DMA internal peripheral

Stable: 11.02.2019 - 13:21 / Revision: 18.01.2019 - 18:05

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 MCU SRAM, 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

DMA instances 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 frameworks			Comment
		Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)	
Core/DMA	DMA		Linux dmaengine framework	STM32Cube DMA driver	

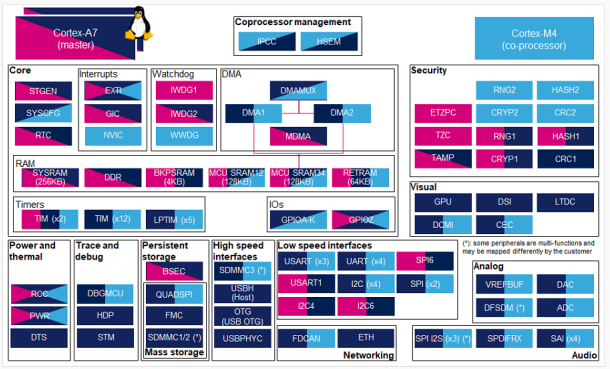
3.2.3 Peripheral configuration

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.



Internal peripherals

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.

Domain	Peripheral	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

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

Direct Memory Access

Doubledata rate (memory domain)

Microcontroller Unit

Microprocessor Unit

Open Portable Trusted Execution Environment