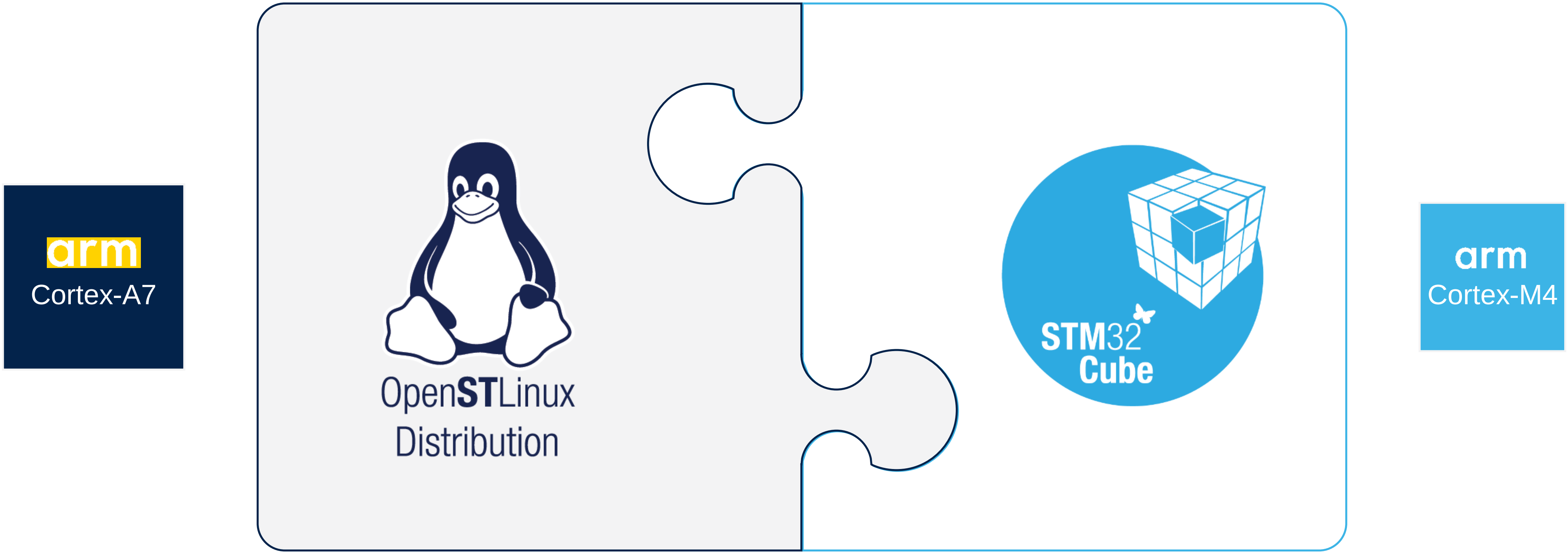
life.augmented

Openstlinux : STM32MP1 workshop

Bossen WU

A fully integrated design suite leveraging the stm32cube environment

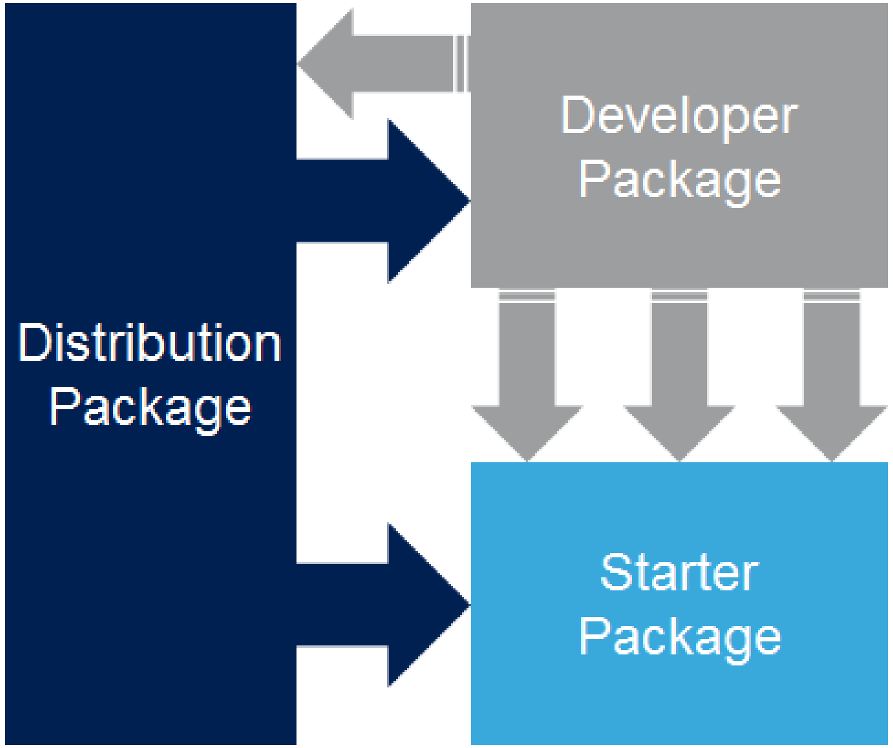


STM32MP1 Embedded Software Distribution

life.augmented

One distribution, three packages

- Open STLinux
- Starter Package
  - To quickly and easily start with any STM32MP1
- Developer Package
  - To add your own developments on top of the STM32MP1 Embedded Software distribution
- Distribution Package
  - To create your own Linux® distribution, your own Starter Package and your own Developer Package

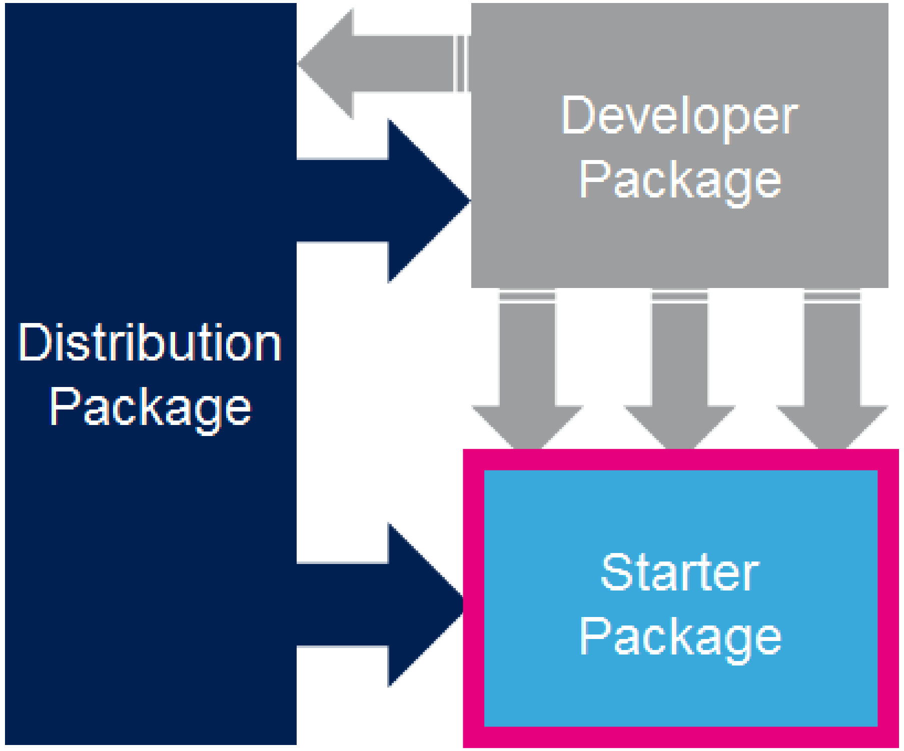


Openstlinux : starter package

Bossen WU

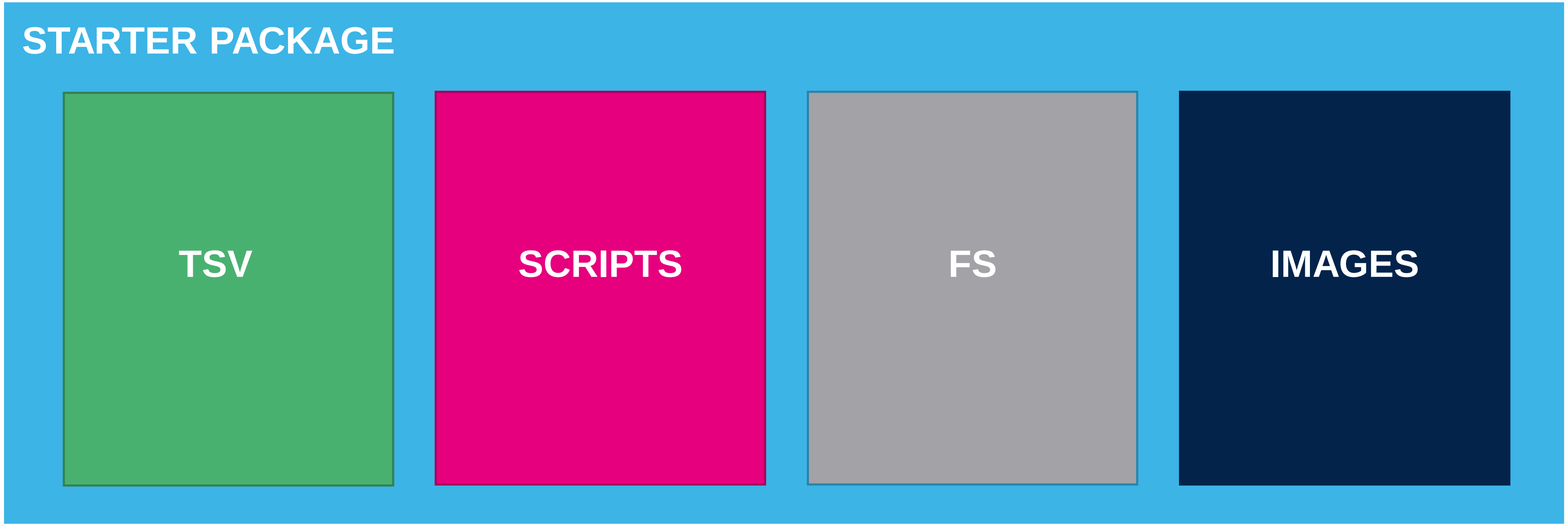
Starter package

- The starter package is the final result that each customer will use in their product based on STM32MP1.
- This hands on focus on the Starter package generated for ST boards.
- The ST starter package can be used as a reference, and help to be ready to analyze and understand the result generated by customer.
- distribution and Developer package will be addressed on later “hands on”.



Starter package : files

- The starter package is composed of different kind of files.



STARTER PACKAGE

TSV

SCRIPTS

FS

IMAGES

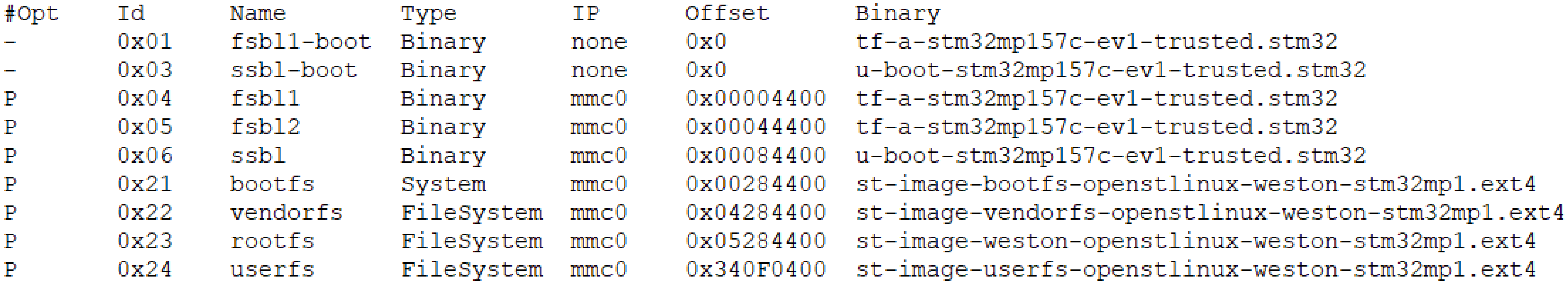
TSV files

- A TSV file is the flash layout describing the flash mapping. STM32CubeProgrammer uses the TSV file as an input configuration file.
- One TSV file describe one product flash configuration, you have to use one TSV for each flash configuration/product.
- TSV file can be modified and edited as a TXT file.
- TSV file can also be use/modify thanks to the STM32Programmer GUI.

TSV files

The Flash layout description can be found in the wiki :

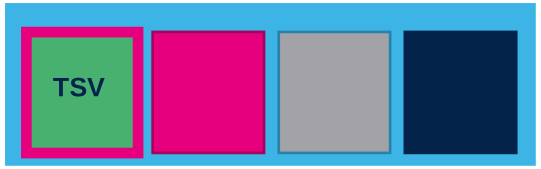
https://wiki.st.com/stm32mpu/wiki/STM32CubeProgrammer_flashlayout



```
#Opt    Id      Name         Type        IP      Offset       Binary
-       0x01    fsbl1-boot   Binary      none    0x0          tf-a-stm32mp157c-ev1-trusted.stm32
-       0x03    ssbl-boot    Binary      none    0x0          u-boot-stm32mp157c-ev1-trusted.stm32
P       0x04    fsbl1        Binary      mmc0    0x00004400   tf-a-stm32mp157c-ev1-trusted.stm32
P       0x05    fsbl2        Binary      mmc0    0x00044400   tf-a-stm32mp157c-ev1-trusted.stm32
P       0x06    ssbl         Binary      mmc0    0x00084400   u-boot-stm32mp157c-ev1-trusted.stm32
P       0x21    bootfs       System      mmc0    0x00284400   st-image-bootfs-openstlinux-weston-stm32mp1.ext4
P       0x22    vendorfs     FileSystem  mmc0    0x04284400   st-image-vendorfs-openstlinux-weston-stm32mp1.ext4
P       0x23    rootfs       FileSystem  mmc0    0x05284400   st-image-weston-openstlinux-weston-stm32mp1.ext4
P       0x24    userfs       FileSystem  mmc0    0x340F0400   st-image-userfs-openstlinux-weston-stm32mp1.ext4
```

TSV files

- OPT field :
  - '-' : no action
  - 'P' : update = program the partition or the flash device
  - 'PE' : do not update (also 'EP') : allow GPT partitioning with empty partition for block device but equivalent to '-' for RAW flash device
  - 'PD' : delete and update (also 'DP’)
  - 'PDE' : delete and keep empty (also 'PED' / 'DPE' / 'DEP' / 'EPD' / 'EDP’)
- Hands on :
  - Create an TSV file to update only u-boot on your board.
  - Create an TSV to delete TF-A (FSBL1 only). Are you still booting? why?
  - Create a TSV to clear all the flash used previously.



TSV files

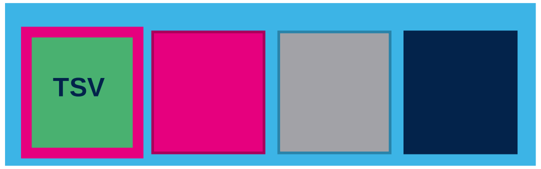
- ID field :
  - Two ranges :
    - 0x01 to 0x0F Boot partitions with STM32 header: SSBL, FSBL, other (TEE, Cortex-M4 firmware)
    - 0x10 to 0xF0 user partitions programmed without header (uimage, dtb, rootfs, vendorfs, userfs)
  - Reserved Id
    - 0x00 FlashLayout (used internally, cannot be used in FlashLayout file)
    - 0x01 FSBL (first copy) : used by ROM code (load in RAM)
    - 0x03 SSBL : used by FSBL=TF-A (load in RAM)
    - 0xF1 to 0xFD "virtual partition": used internally
    - 0xF1 Command GetPhase
    - 0xF2 OTP
    - 0xF3 SSP
    - 0xF4 PMIC NVM
    - 0xFE End of operation
    - 0xFF Reset

TSV files

- Type field :
  - Binary : raw data
  - Binary(N) : raw data duplicate N times (used for MTD only)
  - FileSystem :
    - GPT : Linux filesystem data (EXT4, EXT2, FAT)
    - MTD : raw data
  - System :
    - GPT : Linux fiel system marked as bootable by u-boot
    - MTD : UBI file system
  - RAWImage
    - Write a raw image on all the device.

TSV files

- IP field :
  - Select the targeted device and the instance (starting at 0) as defined by U-Boot device tree
  - mmc + instance : 'mmc0' : It is used for e•MMC or SD card on SDMMC.
  - nor + instance : 'nor0' : It is used for NOR on QUADSPI.
  - nand + instance : 'nand0' : It is used for parallel NAND Flash memories on FMC.
  - none for partition only used to load the binary in RAM
- Several devices can be mixed in the same FlashLayout file.
- Hands on :
  - Create a TSV that used different binary for the communication than the one programed on the flash. This feature can be really interesting for a product.



TSV files

- Offset field :
  - The supported values are:
    - boot1: first boot area partition of eMMC (offset is 0x0)
    - boot2: second boot area partition of eMMC (offset is 0x0)
    - Offset in bytes: offset in Flash memory area (in the user data area for eMMC)
- Hands on :
  - Modify the TSV of your board increasing the rootfs partition size.
  - Can I increase the TF-A partition size?

Scripts

- One script is available to generate a full SDcard image (raw image).
- Used the script.
- Flash the result using the dd command (command can be found in the “how to” txt file generated by the script).
- This tools can be useful when you plug want to update the SD card from your laptop.
- Create a TSV that program the raw generated by the script.

Memory mapping

- Read the article dedicated to the memory mapping on the wiki :

https://wiki.st.com/stm32mpu/wiki/STM32MP15_Flash_mapping

- We are using 4 file systems : bootfs, vendorfs, rootfs and userfs
- We are using two partition method : GPT and UBI
- We are using two kind of file system : EXT4 and UBIFS
- FSBL and SSBL raw binaries are inside a partition or at a given offset (NOR).

Flash partitions (minimal)

Size	Component	Comment
Remaining area	userfs	The user file system contains user data and examples
768MB	rootfs	Linux root file system contains all user space binaries (executable, libraries, …) and kernel modules
16MB	vendorfs	This partition is preferred to the rootfs to put third parties proprietary binaries and ensure that they are not contaminated by any open source licence, such as GPL v3
64MB	bootfs	The boot file system contains: - (option) the init ram file system, that can be copied to the external RAM and used by Linux before mounting a fatter rootfs - Linux kernel device tree (can be in a Flattened Image Tree - FIT) - Linux kernel U-Boot image (can be in a Flattened Image Tree - FIT) - For all flashes but the NOR: the boot loader splash screen image, displayed by U-Boot - U-Boot distro config file extlinux.conf (can be in a Flattened Image Tree – FIT)
2MB	ssbl	The Second Stage Boot Loader (SSBL) is U-Boot, with its device tree blob (dtb) appended at the end
256kB to 512kB (*)	fsbl	The First Stage Boot Loader is ARM Trusted Firmware (TF-A) or U-Boot Secondary Program Loader (SPL), with its device tree blob (dtb) appended at the end. At least two copies are embedded. Note: due to ROM code RAM needs, FSBL payload is limited to 247kB.

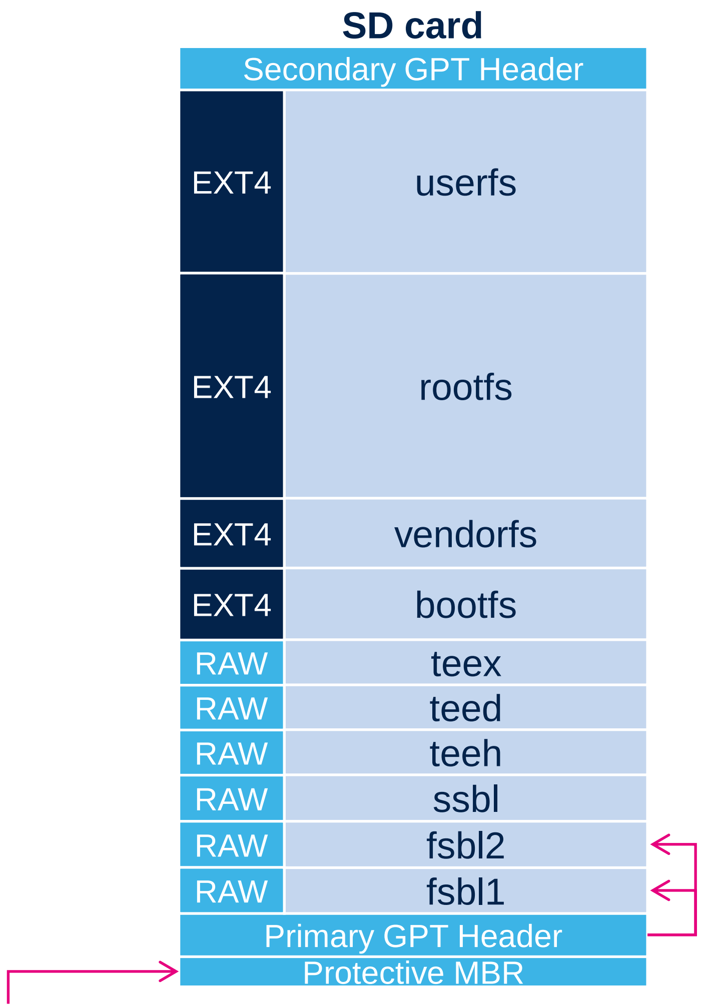
(*): the partition size depends on the flash technology, to be aligned on block erase size for NOR (256kB) / NAND (512kB)

Flash partitions (optional)

Size	Component	Comment
256kB (*)	logo	This partition contains the boot loader splash screen image while booting on NOR flash (for all other flashes, the image is stored in the bootfs partition)
256kB to 512kB (*)	teeh	OP-TEE header
256kB to 512kB (*)	teed	OP-TEE pageable code and data
256kB to 512kB (*)	teex	OP-TEE pager

(*): the partition size depends on the flash technology, to be aligned on block erase size for NOR (256kB) / NAND (512kB)

SD card memory mapping

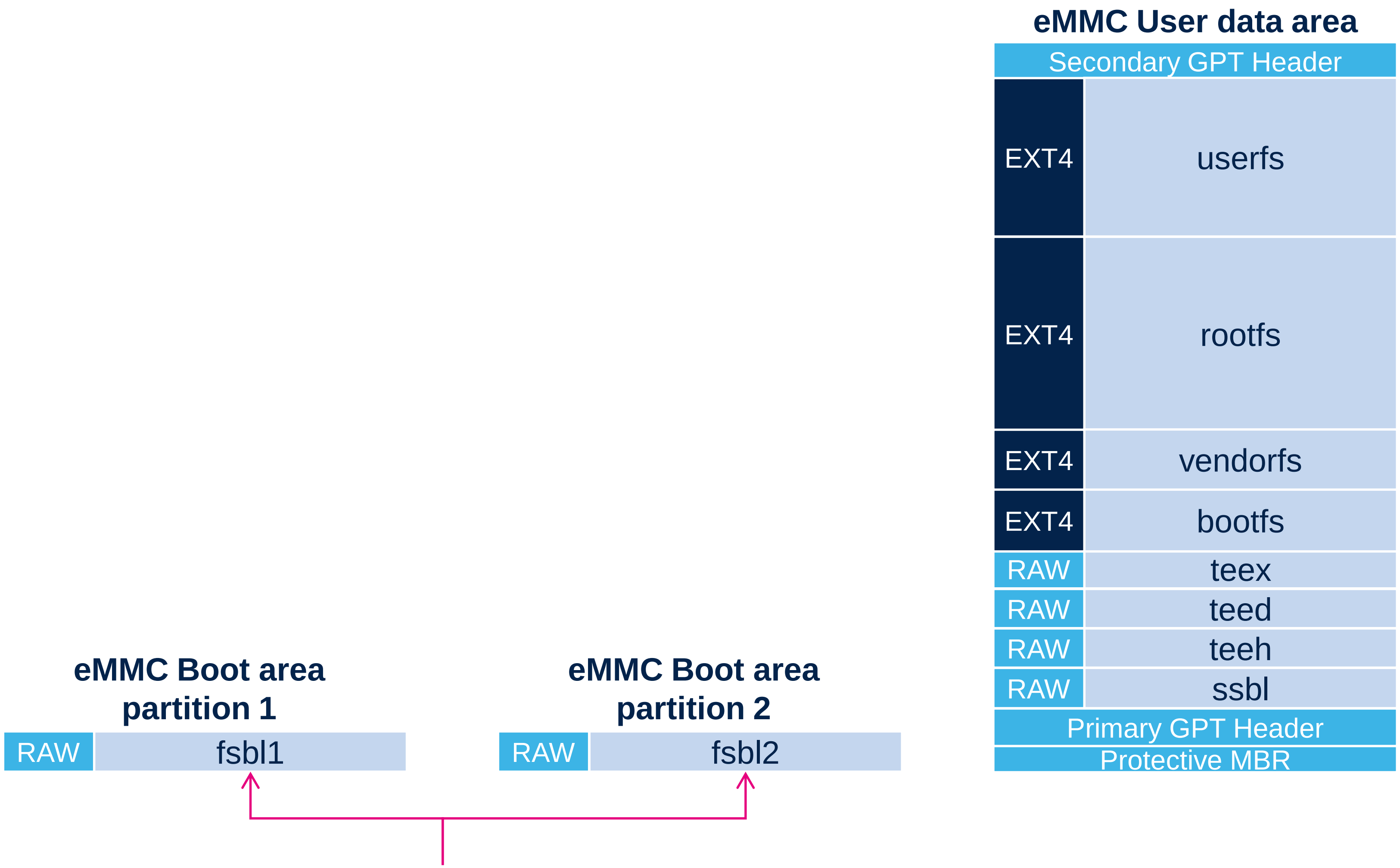


SD card

Secondary GPT Header	
EXT4	userfs
EXT4	rootfs
EXT4	vendorfs
EXT4	bootfs
RAW	teex
RAW	teed
RAW	teeh
RAW	ssbl
RAW	fsbl2
RAW	fsbl1
Primary GPT Header	
Protective MBR	

ROM code path to look for FSBL

Emmc memory mapping



eMMC User data area

Secondary GPT Header	
EXT4	userfs
EXT4	rootfs
EXT4	vendorfs
EXT4	bootfs
RAW	teex
RAW	teed
RAW	teeh
RAW	ssbl
Primary GPT Header	
Protective MBR	

eMMC Boot area partition 1

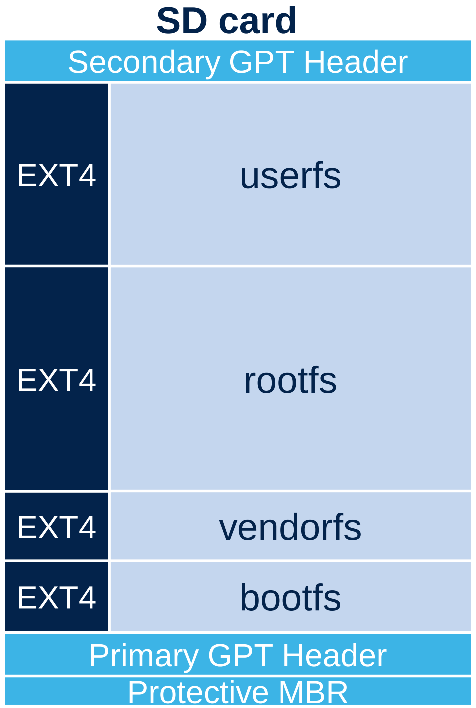
RAW	fsbl1

eMMC Boot area partition 2

RAW	fsbl2

ROM code path to look for FSBL

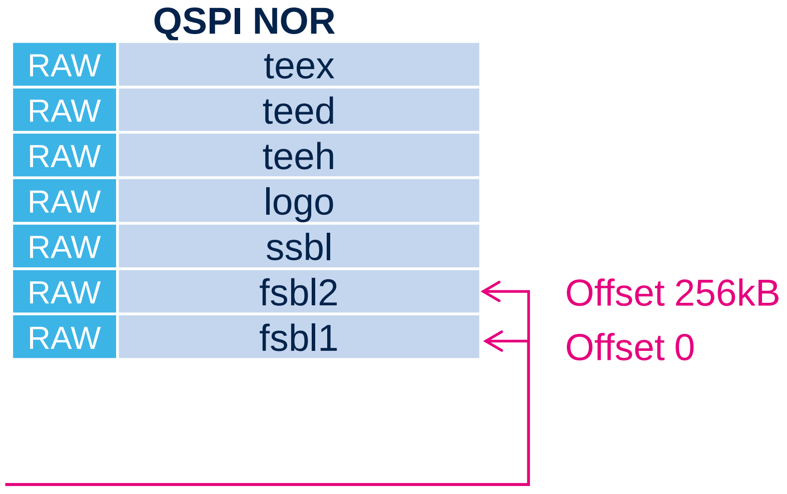
NOR memory mapping



SD card

Secondary GPT Header	
EXT4	userfs
EXT4	rootfs
EXT4	vendorfs
EXT4	bootfs
Primary GPT Header	
Protective MBR	

Note: SD card used as second stage boot device because the NOR flash is too small to contain Linux file systems. It is possible to use another second stage boot device, like eMMC or NAND.



QSPI NOR

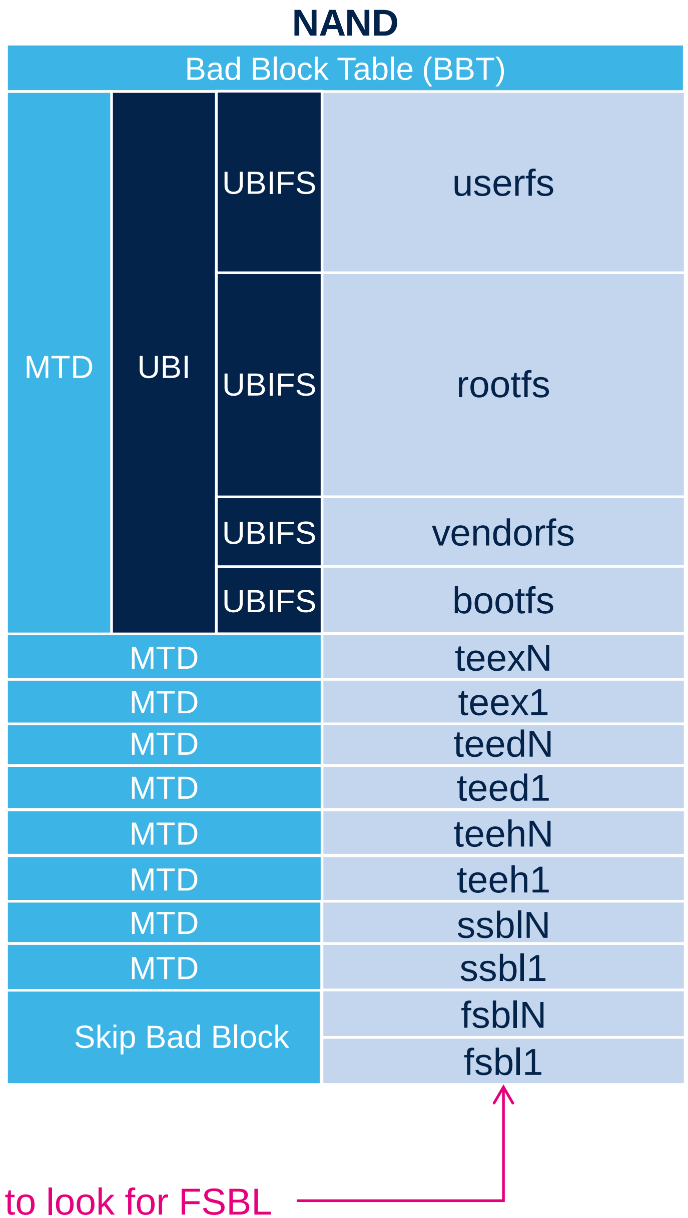
RAW	teex
RAW	teed
RAW	teeh
RAW	logo
RAW	ssbl
RAW	fsbl2
RAW	fsbl1

Offset 256kB

Offset 0

ROM code path to look for FSBL

NAND memory mapping



NAND

Bad Block Table (BBT)			
MTD	UBI	UBIFS	userfs
		UBIFS	rootfs
		UBIFS	vendorfs
		UBIFS	bootfs
MTD			teexN
MTD			teex1
MTD			teedN
MTD			teed1
MTD			teehN
MTD			teeh1
MTD			ssblN
MTD			ssbl1
Skip Bad Block			fsblN
			fsbl1

ROM code path to look for FSBL

Note: SSBL and OP-TEE partitions may move to UBI format when FSBL supports it

Note: in the Skip Bad Block area, the number of copies and the margin have to defined in STM32CubeProgrammer flash layout, depending on the product expected life time and firmware update strategy

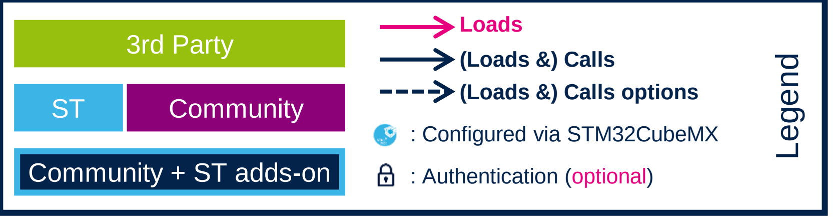
EXT4 file system

- EXT4 is file system type used mainly by linux :
  - https://en.wikipedia.org/wiki/Ext4
- Try to mount each ext4 file system in your laptop thanks to the mount command:
  - mkdir /tmp/userfs
  - mount ./st-image-userfs-openstlinux-weston-stm32mp1.ext4 /tmp/userfs
- Add a file inside and update it on your board thanks to the programmer.
- Check on the board that your new file is here.

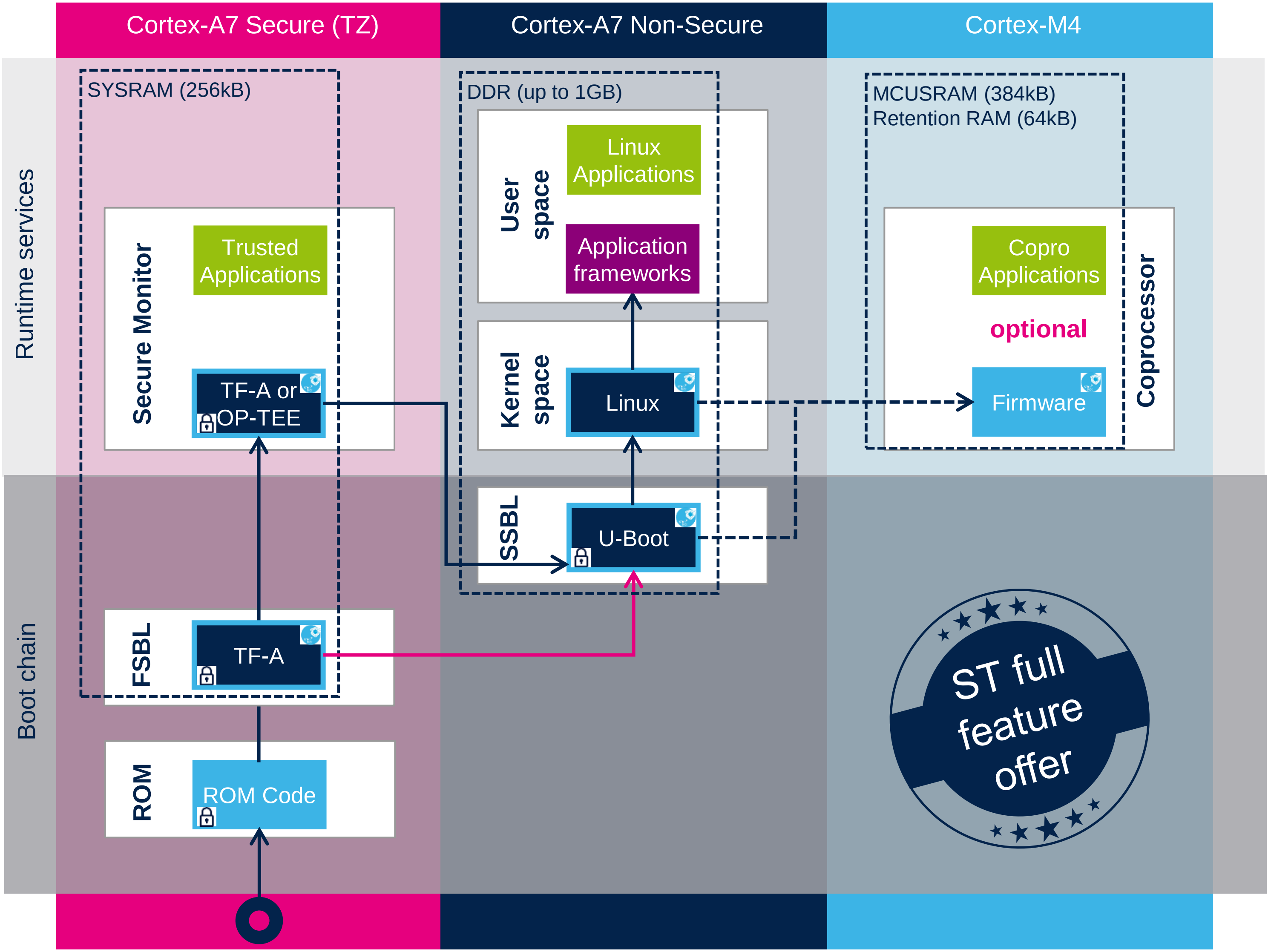
UBI file system

- UBIFS is file system type used in linux to manage mainly Nand flash :
  - https://en.wikipedia.org/wiki/UBIFS
- The ubi file system in the starter package has been generated for the evaluation board NAND
  - https://4donline.ihs.com/images/VipMasterIC/IC/MICT/MICTS03969/MICTS03969-1.pdf
- With the given command and the information inside the datasheet mount the UBI file system on your laptop.
- Commands :
  - modprobe nandsim id_bytes=NAND-ID
  - ubiformat /dev/mtd0 –s page-size -f ./st-image-weston-openstlinux-weston-stm32mp1_nand_4_256_multivolume.ubi
  - modprobe ubi
  - ubiattach -O page-size /dev/ubi_ctrl -m 0

Trusted boot chain



TF-A is:
-BSD licence
-Trusted writing
-ARMv8 future proof



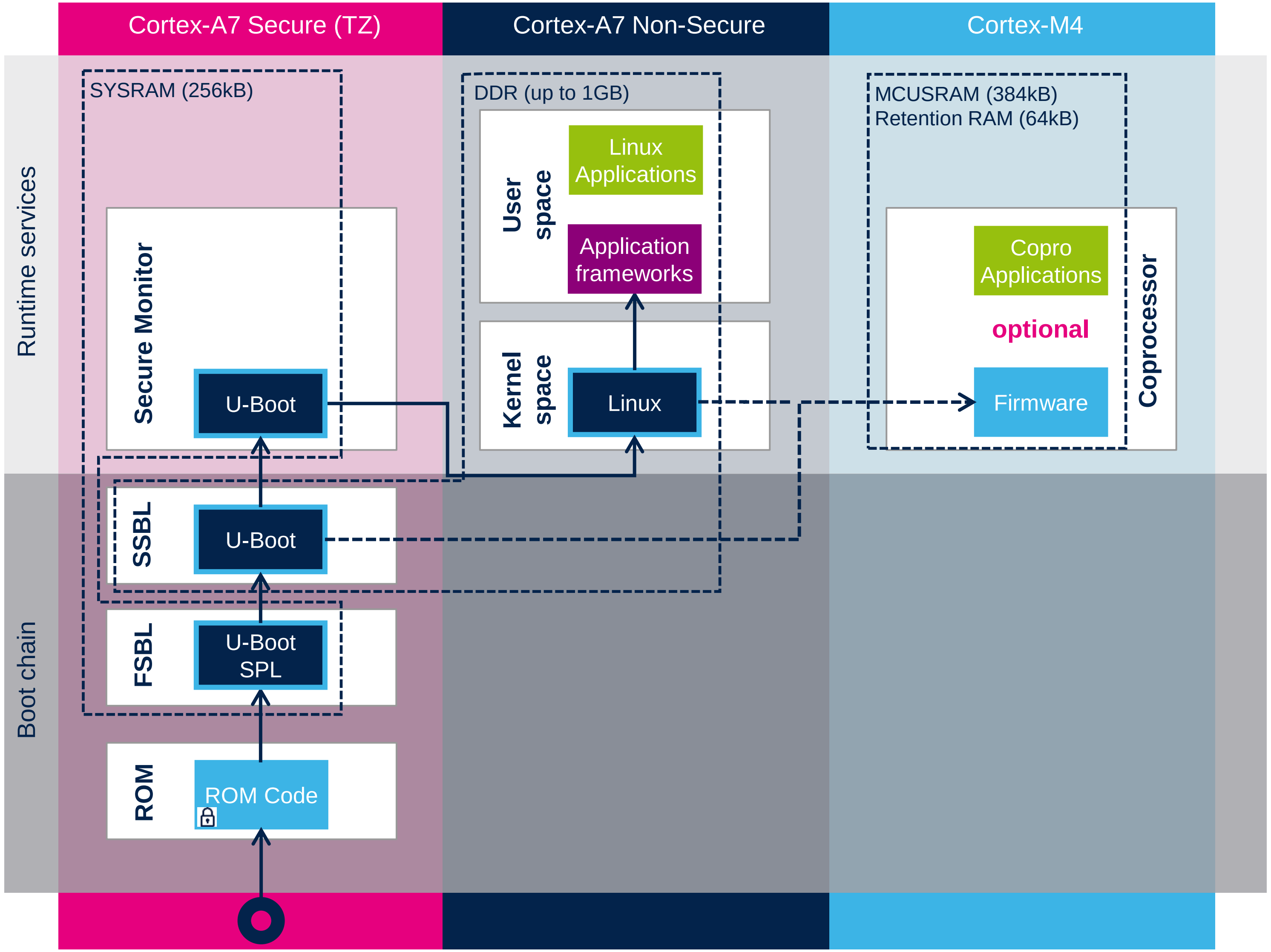
Secure Monitor: Trusted Firmware for Cortex-A (TF-A) Secure Monitor is used if there is no Secure OS (OP-TEE is optional)

life.augmented

Basic boot chain

Legend

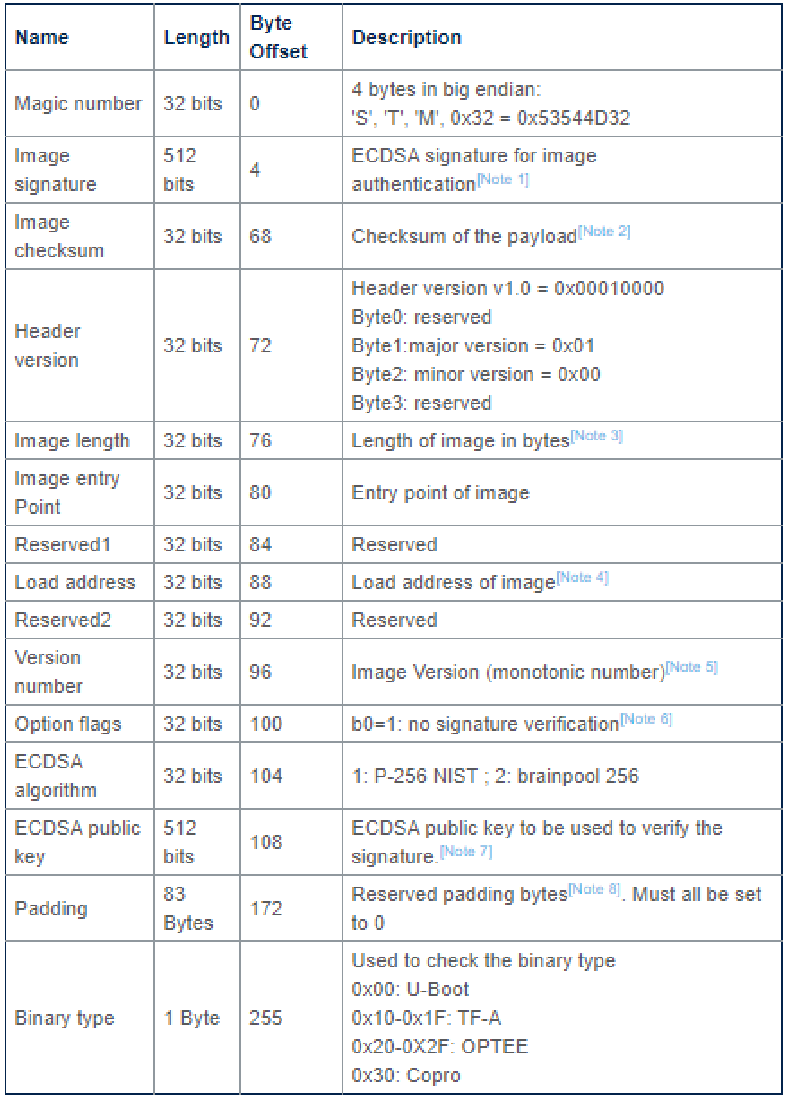
- 3rd Party
- ST
- Community
- Community + ST adds-on
- → Loads
- → (Loads &) Calls
- - - → (Loads &) Calls options
- : Configured via STM32CubeMX
- 🔒 : Authentication (optional)



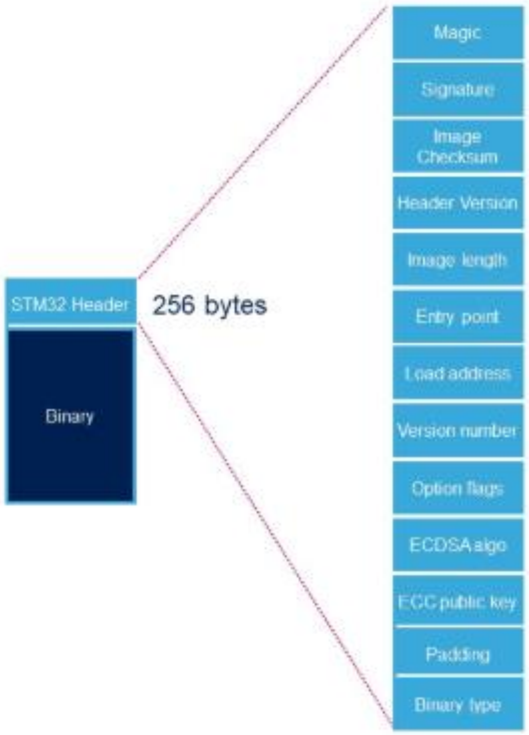
Img & stm32

- Two kind of raw binares :
  - the .STM32 binary and the .IMG binary.
- Use the command “file” on both file type.
  - file ./xxxx.stm32
  - file ./xxxx.img
- Have a look on the secure boot wiki page :
  - https://wiki.st.com/stm32mpu/wiki/STM32MP15_secure_boot
- SPL is a IMG or STM32 ?
  - Use hexdump command to check if the STM32 header is here or not :
  - hexdump -C -n 1024 ./u-boot-spl.stm32

Img & stm32



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication[Note 1]
Image checksum	32 bits	68	Checksum of the payload[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1:major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image[Note 4]
Reserved2	32 bits	92	Reserved
Version number	32 bits	96	Image Version (monotonic number)[Note 5]
Option flags	32 bits	100	b0=1: no signature verification[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature.[Note 7]
Padding	83 Bytes	172	Reserved padding bytes[Note 8]. Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0X2F: OPTEE 0x30: Copro



License & manifest

- Have a look inside the License file.
  - Why there is no GPL license here?
- https://wiki.st.com/stm32mpu/wiki/OpenSTLinux_licenses
- Vendorfs is created to separate 3rd party code (library) from GPLv3 code. For example Gcnano library can’t be in contact with GPLv3.
  This architecture and implementation has been done following ST legals interpretation of Gcnano licence. Customer legal service interpretation can be different.
- What is the manifest file? Look inside (“more” command).
  - Which python version we are using?

Console

- A console allow you to interact with Linux and U-boot, and also allow to get some information on what is going on in the system.
- This is most important tools to debug.
- Open a console using UART4 on your board, using your favorite terminal (minicom, putty…)
- On Linux the default terminal used is minicom :
  - minicom -D /dev/ttyACM0
- In case of trouble you can read the wiki page :
  - https://wiki.st.com/stm32mpu/wiki/How_to_get_Terminal

life.augmented

Console

- There is several ways to open a terminal, please try all :
  - You can also use the local terminal connecting a mouse and a keyboard directly to the board (terminal icon on bottom left). You can use the touchscreen to open it and close it.
  - Connect the board thanks to an Ethernet cable and use SSH on your laptop to open a terminal.
    - https://wiki.st.com/stm32mpu/wiki/How_to_perform_ssh_connection
    - “Ifconfig” on the target to get IP address
  - On the target, the USB ethernet gadget is used and not the mass storage gadget. So you can also use a USB cable to open an SSH command on your target.
    - Plug the USB OTG cable on your laptop
    - Ifconfig interface_name 192.168.7.2
    - Use SSH command : ssh root@192.168.7.1
- We can also use other UART or wifi to open an terminal

Transfer file to the board

- Transfer an image (jpg) from your laptop to the board and display it.
- Transfer the image using the UART link : (not recommended)
  - https://wiki.st.com/stm32mpu/wiki/How_to_transfer_a_file_over_serial_console
- Transfer the image using the Ethernet link or USB Ethernet gadget:
  - scp filename root@<board ip address>:/usr/local
- Transfer the image thanks to USB mass storage feature in u-boot.
  - https://wiki.st.com/stm32mpu/wiki/How_to_use_USB_mass_storage_in_U-Boot
- You can also plug the SD-card in your Linux laptop or add the file before to flash the files system.

GPIO hands on

- Use the libgpiod to control the LED on GPIOA13 and GPIOA14 in your board:
- Commands :
  - gpiodetect
  - gpiofind
  - gpioget
  - gpioinfo
  - gpiomon
  - gpioset
- Write one command line that set the GPIOA14 depending of GPIOA13 input state.

STM32 MP1

Releasing your creativity with the STM32

/STM32

@ST_World

community.st.com

Famous video here

www.st.com/stm32

life.augmented

Quizz

- What is the stm32 extension?
- What is the img extension?