Category:DMA peripherals

Contents

Stable: 17.06.2020 - 15:26 / Revision: 16.01.2020 - 12:52

Stable: 17.06.2020 - 10:20 | Revision: 16.01.2020 - 12:32

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 **DMA** internal peripherals (hardware blocks) embedded in the STM32 MPUs microprocessor devices.

Pages in category "DMA peripherals"

The following 3 pages are in this category, out of 3 total.

- DMA internal peripheral
- DMAMUX internal peripheral
- MDMA internal peripheral

Stable: 13.10.2020 - 08:29 / Revision: 13.10.2020 - 08:29

A quality version of this page, approved on *13 October 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the DMA 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 DMA peripheral.

2 Peripheral overview

The **DMA** peripheral is used to perform direct accesses from/to a device or a memory. Each DMA instance supports 8 channels. The selection of the device connected to each DMA channel and controlling the DMA transfers is done via the DMAMUX.

Note: Directly accessing DDR from the DMA is not recommended for high-bandwith or latency-critical transfers. This means that DMA transfers configured by the Arm® Cortex®-A7 operating system, that usually target buffers in external memory, require a hardware mechanism to chain the DMA and a MDMA channel in order to achieve the following flow:

DDR<-> MDMA <-> MCU SRAM <-> DMA <-> device

This feature was already present on STM32H7 microcontroller Series. It is documented in application note AN5001[1].

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

The DMA is a **non-secure** peripheral.

3 Peripheral usage and associated software

3.1 Boot time

The DMA is not used at boot time.

3.2 Runtime

3.2.1 Overview

Each DMA instance can be allocated to:

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

or

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

3.2.2 Software frameworks

Domain	Peripheral	Software components			Comment
OP-TEE	Linux	STM32Cube			
Core/DMA	DMA		Linux dmaengine framework	STM32Cube DMA driver	

3.2.3 Peripheral configuration

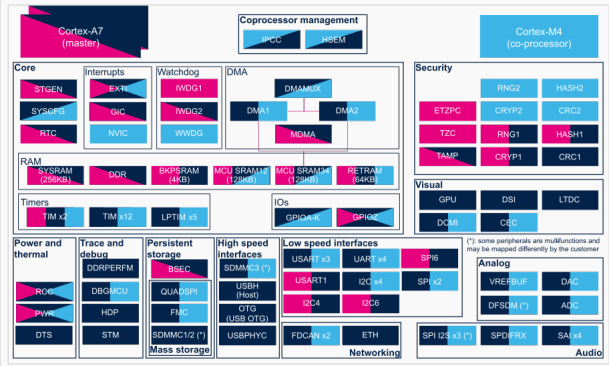
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in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

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	Periphera l	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core/DMA	DMA	DMA1				Assignment (single choice)
		DMA2				Assignment (single choice)

4 References

- http://www.st.com/resource/en/application_note/dm00360392.pdf

Stable: 04.02.2020 - 15:42 / Revision: 04.02.2020 - 15:34

A quality version of this page, approved on *4 February 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the DMAMUX 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 DMAMUX peripheral.

2 Peripheral overview

The **DMAMUX** peripheral is used to perform requestor line (or device controller) selection for each channel from DMA instances: there is a single DMAMUX instance to cover both DMA1 and DMA2.

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

The DMAMUX is a **non secure** peripheral.

3 Peripheral usage and associated software

3.1 Boot time

The DMAMUX is not used at boot time.

3.2 Runtime

3.2.1 Overview

The DMAMUX manages DMA1 and DMA2 requestor line selection via different registers so it is possible to concurrently access to DMAMUX from Cortex®-A7 non-secure and Cortex®-M4 contexts, as far as each core is only configuring the requestor lines for the DMA instances (DMA1 and/or DMA2) assigned to itself.

Finally, DMAMUX can be allocated to:

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

or

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

3.2.2 Software frameworks

Domain	Peripheral	Software components			Comment
OP-TEE	Linux	STM32Cube			
Core/DMA	DMAMUX		Linux dmaengine framework	STM32Cube DMAMUX driver	

3.2.3 Peripheral configuration

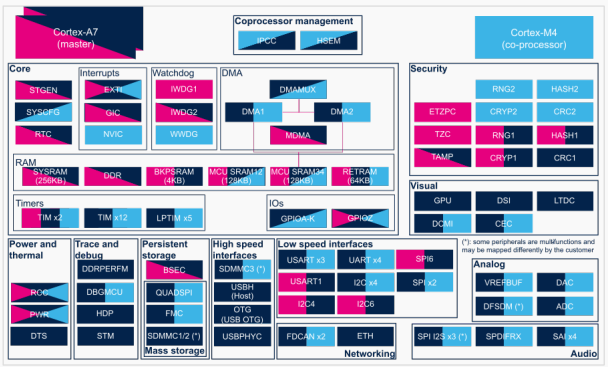
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in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

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	Periphera l	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core/DMA	DMAMUX	DMAMUX				Shareable (multiple choices supported)

Stable: 13.10.2020 - 08:31 / Revision: 13.10.2020 - 08:31

A quality version of this page, approved on *13 October 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the MDMA 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 MDMA peripheral.

2 Peripheral overview

The **MDMA** is used to perform high-speed data transfers between memory and memory or between peripherals and memory. The MDMA controller offers 32 channels. The selection of the device connected to each channel and controlling DMA transfers is done in MDMA peripheral.

Among all the requestor lines described in the STM32MP15 reference manuals, DMA channels are the only lines that allow to perform transfers with chained DMA and MDMA (refer to DMA internal peripheral article). As a result, when a device is not connected to the MDMA, it is anyway possible to operate in DMA mode via the DMA controller and chain DMA and MDMA.

2.1 Features

Refer to 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

The MDMA is a **secure** peripheral. This means that it performs each transfer in the context of the master that requested it:

- a transfer requested by the Arm® Cortex®-A7 **non-secure** core propagates **non-secure accesses** to the targeted device and /or memory.
- a transfer requested by Arm Cortex-A7 **secure** core propagates **secure accesses** to the targeted device and/or memory.

3 Peripheral usage and associated software

3.1 Boot time

The MDMA is used at boot time by the FMC.

3.2 Runtime

3.2.1 Overview

The MDMA is visible from the Arm Cortex-M4 core. However, it is not supported in this context by STM32MPU Embedded Software distribution.

As stated in the 'Security support' chapter above, the MDMA is a secure peripheral. This means that its channels have to be allocated to:

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

and

- the Arm Cortex-A7 secure core to be controlled by the MDMA OP-TEE driver

STM32CubeMX allows to distinguish between non-secure and secure channels, among all the available channels.

3.2.2 Software frameworks

Domain	Peripheral	Software components		Comment
OP-TEE	Linux	STM32Cube		
Core/DMA	MDMA	OP-TEE MDMA driver	Linux dmaengine framework	

3.2.3 Peripheral configuration

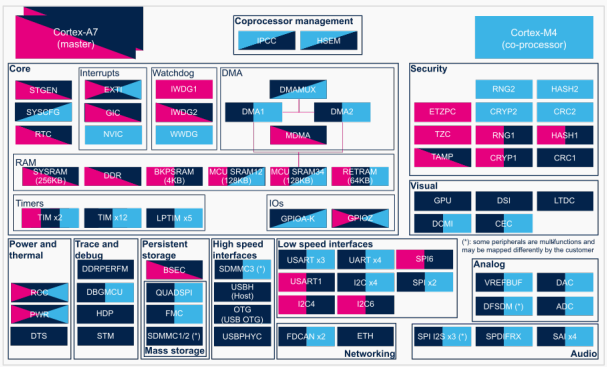
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in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

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	Periphera l	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core/DMA	MDMA	MDMA				Shareable (multiple choices supported)

4 How to go further

Not applicable

5 References