



life.augmented

Developer package classroom

Cindy Ge

March 2021

Agenda

1 Developer package
Overview

2 SDK

3 Source code

4 Source code configuration

5 Device tree

6 STM32CubeMx

7 References

Developer package Overview



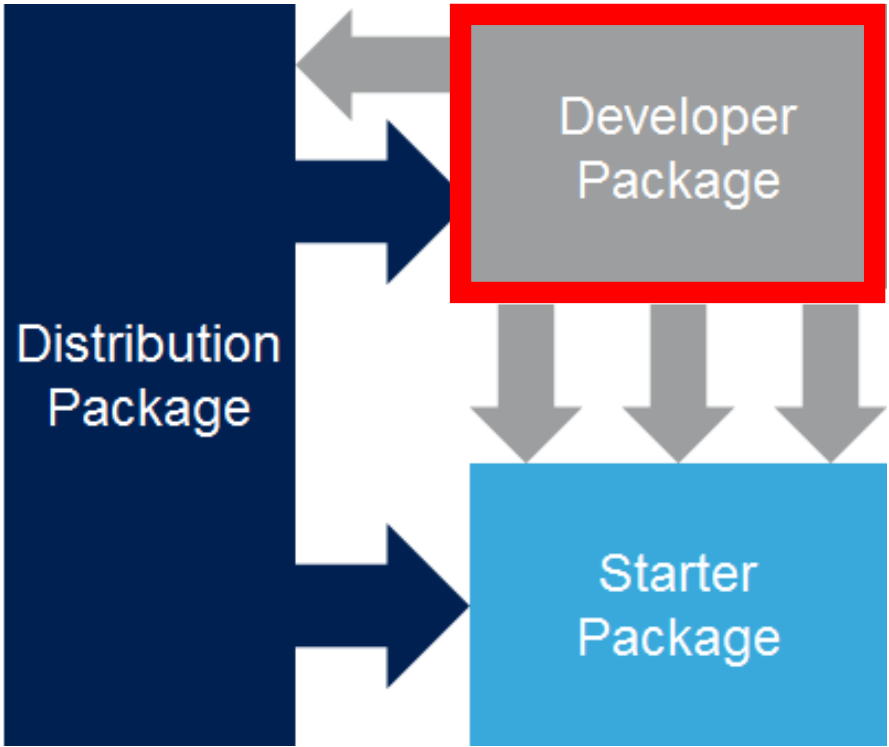
The developer package provides SDK environment to be able to re-generate starter package binaries, DTB* and customers applications too. The results can be used directly in the starter package or/and integrated inside the Distribution package.

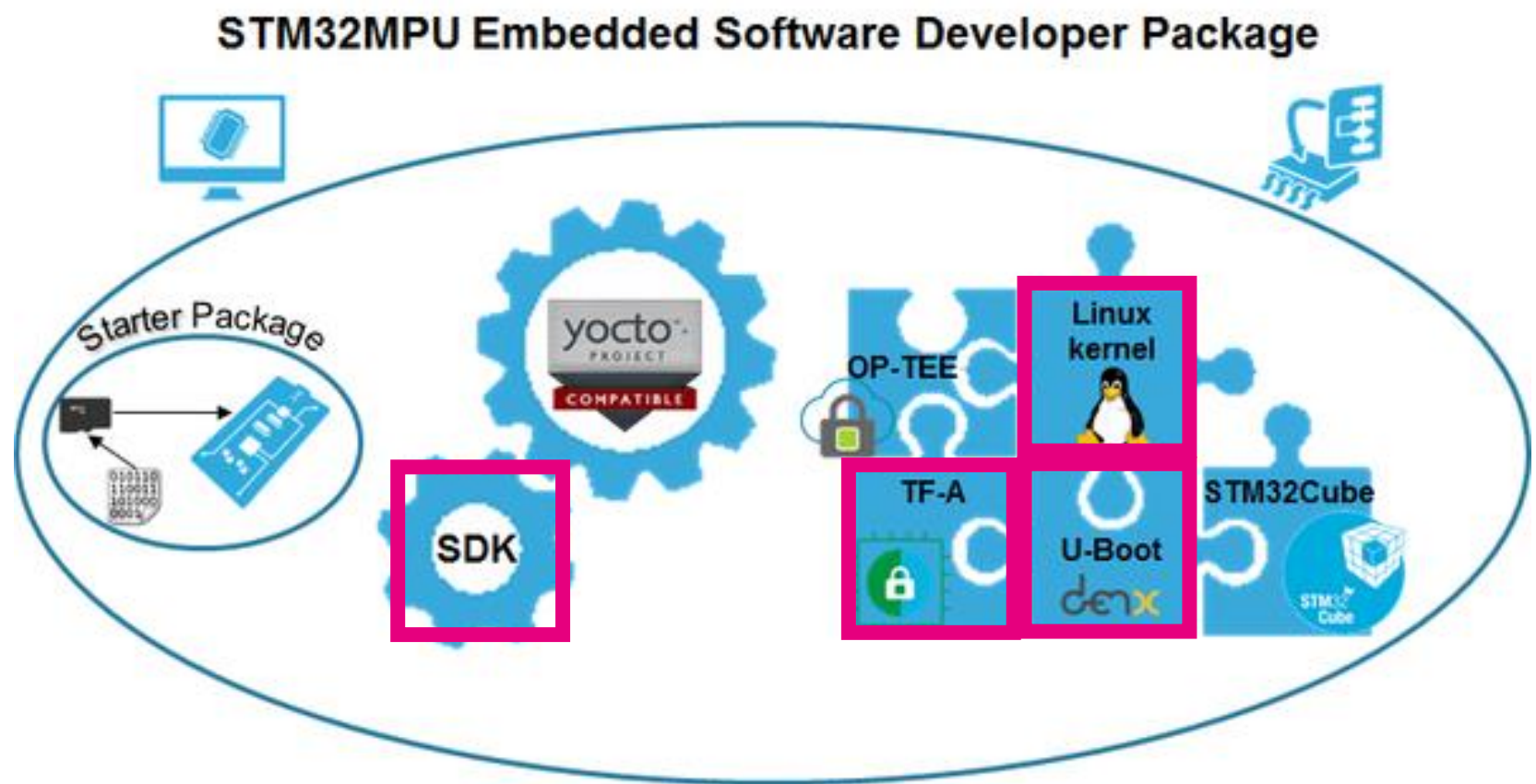
On top of SDK environment, source code of Linux kernel, U-Boot, TF-A, STM32Cube MPU and Optee are included.

During this training STM32Cube MPU will be introduced in other lesson/hands on and Optee will be not addressed.

On the USB key you can find the developer package in:
/work/developer/

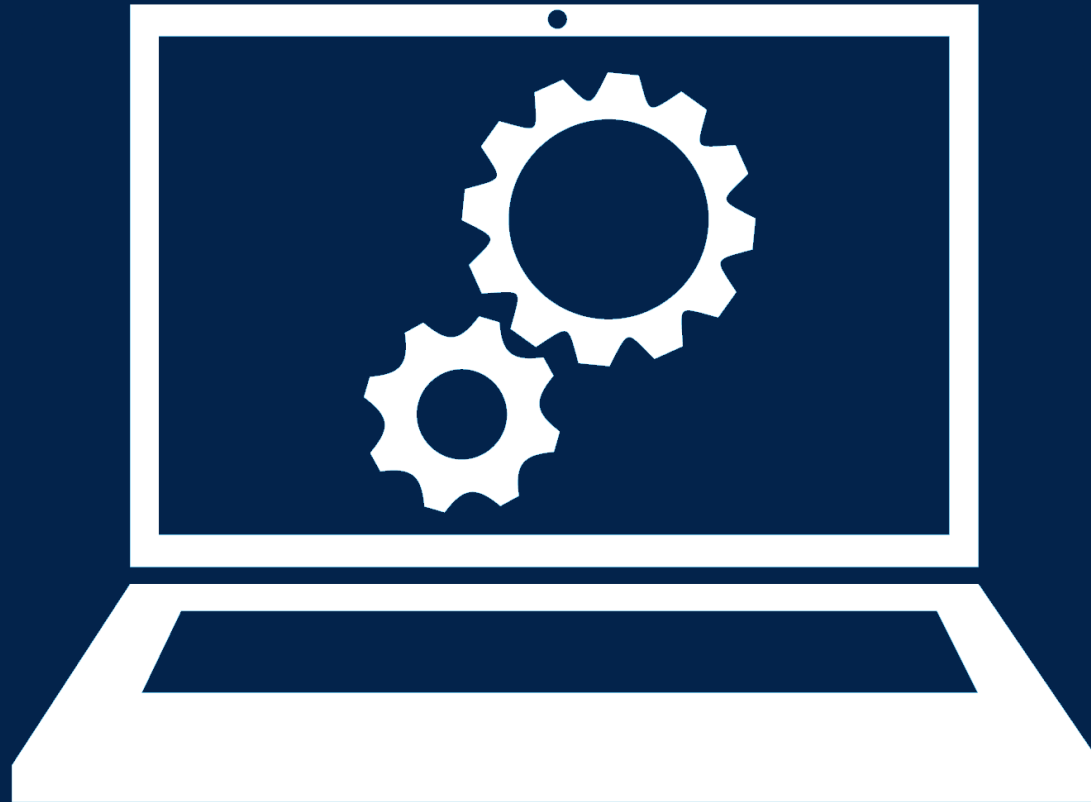
A new SDK can be generated thanks to the distribution Package.





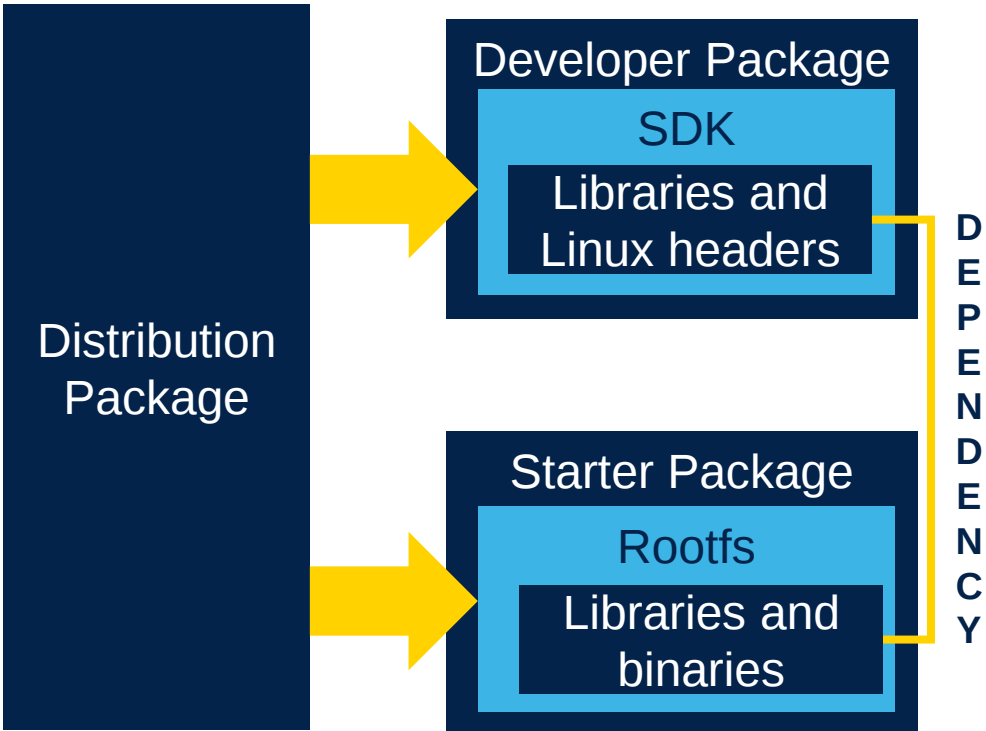
 Today lesson

SDK



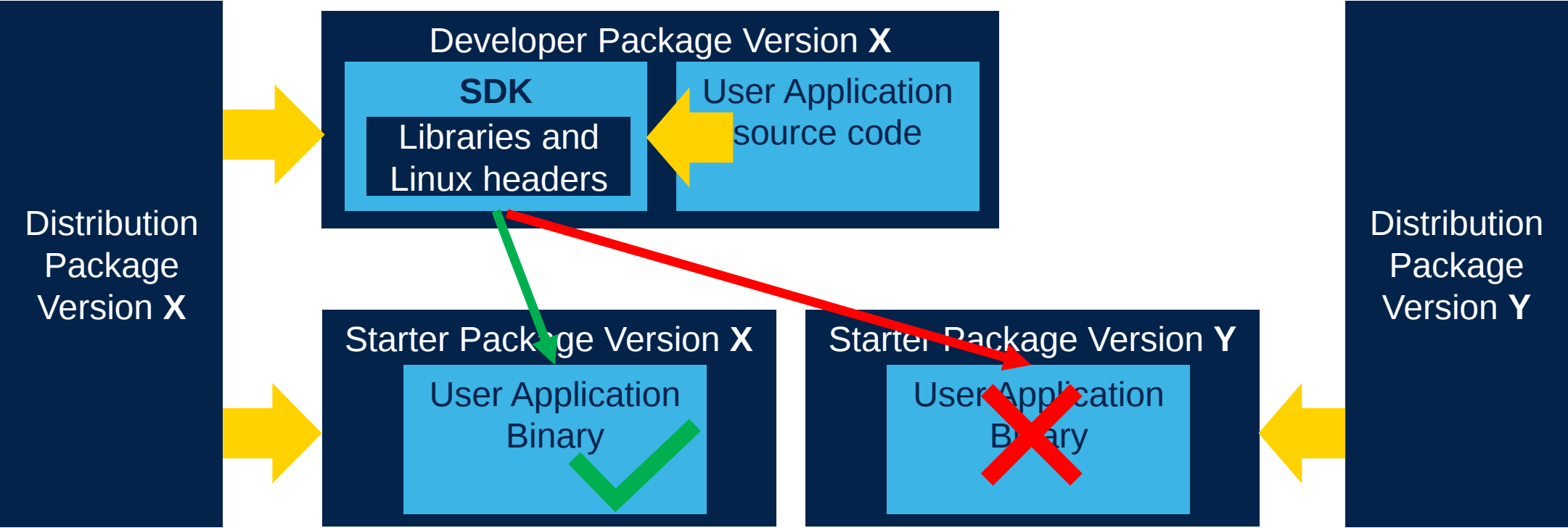
The software development kit (SDK) for the OpenSTLinux distribution is a customization of the Yocto SDK, which provides a stand-alone cross-development toolchain and libraries tailored to the contents of a specific image. The OpenSTLinux SDK is part of the STM32MPU Embedded Software Developer Package.

The SDK might be generated, through the STM32MPU Embedded Software Distribution Package, during the compilation of a software release, which guarantees the alignment of this SDK with the software images (binaries) built for the Starter Package of the STM32MPU Embedded Software.



If some packages, framework or middleware have been added or their version have been changed in the starter package the SDK has to be updated too. This dependency is mainly on the Linux application build process.

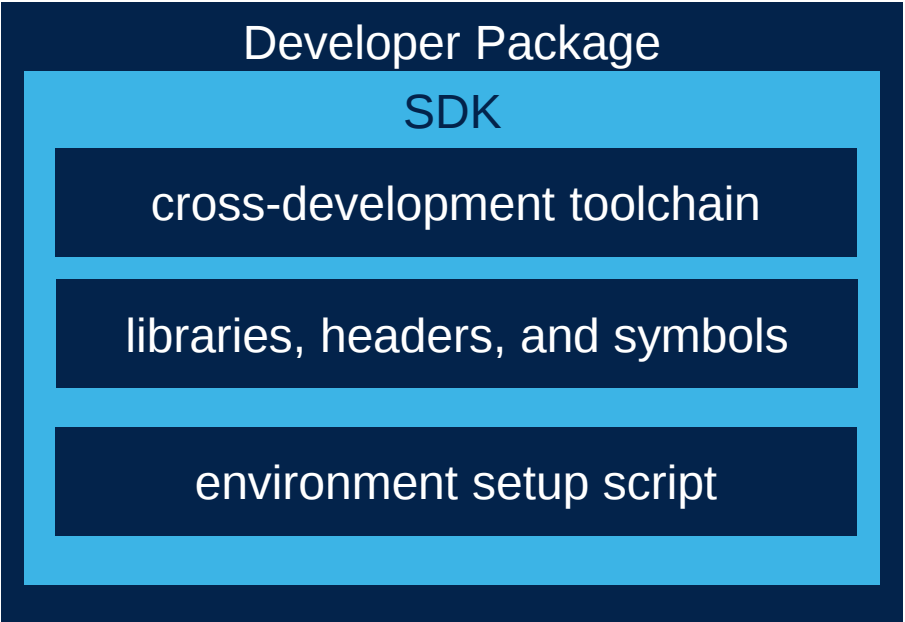
Many SDKs can coexist on the same host machine.



The OpenSTLinux SDK is based on the standard Yocto project SDK.

A standard SDK consists of the following:

- a cross-development toolchain: this toolchain contains a compiler, linker, debugger, and various miscellaneous tools
- libraries, headers, and symbols (target and native sysroots): the libraries, headers, and symbols are specific to the image (that is, they match the image)
- an environment setup script: this *.sh file, once run, sets up the cross-development environment by defining variables and preparing it for SDK use



The installation SDK is in the <xxxxxxxx> :

<SDK installation directory>

└─ environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi

└─ site-config-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi

└─ sysroots

└─┬─ cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi

└─┬─┬─ [...]

└─┬─ x86_64-ostl_sdk-linux

└─┬─┬─ [...]

└─ version-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi

Details in [Standard SDK directory structure](#) article

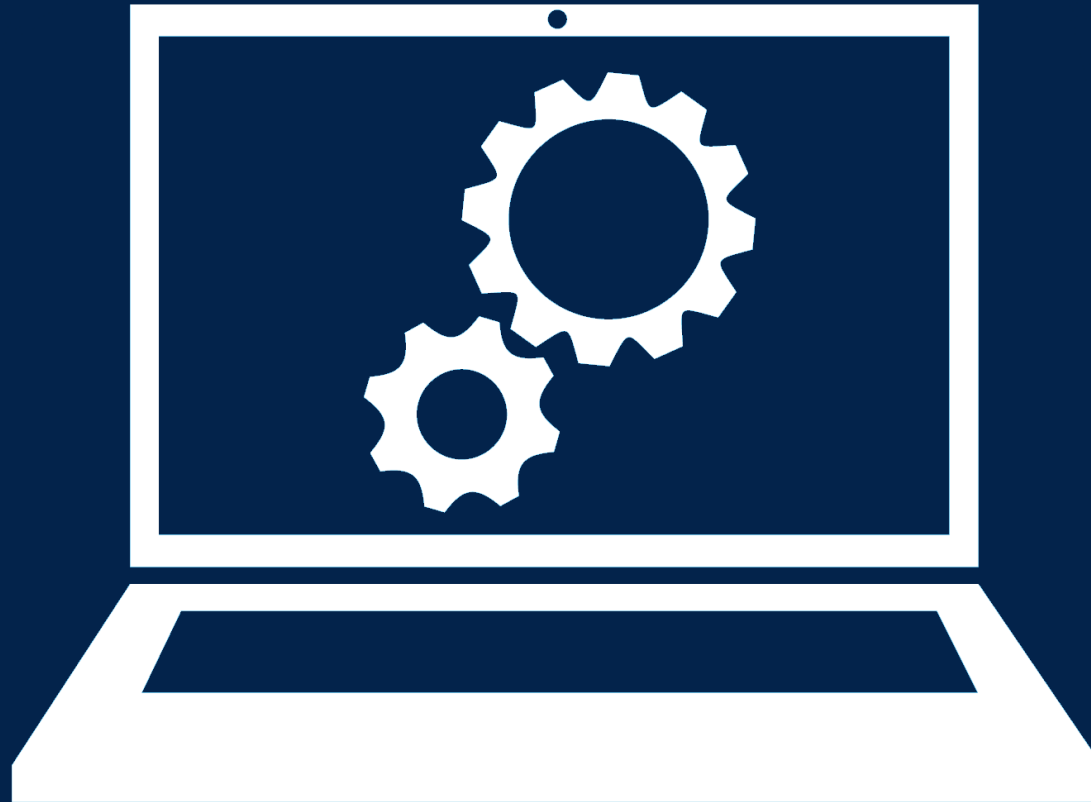
Environment setup script for Dev. Package

Target sysroot (libraries, headers, and symbols)

Native sysroot (libraries, headers, and symbols)

This package can be downloaded only from ST.com

Source code



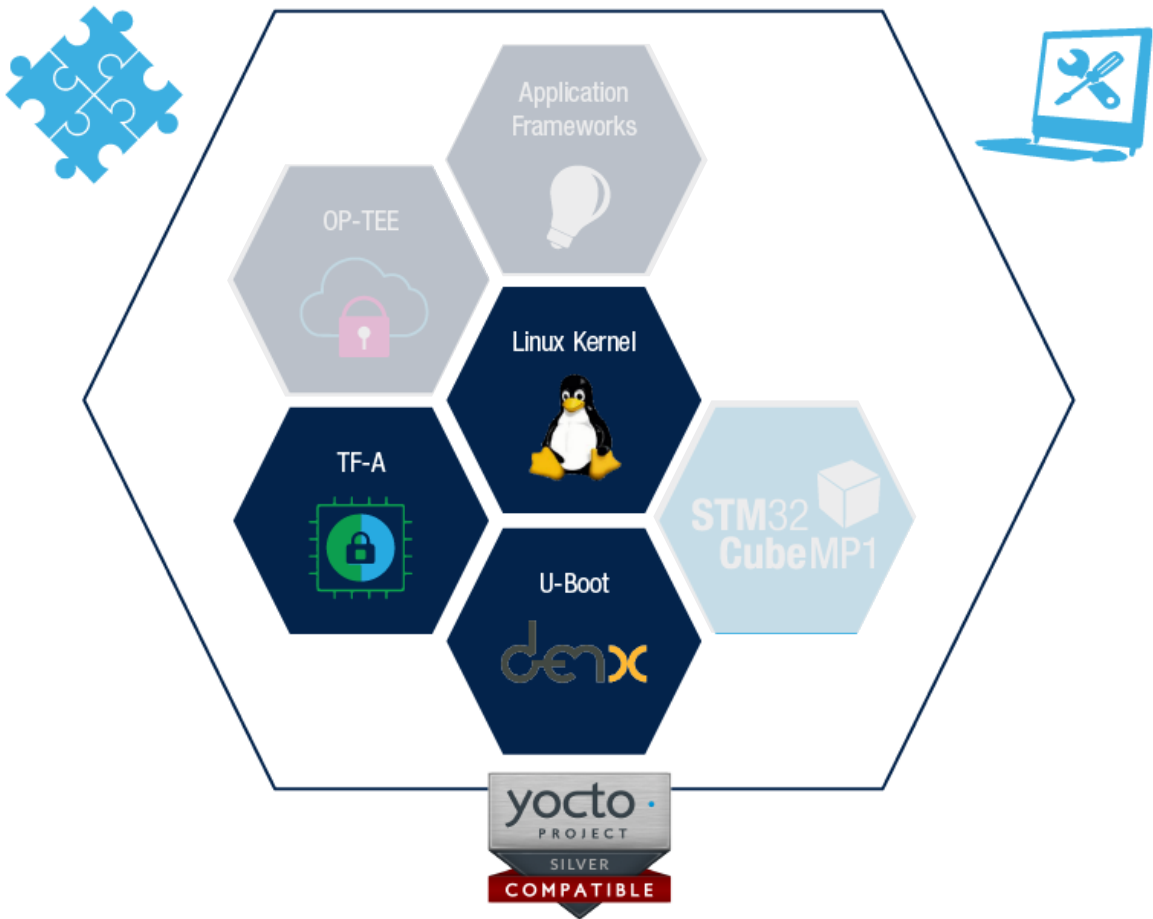
The following pieces of software in source code will be studied during this training:

- Trusted Firmware-A (TF-A)
- U-Boot
- Linux® kernel

All the source code can be downloaded from the **ST.com** or **github.com** (see reference).

For this training source code are already on a USB key :

/work/developer/



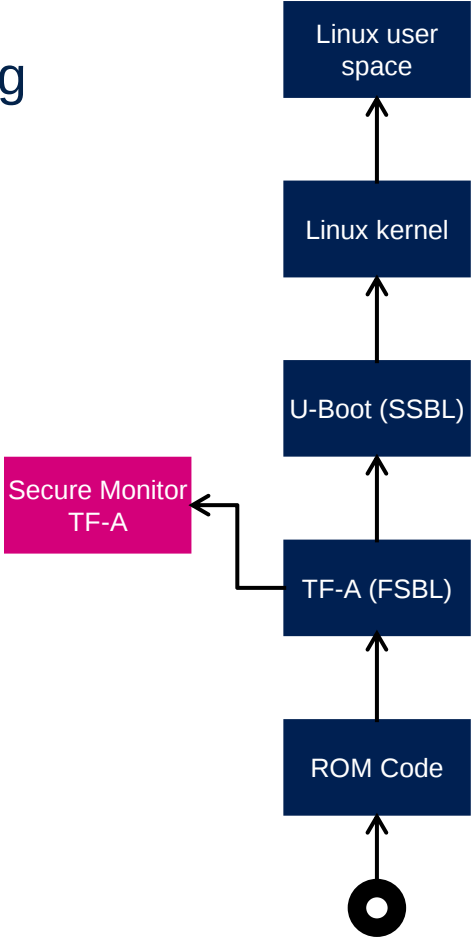
Trusted Firmware-A (TF-A) is a reference implementation of secure-world software provided by Arm®. It was first designed for Armv8-A platforms and has been adapted to be used on Armv7-A platforms by STMicroelectronics. Arm is transferring the Trusted Firmware project to be managed as an open-source project by Linaro.

It is used as the first-stage boot loader (FSBL) on STM32 MPU platforms.

The code is open source, under a BSD-3-Clause license and can be found on [github](https://github.com/ARM-software/TrustedFirmware-A).

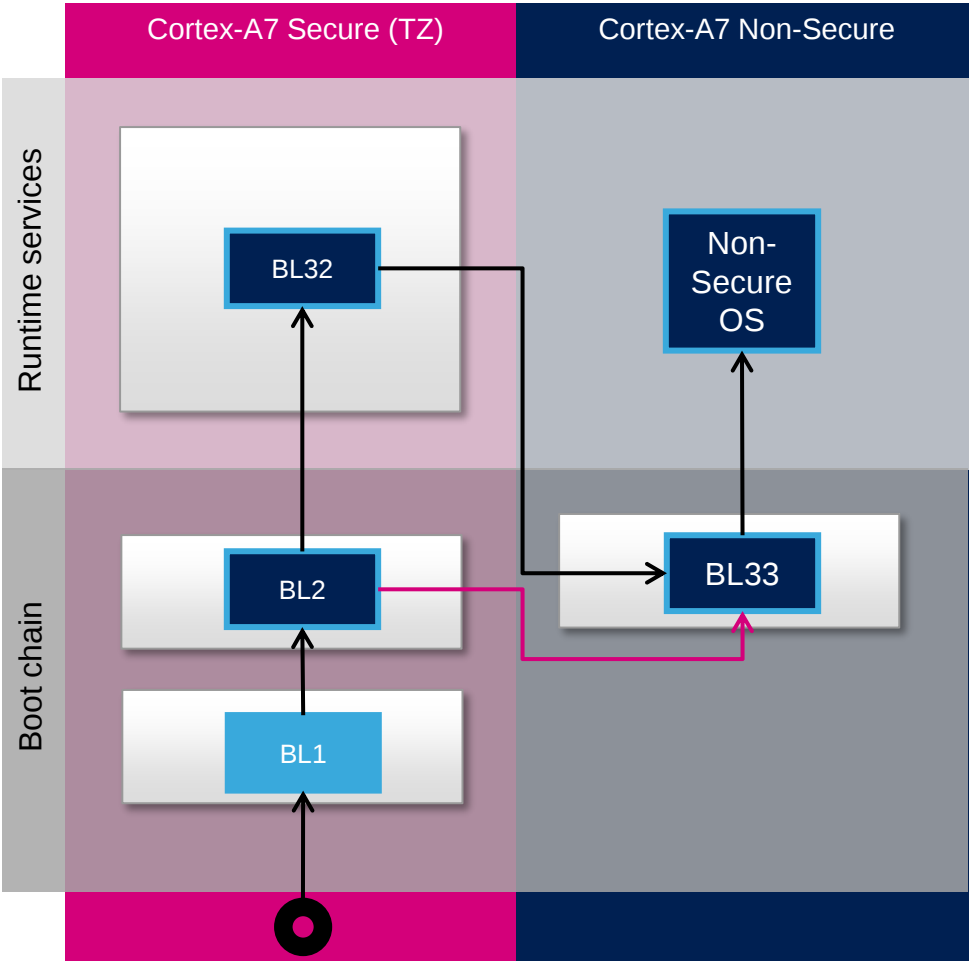
Trusted Firmware-A also implements a secure monitor with various Arm interface standards:

- The power state coordination interface (PSCI).
- Trusted board boot requirements (TBBR).
- SMC calling convention.
- System control and management interface.

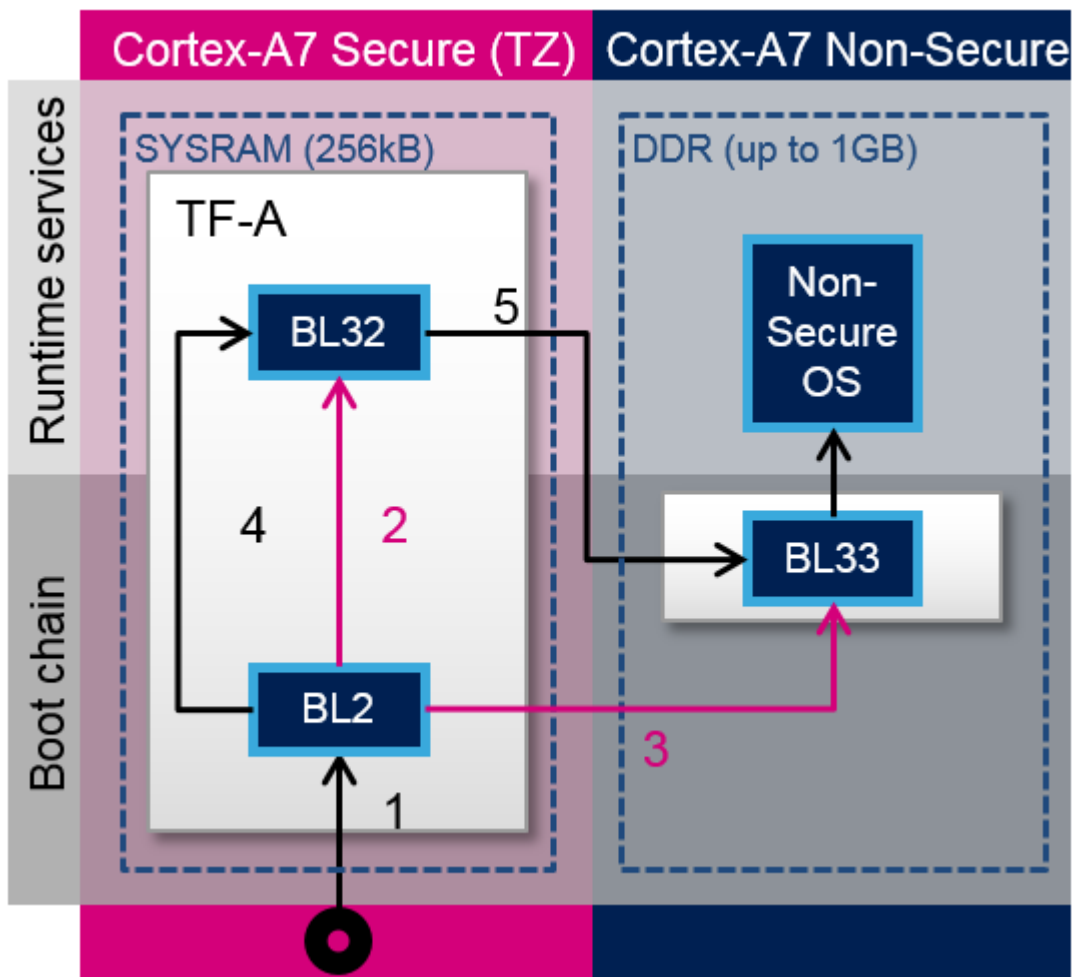


TF-A: architecture

- TF-A is divided into several binaries, each with a dedicated main role. For 32-bit Arm processors (AArch32), it is divided into four steps (in order of execution):
- Boot loader stage 1 (BL1) application processor trusted ROM
 - Boot loader stage 2 (BL2) trusted boot firmware
 - Boot loader stage 3-2 (BL32) runtime software
 - Boot loader stage 3-3 (BL33) non-trusted firmware
 - BL1, BL2 and BL32 are parts of TF-A.
- The global architecture of TF-A is explained in the Trusted Firmware-A doc folder in the source code.



TF-A: STM32MP15x architecture



BL1 is optional and can be removed by enabling the compilation flag: `BL2_AT_EL3`. It is then removed for the STM32MP1, as all BL1 tasks are done by ROM code or BL2.

BL33 is outside of TF-A. This is the first non-secure code loaded by TF-A. During the boot sequence, this is the secondary stage boot loader (SSBL). For STM32 MPU platforms, the SSBL is U-Boot by default.

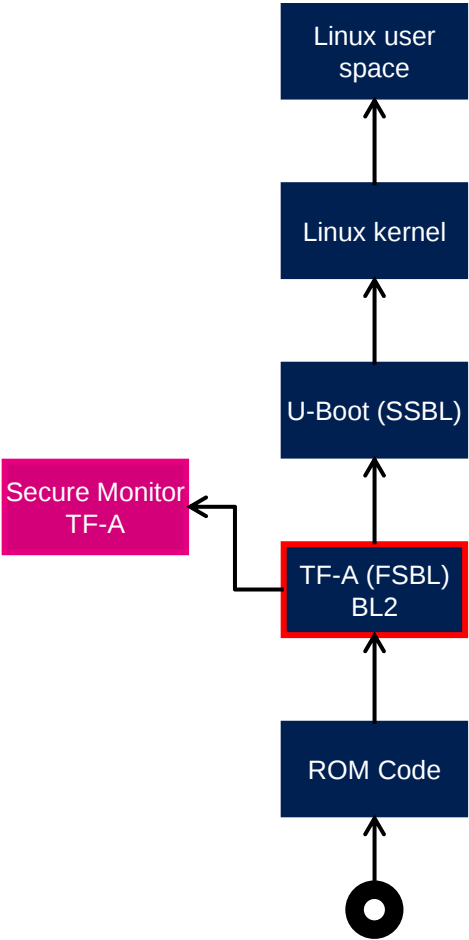
TF-A can manage its configuration with a device tree, as this is the case on STM32MP1. It is a reduced version of the Linux kernel one, with only the devices used during boot. It can be configured with STM32CubeMX.

In STMicroelectronics' implementation, the 2 binaries, BL2 and BL32, and the device tree are put together in a single binary, to be loaded at once to the SYSRAM by the ROM code.

BL2 oversees loading of the next-stage images, “secure” and “non-secure”. To achieve this role, BL2 must initialize all the required peripherals.

In “STM32MP15” implementation BL2 must initialize :

1. the security components :
 - boot and security, and OTP control (BSEC)
 - extended TrustZone protection controller (ETZPC)
 - TrustZone address space controller for DDR (TZC)
2. The DDR and clock tree.
3. The boot peripheral in use:
 - SD-card or eMMC via the SDMMC
 - NAND via the FMC
 - NOR or NAND via the QUADSPI
 - USB (OTG) or UART are used when Flashing.



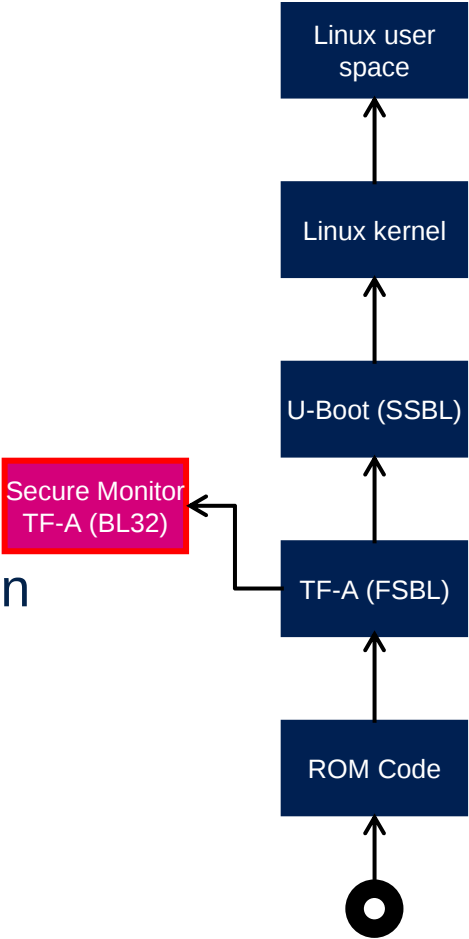
BL32 provides runtime secure services. In TF-A, the BL32 default implementation is SP_min solution.

This minimal implementation can be replaced with a trusted OS or trusted environment execution (TEE), such as OP-TEE. Both solutions (SP_min or OP-TEE) are supported by STMicroelectronics for STM32MP1.

BL32 acts as a secure monitor and thus provides secure services to non-secure OSs. These services are called by non-secure software with secure monitor calls.

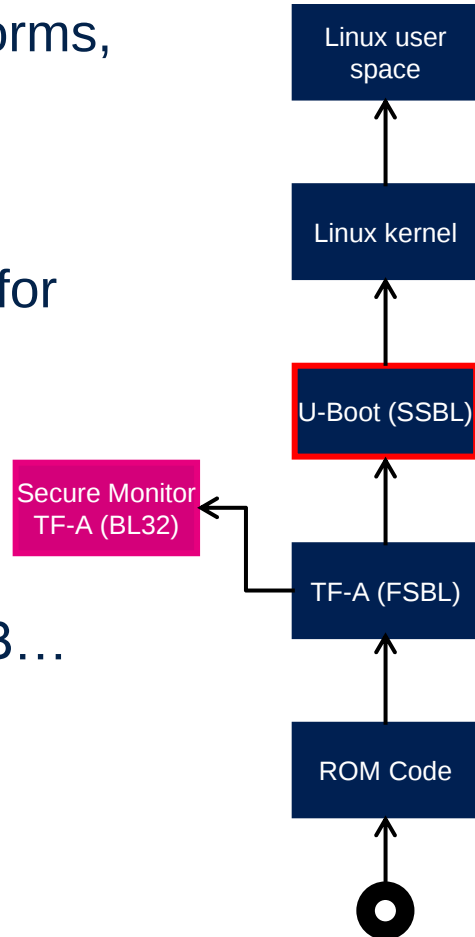
This code is in charge of standard service calls like PSCI (Power State Coordination Interface).

It also provides STMicroelectronics dedicated services, to access secure peripherals. On the STM32MP1, these services are used to access RCC, PWR, RTC or BSEC.



U-Boot is the default second stage boot loader (SSBL) for the STM32 MPU platforms, U-Boot is used because :

- it is configurable and expendable.
- it has a simple command line interface (CLI), usually over a serial console port for interaction with the user.
- it provides scripting capabilities.
- it loads the kernel into RAM and passes control to the kernel.
- it manages many internal and external devices like NAND, NOR, Ethernet, USB...
- it has many supported features and commands to manage :
 - file systems: FAT, UBI/UBIFS, JFFS
 - IP stack: FTP
 - display: LCD, HDMI, BMP for splashscreen
 - USB: host profile (mass storage) or device profile (DFU stack)



U-Boot runs through the following main phases in DDR:

Pre-relocation initialization (common/board_f.c): minimal init (cpu, clock, reset, ddr, console,...) running at the load address CONFIG_SYS_TEXT_BASE

Relocation: copy the code to the end of DDR

Post-relocation initialization:(common/board_r.c): init all the drivers

Execution of commands: through autoboot (CONFIG_AUTOBOOT) or console shell

- execute the boot command (bootcmd=CONFIG_BOOTCOMMAND by default):
for example, execute the command 'bootm' to:
 - load and check images (kernel, device tree, ramdisk....)
 - fixup kernel device tree
 - install secure monitor (optional)
 - pass control to the Linux kernel (or other target application)

This is an overview of Linux kernel drivers implemented for the STM32MP15 support, with their respective frameworks.

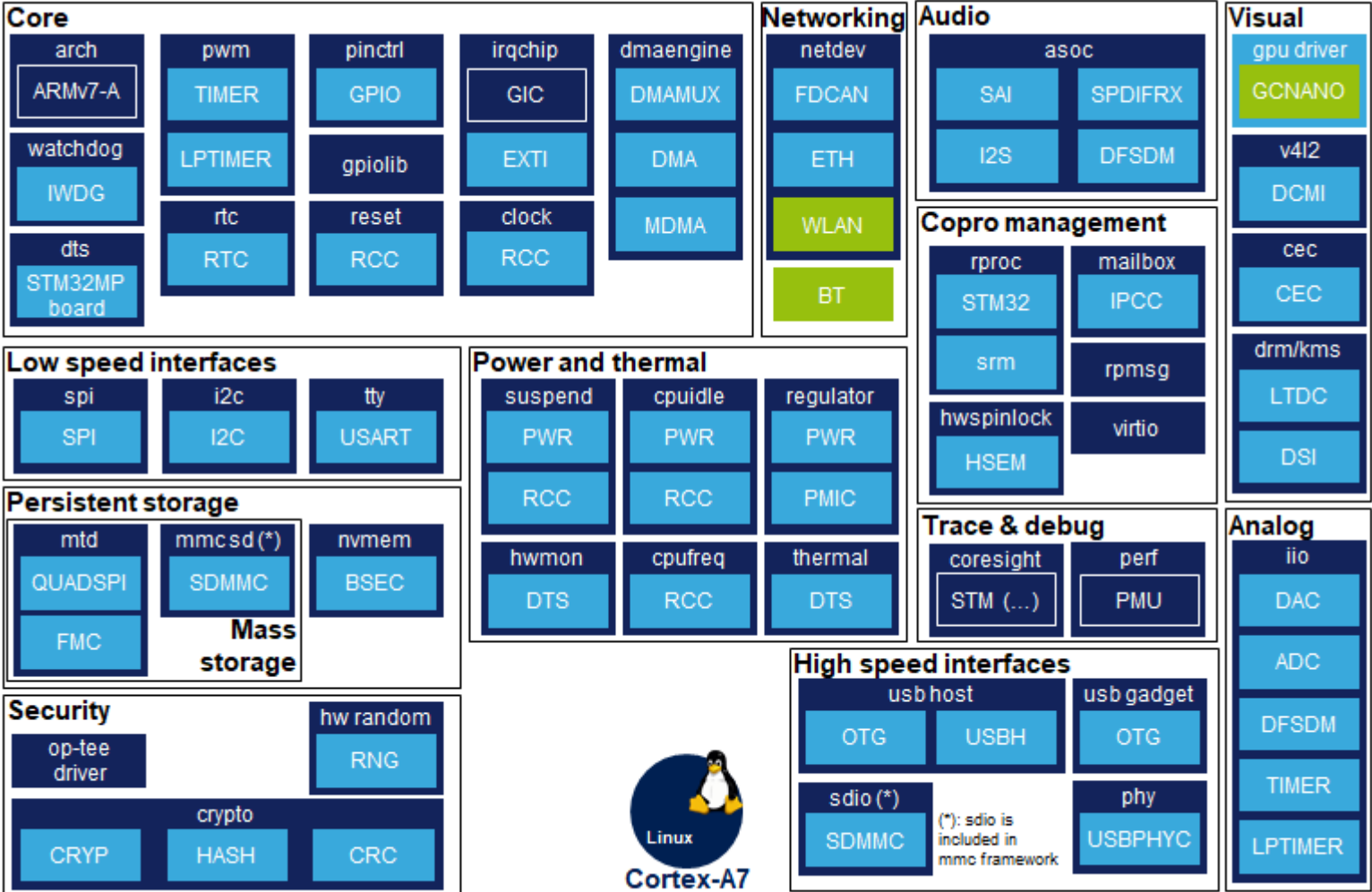
The components are grouped per functional domains.

3rd Party

STCommunity

- lowercase = community framework
- UPPERCASE = peripheral driver

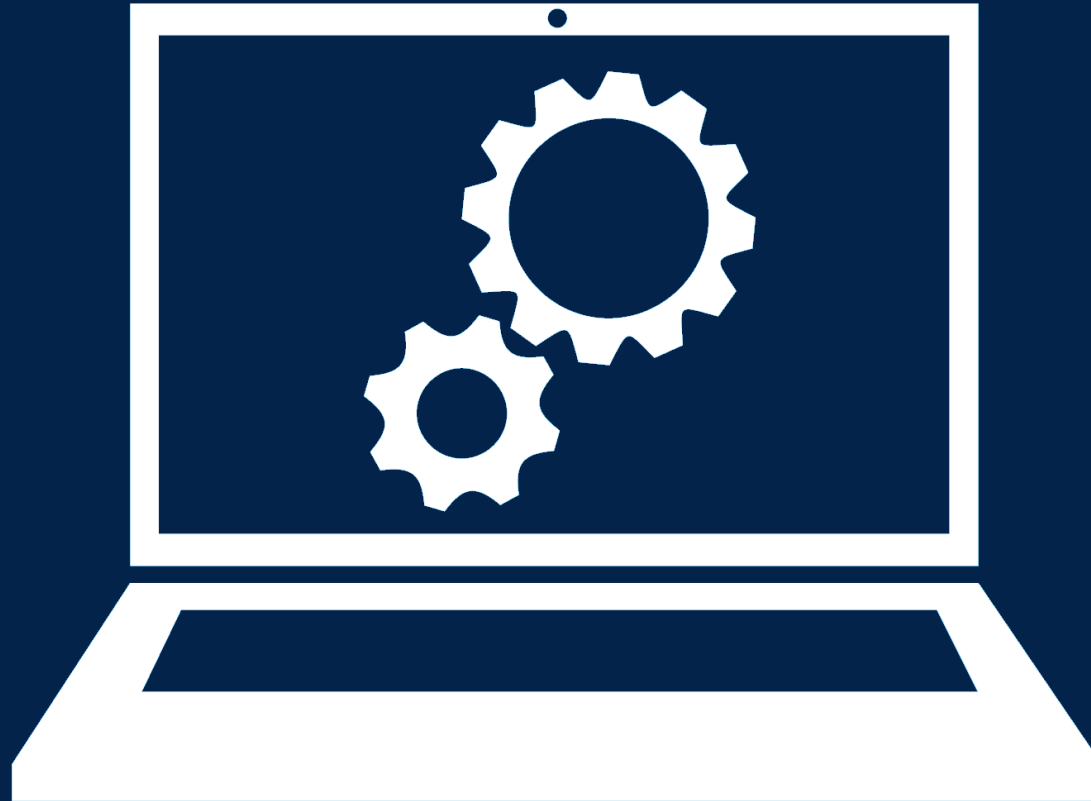
Legend



Cortex-A7

Non-Secure

Source code configuration



Configuration files

TF-A :

No configuration files to configure TF-A the configuration is done via make command line and device tree. (this configuration will be show during hands on)

U-Boot:

defconfig file :

- configs/stm32mp15_basic_defconfig
- configs/stm32mp15_optee_defconfig
- configs/stm32mp15_trusted_defconfig

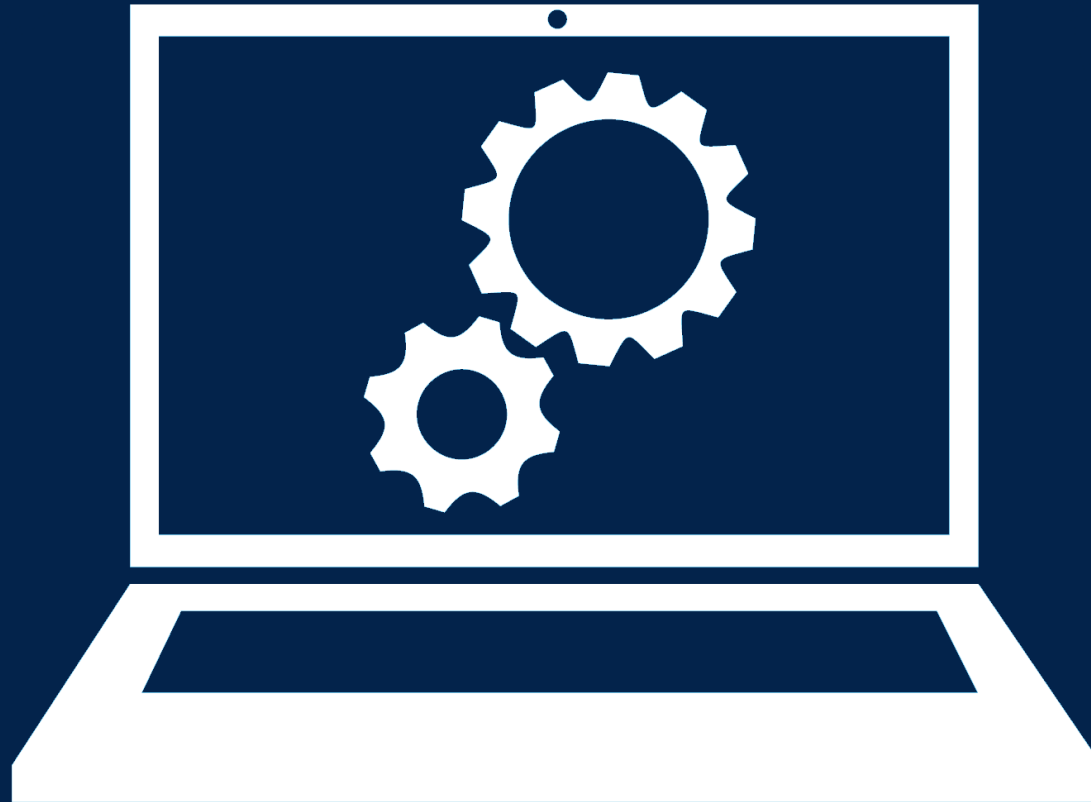
config files :

- include/configs/stm32mp1.h

Linux :

- Configuration via make menuconfig

Device Tree



Definition

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.

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

- The Device tree specification for generic bindings.
- The software component documentations:
 - Linux® Kernel: Documentation/devicetree/bindings
 - U-Boot: Documentation/devicetree/bindings

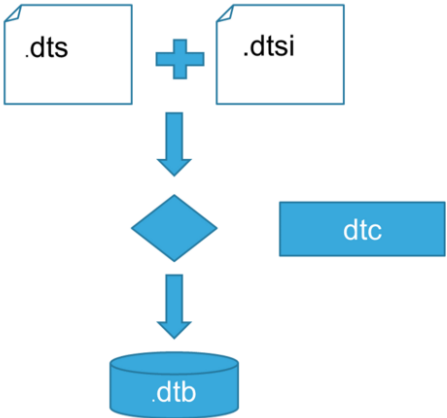
Files and generation

Files format :

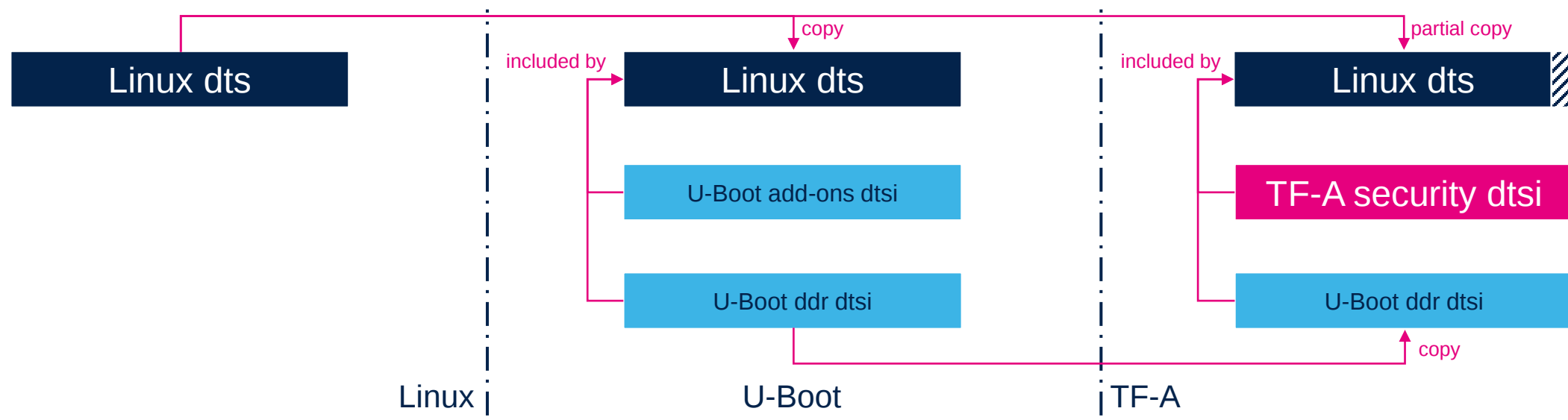
- .dts: The device tree source (DTS). This format is a textual representation of a device tree in a form that can be processed by DTC (Device Tree Compiler) into a binary device tree.
- .dtsi: Source files that can be included from a DTS.

A tool named DTC (Device Tree Compiler) allows compiling the DTS sources into a binary.

- input file: the .dts and dtsi files
- output file: the .dtb file



STM32MP15x Device tree structure



- In Linux, STM32MP15 is supported via a set of device tree source files (dts)
- In **U-Boot**, Linux dts files are copied and overloaded with U-Boot add-ons properties and DDR configuration
- In **TF-A**, Linux files are partly copied then completed with the DDR configuration (copied from U-Boot) and security configuration (firewalling)

Device tree & part numbers



MPU
@ 800 MHz



MPU
@ 650 MHz

STM32 MP151D

1520 + 260 DMIPS
800 MHz Cortex-A7
209 MHz Cortex-M4

MP151F

-
-
-
Security

STM32 MP153D

3040 + 260 DMIPS
800 MHz 2x Cortex A7
209 MHz Cortex-M4
CAN FD

MP153F

-
-
-
Security

STM32 MP157D

3040 + 260 DMIPS
800 MHz 2x Cortex-A7
209 MHz Cortex-M4
CAN FD - 3D GPU - DSI

MP157F

-
-
-
Security

STM32 MP151A

1235 + 260 DMIPS
650 MHz Cortex-A7
209 MHz Cortex-M4

MP151C

-
-
-
Security

STM32 MP153A

2470 + 260 DMIPS
650 MHz 2x Cortex-A7
209 MHz Cortex-M4
CAN FD

MP153C

-
-
-
Security

STM32 MP157A

2470 + 260 DMIPS
650 MHz 2x Cortex-A7
209 MHz Cortex-M4
CAN FD - 3D GPU - DSI

MP157C

-
-
-
Security



All references are available in **4 Packages**

TFBGA257 10x10mm p0.5 (4 layers PTH PCB) - smallest package for dual Cortex-A GP MPU

TFBGA361 12x12mm p0.5 (4 layers PTH + Laser via PCB)

LFBGA354 16x16mm p0.8 (4 layers PTH PCB)

LFBGA448 18x18mm p0.8 (6 layers PTH PCB)



All parts
are software
and pin to pin
compatible

Arm® Cortex® core

Cortex-A7 + Cortex-M4

Dual Cortex-A7 + Cortex-M4

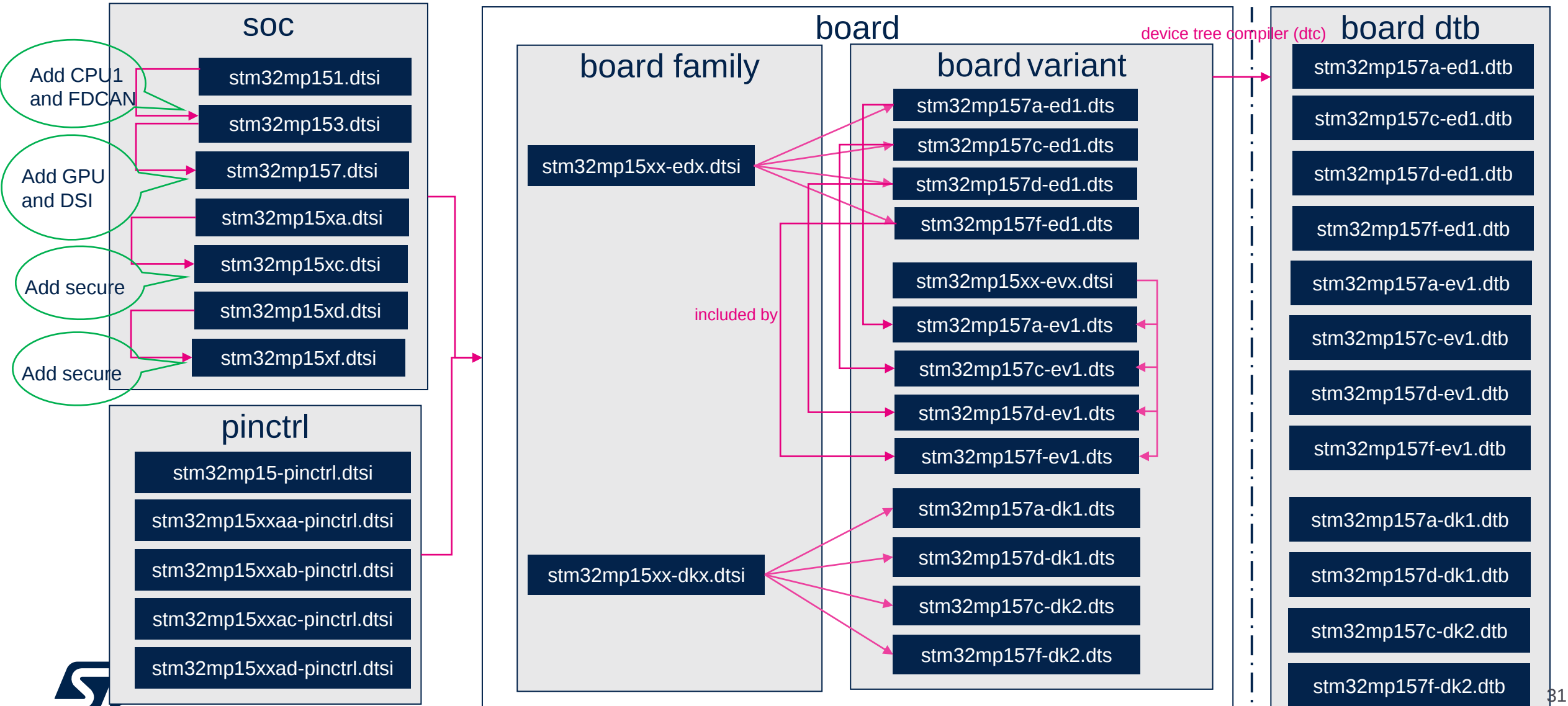


life.augmented

- ed: evaluation daughter board (embedding the STM32MP1)
- ev: evaluation board (ed plugged on the mother board)
- dk: discovery kit

Legend

device tree structure in Linux

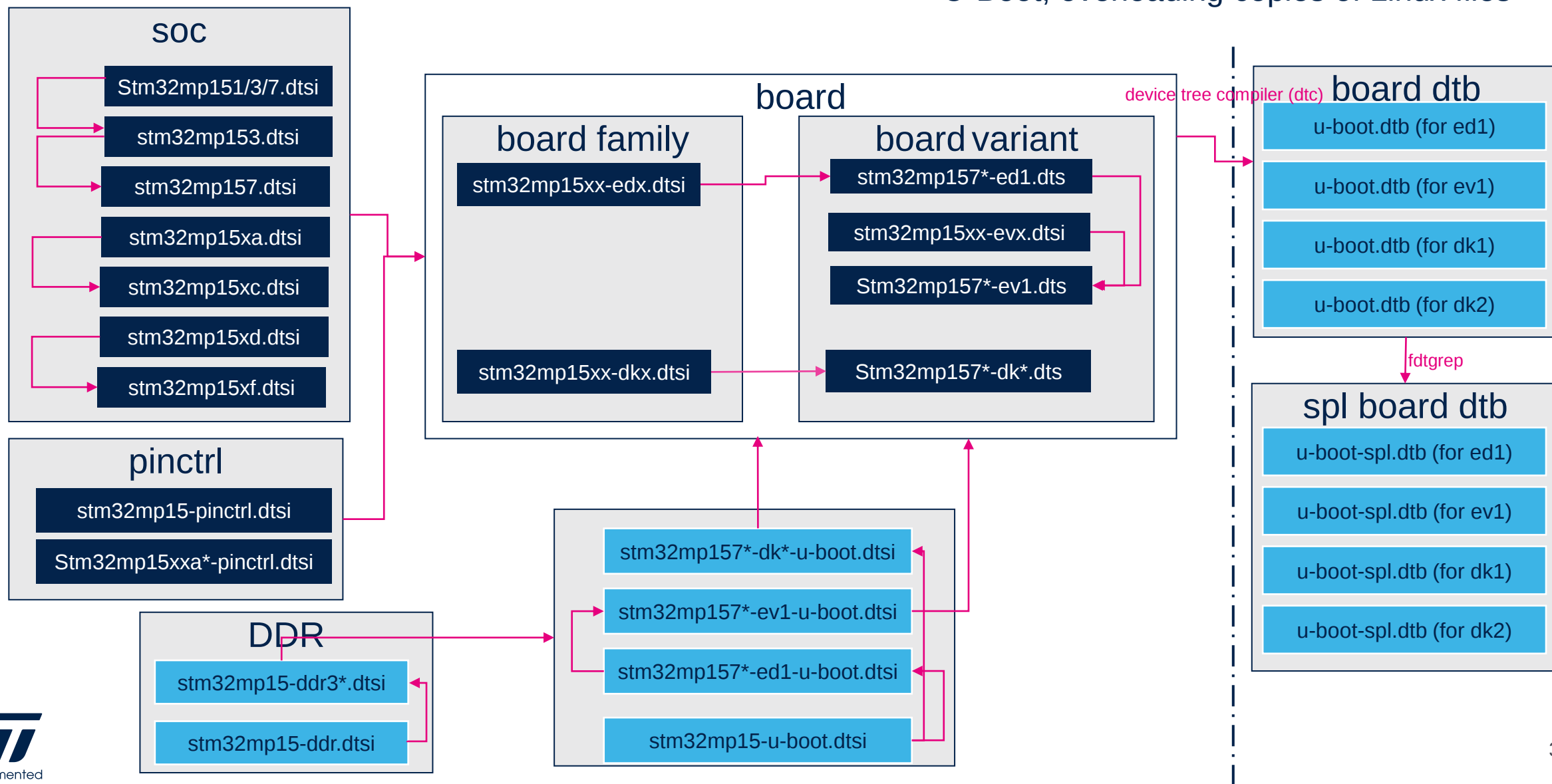


- ed: evaluation daughter board (embedding the STM32MP1)
- ev: evaluation board (ed plugged on the mother board)
- dk: discovery kit

Legend

device tree structure in U-boot

U-Boot, overloading copies of Linux files

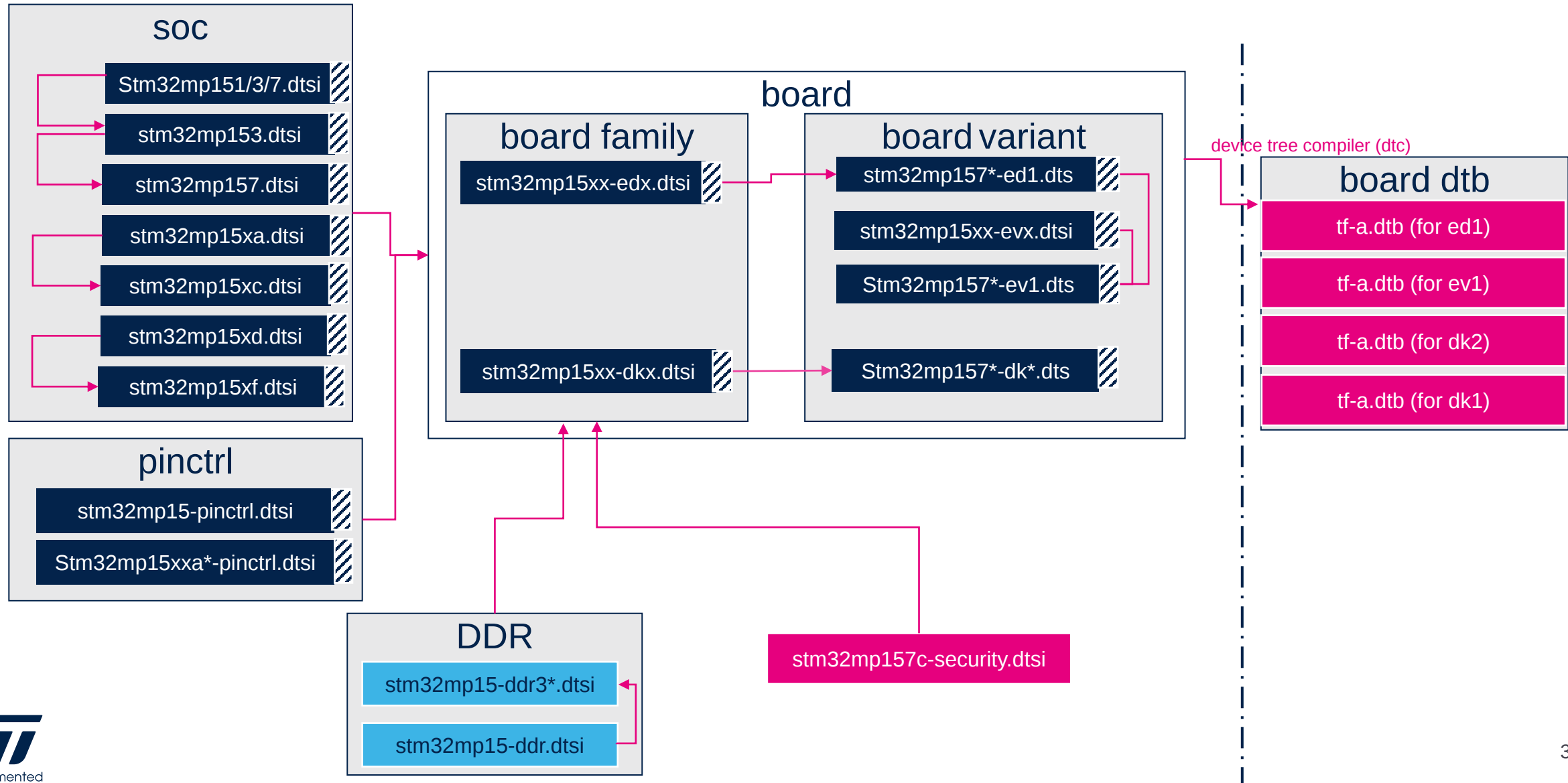


- ed: evaluation daughter board (embedding the STM32MP1)
- ev: evaluation board (ed plugged on the mother board)
- dk: discovery kit

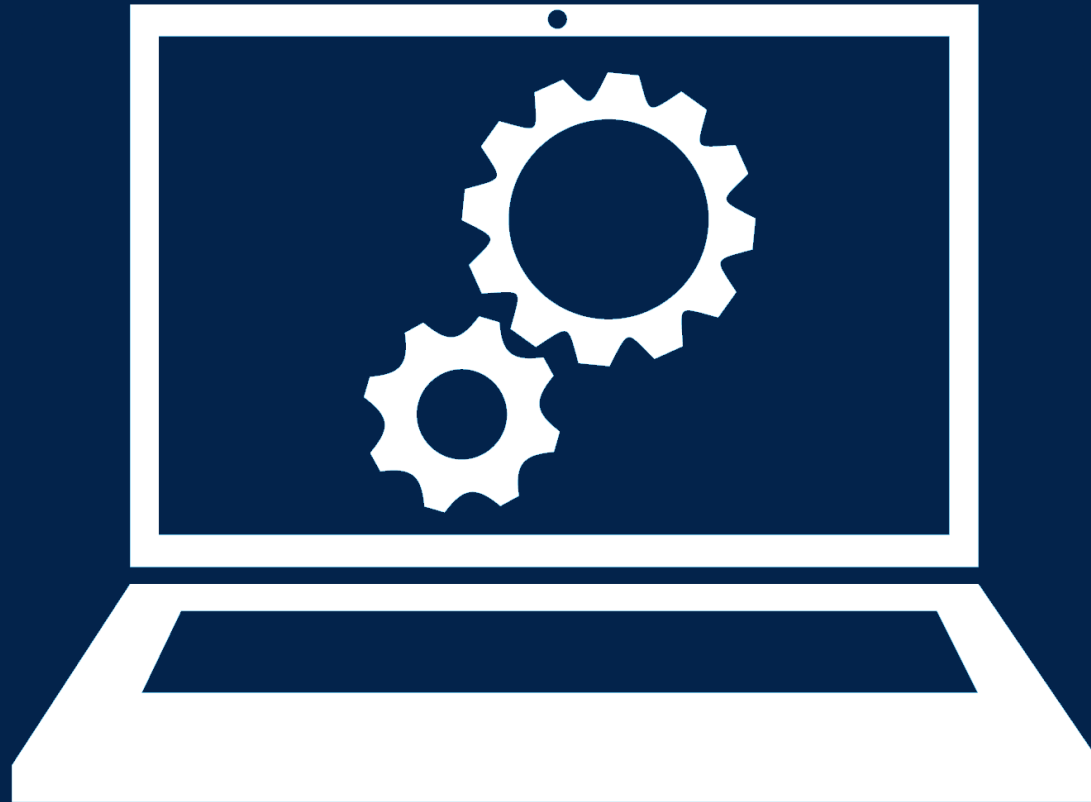
Legend

device tree structure in T-FA

TF-A, overloading subsets of Linux files copies and uses DDR config from U-Boot

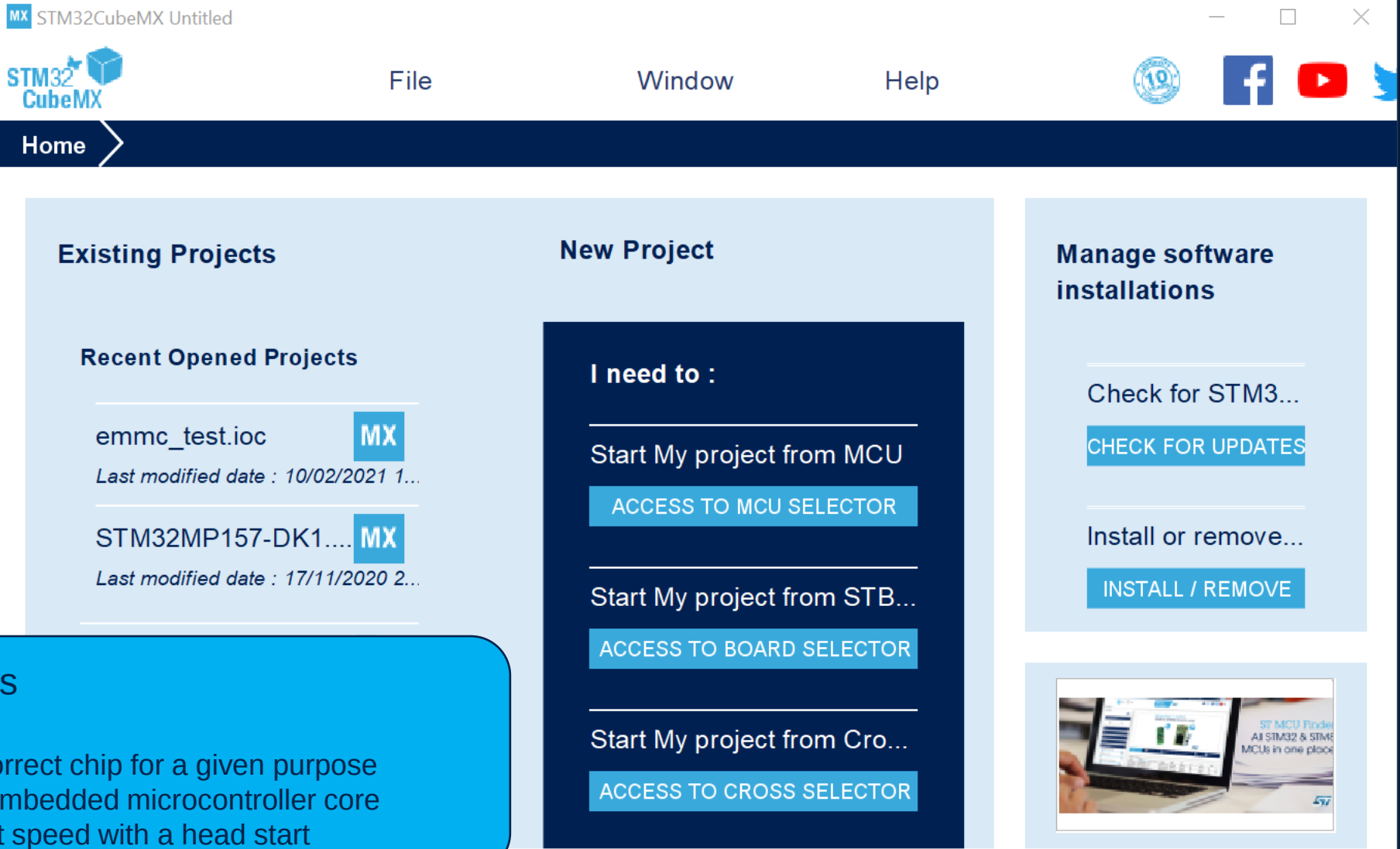


STM32CubeMx



STM32CubeMX Overview

- Choose MPU and configure:
 - Pinouts
 - Clock tree setup
 - Internal peripherals
 - Device tree source files generating
 - Middleware
 - Memory



Application benefits

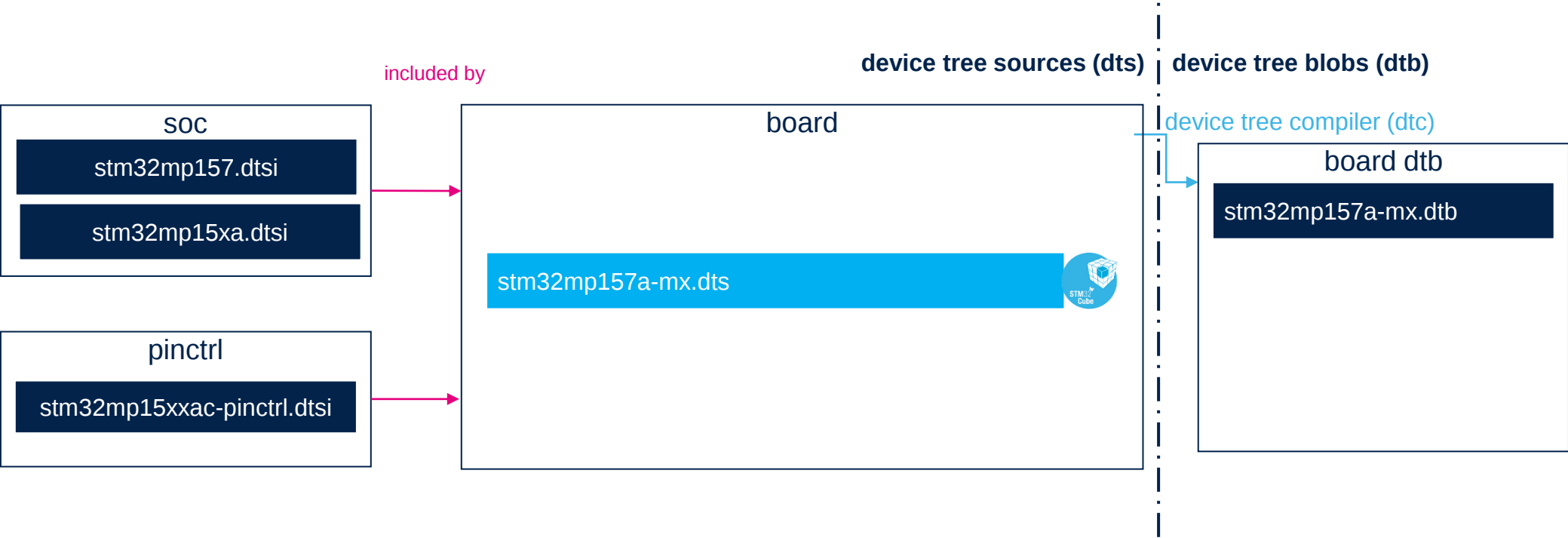
- Helps choose the correct chip for a given purpose
- Generate code for embedded microcontroller core
- Boosts development speed with a head start

STM32CubeMX: Key features

- Peripheral and middleware parameters and assignment to core context
- Power consumption calculator
- Code generation
 - Possible to re-generate HAL code for M4 while keeping user code intact
 - Device tree sources generation for the application core(Linux)
- DDR Test Tool for testing and tuning
- MPU selector
 - Filter by family, package, peripherals or memory sizes
 - Search for similar product
- Pinout configuration
 - Choose peripherals to use and assign GPIO and alternate functions to pins
- Configure NVIC and DMA
- Clock tree initialization
 - Choose oscillator and set PLL and clock dividers

STM32CubeMX: dts generation

Linux

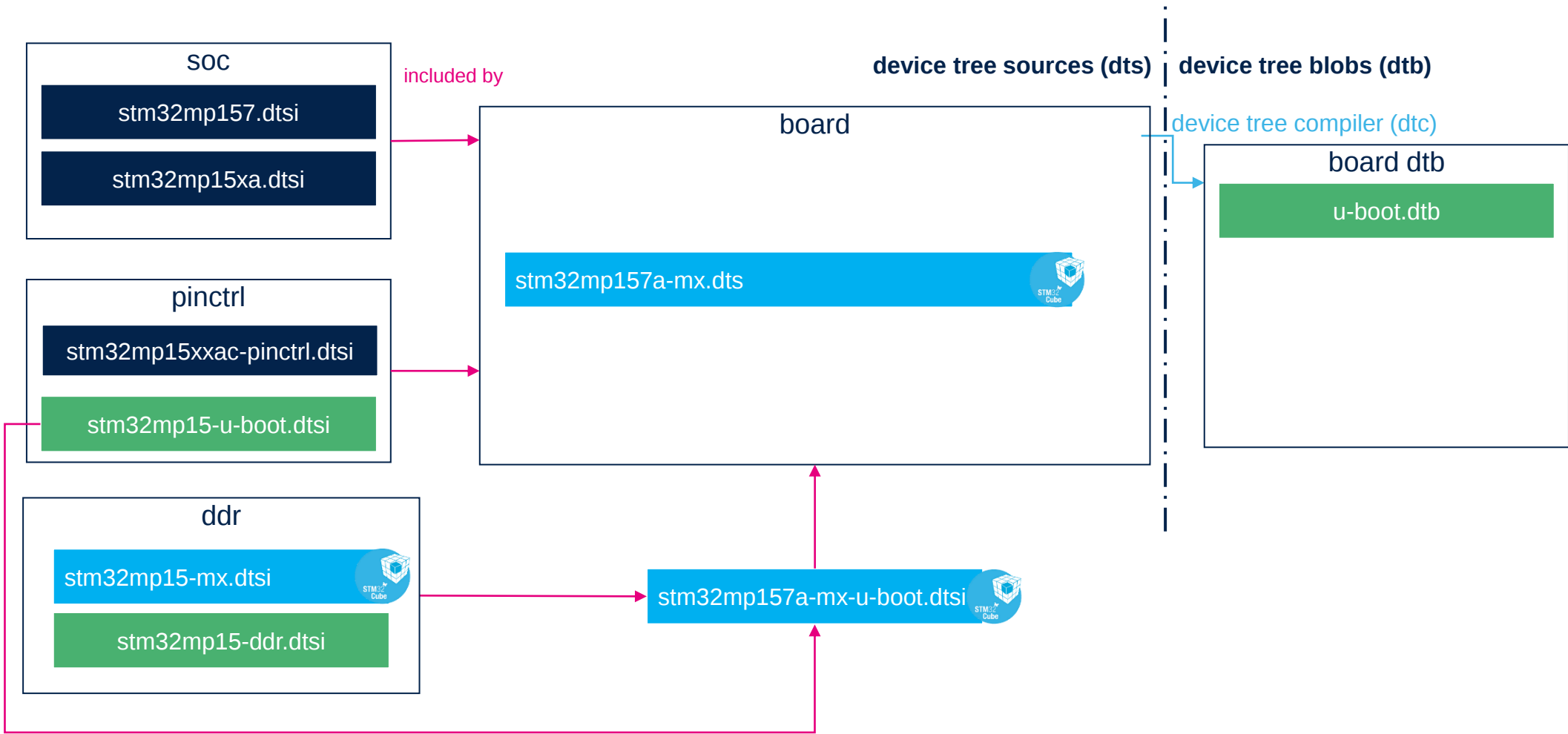


Steps to generate new device tree source in CubeMX are:

- Select the board
- Add the peripheral
- Check/modify the pins use for the peripheral
- Check/modify the clock configuration
- Generate the device tree

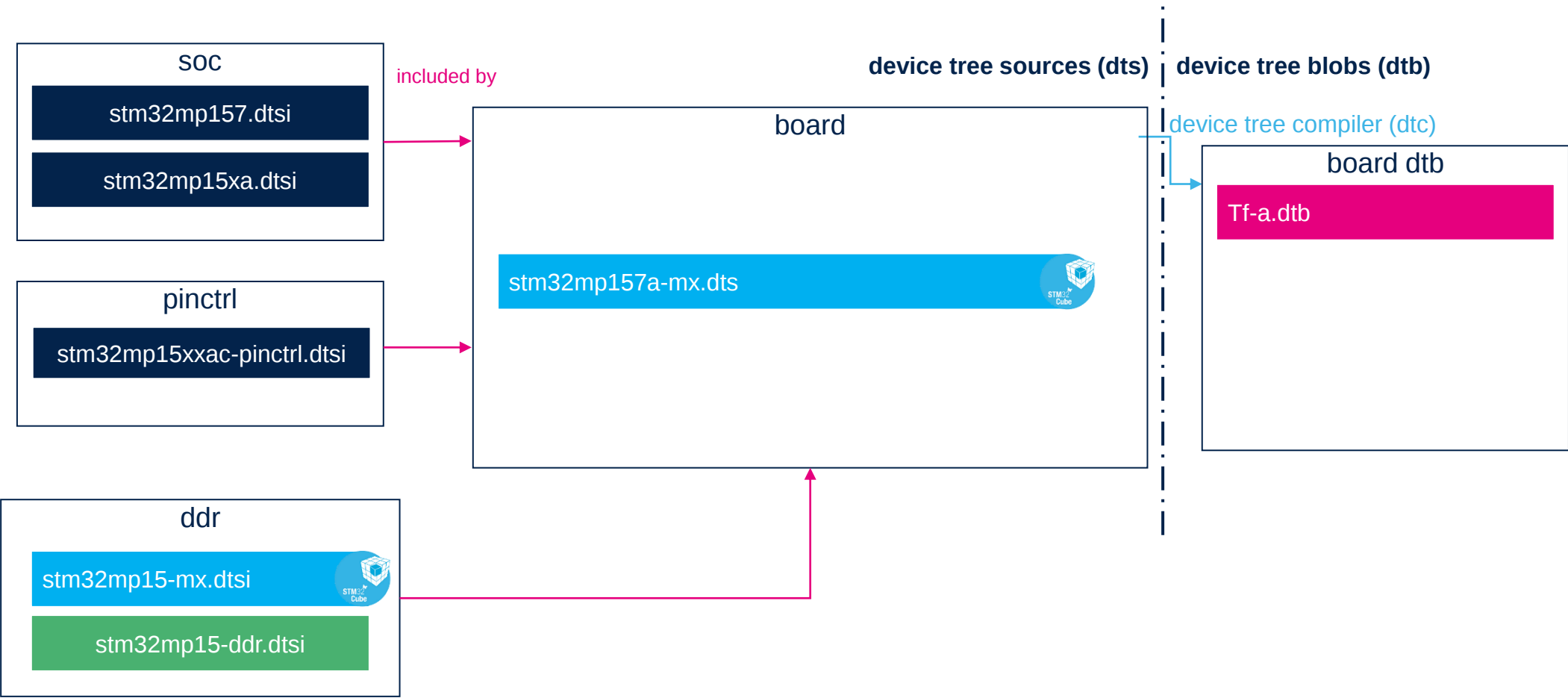
STM32CubeMX: dts generation

U-Boot, overloading copies of Linux files



STM32CubeMX: dts generation

TF-A, overloading subsets of Linux files copies and using DDR config from U-Boot



References



References

Developer package:

https://wiki.st.com/stm32mpu/wiki/STM32MP1_Developer_Package

SDK :

https://wiki.stmicroelectronics.cn/stm32mpu/wiki/SDK_for_OpenSTLinux_distribution

TF-A :

[https://wiki.stmicroelectronics.cn/stm32mpu/wiki/Category:Trusted_Firmware-A_\(TF-A\)](https://wiki.stmicroelectronics.cn/stm32mpu/wiki/Category:Trusted_Firmware-A_(TF-A))

U-boot :

<https://wiki.stmicroelectronics.cn/stm32mpu/wiki/Category:U-Boot>

Linux :

https://wiki.stmicroelectronics.cn/stm32mpu/wiki/Category:Linux_Operating_System

Device tree :

Device tree :

https://wiki.stmicroelectronics.cn/stm32mpu/wiki/Device_tree

Thank you

- How to create an SDK for OpenSTLinux distribution?

- The Distribution Package is installed
- The build environment script has been executed

PC \$ > DISTRO=<distro> MACHINE=<machine> source layers/meta-st/scripts/envsetup.sh

Example:

PC \$ > DISTRO=openstlinux-weston MACHINE=stm32mp1 source layers/meta-st/scripts/envsetup.sh

- The selected image has been rebuilt

PC \$ > bitbake <image>

Example:

PC \$ > bitbake st-image-Weston

- SDK generation

PC \$ > bitbake -c populate_sdk <image>

Example:

PC \$ > bitbake -c populate_sdk st-image-weston

- The **OP-TEE installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi* directory, and is named *optee-os-stm32mp-<OP-TEE version>*:

```
optee-os-stm32mp-3.9.0.r2-r0
├── [*].patch
├── optee-os-stm32mp-3.9.0.r2
├── Makefile.sdk
├── README.HOW_TO.txt
├── series
└── optee-os-stm32mp-3.9.0.r2-r0.tar.gz
```

OP-TEE installation directory

ST patches to apply during the OP-TEE preparation

OP-TEE source code directory

Makefile for the OP-TEE compilation

Helper file for OP-TEE management: reference for OP-TEE build

List of all ST patches to apply

Tarball file of the OP-TEE source code

- Have a look in the TF-A wiki page

https://wiki.st.com/stm32mpu/wiki/STM32MP15_OP-TEE