SDMMC device tree configuration

Contents

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

Stable: 14.05.2020 - 07:27 / 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";
   };
```

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

Linux® is a registered trademark of Linus Torvalds.

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)

Stable: 19.03.2021 - 08:52 / Revision: 19.03.2021 - 08:49

A quality version of this page, approved on *19 March 2021*, was based off this revision.

Contents

1 Purpose

The objective of this chapter is to give general information about the device tree.
An extract of the **device tree specification**[1] explains it as follows:

"A device tree is a tree data structure with nodes that describe the devices in a system. Each node has property/value pairs that describe the characteristics of the device being represented. Each node has exactly one parent except for the root node, which has no parent. ... Rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time."

In other words, a device tree describes the hardware that can not be located by probing. For more information, please refer to the device tree specification[1]

1.1 Source files

- **.dts:** The device tree source (DTS). This format is a textual representation of a device tree in a form that can be processed by DTC (Device Tree Compiler) into a binary device tree in the form expected by software components: Linux® Kernel, U-Boot and TF-A.

- **.dtsi:** Source files that can be included from a DTS file.

1.2 Bindings

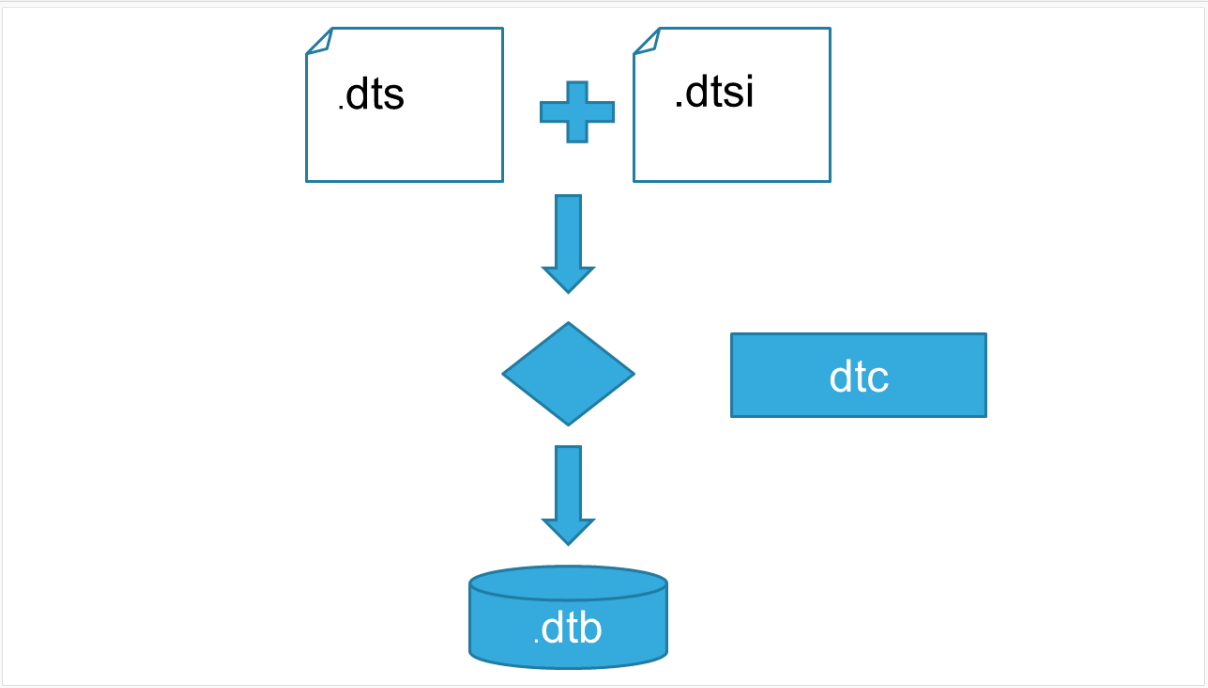
The device tree data structures and properties are named **bindings**. Those bindings are described in:

- The Device tree specification[1] for generic bindings.
- The software component documentations:
  - Linux® Kernel: Linux kernel device tree bindings
  - U-Boot: doc/device-tree-bindings/
  - TF-A: TF-A device tree bindings

1.3 Build

- A tool named DTC (Device Tree Compiler) allows compiling the DTS sources into a binary.
- input file: the *.dts* file described in section above.
- output file: the *.dtb* file described in section above.
- More information are available in DTC manual[2].

- DTC source code is located here[3]. DTC tool is also available directly in particular software components: **Linux Kernel, U-Boot, TF-A ...**. For those components, the device tree building is directly integrated in the component build process.



Device tree build process

If dts files use some defines, dts files should be preprocessed before being compiled by DTC.

1.4 Tools

The device tree compiler offers also some tools:

- **fdtdump:** Print a readable version of a flattened device tree file (dtb)
- **fdtget:** Read properties from a device tree
- **fdtput:** Write properties to a device tree
- ...

There are several ways to get those tools:

- In the device tree compiler project source code[3]
- Directly in software components: **Kernel, u-boot, tf-a ...**
- Available in Debian package[4]

2 STM32

For STM32MP1, the device tree is used by three software components: Linux® kernel, U-Boot and TF-A.

The device tree is part of the OpenSTLinux distribution. It can also be generated by STM32CubeMX tool.

To have more information about the device tree usage on STM32MP1 (how the device tree source files are split, how to find the device tree source files per software components, how is STM32CubeMX generating the device tree ...) see STM32MP15 device tree page.

3 How to go further

- Device Tree for STM32MP [5]
- Device Tree Reference[6] - eLinux.org
- Device Tree usage[7] - eLinux.org

4 References

- 1.01.11.2 https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.2(latest) ,Device tree specification
- https://git.kernel.org/pub/scm/utils/dtc/dtc.git/tree/Documentation/manual.txt(master) ,DTC manual
- 3.03.1 https://git.kernel.org/pub/scm/utils/dtc/dtc.git(master) ,DTC source code
- https://packages.debian.org/search?keywords=device-tree-compiler(master) ,DTC debian package
- https://www.youtube.com/watch?v=a9CZ1Uk3OYQ, Device Tree for STM32MP
- Device Tree Reference, eLinux.org
- Device Tree Usage, eLinux.org

Device Tree Source (in software context) or Digital Temperature Sensor (in peripheral context)

Linux® is a registered trademark of Linus Torvalds.

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

Trusted Firmware for Arm® Cortex®-A

Stable: 08.03.2021 - 16:13 / Revision: 16.02.2021 - 17:11

A quality version of this page, approved on *8 March 2021*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the software that assigns a peripheral to a runtime context.

2 Introduction

A peripheral can be **assigned** to a runtime context via the configuration defined in the device tree. The device tree can be either generated by the STM32CubeMX tool or edited manually.

On STM32MP15 line devices, the assignement can be strengthened by a hardware mechanism: the ETZPC internal peripheral, which is configured by the TF-A boot loader. The ETZPC internal peripheral isolates the peripherals for the Cortex-A7 secure or the Cortex-M4 context. The peripherals assigned to the Cortex-A7 non-secure context are visible from any context, without any isolation.

The components running on the platform after TF-A execution (such as U-Boot, Linux, STM32Cube and OP-TEE) must have a **configuration** that is consistent with the assignment and the isolation configurations.

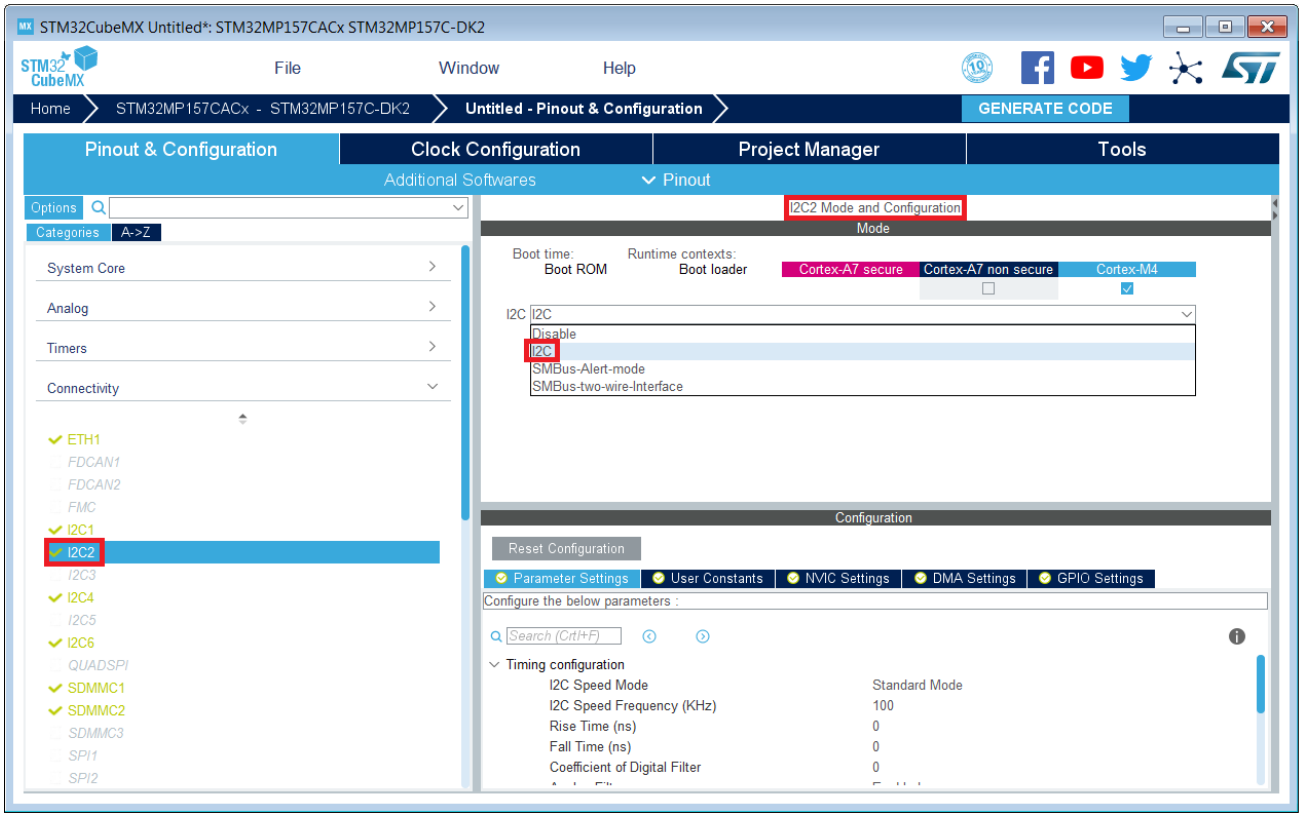
The following sections describe how to configure TF-A, U-Boot, Linux and STM32Cube accordingly.

Beyond the peripherals assignment, explained in this article, it is also important to understand How to configure system resources (i.e clocks, regulator, gpio,...), shared between the Cortex-A7 and Cortex-M4 contexts

3 STM32CubeMX generated assignment

The screenshot below shows the STM32CubeMX user interface:

- **I2C2** peripheral is selected, on the left
- **I2C2 Mode and Configuration** panel, on the right, shows that this I2C instance can be assigned to the Cortex-A7 non-secure or the Cortex-M4 (that is selected) runtime context
- **I2C** mode is enabled in the drop down menu



The context assignment table is displayed inside each peripheral **Mode and Configuration** panel but it is possible to display it for all the peripherals in the **Options** menu via the **Show contexts** option

The **GENERATE CODE** button, on the top right, produces the following:

- The **TF-A device tree** with the ETZPC configuration that isolates the I2C2 instance (in the example) for the Cortex-M4 context. This same device tree can be used by **OP-TEE**, when enabled
- The **U-Boot device tree** widely inherited from the Linux one, just below
- The **Linux kernel device tree** with the I2C node disabled for Linux and enabled for the coprocessor
- The **STM32Cube project** with I2C2 HAL initialization code

The Manual assignment section, just below, illustrates what STM32CubeMX is generating as it follows the same example.

In addition of this generation, the user may have to manually complete the system resources

configuration in the user sections embedded in the STM32CubeMX generated device tree. Refer to How to configure system resources for details.

4 Manual assignment

This section gives step by step instructions, per software components, to manually perform the peripherals assignments.
It takes the same I2C2 example as the previous section, that showed how to use STM32CubeMX, in order to make the move from one approach to the other easier.

> The assignments combinations described in the STM32MP15 peripherals overview article are naturally supported by STM32MPU Embedded Software distribution. Note that the STM32MP15 reference manual may describe more options that would require embedded software adaptations

4.1 TF-A

The assignment follows the ETZPC device tree configuration, with below possible values:

- **DECPROT_S_RW** for the Cortex-A7 secure (Secure OS like OP-TEE)
- **DECPROT_NS_RW** for the Cortex-A7 non-secure (Linux)
  - As stated earlier in this article, there is no hardware isolation for the Cortex-A7 non-secure so this value allows accesses from any context
- **DECPROT_MCU_ISOLATION** for the Cortex-M4 (STM32Cube)

Example:

```
@etzpc: etzpc@5C007000 {
        st,decprot = <
            DECPROT(STM32MP1_ETZPC_I2C2_ID, DECPROT_MCU_ISOLATION, DECPROT_UNLOCK)
        >;
};
```

> The value **DECPROT_NS_RW** can be used with **DECPROT_LOCK** as last parameter. In Cortex-M4 context, this specific configuration allows the generation of an error in the resource manager utility while trying to use on Cortex-M4 side a peripheral that is assigned to the Cortex-A7 non-secure context. If **DECPROT_UNLOCK** is used, then the utility allows the Cortex-M4 to use a peripheral that is assigned to the Cortex-A7 non-secure context.

4.2 U-boot

No specific configuration is needed in U-Boot to configure the access to the peripheral.

> U-Boot does not perform any check with regards to ETZPC configuration before accessing to a peripheral. In case of inconsistency an illegal access is generated.

> U-Boot checks the consistency between ETZPC isolation configuration and Linux kernel device tree configuration to guarantee that Linux kernel do not access an unauthorized device. In order to avoid the access to an unauthorized device, the U-boot fixes up the Linux

kernel device tree to disable the peripheral nodes which are not assigned to the Cortex-A7 non-secure context.

4.3 Linux kernel

Each assignable peripheral is declared twice in the Linux kernel device tree:

- Once in the **soc** node from arch/arm/boot/dts/stm32mp151.dtsi , corresponding to Linux assigned peripherals
  - Example: i2c2
- Once in the **m4_rproc** node from arch/arm/boot/dts/stm32mp15-m4-srm.dtsi , corresponding to the Cortex-M4 context.

Those nodes are disabled, by default.

  - Example: m4_i2c2

In the board device tree file (*.dts), each assignable peripheral has to be enabled only for the context to which it is assigned, in line with TF-A configuration.
As a consequence, a peripheral assigned to the Cortex-A7 secure has both nodes disabled in the Linux device tree.

Example:

```
&i2c2 {
        status = "disabled";
};
...
&m4_i2c2 {
        status = "okay";
};
```

In addition of this assignment, the user may have to complete the system resources configuration in the device tree nodes. Refer to How to configure system resources for details.

4.4 STM32Cube

There is no configuration to do on STM32Cube side regarding the assignment and isolation. Nevertheless, the resource manager utility, relying on ETZPC configuration, can be used to check that the corresponding peripheral is well assigned to the Cortex-M4 before using it.

Example:

```
int main(void)
{
...
 /* Initialize I2C2-------------------------------------------------------- */
 /* Ask the resource manager for the I2C2 resource */
 ResMgr_Init(NULL, NULL);
 if (ResMgr_Request(RESMGR_ID_I2C2, RESMGR_FLAGS_ACCESS_NORMAL | \
                                    RESMGR_FLAGS_CPU1, 0, NULL) != RESMGR_OK)
 {
   Error_Handler();
 }
```

```
...
 if (HAL_I2C_Init(&I2C2) != HAL_OK)
 {
   Error_Handler();
 }
}
```

4.5 OP-TEE

The OP-TEE OS may use STM32MP1 resources. OP-TEE STM32MP1 drivers register the device driver they intend to used in a secure context. This information is used to consolidate system configuration including secure hardening of configurable peripherals.

In most case, the OP-TEE driver probe relies on OP-TEE device tree porperty *secure-status = "okay"*.

Cortex®

Trusted Firmware for Arm® Cortex®-A

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

Linux® is a registered trademark of Linus Torvalds.

Inter-Integrated Circuit (Bi-directional 2-wire bus standard for efficient inter-IC control.)

Open Portable Trusted Execution Environment

Hardware Abstraction Layer

Operating System

Microcontroller Unit (MCUs have internal flash memory and are intended to operate with a minimum amount of external support ICs. They commonly are a self-contained, system-on-chip (SoC) designs.)

Extended TrustZone Protection Controller

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

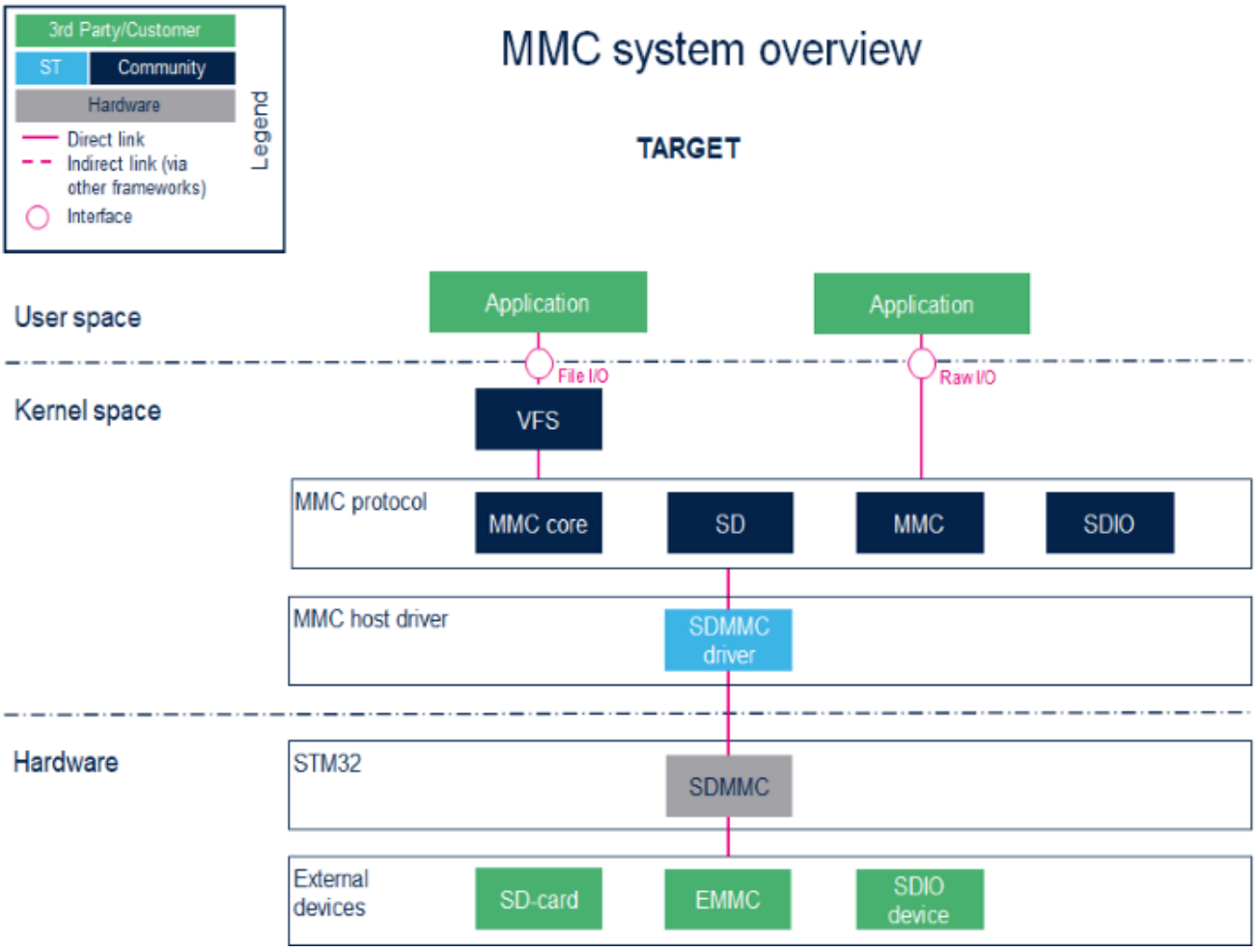
1 Framework purpose

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: 06.09.2021 - 16:08 / Revision: 06.09.2021 - 16:03

A quality version of this page, approved on *6 September 2021*, was based off this revision.

Contents

1 Purpose

The purpose of this article is to explain how to configure the GPIO internal peripheral through **the pin controller (pinctrl) framework**, **when this peripheral is assigned to Linux®OS (Cortex-A)**. The configuration is performed using the Device tree.

To better understand I/O management, it is recommended to read the Overview of GPIO pins article.

This article also provides an example explaining how to add a new pin in the device tree.

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 Pinctrl device tree bindings are composed of:

- generic DT bindings[1] used by the pinctrl framework.
- vendor pinctrl DT bindings[2] used by the stm32-pinctrl driver: this binding document explains how to write device tree files for pinctrl.

3 DT configuration

3.1 DT configuration (STM32 level)

The pin controller node is composed of several parts:

3.1.1 STM32 pin controller information

The pin controller node is located in the SOC dtsi file *stm32mp151.dtsi*[3].

```
pinctrl: pin-controller@50002000 {                Comments
         #address-cells = <1>;
         #size-cells = <1>;
         ranges = <0 0x50002000 0xa400>;     -->Provides IP start address and memory map
device size
         interrupt-parent = <&exti>;         -->Provides interrupt parent controller (used
when the GPIO is configured as an external interrupt)
         st,syscfg = <&exti 0x60 0xff>;      -->Provides phandle for IRQ mux selection
         pins-are-numbered;

        ...
};
```

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

3.1.2 GPIO bank information

The GPIO bank information nodes are located in the SOC dtsi file *stm32mp151.dtsi*[3].

```
pinctrl: pin-controller@50002000 {                Comments
         ...

        gpioa: gpio@50002000 {
                gpio-controller;
                #gpio-cells = <2>;
                interrupt-controller;      -->Indicates that this GPIO bank can be used
as interrupt controller
                #interrupt-cells = <2>;
                reg = <0x0 0x400>;         -->Provides offset in pinctrl addres map for
the GPIO bank
                clocks = <&rcc GPIOA>;     -->phandle on GPIO bank clock
                st,bank-name = "GPIOA";
                status = "disabled";
        };

        gpiob: gpio@50003000 {
                gpio-controller;
                #gpio-cells = <2>;
                interrupt-controller;
                #interrupt-cells = <2>;
                reg = <0x1000 0x400>;
                clocks = <&rcc GPIOB>;
                st,bank-name = "GPIOB";
```

```
                status = "disabled";
        };

        ...
};
```

The GPIO bank definition is completed by the pinctrl package dtsi files, as, for example *stm32mp15xxaa-pinctrl.dtsi*[4].

```
&pinctrl {
        st,package = <STM32MP_PKG_AA>;

        gpioa: gpio@50002000 {
                status = "okay";
                ngpios = <16>;
                gpio-ranges = <&pinctrl 0 0 16>;
        };

        gpiob: gpio@50003000 {
                status = "okay";
                ngpios = <16>;
                gpio-ranges = <&pinctrl 0 16 16>;
        };
```

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

3.1.3 Pin state definition

The pin states are defined in the pinctrl dtsi file *stm32mp15-pinctrl.dtsi* [5]

```
&pinctrl {
        ...
         usart3_pins_a: usart3@0 {                                                  Comme
nts
                pins1 {
                        pinmux = <STM32_PINMUX('B', 10, AF7)>, /* USART3_TX */      --
>Pin muxing information: AF7 (alternate function 7) selected on PB10 pin
                                 <STM32_PINMUX('G', 8, AF8)>; /* USART3_RTS */      --
>Pin muxing information: AF8 (alternate function 8) selected on PG8 pin
                        bias-disable;                                               --
>Generic bindings corresponding to "no pull-up" and "no pull-down"
                        drive-push-pull;                                            --
>Generic bindings to select pin driving information
                        slew-rate = <0>;                                            --
>Generic bindings to select pin speed
                };
                pins2 {
                        pinmux = <STM32_PINMUX('B', 12, AF8)>, /* USART3_RX */
                                 <STM32_PINMUX('I', 10, AF8)>; /* USART3_CTS_NSS */
                        bias-disable;
                };
       };
       ...
};
```

- Refer to GPIO internal peripheral for more details on hardware pin configuration.

3.2 DT configuration (board level)

As seen in Pin controller configuration (pin state definition part), all pin states are defined inside the pin controller node.

Each device that requires pins has to select the desired pin state phandle inside the board device tree file (see Device tree for more explanations about device tree file split).

The STM32MP1 devices feature a lot of possible pin combinations for a given internal peripheral. From one board to another, different sets of pins can consequently be used for an internal peripheral. Note that "_a", "_b" suffixes are used to identify pin muxing combinations in the device tree pinctrl file. The right suffixed combination must then be used in the device tree board file.

- Example:

```
&usart3 {
        ...
        pinctrl-names = "default","sleep";
        pinctrl-0 = <&usart3_pins_a>;
        pinctrl-1 = <&usart3_sleep_pins_a>;
        ...
};
```

3.3 DT configuration examples

3.3.1 How to add new pin states

To add new pin states and affect them to a foo_device, proceed as follows:

1. Find the pins you need:

   In the example below, the foo_device needs to configure PC13, PG8 and PI2.

   AF2 is selected as alternate function on PC13, and AF5 on PG8 and PI2.

   Each pin requires an internal pull-up.

2. Write your pin state phandle in *stm32mp15-pinctrl.dtsi*.

```
 &pinctrl {
         ...
          foo_pins_a: foo@0 {
                 pins {
                         pinmux = <STM32_PINMUX('C', 13, AF2)>,
                                  <STM32_PINMUX('G', 8, AF5)>,
                                   <STM32_PINMUX('I', 2, AF5)>;
                         bias-pull-up;
                 };
        };
        ...
 };
```

   All the possible settings are described in GPIO internal peripheral.

3. Select the pin state phandle required for your device in the board file.

```
&foo {
        ...
        pinctrl-names = "default";
        pinctrl-0 = <&foo_pins_a>;
        ...
};
```

4 How to configure GPIOs 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:

- Documentation/devicetree/bindings/pinctrl/pinctrl-bindings.txt , Generic pinctrl device tree bindings
- Documentation/devicetree/bindings/pinctrl/st,stm32-pinctrl.yaml , STM32 pinctrl device tree bindings
- 3.03.1 stm32mp151.dtsi STM32MP15 SOC device tree file
- stm32mp15xxaa-pinctrl.dtsi STM32MP15 Pinctrl device tree file
- stm32mp15-pinctrl.dtsi STM32MP15 pinctrl device tree file

Linux® is a registered trademark of Linus Torvalds.

Operating System

Cortex®

Device Tree

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Transmit

Receive

Compatibility Test Suite (Android specific) or Clear to send (in UART context)

Stable: 14.05.2020 - 07:13 / Revision: 14.05.2020 - 07:12

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

Contents

1 Article purpose

The purpose of this article is to

- briefly introduce the SDMMC peripheral and its main features
- indicate the level of security supported by this hardware block
- explain how each instance can be allocated to the three runtime contexts and linked to the corresponding software components
- explain, when necessary, how to configure the SDMMC peripheral.

2 Peripheral overview

The **SDMMC** peripheral is used to interconnect STM32 MPU to SD memory cards, SDIO and MMC devices.

2.1 Features

Refer to the STM32MP15 reference manuals for the complete list of features, and to the software components, introduced below, to see which features are implemented.

2.2 Security support

SDMMC1/2/3 instances are either **non-secure** or **secure** peripherals (under ETZPC control).

- **When an SDMMC instance is secure internal, the DMA cannot be used to perform data transfers.**
- **STMicroelectronics does not provide secure MMC driver (see below chapter)**

3 Peripheral usage and associated software

3.1 Boot time

SDMMC1/2 instances can be used to support memory boot on SD or MMC Flash devices.

The SDMMC3 is not used at boot time.

The SDMMC instances are ordered by address in the device tree arch/arm/boot/dts /stm32mp151.dtsi file:

```
sdmmc3: sdmmc@48004000 {
...
sdmmc1: sdmmc@58005000 {
...
sdmmc2: sdmmc@58007000 {
```

By default, in OpenSTLinux distribution, **sdmmc3 is disabled** so the sdmmc1 (SD card on Evaluation boards and Discovery kits) and sdmmc2 (eMMC on Evaluation boards and Wifi on Discovery kits) are respectively aliased to mmc0 and mmc1.
If you enable sdmmc3, it will take the mmc0 alias and the aliases above will shift, so don't forget to update the Linux kernel boot command accordingly!
For instance, 'root=/dev/mmcblk0p6' will become 'root=/dev/mmcblk1p6' to mount the rootfs from the sdmmc1 (SD card) when sdmmc3 is enabled.

3.2 Runtime

3.2.1 Overview

SDMMC1/2/3 instances can be allocated to:

- the Arm®Cortex®-A7 non-secure core to be controlled in Linux® by the MMC framework

or

- the Arm®Cortex®-M4 to be controlled in STM32Cube MPU Package by STM32Cube SDMMC driver

Chapter #Peripheral assignment describes which peripheral instance can be assigned to which context.

3.2.2 Software frameworks

Domain	Peripheral	Software components			Comment
OP-TEE	Linux	STM32Cube			
Mass storage	SDMMC		Linux MMC framework	STM32Cube SDMMC driver	

3.2.3 Peripheral configuration

The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in the corresponding software framework article.

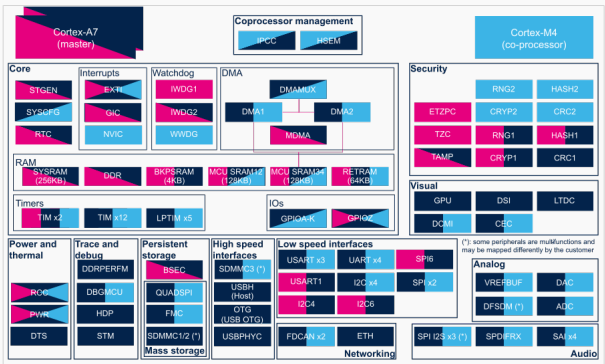
For Linux® kernel configuration, please refer to SDMMC device tree configuration.

3.2.4 Peripheral assignment

Check boxes illustrate the possible peripheral allocations supported by STM32 MPU Embedded Software:

- means that the peripheral can be assigned () to the given runtime context.
- is used for system peripherals that cannot be unchecked because they are statically connected in the device.

Refer to How to assign an internal peripheral to a runtime context for more information on how to assign peripherals manually or via STM32CubeMX.
The present chapter describes STMicroelectronics recommendations or choice of implementation. Additional possiblities might be described in STM32MP15 reference manuals .



Internal peripherals

Domain	Peripheral	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Mass storage	SDMMC	SDMMC1				
		SDMMC2				
		SDMMC3				Assignment (single choice)

4 How to go further

5 References

Microprocessor Unit

MultimediaCard

Direct Memory Access

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

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

Linux® is a registered trademark of Linus Torvalds.

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

Cortex®

Open Portable Trusted Execution Environment

Stable: 23.09.2020 - 13:22 / Revision: 12.06.2020 - 13:25

A quality version of this page, approved on *23 September 2020*, was based off this revision.

1 STM32CubeMX overview

This article describes STM32CubeMX, an official STMicroelectronics graphical software configuration tool.

The STM32CubeMX application helps developers to use the STM32 by means of a user interface, and guides the user through to the initial configuration of a firmware project.

It provides the means to:

- configure pin assignments, the clock tree, or internal peripherals
- simulate the power consumption of the resulting project
- configure and tune DDR parameters
- generate HAL initialization code for Cortex-M4
- generate the Device Tree for a Linux kernel, TF-A and U-Boot firmware for Cortex-A7

It uses a rich library of data from the STM32 microcontroller portfolio.

The application is intended to ease the initial development phase by helping developers to select the best product in terms of features and power.

2 STM32CubeMX main features

- Peripheral and middleware parameters

  Presents options specific to each supported software component

- Peripheral assignment to processors

  Allows assignment of each peripheral to Cortex-A Secure, Cortex-A Non-Secure, or Cortex-M processors

- Power consumption calculator

  Uses a database of typical values to estimate power consumption, DMIPS, and battery life

- Code generation

  Makes code regeneration possible, while keeping user code intact

- Pinout configuration

  Enables peripherals to be chosen for use, and assigns GPIO and alternate functions to pins

- Clock tree initialization

  Chooses the oscillator and sets the PLL and clock dividers

- DDR tunning tool

  Ensures the configuration, testing, and tuning of the MPU DDR parameters. Using U-Boot-SPL Embedded Software.

3 How to get STM32CubeMX

Please, refer to the following link STM32CubeMX to find STM32CubeMX, the Release Note, the User Manual and the product specification.

Doubledata rate (memory domain)

Hardware Abstraction Layer

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Trusted Firmware for Arm® Cortex®-A

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

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Microprocessor Unit