How to support EXT4 through MMC

Contents

Stable: 16.02.2021 - 15:56 / Revision: 16.02.2021 - 14:47

Stable: 16.02.2021 - 10:09 / Revision: 16.02.2021 - 14:47

A quality version of this page, approved on *16 February 2021*, was based off this revision.

Contents

1 Purpose

The purpose of this article is to introduce EXT4 filesystem:

- General information
- Main components
- How to use EXT4

2 Overview

EXT4 (fourth extended file system)[1][2] is an advanced level of the EXT3 filesystem which incorporates scalability and reliability enhancements for supporting large filesystems (64 bit) in keeping with increasing disk capacities and state-of-the-art feature requirements.

- EXT4 is backward-compatible with EXT3 and EXT2. It is possible to mount both EXT3 and EXT2 filesystems directly using the EXT4 filesystem driver.
- EXT4 can support volumes with sizes up to 1 exbibyte (EiB) and files with sizes up to 16 tebibytes (TiB).

This file system may be used on emmc/sd-card (please refer to the MMC framework). It does not work for raw Flash memory like NOR/NAND.

3 Kernel configuration

EXT4 support is activated by default in ST deliveries. Nevertheless, if a specific configuration is needed, this section indicates how EXT4 can be activated/deactivated in the kernel.

Activate EXT4 in the kernel configuration with the Linux Menuconfig tool: Menuconfig or how to configure kernel.

```
File systems  --->
    <*> The Extended 4 (ext4) filesystem
    [*]   Use ext4 for ext2 file systems
```

4 Using a EXT4 partition as root file system

Assuming a rootfs EXT4 image is already flashed to the memory device, the user has to provide:

- The partition that has to be mounted, using root=<partition_device_path> or root=PARTUUID=XXXX where X represents the unique id of a partition.
- The file system type (rootfstype=ext4 in that case). Optional, by default the kernel find the file system type of partition.

Please refer to the SD card memory mapping to check the "rootfs" location in ST deliveries.

In this case, the kernel command-line parameters [3] that have to be added are:

- In case PARTUUID is used.

```
root=PARTUUID=45e5fc02-d536-43a4-a941-94a8329afeaf
```

- In case the partition device path is used.

```
root=/dev/mmcblk0p6
```

5 Mounting an EXT4 partition

Assuming that the "userfs" partition has been flashed on partition 7, the below steps show how to mount this partition.
Please refer to the SD card memory mapping to check the "userfs" location in ST deliveries.

- Mount "userfs".

```
Board $> mount /dev/mmcblk0p7 /media/
```

- Check that "userfs" partition is mounted.

```
Board $> mount | grep "/media"
/dev/mmcblk0p7 on /media type ext4 (rw,sync,relatime)
```

6 Create a default EXT4 filesystem on a MMC partition

- Format a MMC partition (mmcblk0p7 will be used in this example).

```
Board $> mke2fs -t ext4 -L "testfs" /dev/mmcblk0p7
mke2fs 1.43.5 (04-Aug-2017)
/dev/mmcblk0p7 contains a ext4 file system
        created on Tue Aug  7 08:28:50 2018
Proceed anyway? (y,N) y
Creating filesystem with 163595 4k blocks and 40960 inodes
Filesystem UUID: b7c6e8f5-373c-4c91-aace-0c8f69649165
Superblock backups stored on blocks:
        32768, 98304

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

- Mount "testfs" with device partition path or with label.

```
Board $> mount /dev/mmcblk0p7 /media
```

```
Board $> mount /dev/disk/by-label/testfs /media
```

- Check that the file system is empty.

```
Board $> ls -la /media
total 21
drwxr-xr-x 3 root root  4096 Aug  7 08:34 .
drwxr-xr-x 3 root root  1024 Aug  7 08:38 ..
drwx------ 2 root root 16384 Aug  7 08:34 lost+found
```

- Create a random data file.

```
Board $> dd if=/dev/urandom of=/tmp/random.hex bs=1M count=100 conv=fsync
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 6.49739 s, 16.1 MB/s
```

- Copy the random data file in /media.

```
Board $> cp /tmp/random.hex /media/
```

- Un-mount /media.

```
Board $> umount /media
```

- Mount "testfs".

```
Board $> mount /dev/disk/by-label/testfs /media
```

- Check that the random data file created is identical in /tmp and /media.

```
Board $> md5sum /tmp/random.hex /media/random.hex
6ab2f920c81bba53b01f9e758116a172  /tmp/random.hex
6ab2f920c81bba53b01f9e758116a172  /media/random.hex
```

- Un-mount /media.

```
Board $> umount /media
```

7 References

Please refer to the following links for full description:

- Official ext4 wiki
- Kernel.org Documentation
- The kernel's command-line parameters

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.

Linux® is a registered trademark of Linus Torvalds.

MultimediaCard

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

Stable: 14.05.2020 - 09:26 / Revision: 14.05.2020 - 09:25

A quality version of this page, approved on *14 May 2020*, was based off this revision.

The MMC (MultiMediaCard) / SD (secure digital) / SDIO (secure digital input/output) subsystem implements a standard Linux® host driver to interface with MMC / SD memory cards or SDIO cards.

Contents

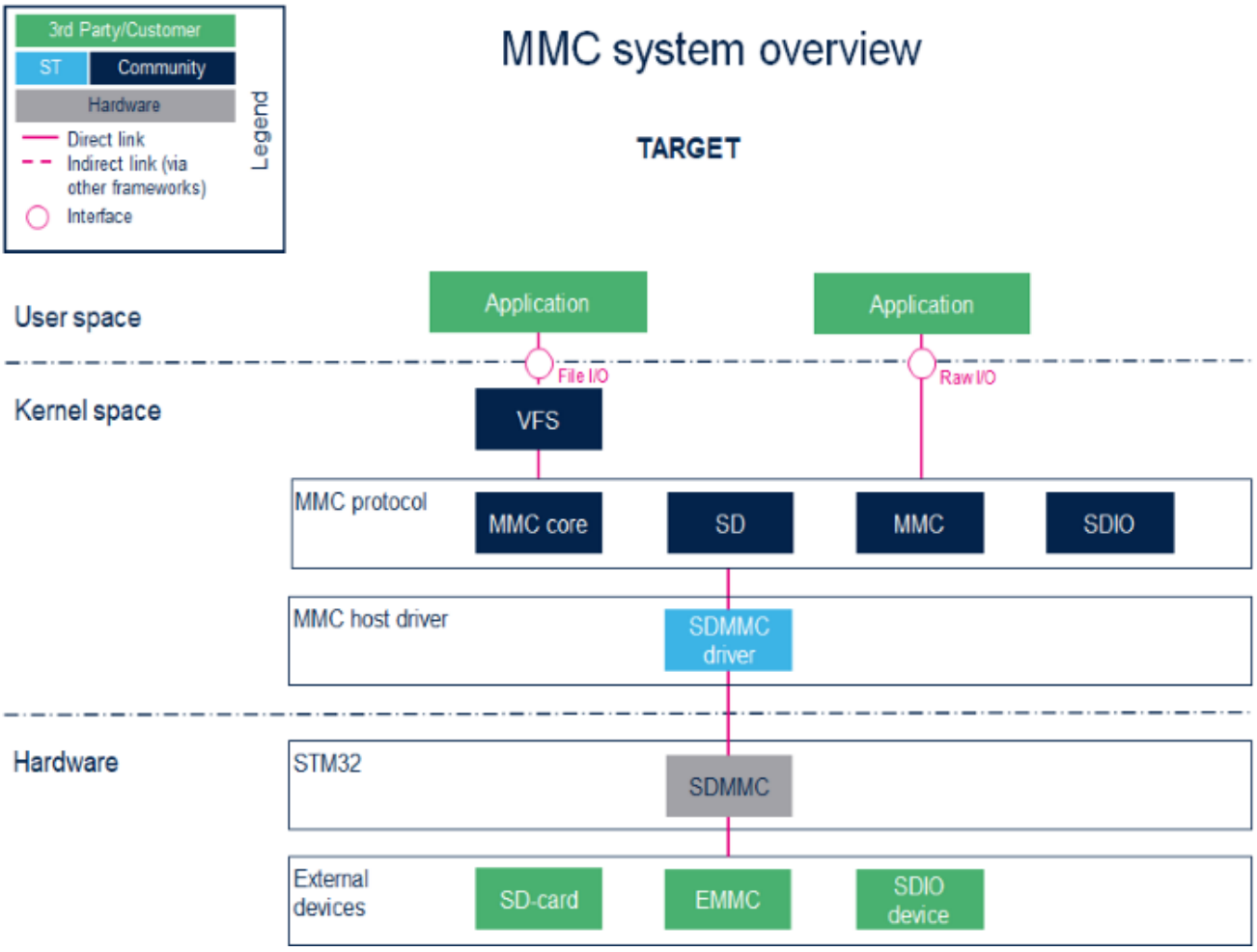
1 Framework purpose

The purpose of this article is to introduce the MMC Linux® subsystem (MMC / SD) by:

- providing general information
- describing the main components/stakeholders

The SDIO is addressed in the WLAN overview.

2 System overview



2.1 Component description

- User space applications handle **file I/O** management to view the card memory as a disk, whereas programs that perform **raw I/O** accesses see the memory as a block device[1].
- **VFS** (Kernel space)

Virtual File System. Please refer to the VFS documentation [2].

- **MMC core/SD/MMC/SDIO** (Kernel space)

The **MMC core** ensures compliance with MultiMediaCard (**MMC**)[3] / secure digital (**SD**)[4] / secure digital input/output (**SDIO**)[5].

- **SDMMC driver** (Kernel space) / **SDMMC** (hardware)

The **SDMMC driver** handles:

- the registers, the clock, the interrupt and the IDMA control.
- the communications over the bus based on command/response and data transfers.

Please refer to the SDMMC internal peripheral.

2.2 API description

The MMC core handles the file system read/write calls.

3 Configuration

3.1 Kernel configuration

The MMC framework is activated by default in ST deliveries. If a specific configuration is needed, this section indicates how the MMC framework can be activated/inactivated in the kernel.

The MMC framework can be activated in the kernel configuration via Linux® Menuconfig tool: Menuconfig or how to configure kernel

```
[*] Device Drivers
    [*] MMC/SD/SDIO card support
        <*> HW reset support for eMMC
        <*> Simple HW reset support for MMC
        <*> MMC block device driver
            (16) Number of minors per block device
        ...
        <*> ARM AMBA Multimedia Card Interface support
        [*] STMicroelectronics STM32 SDMMC Controller
```

3.2 Device tree configuration

DT configuration can be done thanks to STM32CubeMX.

Please refer to the SDMMC device tree configuration.

4 How to use the framework

A file system, which handles read/write/erase operations, can be used with the MMC framework. Please refer to the EXT4 support through MMC.

5 How to trace and debug the framework

5.1 How to monitor

The sysfs interface provides detailed information on each mmc device:

```
root:~# cat /sys/kernel/debug/mmc0/ios
clock:          50000000 Hz
vdd:            21 (3.3 ~ 3.4 V)
bus mode:       2 (push-pull)
chip select:    0 (don't care)
power mode:     2 (on)
bus width:      2 (4 bits)
timing spec:    2 (sd high-speed)
signal voltage: 0 (3.30 V)
driver type:    0 (driver type B)
```

5.2 How to trace

For details on dynamic trace usage, refer to How to use the kernel dynamic debug.

root:~# echo "file drivers/mmc/* +p" > /sys/kernel/debug/dynamic_debug/control

6 Source code location

The MMC framework is available here .

7 References

Please refer to the following links for a full description of the MMC framework:

- https://en.wikipedia.org/wiki/Device_file#Block_devices
- VFS
- MultiMediaCard, embedded MultiMediaCard specification
- Secure Digital, secure digital specification
- Secure Digital Input Output, Secure Digital Input Output specification

MultimediaCard

Linux® is a registered trademark of Linus Torvalds.

Secure digital

Virtual File System

Secure digital input/output

Application programming interface

SDIO is an SD-size card with extended input/output functions

former spelling for e•MMC ('e' in italic)

Device Tree

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)

Contents

1 Linux configuration genericity

The process of building a kernel has two parts: configuring the kernel options and building the source with those options.

The Linux® kernel configuration is found in the generated file: .config.

.config is the result of configuring task which is processing platform defconfig and fragment files if any.

For OpenSTLinux distribution the defconfig is located into the kernel source code and fragments into stm32mp BSP layer :

- arch/arm/configs/**multi_v7_defconfig**

Every new kernel version brings a bunch of new options, we do not want to back port them into a specific defconfig file each time the kernel releases, so we use the same defconfig file based on ARM SoC v7 architecture.
STM32MP1 specificities are managed with fragments config files.

- meta-st/meta-st-stm32mp/recipes-kernel/linux/linux-stm32mp/<kernel version>/**fragment-*.config**

.config result is located in the build folder:

- build-openstlinuxweston-stm32mp1/tmp-glibc/work/stm32mp1-ostl-linux-gnueabi/linux-stm32mp/4.14-48/linux-stm32mp1-standard-build/**.config**

To modify the kernel options, it is not recommended to edit this file directly.

- A user runs either a text-mode :

```
   PC $> make config
starts a character based question and answer session (Figure 1)
```

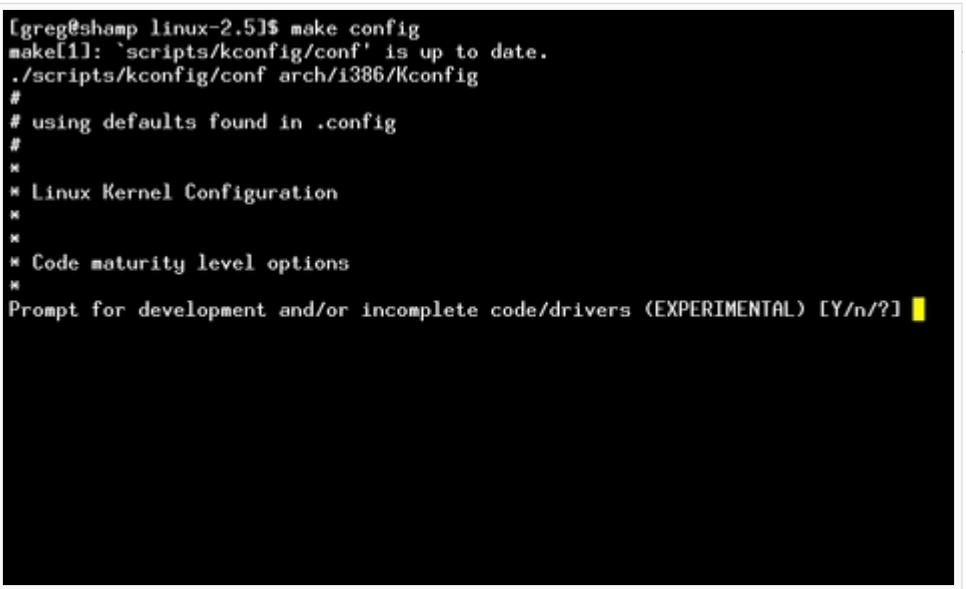


Figure 1. Configuring the kernel with make config

```
   PC $> make
menuconfig
starts a terminal-
oriented
configuration tool
(using ncurses)
(Figure 2)
The ncurses text
version is more
popular and is run
with the make
menuconfig option.
Wikipedia Menuconfig[1
] also explains how
to "navigate" within
the configuration
menu, and highlights
main key strokes.
```

- or a graphical kernel

configurator :

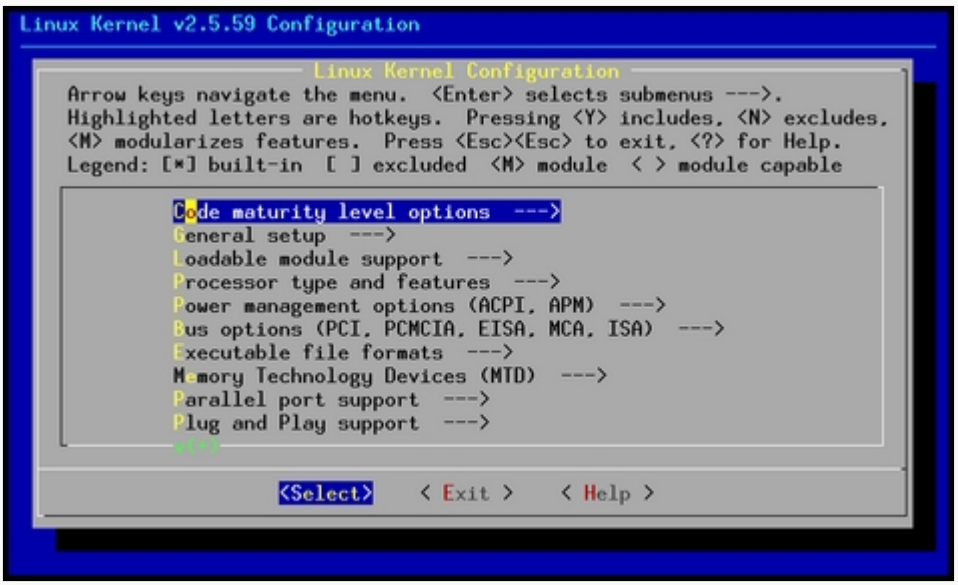


Figure 2. Make menuconfig makes it easier to back up and correct mistakes

PC $> make xconfig starts a X based configuration tool (Figure 3)

Ultimately these configuration tools edit the .config file.

An option indicates either some driver is built into the kernel ("=y") or will be built as a module ("=m") or is not selected.

The unselected state can either be indicated by a line starting with "#" (e.g. "# CONFIG_SCSI is not set") or by the absence of the relevant line from the .config file.

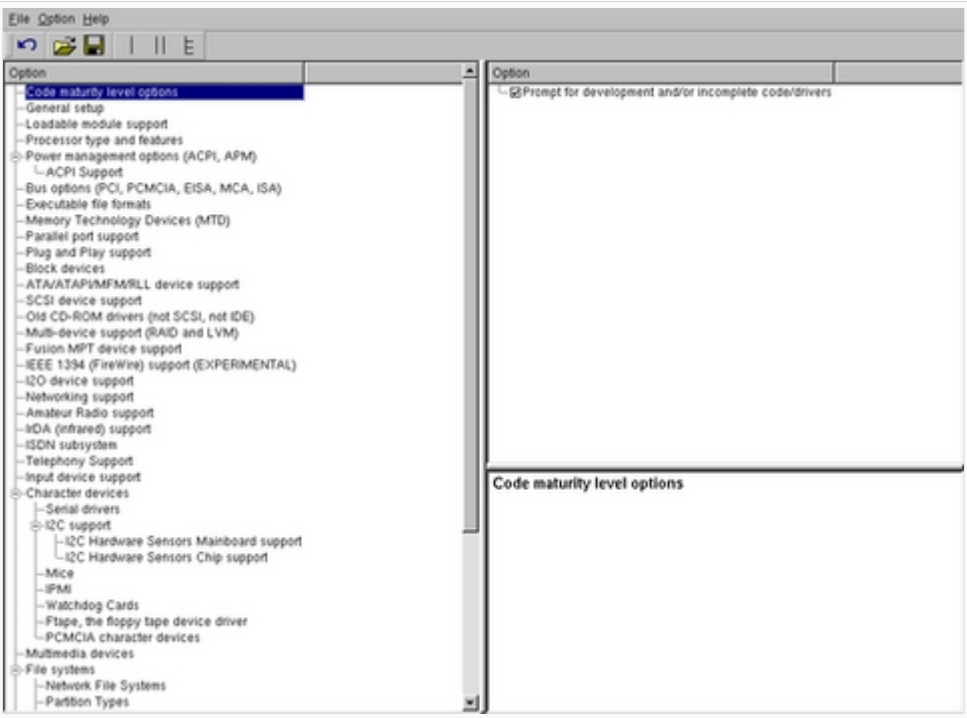


Figure 3. The Qt-Based make xconfig

The 3 states of the main selection option for the SCSI subsystem (which actually selects the SCSI mid level driver) follow. Only one of these should appear in an actual .config file:

```
CONFIG_SCSI=y
CONFIG_SCSI=m
# CONFIG_SCSI is not
set
```

2 Menuconfig and Developer Package

For this use case, the prerequesite is that OpenSTLinux SDK has been installed and configured.

To verify if your cross-compilation environment has been put in place correctly, run the following command:

```
PC $> set | grep CROSS
CROSS_COMPILE=arm-ostl-linux-gnueabi-
```

For more details, refer to *<Linux kernel installation directory>/README.HOW_TO.txt* helper file (the latest version of this helper file is also available in GitHub: README.HOW_TO.txt).

- Go to the <Linux kernel build directory>

```
PC $> cd <Linux kernel build directory>
```

- Save initial configuration (to identify later configuration updates)

```
PC $> make arch=ARM savedefconfig
Result is stored in defconfig file
PC $> cp defconfig defconfig.old
```

- Start the Linux kernel configuration menu

```
PC $> make arch=ARM menuconfig
```

- Navigate forwards or backwards directly between feature
  - un/select, modify feature(s) you want
  - When the configuration is OK : exit and save the new configuration

```
useful keys to know:
enter: enter in config subdirectory
space: hit several times to either select [*], select in module [m] or unselect [ ]
/: to search for a keyword, this is usefull to navigate in tree
?: to have more information on selected line
```

- Compare the old and new config files after operating modifications with menuconfig

```
PC $> make arch=ARM savedefconfig
```

Retrieve configuration updates by comparing the new defconfig and the old one

```
PC $> meld defconfig defconfig.old
```

- Cross-compile the Linux kernel (please check the load address in the *README.HOW_TO.txt* helper file)

```
PC $> make arch=ARM uImage LOADADDR=<loadaddr of kernel>
PC $> cp arch/arm/boot/uImage install_artifact/boot/
```

- Update the Linux kernel image on board

```
PC $> scp install_artifact/boot/uImage root@<board ip address>:/boot/
```

Information

If the */boot* mounting point doesn't exist yet, please see how to create a mounting point

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

Note that this use case modifies the configuration file in the Linux kernel build directory, not in the Linux kernel source directory: this is a temporary modification useful for a prototyping.

- To make this temporary modification permanent, the delta between defconfig and defconfig.old must be saved in a configuration fragment file (fragment-*.config) based on fragment.cfg file, and the Linux kernel configuration/compilation steps must be re-executed (as explained in the README.HOW_TO.txt helper file).

3 Menuconfig and Distribution Package

- Start the Linux kernel configuration menu

```
PC $> bitbake virtual/kernel -c menuconfig
```

- Navigate forwards or backwards directly between feature
  - un/select, modify feature(s) you want
  - When the configuration is OK : exit and save the new configuration

```
useful keys to know:
enter: enter in config subdirectory
space: hit several times to either select [*], select in module [m] or unselect [ ]
/: to search for a keyword, this is usefull to navigate in tree
?: to have more information on selected line
```

- Cross-compile the Linux kernel

```
PC $> bitbake virtual/kernel
```

- Update the Linux kernel image on board

```
PC $> scp <build dir>/tmp-glibc/deploy/images/<machine name>/uImage root@<board ip
address>:/boot
```

Information

If the */boot* mounting point does not exist yet, please see how to create a mounting point

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

Note that this use case modifies the configuration file in the Linux kernel build directory, not in the Linux kernel source directory: this is a temporary modification useful for a prototyping.

- To make this temporary modification permanent, it must be saved in a configuration fragment file (fragment-*.config) based on **fragment.cfg** file, and the Linux kernel configuration/compilation steps must be re-executed: **bitbake <name of kernel recipe>**.

4 References

- Wikipedia Menuconfig

Linux® is a registered trademark of Linus Torvalds.

Board support package

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

Stable: 20.11.2020 - 17:03 / Revision: 20.11.2020 - 17:02

A quality version of this page, approved on *20 November 2020*, was based off this revision.

Contents

1 Supported Flash memory technologies

STM32MP15 boards support the following types of Flash memory:

- SD card on the SDMMC interface present on EVAL and DISCO boards
- *e*•MMC on the SDMMC interface present on EVAL board only
- Serial NOR Flash memory on the Dual QSPI interface present on EVAL board only
- NAND Flash memory on the FMC interface present on EVAL board only.

The next section lists all partitions used on STM32MP15 boards (size, name, and content), and the following sections show how they are mapped on the different types of Flash memory.

2 Flash partitions

The tables below list the partitions defined for STMP32MP15 boards.

2.1 Minimal

Size	Component	Comment
Remaining area	userfs	The user file system contains user data and examples
768 Mbytes	rootfs	Linux root file system contains all user space binaries (executable, libraries, and so on), and kernel modules
16 Mbytes	vendorfs	This partition is preferred to the rootfs for third-party proprietary binaries, and ensures that they are not contaminated by any open source licence, such as GPL v3
64 Mbytes	bootfs	The boot file system contains: • (option) the init RAM file system, which 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 except for 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)
2 Mbytes	ssbl	The second stage boot loader (SSBL) is U-Boot, with its device tree blob (dtb) appended at the end
256 Kbytes to 512 Kbytes (*)	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, the FSBL payload is limited to 247 Kbytes.

(*): The partition size depends on the Flash technology, to be aligned to the block erase size of the Flash memory present on the board: NOR (256 Kbytes) / NAND (512 Kbytes).

Information

Some boards can be equipped with multiple Flash devices, like the EVAL board, where all of the Flash devices can be programmed with STM32CubeProgrammer. However, caution must be taken for the serial NOR/NAND and SLC NAND because a **static bootable MTD partitioning** is defined in U-Boot include/configs/stm32mp1.h (look for *STM32MP_MTDPARTS*), with the consequence that up to 6 Mbytes of space is lost at the beginning of each such device, **even those which are not bootable**.

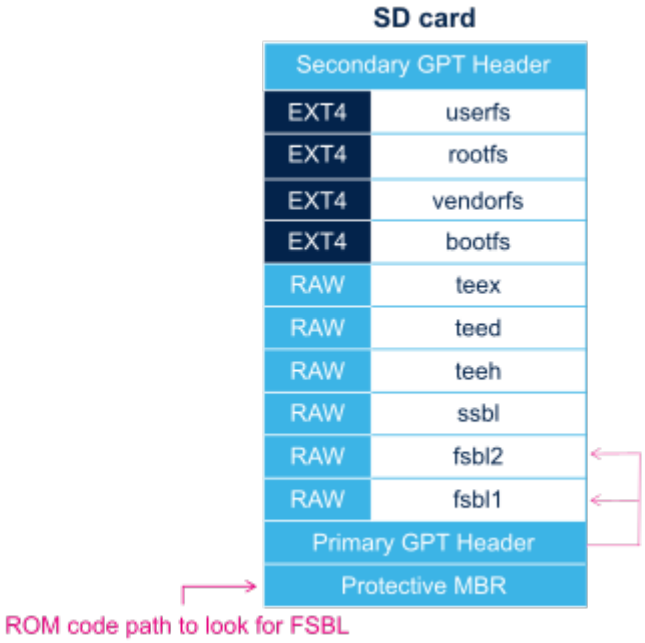
2.2 Optional

Size	Component	Comment
2 * 256 Kbytes (*)	env	This partition is used to store the U-Boot environment while booting from NOR Flash. The information is stored twice, for redundancy. For all other Flash devices, the U-Boot environment is stored either in an EXT4 bootfs partition (*e*•MMC, SD card), or UBI volumes (NAND).
256 Kbytes to 512 Kbytes (*)	teeh	OP-TEE header
256 Kbytes to 512 Kbytes (*)	teed	OP-TEE pageable code and data
256 Kbytes to 512 Kbytes (*)	tex	OP-TEE pager

(*): The partition size depends on the Flash technology, as it should be aligned to the block erase size of the Flash device present on the board (256 Kbytes for NOR and 512 Kbytes for NAND).

3 SD card memory mapping

The SD card has to be partitioned with GPT format in order to be recognized by the STM32MP15. The easiest way to achieve this is to use STM32CubeProgrammer.
The ROM code looks for the GPT entries whose name begins with "fsbl": fsbl1 and fsbl2 for example.
Note: The SD card can be unplugged from the board and inserted into a Linux host computer for direct partitioning with Linux utilities and access to the **bootfs**, **rootfs** and **userfs** partitions. The file system is Linux EXT4.



4 e•MMC memory mapping

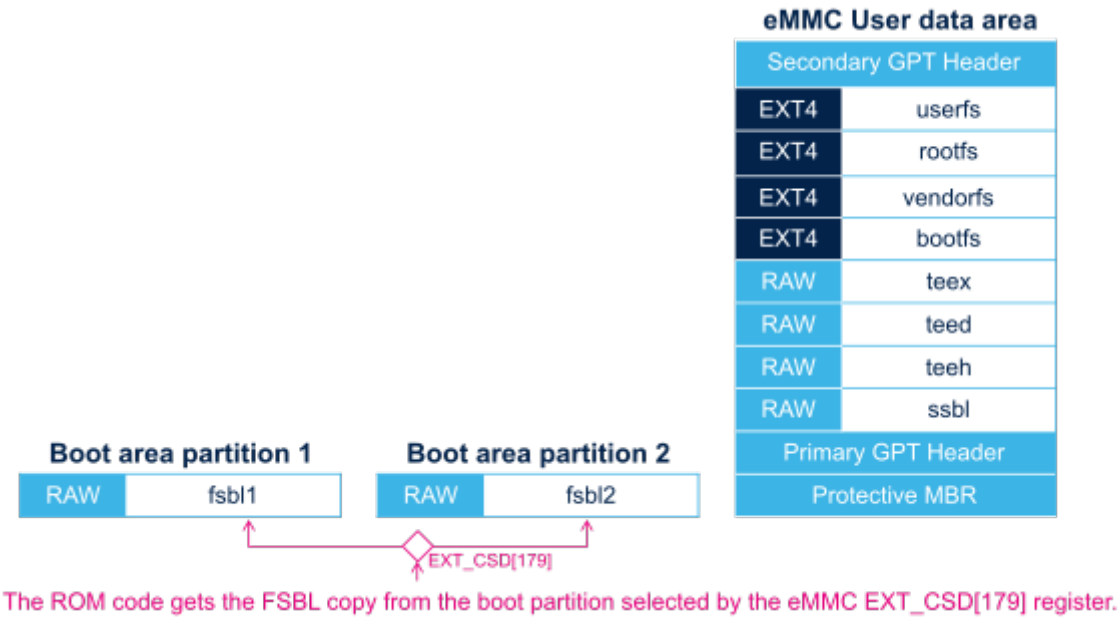
The e•MMC embeds four physical partitions:

- Boot area partition 1: one copy of the FSBL
- Boot area partition 2: one copy of the FSBL
- User data area: formatted with GPT partitioning and used to store all remaining partitions
- Replay Protected Memory Block (RPMB): not shown in the figure below, since not involved in the current boot chain.

STM32CubeProgrammer has to be used to prepare the e•MMC with the layout shown below, and to populate each partition.

Information

The boot area partition used by the e•MMC boot sequence is selected via the EXT_CSD[179] register in the e•MMC. The STM32CubeProgrammer execution is concluded with the selection of the last written partition from the flashlayout file, typically partition 2. The other copy is never used as long as the user does not explicitly change the e•MMC EXT_CSD[179] register to select it.



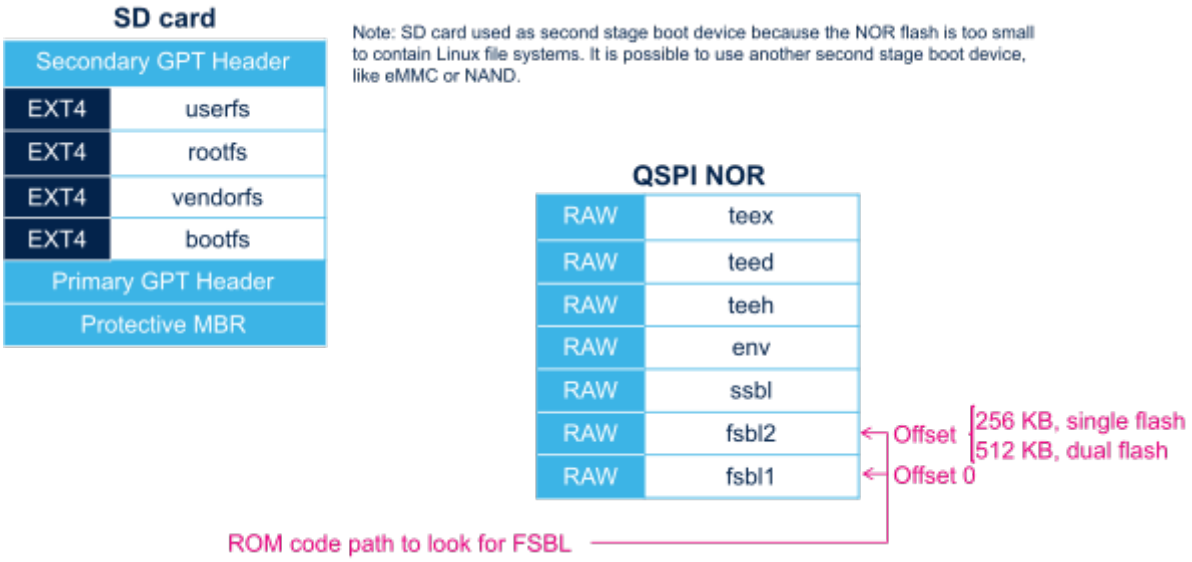
The ROM code gets the FSBL copy from the boot partition selected by the eMMC EXT_CSD[179] register.

5 NOR memory mapping

As NOR Flash memory is expensive, its size is usually limited to the minimum needed to store only the bootloaders. The system files (bootfs, rootfs and userfs) are usually stored in another Flash memory, such as the SD card in OpenSTLinux distribution.
STM32CubeProgrammer must be used to prepare the NOR Flash and the SD card with the layout shown below, and to populate each partition.

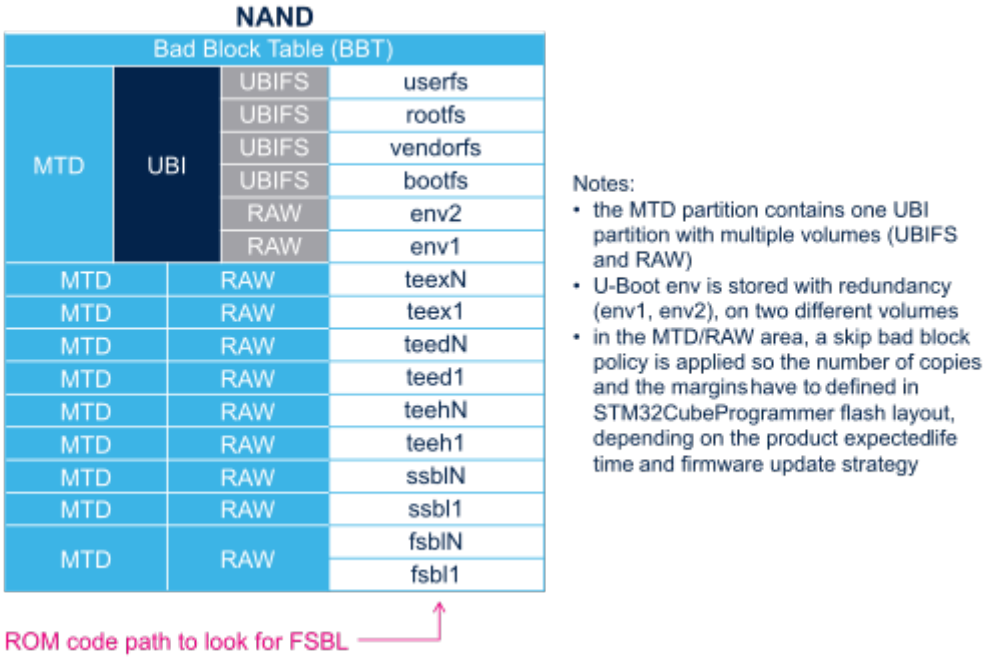
It is possible to use an *e*•MMC card or NAND as second-level Flash memory, rather than an SD card. This requires the following aspects to be changed:

- The Flash memory layout, using STM32CubeProgrammer in order to write the rootfs and userfs to the targeted Flash memory
- The Linux kernel parameters, using U-Boot, in order to indicate where the rootfs and userfs have to be mounted.



6 NAND memory mapping

STM32CubeProgrammer has to be used to prepare the NAND Flash memory with the layout shown below, and to populate each partition.



Warning

In the **RAW/MTD area**, the number of copies to embed and the number of blocks to reserve at the end of each partition, for future bad blocks replacement, are a critical part of NAND based product dimensioning !

The strategy has to take into account many parameters such as:

- the binaries sizes,
- the read accesses to those partitions during product life, that may generate read disturb effect,
- the capability of the product to refresh the partitions content when erroneous bits are detected,
- the number of software updates estimated on this partition,
- the selected NAND flash characteristics

ST set foundations in the STM32MP15 device in order to allow the integration of NAND flash memories but the product defintion remains the customer responsibility. Please, contact your memory provider for further advice.

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.

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

MultimediaCard

Linux® is a registered trademark of Linus Torvalds.

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 for volatile random-acces ssemiconductor memories.)

Flattened uImage Tree is a packaging format used by U-Boot

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

Second Stage Boot Loader

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Trusted Firmware for Arm Cortex-A

Secondary Program Loader, *Also known as* ***U-Boot SPL***

Read Only Memory

First Stage Boot Loader

Single-Level Cell is a kind of NAND flash

Memory Technology Device

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

Open Portable Trusted Execution Environment

GUID Partition Table