



How to assign an internal peripheral to a runtime context



Contents

1. How to assign an internal peripheral to a runtime context	3
2. Getting started with STM32 MPU devices	7
3. Device tree	8
4. STM32CubeMX	8
5. ETZPC internal peripheral	8
6. TF-A overview	8
7. U-Boot overview	8
8. STM32MP15 Linux kernel overview	8
9. STM32CubeMP1 architecture	8
10. OP-TEE overview	9
11. STM32MP15 peripherals overview	9
12. STM32MPU Embedded Software distribution	9
13. STM32MP15 resources	9
14. ETZPC device tree configuration	9
15. Resource manager for coprocessing	9
16. STM32MP15 device tree	10
17. How to configure system resources	10



How to assign an internal peripheral to a runtime context

Stable: 22.01.2020 - 16:08 / Revision: 22.01.2020 - 10:33

Contents

1 Article purpose	3
2 Introduction	3
3 STM32CubeMX generated assignment	3
4 Manual assignment	4
4.1 TF-A	5
4.2 U-boot	5
4.3 Linux kernel	6
4.4 STM32Cube	6
4.5 OP-TEE	7

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 assignment 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.

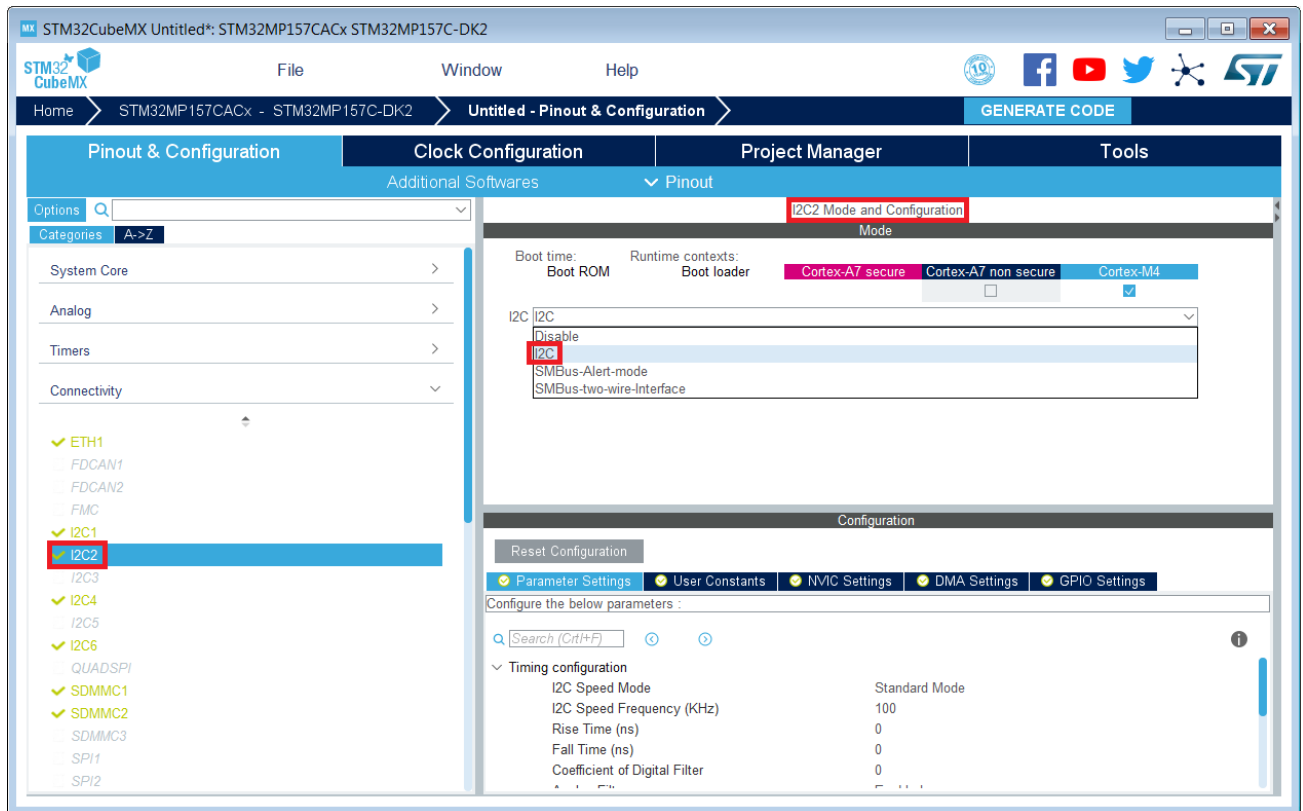
The following sections describe how to configure TF-A, U-Boot, Linux and STM32Cube accordingly.

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



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.

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.



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)
    >;
};
```



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.



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.



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/stm32mp157c.dtsi`, corresponding to Linux assigned peripherals
 - Example: `i2c2`
- Once in the **m4_rproc** node from `arch/arm/boot/dts/stm32mp157c-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";  
};
```



Beyond the peripherals assignment, explained in this article, it is also important to understand [How to configure system resources](#), shared between the Cortex-A7 and Cortex-M4 contexts

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:

- For ecosystem release v1.1.0

```
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();
}
}
```

- For ecosystem release v1.0.0

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



Beyond the peripherals assignment, explained in this article, it is also important to understand [How to configure system resources](#), shared between the Cortex-A7 and Cortex-M4 contexts

4.5 OP-TEE

No specific configuration is needed in OP-TEE to configure the access to the peripheral.

Trusted Firmware for Arm Cortex-A

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

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

Central processing unit

Getting started with STM32 MPU devices



How to assign an internal peripheral to a runtime context

Stable: 22.01.2020 - 16:06 / Revision: 22.01.2020 - 10:27

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [Getting started with STM32 MPU devices](#).

Device tree

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

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [Device tree](#).

STM32CubeMX

Stable: 31.01.2020 - 13:04 / Revision: 31.01.2020 - 13:02

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [STM32CubeMX](#).

ETZPC internal peripheral

Stable: 20.02.2019 - 10:27 / Revision: 20.02.2019 - 10:27

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [ETZPC internal peripheral](#).

TF-A overview

Stable: 30.01.2020 - 13:42 / Revision: 30.01.2020 - 13:40

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [TF-A overview](#).

U-Boot overview

Stable: 23.01.2020 - 13:52 / Revision: 23.01.2020 - 13:46

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [U-Boot overview](#).

STM32MP15 Linux kernel overview

Stable: 15.10.2019 - 13:59 / Revision: 15.10.2019 - 13:58

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [STM32MP15 Linux kernel overview](#).



STM32CubeMP1 architecture

Stable: 21.02.2020 - 08:39 / Revision: 04.02.2020 - 15:22

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [STM32CubeMP1 architecture](#).

OP-TEE overview

Stable: 12.03.2020 - 12:15 / Revision: 14.10.2019 - 14:35

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [OP-TEE overview](#).

STM32MP15 peripherals overview

Stable: 22.01.2020 - 16:15 / Revision: 22.01.2020 - 13:48

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [STM32MP15 peripherals overview](#).

STM32MPU Embedded Software distribution

Stable: 09.10.2019 - 09:37 / Revision: 26.09.2019 - 10:05

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [STM32MPU Embedded Software distribution](#).

STM32MP15 resources

Stable: 21.02.2020 - 08:59 / Revision: 14.02.2020 - 10:13

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [STM32MP15 resources](#).

ETZPC device tree configuration

Stable: 06.02.2020 - 14:20 / Revision: 06.02.2020 - 14:18

Invalid target: no **reviewed** revision corresponds to the given ID.

Return to [ETZPC device tree configuration](#).

Resource manager for coprocessing

Stable: 27.09.2019 - 07:08 / Revision: 18.09.2019 - 07:38

Invalid target: no **reviewed** revision corresponds to the given ID.



How to assign an internal peripheral to a runtime context

[Return to Resource manager for coprocessing.](#)

STM32MP15 device tree

Stable: 21.02.2020 - 12:52 / Revision: 21.02.2020 - 09:24

Invalid target: no **reviewed** revision corresponds to the given ID.

[Return to STM32MP15 device tree.](#)

How to configure system resources

Stable: 03.06.2020 - 12:56 / Revision: 03.06.2020 - 12:55

Invalid target: no **reviewed** revision corresponds to the given ID.

[Return to How to configure system resources.](#)