STM32CubeProgrammer flashlayout

Contents

1 Article purpose

This article describes the FlashLayout file format.

This file is used as an input by STM32CubeProgrammer tool in order to:

1. define the Flash memory partitions (see STM32MP15_Flash_mapping)
2. select the files used to boot (see Boot chains overview) and then populate each partition.

The embedded programming service processes this file on the device and interacts with STM32CubeProgrammer to update the Flash memory.
This is done by the **stm32prog** command in U-Boot. This command is automatically executed for a serial boot (on USB or UART). However you can launch it manually from the U-Boot console: this is useful if you want to perform a non-virgin board update front of STM32CubeProgrammer, without manipulating the boot pins.

See AN5275: USB DFU/USART protocols used in STM32MP1 Series bootloaders for protocol details and refer to STM32CubeProgrammer article to know how to use this tool.

The next chapters:

- describe the FlashLayout file format
- give Typical FlashLayout file for boot scenario without TEE
- give Typical FlashLayout file with TEE
- give FlashLayout file to load and start kernel that can be used to load a Linux kernel for direct execution
- give some hints on the Using of the provided FlashLayout files that come as part of STM32 MPU ecosystem deliveries

The FSBL and SSBL definitions can be found in the Boot chains overview.

2 FlashLayout file format

The FlashLayout is a text file with a tab-separated-value format (tsv). It includes the below elements:

- one line per partition or device
- seven columns, one for each field, provided in the following order:
  - Options
  - Id
  - Name
  - Type
  - Device
  - Offset
  - Binary

The lines beginning with the '#' character are ignored and treated as comments.

The "Binary" last column is not used by U-Boot. It is used by STM32CubeProgrammer on the host computer to select the files to be sent to the target.

Several tabulations (<tab>) can be used to allow the correct column alignment in the editor. They are ignored by STM32CubeProgrammer and by U-Boot.

Empty fields are not allowed. The FlashLayout file format supports the none reserved word.

2.1 Examples

Below some valid FlashLayout files:

```
#opt        Id        Name        Type                Device        Offset
Binary
P        0x01        fsbl1        Binary                mmc0        0x00004400
fsbl.stm32
P        0x02        fsbl2        Binary                mmc0        0x00044400
fsbl.stm32
P        0x03        ssbl        Binary                mmc0        0x00084400        ssbl.
stm32
P        0x10        bootfs        System                mmc0        0x00284400
bootfs.ext4
P        0x11        rootfs        FileSystem        mmc0        0x08284400        rootfs.
ext4
PE        0x12        userfs        FileSystem        mmc0        0x28284400        none
```

Above, the first line contains only header information. This is not mandatory, as shown below:

```
-        0x01        fsbl_boot        Binary                none        0x0        fsbl.
stm32
-        0x03        ssbl_boot        Binary                none        0x0        ssbl.
stm32
P        0x32        sdcard                RawImage        mmc0        0x0        sdcard.
bin
```

2.2 Field1: Options

The Options field defines the operations to perform with a combination of characters: - P D E provided in any order;
First select the line of the FlashLayout with '-' or 'P':

- '-' : **none** option, the partition or the device is not modified (mandatory if #Field5: Device = none)
- 'P' : **Program the partition or the device**

  U-Boot requests the binary to STM32CubeProgrammer and programs the partition or the Flash device.

  On the block devices (SD card or *e*•MMC), the GPT partitioning is performed if all partitions of the device are selected with P.

  For the 'P' option, two optional modifiers can be added:
  - 'E' : **Empty partition or device**, update is not requested (associated "Id" is skipped)
  - 'D' : **Delete partition or device**

The only supported combinations are the following (with character in any order):

- - : **no action**
- P : **update** = program the partition or the flash device
- PE : **do not update** (also {{HighlightParam|EP}) : allow the GPT partitioning with empty partition for the 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)

All other combinations are invalid.

2.3 Field2: Id

Id identifies in a unique way the "download phase" requested by the device to STM32CubeProgrammer.
It is used by the embedded programming service to identify the next binary that is downloaded to the device:

- ROM code and FSBL: binary loaded in RAM
- SSBL (U-Boot): binary populated in Flash memory

The FlashLayout supported ranges are :

Range	Partition
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)

The Id 0x01 and 0x03 are reserved for FSBL and SSBL, respectively. They are loaded in RAM by ROM code and by FSBL.

2.3.1 Reserved Id

The reserved values are the following:

Code	Partition
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)

Code	Partition
0xF1 to 0xFD	"virtual partition": used internally
0xF1	Command GetPhase
0xF2	OTP
0xF3	SSP
0xF4	PMIC NVM
0xFE	End of operation
0xFF	Reset

2.3.2 Using Id for boot partition

The same FSBL and SSBL binaries can be loaded in RAM and programmed in Flash memory. Then a simple mapping is used:

Code	Partition
0x01 (reserved)	FSBL (first copy)
0x02 (default)	FSBL (second copy)
0x03 (reserved)	SSBL

However, in the FlashLayout file, any other Id lower than 0x10 (boot partition with STM32 header) can identify the FSBL and SSBL binaries to be programmed in Flash memory.
It enables having different binaries loaded in RAM and programmed in Flash memory, for example when an updated feature is deactivated in the binaries to be programmed in Flash memory.

You can then use the boot partitions (see example in #Load and program different binaries)

Code	Partition
0x01 (reserved)	FSBL to boot : loaded by ROM code
0x03 (reserved)	SSBL to boot : loaded by FSBL
0x02	FSBL to program in Flash memory (first copy)
0x04	FSBL to program in Flash memory (second copy)
0x05	SSBL to program in Flash memory

This behavior is needed for the STM32 MPU boot chain with Trusted Firmware-A (TF-A) as FSBL and U-Boot as SSBL; The used mapping is:

Code	File	Description
0x01 (reserved)	tf-a-serialboot.stm32	FSBL to boot : TF-A with serial load support
0x03 (reserved)	u-boot.stm32	SSBL to boot and to program
0x02	tf-a.stm32	FSBL to program: first copy

Code	File	Description
0x04	tf-a.stm32	FSBL to program: second copy

2.4 Field3: Name

Name of the alternate setting of the USB DFU[1] for U-Boot enumeration. This is a string descriptor that indicates the target memory segment (see *Interface Descriptor* in DFU spec [2])

This is also the name of the Block device GPT partition: SD card / *e*•MMC.

The requirements for the GPT partitions are:

- FSBL for SD card boot: the name must start with **"fsbl"**= fsbl, fsbl1, fsbl2... (ROM code requirement)
- SSBL for *e*•MMC/SD card boot: the name must be **"ssbl"** (TF-A requirement)
- TEE partition names (when present) must use the names expected by TF-A :
  - OPTEE_HEADER_IMAGE_NAME **"teeh"**
  - OPTEE_PAGED_IMAGE_NAME **"teed"**
  - OPTEE_PAGER_IMAGE_NAME **"teex"**
- a partition named **"rootfs"**, see specific behavior described in #GPT partuuid of rootfs partition.

These requirements are not verified by U-Boot during the Flash programming. If they are not fulfilled, the ROM code or TF-A does not find the boot stage binary and the boot from Flash memory fails.

2.5 Field4: Type

Type is only used in U-Boot to select the part of Flash memory to be updated:

- one partition
  - Block device GPT partition: SD card / *e*•MMC
  - #Raw Flash device (NAND/NOR Flash memories) MTD partition : see MTD overview
- all the #Hardware device = **RawImage**

The supported values are:

	GPT		MTD		
Type	**SD card**	***e*•MMC**	**NAND Flash memory**	**NOR Flash memory**	**RAM**
Binary	x	x	x	x	x
Binary(N)			ssbl		
FileSystem	x	x	x	x	dtb
System	x	x	UBI	UBI	kernel
RawImage	x	user data	x	x	

2.5.1 Block device GPT partition: SD card / *e*•MMC

Refer to GPT standard for details [3].

The supported values, with associated partition type GUID (globally unique identifiers[4]), are:

- **Binary** : raw data / linux reserved

(GUID = 8DA63339-0007-60C0-C436-083AC8230908)

- FileSystem : Linux filesystem data

(GUID = 0FC63DAF-8483-4772-8E79-3D69D8477DE4)

ext2/ext4/fat file system

- System : *FileSystem* partition marked as bootable and used by U-Boot to find extlinux.conf configuration file (normally only one in the device, generic DISTRO feature)

For a Block device, the GPT header is updated only if all the partitions of this device are selected with the P option (full update).

2.5.2 Raw Flash device (NAND/NOR Flash memories) MTD partition

The supported values are:

- Binary: raw data, skip bad block (partition erase is not needed)
- Binary(N): raw data, skip bad block. The loaded binary is repeated N times.

It is only supported for NAND Flash memories. It is used to avoid disturbances during the first boot (uncorrectable ECC errors). The first good block is read from NAND and duplicated N times in the same partition (write skip bad block).

- FileSystem: unspecified File system, raw data
- System : normally UBI volume, U-Boot erases all the blocks following the last data in the MTD partition to avoid mount errors.

2.5.3 Hardware device

Export the associated device as one alternate setting by using Type=RawImage.

- For SD card, NOR and NAND Flash memories: all the devices
- For *e*•MMC: the user data area of *e*•MMC (see #Field6: Offset for access to the boot area partitions)

For RawImage, Offset=0x0 and PartId >= 0x10

2.5.4 RAM device (DDR)

The supported values are:

- Binary: raw data
- FileSystem: device tree used in the bootm command
- System : kernel image used in the bootm command (uImage.bin for example)

When a RAM device is present in the FlashLayout, the programming service will not reboot but start the loaded kernel image with the associated device tree (when present, when it is absent the U-Boot device tree is used).

2.6 Field5: Device

Select the targeted device and the instance (starting at 0) as defined by U-Boot device tree:

- **mmc + instance** : mmc0, mmc1, mmc2

It is used for *e*•MMC or SD card on SDMMC. In the below examples:

  - SD card = mmc0 (SDMMC1)
  - *e*•MMC = mmc1 (SDMMC2)

- **nor + instance** : nor0

It is used for NOR on QUADSPI.

- **nand + instance** : nand0

It is used for parallel NAND Flash memories on FMC.

- **spi-nand + instance** : spi-nand0

It is used for serial NAND Flash memories on QSPI.

- none only used to load the programming service in RAM

It is allowed only for the reserved bootloaders partition (FSBL=0x1 and SSBL=0x3). In this case, the only allowed fields are: Type=Binary, Offset=0x0 and option=-

- ram + instance : ram0

It is used for files loaded in RAM by the programming service, for example to load and execute kernel.

Several devices can be mixed in the same FlashLayout file.

2.7 Field6: Offset

The supported values are:

- boot1: first boot area partition of *e*•MMC (offset is 0x0)
- boot2: second boot area partition of *e*•MMC (offset is 0x0)
- Offset in Bytes: offset in Flash memory area (in the user data area for *e*•MMC)

Refer to #Partition sizes for offset constraints.

2.8 Field7: Binary

This file is used by STM32CubeProgrammer to find the file associated to each Id when it is requested by embedded programming service .

The file can be absent (Binary=none in the tsv file) only for skipped partitions tagged with the E option. In all other cases, this file is sent to U-Boot to update the Flash memory only for the partitions selected with the P option.

2.9 GPT partuuid of rootfs partition

If #Field3: Name = rootfs for block device (SD card / *e*•MMC), U-Boot also sets a unique partition guid [4](PARTUUID) for each instance:

- mmc0: PARTUUID = "e91c4e10-16e6-4c0e-bd0e-77becf4a3582"
- mmc1: PARTUUID = "491f6117-415d-4f53-88c9-6e0de54deac6"
- mmc2: PARTUUID = "fd58f1c7-be0d-4338-8ee9-ad8f050aeb18"

This partition PARTUUID is distinct from the filesystem UUID and it is persistent.

Refer to GPT standard for details. [3]

This value can be used with the "root" argument in the kernel bootargs to identify the partition used for the "Root filesystem".

For instance, "root=PARTUUID=e91c4e10-16e6-4c0e-bd0e-77becf4a3582" [5] starts with the partition named rootfs on mmc0.

2.10 Partition operations

To update only one partition, use the same FlashLayout file, keep the #Field1: Options=P for the partition to update and change the others to -.

To update ssbl partition:

```
-        0x01        fsbl1        Binary                mmc0        0x00004400
fsbl.stm32
-        0x02        fsbl2        Binary                mmc0        0x00044400
fsbl.stm32
P        0x03        ssbl        Binary                mmc0        0x00084400        ssbl.
stm32
```

```
-        0x10        bootfs        System                mmc0        0x00284400
bootfs.ext4
-        0x11        rootfs        FileSystem        mmc0        0x08284400        rootfs.
ext4
-        0x12        userfs        FileSystem        mmc0        0x28284400        userfs.
ext4
```

To delete only one partition, add the DE option on the corresponding line.
To delete ssbl partition:

```
-        0x01        fsbl1        Binary                mmc0        0x00004400
fsbl.stm32
-        0x02        fsbl2        Binary                mmc0        0x00044400
fsbl.stm32
PDE         0x03          ssbl          Binary                 mmc0          0x00084400
ssbl.stm32
-        0x10        bootfs        System                mmc0        0x00284400
bootfs.ext4
-        0x11        rootfs        FileSystem        mmc0        0x08284400        rootfs.
ext4
-        0x12        userfs        FileSystem        mmc0        0x28284400        userfs.
ext4
```

2.11 Partition sizes

The partitions are contiguous (no holes in Flash memory).
The last partition continues until the end of the selected Flash memory.

To reduce the size of the last partition, use an 'Empty' partition and leave it unused.

All the partitions need to be present in the FlashLayout file, even if they are not selected or empty.
Then the offset and size of each partition are compared with:

- pre-existing GPT partitioning, for updates on block devices (*e*•MMC or SD card)
- predefined partitioning for MTD devices (NOR and NAND): see mtdparts environment variable in U-Boot for more information.

In case of partition size error, compare the existing partition size in U-Boot with the offset in the FlashLayout file.

2.11.1 GPT partition sizes

Each GPT partition must be aligned to:

- 512 bytes (LBA)
- *e*•MMC erase group size

The first partition starts after 17 Kbytes (default size of GPT header for 128 entries in U-Boot).

Prior to updating partitions in a block device, check the partition size by executing the U-Boot command "part list" on the selected device:

Board $> part list mmc 0

```
Partition Map for MMC device 0  --   Partition Type: EFI

Part        Start LBA        End LBA                Name
        Attributes
        Type GUID
        Partition GUID
  1        0x00000022        0x00000221        "fsbl1"
```

```
        attrs:         0x0000000000000000
        type:         ebd0a0a2-b9e5-4433-87c0-68b6b72699c7
        type:         data
        guid:         8ef917d1-2c6f-4bd0-a5b2-331a19f91cb2
  2        0x00000222         0x00000421         "fsbl2"
        attrs:         0x0000000000000000
        type:         ebd0a0a2-b9e5-4433-87c0-68b6b72699c7
        type:         data
        guid:         77877125-add0-4374-9e60-02cb591c9737
  3        0x00000422         0x00001821         "ssbl"
        attrs:         0x0000000000000000
        type:         ebd0a0a2-b9e5-4433-87c0-68b6b72699c7
        type:         data
        guid:         b4b84b8a-04e3-48ae-8536-aff5c9c495b1
  4        0x00001822         0x00021821         "bootfs"
        attrs:         0x0000000000000004
        type:         0fc63daf-8483-4772-8e79-3d69d8477de4
        type:         linux
        guid:         35219908-c613-4b08-9322-3391ff571e19
  5        0x00021822         0x00029821         "vendorfs"
        attrs:         0x0000000000000000
        type:         0fc63daf-8483-4772-8e79-3d69d8477de4
        type:         linux
        guid:         8e123a33-e3d3-4db9-92f4-d3ebd9b3224f
  6        0x00029822         0x001a9821         "rootfs"
        attrs:         0x0000000000000000
        type:         0fc63daf-8483-4772-8e79-3d69d8477de4
        type:         linux
        guid:         02a90af2-5d1c-4a29-9177-97a513e3cae4
  7        0x001a9822         0x003a9fdc         "userfs"
        attrs:         0x0000000000000000
        type:         0fc63daf-8483-4772-8e79-3d69d8477de4
        type:         linux
        guid:         3d5088db-a534-413e-836d-c757cb682814
```

Warning: Start and end are indicated in multiple of LBA (512 bytes by default).

To check the *e*•MMC erase group size in U-Boot, select the mmc device (here 1) and use the command "mmc info" in U-Boot.

Board $> mmc dev 1

```
switch to partitions #0, OK
mmc1(part 0) is current device
```

Board $> mmc info

```
Device: STM32 SDMMC2
Manufacturer ID: 11
OEM: 100
Name: 004G6
Bus Speed: 52000000
Mode : MMC High Speed (52MHz)
Rd Block Len: 512
MMC version 5.0
High Capacity: Yes
Capacity: 3.7 GiB
Bus Width: 8-bit
Erase Group Size: 512 KiB
HC WP Group Size: 4 MiB
User Capacity: 3.7 GiB WRREL
Boot Capacity: 2 MiB ENH
RPMB Capacity: 512 KiB ENH
```

2.11.2 MTD partition sizes

Each MTD partition must be aligned to the device erase block size (NOR/NAND Flash memory).

In U-Boot command :

Board $> nand info

```
Device 0: nand0, sector size 256 KiB
  Page size       4096 b
  OOB size         224 b
  Erase size    262144 b
  subpagesize     4096 b
  options     0x00084200
  bbt options 0x00060000
```

Board $> sf probe

```
SF: Detected mx25l51235f with page size 256 Bytes, erase size 64 KiB, total 64 MiB
```

For MTD, the U-Boot uses the mtdparts variable. Execute the U-Boot command mtdparts to know the current value:

Board $> mtdparts

By default on ST board, the mtdpart variable is built dynamically in board_mtdparts_default() under CONFIG_SYS_MTDPARTS_RUNTIME with the ST configs:

- CONFIG_MTDPARTS_NAND0_BOOT = "2m(fsbl),2m(ssbl1),2m(ssbl2)"
- CONFIG_MTDPARTS_NAND0_TEE = "512k(teeh),512k(teed),512k(teex)"
- CONFIG_MTDPARTS_NOR0_BOOT = "256k(fsbl1),256k(fsbl2),2m(ssbl),512k(u-boot-env)"
- CONFIG_MTDPARTS_NOR0_TEE = "256k(teeh),512k(teed),256k(teex)"
- CONFIG_MTDPARTS_SPINAND0_BOOT = 2m(fsbl),2m(ssbl1),2m(ssbl2)"
- CONFIG_MTDPARTS_SPINAND0_TEE = "512k(teeh),512k(teed),512k(teex)"

On NAND Flash, the last partition 'UBI' uses the remaining space.

On NOR Flash, the last partition named 'nor_user', is a free MTD partition which uses the remaining space.

We align each partition size on max supported erase block size (512 Kbytes on NAND and 256 Kbytes on NOR).

To change the MTP partitioning on NOR and NAND Flash memories, update these configurations in your U-Boot defconfig as explained in U-Boot or override this behavior in your board.

3 Typical FlashLayout file

This chapter describes the Layout file for the typical boot use cases based on STM32MP15_Flash_mapping when TEE is not used and associated partitions are absent (teeh, teed and teex).

Data are presented in tables for better readability despite the Layout file is plain text.

In the next examples, the STM32MP15 boot chain is used together with the following files:

- FSBL = TF-A, with 2 different binary as by default TF-A as no STM32CubeProgrammer support
  - FSBL to boot = tf-a-serialboot.stm32 (TF-A with STM32CubeProgrammer support)
  - FSBL to program = tf-a.stm32 (TF-A without OP-TEE support)
- SSBL = u-boot.stm32

3.1 NOR Flash memory and SD card

NOR Flash memory in RAW, containing the bootloaders. It uses two copies for FSBL (for failsafe update) and one copy for SSBL.
SD card using GPT: several user EXT4 partitions.

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl1	Binary	nor0	0x0	tf-a.stm32
P	0x04	fsbl2	Binary	nor0	0x40000	tf-a.stm32
P	0x03	ssbl	Binary	nor0	0x80000	u-boot.stm32
PED	0x20	env	Binary	nor0	0x280000	none
PE	0x21	unused	Binary	nor0	0x300000	none
P	0x10	bootfs	System	mmc0	0x00004400	bootfs.ext4
P	0x11	vendorfs	FileSystem	mmc0	0x04284400	vendorfs.ext4
P	0x12	rootfs	FileSystem	mmc0	0x05284400	rootfs.ext4
P	0x13	userfs	FileSystem	mmc0	0x35284400	userfs.ext4

The PartId 0x20 is empty/deleted in nor0, the U-Boot environment is cleared.

The PartId 0x21 is an empty/free user partition associated to 'nor_user' MTD partition in U-Boot

The 'System' partition named "bootfs" is marked 'bootable'.

The partition named "rootfs" has a specific PARTUUID.

3.2 NAND Flash memory

BootLoader using RAW and then U-Boot environment and file system in UBI volume.

FSBL (TF-A) uses two RAW copies to avoid NAND Flash memory disturbances inside one partition.

SSBL (U-Boot) uses two copies in two RAW partitions to avoid NAND Flash memory disturbances inside one partition.

One MTD partition for UBI with several volumes: uboot_config, uboot_config_r, rootfs, vendorfs, and userfs.

For parallel NAND, the MTD device is nand0:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl	Binary(2)	nand0	0x0	tf-a.stm32
P	0x03	ssbl1	Binary	nand0	0x200000	u-boot.stm32
P	0x04	ssbl2	Binary	nand0	0x400000	u-boot.stm32
P	0x10	UBI	FileSystem	nand0	0x600000	ubi.bin

For serial NAND, the MTD device is spi-nand0:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl	Binary(2)	spi-nand0	0x0	tf-a.stm32
P	0x03	ssbl1	Binary	spi-nand0	0x200000	u-boot.stm32
P	0x04	ssbl2	Binary	spi-nand0	0x400000	u-boot.stm32
P	0x10	UBI	FileSystem	spi-nand0	0x600000	ubi.bin

3.3 *e*•MMC

TF-A is copied in the two boot area partitions of *e*•MMC (hidden partition).

The GPT partitioning is used on the user area. U-Boot starts just after the GPT header at 17-Kbyte offset.

The other partitions are pre-populated with ext4 partition.

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl1	Binary	mmc1	boot1	tf-a.stm32
P	0x04	fsbl2	Binary	mmc1	boot2	tf-a.stm32
PD	0x03	ssbl	Binary	mmc1	0x00080000	u-boot.stm32
P	0x10	bootfs	System	mmc1	0x00280000	bootfs.ext4
P	0x11	vendorfs	FileSytem	mmc1	0x04280000	vendorfs.ext4
P	0x12	rootfs	FileSytem	mmc1	0x05280000	rootfs.ext4
P	0x13	userfs	FileSytem	mmc1	0x35280000	userfs.ext4

The partition 0x3 U-Boot is erased before update with option D to clean the U-Boot environment located at the end of this partition.

The 'System' partition named "bootfs" is marked 'bootable'.

The partition named "rootfs" has a specific PARTUUID.

3.4 SD card

RAW partition: two TF-A partitions, then U-Boot. The GPT partitioning is used so fsbl1 starts just after the GPT header at 17-Kbyte offset.

The other partitions are pre-populated with ext4 partition.

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl1	Binary	mmc0	0x4400	tf-a.stm32
P	0x04	fsbl2	Binary	mmc0	0x00044400	tf-a.stm32
PD	0x03	ssbl	Binary	mmc0	0x00084400	u-boot.stm32
P	0x10	bootfs	System	mmc0	0x00284400	bootfs.ext4
P	0x11	vendorfs	FileSytem	mmc0	0x04284400	vendorfs.ext4
P	0x12	rootfs	FileSytem	mmc0	0x05284400	rootfs.ext4
P	0x13	userfs	FileSytem	mmc0	0x35284400	userfs.ext4

The partition 0x3 U-Boot is erased before update with option D to clean the U-Boot environment located at the end of this partition.

The 'System' partition named "bootfs" is marked 'bootable'.

The partition named "rootfs" has a specific PARTUUID.

4 Typical FlashLayout file with TEE

This chapter describes the Layout file with TEE partitions for the trusted typical boot use case based on STM32MP15_Flash_mapping.

Since TF-A binary for OPTEE does not support STM32CubeProgrammer, two different FSBL binaries are used:

- FSBL to boot = tf-a-serialboot.stm32 (TF-A with STM32CubeProgrammer support)
- FSBL to program= tf-a-optee.stm32(TF-A with OP-TEE support)

4.1 NOR Flash memory and SD card with TEE

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl1	Binary	nor0	0x0	tf-a-optee.stm32
P	0x04	fsbl2	Binary	nor0	0x40000	tf-a-optee.stm32
P	0x03	ssbl	Binary	nor0	0x80000	u-boot.stm32
PED	0x20	env	Binary	nor0	0x280000	none
P	0x05	teeh	Binary	nor0	0x00300000	tee-header.stm32
P	0x06	teed	Binary	nor0	0x00340000	tee-pageable.stm32
P	0x07	teex	Binary	nor0	0x00380000	tee-pager.stm32
PE	0x21	unused	Binary	nor0	0x400000	none
P	0x10	bootfs	System	mmc0	0x00004400	bootfs.ext4
P	0x11	vendorfs	FileSystem	mmc0	0x04284400	vendorfs.ext4
P	0x12	rootfs	FileSystem	mmc0	0x05284400	rootfs.ext4
P	0x13	userfs	FileSystem	mmc0	0x35284400	userfs.ext4

4.2 NAND Flash memory with TEE

WARNING: the 'tee' partitions are not duplicated in the next example, possible reliability issue with NAND disturbance.

For parallel NAND, the MTD device is nand0:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl1	Binary(2)	nand0	0x0	tf-a-optee.stm32
P	0x03	ssbl1	Binary	nand0	0x200000	u-boot.stm32

Opt	Part	Name	Type	Device	Offset	Binary
P	0x04	ssbl2	Binary	nand0	0x400000	u-boot.stm32
P	0x05	teeh	Binary	nand0	0x600000	tee-header.stm32
P	0x06	teed	Binary	nand0	0x680000	tee-pageable.stm32
P	0x07	teex	Binary	nand0	0x700000	tee-pager.stm32
P	0x10	UBI	FileSystem	nand0	0x780000	ubi.bin

For serial NAND, the MTD device is spi-nand0:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl1	Binary(2)	spi-nand0	0x0	tf-a-optee.stm32
P	0x03	ssbl1	Binary	spi-nand0	0x200000	u-boot.stm32
P	0x04	ssbl2	Binary	spi-nand0	0x400000	u-boot.stm32
P	0x05	teeh	Binary	spi-nand0	0x600000	tee-header.stm32
P	0x06	teed	Binary	spi-nand0	0x680000	tee-pageable.stm32
P	0x07	teex	Binary	spi-nand0	0x700000	tee-pager.stm32
P	0x10	UBI	FileSystem	spi-nand0	0x780000	ubi.bin

4.3 e•MMC with TEE

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl1	Binary	mmc1	boot1	tf-a-optee.stm32
P	0x04	fsbl2	Binary	mmc1	boot2	tf-a-optee.stm32
PD	0x03	ssbl	Binary	mmc1	0x00080000	u-boot.stm32
P	0x05	teeh	Binary	mmc1	0x00280000	tee-header.stm32
P	0x06	teed	Binary	mmc1	0x00300000	tee-pageable.stm32
P	0x07	teex	Binary	mmc1	0x00380000	tee-pager.stm32
P	0x10	bootfs	System	mmc1	0x00400000	bootfs.ext4.stm32
P	0x11	vendorfs	FileSytem	mmc1	0x04400000	vendorfs.ext4
P	0x12	rootfs	FileSytem	mmc1	0x05400000	rootfs.ext4

Opt	Part	Name	Type	Device	Offset	Binary
P	0x13	userfs	FileSytem	mmc1	0x35400000	userfs.ext4

4.4 SD card with TEE

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl1	Binary	mmc0	0x4400	tf-a-optee.stm32
P	0x04	fsbl2	Binary	mmc0	0x00044400	tf-a-optee.stm32
PD	0x03	ssbl	Binary	mmc0	0x00084400	u-boot.stm32
P	0x05	teeh	Binary	mmc0	0x00284400	tee-header.stm32
P	0x06	teed	Binary	mmc0	0x002C4400	tee-pageable.stm32
P	0x07	teex	Binary	mmc0	0x00344400	tee-pager.stm32
P	0x10	bootfs	System	mmc0	0x00384400	bootfs.ext4
P	0x11	vendorfs	FileSytem	mmc0	0x04384400	vendorfs.ext4
P	0x12	rootfs	FileSytem	mmc0	0x05384400	rootfs.ext4
P	0x13	userfs	FileSytem	mmc0	0x35304400	userfs.ext4

5 FlashLayout file to load and start kernel

Load programming service with Device=none.

Load kernel uImage and device tree in DDR, Device=ram0.

This Linux kernel is started by a bootm command after the STM32CubeProgrammer detach command.

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
-	0x03	ssbl	Binary	none	0x0	u-boot.stm32
P	0x10	kernel	System	ram0	0xC2000000	uImage.bin
P	0x11	dtb	FileSytem	ram0	0xC4000000	stm32mp157f-ev1.dtb

6 Using provided FlashLayout files

The binary and the associated pre-defined FlashLayout files are provided by ST in the Discovery kit.

For example, in STM32MP15 Discovery kits, you can find the distribution images as well as the file FlashLayout_sdcard_stm32mp157f-ev1-trusted.tsv:

```
#Opt        Id        Name        Type        IP        Offset        Binary
-        0x01        fsbl1-boot        Binary        none        0x0        arm-trusted-
firmware/tf-a-stm32mp157f-ev1-serialboot.stm32
-        0x03        ssbl-boot        Binary        none        0x0        bootloader/u-
boot-stm32mp157f-ev1-trusted.stm32
P        0x04        fsbl1        Binary        mmc0        0x00004400        arm-trusted-
firmware/tf-a-stm32mp157f-ev1-trusted.stm32
P        0x05        fsbl2        Binary        mmc0        0x00044400        arm-trusted-
firmware/tf-a-stm32mp157f-ev1-trusted.stm32
PD        0x06        ssbl        Binary        mmc0        0x00084400        bootloader
/u-boot-stm32mp157f-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        0x33C84400        st-
image-userfs-openstlinux-weston-stm32mp1.ext4
```

You can use these FlashLayout files as a starting point and simply modify them:

- #Updating partitions
- #Updating partitions using official ST bootloaders

6.1 Updating partitions

To update only some partitions, change the FlashLayout and only "select" the partitions that need to be updated:
Options field inside the FlashLayout is kept to **P** for partition(s) that need to be updated, others are changed to **-**.
Then execute STM32CubeProgrammer as before.

Example to update only U-Boot binary and st-image-bootfs filesystem :

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl1-boot	Binary	none	0x0	arm-trusted-firmware/tf-a-stm32mp157f-ev1-serialboot.stm32
-	0x03	ssbl-boot	Binary	none	0x0	bootloader/u-boot-stm32mp157f-ev1-trusted.stm32
-	0x04	fsbl1	Binary	mmc0	0x00004400	arm-trusted-firmware/tf-a-stm32mp15f-ev1-trusted.stm32
-	0x05	fsbl2	Binary	mmc0	0x00044400	arm-trusted-firmware/tf-a-stm32mp15f-ev1-trusted.stm32

Opt	Part	Name	Type	Device	Offset	Binary
PD	0x06	ssbl	Binary	mmc0	0x00084400	bootloader/ u-boot-stm32mp15f-ev1-trusted.stm32
P	0x10	bootfs	System	mmc0	0x00284400	st-image-bootfs-openstlinux-weston-stm32mp1.ext4
-	0x11	vendorfs	FileSystem	mmc0	0x04284400	st-image-vendorfs-openstlinux-weston-stm32mp1.ext4
-	0x12	rootfs	FileSystem	mmc0	0x05284400	st-image-core-openstlinux-weston-stm32mp1.ext4
-	0x13	userfs	FileSystem	mmc0	0x33C84400	st-image-userfs-openstlinux-weston-stm32mp1.ext4

The associated text file becomes :

```
#Opt        Id        Name        Type        IP        Offset        Binary
-        0x01        fsbl1-boot        Binary        none        0x0        arm-trusted-
firmware/tf-a-stm32mp157f-ev1-serialboot.stm32
-        0x03        ssbl-boot        Binary        none        0x0        bootloader/u-
boot-stm32mp157f-ev1-trusted.stm32
-        0x04        fsbl1        Binary        mmc0        0x00004400        arm-trusted-
firmware/tf-a-stm32mp157f-ev1-trusted.stm32
-        0x05        fsbl2        Binary        mmc0        0x00044400        arm-trusted-
firmware/tf-a-stm32mp157f-ev1-trusted.stm32
PD        0x06        ssbl        Binary        mmc0        0x00084400        bootloader
/u-boot-stm32mp157f-ev1-trusted.stm32
P        0x21        bootfs        System        mmc0        0x00284400        st-image-
bootfs-openstlinux-weston-stm32mp1.ext4
-        0x22        vendorfs        FileSystem        mmc0        0x04284400        st-
image-vendorfs-openstlinux-weston-stm32mp1.ext4
-        0x23        rootfs        FileSystem        mmc0        0x05284400        st-
image-weston-openstlinux-weston-stm32mp1.ext4
-        0x24        userfs        FileSystem        mmc0        0x33C84400        st-
image-userfs-openstlinux-weston-stm32mp1.ext4
```

6.2 Updating partitions using official ST bootloaders

If bootloader, FSBL or SSBL are modified, and the STM32CubeProgrammer support is lost for any reason (for example if the stm32prog command is removed), you can still program these new files by selecting the correct binary setting for the **partitions 0x01 and 0x03** with Device=**none** and change the Id for the binaries to program in Flash, as indicated in chapter "Field2: Id".

For example, with ST board, you can flash Customer-modified binary by using the ST original file. The new Layout file is:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl1-boot	Binary	none	0x0	arm-trusted-firmware/tf-a-stm32mp157f-ev1-serialboot.stm32
-	0x03	ssbl-boot	Binary	none	0x0	bootloader/u-boot-stm32mp157f-ev1-trusted.stm32

Opt	Part	Name	Type	Device	Offset	Binary
P	0x04	fsbl1	Binary	mmc0	0x00004400	**<Customer-tf-a>.stm32**
P	0x05	fsbl2	Binary	mmc0	0x00044400	**<Customer-tf-a>.stm32**
PD	0x06	ssbl	Binary	mmc0	0x00084400	**<Customer-u-boot>.stm32**
P	0x10	bootfs	System	mmc0	0x00284400	st-image-bootfs-openstlinux-weston-stm32mp1.ext4
P	0x11	vendorfs	FileSystem	mmc0	0x04284400	st-image-vendorfs-openstlinux-weston-stm32mp1.ext4
P	0x12	rootfs	FileSystem	mmc0	0x05284400	st-image-core-openstlinux-weston-stm32mp1.ext4
P	0x13	userfs	FileSystem	mmc0	0x33C84400	st-image-userfs-openstlinux-weston-stm32mp1.ext4

The associated text file becomes :

```
#Opt        Id         Name        Type        IP        Offset        Binary
-        0x01        fsbl1-boot        Binary        none        0x0        arm-trusted-
firmware/tf-a-stm32mp157f-ev1-serialboot.stm32
-        0x03        ssbl-boot        Binary        none        0x0        bootloader/u-
boot-stm32mp157f-ev1-trusted.stm32
P        0x04        fsbl1        Binary        mmc0        0x00004400        tf-a.stm32
P        0x05        fsbl2        Binary        mmc0        0x00044400        tf-a.stm32
PD        0x06        ssbl        Binary        mmc0        0x00084400        u-boot.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        0x33C84400        st-
image-userfs-openstlinux-weston-stm32mp1.ext4
```

7 Other FlashLayout examples

7.1 NOR and NAND Flash memories

NOR Flash memory in RAW: TF-A uses several partitions for failsafe update, then U-Boot.
NAND Flash memory in UBI: only one large MTD partition for UBI volumes and UBIFS.

Opt	Part	Name	Type	Device	Offset	Binary
P	0x01	fsbl1	Binary	nor0	0x0	tf-a.stm32
P	0x02	fsbl2	Binary	nor0	0x40000	tf-a.stm32
P	0x03	ssbl	Binary	nor0	0x80000	u-boot.stm32
PED	0x20	env	Binary	nor0	0x280000	none
PE	0x21	unused	Binary	nor0	0x300000	none
P	0x10	UBI	FileSystem	nand0	0x0	ubi.bin

The PartId 0x21 is an empty/free user partition associated to 'nor_user' MTD partition in U-Boot.

Or with TEE support:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a.stm32
P	0x02	fsbl1	Binary	nor0	0x0	tf-a-optee.stm32
P	0x04	fsbl2	Binary	nor0	0x40000	tf-a-optee.stm32
P	0x03	ssbl	Binary	nor0	0x80000	u-boot.stm32
PE	0x20	env	Binary	nor0	0x280000	none
P	0x05	teeh	Binary	nor0	0x00300000	tee-header.stm32
P	0x06	teed	Binary	nor0	0x00340000	tee-pageable.stm32
P	0x07	teex	Binary	nor0	0x003C0000	tee-pager.stm32
PE	0x21	unused	Binary	nor0	0x400000	none
P	0x10	UBI	FileSystem	nand0	0x0	ubi.bin

7.2 NAND with SSBL in UBI

FSBL (TF-A) uses two partitions (for failsafe update), with two copies in each partition .

SSBL (U-Boot) is present in a RAW volume of the UBI partition.
The binary is also present in FlashLayout file to be loaded in RAM (Opt=-, Part=0x3, Device=none).

One MTD partition for UBI with several volumes: u-boot, uboot_config, uboot_config_r, rootfs, vendorfs, and userfs.

As it is not the ST default configuration, the MTD partitioning must be changed in U-Boot.

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a-serialboot.stm32
P	0x02	fsbl	Binary(2)	nand0	0x0	tf-a.stm32
P	0x04	fsbl	Binary(2)	nand0	0x00100000	tf-a.stm32
-	0x03	ssbl	Binary	none	0x0	u-boot.stm32
P	0x10	UBI	FileSystem	nand0	0x00200000	ubi.bin

For OPT-TEE support, program tf-a-optee.stm32 with OP-TEE support and add the needed optee UBI volumes: teeh, teed, teex.

Warning: UBI is not supported in TFA

7.3 SD card: FAT

Below an example with FAT bootfs partition (kernel and RAMFS) and an empty userfs partition (deleted and formatted by Linux on first boot):

Opt	Part	Name	Type	Device	Offset	Binary
P	0x01	fsbl1	Binary	mmc0	0x0	tf-a.stm32
P	0x02	fsbl2	Binary	mmc0	0x40000	tf-a.stm32
P	0x03	ssbl	Binary	mmc0	0x80000	u-boot.stm32
P	0x10	bootfs	System	mmc0	0x200000	bootfs.vfat
PED	0x11	userfs	Empty	mmc0	0x400000	none

7.4 Load and program different binaries

Same example as above with the first two partitions (named "2boot"). They are not programmed and only loaded in RAM.

Userfs is empty but not deleted.

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl2boot	Binary	none	0x0	tf-a2boot.stm32
-	0x03	ssbl2boot	Binary	none	0x0	u-boot2boot.stm32
P	0x02	fsbl1	Binary	mmc0	0x40000	tf-a.stm32
P	0x04	fsbl2	Binary	mmc0	0x0	tf-a.stm32
P	0x05	ssbl	Binary	mmc0	0x80000	u-boot.stm32
P	0x10	rootfs	System	mmc0	0x200000	rootfs.vfat

Opt	Part	Name	Type	Device	Offset	Binary
PE	0x11	userfs	FileSytem	mmc0	0x400000	none

7.5 RawImage

The SD card content is exported as RAW device and updated with the image.sdcard file:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a.stm32
-	0x03	ssbl	Binary	none	0x0	u-boot.stm32
P	0x10	sdcard	RawImage	mmc0	0x0	image.sdcard

You can also erase the device before performing the update, by adding D in option.

7.6 Deleting device content

For example, NOR and NAND Flash memories are deleted with:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	none	none	0x0	tf-a.stm32
-	0x03	ssbl	none	none	0x0	u-boot.stm32
PDE	0x10	nor	RawImage	nor0	0x0	none
PDE	0x11	nand	RawImage	nand0	0x0	none

To erase all other devices, including the *e*•MMC boot partition, proceed as follows:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl1	Binary	none	0x0	tf-a.stm32
-	0x03	ssbl	Binary	none	0x0	u-boot.stm32
PDE	0x10	sdcard	RawImage	mmc0	0x0	none
PED	0x20	emmc_boot1	Binary	mmc1	boot1	none
DPE	0x21	emmc_boot2	Binary	mmc1	boot2	none
EPD	0x22	emmc	RawImage	mmc1	0x0	none
DPE	0x30	nand	RawImage	nand0	0x0	none
PDE	0x40	nor	RawImage	nor0	0x0	none

Warning: A timeout may occur in STM32CubeProgrammer since deleting NOR Flash memory might be slow. To avoid this issue you can delete only the used partitions (Option=PED), for example:

```
#Opt        Id        Name               Type               Device
Offset                 Binary
-        0x01        fsbl_boot        Binary               none
0x0                fsbl.stm32
-        0x03        ssbl_boot        Binary               none
0x0                ssbl.stm32
#delete ALL devices
EPD        0x10        sdcard               RawImage        mmc0
0x0                none
PED        0x02        emmc_b1               Binary               mmc1
boot1                none
PED        0x04        emmc_b2               Binary               mmc1
boot2                none
PED        0x20        emmc                RawImage        mmc1        0x0
none
PED        0x30        nand                RawImage        nand0
0x0                none
# on NOR (slow device): delete ALL used partitions
PE        0x40        nor                 RawImage        nor0        0x0
none
PED        0x41        fsbl1_nor        Binary               nor0
0x00000000        none
PED        0x42        fsbl2_nor        Binary               nor0
0x00040000        none
PED        0x43        ssbl_nor        Binary               nor0
0x00080000        none
PED        0x44        env_nor        Binary               nor0        0x00280000
none
PE        0x45        unused               Binary               nor0
0x002C0000        none
```

7.7 Complex use case

Update SD card (mmc0 on SDMMC1). The GPT partition is created since all partitions are selected and Userfs is deleted and empty.

Erase all other devices, including *e*•MMC hidden boot partition.

Opt	Part	Name	Type	Device	Offset	Binary
P	0x01	fsbl1	Binary	mmc0	0x04400	tf-a.stm32
P	0x02	fsbl2	Binary	mmc0	0x44400	tf-a.stm32
P	0x03	ssbl	Binary	mmc0	0x84400	u-boot.stm32
P	0x10	bootfs	System	mmc0	0x284400	bootfs.stm32
P	0x11	rootfs	FileSytem	mmc0	0x08284400	rootfs.ext4
DEP	0x12	userfs	FileSytem	mmc0	0x28284400	none
PED	0x60	emmc_boot1	Binary	mmc1	boot1	none
DPE	0x61	emmc_boot2	Binary	mmc1	boot2	none
EPD	0x62	emmc	RawImage	mmc1	0x0	none
D	0x10	nand	RawImage	nand0	0x0	none

8 Reference list

- https://en.wikipedia.org/wiki/USB#Device_Firmware_Upgrade
- USB Device Class Specification for Device Firmware Upgrade, Version 1.1, (available at http://www.usb.org/developers/docs): https://www.usb.org/sites/default/files/DFU_1.1.pdf
- 3.03.1 https://en.wikipedia.org/wiki/GUID_Partition_Table
- 4.04.1 https://en.wikipedia.org/wiki/Universally_unique_identifier
- https://wiki.archlinux.org/index.php/persistent_block_device_naming

Flash memories combine high density and cost effectiveness of EPROMs with the electrical erasability of EEPROMs. For this reason, the Flash memory market is one of the most exciting areas of the semiconductor industry today and new applications requiring in system reprogramming, such as cellular telephones, automotive engine management systems, hard disk drives, PC BIOS software for Plug & Play, digital TV, set top boxes, fax and other modems, PC cards and multimedia CD-ROMs, offer the prospect of very high volume demand.

Universal Asynchronous Receiver/Transmitter

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

Trusted Execution Environment

Microprocessor Unit

First Stage Boot Loader

Second Stage Boot Loader

SD memory card (https://www.sdcard.org)

MultimediaCard

GUID Partition Table

Embedded Display Port (VESA standard). See http://www.displayport.org/ for more details

Read Only Memory

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today forvolatile random-accesssemiconductor memories.)

Trusted Firmware for Arm Cortex-A

One Time Programmed

Secure Secret Provisioning

Secure secrets provisioning - NEW

Power Management Integrated Circuit

Non Volatile Memory, like a flash memory

Device Firmware Upgrade

Memory Technology Device

Elliptic curve cryptography

Error Correction Capability

Doubledata rate (memory domain)

universally unique identifier (https://en.wikipedia.org/wiki/Universally_unique_identifier)

Logical Block Addressing

Original Equipment Manufacturer - NEW

Flash memory shortened to gain space in titles, tables and block diagrams

Open Portable Trusted Execution Environment