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.

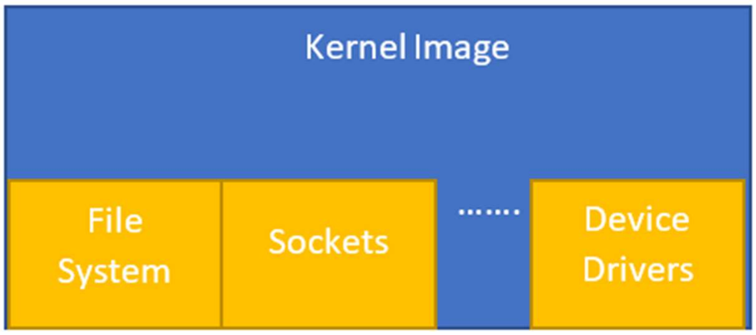


Figure 1. Kernel image w/out modules.

   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

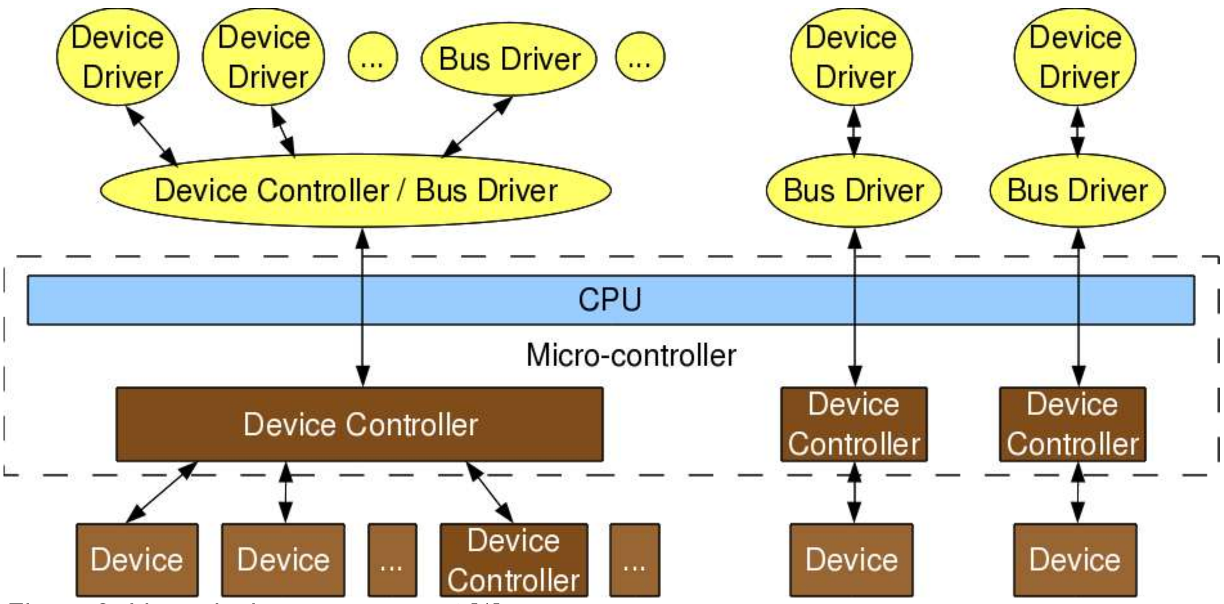


Figure 2. Linux device management [1].

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.

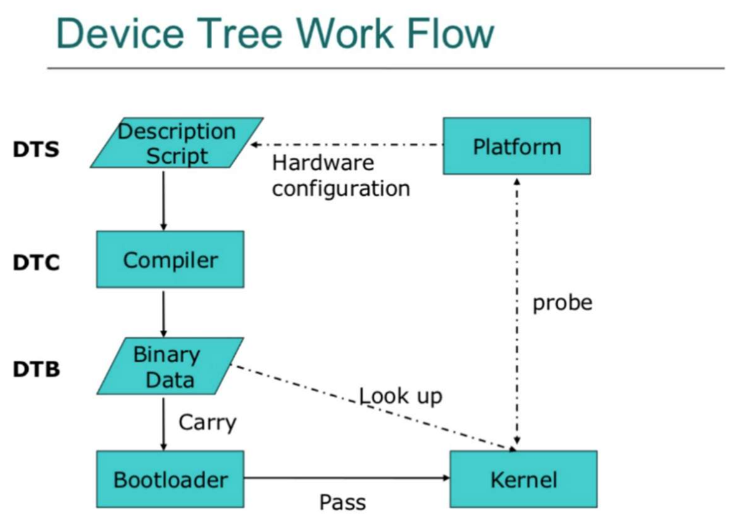


Figure 3. Device tree workflow [2].

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.

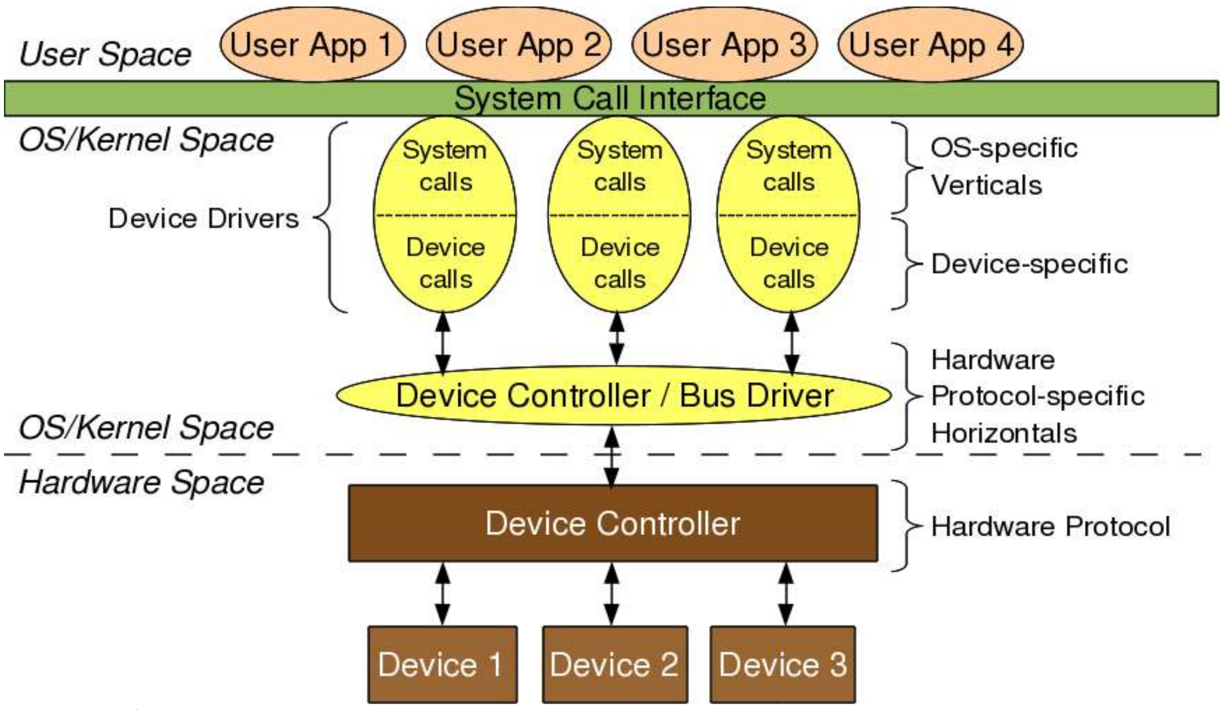


Figure 4. System calls to device drivers [1].

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.

4. MAX31328 Device Driver Package

4.1. Supported Features

4.1.1. Kernel Version

Table 2. MAX31328 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	N/A	

Here is the driver for Linux Kernel 5.x.x.

rtc-max31328.c

4.2. Testing the Maxim Driver on Raspberry Pi 4

A project example was prepared for RPi 4. The i^{2}c-rtc overlay file must be updated to comprise the MAX31328 driver. The driver device tree file overlays on the Raspberry Pi kernel device tree.

Table 3. MAX31328 Pin Configuration.

MAX31328	RPi 4
V_{CC}	Pin1 - 3V3
GND	Pin6 - Ground
SDA	Pin3 - GPIO2
SCL	Pin5 - GPIO3

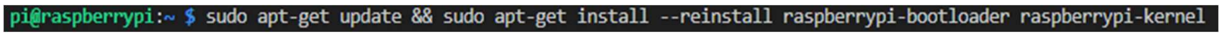
Here is the kernel.org version of the device tree overlay for Raspberry Pi.

max31328-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.



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

Figure 5

b. Kernel headers must be installed to compile the MAX31328 LKM.

- sudo apt-get install raspberrypi-kernel-headers

  The command installs the necessary kernel headers for development.

c. RPi 4 does not use i^{2}c automatically. So, enable the i^{2}c bus and rtc in /boot/config.txt.

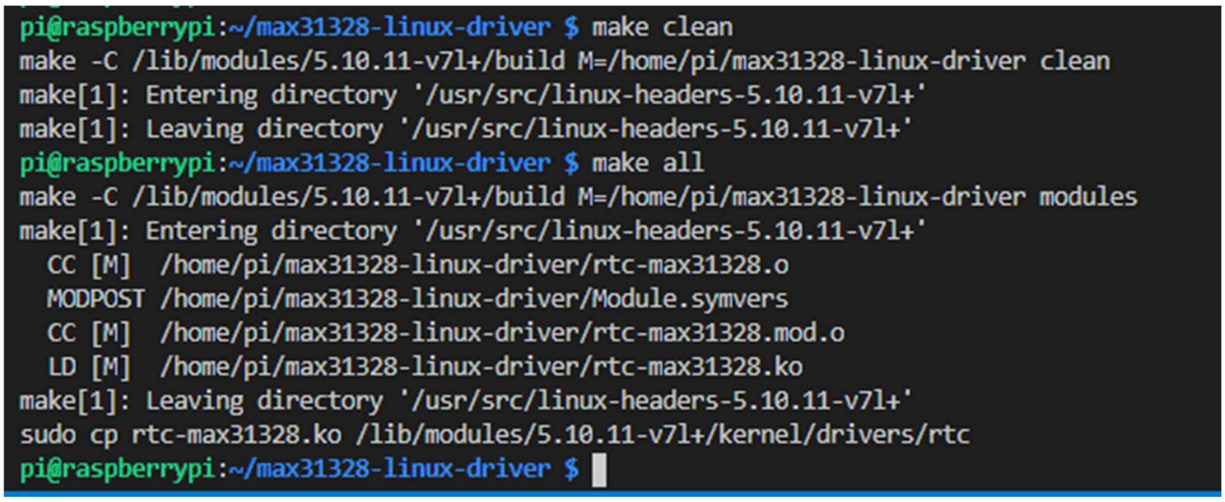
- sudo nano /boot/config.txt

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

  - dtparam=i2c_arm=on
  - dtoverlay=rtc-max31328,max31328

d. A Makefile is required to create the LKM from the driver file. A Makefile is provided to work with Raspberry Pi 4. The Makefile can be used to compile the driver, device tree, and install the LKM to run the Linux system.

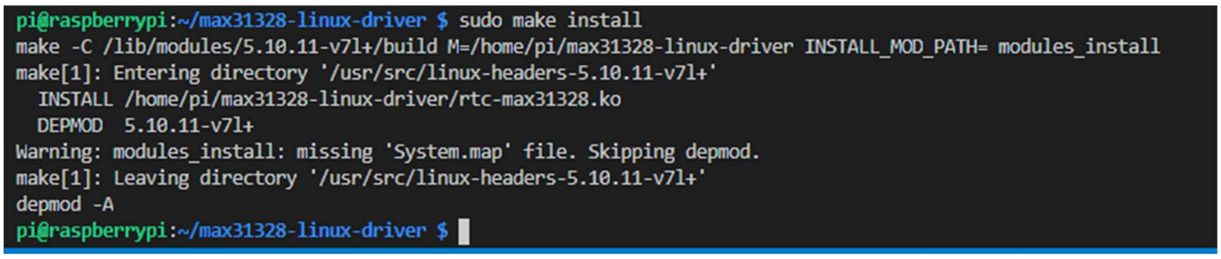
- Go to the folder with the Makefile.
- The **make clean** command clears all outputs of the driver.
- The **make all** command compiles the driver and generates the LKM (.ko) file.



```
pi@raspberrypi:~/max31328-linux-driver $ make clean
make -C /lib/modules/5.10.11-v7l+/build M=/home/pi/max31328-linux-driver clean
make[1]: Entering directory '/usr/src/linux-headers-5.10.11-v7l+'
make[1]: Leaving directory '/usr/src/linux-headers-5.10.11-v7l+'
pi@raspberrypi:~/max31328-linux-driver $ make all
make -C /lib/modules/5.10.11-v7l+/build M=/home/pi/max31328-linux-driver modules
make[1]: Entering directory '/usr/src/linux-headers-5.10.11-v7l+'
  CC [M]  /home/pi/max31328-linux-driver/rtc-max31328.o
  MODPOST /home/pi/max31328-linux-driver/Module.symvers
  CC [M]  /home/pi/max31328-linux-driver/rtc-max31328.mod.o
  LD [M]  /home/pi/max31328-linux-driver/rtc-max31328.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.10.11-v7l+'
sudo cp rtc-max31328.ko /lib/modules/5.10.11-v7l+/kernel/drivers/rtc
pi@raspberrypi:~/max31328-linux-driver $ 
```

Figure 6

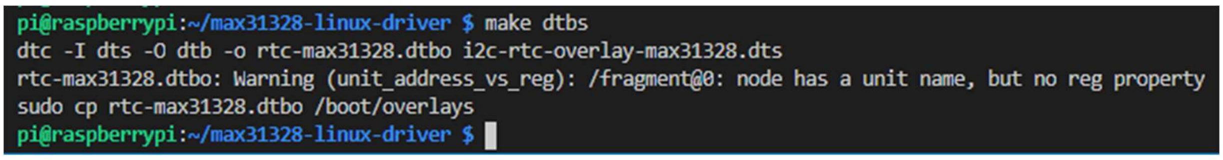
- The **sudo make install** command installs the driver into the LKM folder. Most system folders are not accessible to users. Use the sudo extension for the commands for access. This extension provides privileged access.



```
pi@raspberrypi:~/max31328-linux-driver $ sudo make install
make -C /lib/modules/5.10.11-v7l+/build M=/home/pi/max31328-linux-driver INSTALL_MOD_PATH= modules_install
make[1]: Entering directory '/usr/src/linux-headers-5.10.11-v7l+'
  INSTALL /home/pi/max31328-linux-driver/rtc-max31328.ko
  DEPMOD  5.10.11-v7l+
Warning: modules_install: missing 'System.map' file. Skipping depmod.
make[1]: Leaving directory '/usr/src/linux-headers-5.10.11-v7l+'
depmod -A
pi@raspberrypi:~/max31328-linux-driver $ 
```

Figure 7

- The **make dtbs** command compiles the device tree overlay (i2c-rtc-overlay-max31328.dts) and generates the rtc-max31328.dtbo output. It copies the output to RPi's /boot/overlays folder if the compilation is successful.



```
pi@raspberrypi:~/max31328-linux-driver $ make dtbs
dtc -I dts -O dtb -o rtc-max31328.dtbo i2c-rtc-overlay-max31328.dts
rtc-max31328.dtbo: Warning (unit_address_vs_reg): /fragment@0: node has a unit name, but no reg property
sudo cp rtc-max31328.dtbo /boot/overlays
pi@raspberrypi:~/max31328-linux-driver $ 
```

Figure 8

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

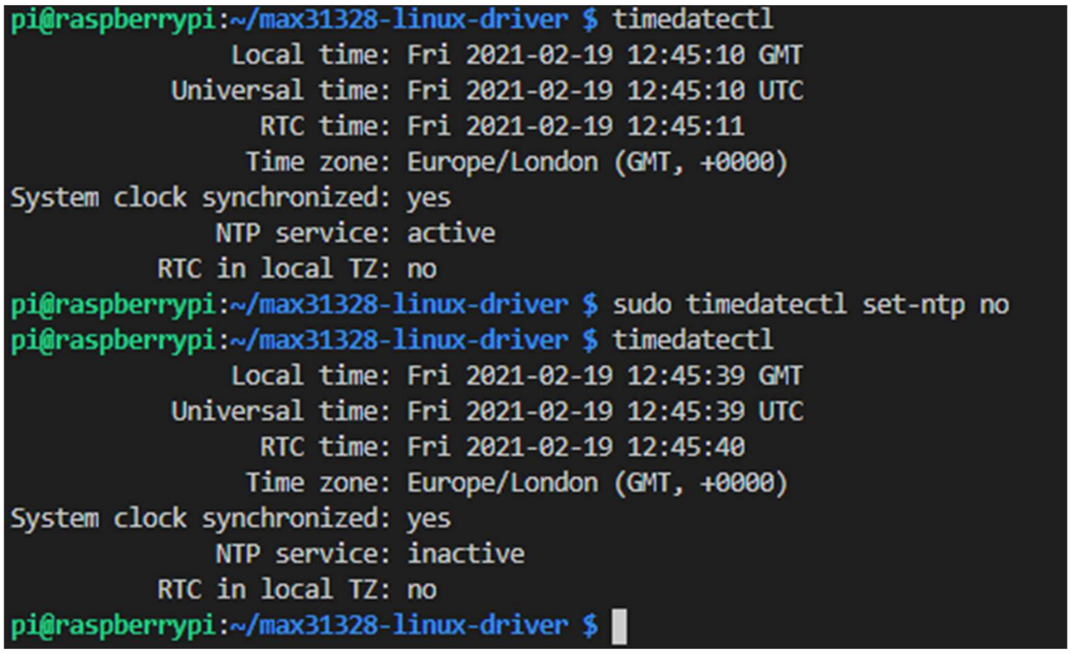
4.2.1. Basic Tests

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

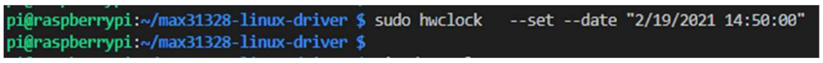


```
pi@raspberrypi:~/max31328-linux-driver $ timedatectl
               Local time: Fri 2021-02-19 12:45:10 GMT
           Universal time: Fri 2021-02-19 12:45:10 UTC
                 RTC time: Fri 2021-02-19 12:45:11
                Time zone: Europe/London (GMT, +0000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
pi@raspberrypi:~/max31328-linux-driver $ sudo timedatectl set-ntp no
pi@raspberrypi:~/max31328-linux-driver $ timedatectl
               Local time: Fri 2021-02-19 12:45:39 GMT
           Universal time: Fri 2021-02-19 12:45:39 UTC
                 RTC time: Fri 2021-02-19 12:45:40
                Time zone: Europe/London (GMT, +0000)
System clock synchronized: yes
              NTP service: inactive
          RTC in local TZ: no
pi@raspberrypi:~/max31328-linux-driver $
```

Figure 9

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

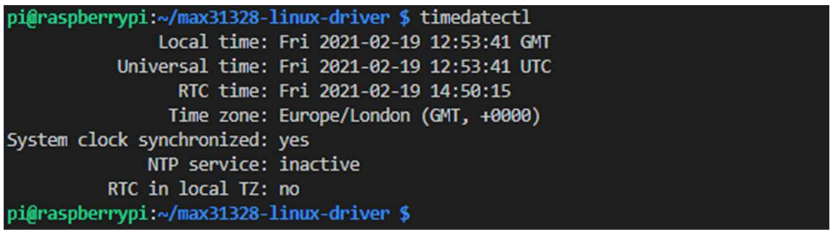
- Set RTC time: “sudo hwclock   --set --date "2/19/2021 14:50:00" “



```
pi@raspberrypi:~/max31328-linux-driver $ sudo hwclock   --set --date "2/19/2021 14:50:00"
pi@raspberrypi:~/max31328-linux-driver $
```

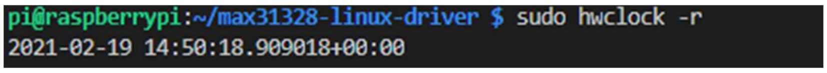
Figure 10

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



```
pi@raspberrypi:~/max31328-linux-driver $ timedatectl
               Local time: Fri 2021-02-19 12:53:41 GMT
           Universal time: Fri 2021-02-19 12:53:41 UTC
                 RTC time: Fri 2021-02-19 14:50:15
                Time zone: Europe/London (GMT, +0000)
System clock synchronized: yes
              NTP service: inactive
          RTC in local TZ: no
pi@raspberrypi:~/max31328-linux-driver $
```

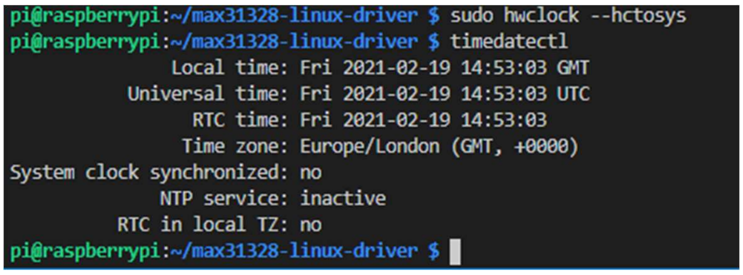
Figure 11



```
pi@raspberrypi:~/max31328-linux-driver $ sudo hwclock -r
2021-02-19 14:50:18.909018+00:00
```

Figure 12

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



```
pi@raspberrypi:~/max31328-linux-driver $ sudo hwclock --hctosys
pi@raspberrypi:~/max31328-linux-driver $ timedatectl
               Local time: Fri 2021-02-19 14:53:03 GMT
           Universal time: Fri 2021-02-19 14:53:03 UTC
                 RTC time: Fri 2021-02-19 14:53:03
                Time zone: Europe/London (GMT, +0000)
System clock synchronized: no
              NTP service: inactive
          RTC in local TZ: no
pi@raspberrypi:~/max31328-linux-driver $
```

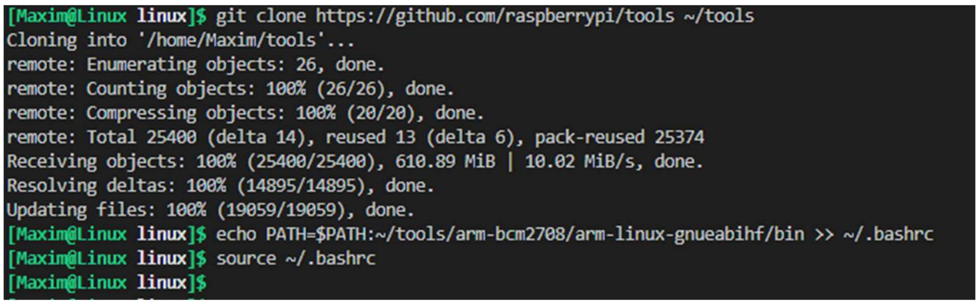
Figure 13

4.3. 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



```
[Maxim@Linux linux]$ git clone https://github.com/raspberrypi/tools ~/tools
Cloning into '/home/Maxim/tools'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 25400 (delta 14), reused 13 (delta 6), pack-reused 25374
Receiving objects: 100% (25400/25400), 610.89 MiB | 10.02 MiB/s, done.
Resolving deltas: 100% (14895/14895), done.
Updating files: 100% (19059/19059), done.
[Maxim@Linux linux]$ echo PATH=$PATH:~/tools/arm-bcm2708/arm-linux-gnueabihf/bin >> ~/.bashrc
[Maxim@Linux linux]$ source ~/.bashrc
[Maxim@Linux linux]$
```

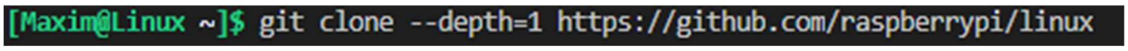
Figure 14

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

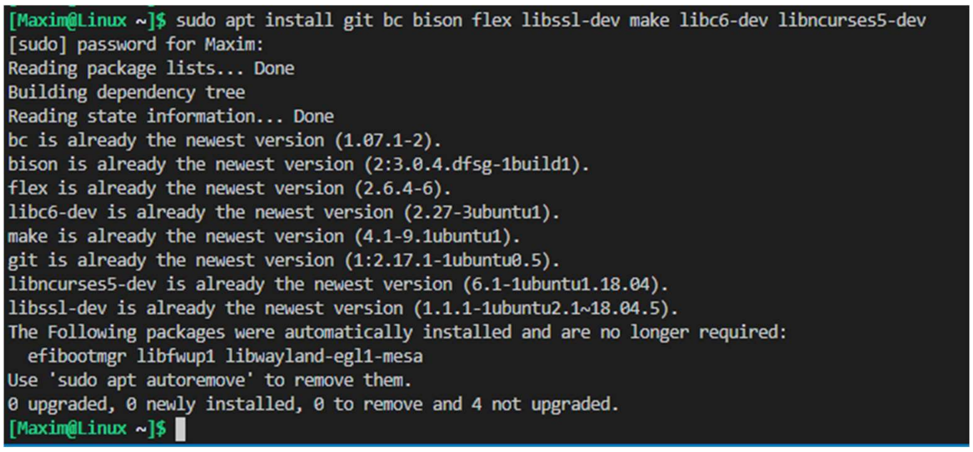


```
[Maxim@Linux ~]$ git clone --depth=1 https://github.com/raspberrypi/linux
```

Figure 15

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

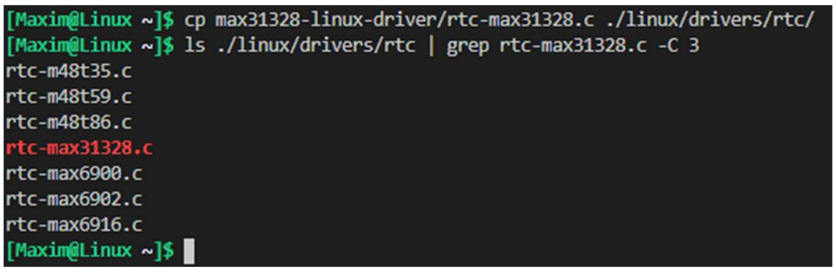
- sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev



```
[Maxim@Linux ~]$ sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev
[sudo] password for Maxim:
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.
[Maxim@Linux ~]$
```

Figure 16

Copy rtc-max31328.c in ./linux/drivers/rtc after these steps.



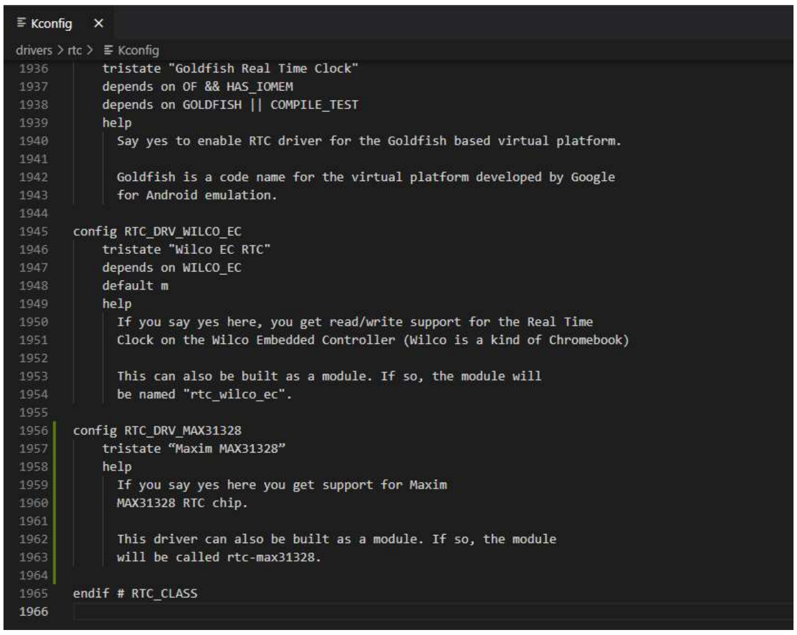
```
[Maxim@Linux ~]$ cp max31328-linux-driver/rtc-max31328.c ./linux/drivers/rtc/
[Maxim@Linux ~]$ ls ./linux/drivers/rtc | grep rtc-max31328.c -C 3
rtc-m48t35.c
rtc-m48t59.c
rtc-m48t86.c
rtc-max31328.c
rtc-max6900.c
rtc-max6902.c
rtc-max6916.c
[Maxim@Linux ~]$
```

Figure 17

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

```
config RTC_DRV_MAX31328
  tristate "Maxim MAX31328"
  help
    If you say yes here you get support for Maxim
    MAX31328 RTC chip.
```

This driver can also be built as a module. If so, the module will be called rtc-max31328.

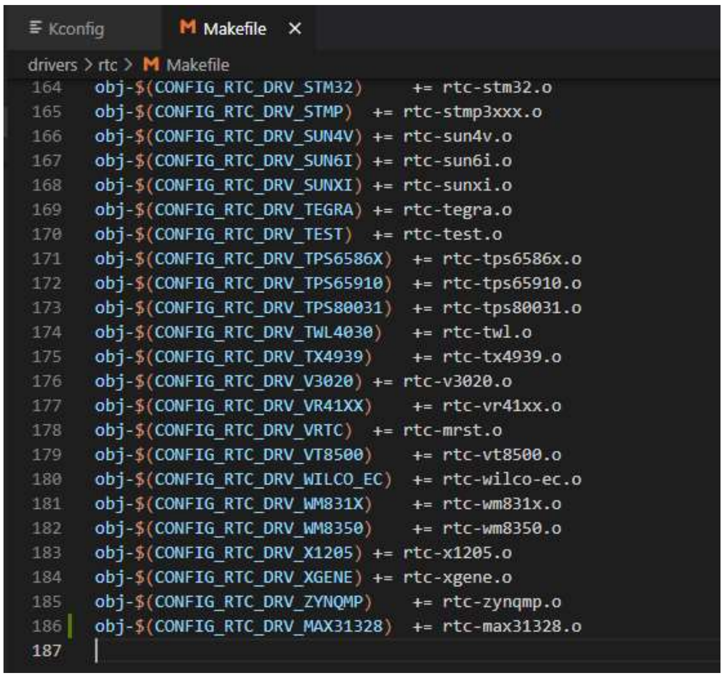


```
	tristate "Goldfish Real Time Clock"
	depends on OF && HAS_IOMEM
	depends on GOLDFISH || COMPILE_TEST
	help
	  Say yes to enable RTC driver for the Goldfish based virtual platform.

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

config RTC_DRV_WILCO_EC
	tristate "Wilco EC RTC"
	depends on WILCO_EC
	default m
	help
	  If you say yes here, you get read/write support for the Real Time
	  Clock on the Wilco Embedded Controller (Wilco is a kind of Chromebook)

	  This can also be built as a module. If so, the module will
	  be named "rtc_wilco_ec".

config RTC_DRV_MAX31328
	tristate “Maxim MAX31328”
	help
	  If you say yes here you get support for Maxim
	  MAX31328 RTC chip.

	  This driver can also be built as a module. If so, the module
	  will be called rtc-max31328.

endif # RTC_CLASS
```

Figure 18

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

obj-$(CONFIG_RTC_DRV_MAX31328) += rtc-max31328.o

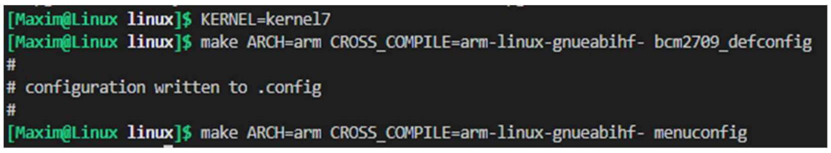


```
obj-$(CONFIG_RTC_DRV_STM32)	+= rtc-stm32.o
obj-$(CONFIG_RTC_DRV_STMP)	+= rtc-stmp3xxx.o
obj-$(CONFIG_RTC_DRV_SUN4V)	+= rtc-sun4v.o
obj-$(CONFIG_RTC_DRV_SUN6I)	+= rtc-sun6i.o
obj-$(CONFIG_RTC_DRV_SUNXI)	+= rtc-sunxi.o
obj-$(CONFIG_RTC_DRV_TEGRA)	+= rtc-tegra.o
obj-$(CONFIG_RTC_DRV_TEST)	+= rtc-test.o
obj-$(CONFIG_RTC_DRV_TPS6586X)	+= rtc-tps6586x.o
obj-$(CONFIG_RTC_DRV_TPS65910)	+= rtc-tps65910.o
obj-$(CONFIG_RTC_DRV_TPS80031)	+= rtc-tps80031.o
obj-$(CONFIG_RTC_DRV_TWL4030)	+= rtc-twl.o
obj-$(CONFIG_RTC_DRV_TX4939)	+= rtc-tx4939.o
obj-$(CONFIG_RTC_DRV_V3020)	+= rtc-v3020.o
obj-$(CONFIG_RTC_DRV_VR41XX)	+= rtc-vr41xx.o
obj-$(CONFIG_RTC_DRV_VRTC)	+= rtc-mrst.o
obj-$(CONFIG_RTC_DRV_VT8500)	+= rtc-vt8500.o
obj-$(CONFIG_RTC_DRV_WILCO_EC)	+= rtc-wilco-ec.o
obj-$(CONFIG_RTC_DRV_WM831X)	+= rtc-wm831x.o
obj-$(CONFIG_RTC_DRV_WM8350)	+= rtc-wm8350.o
obj-$(CONFIG_RTC_DRV_X1205)	+= rtc-x1205.o
obj-$(CONFIG_RTC_DRV_XGENE)	+= rtc-xgene.o
obj-$(CONFIG_RTC_DRV_ZYNQMP)	+= rtc-zynqmp.o
obj-$(CONFIG_RTC_DRV_MAX31328)	+= rtc-max31328.o
```

Figure 19

Ensure the Maxim MAX31328 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 4. Use **this link** for another system.)

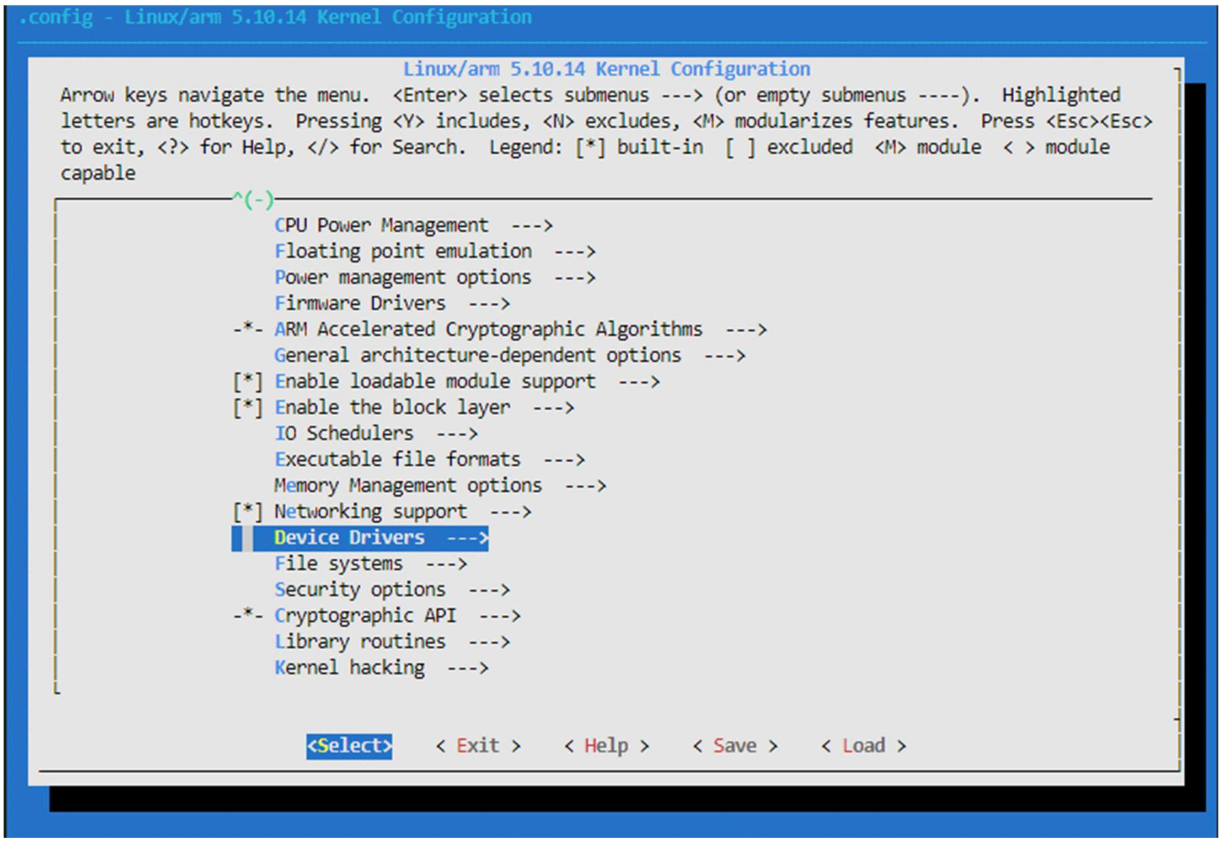
- KERNEL=kernel7
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig



```
[Maxim@Linux linux]$ KERNEL=kernel7
[Maxim@Linux linux]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
#
# configuration written to .config
#
[Maxim@Linux linux]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

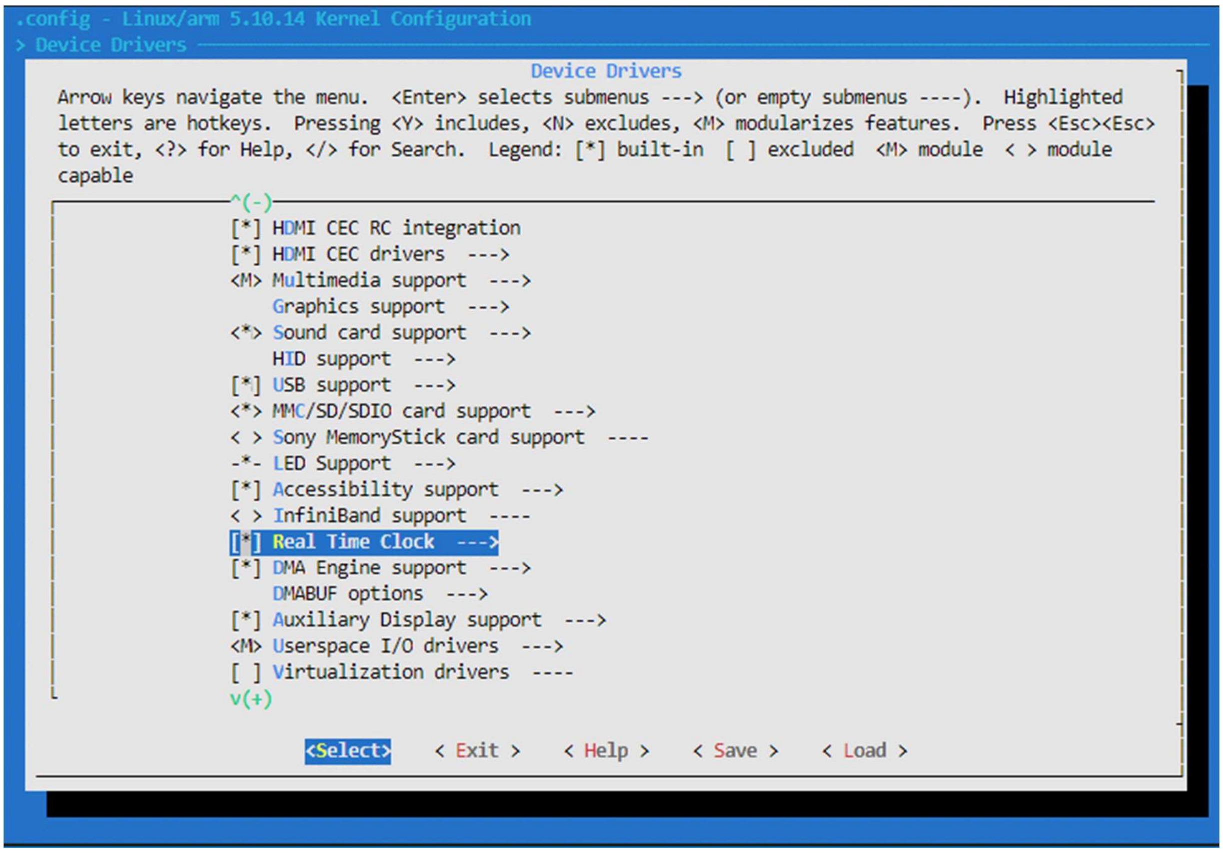
Figure 20

Follow these figures:



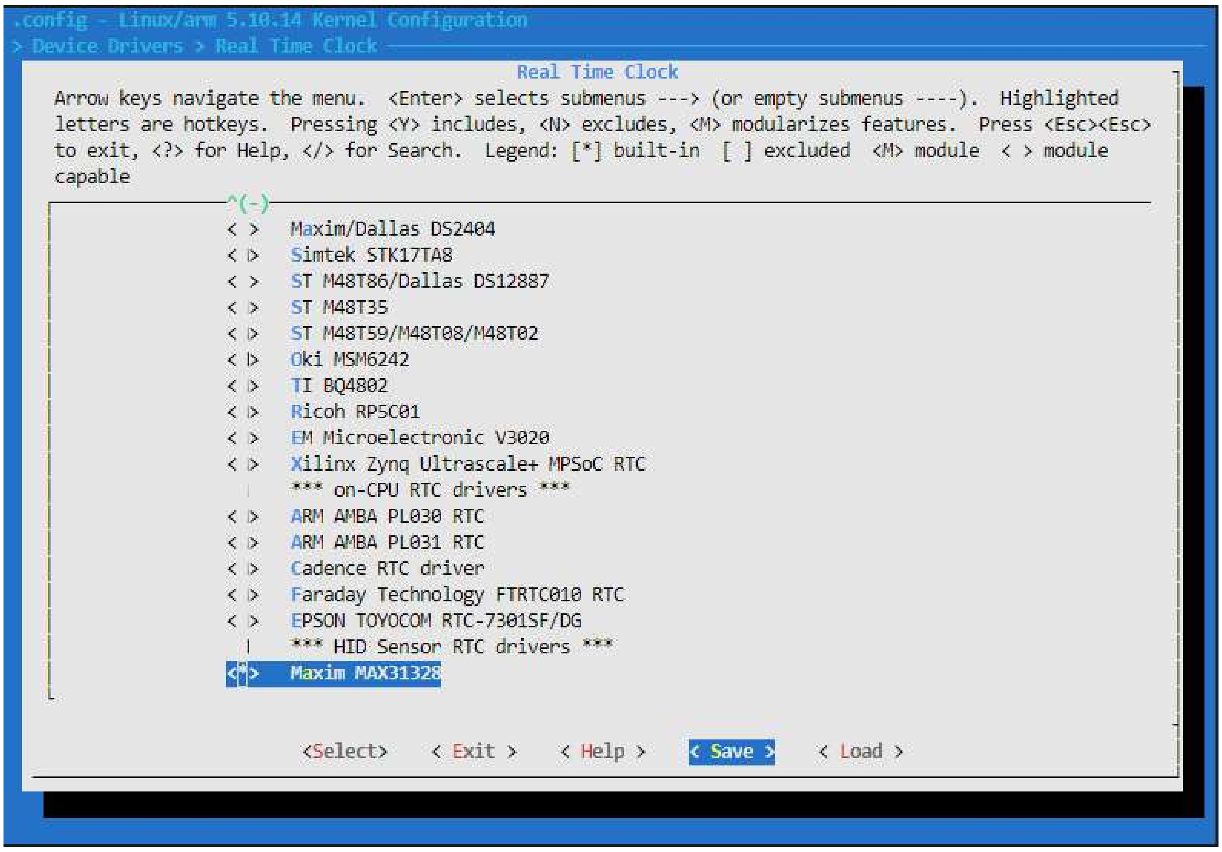
```
.config - Linux/arm 5.10.14 Kernel Configuration

                      Linux/arm 5.10.14 Kernel Configuration
  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
                  ^(-)
                      CPU Power Management  --->
                      Floating point emulation  --->
                      Power management options  --->
                      Firmware Drivers  --->
                  -*- ARM Accelerated Cryptographic Algorithms  --->
                      General architecture-dependent options  --->
                  [*] Enable loadable module support  --->
                  [*] Enable the block layer  --->
                      IO Schedulers  --->
                      Executable file formats  --->
                      Memory Management options  --->
                  [*] Networking support  --->
                      Device Drivers  --->
                      File systems  --->
                      Security options  --->
                  -*- Cryptographic API  --->
                      Library routines  --->
                      Kernel hacking  --->

                <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 21



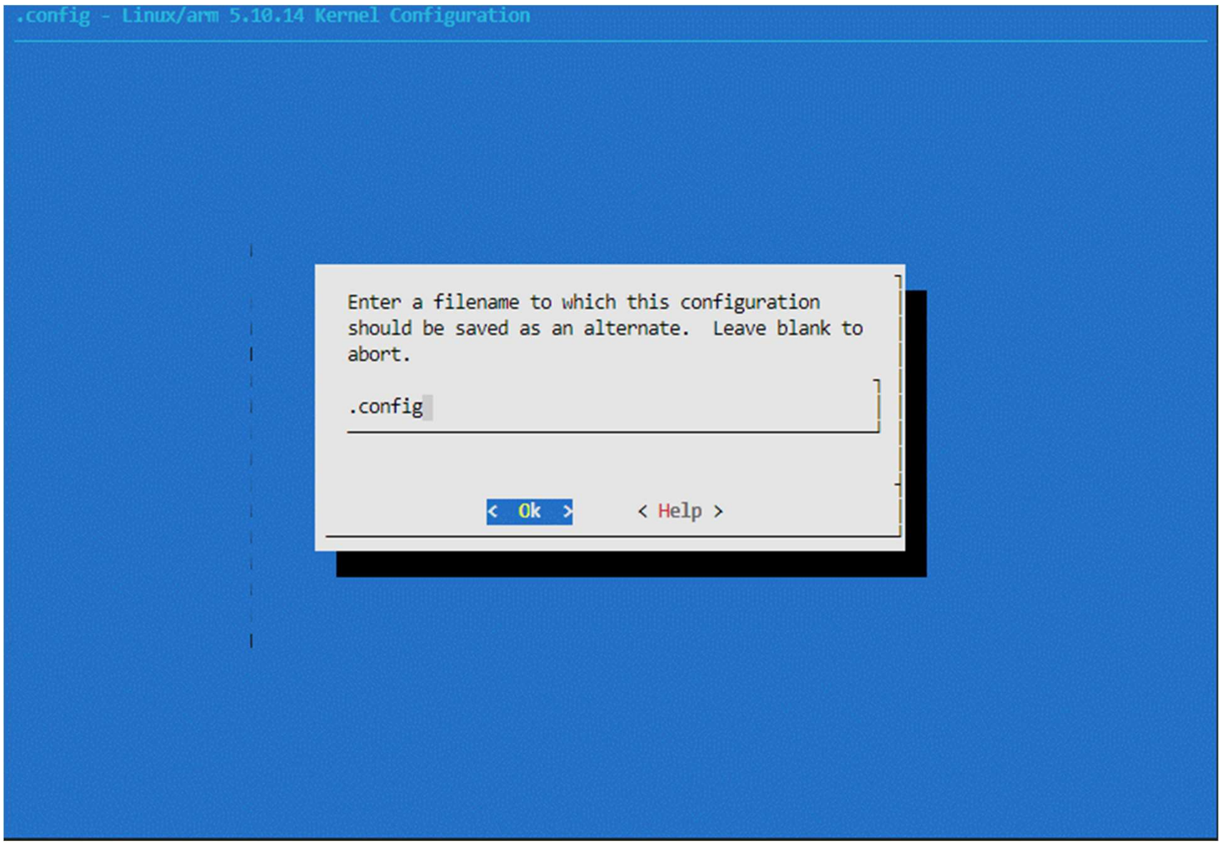
```
.config - Linux/arm 5.10.14 Kernel Configuration
> Device Drivers ──────────────────────────────────────────────────────────────
                              Device Drivers
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted
  letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc>
  to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module
  capable
 ┌────────────────^(-)──────────────────────────────────────────────────────────────────────────┐
 │                [*] HDMI CEC RC integration
 │                [*] HDMI CEC drivers  --->
 │                <M> Multimedia support  --->
 │                    Graphics support  --->
 │                <*> Sound card support  --->
 │                    HID support  --->
 │                [*] USB support  --->
 │                <*> 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  --->
 │                <M> Userspace I/O drivers  --->
 │                [ ] Virtualization drivers  ----
 └                v(+)

                      <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 22



```
.config - Linux/arm 5.10.14 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
                  ^(-)
                  < >   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
                  < >   Cadence RTC driver
                  < >   Faraday Technology FTRTC010 RTC
                  < >   EPSON TOYOCOM RTC-7301SF/DG
                        *** HID Sensor RTC drivers ***
                  <*>   Maxim MAX31328

                <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 23

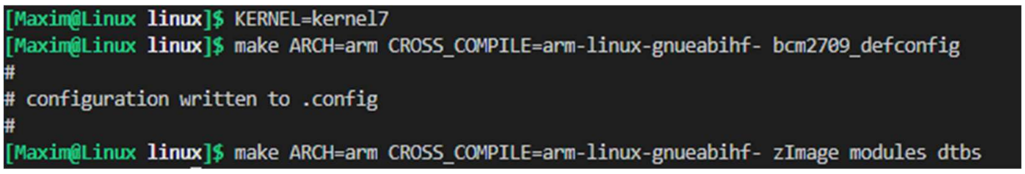


```
.config - Linux/arm 5.10.14 Kernel Configuration

Enter a filename to which this configuration
should be saved as an alternate.  Leave blank to
abort.

.config
```

Figure 24

Compile the kernel with these commands after all these steps:

- KERNEL=kernel7
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs



```
[Maxim@Linux linux]$ KERNEL=kernel7
[Maxim@Linux linux]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
#
# configuration written to .config
#
[Maxim@Linux linux]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
```

Figure 25

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	02/21	Initial release	—