Category:Architecture overview

Contents

Stable: 17.06.2020 - 15:26 / Revision: 17.01.2020 - 08:28

Category:Architecture overview

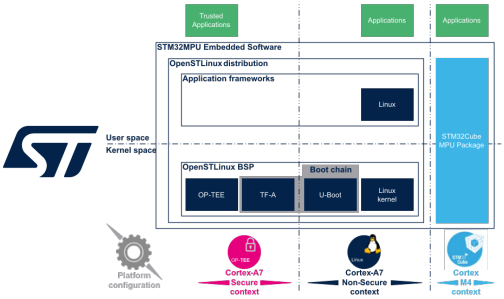
A quality version of this page, approved on *17 June 2020*, was based off this revision.

This category groups together all articles related to the software architecture for the STM32MPU microprocessor devices and their associated boards.

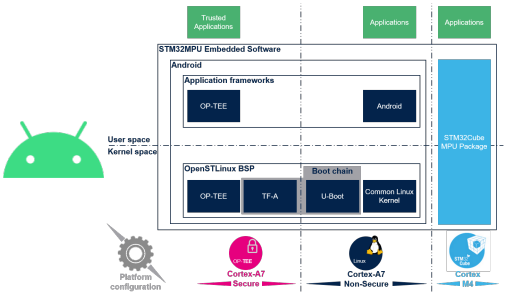
STMicroelectronics embedded software architectures

What are the software architectures - Linux® and RTOS?
What is the role of each kernel component?

Click on the links in the frame below and let you guide!



STM32MPU Embedded Software architecture overview



STM32MPU Embedded Software for Android architecture overview

Pages in category "Architecture overview"

The following 5 pages are in this category, out of 5 total.

- OpenSTLinux architecture overview
- OpenSTLinux BSP architecture overview
- Security overview
- STM32MPU Embedded Software architecture overview
- STM32MPU Embedded Software for Android architecture overview

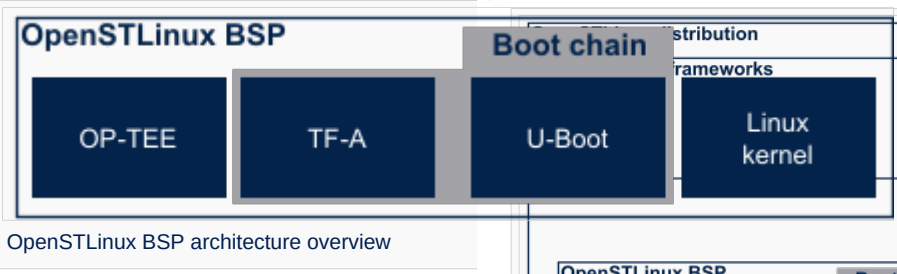
Stable: 15.10.2019 - 09:53 / Revision: 15.10.2019 - 09:41

A quality version of this page, approved on *15 October 2019*, was based off this revision.

The **OpenSTLinux BSP** encompasses the following components:

- The **boot chain** based on **TF-A** and **U-Boot**
- The **OP-TEE secure OS** running on the Cortex-A in secure mode
- The **Linux kernel** running on the Cortex-A in non-secure mode

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



OpenSTLinux BSP architecture overview

Zoom out to OpenSTLinux distribution

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

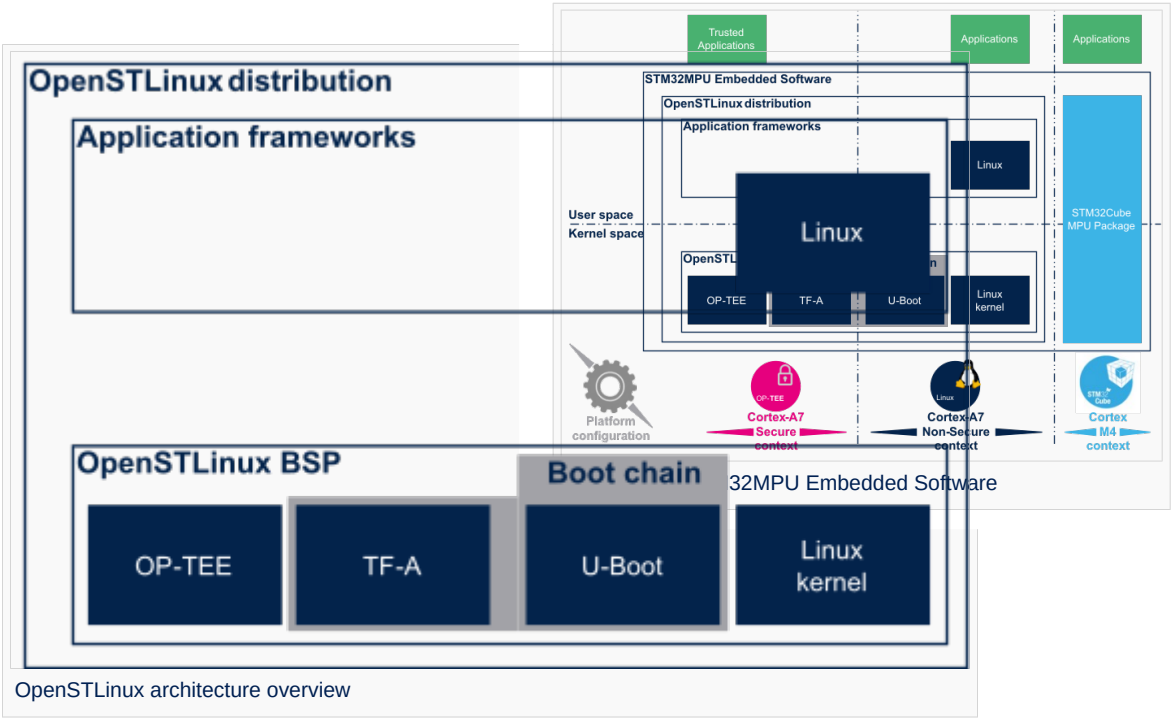
Stable: 26.03.2021 - 13:18 / Revision: 12.03.2021 - 11:26

A quality version of this page, approved on *26 March 2021*, was based off this revision.

The **OpenSTLinux distribution** encompasses the following components:

- The **OpenSTLinux BSP** that offers services, to the application frameworks in the same context, from:
  - The **boot chain** based on **TF-A** and **U-Boot**
  - The **OP-TEE secure OS** running on the Cortex-A in secure mode
  - The **Linux® kernel** running on the Arm® Cortex®-A in non-secure mode
- The **Application frameworks** that rely on the services provided by the OpenSTLinux BSP, to provide particular functionalities (code libraries, APIs, tool sets...) to facilitate the development of software applications:
  - The **Linux application frameworks** (aka Linux middlewares) running on the user space of the Linux OS: e.g. libusb C library for a generic access to USB devices, ALSA user-space bundle for audio functionalities, GStreamer multimedia framework...
  - The **U-Boot application frameworks** (not shown in the diagram), as part of the boot chain: e.g. configuration scripts
- On **OP-TEE** side, the **Trusted Applications (TA)** relies on the OP-TEE core for secrets operations (not visible from the Linux and STM32Cube MPU Package)

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



OpenSTLinux architecture overview

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

Stable: 26.03.2021 - 11:32 / Revision: 12.03.2021 - 11:07

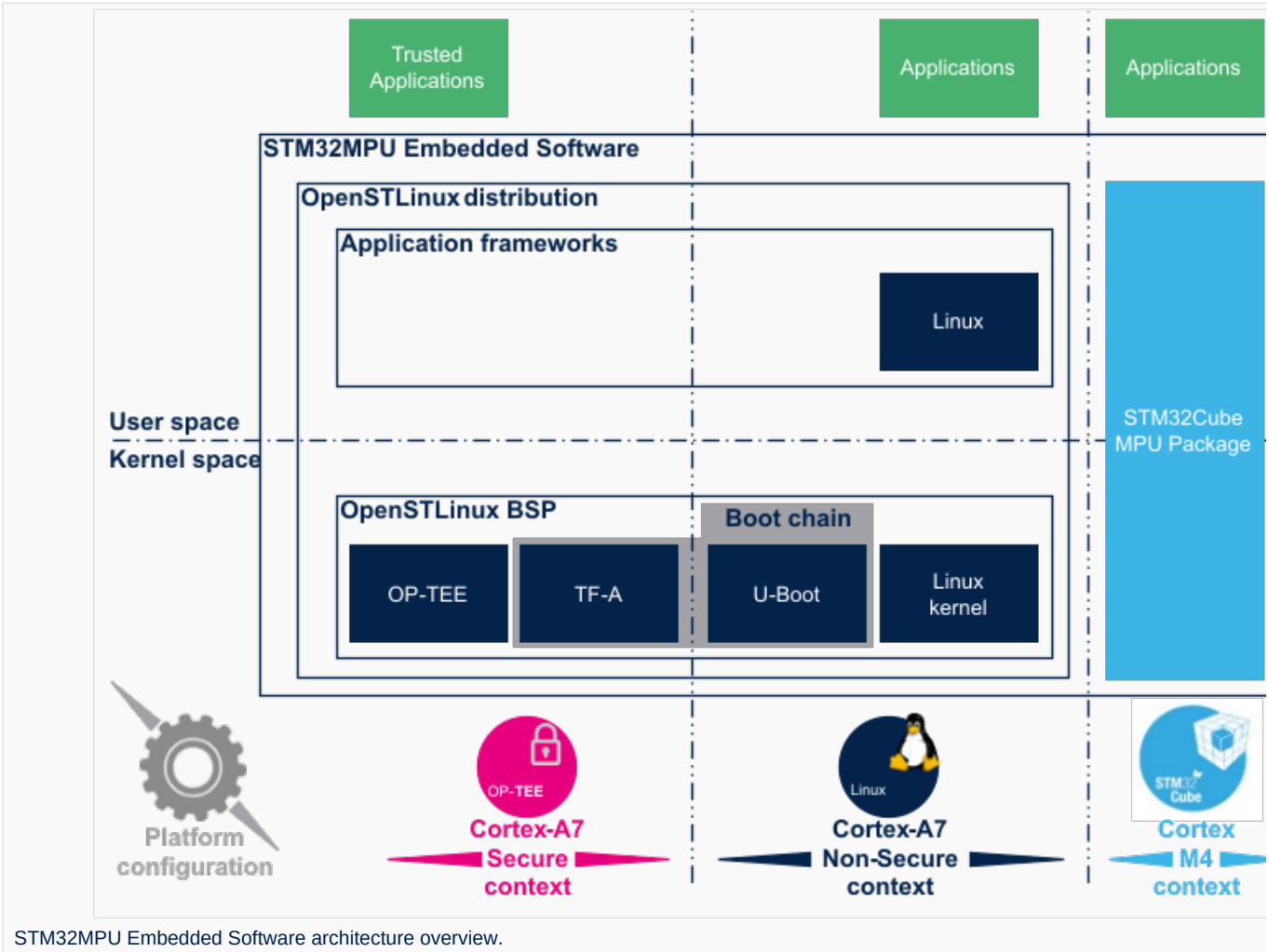
A quality version of this page, approved on *26 March 2021*, was based off this revision.

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 **Linux** side, to run **Applications** that typically interact with the user via the display, the touchscreen, etc.
  - On **OP-TEE** side, the **Trusted Applications (TA)** relies on the OP-TEE core for secrets operations (not visible from the Linux and STM32Cube MPU Package)
- 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.


3rd Party
ST
Community
Legend

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: 16.03.2021 - 16:12 / Revision: 11.03.2021 - 16:27

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

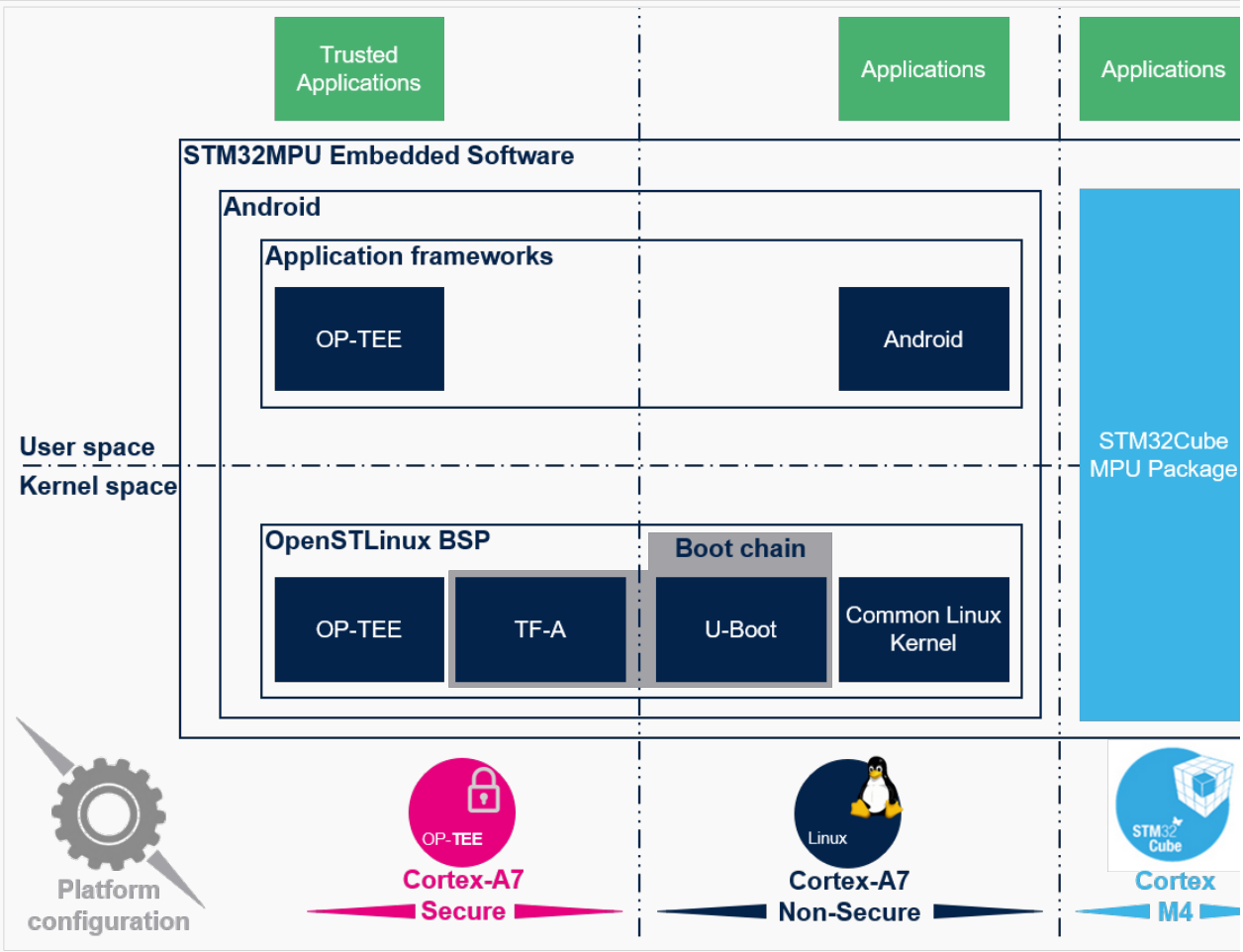
The diagram below shows STM32MPU Embedded Software distribution for Android main components:

- The **STM32MPU distribution for Android™** running on the Arm® Cortex®-A core. It includes:
  - The **OpenSTLinux BSP** consisting of:
    - 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.
  - **Application frameworks** composed of middleware components relying on the BSP and providing a set of APIs:
    - **OP-TEE** APIs to run **Trusted Applications (TA)** that allow manipulating secrets (information not visible from Linux® and from the STM32Cube MPU Package)
    - **Android** APIs to run **Applications** that typically interact with the user via a display or a touchscreen.
- The **STM32Cube MPU Package**, running on the Arm® Cortex®-M. As for STM32 MCUs, it is based on HAL drivers and middleware components and completed with a coprocessor management module.

Warning

The STM32MPU distribution for Android™ is not yet available in the v3 ecosystem releases: please refer to the STM32 MPU Embedded Software for Android architecture overview page for the v2 ecosystem releases (in archived wiki).

The figure below provides an overview of the STM32MPU Embedded Software architecture. Click a sublevel block to jump to the corresponding article.



STM32MPU Embedded Software architecture overview

1 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 usage of open source software and contributes to open source software communities. 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 until it is merged in the targeted repository (see STM32MP1 Distribution Package for Android).

2 References

- https://en.wikipedia.org/wiki/Open-source_software

Stable: 06.04.2021 - 08:21 / Revision: 06.04.2021 - 08:20

A quality version of this page, approved on *6 April 2021*, was based off this revision.

Contents

1 Article purpose

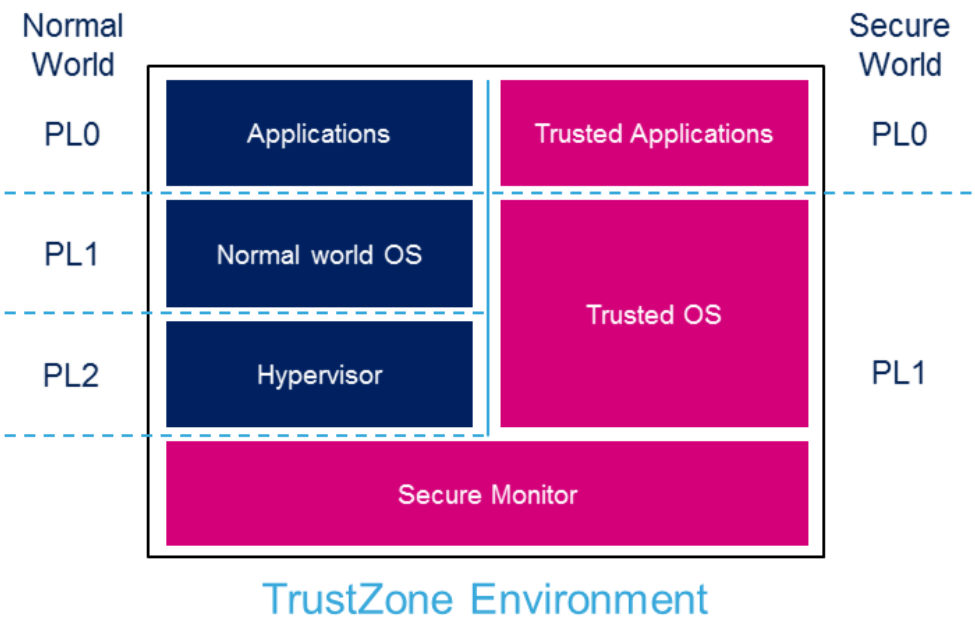
The purpose of this article is to explain how to secure an STM32 MPU-based platform thanks to several hardware mechanisms, and to briefly introduce the software components responsible for the secure configuration.

2 Introduction

The STM32 MPU is based on the Arm® Cortex®-A core, which is using the Arm® TrustZone[1] architecture that enables context isolation: the **normal world** holds the applications whereas the **secure world** isolates all the trusted applications and core secure services so that they can safely manipulate platform secret data. The MPU includes Firewall mechanisms that allow the secure world to forbid read/write accesses from the normal world to given peripherals.

Arm-v7A defines PL0, PL1 and PL2 privilege levels:

- PL0 is the lowest software execution level (unprivileged calls allowed for applications).
- PL1 is the execution level for the OS.
- PL1 (secure) is also the privilege level for secure monitor execution, to switch from the secure to the normal world.
- PL2 is dedicated to the hypervisor (only non-secure).



The **normal world** is used to run rich OSs such as Linux Kernel and its applications framework.
The **secure world** runs a secure monitor with minimal services (i.e. TF-A SP_MIN) or a TEE as secure OS (i.e. OP-TEE OS).

The secure boot is a key feature of this multiple execution contexts environment. It allows the boot chain to be authenticated by the ROM code as well as the authentication of the components that are launched in the secure and normal worlds.

The TrustZone environment is a complete system solution that is not limited to the Cortex context. It provides a bus and peripheral infrastructure to the MPU in order to ensure that the secure world relies on a completely secured pipe when it controls a secure peripheral. The assignment of the peripherals to a given world is done thanks to a firewall mechanism, which is set up during the secure world initialization.

Dedicated secure and normal contexts also impact the debugging facilities: depending on the targeted user, the debug can be opened to both worlds (e.g. for a secure aware developer), to normal world only (for a Linux® developer) or completely closed (for the end user). This is achieved by configuring the Debug control.

Some internal or external peripherals can be used by the secure world to support cryptographic operations. Refer to security peripherals for an introduction.

3 Secure boot

The secure boot is essential to ensure the integrity and security of the platform at runtime.

The STM32 MPU trusted boot chain is designed to guarantee such a secure boot sequence.
It performs the following tasks:

- Configuration of the platform firewall, which is the foundation for a safe execution of the platform
- Configuration of the platform debug capabilities
- Verification of the integrity (thanks to a hash algorithm) and authentication (using asymmetric cryptography algorithms) of the started software components, including the Secure and non-secure worlds.

TF-A is the recommended open source bootloader. Its implementation supports the trusted boot and peripheral access control with firewall.

3.1 STM32MP1 secure boot

The STM32MP1 secure boot implementation is described in :

- the STM32MP15 ROM code secure boot article for the ROM code authentication process based on STM32 header.
- the TF-A BL2 trusted boot management using FIP authentication mechanism.
- the coprocessor authentication.

4 Firewall

MPU firewalls comprise access filters for MPU peripherals and subsystems memory mapped interfaces, internal RAMs/ROMs and external memory (DDR). Depending on the assignment, an MPU interface can be dedicated or shared between several hardware execution context(s).

4.1 STM32MP1 firewall

- ETZPC:
  - assigns access rights to MPU peripherals from Cortex-A7 contexts (secure or normal) and Cortex-M4 context.
  - assigns access rights to internal ROM/RAM from Cortex-A7 and Cortex-M4.
- TZC: assigns access rights to DDR regions.
- RCC: can restrict the access of some of its registers to the secure execution context.
- PWR: can restrict the access of some of its registers to the secure execution context.
- BSEC: The OTP memory can be fused to restrict the access to some of its content to the secure execution context.
- RTC: This MPU interface can restrict some of its interface registers to the secure execution context.
- GPIO: GPIO bank Z can be configured to restrict some GPIO configuration to the secure execution context.
- TAMP: can restrict the access of some of its registers to the secure execution context.
- EXTI: can restrict the access of some of its registers to the secure execution context.
- GIC: can restrict the access of some of its registers to the secure execution context.
- MDMA: can configure MDMA interrupt execution context.

5 Secure debug

The STM32 MPU offers the possibility to manage the platform debug configuration.
It is indeed possible to enable/disable independently secure and non-secure debug accesses.

5.1 STM32MP1 secure debug

Debug accesses are controlled through BSEC peripheral.
By default, the STM32 MPU is started by the ROM code with both secure and non-secure debug enabled. When the trusted boot is enabled, the ROM code disables debug accesses and relies on secure OS to configure them. When a FSBL debug is required, a dedicated wrapper exists (that can be also signed) to open the debug.

6 Secure and non-secure worlds

Thanks to Arm® TrustZone, some portions of the executing code can be assigned to a non-secure world or to a secure world.

The secure world offers an isolated context that guarantee code and data integrity up to the hardware support. The secure world can be used to host a Trusted Execution Environment (TEE) that executes in parallel with the rich OS and provides secure services.

The firmware and more generally software executing in secure world implicitly use the TrustZone(r) NS signal/bit to be granted access to sensitive resources. These resources can be DDR locations, SoC peripheral interfaces or SoC internal resources such as clocks, debug facilities, and more.

6.1 TF-A SP_MIN runtime services

The TF-A configuration supports the installation of a minimal runtime secure service provider and peripheral access control with firewall. This runtime firewall is built from TF-A BL32 SP_MIN image.

Run time services provided by TF-A includes not restricted to, PSCI controls, SCMI resources, some SiP & OEM SMCCC services for power states transition and other platform facilities.

6.2 OP-TEE OS runtime services

The OP-TEE is recommended by STMicroelectronics as it is an open source TEE solution. The package provides additional secure services to the platform since it can host core secure services and run trusted applications. OP-TEE provides the same level of services as TF-A listed above: PSCI, SCMI, SiP & OEM SMCCC. OP-TEE provides other high level services, such as running trusted applications and possibly generic services used by standard non-secure components such as random number generation.

7 Security frameworks

The platform non-secure world accesses to specific resources using generic frameworks. These operations are used to managed overall systems and must be implemented inside the TEE.

These accesses are using the SMC Calling convention [2]

Identifier	Name	Description
0x84000000-0x8400001F	PSCI 32-bit calls	32-bit Power Secure Control Interface
0x82000000-0x8200FF00	SiP Service Calls	Interfaces to SoC implementation-specific services

7.1 PSCI

PSCI[3] manages the power state of the CPU and the overall system. This framework is an Arm® generic implementation targeting CPU related power management services among which secondary CPUs boot up, dynamic addition and removal of CPUs, CPU idle management and system shutdown and reset.

7.1.1 STM32MP1 PSCI support

Both TF-A SP_MIN and OP-TEE implement PSCI v1.1 [4] and offer standard services for:

- Core idle management
- Boot up for secondary boot core
- Dynamic addition and removal of cores
- System shutdown and reset

7.2 SCMI

The SCMI[5] specification defines a standard to access resources for power, performance and system management. This framework is an Arm® generic implementation.
SCMI protocols can be used to create device interfaces for external resource that are controlled by a SCMI server implementation. Targeted devices are of, not limited to:

- Power domain management
- Performance management
- Clock management
- Sensor management
- Reset controller management

The SCMI specification does not specify the SMC IDs to use. It relies on specific OEM IDs.

7.2.1 STM32MP1 SCMI support

See the specific article on SCMI overview. It presents SCMI used by the secure world (TF-A SP_MIN, OP-TEE) to expose system resources, as input source clocks and PLL output clocks, reset controllers to non-secure world (U-Boot bootloader, Linux kernel).

7.3 Platform SiP and OEM SMCCC services

7.3.1 STM32MP1

The STM32MP1 embeds SiP and OEM SMCCC services for non-secure world to access low-power transition facilities, clock calibration facilities, BSEC OTP access, possibly also some test facilities. Both TF-A and OP-TEE implement the same level of services aside possible test facilities.

Identifier	Name	Description
0x82001000	RCC	Access restricted to CIER/CIFR registers
0x82001001	PWR	Access restricted to CR3/WKUPCR/MPUWKUPENR registers
0x82001002	RCC Calibration	Secure services to start a HSI or CSI calibration
0x82001003	BSEC	Secure service to access OTP (read/write/programmation)
0x82001008	PD_DOMAIN	Access to program the power domain for low-power management
0x82001009	RCC OPP	Access to program CPU frequency
0x82002000	SCMI_AGENT0	Access to SCMI services Channel 0
0x82002001	SCMI_AGENT1	Access to SCMI services Channel 1 (MCKPROT Resources)

8 Security peripherals

8.1 Cryptographic hardware acceleration

The STM32 MPU embeds multiple peripherals for cryptographic acceleration:

- CRYP
- HASH

8.2 Trusted platform module (TPM)

The STM32 MPU can be associated to an external trusted platform module (TPM).
It provides secret data storage capabilities as well as cryptographic capabilities allowing to use them.

8.3 Tamper event detection

The STM32 MPU embeds internal and external tampers detections mechanisms. They can be used to detect physical attack, out-of-spec voltage, etc. A tamper detection will result in the automatic clearing of sensitive data in the system: it is a key feature for security product.

8.3.1 STM32MP1

The tamper configuration is described in STM32MP15 Tamper configuration.

9 Secure Secret Provisioning

The STM32 MPU allows secrets to be provisioned in BSEC fuses in a secure way. This is the Secure Secret Provisioning (SSP) feature.

This feature is a secure process which runs between the software programmer (including a HSM), the ROM code and a customized TF-A BL2. The security relies on the chip attestation and a package encryption exchange between the programmer and the STM32 MPU.

This feature is fully describes in the AN5510: Overview of the secure secret provisioning (SSP) on STM32MP1 Series.

9.1 Fuses provisioning in STM32MP1

The build process for the TF-A SSP firmware is described here.

10 References

- https://www.arm.com/why-arm/technologies/trustzone-for-cortex-a
- https://developer.arm.com/documentation/den0028/latest/
- https://developer.arm.com/architectures/system-architectures/software-standards/psci
- https://developer.arm.com/docs/den0022/latest
- https://developer.arm.com/architectures/system-architectures/software-standards/scmi

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