HASH device tree configuration

Contents

Stable: 13.05.2020 - 08:37 / Revision: 13.05.2020 - 08:36

Stable: 13.05.2020 - 08:37 / Revision: 13.05.2020 - 08:38

A quality version of this page, approved on *13 May 2020*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the **HASH** internal peripheral when it is assigned to the Linux®OS. In that case, it is controlled by the Crypto framework.

The configuration is performed using the device tree mechanism that provides a hardware description of the HASH peripheral, used by the STM32 HASH Linux driver.

If the peripheral is assigned to another execution context, refer to How to assign an internal peripheral to a runtime context article for guidelines on peripheral assignment and configuration.

2 DT bindings documentation

The HASH is represented by the STM32 HASH 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 HASH node is declared in stm32mp151.dtsi[2]. It describes the hardware register address, clock, interrupt, reset and dma.

```
 hash1: hash@54002000 {                                                    Comments
         compatible = "st,stm32f756-hash";
         reg = <0x54002000 0x400>;                                          -->
Register location and length
         interrupts = <GIC_SPI 80 IRQ_TYPE_LEVEL_HIGH>;                     --> The
interrupt number used
         clocks = <&scmi0_clk CK_SCMI0_HASH1>;
         resets = <&scmi0_reset RST_SCMI0_HASH1>;
         dmas = <&mdma1 31 0x10 0x1000A02 0x0 0x0 0x0>;                     --> DMA
specifiers[3]
         dma-names = "in";
         dma-maxburst = <2>;
         status = "disabled";
 };
```

Warning

This device tree part is related to STM32 microprocessors. It must be kept as is, without being modified by the end-user.

3.2 DT configuration (board level)

This part is used to enable the HASH used on a board which is done by setting the **status** property to **okay**.

3.3 DT configuration examples

```
&hash1 {
        status = "okay";
};
```

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.
The STM32CubeMX may 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 file
- Documentation/devicetree/bindings/dma/stm32-mdma.txt , STM32 MDMA controller

Operating System

Device Tree

Generic Interrupt Controller

Serial Peripheral Interface

Direct Memory Access

Stable: 19.10.2020 - 09:54 / Revision: 19.10.2020 - 09:52

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

The Crypto API is a cryptography framework in the Linux® kernel. It is dedicated to the parts of the kernel that deal with cryptography, such as IPsec and dm-crypt.

Contents

1 Framework purpose

The purpose of this article is to introduce the Crypto API framework:

- general information
- main component/stakeholders
- how to use the Crypto API
- use cases

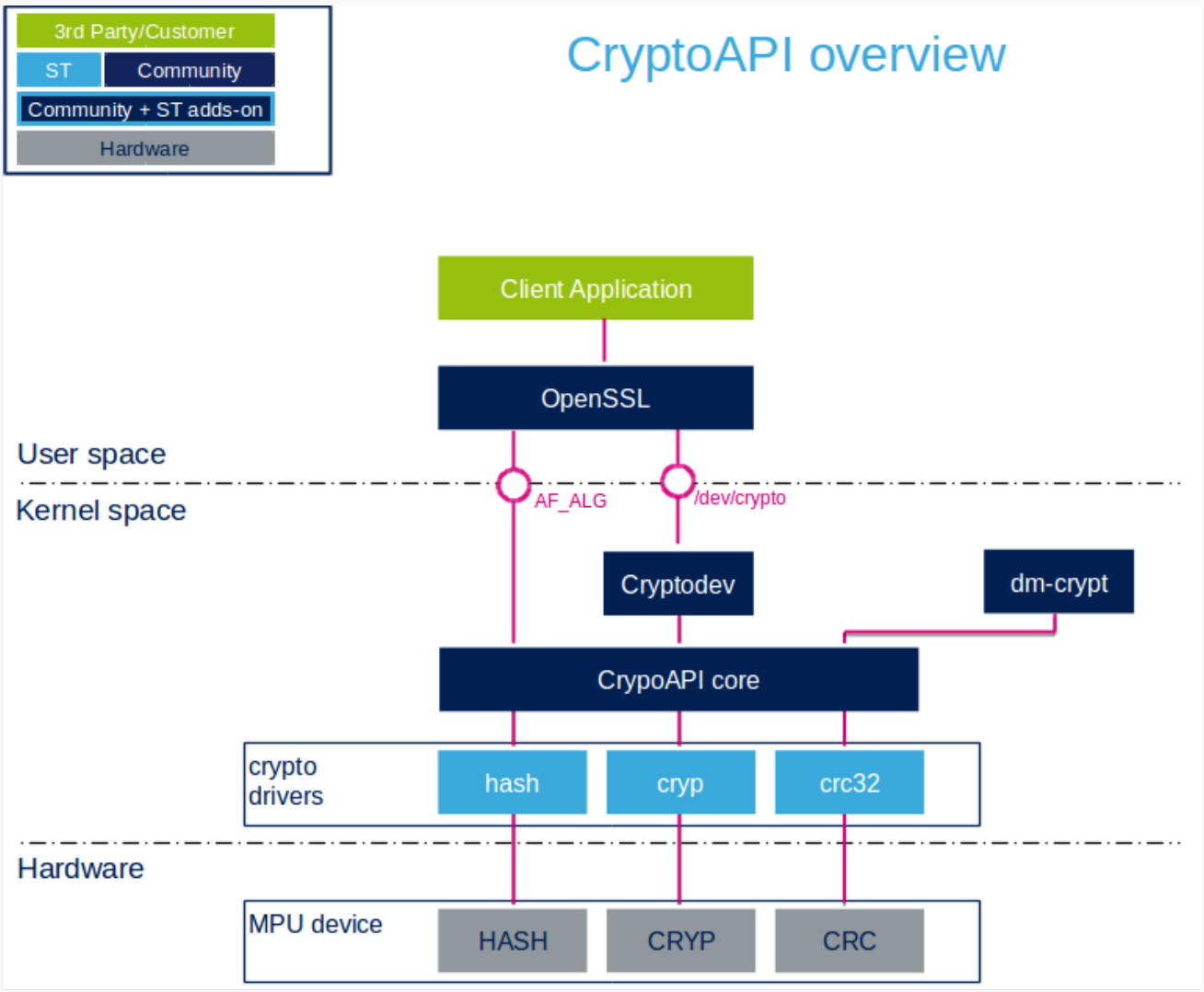
The Crypto API framework mainly includes all popular **hash** and **block ciphers** (encryption) functions.

A **hash** is a string or number generated from a text string. The length of the resulting string or number is fixed and widely varies with small variations of the input. The best hashing algorithms are designed so that it is impossible to turn a hash back into its original string. Hashing is particularly useful to compare a value with a stored value. However it cannot store its plain representation for security reasons. This makes hashing an ideal solution to store passwords.

Encryption turns data into a series of unreadable characters which length is not fixed. The encrypted strings can reversed back into their original decrypted form if the right key is not provided. Encrypting a confidential file is a good way to prevent anyone from accessing its content.

Drivers for CRYP (block cipher), HASH (hash) and CRC (cyclic redundancy check) are integrated within the Crypto API kernel service.

2 System overview



Crypto API

2.1 Description of the components

Information

OpenSSL and dm-crypt are not part of the Crypto API framework but they are typical users of the Crypto API services.

From User space to hardware

- **OpenSSL** (User space)

OpenSSL[1] is a software library supporting the TLS and SSL protocols as well as cryptographic functions. Openssl is available in OpenSTLinux distribution.

- **dm-crypt** (Kernel space)

dm-crypt[2] is a kernel disk encryption subsystem. It is natively available in the standard Linux kernel.

- **Cryptodev** (Kernel space)

Cryptodev[3] is a device driver which provides a general interface for userland applications. Although it is not part of the standard Linux kernel, it is available in OpenSTLinux distribution.

- **CryptoAPI core** (Kernel space)

This layer represents the standard Linux kernel cryptographic framework.

- **hash**, **cryp** and **crc32** (Kernel space)

These are the cryptographic Linux drivers handling the internal peripherals.

- **HASH**, **CRYP** and **CRC** (Hardware)

These HW blocks handle hash, ciphering, and CRC checksum.

2.2 API description

The Crypto API is documented in the Linux Kernel Crypto API section of the Linux Kernel documentation[4]. It offers both a kernel and a userland interface:

- kernel internal interface, used in particular by dm-crypt.
- userland algorithm interface (socket) named AF_ALG[5]. OpenSSL can use this interface.

In addition to the socket user interface, a more friendly interface, the cryptodev, can be used. It offers the /dev/crypto ioctl API. It is roughly described by the cryptodev.h[6] header file. OpenSSL can be configured to use this interface as an alternative to the historical AF_ALG interface.

3 Configuration

3.1 Kernal configuration

The Crypto API is activated by default in ST deliveries. Nevertheless, if a specific configuration is required, you can use Linux Menuconfig tool: Menuconfig or how to configure kernel and select:

```
[*] Cryptographic API  --->
    [*]   Hardware crypto devices  --->
        [*]   Support for STM32 crc accelerators
        [*]   Support for STM32 hash accelerators
        [*]   Support for STM32 crypto accelerators
```

3.2 Devicetree configuration

By default the drivers are not enabled, so this needs to be added if you want to use HW accelerators.

- crc: CRC_device_tree_configuration.
- hash: HASH_device_tree_configuration.
- crypto: CRYP_device_tree_configuration.

4 How to use the Crypto API framework

The Crypto API framework can be used by other kernel modules.

The Crypto API documentation provides kernel code examples[7]:

- Symmetric-key cipher operation.
- Operational state memory with SHASH.

5 Use cases

- Disk encryption

This is a typical example of Crypto API framework usage. Refer to LUKS[8] for a standard disk encryption process.

6 How to trace and debug the framework

6.1 How to monitor

The list of available ciphers is given in /proc/crypto:

```
Board $> cat /proc/crypto
```

Output part showing that an STM32 driver provides with the CRC32 cipher:

```
...
name         : crc32
driver       : stm32-crc32
module       : kernel
priority     : 200
refcnt       : 1
selftest     : passed
internal     : no
type         : shash
blocksize    : 1
digestsize   : 4
...
```

6.2 How to trace

There are no specific traces for this framework.

6.3 How to debug

There are no specific debug means for this framework.

7 Generic source code location

- CryptoAPI core
- CryptoAPI interface
- stm32 crypto drivers

8 References

- OpenSSL a software library supporting the TLS and SSL protocols as well as cryptographic functions.
- dm-crypt a kernel disk encryption subsystem
- Cryptodev a device driver which provides a general interface for userland applications
- Linux Kernel Crypto API the official crypto API kernel documentation
- Crypto API Userland interface specification of the userland API
- cryptodev.h cryptodev header file specifying the userland API
- Crypto API kernel code examples some kernel code examples using the Crypto API framework
- LUKS (Linux Unified Key Setup) a disk encryption specification

Application programming interface

Linux® is a registered trademark of Linus Torvalds.

Cryptographic processor

Cyclic redundancy check calculation unit

GPIO alternate function

Stable: 04.02.2020 - 07:47 / Revision: 04.02.2020 - 07:34

A quality version of this page, approved on *4 February 2020*, was based off this revision.

Contents

1 Purpose

The objective of this chapter is to give general information about the device tree.
An extract of the **device tree specification**[1] explains it as follows:

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

In other words, a device tree describes the hardware that can not be located by probing. For more information, please refer to the device tree specification[1]

1.1 Source files

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

- **.dtsi:** Source files that can be included from a DTS file.

1.2 Bindings

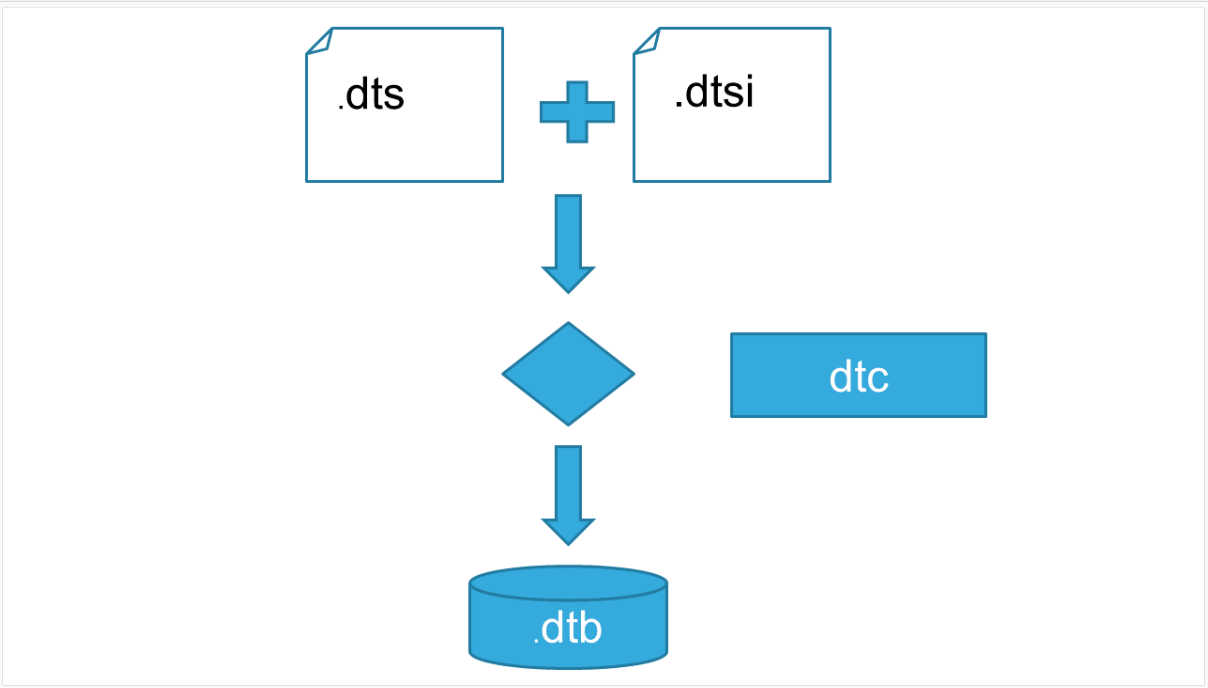
The device tree data structures and properties are named **bindings**. Those bindings are described in:

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

1.3 Build

- A tool named DTC (Device Tree Compiler) allows compiling the DTS sources into a binary.
- input file: the *.dts* file described in section above.
- output file: the *.dtb* file described in section above.
- More information are available in DTC manual[2].

- DTC source code is located here[3]. DTC tool is also available directly in particular software components:
  Linux Kernel, U-Boot, TF-A .... For those components, the device tree building is directly integrated in the component build process.



Device tree build process

Information

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

1.4 Tools

The device tree compiler offers also some tools:

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

There are several ways to get those tools:

- In the device tree compiler project source code[3]
- Directly in software components: **Kernel, u-boot, tf-a ...**
- Available in Debian package[4]

2 STM32

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

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

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

3 How to go further

- Device Tree for Dummies[5] - Free Electrons
- Device Tree Reference[6] - eLinux.org
- Device Tree usage[7] - eLinux.org

4 References

- 1.01.11.2 https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.2(latest) ,Device tree specification
- https://git.kernel.org/pub/scm/utils/dtc/dtc.git/tree/Documentation/manual.txt(master) ,DTC manual
- 3.03.1 https://git.kernel.org/pub/scm/utils/dtc/dtc.git(master) ,DTC source code
- https://packages.debian.org/search?keywords=device-tree-compiler(master) ,DTC debian package
- Device Tree for Dummies, Free Electrons
- Device Tree Reference, eLinux.org
- Device Tree Usage, eLinux.org

Device Tree Source (in software context) or Digital Temperature Sensor (in peripheral context)

Linux® is a registered trademark of Linus Torvalds.

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

Trusted Firmware for Arm Cortex-A

Stable: 12.02.2020 - 16:46 / Revision: 12.02.2020 - 16:44

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

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the HASH peripheral 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 necessary, how to configure the HASH peripheral.

2 Peripheral overview

The **HASH** peripheral is used to compute a message digest.
Digest algorithms could be:

- MD5[1]
- SHA[2] (1, 224, 256)

The **HASH** peripheral is also able to give the HMAC[3] used for authentication using the same algorithm support.

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

HASH1 is a **secure** peripheral (under ETZPC control)
HASH2 is a **non secure** peripheral .

3 Peripheral usage and associated software

3.1 Boot time

HASH1 instance is used as boot device to support binary authentication.
HASH2 is not used at boot time.

3.2 Runtime

3.2.1 Overview

HASH1 instance can be allocated to:

- the Arm®Cortex®-A7 secure core to be controlled in OP-TEE by the OP-TEE HASH driver

or

- the Arm®Cortex®-A7 non-secure core to be controlled in Linux® by the Linux Crypto framework

HASH2 instance can be allocated to:

- the Arm®Cortex®-M4 to be controlled in STM32Cube MPU Package by STM32Cube HASH driver

Chapter Peripheral assignment describes which peripheral instance can be assigned to which context.

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Security	HASH	OP-TEE HASH driver	Linux Crypto framework	STM32Cube HASH driver	

3.2.3 Peripheral configuration

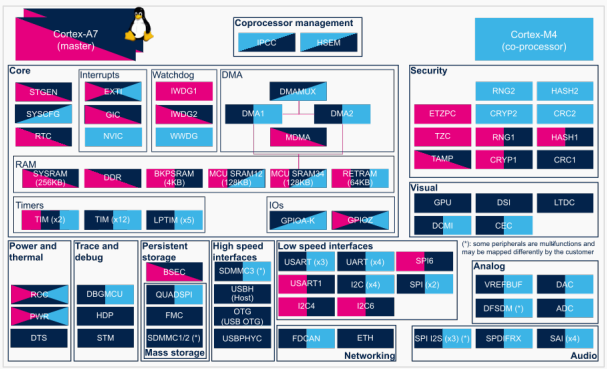
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in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

3.2.4 Peripheral assignment

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

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

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



Internal peripherals

Domain	Periphera l	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Security	HASH	HASH1				Assignment (single choice)
		HASH2				

4 How to go further

Not applicable.

5 References

- https://en.wikipedia.org/wiki/MD5
- https://en.wikipedia.org/wiki/Secure_Hash_Algorithms
- https://en.wikipedia.org/wiki/HMAC

Message Digest 5

Secure Hash Algorithm

Hash-based Message Authentication Code

Cortex®

Open Portable Trusted Execution Environment

Linux® is a registered trademark of Linus Torvalds.

Microprocessor Unit

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.

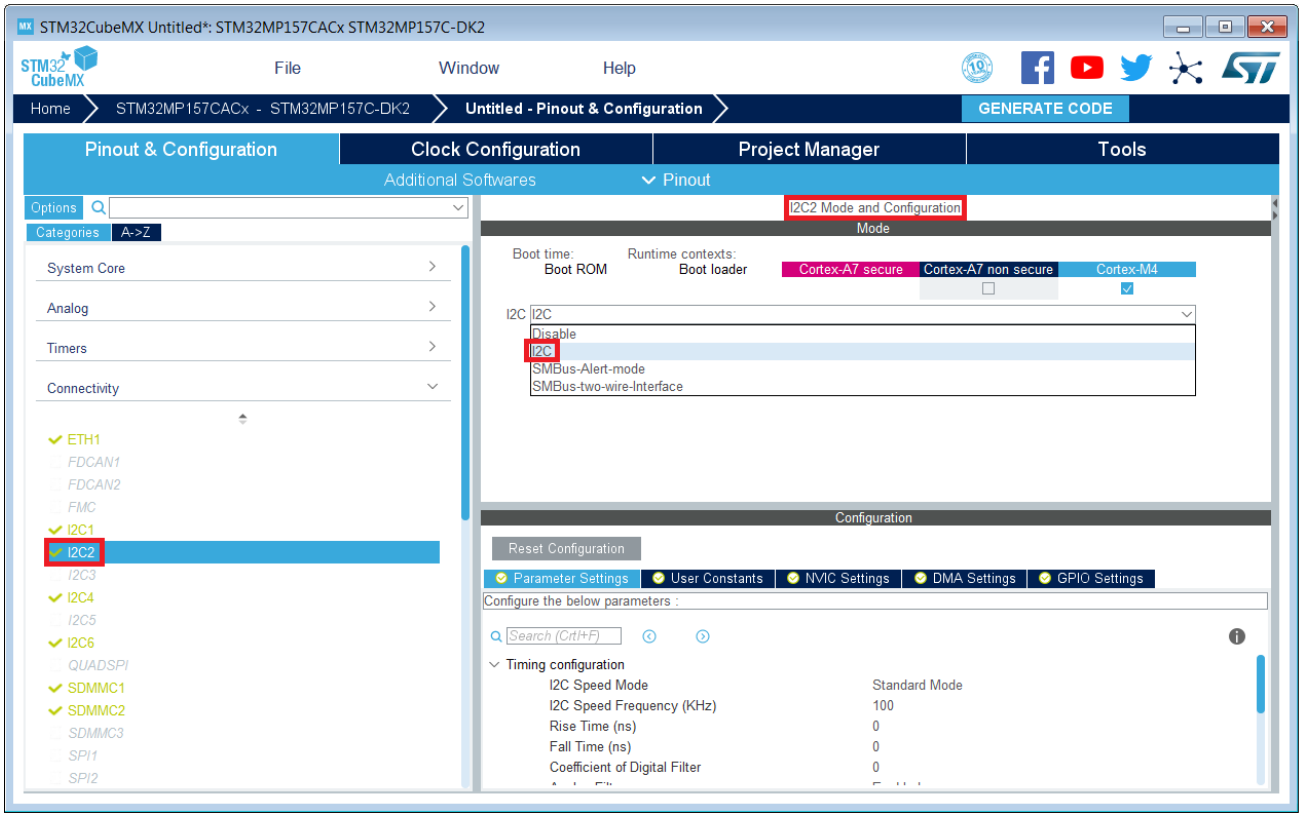
Information

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: 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