



DMA internal peripheral



Contents

1. DMA internal peripheral	3
2. DDRCTRL and DDRPHYC internal peripherals	8
3. DMAMUX internal peripheral	13
4. Dmaengine overview	18
5. How to assign an internal peripheral to a runtime context	23
6. MDMA internal peripheral	28
7. STM32CubeMP1 architecture	33
8. STM32CubeMX	38
9. STM32MP15 resources	43
10. STM32MPU Embedded Software architecture overview	48



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

Contents

1 Article purpose	4
2 Peripheral overview	5
2.1 Features	5
2.2 Security support	5
3 Peripheral usage and associated software	6
3.1 Boot time	6
3.2 Runtime	6
3.2.1 Overview	6
3.2.2 Software frameworks	6
3.2.3 Peripheral configuration	6
3.2.4 Peripheral assignment	6
4 References	8



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-bandwidth 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.

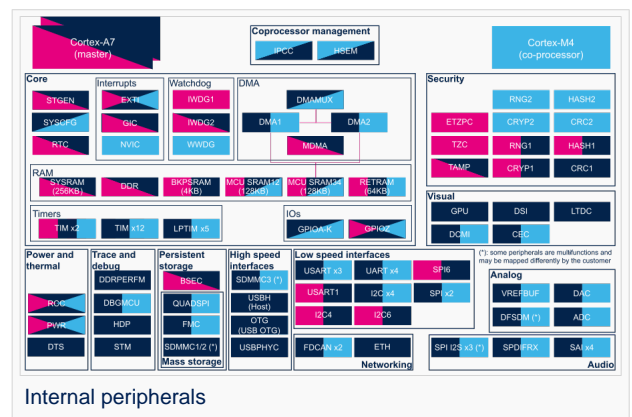
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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



Cortex[®]

Doubledata rate (memory domain)

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.)

Linux[®] is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

Stable: 02.06.2021 - 07:02 / Revision: 23.06.2021 - 15:20

Contents

1 Article purpose	9
2 Peripheral overview	10
2.1 Features	10
2.2 Security support	10
3 Peripheral usage and associated software	11
3.1 Boot time	11
3.2 Runtime	11
3.2.1 Overview	11
3.2.2 Software frameworks	11
3.2.3 Peripheral configuration	11
3.2.4 Peripheral assignment	11
4 References	13



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-bandwidth 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.

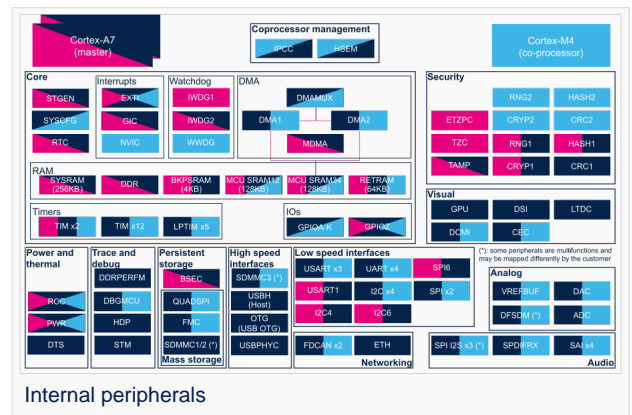
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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



Cortex[®]

Doubledata rate (memory domain)

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.)

Linux[®] is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

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

Contents

1 Article purpose	14
2 Peripheral overview	15
2.1 Features	15
2.2 Security support	15
3 Peripheral usage and associated software	16
3.1 Boot time	16
3.2 Runtime	16
3.2.1 Overview	16
3.2.2 Software frameworks	16
3.2.3 Peripheral configuration	16
3.2.4 Peripheral assignment	16
4 References	18



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-bandwidth 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.

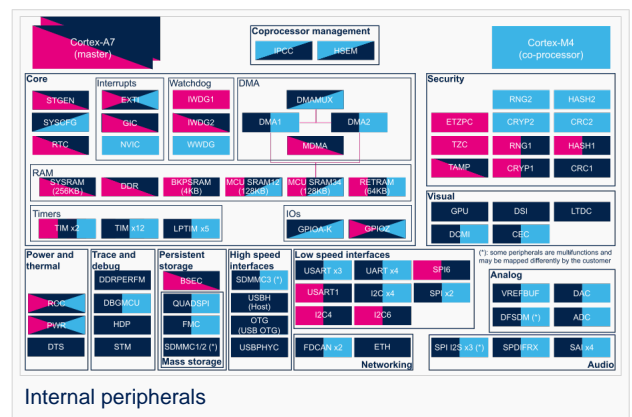
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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



Cortex[®]

Doubledata rate (memory domain)

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.)

Linux[®] is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

Stable: 04.10.2021 - 08:05 / Revision: 04.10.2021 - 08:04

Contents

1 Article purpose	19
2 Peripheral overview	20
2.1 Features	20
2.2 Security support	20
3 Peripheral usage and associated software	21
3.1 Boot time	21
3.2 Runtime	21
3.2.1 Overview	21
3.2.2 Software frameworks	21
3.2.3 Peripheral configuration	21
3.2.4 Peripheral assignment	21
4 References	23



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-bandwidth 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.

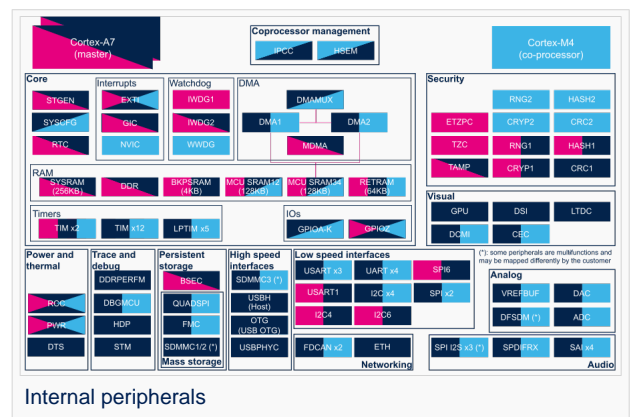
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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



Cortex®

Doubledata rate (memory domain)

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.)

Linux® is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

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

Contents

1 Article purpose	24
2 Peripheral overview	25
2.1 Features	25
2.2 Security support	25
3 Peripheral usage and associated software	26
3.1 Boot time	26
3.2 Runtime	26
3.2.1 Overview	26
3.2.2 Software frameworks	26
3.2.3 Peripheral configuration	26
3.2.4 Peripheral assignment	26
4 References	28



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-bandwidth 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.

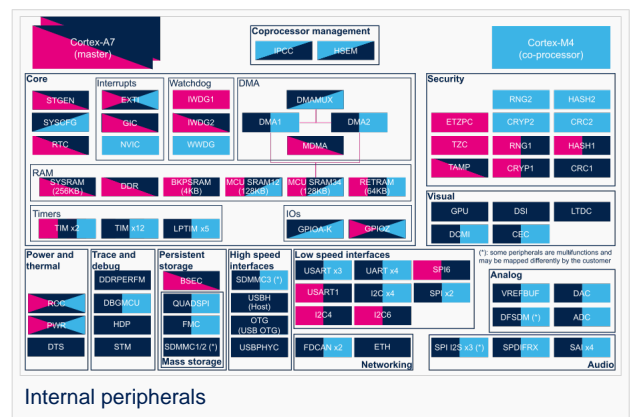
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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



Cortex[®]

Doubledata rate (memory domain)

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.)

Linux[®] is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

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

Contents

1 Article purpose	29
2 Peripheral overview	30
2.1 Features	30
2.2 Security support	30
3 Peripheral usage and associated software	31
3.1 Boot time	31
3.2 Runtime	31
3.2.1 Overview	31
3.2.2 Software frameworks	31
3.2.3 Peripheral configuration	31
3.2.4 Peripheral assignment	31
4 References	33



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-bandwidth 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.

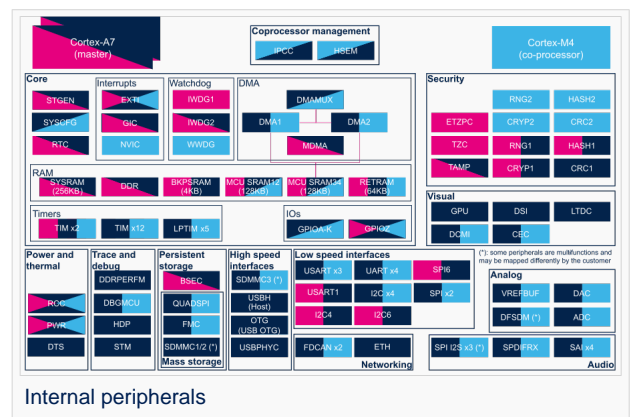
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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

Cortex®

Doubledata rate (memory domain)

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.)

Linux® is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

Stable: 31.03.2021 - 11:58 / Revision: 23.03.2021 - 14:07

Contents

1 Article purpose	34
2 Peripheral overview	35
2.1 Features	35
2.2 Security support	35
3 Peripheral usage and associated software	36
3.1 Boot time	36
3.2 Runtime	36
3.2.1 Overview	36
3.2.2 Software frameworks	36
3.2.3 Peripheral configuration	36
3.2.4 Peripheral assignment	36
4 References	38



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-bandwidth 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.

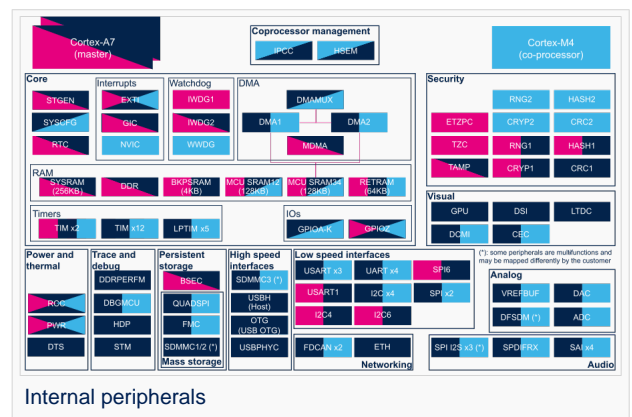
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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



Cortex[®]

Doubledata rate (memory domain)

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.)

Linux[®] is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

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

Contents

1 Article purpose	39
2 Peripheral overview	40
2.1 Features	40
2.2 Security support	40
3 Peripheral usage and associated software	41
3.1 Boot time	41
3.2 Runtime	41
3.2.1 Overview	41
3.2.2 Software frameworks	41
3.2.3 Peripheral configuration	41
3.2.4 Peripheral assignment	41
4 References	43



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-bandwidth 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.

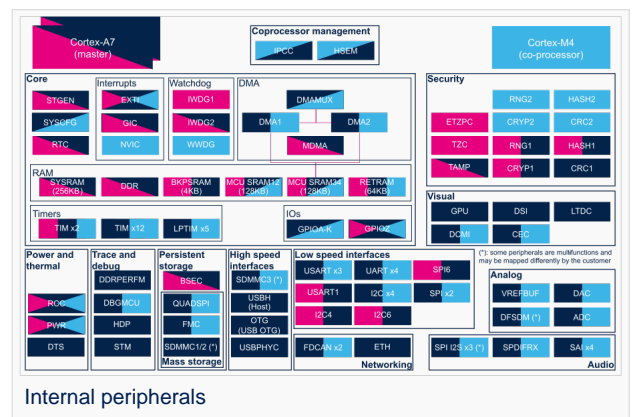
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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



Cortex[®]

Doubledata rate (memory domain)

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.)

Linux[®] is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

Stable: 20.07.2021 - 09:02 / Revision: 11.03.2021 - 08:07

Contents

1 Article purpose	44
2 Peripheral overview	45
2.1 Features	45
2.2 Security support	45
3 Peripheral usage and associated software	46
3.1 Boot time	46
3.2 Runtime	46
3.2.1 Overview	46
3.2.2 Software frameworks	46
3.2.3 Peripheral configuration	46
3.2.4 Peripheral assignment	46
4 References	48



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-bandwidth 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.

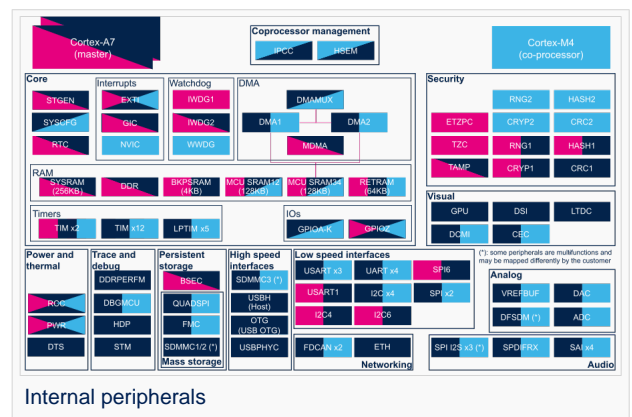
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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



Cortex®

Doubledata rate (memory domain)

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.)

Linux® is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

Stable: 26.03.2021 - 11:32 / Revision: 12.03.2021 - 11:07

Contents

1 Article purpose	49
2 Peripheral overview	50
2.1 Features	50
2.2 Security support	50
3 Peripheral usage and associated software	51
3.1 Boot time	51
3.2 Runtime	51
3.2.1 Overview	51
3.2.2 Software frameworks	51
3.2.3 Peripheral configuration	51
3.2.4 Peripheral assignment	51
4 References	53



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-bandwidth 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.

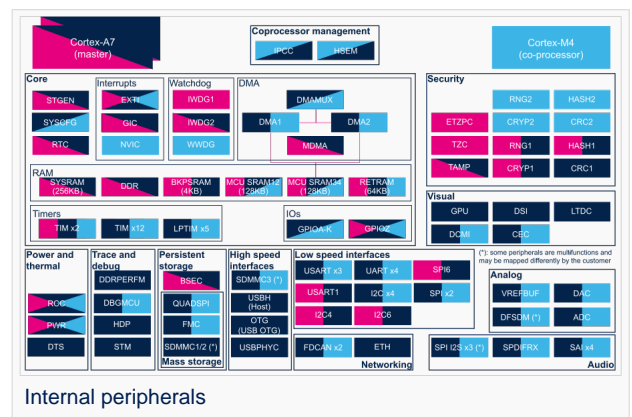
3.2.4 Peripheral assignment

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 possibilities 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

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

Direct Memory Access

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

Cortex[®]

Doubledata rate (memory domain)

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.)

Linux[®] is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment