STM32MP15 Tamper configuration

Contents

Stable: 28.06.2021 - 07:23 / Revision: 28.06.2021 - 07:22

Stable: 28.06.2021 - 01.2.0 / Revision: 28.06.2021 - 01.22

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

Contents

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

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

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
are not erased
                .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 RTCCKL/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));
```

Operating System

Real Time Clock

Low Speed External oscillator (STM32 clock source)

High Speed External oscillator (STM32 clock source)

Open Portable Trusted Execution Environment

Tamper

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.)

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

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 [1] 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 assignement.

or

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

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

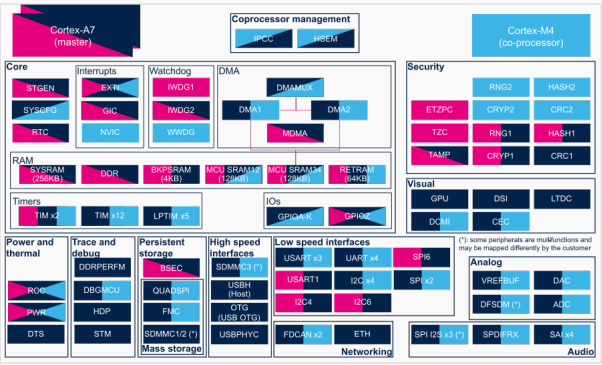
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration by itself can be done via the STM32CubeMX tool for all internal peripherals, and can then be manually be completed (particularly for external peripherals), according to the information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

3 References

- http://infocenter.arm.com/help/topic/com.arm.doc.den0022d/Power_State_Coordination_Interface_PDD_v1_1_DEN0022D.pdf

Power State Coordination Interface

Doubledata rate (memory domain)

Trusted Firmware for Arm® Cortex®-A

Open Portable Trusted Execution Environment

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.)

Stable: 07.01.2021 - 14:51 / Revision: 07.01.2021 - 14:50

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

Contents

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.
By default, the ROM code defines the 10 first backup registers as secure, but this secure/non-secure border can be changed later on from the secure context.

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.

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

3.1 At boot time

- **Non-secure backup registers** are used:
  - during a cold boot:
    - by U-Boot to initialize the **boot counter**, that should be reset later on by the application.
  - after a reset:
    - by the ROM code to get an eventual **forced boot mode** that was set before reset. Notice that the ROM code is only interprating the value 0xFF to trigger a serial boot, as shown 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 but may be interprated by U-Boot, as described just below.
    - by U-Boot to get an eventual **forced boot mode** that was set before reset. This can be useful to set U-Boot in programmer mode after a reboot, for instance.
    - by U-Boot to increment the **boot counter** and perform given actions if a predefined number of successive boots is reached, due to cyclic resets before the application is alive (and clears the counter).
- **Secure backup registers** are used:
  - to tell to the FSBL (TF-A or U-Boot SPL) how to behave:
    - on cold boot, the ROM code sets the **magic number** to 0x0: this value tells to the FSBL that a complete DDR initialization is needed before jumping to the SSBL (U-Boot).
    - on wakeup from Standby with DDR in self-refresh low power mode, if the **magic number** == 0xCA7FACE0 then the FSBL performs a partial DDR initialization to exit Self-Refresh then it branches the Arm®Cortex®-A7 core 0 non-secure execution to the given **branch address** (in Linux® kernel, that was set during secure context saving before the Standby low power mode entering).
  - by Linux® kernel on Arm®Cortex®-A7 core 0 (via a PSCI secure service) to tell to the ROM code how to start Arm® Cortex®-A7 core 1 (and enable the SMP mode): when Arm®Cortex®-A7 core 1 non-secure sees the **magic number** == 0xCA7FACE1 then it jumps to the given **branch address**.
  - by the ROM code during wakeup from Standby low power mode to recover the Cortex®-M4 firmware **integrity check value** and compare it to the one computed on RETRAM before starting the Cortex®-M4 again.

Notice: the ROM code knows if Cortex®-A7 and/or Cortex®-M4 have to be restarted after Standby thanks to RCC_MP_BOOTCR register, so the backup registers are not used here.

3.2 At runtime

- Non secure backup registers
  - own the **boot counter** and should be reset by the application after a successful startup.

- are used to store Cortex®-M4 retention firmware **integrity check value** before going to Standby mode, if the Cortex®-M4 needs to be started on wakeup from Standby mode by the ROM code.

- Secure backup registers
  - are used by secure services to store:
    - Arm®Cortex®-A7 core 0 **branch address** that are used by the ROM code on wakeup from Standby mode.
    - Arm®Cortex®-M4 **security perimeter** that is restored by the ROM code before starting the Cortex®-M4 on wakeup from Standby.

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	ROM / software register name	Comment
TAMP_BKP31R	Non-secure		
TAMP_BKP30R	Non-secure		
TAMP_BKP29R	Non-secure	M4_WAKEUP_AREA_HASH	This register can be used to store a SHA-256 value computed on M4_WAKEUP_AREA_LENGTH bytes in RETRAM starting from M4_WAKEUP_AREA_START, before entering in low power Standby mode. This allows the ROM code to perform an integrity check on wakeup before starting the coprocessor.
	No		

TAMP register	Security	ROM / software register name	Comment
TAMP_BKP28R	non-secure		
TAMP_BKP27R	Non-secure		
TAMP_BKP26R	Non-secure		
TAMP_BKP25R	Non-secure		

TAMP register	Security	ROM / software register name	Comment
TAMP_BKP24R	Non-secure		
TAMP_BKP23R	Non-secure	M4_WAKEUP_AREA_LENGTH	Amount of bytes hashed in RETRAM to compute the integrity check value
TAMP_BKP22R	Non-secure	M4_WAKEUP_AREA_START	Start address in RETRAM from where the integrity check value has to be computed
TAM	Non-s	BOOT_	

TAMP register	Security	ROM / software register name	Comment
P_BKP21R	ecure	COUNTER	Boot counter
TAMP_BKP20R	Non-secure	BOOT_MODE[1]	Boot mode context information
TAMP_BKP19R	Non-secure		(Reserved for future use)
TAMP_BKP18R	Non-secure	CORTEX_M_STATE	Cortex-M state (written by Cortex-M / read by Cortex-A)
TA	Non		

TAMP register	Security	ROM / software register name	Comment
MP_BKP17R	-secure	COPRO_RSC_TBL_ADDRESS	Coprocessor resource table base address
TAMP_BKP16R	Non-secure		(Reserved for future use)
TAMP_BKP15R	Non-secure		(Reserved for future use)
TAMP_BKP14R	Non-secure		(Reserved for future use)
	N		

TAMP register	Security	ROM / software register name	Comment
TAMP_BKP13R	on-secure		(Reserved for future use)
TAMP_BKP12R	Non-secure		(Reserved for future use)
TAMP_BKP11R	Non-secure		(Reserved for future use)
TAMP_	Non-s		

TAMP register	Security	ROM / software register name	Comment
BKP10R	ecure		(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[1]	CPU0 or CPU1 branch address
TAM	Se	MAGIC_	

TAMP register	Security	ROM / software register name	Comment
P_BKP4R	cure	NUMBER[1]	CPU0 or CPU1 boot magic number
TAMP_BKP3R	Secure	M4_SECURITY_PERIMETER_EXTI3	Value of AEIC TZENR3
TAMP_BKP2R	Secure	M4_SECURITY_PERIMETER_EXTI2	Value of AEIC TZENR2
TAMP_BKP1R	Secure	M4_SECURITY_PERIMETER_EXTI1	Value of AEIC TZENR1
TAMP_BKP0R	Secure	WAKEUP_SEC	Wakeup parameters

5 References

- 1.01.11.2 arch/arm/mach-stm32mp/include/mach/stm32.h

Read Only Memory

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

First Stage Boot Loader

Second Stage Boot Loader

Doubledata rate (memory domain)

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

Cortex®

Linux® is a registered trademark of Linus Torvalds.

symetric multiprocessing

Tamper

Secure Hash Algorithm

Stable: 20.07.2021 - 09:02 / Revision: 11.03.2021 - 08:07

A quality version of this page, approved on *20 July 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.

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	NEW v3.0
AN5036	Thermal management guidelines for STM32 applications	AN5036.pdf	v3.0

Reference	Name	Link	Version
Application notes			
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	v1.0
AN5225	USB Type-C™ Power Delivery using STM32xx Series MCUs and STM32xxx Series MPUs	AN5225.pdf	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	NEW 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 v5.0
DS12504	STM32MP157A/D datasheet (basic)	DS12504.pdf	NEW v5.0
DS12503	STM32MP153C/F datasheet (secure)	DS12503.pdf	NEW v5.0
DS1250	STM32MP153A/D datasheet	DS125	

Reference	Name	Link	Version
Application notes			
2	(basic)	02.pdf	NEW v5.0
DS12501	STM32MP151C/F datasheet (secure)	DS12501.pdf	NEW v5.0
DS12500	STM32MP151A/D datasheet (basic)	DS12500.pdf	NEW v5.0
DS12792	STPMIC1 datasheet	DS12792.pdf	NEW v7.0
Errata sheets			
ES0438	STM32MP15xx device errata	ES0438.pdf	NEW v6.0
Reference manuals[1]			
RM0436	STM32MP157 reference manual (STM32MP157xxx advanced Arm®-based 32-bit MPUs)	RM0436.pdf	NEW v5.0
RM0442	STM32MP153 reference manual (STM32MP153xxx advanced Arm®-based 32-bit MPUs)	RM0442.pdf	NEW v5.0
RM0441	STM32MP151 reference manual (STM32MP151xxx advanced Arm®-based 32-bit MPUs)	RM0441.pdf	NEW 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

Reference	Name	Link	Version
Application notes			
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	NEW v2.0
UM2579	Migration guide from System Workbench to STM32CubeIDE	UM2579.pdf	v1.0
UM2553	STM32CubeIDE quick start guide	UM2553.pdf	NEW 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 v3.0
UM1718	STM32CubeMX user manual	UM1718.pdf	NEW v34.0
UM2237	STM32CubeProgrammer tool user manual	UM2237.pdf	NEW v15.0
UM2238	STM32 Trusted Package Creator tool user manual	UM2238.pdf	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-v2.1.0	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)

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: 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

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*"[1]
- Linux Kernel: Backup register management: *Documentation/devicetree/bindings/power/reset/syscon-reboot-mode.yaml*"[2]

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*[3] (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.

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* [4] 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

Device Tree

Tamper

Linux® is a registered trademark of Linus Torvalds.

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.)

Generic Interrupt Controller

Serial Peripheral Interface

Trusted Firmware for Arm® Cortex®-A