RTC overview

Contents

Stable: 14.09.2021 - 14:27 / Revision: 14.09.2021 - 14:25

Stable: 14.09.2021 - 14:27 / Revision: 14.09.2021 - 14:28

A quality version of this page, approved on *14 September 2021*, was based off this revision.

This article gives information about the Linux®RTC framework. The RTC framework is involved in precise time countdown.

Contents

1 Framework purpose

The RTC keeps the system time up to date and can wake up the system from the standby power mode at a programmed time.

A general presentation of the RTC framework is available in the Linux RTC documentation [1].

2 System overview

RTC framework overview

2.1 Component description

- **RTC (hardware)**
  RTC dedicated hardware block in STM32MPU product.
- **rtc-stm32**
  RTC ST driver.
- **rtc**
  Linux RTC framework, it provides API to RTC driver and interfaces to user.
- **Sysfs interface**
  Sysfs interface is accessible via /sys/class/rtc/rtcX/.
- **Procfs interface**
  Procfs interface is accessible via /proc/driver/rtc.
- **Char device interface**
  Device interface is accessible via /dev/rtcX.
- **User application**
  The user application can be an user built application or an community application like hwclock.

2.2 API description

API is described in Linux RTC documentation [1].

3 Configuration

3.1 Kernel Configuration

Activate **rtc-stm32** driver in kernel configuration using the Linux Menuconfig tool: Menuconfig or how to configure kernel

```
Device drivers  --->
    -*- Real Time Clock  --->
        <*> STM32 RTC
```

3.2 Device tree configuration

Please refer to the RTC device tree configuration.

4 How to use the framework

Please refer to the How to use the RTC.

5 How to trace and debug the framework

5.1 How to trace

Dynamic debug traces can be added using the following commands:

```
echo -n 'file rtc-stm32.c +p'>/sys/kernel/debug/dynamic_debug/control
```

6 Source code location

- rtc-stm32: drivers/rtc/rtc-stm32.c
- api: include/linux/rtc.h
- framework:
  - drivers/rtc/class.c
  - drivers/rtc/dev.c
  - drivers/rtc/interface.c
  - drivers/rtc/lib.c
  - drivers/rtc/proc.c
  - drivers/rtc/sysfs.c
- device-tree bindings constants: include/dt-bindings/rtc/rtc-stm32.h

7 References

- 1.01.1 Linux RTC documentation

Linux® is a registered trademark of Linus Torvalds.

Real Time Clock

Application programming interface

Stable: 09.09.2021 - 14:00 / Revision: 07.09.2021 - 14:49

A quality version of this page, approved on *9 September 2021*, was based off this revision.

Contents

1 Purpose

This article describes how to use the RTC.

2 How to set a hardware clock using the hwclock tool

The correct system time must first be set with the date command. Type the date command to see the date and time format of the string:

```
Board $> date
Fri Mar  9 12:18:51 UTC 2018
```

```
Board $> date --set="Fri Mar  9 19:18:51 UTC 2018"
Fri Mar  9 19:18:51 UTC 2018
```

Set the hardware clock to this date:

```
Board $> hwclock
Fri Mar  9 12:21:03 2018  0.000000 seconds

Board $> hwclock --systohc --utc

Board $> hwclock
Fri Mar  9 19:19:52 2018  0.000000 seconds
```

See [1] for more about date settings. See [2] for more about hwclock settings.

3 How to set an alarm

```
# disable the alarm
Board $> echo 0 > /sys/class/rtc/rtc0/wakealarm
```

```
# calculate alarm with 1 minute later
Board $> wakeuptime=`date -d "1 minute" +%s`
```

```
# set the alarm
Board $> echo $wakeuptime > /sys/class/rtc/rtc0/wakealarm
root@stm32mp1:~# [  829.279019] rtc rtc0: Alarm occurred
```

You can also set an alarm 'n' seconds later:

```
Board $> echo +10 > /sys/class/rtc/rtc0/wakealarm
```

4 How to set an alarm and go into a system sleep state with the rtcwake tool

```
# calculate an alarm 1 minute later
Board $> wakeuptime=`date -d "1 minute" +%s`
```

```
# set wakeup on /dev/rtc0
Board $> rtcwake -lt$wakeuptime -m mem
rtcwake: wakeup from "mem" using /dev/rtc0 at Sun Mar 11 10:48:06 2018
[  154.022303] PM: suspend entry (deep)
[  154.024421] PM: Syncing filesystems ... done.
[  154.037258] Freezing user space processes ... (elapsed 0.001 seconds) done.
[  154.044397] OOM killer disabled.
[  154.047555] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[  154.055039] Suspending console(s) (use no_console_suspend to debug)
```

One minute later...

```
NOTICE:  CPU: STM32MP157CAA Rev.B
NOTICE:  Model: STMicroelectronics STM32MP157C eval daughter on eval mother
NOTICE:  Board: MB1263 Var1 Rev.C-01
INFO:    Reset reason (0x810):
INFO:    System exits from STANDBY
INFO:    Using SDMMC
INFO:      Instance 1
INFO:    Boot used partition fsbl1
INFO:    Product_below_2v5=1: HSLVEN update is
INFO:      destructive, no update as VDD>2.7V
NOTICE:  BL2: v2.0(debug):v2.0-stm32mp-19-01-29
NOTICE:  BL2: Built : 15:58:42, Jan 29 2019
INFO:    BL2: Doing platform setup
INFO:    PMIC version = 0x10
INFO:    RAM: DDR3-1066/888 bin G 2x4Gb 533MHz v1.41
INFO:    BL2 runs SP_MIN setup
INFO:    BL2: Loading image id 4
INFO:    Loading image id=4 at address 0x2ffe5000
INFO:    Image id=4 loaded: 0x2ffe5000 - 0x30000000
INFO:    BL2: Skip loading image id 5
INFO:    read version 0 current version 0
NOTICE:  BL2: Booting BL32
INFO:    Entry point address = 0x2ffe5000
INFO:    SPSR = 0x1d3
INFO:    PMIC version = 0x10
NOTICE:  SP_MIN: v2.0(debug):v2.0-stm32mp-19-01-29
NOTICE:  SP_MIN: Built : 15:58:42, Jan 29 2019
INFO:    ARM GICv2 driver initialized
INFO:    stm32mp HSI (18): Secure only
INFO:    stm32mp HSE (20): Secure only
INFO:    stm32mp PLL2 (27): Secure only
INFO:    stm32mp PLL2_R (30): Secure only
INFO:    SP_MIN: Initializing runtime services
INFO:    SP_MIN: Preparing exit to normal world
[  154.074319] dwc2 49000000.usb-otg: suspending usb gadget configfs-gadget
[  154.196804] Disabling non-boot CPUs ...
[  154.250046] CPU1 killed.
[  154.251704] Enabling non-boot CPUs ...
```

```
[  154.252697] CPU1 is up
[  154.254885] rtc rtc0: Alarm occurred
[  154.273940] dwmac4: Master AXI performs any burst length
[  154.273972] stm32-dwmac 5800a000.ethernet eth0: No Safety Features support found
[  154.276938] usb usb2: root hub lost power or was reset
[  154.284703] dwc2 49000000.usb-otg: resuming usb gadget configfs-gadget
[  154.491151] dwc2 49000000.usb-otg: new device is high-speed
[  154.619589] dwc2 49000000.usb-otg: new device is high-speed
[  154.669343] usb 2-1: reset high-speed USB device number 2 using ehci-platform
[  154.697272] dwc2 49000000.usb-otg: new address 2
[  154.722774] configfs-gadget gadget: high-speed config #1: c
[  155.066198] OOM killer enabled.
[  155.069339] Restarting tasks ... done.
[  155.075470] PM: suspend exit
```

See [3] for more about rtcwake settings.

5 How to get RTC status

```
Board $> cat /proc/driver/rtc
rtc_time        : 07:25:13
rtc_date        : 2000-01-01
alrm_time       : 00:00:00
alrm_date       : 2165-01-01
alarm_IRQ       : no
alrm_pending    : no
update IRQ enabled      : no
periodic IRQ enabled    : no
periodic IRQ frequency  : 1
max user IRQ frequency  : 64
24hr            : yes
```

6 References

- http://man7.org/linux/man-pages/man1/date.1.html
- http://man7.org/linux/man-pages/man8/hwclock.8.html
- http://man7.org/linux/man-pages/man8/rtcwake.8.html

Real Time Clock

Out Of Memory

Central processing unit

Boot Loader stage 2

Power Management Integrated Circuit

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Boot Loader stage 3-2

High Speed Internal oscillator (STM32 clock source) or High Speed Synchronous Serial Interface (MIPI® Alliance standard)

High Speed External oscillator (STM32 clock source)

Configuration File System (See https://en.wikipedia.org/wiki/Configfs for more details)

Stable: 31.03.2021 - 08:47 / Revision: 26.03.2021 - 08:44

A quality version of this page, approved on *31 March 2021*, was based off this revision.

Contents

1 Linux configuration genericity

The process of building a kernel has two parts: configuring the kernel options and building the source with those options.

The Linux® kernel configuration is found in the generated file: .config.

.config is the result of configuring task which is processing platform defconfig and fragment files if any.

For OpenSTLinux distribution the defconfig is located into the kernel source code and fragments into stm32mp BSP layer :

- arch/arm/configs/**multi_v7_defconfig**

Every new kernel version brings a bunch of new options, we do not want to back port them into a specific defconfig file each time the kernel releases, so we use the same defconfig file based on ARM SoC v7 architecture.
STM32MP1 specificities are managed with fragments config files.

- meta-st/meta-st-stm32mp/recipes-kernel/linux/linux-stm32mp/<kernel version>/**fragment-*.config**

.config result is located in the build folder:

- build-openstlinuxweston-stm32mp1/tmp-glibc/work/stm32mp1-ostl-linux-gnueabi/linux-stm32mp/5.10.10-rc0/build/**.config**

To modify the kernel options, it is not recommended to edit this file directly.

- A user runs either a text-mode :

```
   PC $> make config
starts a character based question and answer session (Figure 1)
```

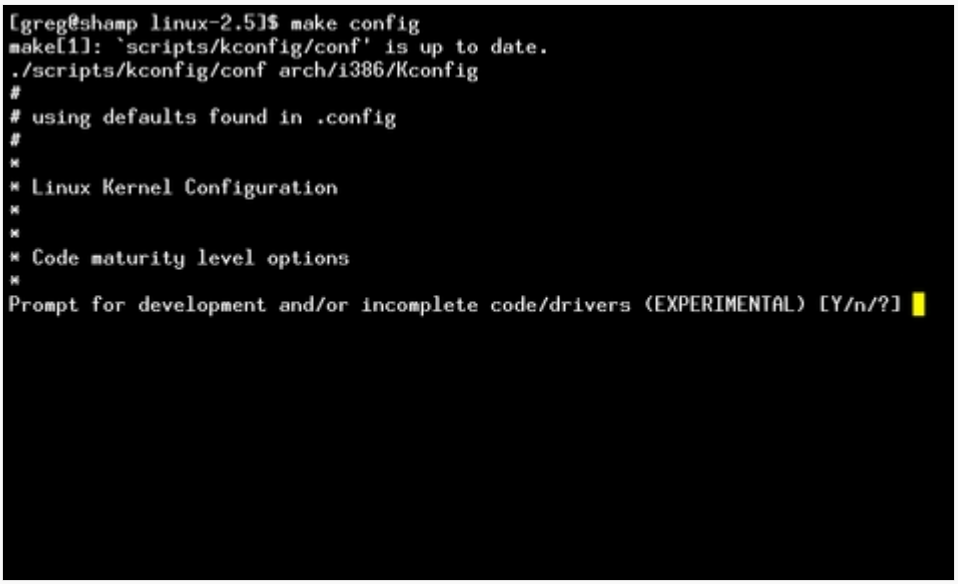

Figure 1. Configuring the kernel with make config

```
   PC $> make
menuconfig
starts a terminal-
oriented
configuration tool
(using ncurses)
(Figure 2)
The ncurses text
version is more
popular and is run
with the make
menuconfig option.
Wikipedia Menuconfig[1
] also explains how
to "navigate" within
the configuration
menu, and highlights
main key strokes.
```

- or a graphical kernel

configurator :

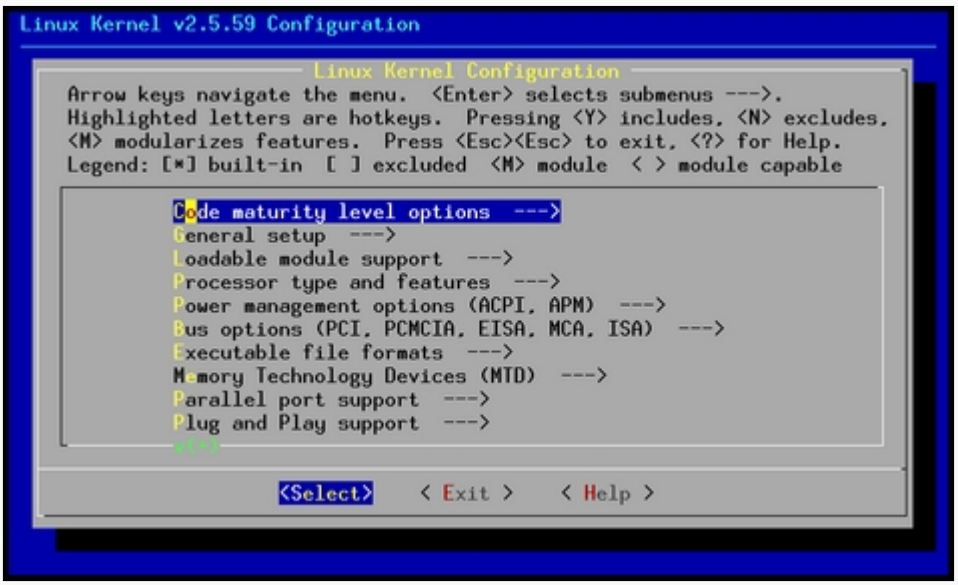

Figure 2. Make menuconfig makes it easier to back up and correct mistakes

PC $> make xconfig starts a X based configuration tool (Figure 3)

Ultimately these configuration tools edit the .config file.

An option indicates either some driver is built into the kernel ("=y") or will be built as a module ("=m") or is not selected.

The unselected state can either be indicated by a line starting with "#" (e.g. "# CONFIG_SCSI is not set") or by the absence of the relevant line from the .config file.

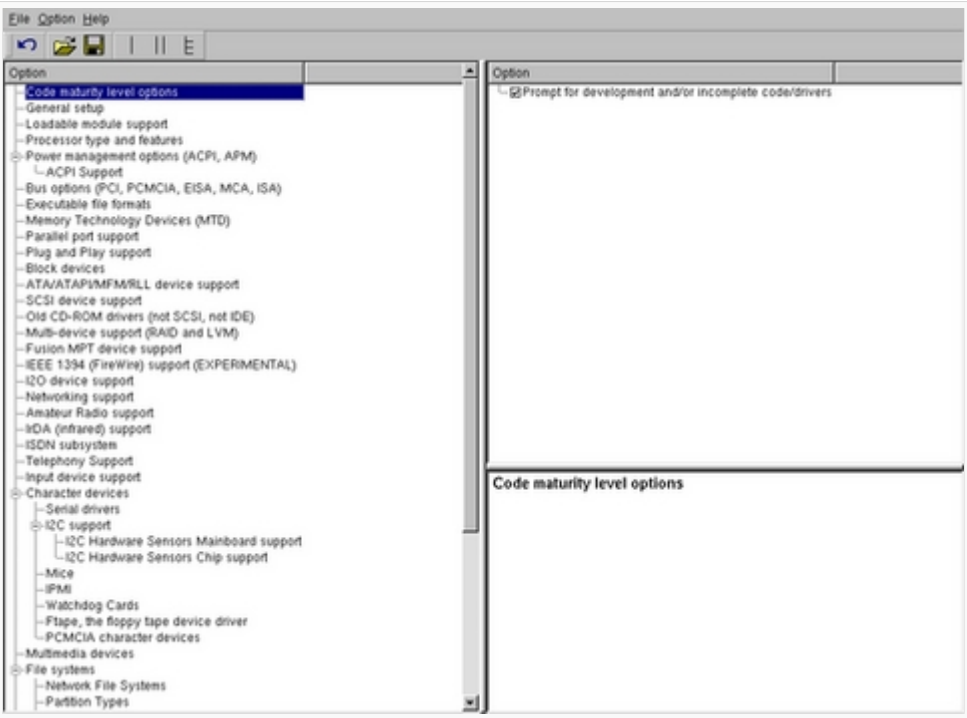

Figure 3. The Qt-Based make xconfig

The 3 states of the main selection option for the SCSI subsystem (which actually selects the SCSI mid level driver) follow. Only one of these should appear in an actual .config file:

```
CONFIG_SCSI=y
CONFIG_SCSI=m
# CONFIG_SCSI is not
set
```

2 Menuconfig and Developer Package

For this use case, the prerequesite is that OpenSTLinux SDK has been installed and configured.

To verify if your cross-compilation environment has been put in place correctly, run the following command:

```
PC $> set | grep CROSS
CROSS_COMPILE=arm-ostl-linux-gnueabi-
```

For more details, refer to *<Linux kernel installation directory>/README.HOW_TO.txt* helper file (the latest version of this helper file is also available in GitHub: README.HOW_TO.txt).

- Go to the <Linux kernel build directory>

```
PC $> cd <Linux kernel build directory>
```

- Save initial configuration (to identify later configuration updates)

```
PC $> make arch=ARM savedefconfig
Result is stored in defconfig file
PC $> cp defconfig defconfig.old
```

- Start the Linux kernel configuration menu

```
PC $> make arch=ARM menuconfig
```

- Navigate forwards or backwards directly between feature
  - un/select, modify feature(s) you want
  - When the configuration is OK : exit and save the new configuration

```
useful keys to know:
enter: enter in config subdirectory
space: hit several times to either select [*], select in module [m] or unselect [ ]
/: to search for a keyword, this is usefull to navigate in tree
?: to have more information on selected line
```

- Compare the old and new config files after operating modifications with menuconfig

```
PC $> make arch=ARM savedefconfig
```

Retrieve configuration updates by comparing the new defconfig and the old one

```
PC $> meld defconfig defconfig.old
```

- Cross-compile the Linux kernel (please check the load address in the *README.HOW_TO.txt* helper file)

```
PC $> make arch=ARM uImage LOADADDR=<loadaddr of kernel>
PC $> cp arch/arm/boot/uImage install_artifact/boot/
```

- Update the Linux kernel image on board

```
PC $> scp install_artifact/boot/uImage root@<board ip address>:/boot/
```

If the */boot* mounting point doesn't exist yet, please see how to create a mounting point

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

Note that this use case modifies the configuration file in the Linux kernel build directory, not in the Linux kernel source directory: this is a temporary modification useful for a prototyping.

- To make this temporary modification permanent, the delta between defconfig and defconfig.old must be saved in a configuration fragment file (fragment-*.config) based on fragment.cfg file, and the Linux kernel configuration/compilation steps must be re-executed (as explained in the README.HOW_TO.txt helper file).

3 Menuconfig and Distribution Package

- Start the Linux kernel configuration menu

```
PC $> bitbake virtual/kernel -c menuconfig
```

- Navigate forwards or backwards directly between feature
  - un/select, modify feature(s) you want
  - When the configuration is OK : exit and save the new configuration

```
useful keys to know:
enter: enter in config subdirectory
space: hit several times to either select [*], select in module [m] or unselect [ ]
/: to search for a keyword, this is usefull to navigate in tree
?: to have more information on selected line
```

- Cross-compile the Linux kernel

```
PC $> bitbake virtual/kernel
```

- Update the Linux kernel image on board

```
PC $> scp <build dir>/tmp-glibc/deploy/images/<machine name>/uImage root@<board ip
address>:/boot
```

If the */boot* mounting point does not exist yet, please see how to create a mounting point

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

Note that this use case modifies the configuration file in the Linux kernel build directory, not in the Linux kernel source directory: this is a temporary modification useful for a prototyping.

- To make this temporary modification permanent, it must be saved in a configuration fragment file (fragment-*.config) based on **fragment.cfg** file, and the Linux kernel configuration/compilation steps must be re-executed: **bitbake <name of kernel recipe>**.

4 References

- Wikipedia Menuconfig

Linux® is a registered trademark of Linus Torvalds.

Board support package

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

Stable: 09.09.2021 - 14:07 / Revision: 09.09.2021 - 14:07

A quality version of this page, approved on *9 September 2021*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the **RTC** internal peripheral when it is assigned to the Linux®OS. In this case, it is controlled by the **RTC framework**.

The configuration is performed using the device tree mechanism that provides a hardware description of the RTC peripheral used by the STM32 RTC Linux driver.

2 DT bindings documentation

The **RTC** is represented by the *STM32 RTC device tree bindings*[1]

3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

3.1 DT configuration (STM32 level)

The **RTC** node is declared in the file *stm32mp151.dtsi* [2]. It describes the hardware register address, clocks and interrupts.

```
rtc: rtc@5c004000 {
        compatible = "st,stm32mp1-rtc";
        reg = <0x5c004000 0x400>;                                --> Register location and
length
        clocks = <&scmi0_clk CK_SCMI0_RTCAPB>,
                 <&scmi0_clk CK_SCMI0_RTC>;
        clock-names = "pclk", "rtc_ck";
        interrupts-extended = <&exti 19 IRQ_TYPE_LEVEL_HIGH>;
        status = "disabled";
};
```

This device tree part is related to STM32 microprocessors. It should be kept as is, without being modified by the end-user.

3.2 DT configuration (board level)

This part is used to enable the **RTC** used on a board, which is done by setting the **status** property to **okay**.

An "st,lsco" property is available to select and enable the RTC output on which RTC low-speed clock is output. The valid ouput values are defined in [3]. A pinctrl state named "default" can be defined to reserve a pin for the RTC output.

3.3 DT configuration examples

```
#include <dt-bindings/rtc/rtc-stm32.h>
...
&rtc {
        st,lsco = <RTC_OUT2_RMP>;
        pinctrl-0 = <&rtc_out2_rmp_pins_a>;
        pinctrl-names = "default";
};
```

4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
STM32CubeMX might not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

5 References

Please refer to the following links for additional information:

- Device tree bindings
- STM32MP151 device tree
- STM32 RTC bindings constants

Linux® is a registered trademark of Linus Torvalds.

Operating System

Real Time Clock

Device Tree