Category:Mass storage

Contents

Stable: 17.06.2020 - 15:26 / Revision: 16.01.2020 - 09:14

Category:Mass storage

Stable: 17.06.2020 - 10:20 | Revision: 16.01.2020 - 09:14

A quality version of this page, approved on *17 June 2020*, was based off this revision.

This category groups together all articles related to the Linux® **mass storage** software frameworks.

It is recommended to first read the MMC overview and MTD overview articles.

Pages in category "Mass storage"

The following 7 pages are in this category, out of 7 total.

- FMC device tree configuration
- How to support EXT4 through MMC
- How to support UBIFS through MTD
- MMC overview
- MTD overview
- QUADSPI device tree configuration
- SDMMC device tree configuration

Stable: 15.04.2021 - 16:08 / Revision: 15.04.2021 - 16:07

A quality version of this page, approved on *15 April 2021*, was based off this revision.

Contents

1 For ecosystem release v2.1.0

1.1 Article purpose

This article explains how to configure the **FMC** internal peripheral when it is assigned to the Linux®OS. In that case, the FMC NAND Flash controller is controlled by the MTD framework.

The configuration is performed using the device tree mechanism that provides a hardware description of the FMC peripheral, used by the STM32 FMC Linux drivers and by the MTD framework.

1.2 DT bindings documentation

The FMC device tree bindings are composed of:

- generic MTD NAND bindings [1].
- FMC NAND Flash controller driver bindings [2].
- FMC external bus interface driver bindings [3].

1.3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

1.3.1 DT configuration (STM32 level)

The FMC peripheral node is located in *stm32mp151.dtsi*[4] file.

```
   fmc: memory-controller@58002000 {                          Comments
       #address-cells = <2>;
       #size-cells = <1>;
       compatible = "st,stm32mp1-fmc2-ebi";
       reg = <0x58002000 0x1000>;                             --> This region contains the
register location
       clocks = <&rcc FMC_K>;
       resets = <&rcc FMC_R>;
       status = "disabled";

       ranges = <0 0 0x60000000 0x04000000>, /* EBI CS 1 */ --> External bus interface
region is used to address up to four external devices
                <1 0 0x64000000 0x04000000>, /* EBI CS 2 */
                <2 0 0x68000000 0x04000000>, /* EBI CS 3 */
                <3 0 0x6c000000 0x04000000>, /* EBI CS 4 */
                <4 0 0x80000000 0x10000000>; /* NAND */     --> NAND Flash controller
region is used  to address NAND Flash memory devices

       nand-controller@4,0 {
           #address-cells = <1>;
           #size-cells = <0>;
           compatible = "st,stm32mp1-fmc2-nfc";
           reg = <4 0x00000000 0x1000>,                       --> Regions 1 to 3
respectively contain the data, command and address space for CS0
```

```
                <4 0x08010000 0x1000>,
                <4 0x08020000 0x1000>,
                <4 0x01000000 0x1000>,                          --> Regions 4 to 6 contain
the same areas for CS1
                <4 0x09010000 0x1000>,
                <4 0x09020000 0x1000>;
            interrupts = <GIC_SPI 48 IRQ_TYPE_LEVEL_HIGH>;      --> The interrupt number used
            dmas = <&mdma1 20 0x2 0x12000a02 0x0 0x0 0x0>,      --> DMA specifiers [5]
                   <&mdma1 20 0x2 0x12000a08 0x0 0x0 0x0>,
                   <&mdma1 21 0x2 0x12000a0a 0x0 0x0 0x0>;
            dma-names = "tx", "rx", "ecc";
            status = "disabled";
        };
    };
```

Warning

This device tree part related to the STM32 should be kept as is, customer should not modify it.

1.3.2 DT configuration of the external bus interface controller (board level)

The FMC external bus interface controller may connect up to four external devices.

```
    &fmc {                                                      Comments
        pinctrl-names = "default", "sleep";                     --> For pinctrl
configuration, please refer to Pinctrl device tree configuration
        pinctrl-0 = <&fmc2_pins_b>;
        pinctrl-1 = <&fmc2_sleep_pins_b>;
        status = "okay";                                        --> Enable the node

        ksz8851: ks8851mll@1,0 {
            compatible = "micrel,ks8851-mll";                   --> Configure the external
device
            reg = <1 0x0 0x2>, <1 0x2 0x20000>;
            interrupt-parent = <&gpioc>;
            interrupts = <3 IRQ_TYPE_LEVEL_LOW>;
            bank-width = <2>;

            st,fmc2-ebi-cs-mux-enable;                          --> Configure the
transactions with the external device
            st,fmc2-ebi-cs-transaction-type = <4>;
            st,fmc2-ebi-cs-buswidth = <16>;
            st,fmc2-ebi-cs-address-setup-ns = <5>;
            st,fmc2-ebi-cs-address-hold-ns = <5>;
            st,fmc2-ebi-cs-bus-turnaround-ns = <5>;
            st,fmc2-ebi-cs-data-setup-ns = <45>;
            st,fmc2-ebi-cs-data-hold-ns = <1>;
        };
    };
```

1.3.3 DT configuration of the NAND Flash controller (board level)

The FMC NAND Flash controller may connect to one SLC NAND Flash memory (with a maximum of 2 dies per package).

```
    &fmc {                                                      Comments
        pinctrl-names = "default", "sleep";                     --> For pinctrl
configuration, please refer to Pinctrl device tree configuration
        pinctrl-0 = <&fmc2_pins_a>;
        pinctrl-1 = <&fmc2_sleep_pins_a>;
        status = "okay";                                        --> Enable the node
```

```
        nand-controller@4,0 {
            status = "okay";                                    --> Enable the NAND
controller node

            nand@0 {
                reg = <0>;                                      --> Describe the CS line
assigned to the NAND chip
                nand-on-flash-bbt;                              --> Store the bad block table
on NAND Flash memory
                nand-ecc-strength = <8>;                        --> Number of bits to correct
per ECC step
                nand-ecc-step-size = <512>;                     --> Number of data bytes that
are covered by a single ECC step
                #address-cells = <1>;
                #size-cells = <1>;
            };
        };
    };
```

The supported ECC strength and step size are:

- nand-ecc-strength = <1>, nand-ecc-step-size = <512> (HAMMING).
- nand-ecc-strength = <4>, nand-ecc-step-size = <512> (BCH4).
- nand-ecc-strength = <8>, nand-ecc-step-size = <512> (BCH8).

Warning

It is recommended to check the ECC requirements in the datasheet of the memory provider.

1.3.4 DT configuration examples

The below example shows how to configure the FMC NAND Flash controller when a SLC 8-bit NAND Flash memory device is connected (ECC requirement: 8 bits / 512 bytes).

```
    &fmc {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&fmc2_pins_a>;
        pinctrl-1 = <&fmc2_sleep_pins_a>;
        status = "okay";

        nand-controller@4,0 {
            status = "okay";

            nand: nand@0 {
                reg = <0>;
                nand-on-flash-bbt;
                #address-cells = <1>;
                #size-cells = <1>;

                partition@0 {
                    ...
                };
            };
        };
    };
```

The below example shows how to configure the FMC NAND Flash controller when a SLC 8-bit NAND Flash memory device is connected (ECC requirement: 4 bits / 512 bytes).

```
    &fmc {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&fmc2_pins_a>;
        pinctrl-1 = <&fmc2_sleep_pins_a>;
        status = "okay";

        nand-controller@4,0 {
            status = "okay";

            nand: nand@0 {
                reg = <0>;
                nand-on-flash-bbt;
                nand-ecc-strength = <4>;
                nand-ecc-step-size =
<512>;
                #address-cells = <1>;
                #size-cells = <1>;

                partition@0 {
                    ...
                };
            };
        };
    };
```

1.4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

1.5 References

Please refer to the following links for full description:

- Documentation/devicetree/bindings/mtd/nand-controller.yaml (v5.4-stm32mp-r2)
- Documentation/devicetree/bindings/mtd/st,stm32-fmc2-nand.yaml (v5.4-stm32mp-r2)
- Documentation/devicetree/bindings/memory-controllers/st,stm32-fmc2-ebi.yaml (v5.4-stm32mp-r2)
- arch/arm/boot/dts/stm32mp151.dtsi (v5.4-stm32mp-r2)
- Documentation/devicetree/bindings/dma/stm32-mdma.txt (v5.4-stm32mp-r2)

2 For ecosystem release v2.0.0

2.1 Article purpose

This article explains how to configure the **FMC** internal peripheral when it is assigned to the Linux®OS. In that case, it is controlled by the MTD framework.

The configuration is performed using the device tree mechanism that provides a hardware description of the FMC peripheral, used by the STM32 FMC Linux driver and by the MTD framework.

2.2 DT bindings documentation

The FMC device tree bindings are composed of:

- generic MTD nand bindings [1].
- FMC driver bindings [2].

2.3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

2.3.1 DT configuration (STM32 level)

The FMC peripheral node is located in *stm32mp151.dtsi*[3] file.

```
   fmc: nand-controller@58002000 {                                 Comments
       compatible = "st,stm32mp15-fmc2";
       reg = <0x58002000 0x1000>,                                  --> First region contains
the register location
             <0x80000000 0x1000>,                                  --> Regions 2 to 4
respectively contain the data, command and address space for CS0
             <0x88010000 0x1000>,
             <0x88020000 0x1000>,
             <0x81000000 0x1000>,                                  --> Regions 5 to 7 contain
the same areas for CS1
             <0x89010000 0x1000>,
             <0x89020000 0x1000>;
       interrupts = <GIC_SPI 48 IRQ_TYPE_LEVEL_HIGH>;              --> The interrupt number used
       dmas = <&mdma1 20 0x10 0x12000A02 0x0 0x0 0>,               --> DMA specifiers [4]
              <&mdma1 20 0x10 0x12000A08 0x0 0x0 0>,
              <&mdma1 21 0x10 0x12000A0A 0x0 0x0 0>;
       dma-names = "tx", "rx", "ecc";
       clocks = <&rcc FMC_K>;
       resets = <&rcc FMC_R>;
       status = "disabled";
   };
```

Warning

This device tree part related to the STM32 should be kept as is, customer should not modify it.

2.3.2 DT configuration (board level)

The FMC peripheral may connect to one SLC NAND Flash memory (with a maximum of 2 dies per package).

```
   &fmc {                                                          Comments
        pinctrl-names = "default", "sleep";                        --> For pinctrl
configuration, please refer to Pinctrl device tree configuration
        pinctrl-0 = <&fmc2_pins_a>;
        pinctrl-1 = <&fmc2_sleep_pins_a>;
        status = "okay";                                           --> Enable the node
        #address-cells = <1>;
        #size-cells = <0>;

        nand: nand@0 {
            reg = <0>;                                             --> Describe the CS line
assigned to the NAND chip
            nand-on-flash-bbt;                                     --> Store the bad block
table on NAND Flash memory
            nand-ecc-strength = <8>;                               --> Number of bits to
correct per ECC step
            nand-ecc-step-size = <512>;                            --> Number of data bytes
that are covered by a single ECC step
            #address-cells = <1>;
            #size-cells = <1>;
        };
   };
```

The supported ECC strength and step size are:

- nand-ecc-strength = <1>, nand-ecc-step-size = <512> (HAMMING).
- nand-ecc-strength = <4>, nand-ecc-step-size = <512> (BCH4).
- nand-ecc-strength = <8>, nand-ecc-step-size = <512> (BCH8).

2.3.3 DT configuration examples

The below example shows how to configure the FMC controller when a SLC 8-bit NAND Flash memory device is connected (ECC requirement: 8 bits / 512 bytes).

```
   &fmc {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&fmc2_pins_a>;
        pinctrl-1 = <&fmc2_sleep_pins_a>;
        status = "okay";
        #address-cells = <1>;
        #size-cells = <0>;

        nand: nand@0 {
            reg = <0>;
            nand-on-flash-bbt;
            #address-cells = <1>;
            #size-cells = <1>;

            partition@0 {
                ...
            };
        };
   };
```

The below example shows how to configure the FMC controller when a SLC 8-bit NAND Flash memory device is connected (ECC requirement: 4 bits / 512 bytes).

```
&fmc {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&fmc2_pins_a>;
    pinctrl-1 = <&fmc2_sleep_pins_a>;
    status = "okay";
    #address-cells = <1>;
    #size-cells = <0>;

    nand: nand@0 {
        reg = <0>;
        nand-on-flash-bbt;
        nand-ecc-strength = <4>;
        nand-ecc-step-size = <512>;
        #address-cells = <1>;
        #size-cells = <1>;

        partition@0 {
            ...
        };
    };
};
```

2.4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

2.5 References

Please refer to the following links for full description:

- Documentation/devicetree/bindings/mtd/nand-controller.yaml (v5.4-stm32mp-r1)
- Documentation/devicetree/bindings/mtd/stm32-fmc2-nand.txt (v5.4-stm32mp-r1)
- arch/arm/boot/dts/stm32mp151.dtsi (v5.4-stm32mp-r1)
- Documentation/devicetree/bindings/dma/stm32-mdma.txt (v5.4-stm32mp-r1)

Operating System

Memory Technology Device

Device Tree

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.

Generic Interrupt Controller

Serial Peripheral Interface

Direct Memory Access

Single-Level Cell is a kind of NAND flash

Elliptic curve cryptography

Error Correction Capability

Stable: 16.02.2021 - 15:56 / 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: 03.02.2020 - 08:05 / Revision: 03.02.2020 - 08:03

A quality version of this page, approved on *3 February 2020*, was based off this revision.

Contents

1 Purpose

The purpose of this article is to introduce the UBIFS file system:

- General information
- Main components
- How to use UBIFS

2 Overview

UBI/UBIFS[1] is designed to work on top of raw Flash memories (NOR/NAND). It does not work on top of block devices (MMC /SD cards).
3 subsystems are involved with UBIFS:

- The UBIFS file system, which works on top of UBI volumes. Please refer to the MTD UBIFS documentation[2].
- The UBI subsystem, which works on top of MTD devices. This subsystem is a wear-leveling and volume management system for raw Flash memories. Please refer to the MTD UBI documentation[3].
- The MTD subsystem, which provides uniform interfaces to access raw Flash memories. Please refer to the MTD framework.

3 Kernel configuration

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

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

```
[*] Device Drivers --->
    <*> Memory Technology Device (MTD) support --->
        <*> Enable UBI- Unsorted block images --->
File systems --->
    [*] Miscellaneous filesystems --->
        <*> UBIFS file system support
```

4 Using a UBIFS partition as root file system

Assuming the UBIFS image is already flashed to the raw Flash memory, the user has to provide:

- The boot arguments to attach the UBI device (using ubi.mtd=X), where X is the MTD device which hosts the root file system volume, named "rootfs".
- The file system type (using rootfstype=ubifs).
- The UBI volume that has to be mounted (using root=ubiX_T or root=ubiX:NAME), where X is the UBI device number, Y is the UBI volume number and NAME is the UBI volume name.

Please refer to the NAND memory mapping to check the "rootfs" location in ST deliveries.
The following is an example of the kernel boot arguments assuming that the rootfs has been flashed on MTD partition 3 (/dev /mtd3) and this MTD partition has an UBI volume named "rootfs". In this case, the kernel command-line[4] will be:

```
ubi.mtd=3 root=ubi0:rootfs rootfstype=ubifs rootwait rw console=ttySTM0,115200
```

5 Mounting a UBIFS partition

Assuming that the "userfs" volume has been flashed on MTD partition 3 (/dev/mtd3), below steps show how to mount this volume.

Please refer to the NAND memory mapping to check the "userfs" location in ST deliveries.

- Attach mtd3 to UBI.

```
 Board $> ubiattach /dev/ubi_ctrl -m 3
 ubi0: attaching mtd3
 ubi0: scanning is finished
 ubi0: attached mtd3 (name "UBI", size 1018 MiB)
 ubi0: PEB size: 262144 bytes (256 KiB), LEB size: 253952 bytes
 ubi0: min./max. I/O unit sizes: 4096/4096, sub-page size 4096
 ubi0: VID header offset: 4096 (aligned 4096), data offset: 8192
 ubi0: good PEBs: 4068, bad PEBs: 4, corrupted PEBs: 0
 ubi0: user volume: 4, internal volumes: 1, max. volumes count: 128
 ubi0: max/mean erase counter: 2/0, WL threshold: 4096, image sequence number: 980643132
 ubi0: available PEBs: 0, total reserved PEBs: 4068, PEBs reserved for bad PEB handling:
76
 ubi0: background thread "ubi_bgt0d" started, PID 1138
 UBI device number 0, total 4068 LEBs (1033076736 bytes, 985.2 MiB), available 0 LEBs (0
bytes), LEB size 253952 bytes (248.0 KiB)
```

- Mount "userfs" volume.

```
 Board $> mount -t ubifs ubi0:userfs /media
 UBIFS (ubi0:3): background thread "ubifs_bgt0_3" started, PID 648
 UBIFS (ubi0:3): UBIFS: mounted UBI device 0, volume 3, name "userfs"
 UBIFS (ubi0:3): LEB size: 253952 bytes (248 KiB), min./max. I/O unit sizes: 4096 bytes
/4096 bytes
 UBIFS (ubi0:3): FS size: 138911744 bytes (132 MiB, 547 LEBs), journal size 9404416 bytes
(8 MiB, 38 LEBs)
 UBIFS (ubi0:3): reserved for root: 0 bytes (0 KiB)
 UBIFS (ubi0:3): media format: w4/r0 (latest is w5/r0), UUID 6A93917C-50A2-4498-B372-
6F8646F5257D, small LPT model
```

- Check "userfs" volume is mounted.

```
 Board $> mount | grep ubifs
 ubi0:userfs on /media type ubifs (rw,relatime,assert=read-only,ubi=0,vol=3)
```

6 Create a default file system on a MTD partition

- Format a MTD partition (mtd3 will be used in this example).

```
 Board $> ubiformat /dev/mtd3
 ubiformat: mtd3 (nand), size 1067450368 bytes (1018.0 MiB), 4072 eraseblocks of 262144
bytes (256.0 KiB), min. I/O size 4096 bytes
 libscan: scanning eraseblock 4071 -- 100 % complete
 ubiformat: 4068 eraseblocks have valid erase counter, mean value is 1
 ubiformat: 4 bad eraseblocks found, numbers: 4068, 4069, 4070, 4071
 ubiformat: formatting eraseblock 4071 -- 100 % complete
```

- Attach mtd3 to UBI subsytem.

```
 Board $> ubiattach /dev/ubi_ctrl -m 3
 ubi0: attaching mtd3
 ubi0: scanning is finished
 ubi0: attached mtd3 (name "UBI", size 1018 MiB)
 ubi0: PEB size: 262144 bytes (256 KiB), LEB size: 253952 bytes
 ubi0: min./max. I/O unit sizes: 4096/4096, sub-page size 4096
 ubi0: VID header offset: 4096 (aligned 4096), data offset: 8192
 ubi0: good PEBs: 4068, bad PEBs: 4, corrupted PEBs: 0
 ubi0: user volume: 0, internal volumes: 1, max. volumes count: 128
 ubi0: max/mean erase counter: 4/2, WL threshold: 4096, image sequence number: 1744321690
 ubi0: available PEBs: 3988, total reserved PEBs: 80, PEBs reserved for bad PEB handling:
76
 ubi0: background thread "ubi_bgt0d" started, PID 613
 UBI device number 0, total 4068 LEBs (1033076736 bytes, 985.2 MiB), available 3988 LEBs
(1012760576 bytes, 965.8 MiB), LEB size 253952 bytes (248.0 Ki)
```

- Check the UBI status.

```
 Board $> ubinfo --all
 UBI version:                    1
 Count of UBI devices:           1
 UBI control device major/minor: 10:57
 Present UBI devices:            ubi0

 ubi0
 Volumes count:                           0
 Logical eraseblock size:                 253952 bytes, 248.0 KiB
 Total amount of logical eraseblocks:     4068 (1033076736 bytes, 985.2 MiB)
 Amount of available logical eraseblocks: 3988 (1012760576 bytes, 965.8 MiB)
 Maximum count of volumes                 128
 Count of bad physical eraseblocks:       4
 Count of reserved physical eraseblocks:  76
 Current maximum erase counter value:     6
 Minimum input/output unit size:          4096 bytes
 Character device major/minor:            240:0
```

- Create a UBI volume named "testfs" related to our partition.

```
 Board $> ubimkvol /dev/ubi0 -N testfs -m
 Set volume size to 1012760576
 Volume ID 0, size 3988 LEBs (1012760576 bytes, 965.8 MiB), LEB size 253952 bytes (248.0
KiB), dynamic, name "testfs", alignment 1
```

- Mount "testfs".

```
 Board $> mount -t ubifs ubi0:testfs /media
 UBIFS (ubi0:0): default file-system created
 UBIFS (ubi0:0): background thread "ubifs_bgt0_0" started, PID 763
 UBIFS (ubi0:0): UBIFS: mounted UBI device 0, volume 0, name "testfs"
 UBIFS (ubi0:0): LEB size: 253952 bytes (248 KiB), min./max. I/O unit sizes: 4096 bytes
/4096 bytes
 UBIFS (ubi0:0): FS size: 1009205248 bytes (962 MiB, 3974 LEBs), journal size 33521664
bytes (31 MiB, 132 LEBs)
 UBIFS (ubi0:0): reserved for root: 4952683 bytes (4836 KiB)
 UBIFS (ubi0:0): media format: w5/r0 (latest is w5/r0), UUID 66AD8600-A3F4-4EF7-9821-
0FE87D7C3DA0, small LPT model
```

- Check that the file system is empty.

```
 Board $> ls -l /media
 total 0
```

- Create a random data file.

```
 Board $> dd if=/dev/urandom of=/tmp/random.hex bs=1M count=10
 10+0 records in
 10+0 records out
 10485760 bytes (10 MB, 10 MiB) copied, 0.667978 s, 15.7 MB/s
 Board $> sync
```

- Copy the random data file in /media.

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

- Un-mount /media.

```
 Board $> umount /media
 UBIFS (ubi0:0): un-mount UBI device 0
 UBIFS (ubi0:0): background thread "ubifs_bgt0_0" stops
```

- Mount "testfs".

```
 Board $> mount -t ubifs ubi0:testfs /media
 UBIFS (ubi0:0): background thread "ubifs_bgt0_0" started, PID 800
 UBIFS (ubi0:0): UBIFS: mounted UBI device 0, volume 0, name "testfs"
 UBIFS (ubi0:0): LEB size: 253952 bytes (248 KiB), min./max. I/O unit sizes: 4096 bytes
/4096 bytes
 UBIFS (ubi0:0): FS size: 1009205248 bytes (962 MiB, 3974 LEBs), journal size 33521664
bytes (31 MiB, 132 LEBs)
 UBIFS (ubi0:0): reserved for root: 4952683 bytes (4836 KiB)
 UBIFS (ubi0:0): media format: w5/r0 (latest is w5/r0), UUID 66AD8600-A3F4-4EF7-9821-
0FE87D7C3DA0, small LPT model
```

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

```
 Board $> md5sum /tmp/random.hex /media/random.hex
 a7e4a0b65560e917805944ca04fc9695  /tmp/random.hex
 a7e4a0b65560e917805944ca04fc9695  /media/random.hex
```

- Un-mount /media.

```
 Board $> umount /media
 UBIFS (ubi0:0): un-mount UBI device 0
 UBIFS (ubi0:0): background thread "ubifs_bgt0_0" stops
```

- Detach mtd3.

```
 Board $> ubidetach -m 3
 ubi0: detaching mtd3
 ubi0: mtd3 is detached
```

7 References

Please refer to the following links for full description:

- UBIFS
- MTD UBIFS
- MTD UBI
- The kernel's command-line parameters

MultimediaCard

Memory Technology Device

Linux® is a registered trademark of Linus Torvalds.

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.

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

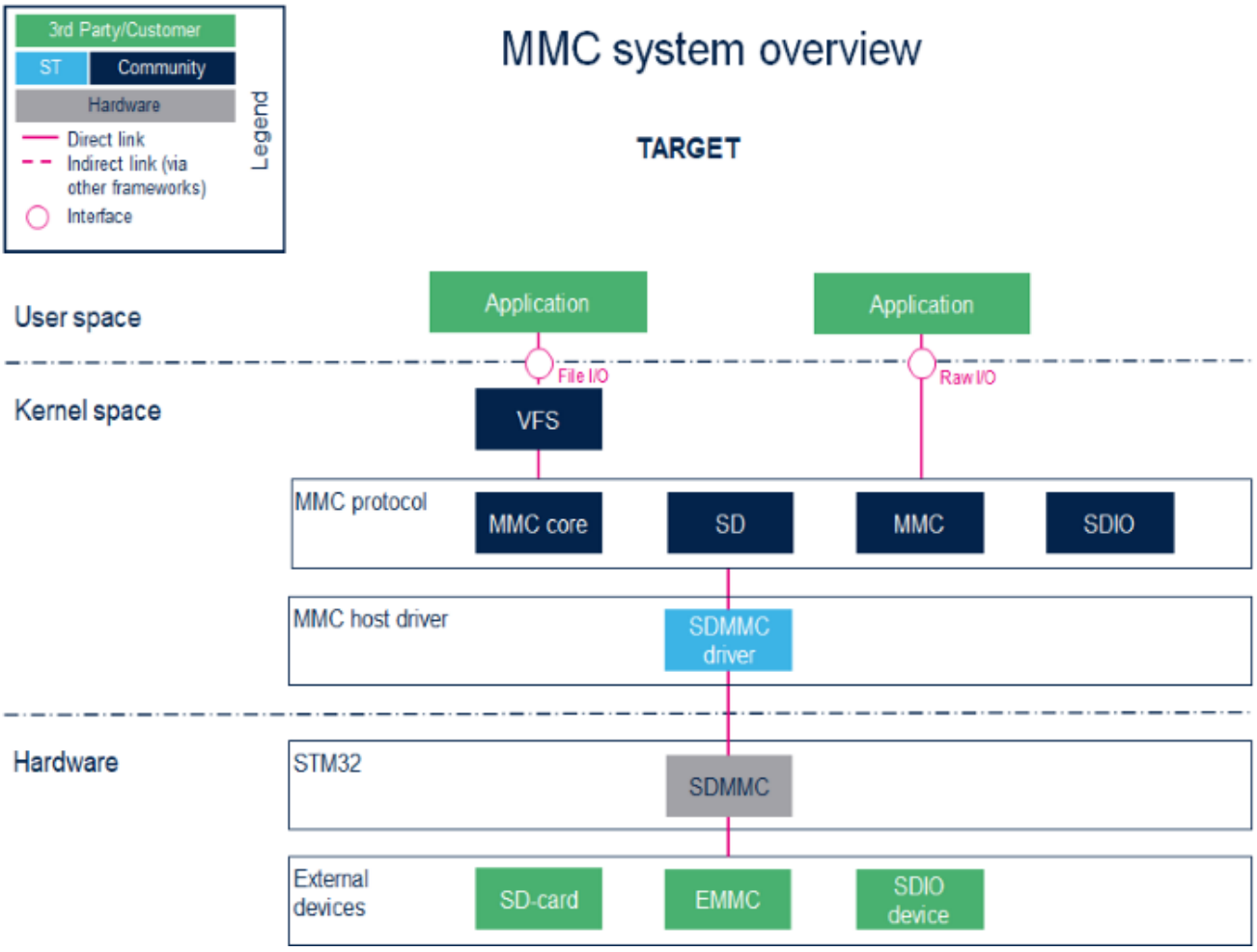
1 Framework purpose

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)

Stable: 05.11.2020 - 08:23 / Revision: 21.10.2020 - 11:37

A quality version of this page, approved on *5 November 2020*, was based off this revision.

The Linux®MTD (Memory Technology Device) subsystem provides an abstraction layer for raw Flash memories. It makes it possible to use the same API when working with different Flash types and technologies, e.g. SLC NAND, SPI NOR, ...

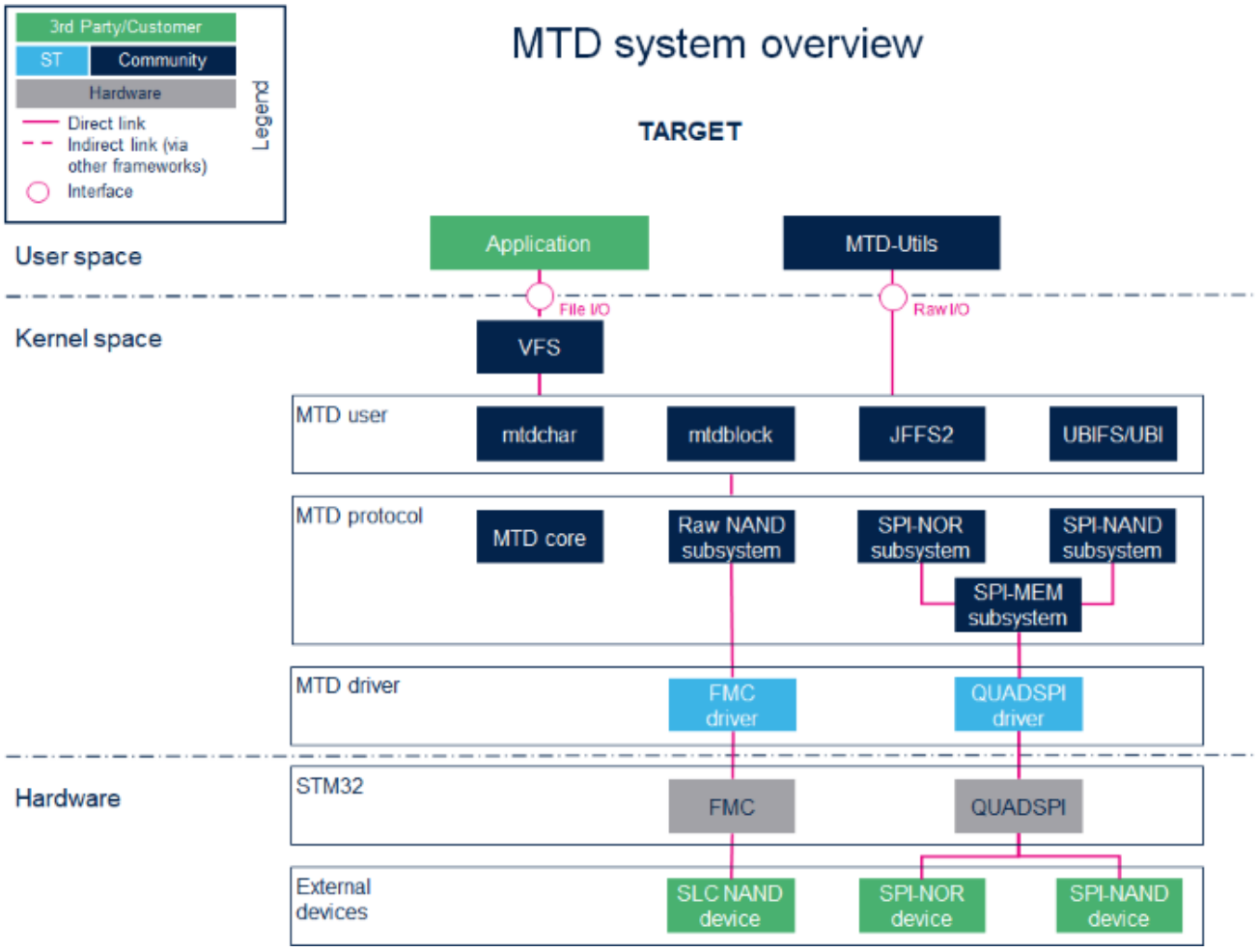
Contents

1 Framework purpose

The purpose of this article is to introduce the MTD Linux subsystem:

- General information
- Main components/stakeholders
- How to use the MTD API

2 System overview



2.1 Component description

- User space applications that perform **file I/O** need to view the Flash memory as if it was a disk, whereas programs that wish to accomplish **raw I/O** access the memory as if it was a character device.
- **VFS** (Kernel space)

Virtual File System. Please refer to the VFS documentation [1].

- **mtdchar** (Kernel space)

Usually referred to as /dev/mtdX. For MTD character devices, please refer to the MTD overview documentation [2].

- **mtdblock** (Kernel space)

Usually referred to as /dev/mtdblockX. Do not use mtdblock unless you know exactly what you are doing. For MTD block devices, please refer to the MTD block documentation [3].

- **JFFS2** (Kernel space)

Journally Flash File System. Please refer to the MTD JFFS2 documentation [4].

- **UBI** (Kernel space)

Unsorted Block Images. Please refer to the MTD UBI documentation [5].

- **UBIFS** (Kernel space)

UBI File System. Please refer to the MTD UBIFS documentation [6].

- **MTD core** (Kernel space)

The MTD core provides an abstraction layer for raw Flash memories.

- **Raw NAND subsystem** (Kernel space)

The Raw NAND protocol is used in the MTD subsystem for interfacing NAND Flash memories.

- **SPI-MEM subsystem** (Kernel space)

The SPI-MEM protocol is used in the MTD subsystem for interfacing all kinds of SPI memories (NORs, NANDs)

- **SPI-NAND subsystem** (Kernel space)

The SPI-NAND protocol is used in the MTD subsystem for interfacing SPI NAND Flash memories.

- **SPI-NOR subsystem** (Kernel space)

The SPI-NOR protocol is used in the MTD subsystem for interfacing SPI NOR Flash memories.

- **FMC driver** (Kernel space) / **FMC** (Hardware)

Please refer to the FMC internal peripheral.

- **QUADSPI driver** (Kernel space) / **QUADSPI** (Hardware)

Please refer to the QUADSPI internal peripheral.

2.2 API description

For the Linux MTD API description, please refer to the MTD API documentation [7].

3 Configuration

3.1 Kernel configuration

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

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

3.1.1 SLC NAND Flash memory

```
[*] Device Drivers --->
    <*> Memory Technology Device (MTD) support --->
        <*> RAW/Parallel NAND Device Support --->
            <*> Support for NAND controller on STM32MP Socs.
```

3.1.2 SPI NOR/NAND Flash memory

```
[*] Device Drivers --->
    <*> Memory Technology Device (MTD) support --->
        Self-contained MTD device drivers --->
            <*> Support most SPI Flash chips (AT26DF, M25P, W25X, ...)
        <*> SPI NAND device Support
        <*> SPI-NOR device support
    <*> SPI support --->
        -*- SPI memory extension
        <*> STMicroelectronics STM32 QUAD SPI controller
```

3.2 Device tree configuration

The DT configuration can be done thanks to STM32CubeMX.

3.2.1 NAND Flash memory

Please refer to the FMC device tree configuration.

3.2.2 SPI NOR/NAND Flash memory

Please refer to the QUADSPI device tree configuration.

4 How to use the framework

A file system, which handles read/write/erase operations, can be used over the MTD Framework. Please refer to the UBIFS support through MTD.

You can also interact with the MTD subsystem using the MTD utilities. The MTD utilities[8] are a set of tools that can be used to perform operations on Flash memories through the MTD character interface.

The most common utilities used are:

- mtdinfo
- flash_erase
- flashcp
- nandwrite
- nanddump

```
root:~# mtdinfo -a
Count of MTD devices:           9
Present MTD devices:            mtd0, mtd1, mtd2, mtd3, mtd4, mtd5, mtd6, mtd7, mtd8
Sysfs interface supported:      yes

mtd0
Name:                           fsbl
Type:                           nand
Eraseblock size:                262144 bytes, 256.0 KiB
Amount of eraseblocks:          8 (2097152 bytes, 2.0 MiB)
Minimum input/output unit size: 4096 bytes
Sub-page size:                  4096 bytes
OOB size:                       224 bytes
Character device major/minor:   90:0
Bad blocks are allowed:         true
Device is writable:             true

mtd1
Name:                           ssbl
Type:                           nand
Eraseblock size:                262144 bytes, 256.0 KiB
Amount of eraseblocks:          8 (2097152 bytes, 2.0 MiB)
Minimum input/output unit size: 4096 bytes
Sub-page size:                  4096 bytes
OOB size:                       224 bytes
Character device major/minor:   90:2
Bad blocks are allowed:         true
Device is writable:             true

mtd2
Name:                           UBI
Type:                           nand
Eraseblock size:                262144 bytes, 256.0 KiB
Amount of eraseblocks:          4078 (1069023232 bytes, 1019.5 MiB)
Minimum input/output unit size: 4096 bytes
Sub-page size:                  4096 bytes
OOB size:                       224 bytes
Character device major/minor:   90:4
Bad blocks are allowed:         true
Device is writable:             true

mtd3
Name:                           fsbl1
```

```
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          4 (262144 bytes, 256.0 KiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:6
Bad blocks are allowed:         false
Device is writable:             true

mtd4
Name:                           fsbl2
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          4 (262144 bytes, 256.0 KiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:8
Bad blocks are allowed:         false
Device is writable:             true

mtd5
Name:                           ssbl
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          32 (2097152 bytes, 2.0 MiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:10
Bad blocks are allowed:         false
Device is writable:             true

mtd6
Name:                           logo
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          4 (262144 bytes, 256.0 KiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:12
Bad blocks are allowed:         false
Device is writable:             true

mtd7
Name:                           nor_user
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          980 (64225280 bytes, 61.2 MiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:14
Bad blocks are allowed:         false
Device is writable:             true

mtd8
Name:                           58003000.qspi
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          1024 (67108864 bytes, 64.0 MiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:16
Bad blocks are allowed:         false
Device is writable:             true
```

5 How to trace and debug the framework

5.1 How to monitor

The sysfs interface provides detail information on each mtd device.

```
root:~# cat /sys/class/mtd/mtd0/name
fsbl
root:~# cat /sys/class/mtd/mtd0/type
nand
root:~# cat /sys/class/mtd/mtd0/erasesize
262144
root:~# cat /sys/class/mtd/mtd0/ecc_strength
8
root:~# cat /sys/class/mtd/mtd0/bad_blocks
0
root:~# cat /sys/class/mtd/mtd0/ecc_failures
0
```

5.2 How to trace

A detail dynamic trace is available here How to use the kernel dynamic debug.

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

6 Source code location

The MTD framework is here .

7 To go further

Please refer to the MTD FAQs documentation [9].

8 References

Please refer to the following links for full description:

- VFS
- MTD overview
- MTD block
- MTD JFFS2
- MTD UBI
- MTD UBIFS
- MTD API
- MTD utils
- MTD FAQs

Linux® is a registered trademark of Linus Torvalds.

Memory Technology Device

Application programming interface

Single-Level Cell is a kind of NAND flash

Serial Peripheral Interface

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.

Virtual File System

Device Tree

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

Stable: 14.05.2020 - 07:08 / Revision: 14.05.2020 - 07:07

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

Contents

1 Article purpose

This article explains how to configure the **QUADSPI** internal peripheral when it is assigned to the Linux®OS. In that case, it is controlled by the MTD framework.

The configuration is performed using the device tree mechanism that provides a hardware description of the QUADSPI peripheral, used by the STM32 QUADSPI Linux driver and by the MTD framework.

If the peripheral is assigned to another execution context, refer to How to assign an internal peripheral to a runtime context article for guidelines on peripheral assignment and configuration.

2 DT bindings documentation

The QUADSPI device tree bindings are composed by:

- generic SPI-NOR / SPI-NAND Flash memory bindings [1].
- QUADSPI driver bindings [2].

3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

3.1 DT configuration (STM32 level)

The QUADSPI peripheral node is located in *stm32mp151.dtsi*[3] file.

```
qspi: spi@58003000 {                                        Comments
    compatible = "st,stm32f469-qspi";
    reg = <0x58003000 0x1000>,                              --> Register location
          <0x70000000 0x10000000>;                          --> Memory mapping address
    reg-names = "qspi", "qspi_mm";
    interrupts = <GIC_SPI 92 IRQ_TYPE_LEVEL_HIGH>;          --> The interrupt number used
    dmas = <&mdma1 22 0x10 0x100002 0x0 0x0 0x0>,           --> DMA specifiers [4]
           <&mdma1 22 0x10 0x100008 0x0 0x0 0x0>;
    dma-names = "tx", "rx";
    clocks = <&rcc QSPI_K>;
    resets = <&rcc QSPI_R>;
    status = "disabled";
};
```

Warning

This device tree part related to the STM32 should be kept as is, the customer should not modify it.

3.2 DT configuration (board level)

The QUADSPI peripheral may connect a maximum of 2 SPI-NOR Flash memories.

SPI-NOR Flash memory nodes [1] must be children of the QUADSPI peripheral node.

```
&qspi {                                                     Comments
    pinctrl-names = "default", "sleep";                     --> For pinctrl
configuration, please refer to Pinctrl device tree configuration
    pinctrl-0 = <&qspi_clk_pins_a &qspi_bk1_pins_a &qspi_bk2_pins_a>;
    pinctrl-1 = <&qspi_clk_sleep_pins_a &qspi_bk1_sleep_pins_a &qspi_bk2_sleep_pins_a>;
    reg = <0x58003000 0x1000>,
          <0x70000000 0x4000000>;                           --> Overwrite the memory map
to the Flash device size, avoid the waste of virtual memory that will not be used
    #address-cells = <1>;
    #size-cells = <0>;
    status = "okay";                                        --> Enable the node

    flash0: mx66l51235l@0 {
        compatible = "jdec,spi-nor";
        reg = <0>;                                          --> Chip select number
        spi-rx-bus-width = <4>;                             --> The bus width (number of
data wires used)
```

```
            spi-max-frequency = <108000000>;                    --> Maximum SPI clocking
speed of device in Hz
            #address-cells = <1>;
            #size-cells = <1>;
        };
    };
```

3.3 DT configuration example

The below example shows how to configure the QUADSPI peripheral when 1 SPI-NAND Flash and 1 SPI-NOR Flash memories are connected.

```
    &qspi {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&qspi_clk_pins_a &qspi_bk1_pins_a &qspi_bk2_pins_a>;
        pinctrl-1 = <&qspi_clk_sleep_pins_a &qspi_bk1_sleep_pins_a &qspi_bk2_sleep_pins_a>;
        reg = <0x58003000 0x1000>,
              <0x70000000 0x4000000>;
        #address-cells = <1>;
        #size-cells = <0>;
        status = "okay";

        flash0: mx66l51235l@0 {
            compatible = "jdec,spi-nor";
            reg = <0>;
            spi-rx-bus-width = <4>;
            spi-max-frequency = <108000000>;
            #address-cells = <1>;
            #size-cells = <1>;
        };

        flash1: mt29f2g01abagd@1 {
            compatible = "spi-nand";
            reg = <1>;
            spi-rx-bus-width = <4>;
            spi-tx-bus-width = <4>;
            spi-max-frequency = <133000000>;
            #address-cells = <1>;
            #size-cells = <1>;
        };
    };
```

4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

5 References

Please refer to the following links for full description:

- 1.01.1 Documentation/devicetree/bindings/spi/spi-controller.yaml
- Documentation/devicetree/bindings/spi/spi-stm32-qspi.txt
- arch/arm/boot/dts/stm32mp151.dtsi
- Documentation/devicetree/bindings/dma/stm32-mdma.txt

Operating System

Memory Technology Device

Device Tree

Serial Peripheral Interface

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.

Generic Interrupt Controller

Direct Memory Access

Stable: 14.05.2020 - 07:28 Revision: 14.05.2020 - 07:27

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

Contents

1 Article purpose

This article explains how to configure the **SDMMC** internal peripheral when it is assigned to the Linux®OS. In that case, it is controlled by the MMC framework.

The configuration is performed using the device tree mechanism that provides a hardware description of the SDMMC peripheral, used by the STM32 SDMMC Linux driver and by the MMC framework.

If the peripheral is assigned to another execution context, refer to How to assign an internal peripheral to a runtime context article for guidelines on peripheral assignment and configuration.

2 DT bindings documentation

The SDMMC device tree bindings are composed of:

- generic MMC device tree bindings [1].
- SDMMC MMC/SD/SDIO interface bindings [2].

3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

3.1 DT configuration (STM32 level)

The SDMMC peripheral node is located in *stm32mp151.dtsi*[3] file.

```
   sdmmc1: sdmmc@58005000 {                                    Comments
       compatible = "arm,pl18x", "arm,primecell";
       arm,primecell-periphid = <0x00253180>;
       reg = <0x58005000 0x1000>,                              --> The controller register
location
             <0x58006000 0x1000>;                              --> The delay block register
location
       interrupts = <GIC_SPI 49 IRQ_TYPE_LEVEL_HIGH>;          --> The interrupt number used
       interrupt-names = "cmd_irq";
       clocks = <&rcc SDMMC1_K>;
       clock-names = apb_pclk
       resets = <&rcc SDMMC1_R>;
       status = "disabled";
   };
```

Warning

This device tree part is related to STM32 microprocessors. It should be kept as is, without being modified by the end-user.

3.2 DT configuration (board level)

The SDMMC peripheral may connect to one SD card, one *e*•MMC™ device or one SDIO card.

```
   &sdmmc1{                                                    Comments
       pinctrl-names = "default", "opendrain", "sleep";        --> For pinctrl
configuration, please refer to Pinctrl device tree configuration
       pinctrl-0 = <&sdmmc1_b4_pins_a &sdmmc1_dir_pins_a>;
       pinctrl-1 = <&sdmmc1_b4_od_pins_a &sdmmc1_dir_pins_a>;
       pinctrl-2 = <&sdmmc1_b4_sleep_pins_a &sdmmc1_dir_sleep_pins_a>;
       st,neg-edge;                                            --> Generate data and
command on sdmmc clock falling edge
       st,sig-dir;                                             --> Allow to select
direction polarity of an external transceiver
       st,use-ckin;                                            --> Use sdmmc_ckin pin from
an external transceiver to sample the receive data
       bus-width = <4>;                                        --> Number of data lines,
can be 1, 4 or 8
```

```
        vmmc-supply = <&vdd_sd>;                                --> Supply node for card's
power
        vqmmc-supply = <&sd_switch>;                            --> Supply node for IO line
power
        status = "okay";                                        --> Enable the node
    };
```

Below optional properties have to be used when an external transceiver is connected:

- st,sig-dir: This property allows to select external transceiver direction signals polarity. When this property is set, the voltage transceiver IOs are driven as output when the direction signals are high. Without setting this property, the voltage transceiver IOs are driven as output when the direction signals are low.
- st,use-ckin: By setting this property, the sdmmc_ckin pin from an external transceiver is used to sample the receive data.

3.3 DT configuration examples

Below example shows how to configure the SDMMC when an *e*•MMC™ is connected with 8 data lines [4].

```
    &sdmmc2{                                                    Comments
        pinctrl-names = "default", "opendrain", "sleep";
        pinctrl-0 = <&sdmmc2_b4_pins_a &sdmmc2_dir_pins_a>;
        pinctrl-1 = <&sdmmc2_b4_od_pins_a &sdmmc2_dir_pins_a>;
        pinctrl-2 = <&sdmmc2_b4_sleep_pins_a &sdmmc2_dir_sleep_pins_a>;
        non-removable;                                          --> Non-removable slot,
assume always present
        no-sd;                                                  --> Avoid to send SD command
during initialization
        no-sdio;                                                --> Avoid to send SDIO
command during initialization
        st,neg-edge;
        bus-width = <8>;
        vmmc-supply = <&v3v3>;
        vqmmc-supply = <&vdd>;
        mmc-ddr-3_3v;                                           --> Host supports e•MMC™ DDR
3.3V
        status = "okay";
    };
```

Below example shows how to configure the SDMMC to SD card (4 data lines) with an external transceiver [4].

```
    &sdmmc1{                                                    Comments
        pinctrl-names = "default", "opendrain", "sleep";
        pinctrl-0 = <&sdmmc1_b4_pins_a &sdmmc1_dir_pins_a>;
        pinctrl-1 = <&sdmmc1_b4_od_pins_a &sdmmc1_dir_pins_a>;
        pinctrl-2 = <&sdmmc1_b4_sleep_pins_a &sdmmc1_dir_sleep_pins_a>;
        broken-cd;                                              --> use polling mode for
card detection
        st,neg-edge;
        st,sig-dir;
        st,use-ckin;
        bus-width = <4>;
        sd-uhs-sdr12;                                           --> sd modes supported [1]
        sd-uhs-sdr25;
        sd-uhs-sdr50;
```

```
        sd-uhs-ddr50;
        sd-uhs-sdr104;
        vmmc-supply = <&vdd_sd>;
        vqmmc-supply = <&sd_switch>;
        status = "okay";
    };
```

4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

5 References

Please refer to the following links for additional information:

- 1.01.1 Documentation/devicetree/bindings/mmc/mmc-controller.yaml
- Documentation/devicetree/bindings/mmc/mmci.txt
- arch/arm/boot/dts/stm32mp151.dtsi
- 4.04.1 arch/arm/boot/dts/stm32mp15xx-edx.dtsi

Operating System

MultimediaCard

Device Tree

Secure digital

Generic Interrupt Controller

Serial Peripheral Interface

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

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

input/output

Doubledata rate (memory domain)