RTC internal peripheral

Contents

Stable: 19.02.2020 - 10:01 / Revision: 04.02.2020 - 15:23

Stable: 19.02.2020 - 10:01 / Revision: 04.02.2020 - 13:23

A quality version of this page, approved on *19 February 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to

- briefly introduce the RTC 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 needed, how to configure the RTC peripheral.

2 Peripheral overview

The **RTC** peripheral is used to provide the date and clock to the application. It supports programmable alarms and wake up capabilities.

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 **RTC** peripheral is a **secure** peripheral.

3 Peripheral usage and associated software

3.1 Boot time

By default after a backup domain power-on reset (performed at boot time), all RTC registers can be read or written in both secure and non-secure modes.
In OpenSTLinux distribution, the first stage bootloader (FSBL, running in secure mode) keeps this default configuration, leaving full control to Linux® at runtime.

The **RTC** peripheral is able to generate two interrupts:

- A secure interrupt, connected to the Arm®Cortex®-A7 GIC, not used in OpenSTLinux distribution.
- A non-secure interrupt, connected both to Arm®Cortex®-A7 GIC and Cortex-M4 NVIC: this interrupt is used on Linux® and by default in OpenSTLinux distribution.

The **RTC** peripheral is part of the backup domain which reset and clock are controlled via the RCC by the first stage bootloader (FSBL, running in secure mode) at boot time.
The RTC reset occurs when the backup domain is reset. To avoid clearing the TAMP register contents, this is only done on cold boot, not on wake up.

3.2 Runtime

3.2.1 Overview

The **RTC** peripheral can be allocated to the Arm®Cortex®-A7 non-secure core to be used under Linux® with RTC framework.

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core	RTC		Linux RTC framework		

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 STM32CubeMX tool for all internal peripherals, and then manually completed (especially for external peripherals) according to the information given in the corresponding software framework article.

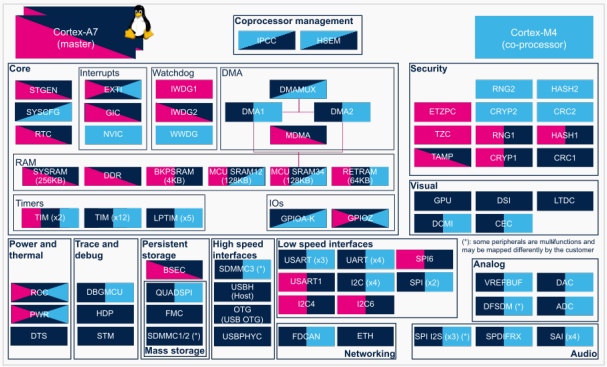
For Linux kernel configuration, please refer to RTC device tree configuration.

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)			
Core	RTC	RTC				RTC is mandatory to resynchronize STGEN after exiting low-power modes.

Real Time Clock

First Stage Boot Loader

Open Portable Trusted Execution Environment

Stable: 08.01.2021 - 15:02 / Revision: 08.01.2021 - 14:32

A quality version of this page, approved on *8 January 2021*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to

- briefly introduce the **GIC** peripheral (generic interrupt controller) 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 GIC peripheral.

2 Peripheral overview

The **GIC** peripheral is the Arm®Cortex®-A7 interrupt controller. It is consequently not accessible from the Arm®Cortex®-M4 core.

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 GIC is a **secure** peripheral (under ETZPC control).

3 Peripheral usage and associated software

3.1 Boot time

The GIC is configured by the FSBL (see Boot chain overview), mainly to define the routing of each interrupt to the secure or non-secure context at runtime.

3.2 Runtime

3.2.1 Overview

The GIC can be allocated:

- to the Arm®Cortex®-A7 secure core to be used under OP-TEE with the GIC OP-TEE driver (or TF-A secure monitor if the OP-TEE is not present)
- or to the Arm®Cortex®-A7 non-secure core to be used under Linux® with the interrupts framework

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core/Interrupts	GIC	OP-TEE GIC driver	Linux interrupt framework		

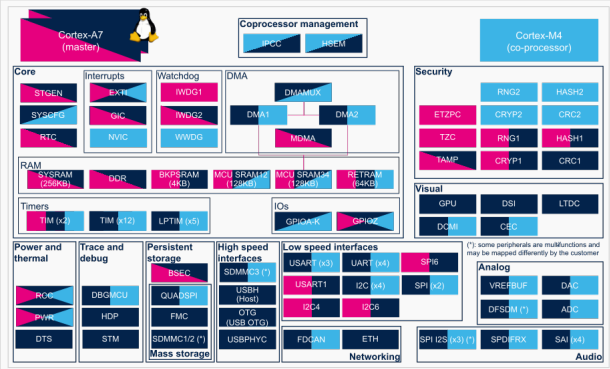
3.2.3 Peripheral configuration

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.



Internal peripherals

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.

Domain	Peripheral	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core /Interrupts	GIC	GIC				

Generic Interrupt Controller

Cortex®

First Stage Boot Loader

Open Portable Trusted Execution Environment

Stable: 16.02.2021 - 17:29 / Revision: 16.02.2021 - 17:11

A quality version of this page, approved on *16 February 2021*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the software that assigns a peripheral to a runtime context.

2 Introduction

A peripheral can be **assigned** to a runtime context via the configuration defined in the device tree. The device tree can be either generated by the STM32CubeMX tool or edited manually.

On STM32MP15 line devices, the assignement can be strengthened by a hardware mechanism: the ETZPC internal peripheral, which is configured by the TF-A boot loader. The ETZPC internal peripheral isolates the peripherals for the Cortex-A7 secure or the Cortex-M4 context. The peripherals assigned to the Cortex-A7 non-secure context are visible from any context, without any isolation.

The components running on the platform after TF-A execution (such as U-Boot, Linux, STM32Cube and OP-TEE) must have a **configuration** that is consistent with the assignment and the isolation configurations.

The following sections describe how to configure TF-A, U-Boot, Linux and STM32Cube accordingly.

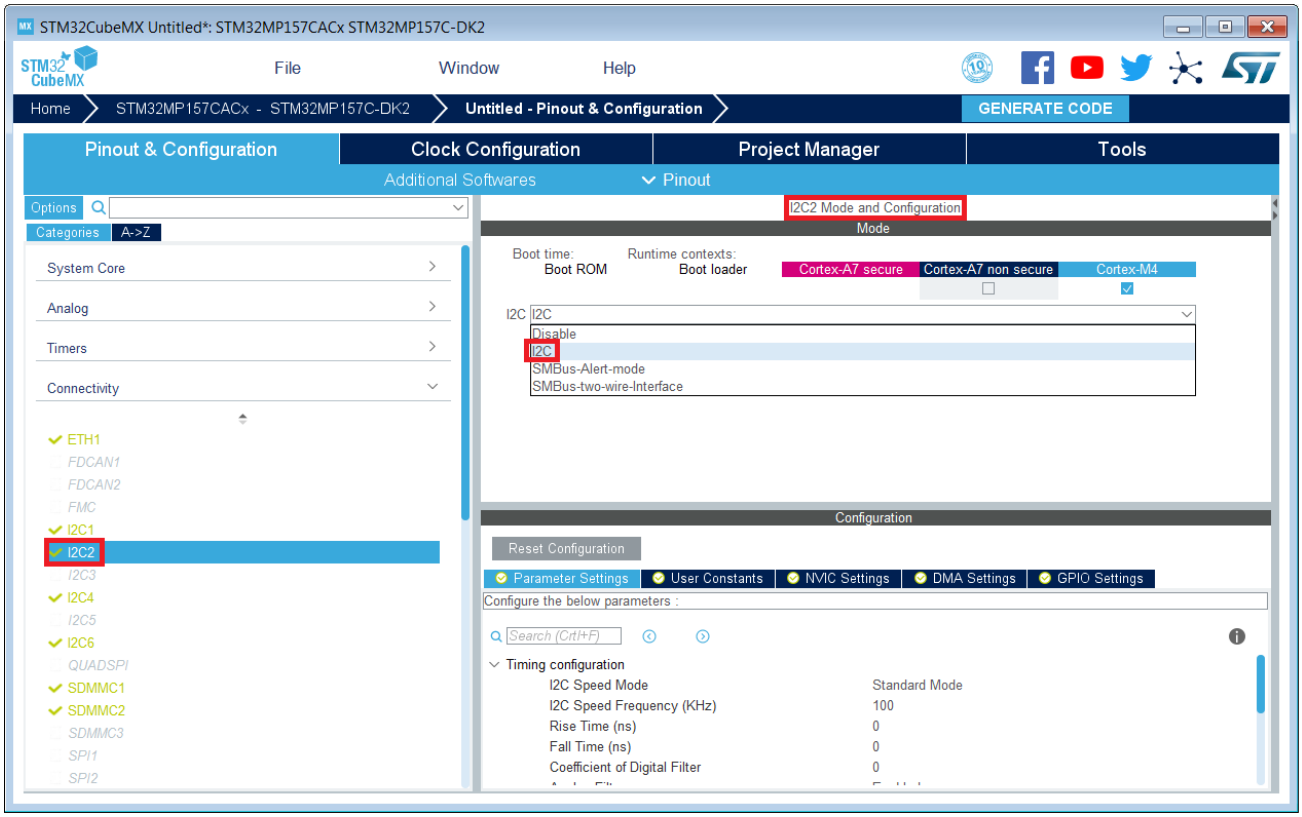
Information

Beyond the peripherals assignment, explained in this article, it is also important to understand How to configure system resources (i.e clocks, regulator, gpio,...), shared between the Cortex-A7 and Cortex-M4 contexts

3 STM32CubeMX generated assignment

The screenshot below shows the STM32CubeMX user interface:

- **I2C2** peripheral is selected, on the left
- **I2C2 Mode and Configuration** panel, on the right, shows that this I2C instance can be assigned to the Cortex-A7 non-secure or the Cortex-M4 (that is selected) runtime context
- **I2C** mode is enabled in the drop down menu



Information

The context assignment table is displayed inside each peripheral **Mode and Configuration** panel but it is possible to display it for all the peripherals in the **Options** menu via the **Show contexts** option

The **GENERATE CODE** button, on the top right, produces the following:

- The **TF-A device tree** with the ETZPC configuration that isolates the I2C2 instance (in the example) for the Cortex-M4 context. This same device tree can be used by **OP-TEE**, when enabled
- The **U-Boot device tree** widely inherited from the Linux one, just below
- The **Linux kernel device tree** with the I2C node disabled for Linux and enabled for the coprocessor
- The **STM32Cube project** with I2C2 HAL initialization code

The Manual assignment section, just below, illustrates what STM32CubeMX is generating as it follows the same example.

Information

In addition of this generation, the user may have to manually complete the system resources configuration in the user sections embedded in the STM32CubeMX generated device tree. Refer to How to configure system resources for details.

4 Manual assignment

This section gives step by step instructions, per software components, to manually perform the peripherals assignments.
It takes the same I2C2 example as the previous section, that showed how to use STM32CubeMX, in order to make the move from one approach to the other easier.

Information

The assignments combinations described in the STM32MP15 peripherals overview article are naturally supported by STM32MPU Embedded Software distribution. Note that the STM32MP15 reference manual may describe more options that would require embedded software adaptations

4.1 TF-A

The assignment follows the ETZPC device tree configuration, with below possible values:

- **DECPROT_S_RW** for the Cortex-A7 secure (Secure OS like OP-TEE)
- **DECPROT_NS_RW** for the Cortex-A7 non-secure (Linux)
  - As stated earlier in this article, there is no hardware isolation for the Cortex-A7 non-secure so this value allows accesses from any context
- **DECPROT_MCU_ISOLATION** for the Cortex-M4 (STM32Cube)

Example:

```
@etzpc: etzpc@5C007000 {
        st,decprot = <
            DECPROT(STM32MP1_ETZPC_I2C2_ID, DECPROT_MCU_ISOLATION, DECPROT_UNLOCK)
        >;
};
```

Information

The value **DECPROT_NS_RW** can be used with **DECPROT_LOCK** as last parameter. In Cortex-M4 context, this specific configuration allows the generation of an error in the resource manager utility while trying to use on Cortex-M4 side a peripheral that is assigned to the Cortex-A7 non-secure context. If **DECPROT_UNLOCK** is used, then the utility allows the Cortex-M4 to use a peripheral that is assigned to the Cortex-A7 non-secure context.

4.2 U-boot

No specific configuration is needed in U-Boot to configure the access to the peripheral.

Information

U-Boot does not perform any check with regards to ETZPC configuration before accessing to a peripheral. In case of inconsistency an illegal access is generated.

Information

U-Boot checks the consistency between ETZPC isolation configuration and Linux kernel device tree configuration to guarantee that Linux kernel do not access an unauthorized device. In order to avoid the access to an unauthorized device, the U-boot fixes up the Linux kernel device tree to disable the peripheral nodes which are not assigned to the Cortex-A7 non-secure context.

4.3 Linux kernel

Each assignable peripheral is declared twice in the Linux kernel device tree:

- Once in the **soc** node from arch/arm/boot/dts/stm32mp151.dtsi , corresponding to Linux assigned peripherals
  - Example: i2c2
- Once in the **m4_rproc** node from arch/arm/boot/dts/stm32mp157-m4-srm.dtsi , corresponding to the Cortex-M4 context.

Those nodes are disabled, by default.

  - Example: m4_i2c2

In the board device tree file (*.dts), each assignable peripheral has to be enabled only for the context to which it is assigned, in line with TF-A configuration.
As a consequence, a peripheral assigned to the Cortex-A7 secure has both nodes disabled in the Linux device tree.

Example:

```
&i2c2 {
        status = "disabled";
};
...
&m4_i2c2 {
        status = "okay";
};
```

Information

In addition of this assignment, the user may have to complete the system resources configuration in the device tree nodes. Refer to How to configure system resources for details.

4.4 STM32Cube

There is no configuration to do on STM32Cube side regarding the assignment and isolation. Nevertheless, the resource manager utility, relying on ETZPC configuration, can be used to check that the corresponding peripheral is well assigned to the Cortex-M4 before using it.

Example:

```
int main(void)
{
...
 /* Initialize I2C2------------------------------------------------------ */
 /* Ask the resource manager for the I2C2 resource */
 ResMgr_Init(NULL, NULL);
 if (ResMgr_Request(RESMGR_ID_I2C2, RESMGR_FLAGS_ACCESS_NORMAL | \
```

```
                                        RESMGR_FLAGS_CPU1, 0, NULL) != RESMGR_OK)
  {
    Error_Handler();
  }
...
  if (HAL_I2C_Init(&I2C2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

4.5 OP-TEE

The OP-TEE OS may use STM32MP1 resources. OP-TEE STM32MP1 drivers register the device driver they intend to used in a secure context. This information is used to consolidate system configuration including secure hardening of configurable peripherals.

In most case, the OP-TEE driver probe relies on OP-TEE device tree porperty *secure-status = "okay"*.

Cortex®

Trusted Firmware for Arm Cortex-A

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

Linux® is a registered trademark of Linus Torvalds.

Inter-Integrated Circuit (Bi-directional 2-wire bus standard for efficient inter-IC control.)

Open Portable Trusted Execution Environment

Hardware Abstraction Layer

Operating System

Microcontroller Unit (MCUs have internal flash memory and are intended to operate with a minimum amount of external support ICs. They commonly are a self-contained, system-on-chip (SoC) designs.)

Extended TrustZone Protection Controller

Stable: 17.02.2021 - 07:50 / Revision: 17.02.2021 - 07:49

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

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the NVIC and its main features
- indicate the level of security supported by this hardware block
- explain how the NVIC can be configured.

2 Peripheral overview

The **NVIC** is the Arm®Cortex®-M4 interrupt controller. As a result, it cannot be accessed by the Arm Cortex-A7 core.

2.1 Features

Refer to 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 NVIC is a **non-secure** peripheral.

3 Peripheral usage and associated software

3.1 Boot time

The NVIC can be configured through the STM32Cube.

3.2 Runtime

3.2.1 Overview

The NVIC can be allocated only to the Arm Cortex-M4 core to be controlled in the STM32Cube by the NVIC HAL driver.

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks		Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)		
Core /Interrupts	NVIC		STM32Cube NVIC driver	

3.2.3 Peripheral configuration

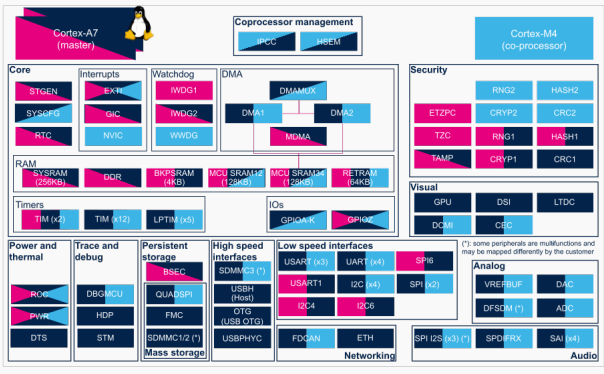
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in 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
	Cortex-A7	Cortex-A7	Cortex-M4	

Domain	Peripheral	Runtime allocation				Comment
Instance	secure (OP-TEE)	non-secure (Linux)	(STM32Cube)			
Core /Interrupts	NVIC	NVIC				

Nested Vectored Interrupt Controller

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

Open Portable Trusted Execution Environment

Linux® is a registered trademark of Linus Torvalds.

Stable: 01.12.2020 - 10:35 / 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.

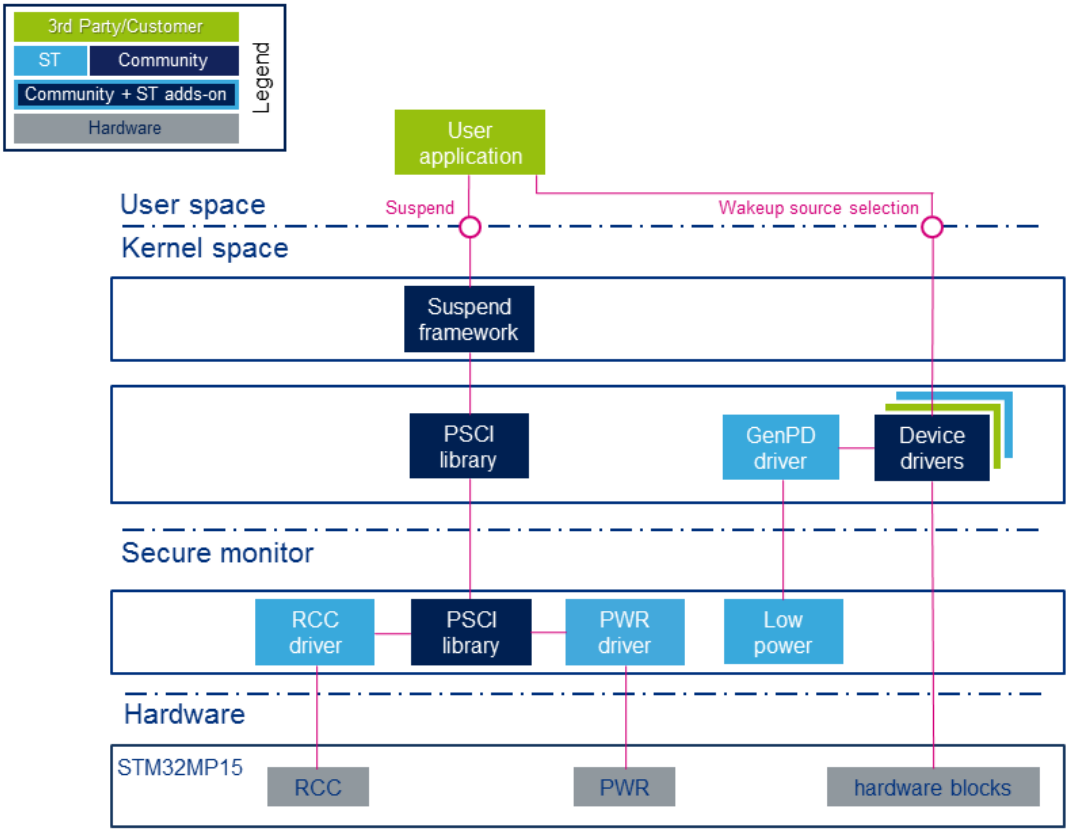
The following table gives the list of wakeup sources available in each mode.

Mode	Available wakeup sources
CStop /CStandby /Stop	BOR, PVD, AVD, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, USB, CEC, ETH, USART, 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.

Microprocessor Unit

Microcontroller Unit (MCUs have internal flash memory and are intended to operate with a minimum amount of external support ICs. They commonly are a self-contained, system-on-chip (SoC) designs.)

Brownout reset

Programmable Voltage Detector

Analog Voltage Detector

Low Speed External oscillator (STM32 clock source)

Cascading Style Sheets (web standard)

Real Time Clock

Tamper

Consumer Electronics Control (HDMI standard)

Ethernet

Universal Synchronous/Asynchronous Receiver/Transmitter

Serial Peripheral Interface

low-power timer (STM32 specific)

Independent Watchdog

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Power State Coordination Interface

Reset and Clock Control

Application programming interface

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.)

Low Power (MIPI® Alliance DSI standard)

Doubledata rate (memory domain)

Out Of Memory

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

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

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 frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Power & Thermal	RCC	OP-TEE RCC driver	Reset framework Clock framework	STM32Cube RCC driver	

3.2.3 Peripheral configuration

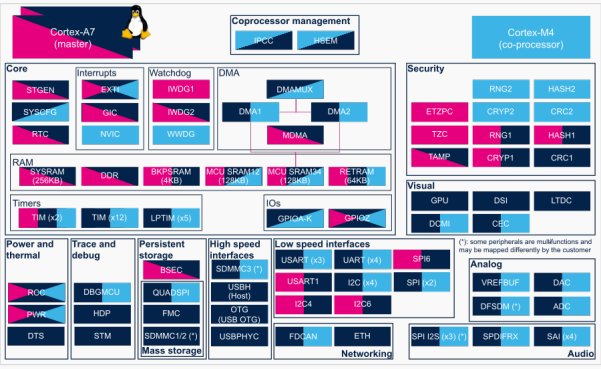
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in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

Reset and Clock Control

Low Speed Internal oscillator (STM32 clock source)

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

Multi Speed Internal oscillator (STM32 clock source)

Low Speed External oscillator (STM32 clock source)

High Speed External oscillator (STM32 clock source)

TrustZone®

Arm® and TrustZone® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

Microcontroller Unit (MCUs have internal flash memory and are intended to operate with a minimum amount of external support ICs. They commonly are a self-contained, system-on-chip (SoC) designs.)

Read Only Memory

First Stage Boot Loader

Second Stage Boot Loader

Linux® is a registered trademark of Linus Torvalds.

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere. arm

Open Portable Trusted Execution Environment

Stable: 20.11.2020 - 17:14 / Revision: 20.11.2020 - 17:14

A quality version of this page, approved on *20 November 2020*, 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 = <&rcc RTCAPB>, <&rcc RTC>;
        clock-names = "pclk", "rtc_ck";
        interrupts-extended = <&intc GIC_SPI 3 IRQ_TYPE_LEVEL_HIGH>,
                              <&exti 19 1>;
        status = "disabled";
};
```

Warning

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

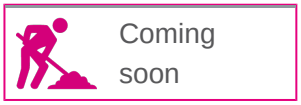
Generic Interrupt Controller

Serial Peripheral Interface

Stable: 11.02.2019 - 11:21 / Revision: 29.01.2019 - 07:55

A quality version of this page, approved on *11 February 2019*, was based off this revision.

Template:ArticleMainWriter Template:ArticleApprovedVersion



Stable: 25.09.2020 - 09:42 / Revision: 25.09.2020 - 09:36

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 STGEN 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 STGEN peripheral.

2 Peripheral overview

The STGEN peripheral provides the reference clock used by the Arm®Cortex®-A7 generic timer for its counters, including the system tick generation.

It is clocked by ACLK (the AXI bus clock), so caution is needed when this clock is changed; otherwise the operating system (running on the Cortex-A7) might run with a varying reference clock.

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 STGEN is a single-instance peripheral that can be accessed via the two following register sets:

- STGEN**C** for the **c**ontrol. That is, a **secure** port (under ETZPC control).
- STGEN**R** for the **r**ead-only access. That is, a **non secure** port.

3 Peripheral usage and associated software

3.1 Boot time

The STGEN is first initialized by the ROM code, then updated by the FSBL (see Boot chain overview) once the clock tree is set up.

3.2 Runtime

3.2.1 Overview

Linux® and OP-TEE use the Arm Cortex-A7 generic timer that gets its counter from the STGEN, but this is transparent at run time.

Hence there is no runtime allocation decision for this peripheral: **both contexts are selected by default**.

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core	STGEN	see comment	see comment		Not applicable as the STGEN peripheral is configured at boot time and not accessed at runtime

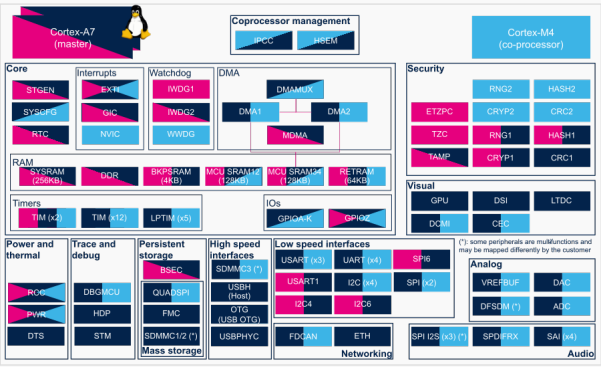
3.2.3 Peripheral configuration

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)			
Core	STGEN	STGEN				

4 How to go further

5 References

System Time Generator

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

First Stage Boot Loader

Linux® is a registered trademark of Linus Torvalds.

Open Portable Trusted Execution Environment

Stable: 23.09.2020 - 13:22 / Revision: 12.06.2020 - 13:25

A quality version of this page, approved on *23 September 2020*, was based off this revision.

1 STM32CubeMX overview

This article describes STM32CubeMX, an official STMicroelectronics graphical software configuration tool.

The STM32CubeMX application helps developers to use the STM32 by means of a user interface, and guides the user through to the initial configuration of a firmware project.

It provides the means to:

- configure pin assignments, the clock tree, or internal peripherals
- simulate the power consumption of the resulting project
- configure and tune DDR parameters
- generate HAL initialization code for Cortex-M4
- generate the Device Tree for a Linux kernel, TF-A and U-Boot firmware for Cortex-A7

It uses a rich library of data from the STM32 microcontroller portfolio.

The application is intended to ease the initial development phase by helping developers to select the best product in terms of features and power.

2 STM32CubeMX main features

- Peripheral and middleware parameters

  Presents options specific to each supported software component

- Peripheral assignment to processors

  Allows assignment of each peripheral to Cortex-A Secure, Cortex-A Non-Secure, or Cortex-M processors

- Power consumption calculator

  Uses a database of typical values to estimate power consumption, DMIPS, and battery life

- Code generation

  Makes code regeneration possible, while keeping user code intact

- Pinout configuration

  Enables peripherals to be chosen for use, and assigns GPIO and alternate functions to pins

- Clock tree initialization

  Chooses the oscillator and sets the PLL and clock dividers

- DDR tunning tool

  Ensures the configuration, testing, and tuning of the MPU DDR parameters. Using U-Boot-SPL Embedded Software.

3 How to get STM32CubeMX

Please, refer to the following link STM32CubeMX to find STM32CubeMX, the Release Note, the User Manual and the product specification.

Doubledata rate (memory domain)

Hardware Abstraction Layer

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Trusted Firmware for Arm Cortex-A

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

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Microprocessor Unit

Stable: 17.11.2020 - 17:06 / Revision: 10.11.2020 - 07:49

A quality version of this page, approved on *17 November 2020*, 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	v2.0
		AN503	

Reference	Name	Link	Version
Application notes			
AN5036	Thermal management guidelines for STM32 applications	6.pdf	v3.0
AN5109	STM32MP1 Series using low-power modes	AN5109.pdf	NEW v4.0
AN5122	STM32MP1 Series DDR memory routing guidelines	AN5122.pdf	v3.0
AN5168	STM32MP1 series DDR configuration	AN5168.pdf	v1.0
AN5225	USB Type-C™ Power Delivery using STM32xx Series MCUs and STM32xxx Series MPUs	AN5225.pdf	NEW v3.0
AN5253	Migration of microcontroller applications from STM32F4x9 lines to STM32MP151, STM32MP153 and STM32MP157 lines microprocessor	AN5253.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	v1.0
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 v4.0
DS12504	STM32MP157A/D datasheet (basic)	DS12504.pdf	NEW v4.0
DS12503	STM32MP153C/F datasheet (secure)	DS12503.pdf	NEW v4.0

Reference	Name	Link	Version
Application notes			
DS12502	STM32MP153A/D datasheet (basic)	DS12502.pdf	NEW v4.0
DS12501	STM32MP151C/F datasheet (secure)	DS12501.pdf	NEW v4.0
DS12500	STM32MP151A/D datasheet (basic)	DS12500.pdf	NEW v4.0
DS12792	STPMIC1 datasheet	stpmic1.pdf	NEW v5.0
Errata sheets			
ES0438	STM32MP15xx device errata	ES0438.pdf	v5.0
Reference manuals[1]			
RM0436	STM32MP157 reference manual (STM32MP157xxx advanced Arm®-based 32-bit MPUs)	RM0436.pdf	v4.0
RM0442	STM32MP153 reference manual (STM32MP153xxx advanced Arm®-based 32-bit MPUs)	RM0442.pdf	v4.0
RM0441	STM32MP151 reference manual (STM32MP151xxx advanced Arm®-based 32-bit MPUs)	RM0441.pdf	v4.0
Boards schematics			
MB1262 schematics	STM32MP157C-EV1 motherboard schematics MB1262-C01 board schematic (Evaluation board)	MB1262-C01.pdf	v1.0
MB1263 schematics	STM32MP157C-EV1 daughterboard schematics MB1263-C01 board schematic (Evaluation board)	MB1263-C01.pdf	v1.0
NEW 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

Reference	Name	Link	Version
Application notes			
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	v1.0
UM2579	Migration guide from System Workbench to STM32CubeIDE	UM2579.pdf	v1.0
UM2553	STM32CubeIDE quick start guide	UM2553.pdf	v1.0
AN5360	Getting started with projects based on the STM32MP1 Series in STM32CubeIDE	AN5360.pdf	v1.0
UM2609	Description of the integrated development environment for STM32 products	UM2609.pdf	v1.0
UM1718	STM32CubeMX user manual	UM1718.pdf	NEW v32.0
UM2237	STM32CubeProgrammer tool user manual	UM2237.pdf	NEW v12.0
UM2238	STM32 Trusted Package Creator tool user manual	UM2238.pdf	NEW v7.0
UM2542	STM32 Series Key Generator tool user manual	UM2542.pdf	v1.0
UM2543	STM32 Series Signing tool user manual	UM2543.pdf	v1.0

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

Archives

STM32MP15 release	ST documentation
STM32MP15-Ecosystem-v2.0.0	STM32MP15 resources - v2.0.0
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)

Doubledata rate (memory domain)

USB port or connector

Microprocessor Unit

Device Firmware Upgrade

Universal Synchronous/Asynchronous Receiver/Transmitter

Printed Circuit Board

Secure Secret Provisioning

Secure secrets provisioning

Display Serial Interface (MIPI® Alliance standard)

Stable: 25.09.2020 - 09:15 / Revision: 25.09.2020 - 09:13

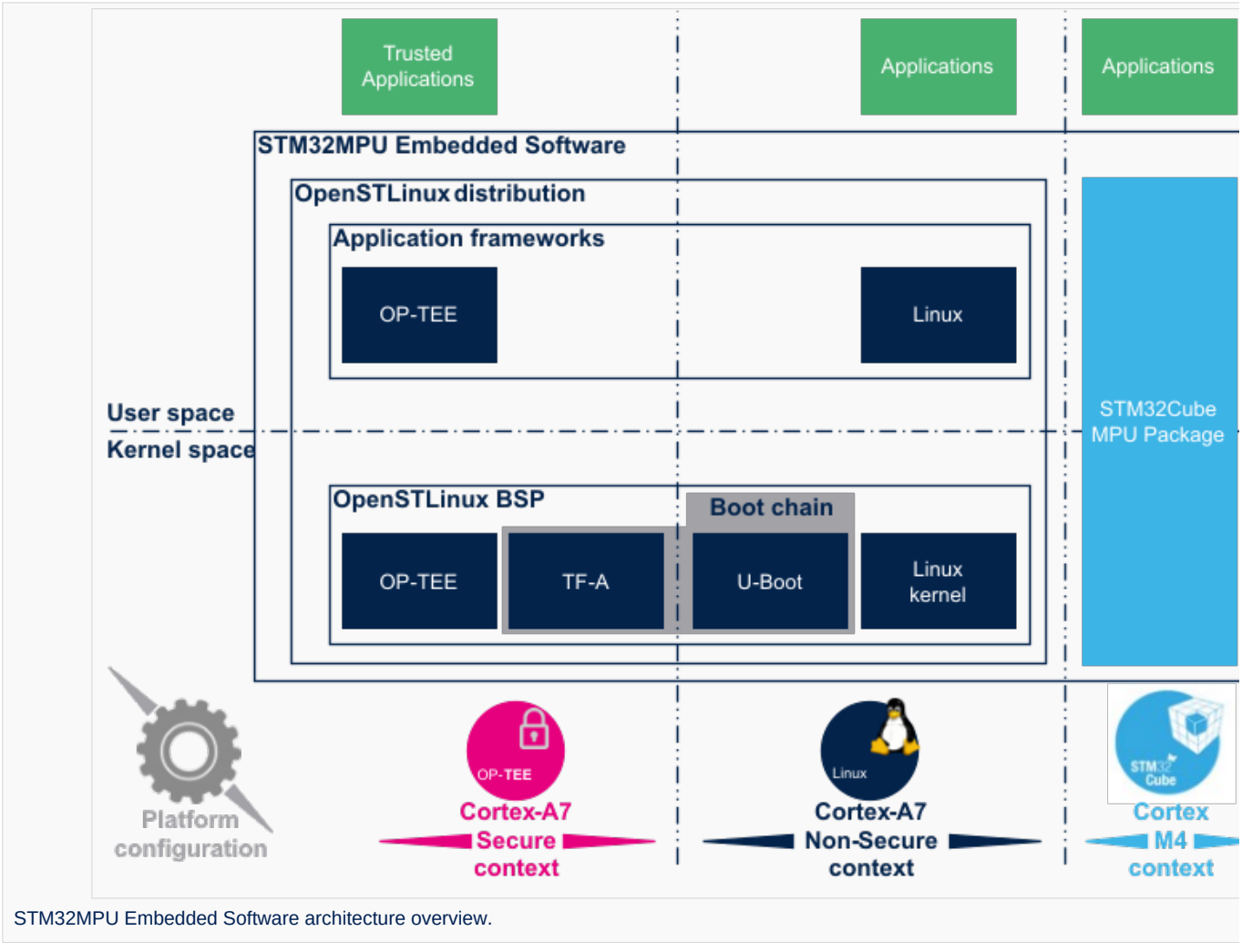
A quality version of this page, approved on *25 September 2020*, was based off this revision.

1 STM32MPU Embedded Software overview

The diagram below shows STM32MPU Embedded Software distribution main components:

- The **OpenSTLinux distribution**, running on the Arm®Cortex®-A, including:
  - The **OpenSTLinux BSP** with:
    - The boot chain based on TF-A and U-Boot.
    - The OP-TEE secure OS running on the Arm®Cortex®-A in secure mode.
    - The **Linux® kernel** running on the Arm®Cortex®-A in non-secure mode.
  - The **application frameworks** are composed of middlewares relying on the BSP and providing API:
    - on the **OP-TEE** side to run **Trusted Applications (TA)** that allow to manipulate secrets (not visible from the Linux and STM32Cube MPU Package)
    - on the **Linux** side to run **Applications** that typically interact with the user via the display, the touchscreen, etc.
- The **STM32Cube MPU Package** is running on the Arm®Cortex®-M: it is based on HAL drivers and middlewares, like other STM32 microcontrollers, completed with coprocessor management.

The figure below is clickable so that the user can directly jump to one of the sub-levels listed above.



STM32MPU Embedded Software architecture overview.

2 Open Source Software (OSS) philosophy

The **Open source software** source code is released under a license in which the copyright holder grants users the rights to study, change and distribute the software to anyone and for any purpose[1].
STMicroelectronics maximizes the using of open source software and contributes to those communities. Notice that, due to the software review life cycle, it can take some time before getting all developments accepted in the communities, so STMicroelectronics can also temporarily provide some source code on github[2], until it is merged in the targeted repository.

3 References

- https://en.wikipedia.org/wiki/Open-source_software
- STM32MP1 Distribution Package

Stable: 12.11.2020 - 10:32 / Revision: 01.10.2020 - 13:24

A quality version of this page, approved on *12 November 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the TAMP 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 TAMP peripheral.

2 Peripheral overview

The **TAMP** peripheral is used to prevent any attempt by an attacker to perform an unauthorized physical or electronic action against the device. It also includes the backup registers that remain powered-on when the platform is switched off.

Information

It is important to notice that the backup registers and the BKPSRAM internal memory can be erased when a tamper detection occurs in TAMP internal peripheral

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 TAMP is a **secure** peripheral. The access to some backup registers can be opened to the **non-secure** world via a direct configuration in TAMP, in **secure** mode.

3 Peripheral usage and associated software

3.1 Boot time

The TAMP is used at boot time to share data between the ROM code, FSBL and SSBL: see backup registers for further information.

3.2 Runtime

3.2.1 Overview

TAMP is seen as a system peripheral, that can be used by the Arm®Cortex®-A7 secure context with OP-TEE or the Arm®Cortex®-A7 non-secure context with Linux.

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Security	TAMP	OP-TEE	Linux syscon framework[1]		

For the Linux kernel configuration, sysconf part, please refer to TAMP device tree configuration.
For the secure configuration, please refer to TAMP device tree configuration and Tamper configuration.

3.2.3 Peripheral configuration

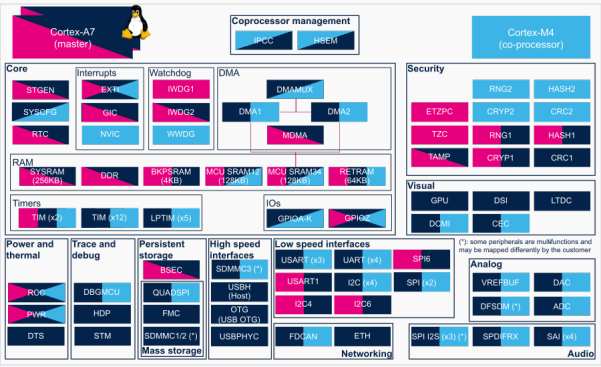
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in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

4 How to go further

5 References

- Documentation/devicetree/bindings/mfd/syscon.txt

Tamper

Read Only Memory

First Stage Boot Loader

Second Stage Boot Loader

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Open Portable Trusted Execution Environment