Power overview

Contents

Stable: 01.12.2020 - 10:35 / Revision: 01.12.2020 - 09:20

Stable: 01.12.2020 - 10:30 / Revision: 01.12.2020 - 09:20

A quality version of this page, approved on *1 December 2020*, was based off this revision.

Contents

1 Framework purpose

The purpose of this article is to explain how to handle the STM32MP15x low-power modes:

- Low-power modes available on the device
- Linux software overview
- How to enter and exit the low-power modes on Arm® Cortex®-A7 core
- How to enter a platform low-power mode

2 Low-power modes available on the device

Refer to STM32MP15 reference manuals for the full description of low-power modes.
The AN5109 low-power application note also gives much more information on these modes, including:

- the detailed description of the operating modes,
- the low-power mode entry and exit sequences,
- the low-power mode control registers.

The modes are handled by the RCC and the PWR peripherals.

The table below summarizes the device hardware states corresponding to each low-power mode.

The term "subsystem" either refers to Arm® Cortex®-A7 (also called MPU) or to Arm® Cortex®-M4 (also called MCU). A mode prefixed by 'C' corresponds to a subsystem mode.

A platform mode is the combination of MPU and MCU modes.

Level	Mode	Vddcore state	Clocks state
Subsystem	MPU CRun	on	on
	MPU CStop	on	Subsystem off
	MPU CStandby	on	Subsystem off
	MCU CRun	on	on
	MCU CStop	on	Subsystem off

MPU mode	MCU mode	Platform mode	Vddcore state	Clocks state
CRun	CRun	Run	On	On
CStop	CRun	Run	On	On
CStandby	CRun	Run	On	On
CRun	CStop	Run	On	On
CStop	CStop	Stop/LPLV-Stop/Standby	On/Retention/Off	Off/Off/Off
CStandby	CStop	Stop/LPLV-Stop/Standby	On/Retention/Off	Off/Off/Off

2.1 Wakeup sources

The above modes are exited due to a wakeup event.

Again, the AN5109 low-power application note details, among other things, the wakeup sources, the software mechanism that ensures the consistency between the low-power mode and the activated wakeup source, and the low-power mode exit sequence.

The following table gives the list of wakeup sources available in each mode.

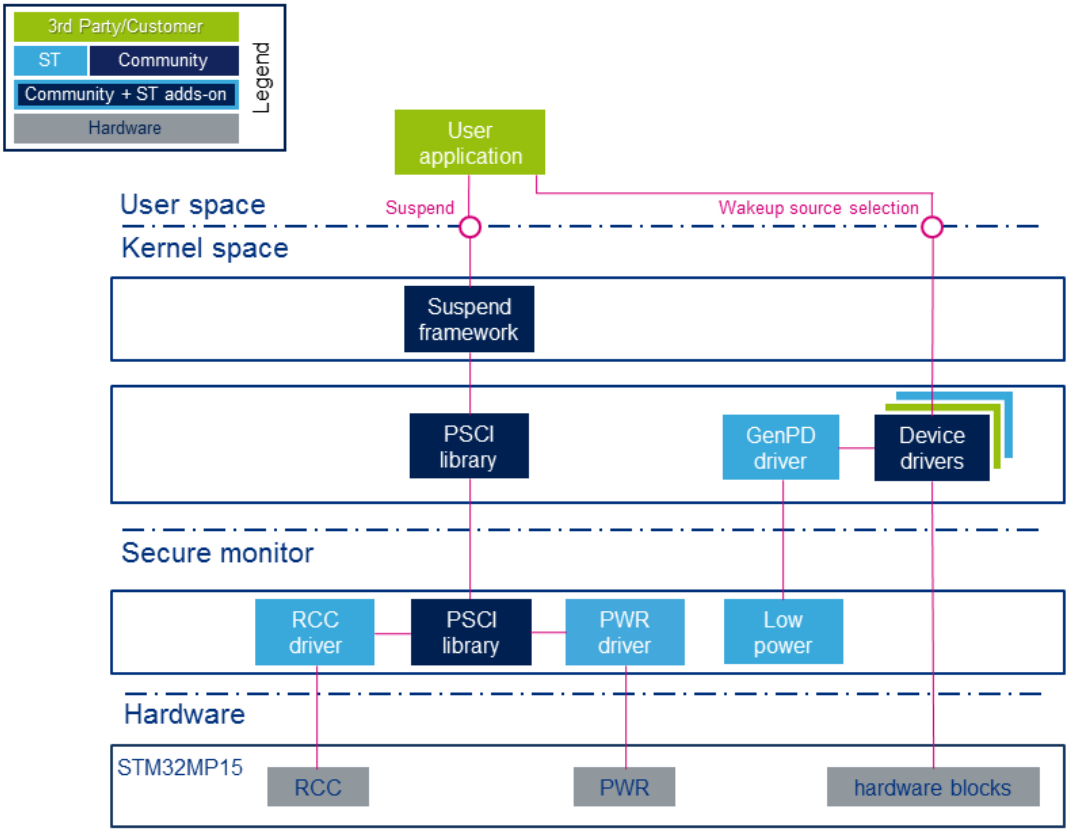
Mode	Available wakeup sources
CStop	BOR, PVD, AVD, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, USB, CEC, ETH, USA

Mode	Available wakeup sources
/CStandby /Stop	RT, I²C, SPI, LPTIM, IWDG, GPIO, Wakeup pins (from PWR)
LPLV-Stop	BOR, PVD, AVD, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, IWDG, GPIO, Wakeup pins (from PWR)
Standby	BOR, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, IWDG, Wakeup pins (from PWR)

3 Software overview

The Linux® suspend framework is used to trigger a low-power mode entry/exit sequence.

Refer to Documentation/power for more details.



The user application issues a suspend request to the kernel. This request is handled by the suspend Framework, which notifies all the device drivers to prepare for low-power entry. It then calls the PSCI service.

In addition to this centralized suspend process, most of the drivers implement the runtime pm feature. It is used to dynamically disable the resources of the peripherals (clocks and power when applicable) in case of inactivity (see Documentation/power /runtime_pm.rst).

3.1 Component description

Kernel components:

- **Suspend framework**: this framework schedules the overall sequence by stopping all the ongoing tasks
- **GenPD driver**: this driver is used for low-power mode selection according to the activated wakeup sources.
- **PSCI library**: this is a set of standardized functions to request a low-power service to the secure monitor
- **RCC driver**: this driver handles the circuit non-secure clocks

Secure monitor components:

- **PWR driver**: this driver is responsible for configuring the low-power mode
- **PSCI library**: this is a set of standardized functions handling the low-power services
- **Low power driver**: the role of this driver is to choose the low-power mode according to the programmed wakeup source(s)
- **RCC driver**: this driver handles the circuit secure clocks

3.2 API description

The suspend process is triggered from the user space through standard commands.

The system sleep control file is the *state* file, located under: /sys/power/

Only the 'mem' command is supported:

- The whole system activity is stopped and a low-power mode is entered. The software selects the deepest mode according to the activated wakeup source(s).

```
Example: Board $> echo mem > /sys/power/state
```

Further details can be found in Documentation/power/interface.rst

STMicroelectronics deliveries propose a default mapping of the low-power modes for each type of board.

Note that this default mapping can be changed thanks to the device tree. Refer to paragraph 3.3.2.

3.3 Software configuration

3.3.1 Menuconfig (Linux® kernel)

The suspend to RAM feature is activated by default in STMicroelectronics deliveries.

It can be deactivated through the kernel menuconfig using Power management options/Suspend to RAM and standby: Menuconfig or how to configure kernel .

3.3.2 Device tree (secure monitor)

The default system low-power mode mapping can be modified through the secure monitor device tree.

Below an example:

```
&pwr {
    system_suspend_supported_modes = <
        STM32_PM_CSLEEP_RUN
        STM32_PM_CSTOP_ALLOW_STOP
```

```
            STM32_PM_CSTOP_ALLOW_LP_STOP
            STM32_PM_CSTOP_ALLOW_LPLV_STOP
            STM32_PM_CSTOP_ALLOW_STANDBY_DDR_SR
        >;
        system_off_soc_mode = <STM32_PM_CSTOP_ALLOW_STANDBY_DDR_OFF>;
    };
```

For detailed information on the device tree concept, refer to Device tree.

3.3.3 Example of wakeup source activation

The activation of a wakeup source is done in the corresponding driver.

For example, activating UART4 as wakeup source is done thanks to the following commands:

```
Board $> echo enabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
Board $> echo enabled > /sys/devices/platform/soc/40010000.serial/power/wakeup
```

It is possible to check the state of each wakeup source (activated or not) by displaying the 'wakeup' attribute.

Note that the software implements a consistency check between the selected wakeup source and the appropriate low-power mode.

4 How to enter and exit low-power modes

4.1 Platform low-power

Select the platform allowed modes depending on the required wakeup source.

Activate the wakeup source(s) (peripheral dependent).

Call the low-power mode on both sides (MPU and MCU).

4.2 MPU side

Activate the wakeup source(s) (peripheral dependent)

Call the low-power mode by issuing the following command:

echo mem > /sys/power/state

```
  Note that in Weston configuration the low-power mode is entered upon a 'systemctl
suspend' command.
```

4.3 MCU side

Please refer to Coprocessor power management for Arm® Cortex®-M4 commands.

4.4 Example: entering/exiting MPU CStop mode

Enable at least one wakeup source from table 2.1 in CStop category, for example USART:

```
Board $> echo enabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
Board $> echo enabled > /sys/devices/platform/soc/40010000.serial/power/wakeup
```

Call the low-power entry:

```
Board $> echo mem > /sys/power/state
```

or for the Weston configuration:

```
Board $> cat /etc/systemd/sleep.conf
[Sleep]
SuspendMode=
HibernateMode=
HybridSleepMode=
SuspendState=mem
HibernateState=mem
HybridSleepState=mem
```

```
 Board $> systemctl suspend
```

The MPU is now in CStop mode, and can be woken up by sending a character to the console.

5 How to trace and debug

The suspend/resume process execution is logged in the MPU console. It gives useful information on the platform state (sleeping or active).

```
 root@stm32mp1:~# echo mem > /sys/power/state
 [ 1072.267571] PM: suspend entry (deep)
 [ 1072.269687] PM: Syncing filesystems ... done.
 [ 1072.279114] Freezing user space processes ... (elapsed 0.008 seconds) done.
 [ 1072.292835] OOM killer disabled.
 [ 1072.296046] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
 [ 1072.303431] Suspending console(s) (use no_console_suspend to debug)
 [ 1072.332520] dwc2 49000000.usb-otg: suspending usb gadget configfs-gadget
 [ 1072.332537] dwc2 49000000.usb-otg: dwc2_hsotg_ep_disable: called for ep0
 [ 1072.332546] dwc2 49000000.usb-otg: dwc2_hsotg_ep_disable: called for ep0
 [ 1072.468536] Disabling non-boot CPUs ...
 [ 1072.507876] CPU1 killed.
 [ 1072.509635] Enabling non-boot CPUs ...
 [ 1072.510508] CPU1 is up
 [ 1072.527553] dwmac4: Master AXI performs any burst length
 [ 1072.527583] stm32-dwmac 5800a000.ethernet eth0: No Safety Features support found
 [ 1072.527621] stm32-dwmac 5800a000.ethernet eth0: ERROR failed to create debugfs
directory
 [ 1072.527631] stm32-dwmac 5800a000.ethernet eth0: stmmac_hw_setup: failed debugFS
registration
 [ 1072.588234] dwc2 49000000.usb-otg: resuming usb gadget configfs-gadget
 [ 1072.738469] OOM killer enabled.
 [ 1072.741575] Restarting tasks ... done.
 [ 1072.752596] PM: suspend exit
```

It is also possible to monitor the hardware signals related to the system low-power modes thanks to the HDP internal peripheral. Please refer to HDP Linux driver for its configuration.

6 To go further

Refer to STM32MP15 reference manuals for a detailed description of low-power modes and peripheral wakeup sources.

The AN5109 low power application note gives additional information on the hardware settings used for low-power management.

Universal Synchronous/Asynchronous Receiver/Transmitter
Stable: 01.12.2020 - 10:35 / Revision: 01.12.2020 - 09:11

A quality version of this page, approved on *1 December 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to give an overview of the software APIs available on the Arm® Cortex®-M4 (also named MCU) side to handle the low power modes.

2 Low power modes available on the chip

Refer to STM32MP15 reference manuals for the full description of the modes.
The AN5109 low power application note also gives details on these modes.

The modes are handled by RCC and PWR peripherals.

The table below explains the chip hardware states corresponding to each low power mode.

Subsystem either refers to Arm® Cortex®-A7 side (also called MPU) or Arm® Cortex®-M4 side (also called MCU). A mode prefixed by 'C' refers to a subsystem mode.

A platform mode is the combination of MPU and MCU modes.

Level	Mode	Vddcore state	Clocks state
Subsystem	MPU CRun	on	on
	MPU CStop	on	Subsystem off
	MPU CStandby	on	Subsystem off
	MCU CRun	**on**	**on**
	MCU CStop	**on**	**Subsystem off**

MPU mode	MCU mode	Platform mode	Vddcore state	Clocks state
CRun	CRun	Run	On	On
CStop	CRun	Run	On	On
CStandby	CRun	Run	On	On
CRun	CStop	Run	On	On
CStop	CStop	Stop/LPLV-Stop/Standby	On/Retention/Off	Off/Off/Off
CStandby	CStop	Stop/LPLV-Stop/Standby	On/Retention/Off	Off/Off/Off

2.1 Wakeup sources

The above modes are left due to a wakeup event. It can be configured by setting the wakeup control feature of the IP and activating the corresponding **EXTI** on MCU side.

The following table gives the list of wakeup sources available in each mode.

Mode	Available wakeup sources
CStop /CStandby /Stop	BOR, PVD, AVD, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, USB, CEC, ETH, USART, I²C, SPI, LPTIM, IWDG, GPIO, Wakeup pins
LPLV-Stop	BOR, PVD, AVD, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, IWDG, GPIO, Wakeup pins

Mode	Available wakeup sources
Standby	BOR, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, IWDG, Wakeup pins

3 Software overview

The power HAL is used to select the Cortex-M4 low power mode. The MPU uses the Linux® RPMsg framework to communicate with the MCU.

Further information on HAL can be found here: STM32CubeMP1 architecture

3.1 APIs description

The power HAL supports the following APIs related to power management:

```
HAL_PWR_EnterSLEEPMode: CSleep mode is entered
```

```
HAL_PWR_EnterStopMode: CStop mode is entered allowing Stop as the deepest platform low power mode
```

```
HAL_PWR_EnterStandbyMode: CStop mode is entered allowing Standby as the deepest platform low power mode
```

3.2 Code source location

STM32CubeMP1 Package provides power HAL driver:
Drivers/STM32MP1xx_HAL_Driver/Src/stm32mp1xx_hal_pwr.c

Universal Synchronous/Asynchronous Receiver/Transmitter
Stable: 05.11.2021 - 11:08 / Revision: 05.11.2021 - 11:05

A quality version of this page, approved on *5 November 2021*, was based off this revision.

Contents

1 Purpose

The objective of this chapter is to give general information about the device tree.

An extract of the **device tree specification**[1] explains it as follows:

"A device tree is a tree data structure with nodes that describe the devices in a system. Each node has property/value pairs that describe the characteristics of the device being represented. Each node has exactly one parent except for the root node, which has no parent. ... Rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time."

In other words, a device tree describes the hardware that can not be located by probing.

1.1 Device tree basis

This webinar will give the foundations of device tree applied to STM32MP1 products and boards. This is highly recommended to start from this if you are beginner on this subject.

- Device Tree for STM32MP [2]

1.2 Source files

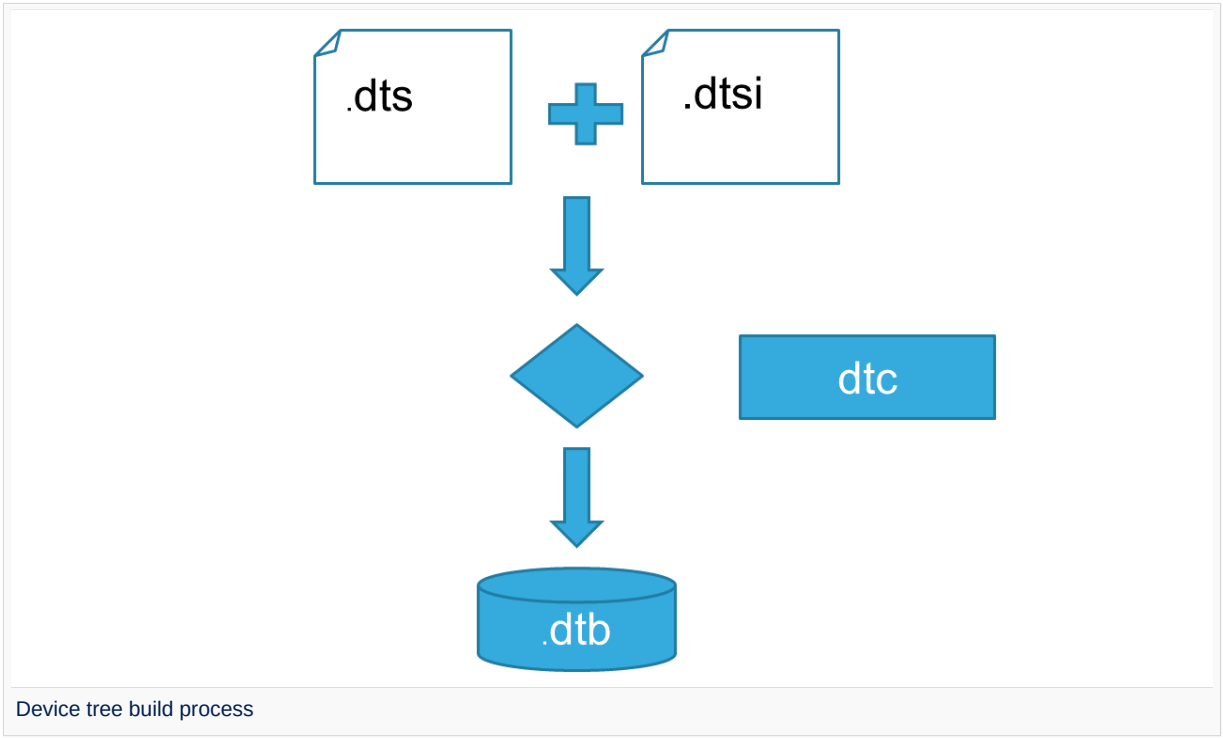
- **.dts:** The device tree source (DTS). This format is a textual representation of a device tree in a form that can be processed by DTC (Device Tree Compiler) into a binary file expected by software components: Linux® Kernel, U-Boot and TF-A.
- **.dtsi:** Source files that can be included from a DTS file.
- **.h:** Header files that can be included from DTS and DTSI files.

1.3 Bindings

The device tree data structures and properties are named **bindings**. Those bindings are described in:

- The Device tree specification[1] for generic bindings.
- The software component documentations:
  - Linux® Kernel: Linux kernel device tree bindings
  - U-Boot: doc/device-tree-bindings/
  - TF-A: TF-A device tree bindings

1.4 Build



Device tree build process

- A tool named DTC[3](Device Tree Compiler) allows compiling the DTS sources into a binary.
  - input file: the *.dts* file described in section above (that includes itself one or several .dtsi and .h files).
  - output file: the *.dtb* file described in section above.

DTC source code is located here[4]. DTC tool is also available directly in particular software components: **Linux Kernel, U-Boot, TF-A ...**. For those components, the device tree building is directly integrated in the component build process.

Information

If dts files use some defines, dts files should be preprocessed before being compiled by DTC.

1.5 Tools

The device tree compiler offers also some tools:

- **fdtdump:** Print a readable version of a flattened device tree file (dtb)
- **fdtget:** Read properties from a device tree
- **fdtput:** Write properties to a device tree
- ...

There are several ways to get those tools:

- In the device tree compiler project source code[4]
- Directly in software components: **Kernel, u-boot, tf-a ...**
- Available in Debian package[5]

2 STM32

For STM32MP1, the device tree is used by three software components: Linux® kernel, U-Boot and TF-A.

The device tree is part of the OpenSTLinux distribution. It can also be generated by STM32CubeMX tool.

To have more information about the device tree usage on STM32MP1 (how the device tree source files are split, how to find the device tree source files per software components, how is STM32CubeMX generating the device tree ...) see STM32MP15 device tree page.

3 How to go further

- Device Tree Reference[6] - eLinux.org
- Device Tree usage[7] - eLinux.org

4 References

- 1.01.1 https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.2(latest) ,Device tree specification
- https://www.youtube.com/watch?v=a9CZ1Uk3OYQ, Device Tree for STM32MP
- https://git.kernel.org/pub/scm/utils/dtc/dtc.git/tree/Documentation/manual.txt(master), DTC manual
- 4.04.1 https://git.kernel.org/pub/scm/utils/dtc/dtc.git(master) ,DTC source code
- https://packages.debian.org/search?keywords=device-tree-compiler(master) ,DTC debian package
- Device Tree Reference, eLinux.org
- Device Tree Usage, eLinux.org

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 21.09.2021 - 14:25 / Revision: 21.09.2021 - 14:24

A quality version of this page, approved on *21 September 2021*, was based off this revision.

Contents

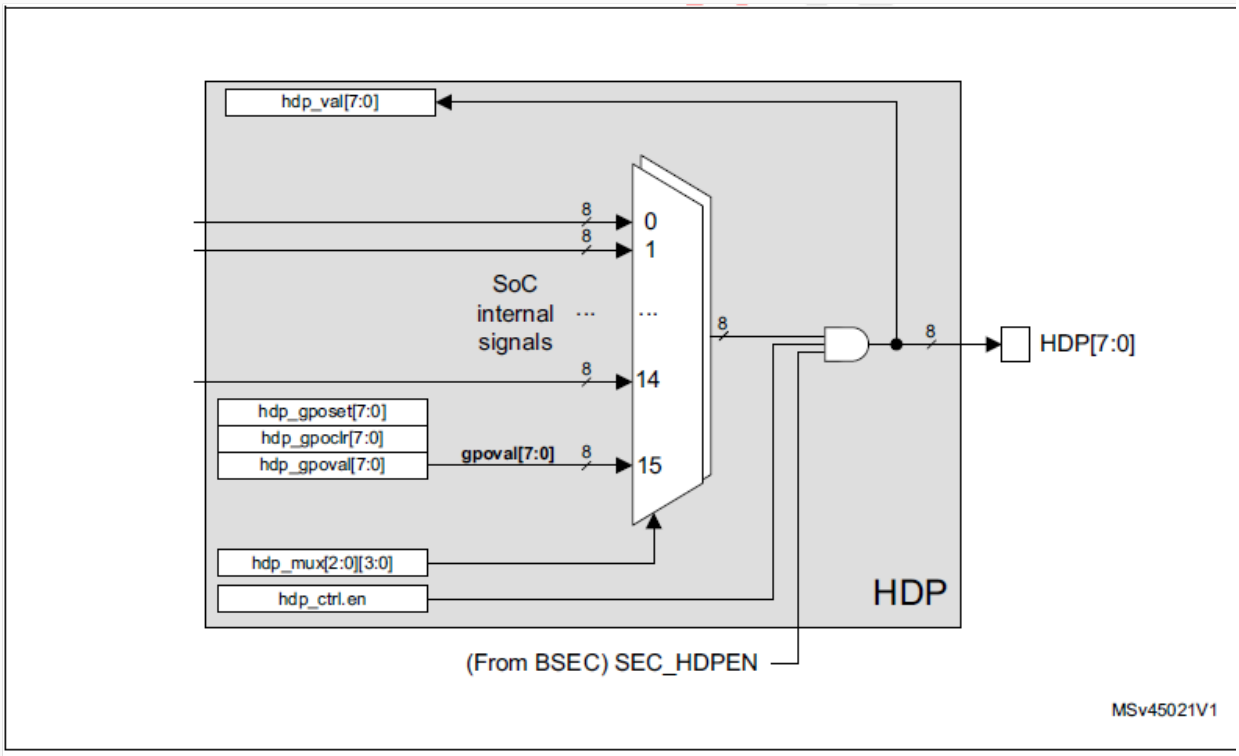
1 Article purpose

This article introduces the **Hardware Debug Port** which allows the observation of internal signals. By using multiplexers, up to 16 signals of each of 8-bit output can be observed. The article explains:

- How to configure, use and debug the driver
- The driver structure, and where the source code can be found.

2 Short description

- 8 output signals
- One of 16 internal signals with individual control
- 8 software-programmable signals for pinout agnostic code debugging
- Output disabling by security signal



HDP Block Diagram

3 Configuration

3.1 Kernel configuration

The **HDP** is enabled and ready to be used in all STM32MPU Embedded Software Distributions, via the Linux® kernel configuration **CONFIG_STM32_HDP**, set to disabled by default.

```
Symbol: STM32_HDP
Location:
  Device Drivers
  [*] SOC (System On Chip) specific Drivers
    [*] STMicroelectronics STM32MP157 Hardware Debug Port (HDP) pin control
```

Please refer to the Menuconfig or how to configure kernel article for instructions on modifying the configuration, and recompiling the Linux kernel image in the Distribution Package context.

3.2 Device tree

Refer to the HDP device tree configuration article when configuring the HDP Linux kernel driver.

4 How to trace and debug

4.1 How to monitor

4.1.1 How to monitor with debugfs

sysfs entry can be used to browse HDP registers.

```
Board $> /sys/kernel/debug/hdp# ls
 ctrl  gpoclr  gposet  gpoval  mux  val
```

See the HDP chapter in the reference manual [1] for further register details.

5 Source code location

The HDP Linux driver source code is composed of:

- drivers/soc/st/stm32_hdp.c : handle common resources: registers, clock.

6 References

- STM32MP15 reference manuals

Stable: 07.09.2021 - 09:41 / Revision: 12.08.2021 - 15:35

A quality version of this page, approved on *7 September 2021*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to

- briefly introduce the **HDP** peripheral (hardware debug port) and its main features
- indicate the level of security supported by this hardware block
- explain how each instance can be allocated to the three runtime contexts and linked to the corresponding software components
- explain, when needed, how to configure the HDP peripheral.

2 Peripheral overview

The **HDP** peripheral is used to output some internal signals on up to 8 GPIO pins.

Follow the sequence below to connect a GPIO to an internal signal via the HDP:

- First of all, look for the internal signal you want to monitor in the HDP signal multiplexing table of the STM32MP15 reference manuals:
  - Search for the HDP signal on which you can get it among eight possible choices.
  - Note the corresponding **HDPx multiplexing value** to select.
- Then, look for the most suitable GPIO pin on which you can output HDPx in the datasheet:
  - Note the GPIO **bank** and **pin**.
  - Note the corresponding GPIO **alternate function** (AF) to select.

The GPIO **bank**, **pin**, **alternate function** and **HDPx multiplexing value** are the information required to configure each HDP signal.

2.1 Features

Refer to STM32MP15 reference manuals for the complete list of features, and to the software components, introduced below, to know which features are really implemented.

2.2 Security support

The HDP is a **non-secure** peripheral.

3 Peripheral usage and associated software

3.1 Boot time

The HDP is not used at boot time.

3.2 Runtime

3.2.1 Overview

The HDP can be allocated to the Arm® Cortex®-A7 non-secure core to be used under Linux®HDP driver.

3.2.2 Software frameworks

Domain	Peripheral	Software components			Comment
OP-TEE	Linux	STM32Cube			
Trace & Debug	HDP		HDP Linux driver		

3.2.3 Peripheral configuration

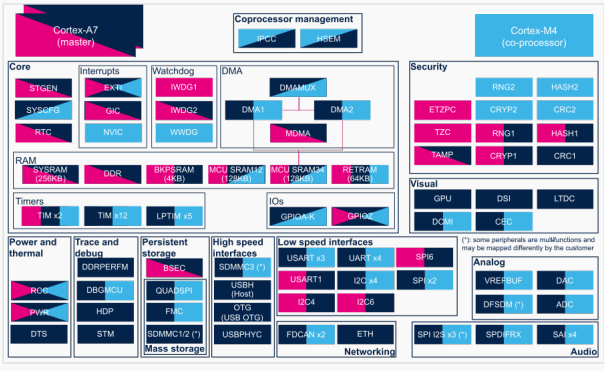
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration by itself can be performed via the STM32CubeMX tool for all internal peripherals. It can then be manually completed (especially for external peripherals) according to the information given in the corresponding software framework article.

3.2.4 Peripheral assignment

Check boxes illustrate the possible peripheral allocations supported by STM32 MPU Embedded Software:

- means that the peripheral can be assigned () to the given runtime context.
- is used for system peripherals that cannot be unchecked because they are statically connected in the device.

Refer to How to assign an internal peripheral to a runtime context for more information on how to assign peripherals manually or via STM32CubeMX.
The present chapter describes STMicroelectronics recommendations or choice of implementation. Additional possiblities might be described in STM32MP15 reference manuals
.



Internal peripherals

Domain	Periphera l	Runtime allocation		Comment
	Cortex-A7	Cortex-A7	Cortex-M4	

Domain	Periphera	Runtime allocation				Comment
Instance	secure (OP-TEE)	non-secure (Linux)	(STM32Cube)			
Trace & Debug	HDP	HDP				

}}

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)
```

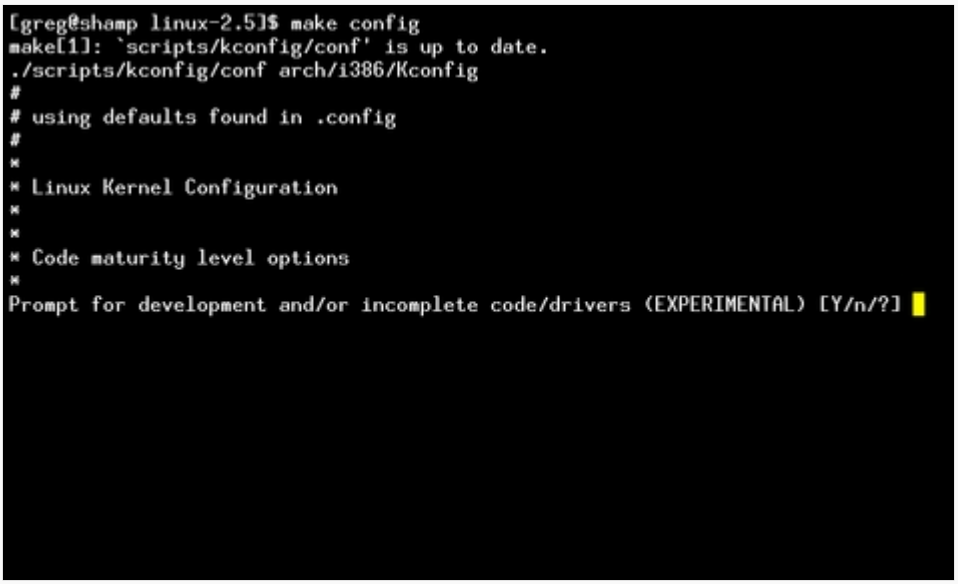


Figure 1. Configuring the kernel with make config

```
   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 :

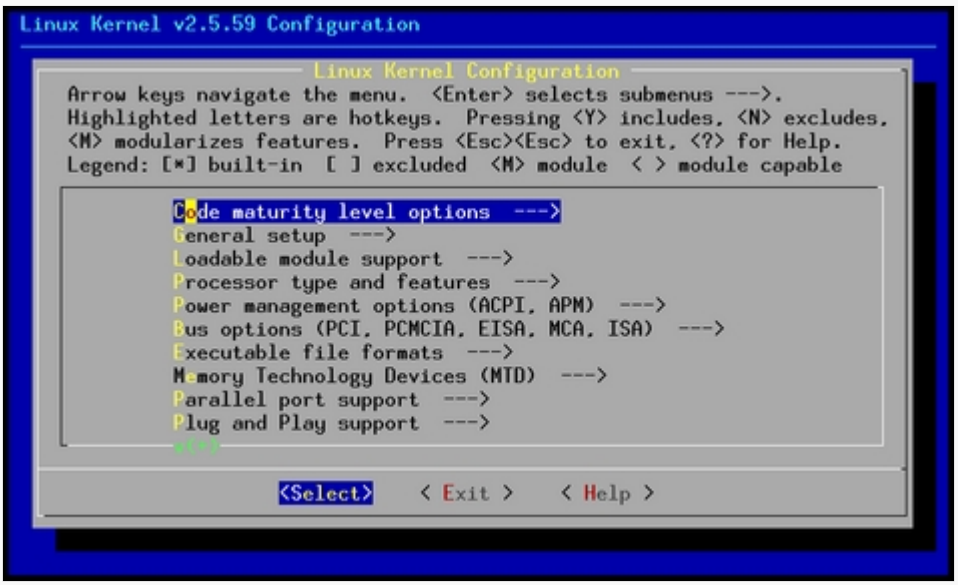


Figure 2. Make menuconfig makes it easier to back up and correct mistakes

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.

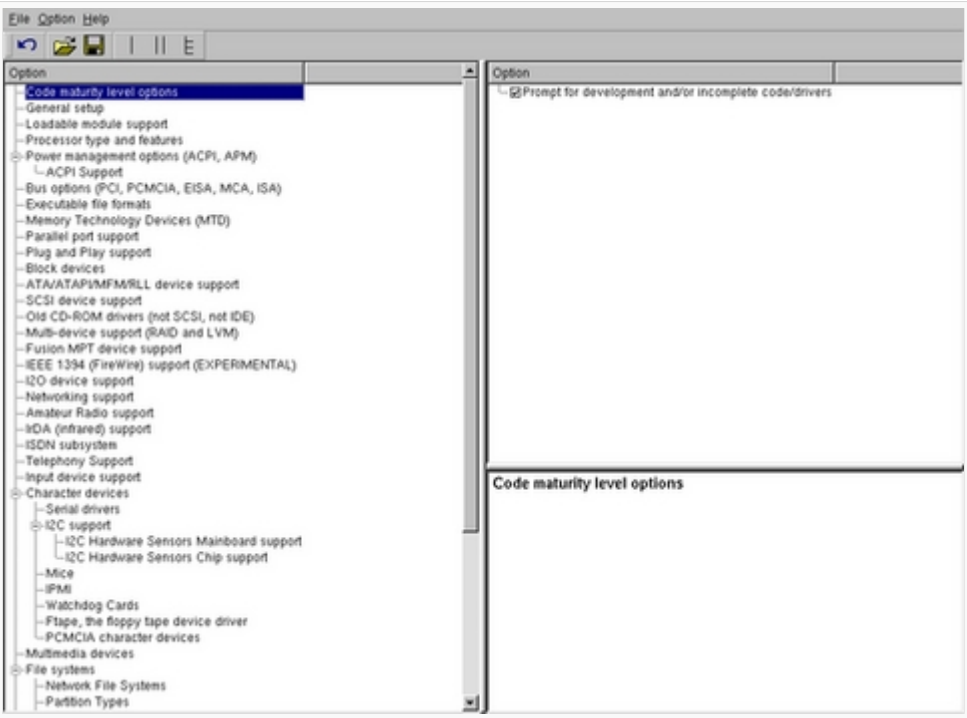


Figure 3. The Qt-Based make xconfig

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/
```

Information

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
```

Information

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

Stable: 25.09.2020 - 09:16 / Revision: 25.09.2020 - 09:15

A quality version of this page, approved on *25 September 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the PWR peripheral and its main features
- indicate the level of security supported by this hardware block
- explain how it can be allocated to the three runtime contexts and linked to the corresponding software components
- explain, when necessary, how to configure the PWR peripheral.

2 Peripheral overview

The **PWR** peripheral is used to control the device power supply configuration.

It has 6 input pins (named wakeup pins) which can be programmed to wake the system up from low power. The wakeup pins are listed with **WKUP** prefix in the STM32MP15 Datasheet.

These pins can be used by the Cortex®-A7 non secure (via Cortex®-A7 secure services) or the Cortex®-M4.

The PWR peripheral provides 2 output hardware lines named PWR_ON and PWR_LP:

- In STPMIC1 configuration, PWR_ON allows to select the register bank (active or low power). PWR_LP is not used.
- In the power discrete solution they drive VDDcore which feeds the Cortex®-A7, the Cortex®-M4 and the peripherals. They also control the DDR power supplies (VDD_DDR, VREF_DDR, VTT_DDR).

2.1 Features

Refer to the STM32MP15 reference manuals for the complete list of features, and to the software components, introduced below, to see which features are implemented.

2.2 Security support

The PWR is **secure aware** with the security control managed via RCC **TZEN** bit.

3 Peripheral usage and associated software

3.1 Boot time

The PWR is closely configured together with RCC by all the boot components: the ROM code, the FSBL, the SSBL and up to Linux® kernel. Its configuration is carried by the device tree.

3.2 Runtime

3.2.1 Overview

The PWR peripheral is shared at runtime:

- the Cortex®-A7 secure controls all secure registers (cf. TZEN description above) with PWR OP-TEE driver.

and

- the Cortex®-A7 non-secure mainly controls it via the regulator framework and the interrupt framework in Linux

and

- the Cortex®-M4 controls it in STM32Cube with PWR HAL driver

A concurrent control from each context is possible because the described management is realized via independent registers.

3.2.2 Software frameworks

Domain	Peripheral	Software components			Comment
OP-TEE	Linux	STM32Cube			
Power & Thermal	PWR	OP-TEE PWR driver	Linux regulator framework	STM32Cube PWR driver	

3.2.3 Peripheral configuration

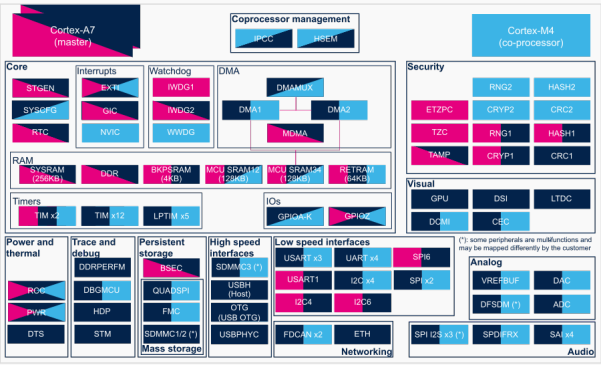
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in the corresponding software framework article.

3.2.4 Peripheral assignment

Check boxes illustrate the possible peripheral allocations supported by STM32 MPU Embedded Software:

- means that the peripheral can be assigned () to the given runtime context.
- is used for system peripherals that cannot be unchecked because they are statically connected in the device.

Refer to How to assign an internal peripheral to a runtime context for more information on how to assign peripherals manually or via STM32CubeMX.
The present chapter describes STMicroelectronics recommendations or choice of implementation. Additional possiblities might be described in STM32MP15 reference manuals.



Internal peripherals

Domain	Periphera l	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Power & Thermal	PWR	PWR				

4 How to go further

The PWR is interfaced with the hardware debug port (HDP) of the STM32MP15. This link offers the flexibility to observe the main PWR state signals on debug pins.

Please refer to STM32MP15 reference manuals for the exact list of signals that can be monitored.

5 References

Stable: 25.09.2020 - 09:10 / Revision: 25.09.2020 - 09:09

A quality version of this page, approved on *25 September 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the RCC peripheral and its main features
- indicate the level of security supported by this hardware block
- explain, when necessary, how to configure the RCC peripheral.

2 Peripheral overview

The **RCC** peripheral is used to control the internal peripherals, as well as the **reset** signals and **clock** distribution. The RCC gets several internal (LSI, HSI and CSI) and external (LSE and HSE) clocks. They are used as clock sources for the hardware blocks, either directly or indirectly, via the four PLLs (PLL1, PLL2, PLL3 and PLL4) that allow to achieve high frequencies.

2.1 Features

Refer to the STM32MP15 reference manuals for the complete list of features, and to the software components, introduced below, to see which features are really implemented.

2.2 Security support

The RCC is a **secure** peripheral. There are two levels of security, which are controlled via two bits in the RCC_TZCR register (only accessible in secure mode):

- **TZEN** allows to set some RCC registers in secure mode, in particular registers for configuring PLL1 and PLL2, in order to secure a TrustZone perimeter for the Cortex®-A7 secure core and its peripherals.
- **MCKPROT** allows extending the TZEN secure clock control perimeter to PLL3 and to the MCU subsystem, so to the Cortex® -M4 and its bus clock.

Please note that all RCC registers can be read from the non-secure world.

3 Peripheral usage and associated software

3.1 Boot time

The RCC security level differs for each boot chain:

- the trusted boot chain sets TZEN to 1 an MCKPROT to 0
- the basic boot chain sets TZEN to 0 an MCKPROT to 0

The RCC is used by all the boot components: the ROM code, the FSBL, the SSBL and up to the Linux® kernel. Nevertheless, the main initialization step is performed by the FSBL that is responsible for the clock tree initialization: it consists in configuring all the input clocks, the PLL and the clock sources that are selected as kernel clocks for all peripherals. The whole configuration is carried out by the device tree.

The STM32CubeMX tool allows configuring in one place the clock tree that will be applied at boot time and used at runtime, so it is highly recommended to use it to generate your device tree. Moreover, the STM32CubeMX integrates all the information documented in the STM32MP15 reference manuals, making this configuration step straighforward.

3.2 Runtime

3.2.1 Overview

The RCC peripheral is shared at runtime:

- the Arm® Cortex®-A7 secure core controls all the secure registers (refer to TZEN and MCKPROT bit descriptions) through the RCC OP-TEE driver. The access to some secure registers from the Cortex®-A7 non-secure core can be achieved via runtime secure services implemented in the secure monitor (from the OP-TEE if it is present, otherwise from the TF-A).
- the Arm® Cortex®-A7 non-secure core controls the clock management via the clock framework, and the reset management via the reset framework in Linux®.
- the Arm® Cortex®-M4 core controls all the clock and reset managements in STM32Cube with the RCC HAL driver

Concurrent control from each context is possible because the above managements are performed via independent registers.

3.2.2 Software frameworks

Domain	Peripheral	Software components			Comment
OP-TEE	Linux	STM32Cube			
Power & Thermal	RCC	OP-TEE RCC driver	Reset framework Clock framework	STM32Cube RCC driver	

3.2.3 Peripheral configuration

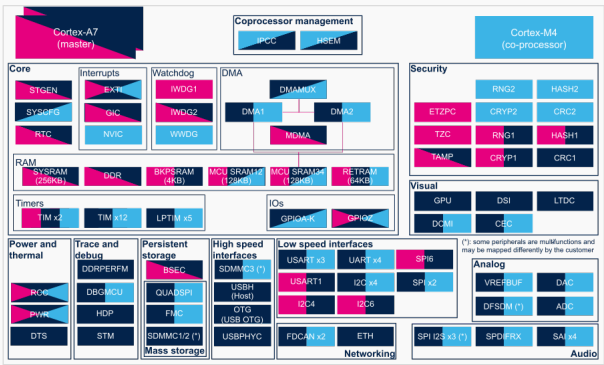
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in the corresponding software framework article.

3.2.4 Peripheral assignment

Check boxes illustrate the possible peripheral allocations supported by STM32 MPU Embedded Software:

- means that the peripheral can be assigned () to the given runtime context.
- is used for system peripherals that cannot be unchecked because they are statically connected in the device.

Refer to How to assign an internal peripheral to a runtime context for more information on how to assign peripherals manually or via STM32CubeMX.
The present chapter describes STMicroelectronics recommendations or choice of implementation. Additional possiblities might be described in STM32MP15 reference manuals .



Internal peripherals

Domain	Peripheral	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Power & Thermal	RCC	RCC				

4 How to go further

The RCC is interfaced with the HDP internal peripheral, thus offering the flexibility to monitor the main RCC state signals on the debug pins.

Please refer to the STM32MP15 reference manuals for the full list of signals that can be monitored.

5 References

Stable: 17.11.2021 - 16:41 / Revision: 17.11.2021 - 10:47

A quality version of this page, approved on *17 November 2021*, was based off this revision.

All the resources for the STM32MP1 Series are located in the Resources area of the STM32MP1 Series web page.

The resources below are referenced in some of the articles of this user guide.

Information

The different **STM32MP15** microprocessor **part numbers** available (with their corresponding internal peripherals, security options and packages) are described in the **STM32MP15 microprocessor part numbers**.

NEW means that the document (or its version) is new compared to what was delivered within the previous ecosystem release.

Reference	Name	Link	Version
Application notes			
AN4803	High-speed SI simulations using IBIS and board-level simulations using HyperLynx® SI on STM32 MCUs and MPUs	AN4803.pdf	v2.0
AN5027	Interfacing PDM digital microphones using STM32 MCUs and MPUs	AN5027.pdf	v2.0
AN5031	Getting started with STM32MP15 Series hardware development	AN5031.pdf	v3.0
AN5036	Thermal management guidelines for STM32 applications	AN5036.pdf	v3.0
AN5109	STM32MP1 Series using low-power modes	AN5109.pdf	v4.0
AN5122	STM32MP1 Series DDR memory routing guidelines	AN5122.pdf	v3.0
AN5168	STM32MP1 series DDR configuration	AN5168.pdf	NEW v2.0
AN5225	USB Type-C™ Power Delivery using STM32xx Series MCUs and STM32xxx Series MPUs	AN5225.pdf	NEW v5.0
	Migration of microcontroller applications from STM32F4x9 lines to	AN525	

Reference	Name	Link	Version
Application notes			
AN5253	STM32MP151, STM32MP153 and STM32MP157 lines microprocessor	3.pdf	v1.0
AN5256	STM32MP151, STM32MP153 and STM32MP157 discrete power supply hardware integration	AN5256.pdf	v2.0
AN5260	STM32MP151/153/157 MPU lines and STPMIC1B integration on a battery powered application	AN5260.pdf	v2.0
AN5275	USB DFU/USART protocols used in STM32MP1 Series bootloaders	AN5275.pdf	v1.0
AN5284	STM32MP1 series system power consumption	AN5284.pdf	v1.0
AN5348	FDCAN peripheral on STM32 devices	AN5348.pdf	v1.0
AN5431	The STPMIC1 PCB layout guidlines	AN5431.pdf	v1.0
AN5438	STM32MP1 Series lifetime estimates	AN5438.pdf	v1.0
AN5510	Overview of the secure secret provisioning (SSP) on STM32MP1 Series	AN5510.pdf	v1.0
Datasheets[1]			
DS12505	STM32MP157C/F datasheet (secure)	DS12505.pdf	NEW v6.0
DS12504	STM32MP157A/D datasheet (basic)	DS12504.pdf	NEW v6.0
DS12503	STM32MP153C/F datasheet (secure)	DS12503.pdf	NEW v6.0
DS12502	STM32MP153A/D datasheet (basic)	DS12502.pdf	NEW v6.0
DS12501	STM32MP151C/F datasheet (secure)	DS12501.pdf	NEW v6.0
DS12500	STM32MP151A/D datasheet (basic)	DS12500.pdf	NEW v6.0
DS12792	STPMIC1 datasheet	DS12792.pdf	NEW v8.0
Errata sheets			
		ES043	

Reference	Name	Link	Version
Application notes			
ES0438	STM32MP15xx device errata	8.pdf	v6.0
Reference manuals[1]			
RM0436	STM32MP157 reference manual (STM32MP157xxx advanced Arm®-based 32-bit MPUs)	RM0436.pdf	v5.0
RM0442	STM32MP153 reference manual (STM32MP153xxx advanced Arm®-based 32-bit MPUs)	RM0442.pdf	v5.0
RM0441	STM32MP151 reference manual (STM32MP151xxx advanced Arm®-based 32-bit MPUs)	RM0441.pdf	v5.0
Boards schematics			
MB1262 schematics	STM32MP157C-EV1 motherboard schematics MB1262-C01 board schematic (Evaluation board)	MB1262-C01.pdf	v1.0
MB1263 schematics	STM32MP157F-EV1 daughterboard schematics MB1263-C04 board schematic (Evaluation board)	MB1263-C04.pdf	v4.0
MB1230 schematics	DSI 720p LCD display daughterboard schematics MB1230-C board schematic (Evaluation board)	MB1230-C.pdf	v1.1
MB1379 schematics	Camera daughterboard schematics MB1379-A01 board schematic (Evaluation board)	MB1379-A01.pdf	v1.0
MB1272 schematics	STM32MP157x-DKx motherboard schematics MB1272-DK2-C01 board schematic (Discovery kit)	MB1272-C01.pdf	v1.0
MB1407 schematics	STM32MP157x-DKx daughterboard schematics MB1407-LCD-C01 board schematic (Discovery kit)	MB1407-C01.pdf	v1.0
Boards user manuals			
UM2535	STM32MP157x-EV1 evaluation board user manual	UM2535.pdf	v2.0
UM2534	STM32MP157x-DKx discovery board user manual	UM2534.pdf	v1.0
Tools user manuals			
UM2563	STM32CubeIDE installation guide	UM2563.pdf	v2.0
		UM257	

Reference	Name	Link	Version
Application notes			
UM2579	Migration guide from System Workbench to STM32CubeIDE	9.pdf	v1.0
UM2553	STM32CubeIDE quick start guide	UM2553.pdf	v2.0
AN5360	Getting started with projects based on the STM32MP1 Series in STM32CubeIDE	AN5360.pdf	v1.0
UM2609	STM32CubeIDE user guide	UM2609.pdf	NEW v4.0
UM1718	STM32CubeMX user manual	UM1718.pdf	NEW v36.0
UM2237	STM32CubeProgrammer tool user manual	UM2237.pdf	NEW v17.0
UM2238	STM32 Trusted Package Creator tool user manual	UM2238.pdf	NEW v9.0
UM2542	STM32 Series Key Generator tool user manual	UM2542.pdf	NEW v2.0
UM2543	STM32 Series Signing tool user manual	UM2543.pdf	NEW v2.0

- 1.01.1 The part numbers are specified in STM32MP15 microprocessor part numbers

Archives

STM32MP15 release	ST documentation
STM32MP15-Ecosystem-v3.0.0	STM32MP15 resources - v3.0.0 page for the previous v3 ecosystem release
STM32MP15-Ecosystem-v2.1.0	STM32MP15 resources - v2.1.0 page for the v2 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v2.0.0	STM32MP15 resources - v2.0.0 page for the v2 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.2.0	STM32MP15 resources - v1.2.0 page for the v1 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.1.0	STM32MP15 resources - v1.1.0 page for the v1 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.0.0	STM32MP15 resources - v1.0.0 page for the v1 ecosystem releases (in archived wiki)

USB port or connector

Universal Synchronous/Asynchronous Receiver/Transmitter