



## STM32MP15 Tamper configuration



---

## Contents

---

1. STM32MP15 Tamper configuration .....	3
2. BKPSRAM internal memory .....	7
3. STM32MP15 backup registers .....	11
4. STM32MP15 resources .....	19
5. TAMP device tree configuration .....	23



---

A quality version of this page, approved on *28 June 2021*, was based off this revision.

## Contents

1 Overview .....	4
2 Software configuration .....	5
2.1 Internal tampers .....	5
2.2 External tampers .....	5



---

## 1 Overview

---

The STM32 MPUs embed tamper detection management. Tamper management and configuration have been added to the secure OS to select and detect events.

STM32MP15 microprocessors offer 5 internal tampers and 3 external tampers.

The internal tampers are the following:

- RTC voltage domain monitoring
- LSE monitoring
- HSE monitoring
- RTC calendar overflow
- Monotonic counter overflow

External tampers can be configured to be passive or active.

On a tamper event detection, the [backup registers](#) are cleared and the [Backup SRAM](#) is read-protected and cannot be accessed until the next reset.

The Automatic erase mode can be configured for external tampers. It is enabled by default but can be turned off if the user application needs to control erase operations.



## 2 Software configuration

### Warning

The tamper driver only exists in the Trusted Firmware-A. It is not yet available in OP-TEE

Internal and external tampers have to be configured in:

- TAMP device tree configuration
- Secure OS main security configuration file .

This second part is statically defined and must be customized depending on the application needs. The file contains two static tables, one for internal tampers, another for external ones.

### 2.1 Internal tampers

Below the structure that registers internal tamper:

```
struct stm32_tamp_int {
    int id;
    void (*func)(int id);
};
```

Internal tamper structure contains:

- an ID, linked to the existing SoC internal tamper
- a function that is called when a tamper event is detected. This function can be customized. By default, it just prints the tamper ID and resets the SoC.

A static list of tampers is automatically registered during the main security loop.

```
static struct stm32_tamp_int int_tamp[PLAT_MAX_TAMP_INT] = {
    {
        .id = ITAMP1,
        .func = stm32mp1_tamper_action,
    },
    ...
};
```

By default, only internal tampers 1, 2, 3 and 4 are enabled.

### 2.2 External tampers

#### Warning

When an external tamper is enabled, the associated PIN is compulsorily attached to the tamper (whatever the previous GPIO config). There is also some priority on the settings inside RTC/TAMP. See [Reference Manual RTC](#) section "GPIOs controlled by the RTC and TAMP".

Below the structure that registers external tampers:



```
struct stm32_tamp_ext {
    int id;
    uint8_t mode;
    uint8_t erase;
    uint8_t evt_mask;
    void (*func)(int id);
};
```

External tamper structure contains:

- an ID, linked to the SoC external tamper
- a mode that depends on the filter count parameter (described below in this article):
  - TAMP\_TRIG\_OFF: Low trigger for passive tamper or input rising edge
  - TAMP\_TRIG\_ON: High trigger for passive tamper or input falling edge
  - TAMP\_ACTIVE: Active tamper selected
- an erase mode: for backup registers and Backup SRAM
  - TAMP\_NOERASE: no automatic erase
  - TAMP\_ERASE: automatic erase
- an event mask:
  - TAMP\_NO\_EVT\_MASK: The tamper event must be cleared by software
  - TAMP\_EVT\_MASK: When the event is detected, the tamper is masked and internally cleared. No erase is performed.
- a function pointer: function that is called when the tamper is detected

Below a configuration example to enable two external tampers:

```
static struct stm32_tamp_ext ext_tamp[PLAT_MAX_TAMP_EXT] = {
{
    .id = EXT_TAMP1,                // External tamper 1
    .mode = TAMP_TRIG_ON,          // Tamper trigger event
    .erase = TAMP_NOERASE,        // Backup registers are not erased
    .evt_mask = TAMP_NO_EVT_MASK, // Mask is not set
    .func = NULL,                 // No function
},
{
    .id = EXT_TAMP2,                // External tamper 2
    .mode = TAMP_ACTIVE,          // Active tamper selected
    .erase = TAMP_NOERASE,        // Backup registers and backup SRAM
    .evt_mask = TAMP_NO_EVT_MASK, // Mask is not set
    .func = NULL,                 // No function
},
    TAMP_UNUSED,
}
```

External tampers require to configure the filtering mode (for passive tamper) or active mode selection. The configuration is performed in the same secure OS main security configuration file inside the `init_sec_peripherals` function.

Filter\_conf and active\_conf values must be configured according to the values defined in `stm32_tamp.h`.

Below an example of filtering configuration for passive tamper with pull-up disabled, 2 RTC cycles for precharge, 1 count event for detection (level detection), and a sampling frequency set to 16,384 Hz:

```
uint32_t filter_conf = TAMP_FLTCR(TAMP_FILTER_PULL_UP_DISABLE,
    TAMP_FILTER_DURATION_2_CYCLES,
    TAMP_FILTER_COUNT_1,
    TAMP_FILTER_SAMPLING_16384);
```



Below an example of active tamper filtering configuration.

Based on the external tamper declaration above, tamper 2 is the only one defined as active.

The configuration (for all active tampers) is the following:

- global active filtering enabled
- tamper compared to defined tamp\_out selection
- output charge period set to 2 x CK\_ATPRE \* prescaler clock period, with clock configured to RTCCCKL/2
- tamp\_out selection 2 set to external tamper 1 output

```
uint32_t active_conf = TAMP_ACT(TAMP_ACTIVE_FILTER_ON,
                                TAMP_ACTIVE_ATO_TAMPOUTSEL,
                                TAMP_ACTIVE_APER_1_OUTPUT,
                                TAMP_ACTIVE_CKSEL_DIV_2,
                                TAMP_ACTIVE_ATOSEL_OUT2_(0));
```

uniprocessor

Stable: 27.05.2021 - 13:01 / Revision: 18.05.2021 - 20:27

A quality version of this page, approved on 27 May 2021, was based off this revision.

## Contents

1 Peripheral overview .....	8
1.1 Features .....	8
1.2 Security support .....	8
2 Peripheral usage and associated software .....	9
2.1 Boot time .....	9
2.2 Runtime .....	9
2.2.1 Overview .....	9
2.2.2 Software frameworks .....	9
2.2.3 Peripheral configuration .....	9
2.2.4 Peripheral assignment .....	9
3 References .....	11



---

## 1 Peripheral overview

---

The **BKPSRAM** internal memory is 4 Kbytes wide and is located in the VSW power domain, allowing it to be supplied during Standby low power mode, or to be switched off.

### 1.1 Features

Refer to [STM32MP15 reference manuals](#) for the complete feature list, and to the software components introduced below, to see which features are currently implemented.

### 1.2 Security support

The BKPSRAM is a **secure** peripheral (under ETZPC control).





## 2 Peripheral usage and associated software

### 2.1 Boot time

The BKPSRAM internal memory is not used during a cold boot or a wake up from Standby with DDR OFF.

The BKPSRAM internal memory is used by the runtime secure monitor (from the FSBL or the OP-TEE secure OS) during wake-up from Standby low power mode with the DDR in Self-Refresh mode. In that case, the BKPSRAM internal memory contains the secure context that has to be restored before jumping back to Linux execution, in DDR.

### 2.2 Runtime

#### 2.2.1 Overview

The BKPSRAM peripheral can be allocated to:

- the Arm® Cortex®-A7 secure to be used under PSCI<sup>[1]</sup> secure services (from the FSBL or OP-TEE secure monitor) to save the secure context before entering Standby low power mode with DDR in Self-Refresh mode. This is the default assignment.

or

- the Cortex-A7 non-secure to be used under Linux® as reserved memory, for instance.

#### Warning

Default OpenSTLinux delivery prevents to define BKPSRAM as non-secure. This requires to modify TF-A source code with one of the following strategies:

- set BKPSRAM as non-secure and degrade low power modes support, removing Standby mode

or

- manage on-the-fly secure/non-secure switch of the BKPSRAM in the secure monitor for sequential usage for Standby management and Linux kernel reserved memory

#### 2.2.2 Software frameworks

Domain	Peripheral	Software components		Comment
OP-TEE	Linux	STM32Cube		
Core/RAM	BKPSRAM	TF-A overview	Linux reserved memory	

#### 2.2.3 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 done via the STM32CubeMX tool for all internal peripherals, and can then be manually be completed (particularly for external peripherals), according to the information given in the corresponding software framework article.

#### 2.2.4 Peripheral assignment

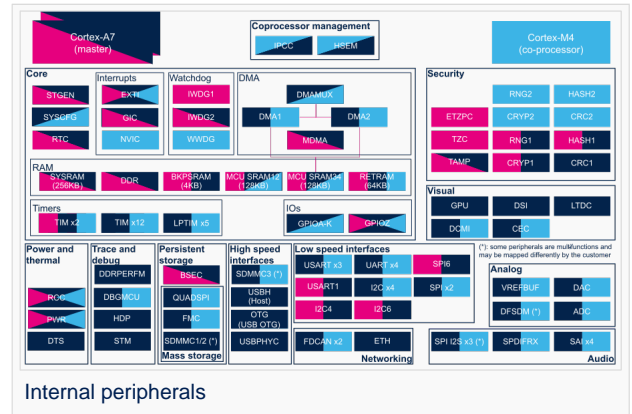
**Check boxes** illustrate the possible peripheral allocations supported by STM32 MPU Embedded Software:

- 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 possibilities might be described in STM32MP15 reference manuals .



Internal peripherals

Domain	Periphera	Runtime allocation		Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)	
Core/RAM	BKPSRAM M	BKPSRAM M		Assignment (single choice)



## 3 References

- [http://infocenter.arm.com/help/topic/com.arm.doc.den0022d/Power\\_State\\_Coordination\\_Interface\\_PDD\\_v1\\_1\\_DEN0022D.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.den0022d/Power_State_Coordination_Interface_PDD_v1_1_DEN0022D.pdf)

Stable: 18.11.2021 - 08:42 / Revision: 16.11.2021 - 15:13

A quality version of this page, approved on *18 November 2021*, was based off this revision.

### Contents

1 Article purpose .....	12
2 Overview .....	13
3 Backup registers usage .....	14
3.1 Boot counter feature .....	14
3.2 Boot mode selection feature .....	14
3.3 DDR and CPU wake up management feature .....	15
3.4 Cortex-M4 wake up feature .....	15
3.5 Cortex-M4 management feature .....	15
4 Memory mapping .....	16
5 References .....	19



## 1 Article purpose

---

The purpose of this article is to explain how the TAMP backup registers are used by STM32MPU Embedded Software.



---

## 2 Overview

---

The STM32MP15 embeds 32 backup registers of 32 bits. A programmable border allows to split those backup registers into a secure and a non-secure group<sup>a</sup>.



## 3 Backup registers usage

This paragraph explains the default backup registers usage by the ROM code and STM32MPU Embedded Software distribution . Then, the next chapter shows the backup register mapping used to fulfill those needs.

### Warning

It is important to notice that the backup registers are erased when a tamper detection occurs in TAMP internal peripheral

### Information

Backup register(s) may be used for another purpose by the application when the feature(s) listed below is/are not used by the said application

### 3.1 Boot counter feature

The `BOOT_COUNTER` is used by U-Boot to detect boot failures between its execution and before the complete Linux application initialization :

- It is incremented by U-Boot
- It is reset by the application

So, if U-Boot reads a non null value after a reset, this means that something went wrong at boot time and can be used to enter a fail safe mode.

### 3.2 Boot mode selection feature

The `BOOT_MODE` register is used to propagate boot mode information from one component to the next boot stage, on cold boot or after a reset:

- The ROM code executes a serial boot if `BOOT_MODE[7:0]` is equal to `0xFF`, as stated in the ROM code boot device selection strategy. In that case, the backup register is reset by the ROM code before proceeding with the serial boot mode. Other values are ignored by the ROM code.
- TF-A gets the selected boot device from the ROM code context in SYSRAM and writes it into `BOOT_MODE[15:8]` for U-Boot [1].
- U-Boot uses the `BOOT_MODE` register to get TF-A and Linux kernel (as explained in the next bullet) information<sup>[1]</sup> in order to select the wished boot mode (among "NORMAL", "STM32PROG", ...) and build the appropriate "chosen" node in the device tree given to the Linux kernel, as explained in "Runtime configuration" paragraph in [Documentation/devicetree/usage-model.rst](#) .
- The Linux kernel can force a reboot-mode writing into the `BOOT_MODE` register. This writing is done via the "reboot" Linux command, that is configured via the compatible "syscon-reboot-mode" in the device tree <sup>[2]</sup>.



### 3.3 DDR and CPU wake up management feature

The `MAGIC_NUMBER` and `BRANCH_ADDRESS` registers allow to control the DDR initialization and the Arm<sup>®</sup> Cortex<sup>®</sup>-A7 CPU cores behaviors on system transitions:

- On cold boot, the ROM code sets the `MAGIC_NUMBER` register to 0x0. When the FSBL TF-A BL2 reads a value different from 0xCA7FACE0 (like 0x0) in `MAGIC_NUMBER`, it performs a complete DDR initialization before jumping to the SSBL (U-Boot).
- Before entering in Standby with DDR in self-refresh low power mode, the PSCI framework writes the return address where the Arm<sup>®</sup> Cortex<sup>®</sup>-A7 core 0 should branch to on wake up into `BRANCH_ADDRESS` register and it sets the `MAGIC_NUMBER` register to 0xCA7FACE0.
- On wakeup from Standby when the FSBL reads the value 0xCA7FACE0 from the `MAGIC_NUMBER` register, it has to perform a partial DDR initialization to exit Self-Refresh, before branching the Arm<sup>®</sup> Cortex<sup>®</sup>-A7 core 0 non-secure execution back to the address given by the `BRANCH_ADDRESS` register.
- On startup, the Linux<sup>®</sup> kernel starts to run on the Arm<sup>®</sup> Cortex<sup>®</sup>-A7 core 0 and it uses the PSCI framework to write the address where the core 1 has to jump in `BRANCH_ADDRESS` register and to set `MAGIC_NUMBER` register to 0xCA7FACE1. Those values are then interpreted by the ROM code, as explained in [secondary core boot](#) paragraph.

### 3.4 Cortex-M4 wake up feature

The ROM code can autonomously start the Arm Cortex<sup>®</sup>-M4 with a firmware kept in RETRAM after a Standby low power mode period:

- On wake up from Standby low power mode, the `M4_WAKEUP_AREA_HASH`, `M4_WAKEUP_AREA_LENGTH` and `M4_WAKEUP_AREA_START` registers are used by the ROM code to perform an integrity check on the Arm Cortex<sup>®</sup>-M4 firmware stored in RETRAM. If successful, the ROM code starts the Cortex-M4 execution.
- Before entering in Standby mode, it is the application responsibility to set the above registers: the application has to compute and store the SHA-256 hash that will be used by the ROM code on wake up. The hash is computed starting from the address stored in `M4_WAKEUP_AREA_START`, for the length defined in `M4_WAKEUP_AREA_LENGTH` and it is stored in `M4_WAKEUP_AREA_HASH`.
- On top of this branching mechanism, the Cortex-M4 coprocessor may need to access to some system resources that are secured (and so, not accessible) by default. To achieve this, the RCC registers `RCC_TZCR`, `EXTI_TZENR1`, `EXTI_TZENR2` and `EXTI_TZENR3` are respectively saved into `WAKEUP_SEC`, `M4_SECURITY_PERIMETER_EXTI1`, `M4_SECURITY_PERIMETER_EXTI2` and `M4_SECURITY_PERIMETER_EXTI3` by the application before entering into Standby mode and restored by the ROM code on wake up.

#### Warning

This feature is enabled in the ROM code but not implemented in STM32 MPU Embedded Software

### 3.5 Cortex-M4 management feature

Two registers are used to control the Arm Cortex<sup>®</sup>-M4 coprocessor from the Arm Cortex<sup>®</sup>-A7:

- The `COPRO_RSC_TBL_ADDRESS` register owns the Coprocessor resource table base address: it is read by the Linux Kernel to retrieve the resource table when the Cortex-M4 firmware is loaded by U-Boot or OP-TEE.
- The `CORTEX_M_STATE` register reflects the Cortex-M4 state : it is written by U-Boot or OP-TEE when loading the firmware, it can be updated by the Cortex-M4 firmware at runtime and it is read by the Linux Kernel to know the coprocessor state.



## 4 Memory mapping

The table below shows the backup register mapping used by STM32MPU Embedded Software.  
The TAMP backup register base address is 0x5C00A100, corresponding to TAMP\_BKP0R.

TAMP register	Security <sup>a</sup>	ROM / software register name	Comment
TAMP_BKP3 1R	Non-secure	M4_WAKEUP_AREA_HASH	SHA-256 value, see <a href="#">Cortex-M4 wake up feature</a>
TAMP_BKP3 0R	Non-secure		
TAMP_BKP2 9R	Non-secure		
TAMP_BKP2 8R	Non-secure		
TAMP_BKP2 7R	Non-secure		
TAMP_BKP2 6R	Non-secure		
TAMP_BKP2 5R	Non-secure		
TAMP_BKP2 4R	Non-secure		
TAMP_BKP2 3R	Non-secure	M4_WAKEUP_AREA_LENGTH	See <a href="#">Cortex-M4 wake up feature</a>
TAMP_BKP2 2R	Non-secure	M4_WAKEUP_AREA_START	See <a href="#">Cortex-M4 wake up feature</a>
TAMP_BKP2 1R	Non-secure	BOOT_COUNTER	See <a href="#">Boot counter feature</a>
TAMP_BKP2 0R	Non-secure	BOOT_MODE	See <a href="#">Boot mode selection feature</a>
TAMP_BKP1 9R	Non-secure		(Reserved for future use)
TAMP_BKP1 8R	Non-secure	CORTEX_M_STATE	See <a href="#">Cortex-M4 management feature</a>
TAMP_BKP1 7R	Non-secure	COPRO_RSC_TBL_ADDRESSES	See <a href="#">Cortex-M4 management feature</a>





TAMP register	Security <sup>a</sup>	ROM / software register name	Comment
TAMP_BKP16R	Non-secure		(Reserved for future use)
TAMP_BKP15R	Non-secure		(Reserved for future use)
TAMP_BKP14R	Non-secure		(Reserved for future use)
TAMP_BKP13R	Non-secure		(Reserved for future use)
TAMP_BKP12R	Non-secure		(Reserved for future use)
TAMP_BKP11R	Non-secure		(Reserved for future use)
TAMP_BKP10R	Non-secure		(Reserved for future use)
TAMP_BKP9R	Secure		(Reserved for future use)
TAMP_BKP8R	Secure		(Reserved for future use)
TAMP_BKP7R	Secure		(Reserved for future use)
TAMP_BKP6R	Secure		(Reserved for future use)
TAMP_BKP5R	Secure	BRANCH_ADDRESS	See <a href="#">DDR and CPU wake up management feature</a>
TAMP_BKP4R	Secure	MAGIC_NUMBER	See <a href="#">DDR and CPU wake up management feature</a>
TAMP_BKP3R	Secure	M4_SECURITY_PERIMETER_EXTI3	See <a href="#">Cortex-M4 wake up feature</a>
TAMP_BKP2R	Secure	M4_SECURITY_PERIMETER_EXTI2	See <a href="#">Cortex-M4 wake up feature</a>
TAMP_BKP1R	Secure	M4_SECURITY_PERIMETER_EXTI1	See <a href="#">Cortex-M4 wake up feature</a>
TAMP_BKP0R	Secure	WAKEUP_SEC	See <a href="#">Cortex-M4 wake up feature</a>



---

a: the security border is configured by the Secure OS (look for "zone" in `plat/st/stm32mp1/sp_min/sp_min_setup.c` for TF-A SP-MIN or in `core/arch/arm/plat-stm32mp1/main.c` for OP-TEE), so the source code has to be modified if a different border is needed.



## 5 References

- 1.01.1 arch/arm/mach-stm32mp/include/mach/stm32.h
- arch/arm/boot/dts/stm32mp151.dtsi

Das U-Boot -- the Universal Boot Loader (see [U-Boot\\_overview](#))

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](#).




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

Reference	Name	Link	Version
<b>Application notes</b>			
AN4803	High-speed SI simulations using IBIS and board-level simulations using HyperLynx® SI on STM32 MCUs and MPUs	<a href="#">AN4803.pdf</a>	v2.0
AN5027	Interfacing PDM digital microphones using STM32 MCUs and MPUs	<a href="#">AN5027.pdf</a>	v2.0
AN5031	Getting started with STM32MP15 Series hardware development	<a href="#">AN5031.pdf</a>	v3.0
AN5036	Thermal management guidelines for STM32 applications	<a href="#">AN5036.pdf</a>	v3.0
AN5109	STM32MP1 Series using low-power modes	<a href="#">AN5109.pdf</a>	v4.0
AN5122	STM32MP1 Series DDR memory routing guidelines	<a href="#">AN5122.pdf</a>	v3.0
AN5168	STM32MP1 series DDR configuration	<a href="#">AN5168.pdf</a>	 v2.0
AN5225	USB Type-C™ Power Delivery using STM32xx Series MCUs and STM32xxx Series MPUs	<a href="#">AN5225.pdf</a>	 v5.0










Reference	Name	Link	Version
<b>Application notes</b>			
AN5253	Migration of microcontroller applications from STM32F4x9 lines to STM32MP151, STM32MP153 and STM32MP157 lines microprocessor	<a href="#">AN5253.pdf</a>	v1.0
AN5256	STM32MP151, STM32MP153 and STM32MP157 discrete power supply hardware integration	<a href="#">AN5256.pdf</a>	v2.0
AN5260	STM32MP151/153/157 MPU lines and STPMIC1B integration on a battery powered application	<a href="#">AN5260.pdf</a>	v2.0
AN5275	USB DFU/USART protocols used in STM32MP1 Series bootloaders	<a href="#">AN5275.pdf</a>	v1.0
AN5284	STM32MP1 series system power consumption	<a href="#">AN5284.pdf</a>	v1.0
AN5348	FDCAN peripheral on STM32 devices	<a href="#">AN5348.pdf</a>	v1.0
AN5431	The STPMIC1 PCB layout guidelines	<a href="#">AN5431.pdf</a>	v1.0
AN5438	STM32MP1 Series lifetime estimates	<a href="#">AN5438.pdf</a>	v1.0
AN5510	Overview of the secure secret provisioning (SSP) on STM32MP1 Series	<a href="#">AN5510.pdf</a>	v1.0
<b>Datasheets<sup>[1]</sup></b>			
DS12505	STM32MP157C/F datasheet (secure)	<a href="#">DS12505.pdf</a>	 v6.0
DS12504	STM32MP157A/D datasheet (basic)	<a href="#">DS12504.pdf</a>	 v6.0
DS12503	STM32MP153C/F datasheet (secure)	<a href="#">DS12503.pdf</a>	 v6.0
DS12502	STM32MP153A/D datasheet (basic)	<a href="#">DS12502.pdf</a>	 v6.0
DS12501	STM32MP151C/F datasheet (secure)	<a href="#">DS12501.pdf</a>	 v6.0
DS12500	STM32MP151A/D datasheet (basic)	<a href="#">DS12500.pdf</a>	 v6.0
DS12792	STPMIC1 datasheet	<a href="#">DS12792.pdf</a>	 v8.0
<b>Errata sheets</b>			



Reference	Name	Link	Version
<b>Application notes</b>			
ES0438	STM32MP15xx device errata	<a href="#">ES0438.pdf</a>	v6.0
<b>Reference manuals<sup>[1]</sup></b>			
RM0436	STM32MP157 reference manual (STM32MP157xxx advanced Arm <sup>®</sup> -based 32-bit MPUs)	<a href="#">RM0436.pdf</a>	v5.0
RM0442	STM32MP153 reference manual (STM32MP153xxx advanced Arm <sup>®</sup> -based 32-bit MPUs)	<a href="#">RM0442.pdf</a>	v5.0
RM0441	STM32MP151 reference manual (STM32MP151xxx advanced Arm <sup>®</sup> -based 32-bit MPUs)	<a href="#">RM0441.pdf</a>	v5.0
<b>Boards schematics</b>			
MB1262 schematics	STM32MP157C-EV1 motherboard schematics MB1262-C01 board schematic (Evaluation board)	<a href="#">MB1262-C01.pdf</a>	v1.0
MB1263 schematics	STM32MP157F-EV1 daughterboard schematics MB1263-C04 board schematic (Evaluation board)	<a href="#">MB1263-C04.pdf</a>	v4.0
MB1230 schematics	DSI 720p LCD display daughterboard schematics MB1230-C board schematic (Evaluation board)	<a href="#">MB1230-C.pdf</a>	v1.1
MB1379 schematics	Camera daughterboard schematics MB1379-A01 board schematic (Evaluation board)	<a href="#">MB1379-A01.pdf</a>	v1.0
MB1272 schematics	STM32MP157x-DKx motherboard schematics MB1272-DK2-C01 board schematic (Discovery kit)	<a href="#">MB1272-C01.pdf</a>	v1.0
MB1407 schematics	STM32MP157x-DKx daughterboard schematics MB1407-LCD-C01 board schematic (Discovery kit)	<a href="#">MB1407-C01.pdf</a>	v1.0
<b>Boards user manuals</b>			
UM2535	STM32MP157x-EV1 evaluation board user manual	<a href="#">UM2535.pdf</a>	v2.0
UM2534	STM32MP157x-DKx discovery board user manual	<a href="#">UM2534.pdf</a>	v1.0
<b>Tools user manuals</b>			
UM2563	STM32CubeIDE installation guide	<a href="#">UM2563.pdf</a>	 v3.0



Reference	Name	Link	Version
<b>Application notes</b>			
UM2579	Migration guide from System Workbench to STM32CubeIDE	UM2579.pdf	v1.0
UM2553	STM32CubeIDE quick start guide	UM2553.pdf	 v3.0
AN5360	Getting started with projects based on the STM32MP1 Series in STM32CubeIDE	AN5360.pdf	v1.0
UM2609	STM32CubeIDE user guide	UM2609.pdf	 v5.0
UM1718	STM32CubeMX user manual	UM1718.pdf	 v36.0
UM2237	STM32CubeProgrammer tool user manual	UM2237.pdf	 v17.0
UM2238	STM32 Trusted Package Creator tool user manual	UM2238.pdf	 v9.0
UM2542	STM32 Series Key Generator tool user manual	UM2542.pdf	 v2.0
UM2543	STM32 Series Signing tool user manual	UM2543.pdf	 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	<a href="#">STM32MP15 resources - v3.0.0</a> page for the previous v3 ecosystem release
STM32MP15-Ecosystem-v2.1.0	<a href="#">STM32MP15 resources - v2.1.0</a> page for the v2 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v2.0.0	<a href="#">STM32MP15 resources - v2.0.0</a> page for the v2 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.2.0	<a href="#">STM32MP15 resources - v1.2.0</a> page for the v1 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.1.0	<a href="#">STM32MP15 resources - v1.1.0</a> page for the v1 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.0.0	<a href="#">STM32MP15 resources - v1.0.0</a> page for the v1 ecosystem releases (in archived wiki)

### USB port or connector

#### Universal Synchronous/Asynchronous Receiver/Transmitter

Stable: 15.09.2021 - 06:46 / Revision: 15.09.2021 - 06:45

A quality version of this page, approved on *15 September 2021*, was based off this revision.

## Contents

1 Article purpose .....	24
2 DT bindings documentation .....	25
3 DT configuration .....	26
3.1 DT configuration (STM32 level) .....	26
3.2 DT configuration (board level) .....	26
3.2.1 STM32MP1 TAMP node append .....	26
3.2.2 STM32MP1 TAMP node append (bootloader specific) .....	27
4 How to configure the DT using STM32CubeMX .....	28
5 References .....	29



## 1 Article purpose

---

This article explains how to configure TAMP internal peripheral.

This article describes the TAMP configuration performed using the device tree mechanism, which provides an hardware description of the TAMP peripheral.





## 2 DT bindings documentation

---

The following binding-related documentation explains how to write device tree files for TAMP:

- TF-A: Tamper related part: [tf-a/docs/devicetree/bindings/soc/st,stm32-tamp.txt<sup>\[1\]</sup>](#)
- Linux Kernel: Backup register management: [Documentation/devicetree/bindings/power/reset/syscon-reboot-mode.yaml<sup>\[2\]</sup>](#)



## 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 STM32MP1 TAMP node is located in the file *stm32mp151.dtsi*<sup>[3]</sup> (see [Device tree](#) for further explanation). TAMP is used in the Linux context to access to the backup registers.

```

/ {
...
    soc {
...
        tamp: tamp@5c00a000 {
            compatible = "simple-bus", "syscon", "simple-mfd";
            reg = <0x5c00a000 0x400>;

            reboot-mode {
                compatible = "syscon-reboot-mode";
                offset = <0x150>; /* reg20 */
                mask = <0xff>;
                mode-normal = <0>;
                mode-fastboot = <0x1>;
                mode-recovery = <0x2>;
                mode-stm32cubeprogrammer = <0x3>;
                mode-ums_mmc0 = <0x10>;
                mode-ums_mmc1 = <0x11>;
                mode-ums_mmc2 = <0x12>;
            };
        };
...
    };
};

```

### 3.2 DT configuration (board level)

#### 3.2.1 STM32MP1 TAMP node append

The board definition in the device tree may include some additional board-specific pin control management.

#### Warning

By default, no pinctrl description is required for tamper, the associated GPIO pin is reserved for the tamper usage, preempting the current GPIO pin configuration. Despite of the hardware management, It may be better to describe the pinctrl to avoid any software double pin request.



```

&tamp {
    pinctrl-0 = <&tamp1_pins_a>;           // Must be defined in the pinctrl
    corresponding to the board
    wakeup-source;                       // Enable the tamper as wake up
    source
};

```

### 3.2.2 STM32MP1 TAMP node append (bootloader specific)

The bootloader-specific STM32MP1 TAMP node append data is located in the file *stm32mp151.dtsi*<sup>[4]</sup> for TF-A (see Device tree for further explanation).

```

tamp: tamp@5c00a000 {
    compatible = "st,stm32-tamp", "simple-bus", "syscon", "simple-mfd";
    reg = <0x5c00a000 0x400>;
    secure-interrupts = <GIC_SPI 197 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&rcc RTCAPB>;
};

```



---

## 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 may not support all the properties described in the documents listed in [DT bindings documentation](#) above. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties that are preserved from one generation to another. Refer to the [STM32CubeMX user manual](#) for further information.



## 5 References

---

Please refer to the following links for additional information:

- [docs/devicetree/bindings/soc/st,stm32-tamp.txt](#) TF-A TAMP binding information file
- [Documentation/devicetree/bindings/power/reset/syscon-reboot-mode.yaml](#)
- [arch/arm/boot/dts/stm32mp151.dtsi](#) : STM32MP151 kernel device tree files
- [fdts/stm32mp151.dtsi](#) STM32MP151 TF-A device tree files