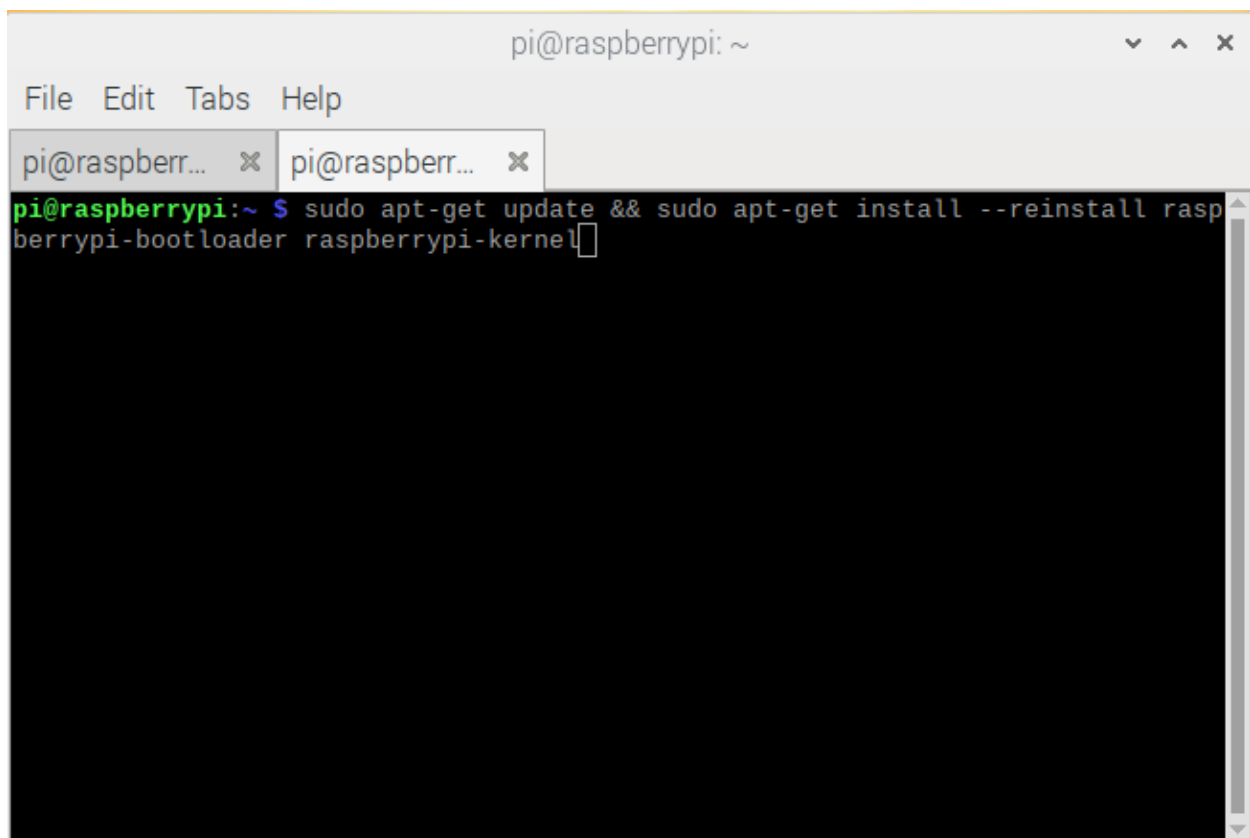
Example project has been prepared for RPi 3 B+. i2c-rtc overlay file should be updated to comprise MAX31341B driver. The driver device tree file will overlay on Raspberry pi kernel device tree.

Table 4 MAX31341B Pin Configuration

MAX31341B	RPi 3 B+
Vcc	Pin1 - 3V3
GND	Pin6 – Ground
SDA	Pin3 – GPIO2
SCL	Pin5 – GPIO3
INTA	Pin36 – GPIO16
INTB	Pin38 – GPIO20

- a. In order to provide portability, the driver should be compiled from the source code. Prior to doing that, Raspberry Pi kernel first needs to be updated to the latest version.
- `sudo apt-get update && sudo apt-get install --reinstall raspberrypi-bootloader raspberrypi-kernel`

and then, RPi needs to reboot with “reboot” command.



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ sudo apt-get update && sudo apt-get install --reinstall raspberrypi-bootloader raspberrypi-kernel
```

Figure 6

- b. Kernel headers must be installed to compile MAX31341B LKM.
- `sudo apt-get install raspberrypi-kernel-headers`
command will install necessary kernel headers for development.

- c. RPi 3 B+ doesn't use i2c automatically, so, the user must enable i2c bus and rtc in /boot/config.txt. Use "
 - sudo nano /boot/config.txt
 to open the file and add the lines below.
 - dtparam=i2c_arm=on
 - dtoverlay=i2c-rtc,max31341b
- d. A Makefile is required to create LKM from driver file. A Makefile is provided to work with Raspberry Pi 3B+. With that Makefile, user can compile the driver, device tree and install LKM to running Linux system.
 - Go to the folder with the Makefile
 - "make clean" command will clear all outputs of driver.
 - "make all" command will compile the driver and generate LKM (.ko) file.

```
pi@raspberrypi:~/Desktop/max31341_linux_driver $ make clean
make -C /lib/modules/4.19.97-v7+/build M=/home/pi/Desktop/max31341_linux_driver clean
make[1]: Entering directory '/usr/src/linux-headers-4.19.97-v7+'
  CLEAN   /home/pi/Desktop/max31341_linux_driver/.tmp_versions
  CLEAN   /home/pi/Desktop/max31341_linux_driver/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-4.19.97-v7+'
pi@raspberrypi:~/Desktop/max31341_linux_driver $ make all
make -C /lib/modules/4.19.97-v7+/build M=/home/pi/Desktop/max31341_linux_driver modules
make[1]: Entering directory '/usr/src/linux-headers-4.19.97-v7+'
  CC [M]   /home/pi/Desktop/max31341_linux_driver/rtc-max31341b.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC       /home/pi/Desktop/max31341_linux_driver/rtc-max31341b.mod.o
  LD [M]   /home/pi/Desktop/max31341_linux_driver/rtc-max31341b.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.19.97-v7+'
pi@raspberrypi:~/Desktop/max31341_linux_driver $
```

Figure 7

- "sudo make install" command will install the driver into LKM folder. Most of the system folder are not accessible by user. To access, user needs to use sudo extension for their command. This extension provides privileged access to user.

```
pi@raspberrypi:~/Desktop/max31341_linux_driver $ sudo make install
make -C /lib/modules/4.19.97-v7+/build M=/home/pi/Desktop/max31341_linux_driver INSTALL_MOD_PATH= modules_install
make[1]: Entering directory '/usr/src/linux-headers-4.19.97-v7+'
  INSTALL /home/pi/Desktop/max31341_linux_driver/rtc-max31341b.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/max31341_linux_driver $
```

Figure 8

- “make dtbs” command will compile device tree overlay (i2c-rtc-overlay.dts) and generates i2c-rtc.dtbo output. If the compilation is successful, it copies output to RPi’s /boot/overlays folder.

```
pi@raspberrypi:~/Desktop/max31341_linux_driver $ make dtbs
cpp -nostdinc -I include -I arch -undef -x assembler-with-cpp i2c-rtc-overlay.dts i2c-rtc-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
pi@raspberrypi:~/Desktop/max31341_linux_driver $
```

Figure 9

After these steps, restart the Raspberry Pi and the driver will be ready for test.

4.3.1 Basic Tests

For basic tests such as time setting/reading, “timedatectl” and “hwclock” commands can be used. These commands are provided by Raspbian OS, so, they don’t require any installation.

- Before doing any test, user should manually disable Network Time Protocol. If it is not disabled, NTP will update the system time periodically.
 - sudo timedatectl set-ntp no

command will disable NTP.

```
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

b. 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

- 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

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

Figure 13

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

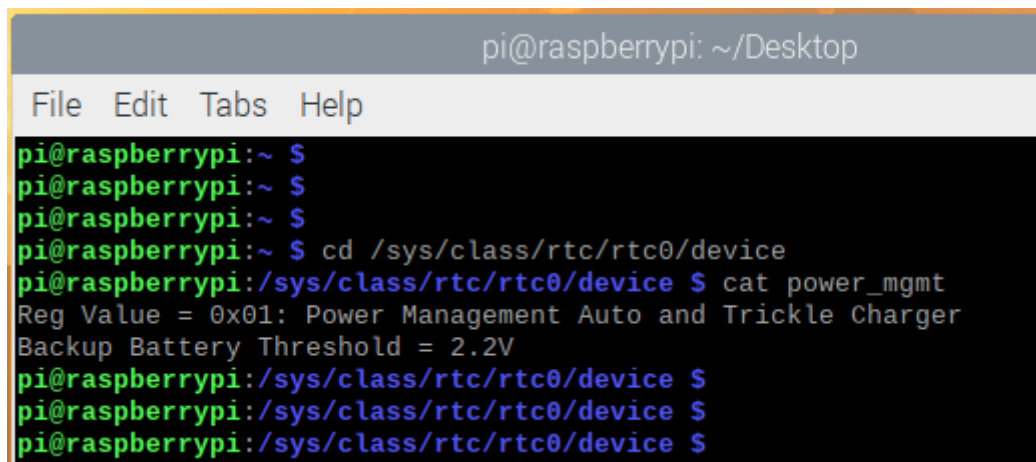
```
pi@raspberrypi:~/Desktop $ sudo hwclock --hctosys
pi@raspberrypi:~/Desktop $ timedatectl
    Local time: Mon 2020-01-27 14:50:09 GMT
    Universal time: Mon 2020-01-27 14:50:09 UTC
        RTC time: Mon 2020-01-27 14:50:09
        Time zone: Europe/London (GMT, +0000)
System clock synchronized: no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

Figure 14

4.3.2 SysFs Entry Test of Maxim Version Driver

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

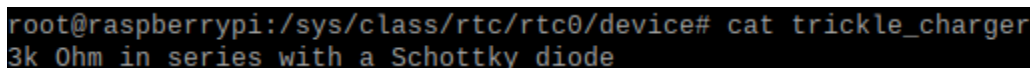
- a. “cat power_mgmt” command will read current power management mode and backup battery threshold from the RTC. User can change these settings using ioctl or device tree.



```
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. “cat trickle_charger” command reads the trickle charger settings. User can change the settings in device tree.



```
root@raspberrypi:/sys/class/rtc/rtc0/device# cat trickle_charger
3k 0hm in series with a Schottky diode
```

Figure 16

- c. If a sysfs entry allows user to write data, user needs to be root user. To change user mode,
 - sudo -scommand should be entered.

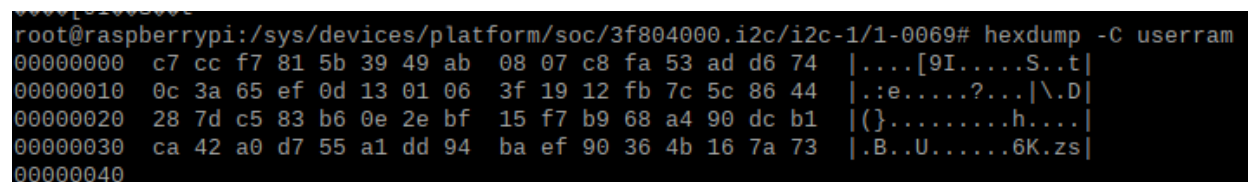


```
pi@raspberrypi:/sys/class/rtc/rtc0/device $ sudo -s
```

Figure 17

Unlike other sysfs entries, userram (nvmem) is binary interface, so, “cat” command doesn’t show meaningful data to user. In order to read,

- hexdump -C userram
- command should be used.



```
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

To write data to userram, root user access is a must. After getting access,

- echo "<data>" >> userram
command will do the work. If you want to write data in binary
- echo -n -e "<data in binary format>" >> userram
will work.

```
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 "cat additional_interrupt" command to see the status of remaining interrupts that are not used by the kernel. Those interrupts are analog power fail int., D1 input interrupt, loss of external clock interrupt 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 IOCTL options including userram, update interrupt, alarm interrupt, and sysfs read values. User should call the program as privileged user, and use
 - sudo chmod 777 rtc_ctest
to make sure the file is readable, writable, and executable
 - sudo ./rtc_ctest

command will call the program.

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

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. To use test program in python, you need to install necessary packages first. You can install them with the commands below:

- `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

After installing them you can run the program with the command below:

- `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 whole Kernel. To build kernel in your computer, you can follow these instructions: (Here Ubuntu is used to build kernel. If you use another system you can look at [this link](#).)

First of all, you need to download and install toolchain.

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

```
ens@ens-VirtualBox:~$ git clone https://github.com/raspberrypi/tools ~/tools
Cloning into '/home/ens/tools'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 25383 (delta 3), reused 8 (delta 3), pack-reused 25374
Receiving objects: 100% (25383/25383), 610.88 MiB | 2.44 MiB/s, done.
Resolving deltas: 100% (14884/14884), done.
Checking out files: 100% (19059/19059), done.
ens@ens-VirtualBox:~$ echo PATH=$PATH:~/tools/arm-bcm2708/arm-linux-gnueabi/bin >> ~/.bashrc
ens@ens-VirtualBox:~$ source ~/.bashrc
ens@ens-VirtualBox:~$
```

Figure 26

If you are using a 32-bit operating system (for example, our Raspberry Pi Desktop for PC), then you may need to install an additional set of libraries:

- `sudo apt install zlib1g-dev:amd64`

After these steps, you should download the kernel source.

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

```
ens@ens-VirtualBox:~$ git clone --depth=1 https://github.com/raspberrypi/linux
```

Figure 27

To build the sources for cross-compilation, make sure you have the dependencies needed on your machine by executing:

- `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

After these steps, `rtc-max31341b.c` should be copied under `./linux/drivers/rtc`.

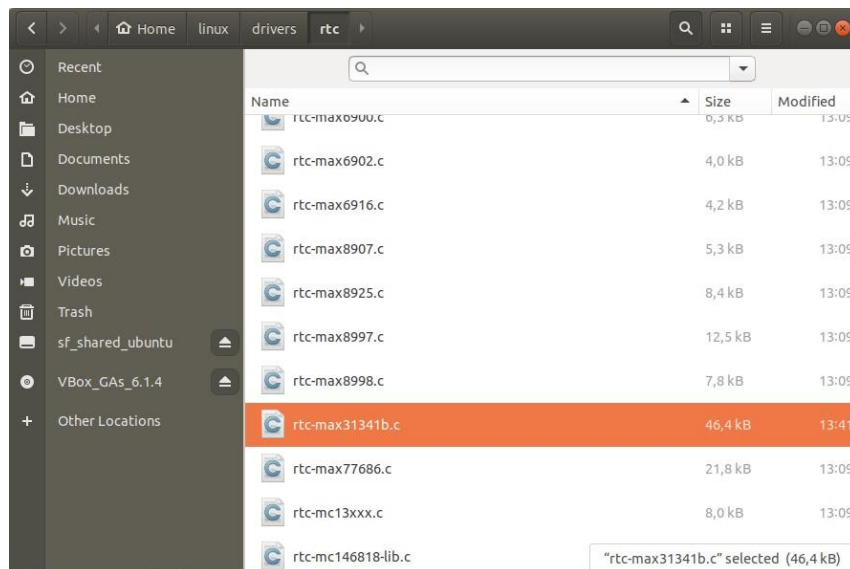


Figure 29

Those lines below should be added to drivers/rtc/Kconfig file:

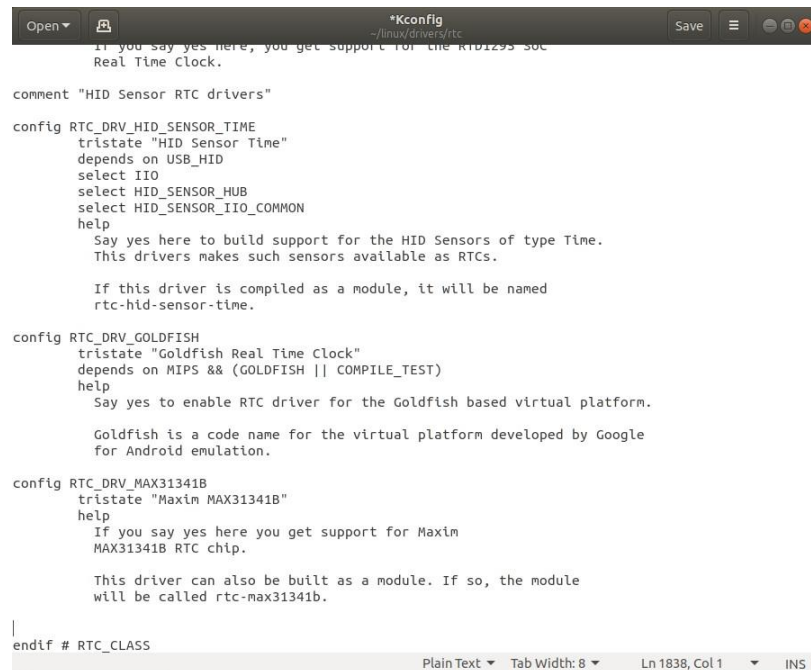
```
config RTC_DRV_MAX31341B
```

```
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
will be called rtc-max31341b.



```
Open ▾  *Kconfig  Save  ~/linux/drivers/rtc
If you say yes here, you get support for the RTD1293 SOC
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

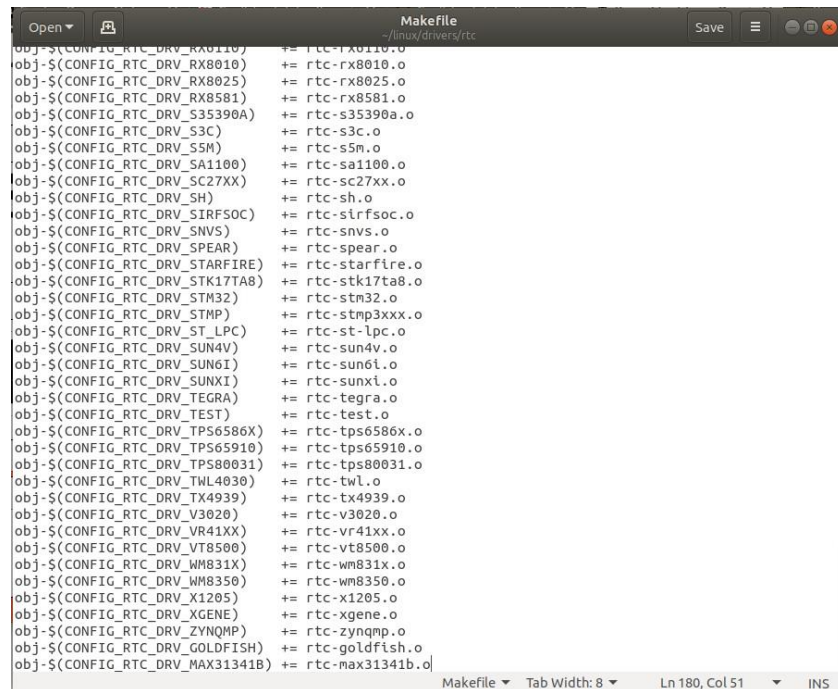


Figure 31

Before building the kernel, user must be sure that Maxim MAX31341B option is selected within Kernel Configuration. To configure kernel, you need to use menuconfig. (Here Ubuntu is used to build kernel for Raspberry Pi 3B+. If you use another system you can look at [this link](#).)

- KERNEL=kernel7
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- bcm2709_defconfig
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- menuconfig

```
ens@ens-VirtualBox:~/linux$ KERNEL=kernel7
ens@ens-VirtualBox:~/linux$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- bcm2709_defconfig
#
# configuration written to .config
#
ens@ens-VirtualBox:~/linux$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- menuconfig
```

Figure 32

You should follow these pictures:

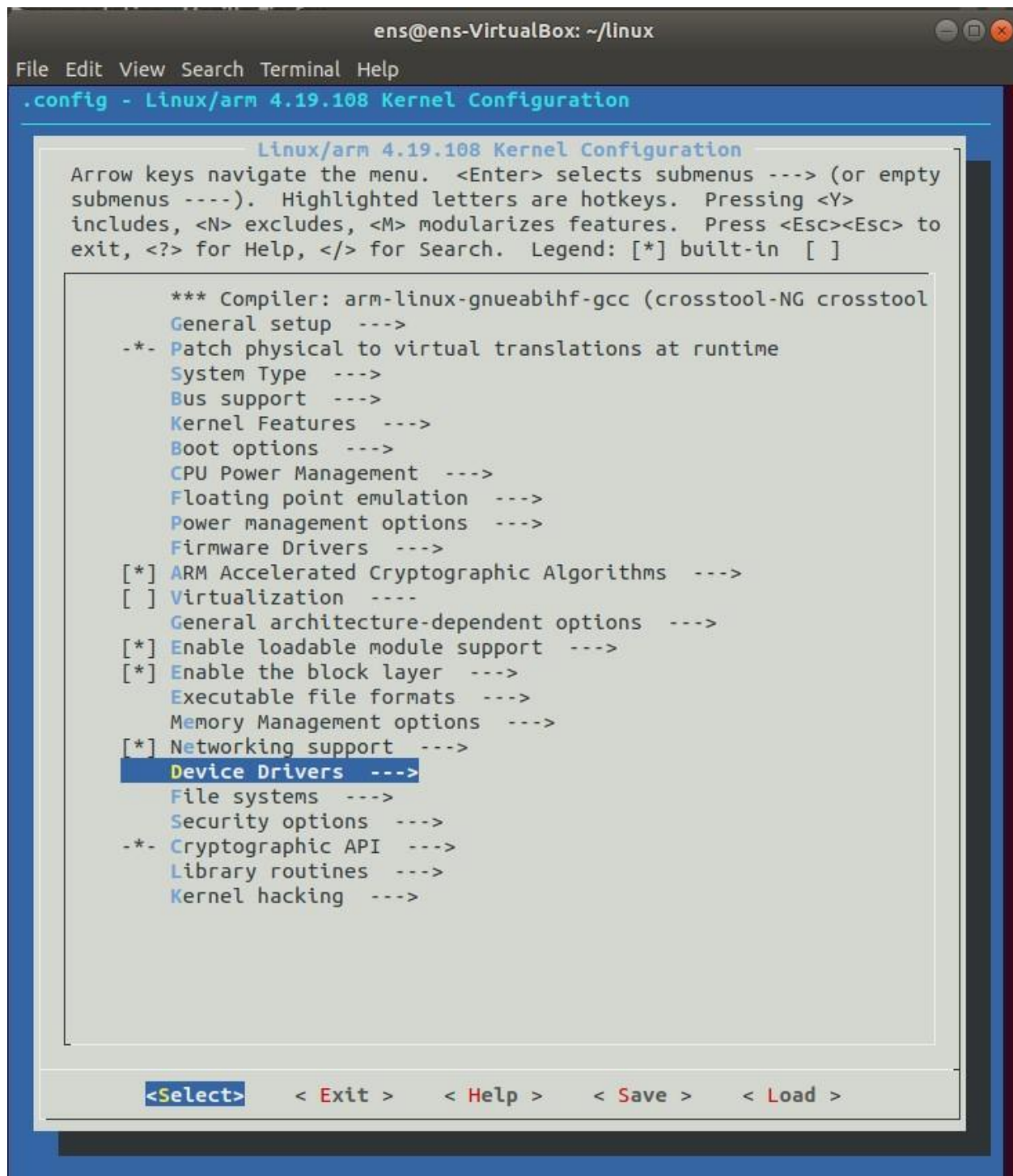


Figure 33

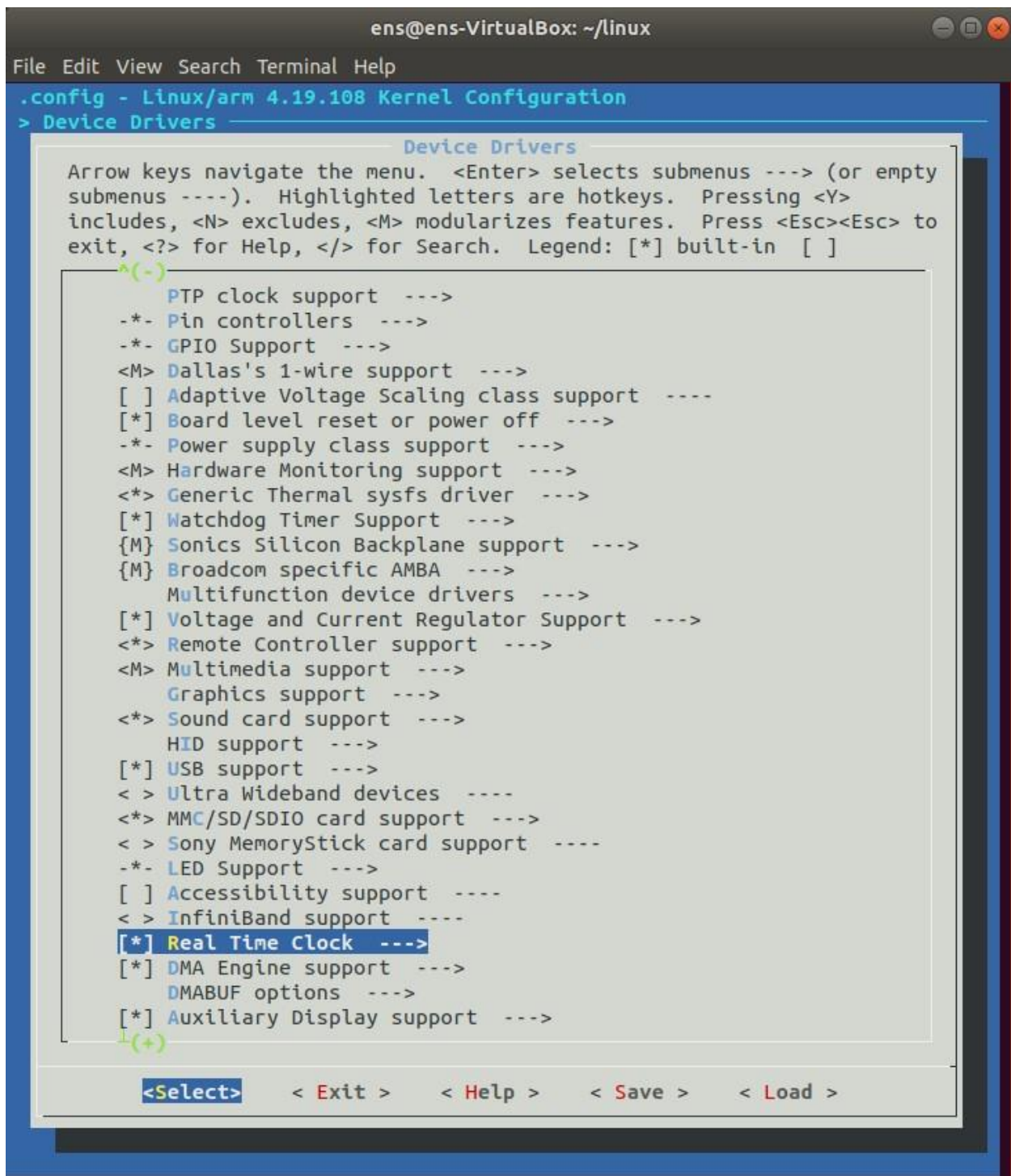


Figure 34

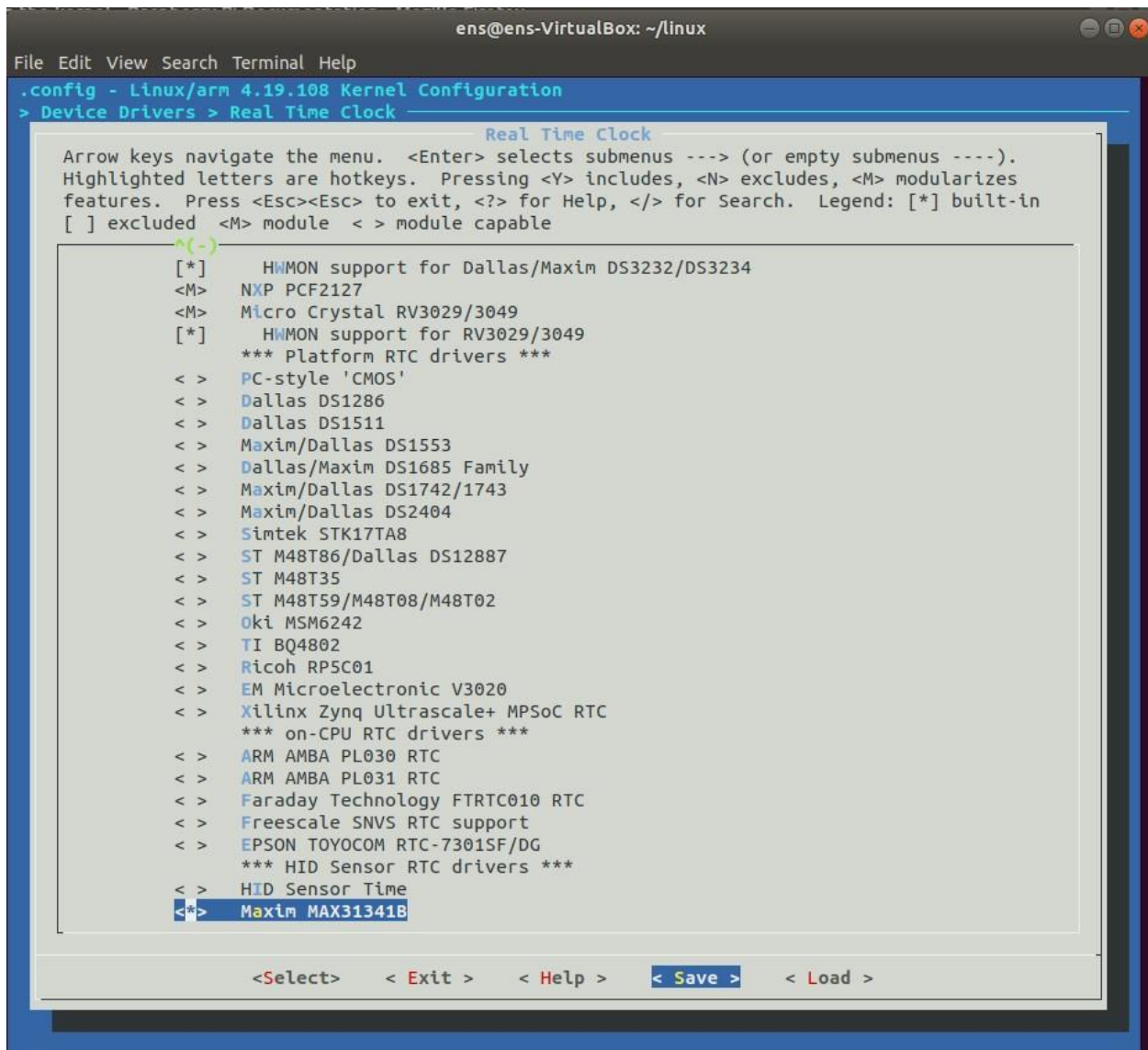


Figure 35

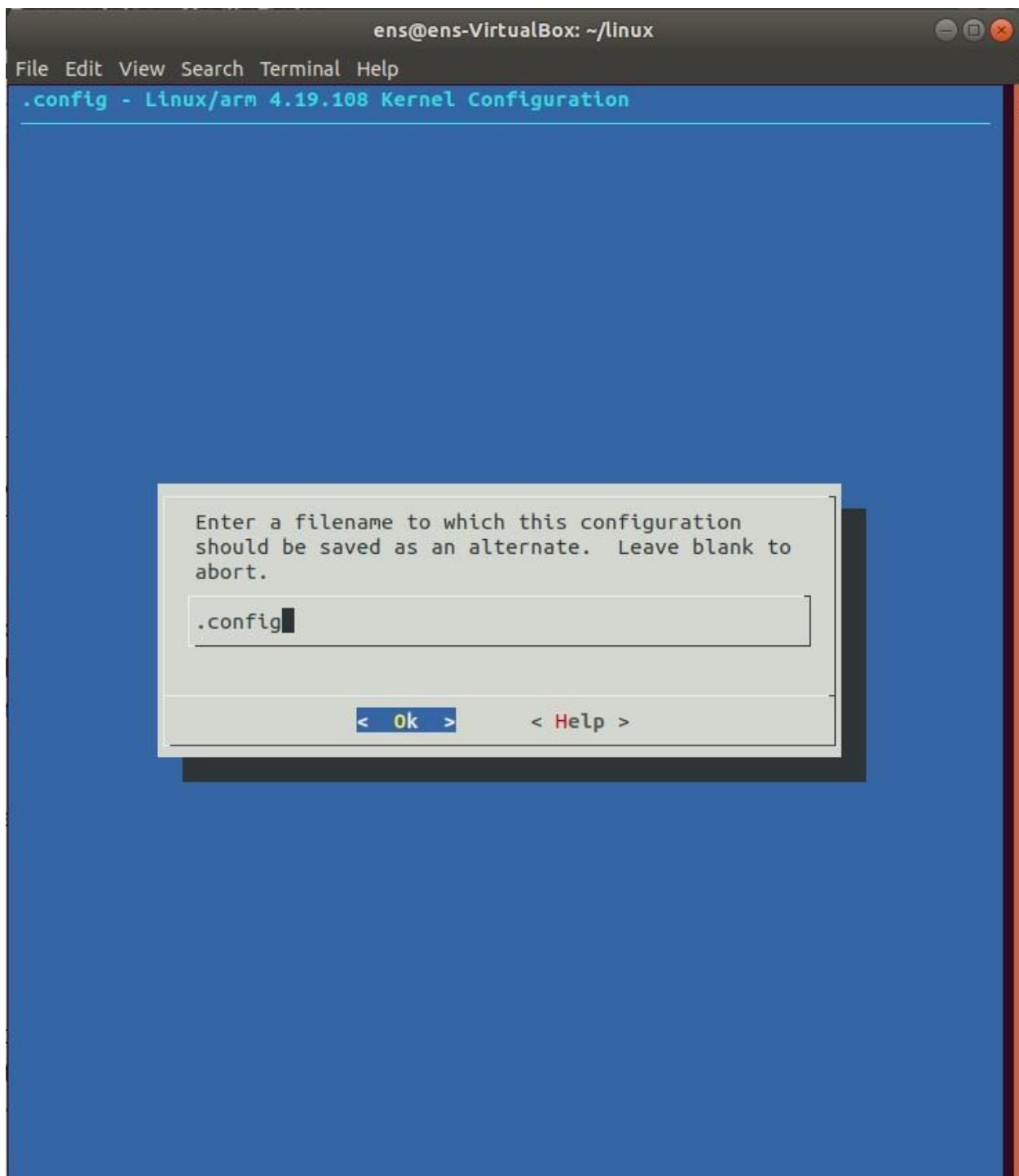
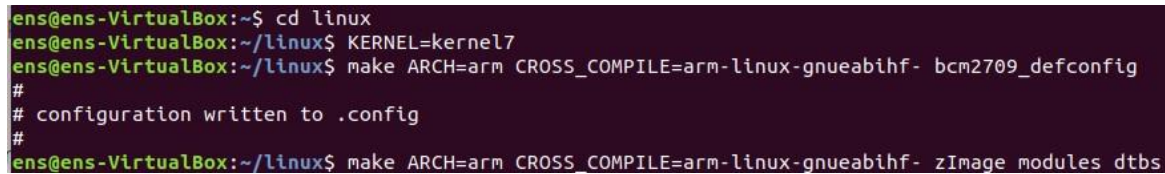


Figure 36

After all these steps, you can compile the kernel with these commands:

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

A terminal window with a dark purple background and light green text. The commands and output are as follows:

```
ens@ens-VirtualBox:~$ cd linux
ens@ens-VirtualBox:~/linux$ KERNEL=kernel7
ens@ens-VirtualBox:~/linux$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
#
# configuration written to .config
#
ens@ens-VirtualBox:~/linux$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
```

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-rtc>

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
1	06/20	Initial release	—

©2019 by Maxim Integrated Products, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. MAXIM INTEGRATED PRODUCTS, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. MAXIM ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering or registered trademarks of Maxim Integrated Products, Inc. All other product or service names are the property of their respective owners.