Category:RAM interfaces

Contents

Stable: 17.06.2020 - 15:26 / Revision: 16.01.2020 - 12:53

Stable: 17:06.2020 - 10:20 | Revision: 10:01.2020 - 12:06

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 **RAM interfaces** internal peripherals and memories (hardware blocks) embedded in the STM32 MPUs microprocessor devices.

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Pages in category "RAM interfaces"

The following 5 pages are in this category, out of 5 total.

- BKPSRAM internal memory
- DDRCTRL and DDRPHYC internal peripherals
- MCU SRAM internal memory
- RETRAM internal memory
- SYSRAM internal memory

Stable: 25.09.2020 - 09:45 / Revision: 25.09.2020 - 09:44

A quality version of this page, approved on *25 September 2020*, was based off this revision.

Contents

1 Peripheral overview

The **BKPSRAM** internal memory is 4 Kbytes wide and is located in the VSW power domain, allowing it to be supplied during Standby low power mode, or to be switched off.

1.1 Features

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

1.2 Security support

The BKPSRAM is a **secure** peripheral (under ETZPC control).

2 Peripheral usage and associated software

2.1 Boot time

The BKPSRAM internal memory is not used during a cold boot or a wake up from Standby with DDR OFF.

The BKPSRAM internal memory is used by the runtime secure monitor (from the FSBL or the OP-TEE secure OS) during wake-up from Standby low power mode with the DDR in Self-Refresh mode. In that case, the BKPSRAM internal memory contains the secure context that has to be restored before jumping back to Linux execution, in DDR.

2.2 Runtime

2.2.1 Overview

The BKPSRAM peripheral can be allocated to:

- the Arm®Cortex®-A7 secure to be used under PSCI [1] secure services (from the FSBL or OP-TEE secure monitor) to save the secure context before entering STANDBY low power mode with DDR in Self-Refresh mode. This is the default assignement.

or

- the Cortex-A7 non-secure to be used under Linux® as reserved memory, for instance.

2.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core/RAM	BKPSRAM	TF-A overview	Linux reserved memory		

2.2.3 Peripheral configuration

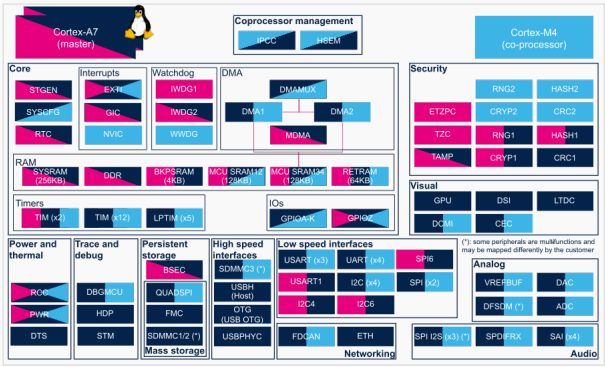
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration by itself can be done via the STM32CubeMX tool for all internal peripherals, and can then be manually be completed (particularly for external peripherals), according to the information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

2.2.4 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/RAM	BKPSRAM	BKPSRAM				Assignment (single choice)

3 References

- http://infocenter.arm.com/help/topic/com.arm.doc.den0022d/Power_State_Coordination_Interface_PDD_v1_1_DEN0022D.pdf

Power State Coordination Interface

Doubledata rate (memory domain)

Open Portable Trusted Execution Environment

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Stable: 25.09.2020 - 09:43 / Revision: 25.09.2020 - 09:39

A quality version of this page, approved on *25 September 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the DDRCTRL and DDRPHYC peripherals and their main features
- indicate the level of security supported by those hardware blocks
- explain how they can be allocated to the three runtime contexts and linked to the corresponding software components
- explain, when necessary, how to configure the DDRCTRL and DDRPHYC peripherals.

2 Peripheral overview

DDRCTRL and **DDRPHYC** peripherals are used to configure the physical interface to the external DDR memory.

2.1 Features

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

2.2 Security support

DDRCTRL and DDRPHYC are **secure aware** (under ETZPC control).

Access to the DDR memory can be filtered via the TZC controller: for instance, it is possible to forbid access from the Cortex®-M4 to the DDR region used by the Cortex®-A7.

3 Peripheral usage and associated software

3.1 Boot time

DDRCTRL and DDRPHYC are kept secure and used by the FSBL to initialize the access to the DDR where it loads the SSBL (U-Boot) for execution.
STMicroelectronics wishes to make the DDR memory configuration as easy as possible, for this reason a dedicated application note[1] has been published and a **DDR tuning** function is available in STM32CubeMX tool in order to generate the device tree configuration that is given to the FSBL to perform this initialization.

3.2 Runtime

3.2.1 Overview

DDRCTRL and DDRPHYC are accessed at runtime by the secure monitor (from the FSBL or OP-TEE) to put the DDR in self-refresh state before going into Stop or Standby low power mode.
On Standby exit, the ROM code loads the FSBL that again configures the DDRCTRL and DDRPHYC before proceeding with the wake-up procedure.
The TZC controller is configured by TF-A to split the DDR in two regions:

- the first region, the largest one, is reserved for Linux (Cortex-A7 non-secure context)
- the second region, 32 Mbytes wide, is dedicated for OP-TEE (Cortex-A7 secure context). This area is used by its pager as a cache area from which it can load trusted applications that are authenticated in the SYSRAM internal memory before execution.

This split is visible in the overall memory mapping.

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core/RAM	DDR via DDRCTRL	Memory mapping	Memory mapping		

3.2.3 Peripheral configuration

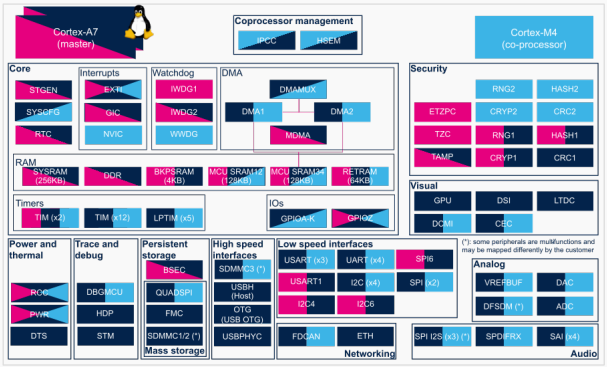
The DDRCTRL and DDRPHYC device tree configuration is generated via STM32CubeMX tool, according to the DDR characteristics (type, size, frequency, speed grade). This configuration is applied during boot time by the FSBL (see Boot chain overview): TF-A or U-Boot SPL.

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/RAM	DDR via DDRCTRL	DDR				

4 References

- AN5168 - DDR configuration on STM32MP1 Series MPU

Doubledata rate (memory domain)

Cortex®

Trusted Firmware for Arm Cortex-A

Linux® is a registered trademark of Linus Torvalds.

Open Portable Trusted Execution Environment

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

First Stage Boot Loader

Stable: 04.02.2020 - 15:59 / Revision: 04.02.2020 - 15:48

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 MCU RAM memory 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 how to configure the MCU RAM memory.

2 Peripheral overview

The **MCU SRAM** internal memory is 384-Kbyte wide and physically near to the Cortex®-M4 for optimized performances from this core. It is split into four separate banks:

- MCU SRAM1 (128 Kbytes)
- MCU SRAM2 (128 Kbytes)
- MCU SRAM3 (64 Kbytes)
- MCU SRAM4 (64 Kbytes)

Those banks have individual security control (cf. security support below) and automatic clock gating (for power management optimization), but they are not supplied when the system goes to Standby low power mode, so their content is lost in that case.

2.1 Features

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

2.2 Security support

Each MCU SRAM1/SRAM2/SRAM3/SRAM4 bank is **secure aware** (under ETZPC control).

3 Peripheral usage and associated software

3.1 Boot time

The ROM code uses the MCU SRAM1 to store the USB context during a boot on USB for Flash programming (with STM32CubeProgrammer).

Linux remoteproc framework (running on the Cortex®-A7) loads the Cortex®-M4 firmware code into the MCU SRAM, except the exception table that must be loaded in the RETRAM since the Cortex®-M4 is looking for its reset entry point at address 0x00000000. The overall memory mapping is shown in the platform memory mapping section.

3.2 Runtime

3.2.1 Overview

Each MCU SRAM bank can be allocated to:

- the Arm®Cortex®-A7 secure for using in OP-TEE

or

- the Arm®Cortex®-A7 non-secure for using in Linux® with reserved memory, that is used by the dmaengine (for DMA buffers management) or RPMsg for interprocess communication with the coprocessor

and/or

- the Arm®Cortex®-M4 for using in STM32Cube

Notice the **and/or** allocation between Cortex®-A7 non-secure and Cortex®-M4, meaning that it is possible to share banks between those cores, typically to realize inter process communication between RPMsg on Linux side and OpenAMP on STM32Cube side.

The default assignement set in STMicroelectronics distribution is in line with the platform memory mapping, that can be adapted by the platform user.

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core/RAM	MCU SRAM	OP-TEE overview	Linux reserved memory	STM32Cube	

3.2.3 Peripheral configuration

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

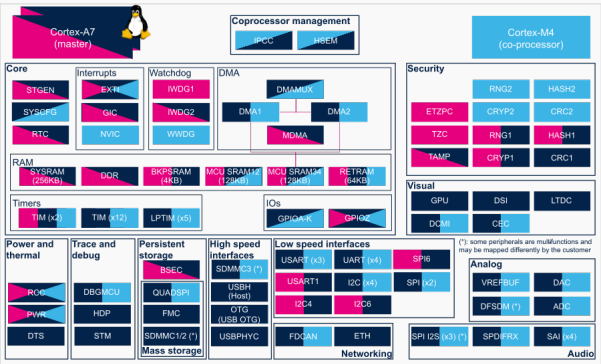
The several SRAM banks are accessible via different address ranges in order to benefit from the Cortex-M4 multiple ports.

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/RAM	MCU SRAM	SRAM1				Assignment (between A7 S and A7 NS / M4) Shareable (between A7 NS and M4)
		SRAM2				Assignment (between A7 S and A7 NS / M4) Shareable (between A7 NS and M4)
		SRAM3				Assignment (between A7 S and A7 NS / M4) Shareable (between A7 NS and M4)
		SRAM4				Assignment (between A7 S and A7 NS / M4) Shareable (between

Domain	Peripheral	Runtime allocation				Comment
						A7 NS and M4)

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.)

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Cortex®

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.

Open Portable Trusted Execution Environment

Stable: 04.02.2020 - 15:59 / Revision: 04.02.2020 - 15:50

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

Contents

1 Peripheral overview

The **RETRAM** internal memory is 64 Kbytes wide and is physically near to the Arm®Cortex®-M4 for optimized performance from the core. It is located in the VSW power domain, allowing it to be supplied during Standby low power mode, and to retain retention firmware that can be executed very quickly by the Cortex-M4 on wake up from Standby mode.

1.1 Features

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

1.2 Security support

The RETRAM is a **secure** peripheral (under ETZPC control).

2 Peripheral usage and associated software

2.1 Boot time

Linux® remoteproc framework (running on the Cortex-A7) loads the Cortex-M4 firmware to the RETRAM, starting at address 0x00000000. At least, it must load the part of the firmware containing the vector table, since the Cortex-M4 reset entry point is address 0x00000004. The rest of the firmware code is loaded into the MCU SRAM. The overall memory mapping is shown in the platform memory mapping section.

2.2 Runtime

2.2.1 Overview

The Cortex-M4 vector table is mapped from address 0x00000000 (so to the RETRAM) at reset, but it can be remapped by software to any other location by means of the vector table offset register (VTOR). Beyond the reset entry point (0x00000004), the exception table also contains the software entries table used by the NVIC to branch the software execution to the right interrupt service routine.

While going to Standby low power mode, the RETRAM can remain supplied, so it can preserve a (small) Cortex-M4 piece of retention firmware that is executed on wake up when the ROM code (running on Cortex-A7) restarts the Cortex-M4.
All these constraints make the RETRAM the minimum (and default) choice for Cortex-M4 firmware.

RETRAM can be allocated to:

- the Cortex-A7 secure to be used under OP-TEE.

or

- the Cortex-A7 non-secure to be used under Linux as reserved memory.

or

- the Cortex-M4 for use with the STM32Cube MPU Package, either for **runtime firmware** that ca be mapped in both RETRAM and MCU SRAM, or for **retention firmware** that only fits into the RETRAM, but could have some data in MCU SRAM (keeping in mind that these data are lost while entering Standby low power mode).

2.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core/RAM	RETRAM	OP-TEE overview	Linux reserved memory	STM32Cube	

2.2.3 Peripheral configuration

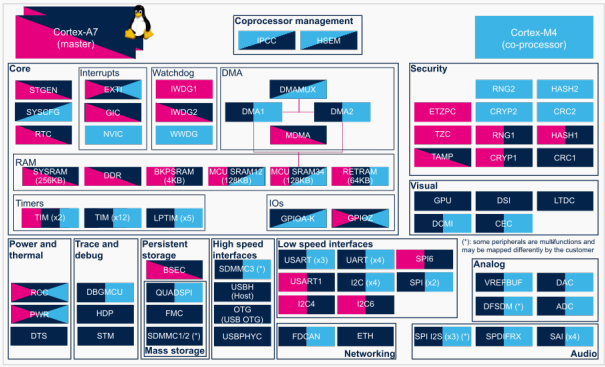
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (especially for external peripherals), according to the information given in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

2.2.4 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)			
Core/RAM	RETRAM	RETRAM				Assignment (single choice)

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

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Microprocessor Unit

Open Portable Trusted Execution Environment

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Stable: 25.09.2020 - 09:45 / Revision: 25.09.2020 - 09:44

A quality version of this page, approved on *25 September 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to briefly introduce the SYSRAM internal memory and indicate the level of security supported by this memory.

2 Peripheral overview

The STM32MP15 **SYSRAM** is a 256-Kbyte internal memory peripheral. It is physically located near the Arm®Cortex-A to optimize the core performance.

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 really implemented.

2.2 Security support

The SYSRAM is a **secure** peripheral (under ETZPC TrustZone memory adapter (TZMA)): it can be split into a secure and a non-secure regions with a 4-Kbyte granularity.

3 Peripheral usage and associated software

3.1 Boot time

The STM32MP15 ROM code mainly configures the SYSRAM as a secure peripheral during its execution. The ROM code uses 9 Kbytes located at the beginning of the SYSRAM to store its read and write data. The ROM code stores the boot context in the first 512 bytes of SYSRAM: this boot context contains several information (such as the selected boot device) and pointers to the ROM code services (used for secure boot authentication). The ROM code loads the FSBL just after the boot context, into the remaining 247 Kbytes of SYSRAM, and eventually branches the Cortex®-A7 core 0 execution to this FSBL.

The FSBL code can use the whole SYSRAM, but it must take care not to overwrite the boot context before taking it into account.

3.2 Runtime

3.2.1 Overview

In STMicroelectronics distribution, the SYSRAM runtime mapping is the one reached at the end of the boot. It is consequently fully secure and contains a minimal secure monitor (from TF-A or U-Boot) or a secure OS (like OP-TEE).

You may decide to split the SYSRAM at runtime. In this case:

- set the SYSRAM bottom secure, for a Cortex®-A7 secure monitor (from TF-A or U-Boot) or a secure OS (such as OP-TEE)

and

- set the SYSRAM top non-secure, for instance for using in Linux® as reserved memory

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Core/RAM	SYSRAM	TF-A overview	Linux reserved memory		

3.2.3 Peripheral configuration

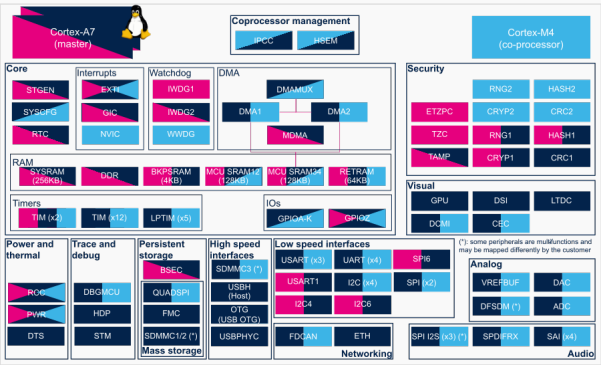
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in 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/RAM	SYSRAM	SYSRAM				Shareable (multiple choices supported)

4 References

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)