



STM32CubeProgrammer flashlayout



STM32CubeProgrammer flashlayout

Stable: 28.04.2020 - 13:16 / Revision: 28.04.2020 - 13:15

Contents

1 Article purpose	3
2 FlashLayout file format	3
2.1 Examples	4
2.2 Field1: Options	4
2.3 Field2: Id	5
2.3.1 Reserved Id	5
2.3.2 Using Id for boot partition	5
2.4 Field3: Name	6
2.5 Field4: Type	6
2.5.1 Block device GPT partition: SD card / eMMC	7
2.5.2 Raw Flash device (NAND/NOR Flash memories) MTD partition	7
2.5.3 Hardware device	7
2.6 Field5: Device	8
2.7 Field6: Offset	8
2.8 Field7: Binary	8
2.9 GPT partuuid of rootfs partition	8
2.10 Partition operations	9
2.11 Partition sizes	9
2.11.1 GPT partition sizes	10
2.11.2 MTD Partition sizes	11
3 Typical FlashLayout file	12
3.1 Basic boot chain support	12
3.2 NOR Flash memory and SD card	12
3.3 NOR and NAND Flash memories	13
3.4 NAND Flash memory	13
3.4.1 FSBL:RAW & SSBL:UBI	13
3.4.2 FSBL+SSBL:RAW	14
3.5 eMMC	14
3.6 SD card	15
4 Typical FlashLayout file with TEE	15
4.1 NOR Flash memory and SD card with TEE	15
4.2 NOR and NAND Flash memories with TEE	16
4.3 NAND Flash memory with TEE	16
4.3.1 FSBL:RAW & SSBL:UBI with TEE	16
4.3.2 FSBL+SSBL:RAW with TEE	17
4.4 eMMC with TEE	17
4.5 SD card with TEE	18
5 Using provided FlashLayout files	18
5.1 Updating partitions	19
5.2 Updating partitions using official ST bootloaders	20



6 Other FlashLayout examples	21
6.1 SD card: FAT	21
6.2 Load and program different binaries	21
6.3 RAWImage	22
6.4 Deleting device content	22
6.5 Complex use case	23
7 Reference list	24

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 USB boot. However you can launch it manually from the U-boot console.

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 file.

The next chapter:

- describes the #FlashLayout file format and
- gives #Typical FlashLayout file for boot scenario without TEE and #Typical FlashLayout file with TEE

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.bin
P        0x11    rootfs    FileSystem mmc0      0x08284400
rootfs.ext4.bin
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 '**L**' or '**P**':

- '**L**': **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 block devices (SD card or eMMC), 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):

- '**L**': **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')

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)
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

In normal use cases, the same FSBL and SSBL binaries are loaded in RAM and programmed in Flash memory. 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 programed 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:

Code	Partition
0x01 (reserved)	FSBL to boot : load 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

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][3]}).

This is also the name of the Block device GPT partition: SD card / eMMC.

The requirements for the GPT partition names are:

- FSBL for SD card boot: the name must start with 'fsbl'= fsbl, fsbl1, fsbl2... (ROM code requirement)
- SSBL for eMMC/SD card boot: the name must be 'ssbl' (TF-A requirement)

These two 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 / eMMC
 - #Raw Flash device (NAND/NOR Flash memories) MTD partition : see MTD overview
- all the #Hardware device = RAWImage

The supported values are:



Type	GPT		MTD	
	SD card	eMMC	NAND Flash memory	NOR Flash memory
Binary	x	x	x	x
Binary(N)			ssbl	
FileSystem	x	x	x	x
System	x	x	UBI	UBI
RAWImage	x	user data	x	x

2.5.1 Block device GPT partition: SD card / eMMC

Refer to GPT standard for details ^[4].

The supported values, with associated [partition type GUID](#) (globally unique identifiers^[5]), 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 eMMC: the user data area of eMMC (see #Field6: Offset for access to the boot area partitions)

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

2.6 Field5: Device

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

- **mmc + instance** : 'mmc0'
It is used for eMMC or SD card on SDMMC. In the below examples:
 - SD card = mmc0 (SDMMC1)
 - eMMC = 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.
- **none** for partition only used to load the binary in RAM
It is allowed only for the reserved boot partition (FSBL=0x1 and SSBL=0x3). In this case, the only allowed fields are :
Type=Binary, Offset=0x0 and option='-'.
Several devices can be mixed in the same FlashLayout file.

2.7 Field6: Offset

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)

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 / eMMC), U-Boot also sets a unique partition guid ^[5](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. ^[4]



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=PARTUID=e91c4e10-16e6-4c0e-bd0e-77becf4a3582" ^[6] 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:

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.bin
-          0x11      rootfs     FileSystem  mmc0      0x08284400
rootfs.ext4.bin
-          0x12      userfs     FileSystem  mmc0      0x28284400
userfs.ext4.bin
```

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.bin
-          0x11      rootfs     FileSystem  mmc0      0x08284400
rootfs.ext4.bin
-          0x12      userfs     FileSystem  mmc0      0x28284400
userfs.ext4.bin
```

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 (eMMC 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)
- eMMC 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     0x00000821   "ssbl"
  attrs:      0x0000000000000000
  type:      ebd0a0a2-b9e5-4433-87c0-68b6b72699c7
  type:      data
  guid:      b4b84b8a-04e3-48ae-8536-aff5c9c495b1
4         0x00000822     0x000021821  "bootfs"
  attrs:      0x0000000000000004
  type:      0fc63daf-8483-4772-8e79-3d69d8477de4
  type:      linux
  guid:      35219908-c613-4b08-9322-3391ff571e19
5         0x000021822   0x000029821  "vendorfs"
  attrs:      0x0000000000000000
  type:      0fc63daf-8483-4772-8e79-3d69d8477de4
  type:      linux
  guid:      8e123a33-e3d3-4db9-92f4-d3ebd9b3224f
6         0x000029822   0x0001a9821  "rootfs"
  attrs:      0x0000000000000000
  type:      0fc63daf-8483-4772-8e79-3d69d8477de4
  type:      linux
  guid:      02a90af2-5d1c-4a29-9177-97a513e3cae4
7         0x0001a9822   0x0003a9fdc  "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 eMMC 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 mtdparts variable is built dynamically (in board_mtdparts_default() under CONFIG_SYS_MTDPARTS_RUNTIME). Execute the U-Boot command mtparts to know the current value, for example when NOR and NAND Flash memories are supported:

Board \$> mtdparts

```
SF: Detected mx25l51235f with page size 256 Bytes, erase size 64 KiB, total 64 MiB  
device nor0 <nor0>, # parts = 5  
#: name      size      offset      mask_flags  
0: fsbl1     0x00040000 0x00000000 0  
1: fsbl2     0x00040000 0x00040000 0  
2: ssbl      0x00200000 0x00080000 0  
3: u-boot-env 0x00040000 0x00280000 0  
4: nor_user  0x03d40000 0x002c0000 0
```

```

device nand0 <nand0>, # parts = 4
#: name      size      offset      mask_flags
0: fsbl      0x00200000  0x00000000  0
1: ssbl1     0x00200000  0x00200000  0
2: ssbl2     0x00200000  0x00400000  0
3: UBI       0x3fa00000  0x00600000  0

active partition: nor0,0 - (fsbl1) 0x00040000 @ 0x00000000

defaults:
mtdids  : nor0=nor0,nand0=nand0
mtdparts: mtdparts=nor0:256k(fsbl1),256k(fsbl2),2m(ssbl),256k(u-boot-env),-(nor_user);
nand0:2m(fsbl),2m(ssbl1),2m(ssbl2),-(UBI)
  
```

NOR Flash last partition, named 'nor_user', is a free MTD partition.

To change the MTD partitioning on NOR and NAND Flash memories, update this variable in U-Boot.

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 example, the trusted boot chain is used together with the following files:

- FSBL = tf-a.stm32
- SSBL = u-boot.stm32

3.1 Basic boot chain support

The layout in NVM is the same for the basic boot chain (see [STM32MP15_Flash_mapping](#)).



The basic boot chain cannot be used for product development (see [Boot chains overview for details](#)).

The files that must be programmed in NVM for the basic boot chain are:

- FSBL = u-boot-spl.stm32 (instead of tf-a.stm32)
- SSBL = u-boot.img (instead of u-boot.stm32).

However, the basic boot chain does not support the STM32CubeProgrammer (use the trusted bootloaders to program the basic boot chain bootloaders in Flash memory) and the SSBL is programmed without STM32 header (with Phaseld 0x10).

3.2 NOR Flash memory and SD card

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



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
PE	0x20	u-boot-env	Binary	nor0	0x280000	none
PE	0x21	unused	Binary	nor0	0x2C0000	none
P	0x10	bootfs	System	mmc0	0x00004400	bootfs.ext4.bin
P	0x11	vendorfs	FileSystem	mmc0	0x04284400	vendorfs.ext4. bin
P	0x12	rootfs	FileSystem	mmc0	0x05284400	rootfs.ext4.bin
P	0x13	userfs	FileSystem	mmc0	0x35284400	userfs.ext4.bin

The PartId 0x20 is empty/free in nor0, no logo is provided.

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

3.3 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
PE	0x20	u-boot-env	Binary	nor0	0x280000	none
PE	0x21	unused	Binary	nor0	0x2C0000	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.

Warning: for compatibility with NAND-only use case and U-Boot configuration on ST board, the offset of UBI in nand0 is set to 0x00600000.

3.4 NAND Flash memory

BootLoader using RAW and then file system in UBI volume.

3.4.1 FSBL:RAW & SSBL:UBI

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

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

Warning: The MTD partitioning must be changed in U-Boot.

3.4.2 FSBL+SSBL:RAW

FSBL (TF-A) uses two copies in two partitions (for failsafe update) to avoid NAND Flash memory disturbances.

SSBL (U-Boot) uses two copies in two partitions (and not in a UBI volume).

One MTD partition for UBI with several volumes: boot, rootfs and userfs.

Opt	Part	Name	Type	Device	Offset	Binary
P	0x01	fsbl1	Binary(2)	nand0	0x0	tf-a.stm32
P	0x02	fsbl2	Binary(2)	nand0	0x100000	tf-a.stm32
P	0x03	ssbl1	Binary	nand0	0x200000	u-boot.bin
P	0x04	ssbl2	Binary	nand0	0x400000	u-boot.bin
P	0x10	UBI	FileSystem	nand0	0x600000	ubi.bin

3.5 eMMC

TF-A copied in the two boot area partitions of eMMC (hidden partition).

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

The userfs partition is pre-populated with ext4 partition.

Opt	Part	Name	Type	Device	Offset	Binary
P	0x01	fsbl1	Binary	mmc1	boot1	tf-a.stm32
P	0x02	fsbl2	Binary	mmc1	boot2	tf-a.stm32
P	0x03	ssbl	Binary	mmc1	0x00080000	u-boot.stm32
P	0x10	bootfs	System	mmc1	0x00280000	bootfs.ext4.stm32
						vendorfs.ext4.



Opt	Part	Name	Type	Device	Offset	Binary
P	0x11	vendorfs	FileSytem	mmc1	0x04280000	bin
P	0x12	rootfs	FileSytem	mmc1	0x05280000	rootfs.ext4.bin
P	0x13	userfs	FileSytem	mmc1	0x35280000	userfs.ext4.bin

3.6 SD card

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

Opt	Part	Name	Type	Device	Offset	Binary
P	0x01	fsbl1	Binary	mmc0	0x4400	tf-a.stm32
P	0x02	fsbl2	Binary	mmc0	0x00044400	tf-a.stm32
P	0x03	ssbl	Binary	mmc0	0x00084400	u-boot.stm32
P	0x10	bootfs	System	mmc0	0x00284400	bootfs.ext4.bin
P	0x11	vendorfs	FileSytem	mmc0	0x04284400	vendorfs.ext4. bin
P	0x12	rootfs	FileSytem	mmc0	0x05284400	rootfs.ext4.bin
P	0x13	userfs	FileSytem	mmc0	0x35284400	userfs.ext4.bin

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 yet support STM32CubeProgrammer, two different FSBL binaries are used:

- to boot = tf-a.stm32 (no TEE support but programmer support)
- to flash = tf-a-optee.stm32

Since the MTD partitioning is hardcoded in U-Boot, the optee variant is used : u-boot-optee.stm32.

4.1 NOR Flash memory and SD card with TEE

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
						u-boot-optee.



Opt	Part	Name	Type	Device	Offset	Binary
P	0x03	ssbl	Binary	nor0	0x80000	stm32
PE	0x20	u-boot-env	Binary	nor0	0x280000	none
P	0x05	teeh	Binary	nor0	0x002C0000	tee-header.stm32
P	0x06	teed	Binary	nor0	0x00300000	tee-pageable.stm32
P	0x07	teex	Binary	nor0	0x00340000	tee-pager.stm32
PE	0x21	unused	Binary	nor0	0x380000	none
P	0x10	bootfs	System	mmc0	0x00004400	bootfs.ext4.bin
P	0x11	vendorfs	FileSystem	mmc0	0x04284400	vendorfs.ext4.bin
P	0x12	rootfs	FileSystem	mmc0	0x05284400	rootfs.ext4.bin
P	0x13	userfs	FileSystem	mmc0	0x35284400	userfs.ext4.bin

4.2 NOR and NAND Flash memories with TEE

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-optee.stm32
PE	0x20	u-boot-env	Binary	nor0	0x280000	none
P	0x05	teeh	Binary	nor0	0x002C0000	tee-header.stm32
P	0x06	teed	Binary	nor0	0x00300000	tee-pageable.stm32
P	0x07	teex	Binary	nor0	0x00340000	tee-pager.stm32
PE	0x21	unused	Binary	nor0	0x380000	none
P	0x10	UBI	FileSystem	nand0	0x0	ubi.bin

4.3 NAND Flash memory with TEE

4.3.1 FSBL:RAW & SSBL:UBI with TEE

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a.stm32



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

4.3.2 FSBL+SSBL:RAW with TEE

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

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a.stm32
P	0x05	fsbl1	Binary(2)	nand0	0x0	tf-a-optee.stm32
P	0x04	fsbl2	Binary(2)	nand0	0x100000	tf-a-optee..stm32
P	0x03	ssbl1	Binary	nand0	0x200000	u-boot-optee.bin
P	0x02	ssbl2	Binary	nand0	0x400000	u-boot-optee.bin
P	0x06	teeh	Binary	nand0	0x600000	tee-header.stm32
P	0x07	teed	Binary	nand0	0x680000	tee-pageable.stm32
P	0x08	teex	Binary	nand0	0x700000	tee-pager.stm32
P	0x10	UBI	FileSystem	nand0	0x780000	ubi.bin

4.4 eMMC with TEE

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a.stm32
P	0x02	fsbl1	Binary	mmc1	boot1	tf-a-optee.stm32
P	0x04	fsbl2	Binary	mmc1	boot2	tf-a-optee.stm32
P	0x03	ssbl	Binary	mmc1	0x00080000	u-boot-optee.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.bin
P	0x12	rootfs	FileSytem	mmc1	0x05400000	rootfs.ext4.bin



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

4.5 SD card with TEE

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl	Binary	none	0x0	tf-a.stm32
P	0x02	fsbl1	Binary	mmc0	0x4400	tf-a-optee.stm32
P	0x04	fsbl2	Binary	mmc0	0x00044400	tf-a-optee.stm32
P	0x03	ssbl	Binary	mmc0	0x00084400	u-boot-optee.stm32
P	0x05	teeh	Binary	mmc0	0x00284400	tee-header.stm32
P	0x06	teed	Binary	mmc0	0x00294400	tee-pageable.stm32
P	0x07	teex	Binary	mmc0	0x002C4400	tee-pager.stm32
P	0x10	bootfs	System	mmc0	0x00304400	bootfs.ext4.bin
P	0x11	vendorfs	FileSytem	mmc0	0x04304400	vendorfs.ext4.bin
P	0x12	rootfs	FileSytem	mmc0	0x05304400	rootfs.ext4.bin
P	0x13	userfs	FileSytem	mmc0	0x35304400	userfs.ext4.bin

5 Using provided FlashLayout files

The binary and the associated pre-defined FlashLayout file 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_stm32mp157c-ev1-trusted.tsv`:

```
#Opt      Id      Name      Type      Device
Offset   Binary
P         0x01   fsbl1     Binary    mmc0
0x00004400   tf-a-stm32mp15c-ev1-trusted.stm32
P         0x02   fsbl2     Binary    mmc0
0x00044400   tf-a-stm32mp15c-ev1-trusted.stm32
P         0x03   ssbl      Binary    mmc0
0x00084400   u-boot-stm32mp15c-ev1-trusted.stm32
P         0x10   bootfs    System    mmc0
0x00294400   st-image-bootfs-openstlinux-weston-stm32mp1.ext4
P         0x11   vendorfs  FileSystem mmc0      0x00294400
st-image-vendorfs-openstlinux-weston-stm32mp1.ext4
P         0x12   rootfs    FileSystem mmc0
0x02294400   st-image-core-openstlinux-weston-stm32mp1.ext4
P         0x13   userfs    FileSystem mmc0
0x22295800   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

5.1 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	Binary	mmc0	0x00004400	tf-a-stm32mp15c-ev1-trusted.stm32
-	0x02	fsbl2	Binary	mmc0	0x00044400	tf-a-stm32mp15c-ev1-trusted.stm32
P	0x03	ssbl	Binary	mmc0	0x00084400	u-boot-stm32mp15c-ev1-trusted.stm32
P	0x10	bootfs	System	mmc0	0x00304400	st-image-bootfs-openstlinux-weston-stm32mp1.ext4
-	0x11	vendorfs	FileSystem	mmc0	0x04304400	st-image-vendorfs-openstlinux-weston-stm32mp1.ext4
-	0x12	rootfs	FileSystem	mmc0	0x05304400	st-image-core-openstlinux-weston-stm32mp1.ext4
-	0x13	userfs	FileSystem	mmc0	0x35304400	st-image-userfs-openstlinux-weston-stm32mp1.ext4

The associated text file becomes :

```
#Opt      Id      Name      Type      Device
Offset
-         0x01    fsbl1     Binary    mmc0
0x00004400    tf-a-stm32mp15c-ev1-trusted.stm32
-         0x02    fsbl2     Binary    mmc0
0x00044400    tf-a-stm32mp15c-ev1-trusted.stm32
P         0x03    ssbl      Binary    mmc0
0x00084400    u-boot-stm32mp15c-ev1-trusted.stm32
P         0x10    bootfs    System    mmc0
0x00294400    st-image-bootfs-openstlinux-weston-stm32mp1.ext4
-         0x11    vendorfs  FileSystem mmc0      0x00294400
st-image-vendorfs-openstlinux-weston-stm32mp1.ext4
-         0x12    rootfs    FileSystem mmc0
0x02294400    st-image-core-openstlinux-weston-stm32mp1.ext4
-         0x13    userfs    FileSystem mmc0
0x22295800    st-image-userfs-openstlinux-weston-stm32mp1.ext4
```



5.2 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 ST original file. The new Layout file is:

Opt	Part	Name	Type	Device	Offset	Binary
-	0x01	fsbl-boot	Binary	none	0x0	tf-a-stm32mp15c-ev1-trusted.stm32
-	0x03	ssbl-boot	Binary	none	0x0	u-boot-stm32mp15c-ev1-trusted.stm32
P	0x02	fsbl2	Binary	mmc0	0x00004400	<Customer-tf-a>.stm32
P	0x04	fsbl1	Binary	mmc0	0x00044400	<Customer-tf-a>.stm32
P	0x05	ssbl	Binary	mmc0	0x00084400	<Customer-u-boot>.stm32
P	0x10	bootfs	System	mmc0	0x00304400	st-image-bootfs-openstlinux-weston-stm32mp1.ext4
P	0x11	vendorfs	FileSyste m	mmc0	0x04304400	st-image-vendorfs-openstlinux-weston-stm32mp1.ext4
P	0x12	rootfs	FileSyste m	mmc0	0x05304400	st-image-core-openstlinux-weston-stm32mp1.ext4
P	0x13	userfs	FileSyste m	mmc0	0x35304400	st-image-userfs-openstlinux-weston-stm32mp1.ext4

The associated text file becomes :

```

#Opt      Id      Name      Type      Device
Offset
-         0x01   fsbl1     Binary    none
0x0
-         0x03   ssbl      Binary    none
0x0
P         0x02   fsbl1     Binary    mmc0
0x00004400   tf-a.stm32
P         0x04   fsbl2     Binary    mmc0
0x00044400   tf-a.stm32
P         0x05   ssbl      Binary    mmc0
0x00084400   u-boot.stm32
P         0x10   bootfs    System    mmc0

```



```

0x00294400 st-image-bootfs-openstlinux-weston-stm32mp1.ext4
P          0x11 vendorfs System mmc0
0x00294400 st-image-vendorfs-openstlinux-weston-stm32mp1.ext4
P          0x12 rootfs FileSystem mmc0
0x02294400 st-image-core-openstlinux-weston-stm32mp1.ext4
P          0x13 userfs FileSystem mmc0
0x22295800 st-image-userfs-openstlinux-weston-stm32mp1.ext4

```

6 Other FlashLayout examples

6.1 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. bin
PED	0x11	userfs	Empty	mmc0	0x400000	none

6.2 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.bin
PE	0x11	userfs	FileSytm	mmc0	0x400000	none

6.3 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.

6.4 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 eMMC 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
-         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     u-boot-env_nor Binary    nor0
0x00280000 none
PE       0x45     unused    Binary    nor0
0x002C0000 none

```

6.5 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 eMMC 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	FileSytam	mmc0	0x08284400	rootfs.ext4. bin
DEP	0x12	userfs	FileSytam	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



7 Reference list

- https://en.wikipedia.org/wiki/USB#Device_Firmware_Upgrade
- http://www.usb.org/developers/docs/devclass_docs/DFU_1.1.pdf Universal Serial Bus Device Class Specification for Device Firmware Upgrade, Version 1.1
- https://web.archive.org/web/20141011015811/http://www.usb.org/developers/docs/devclass_docs/DFU_1.1.pdf Archived from the original on 11 October 2014
- 4.04.1 https://en.wikipedia.org/wiki/GUID_Partition_Table
- 5.05.1 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.

Trusted Execution Environment

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 had serial 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 for volatile random-access semiconductor memories.)

Trusted Firmware for Arm Cortex-A

One Time Programmed

Secure Secret Provisioning

Power Management Integrated Circuit

Non Volatile Memory, like a flash memory

Device Firmware Upgrade

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

Memory Technology Device

Elliptic curve cryptography

Error Correction Capability

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



Logical Block Addressing