STM32MP15 RAM mapping

Contents

Stable: 05.01.2021 - 17:13 / Revision: 05.01.2021 - 17:08

Stable: 05.01.2021 - 17:13 / Revision: 05.01.2021 - 17:08

A quality version of this page, approved on *12 October 2020*, was based off this revision.

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

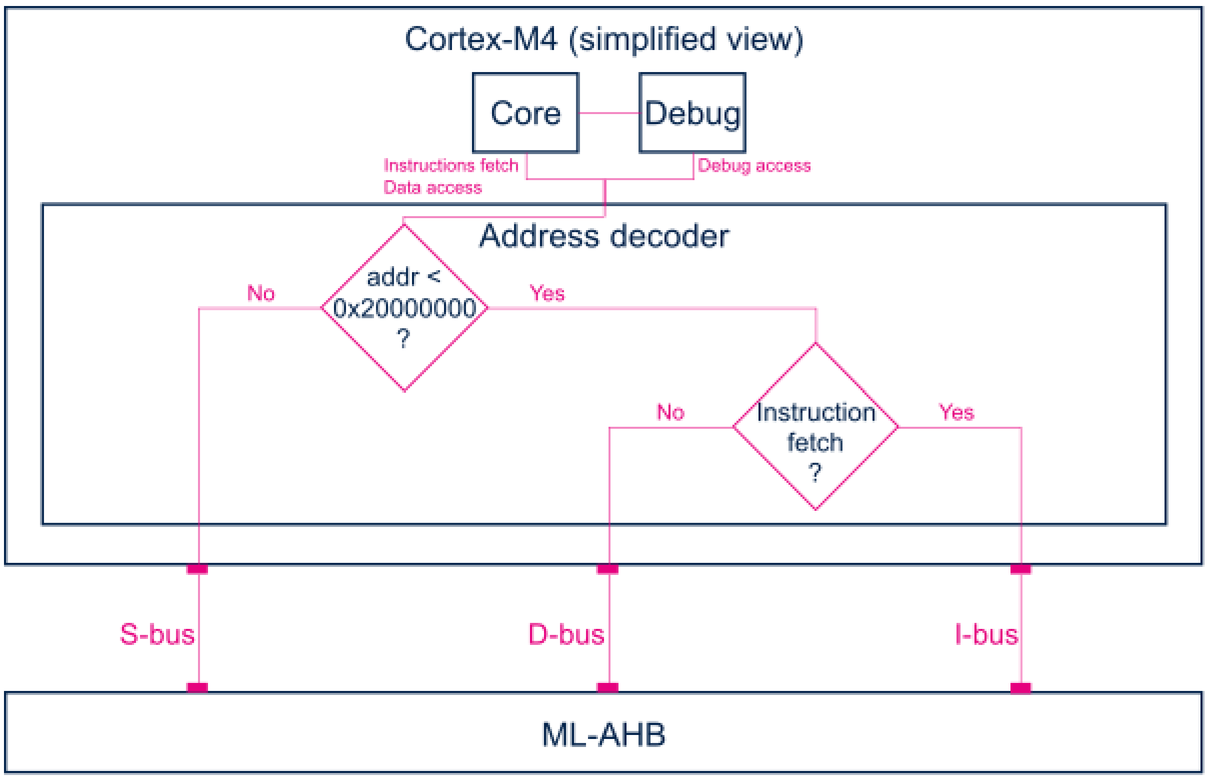
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



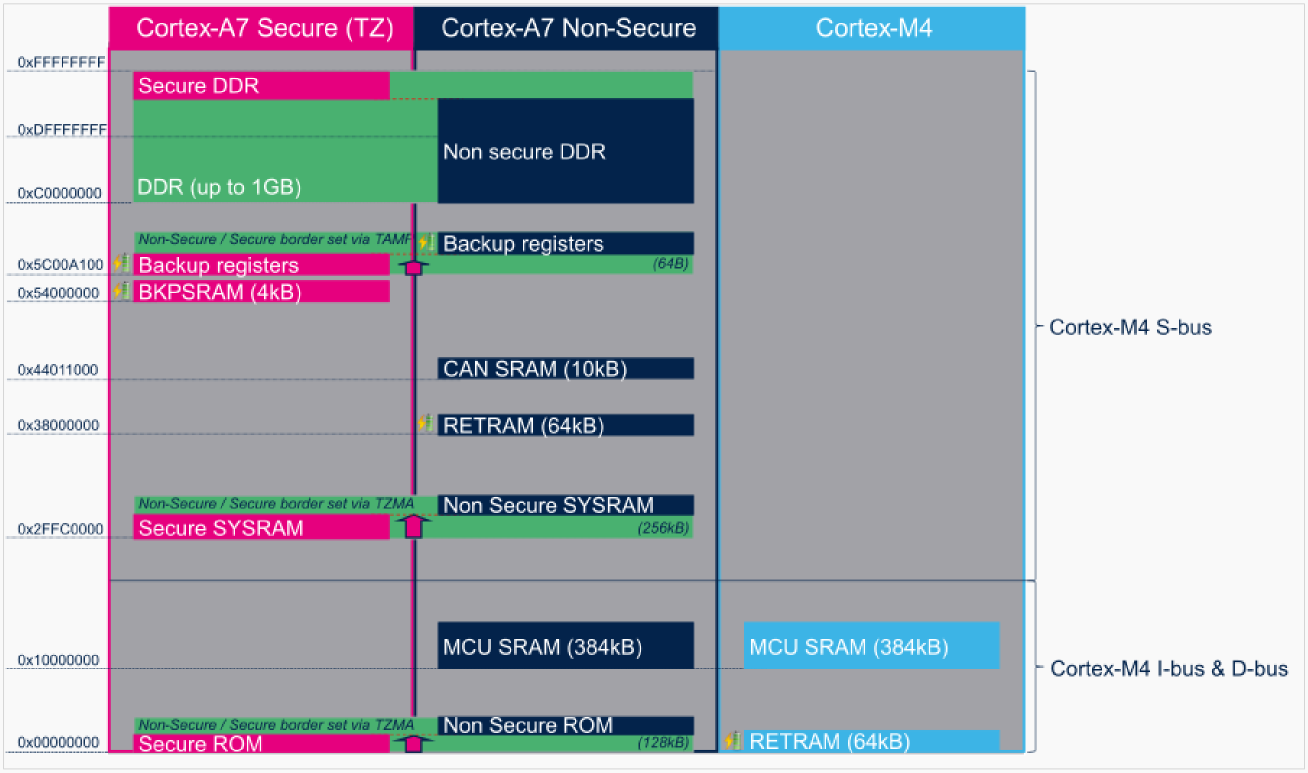
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

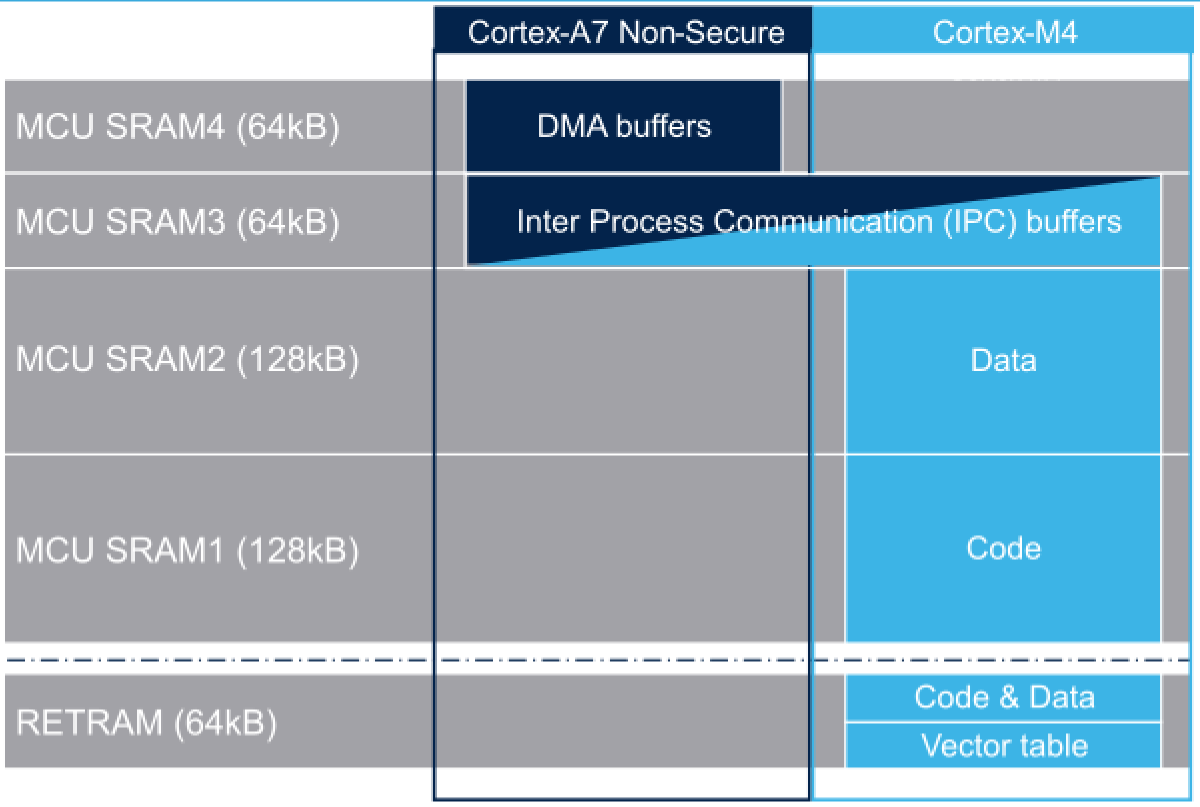
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 MCU SRAM3

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                                 <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                                 <0x30000000 0x30000000 0x60000>,
                                 <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 MCU SRAM1 & SRAM2

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4 
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4 
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                 <0x30000000 0x30000000 0x60000>,
                 <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

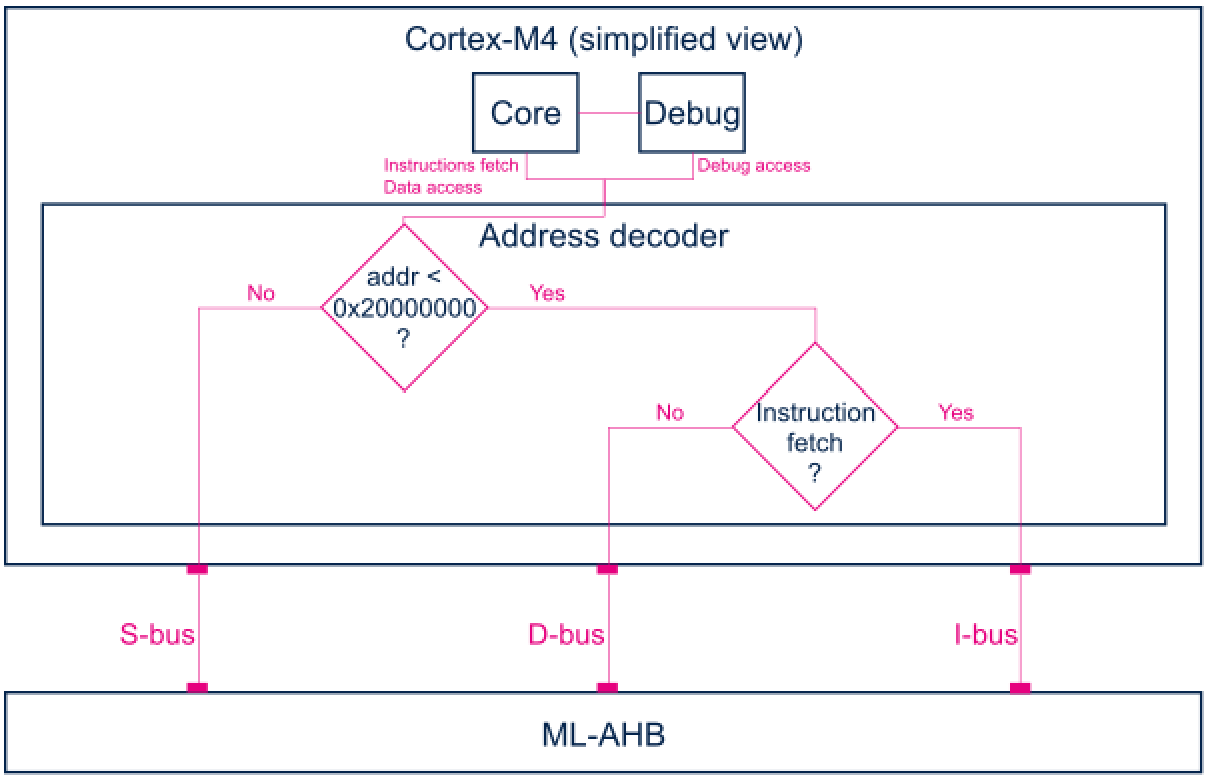
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



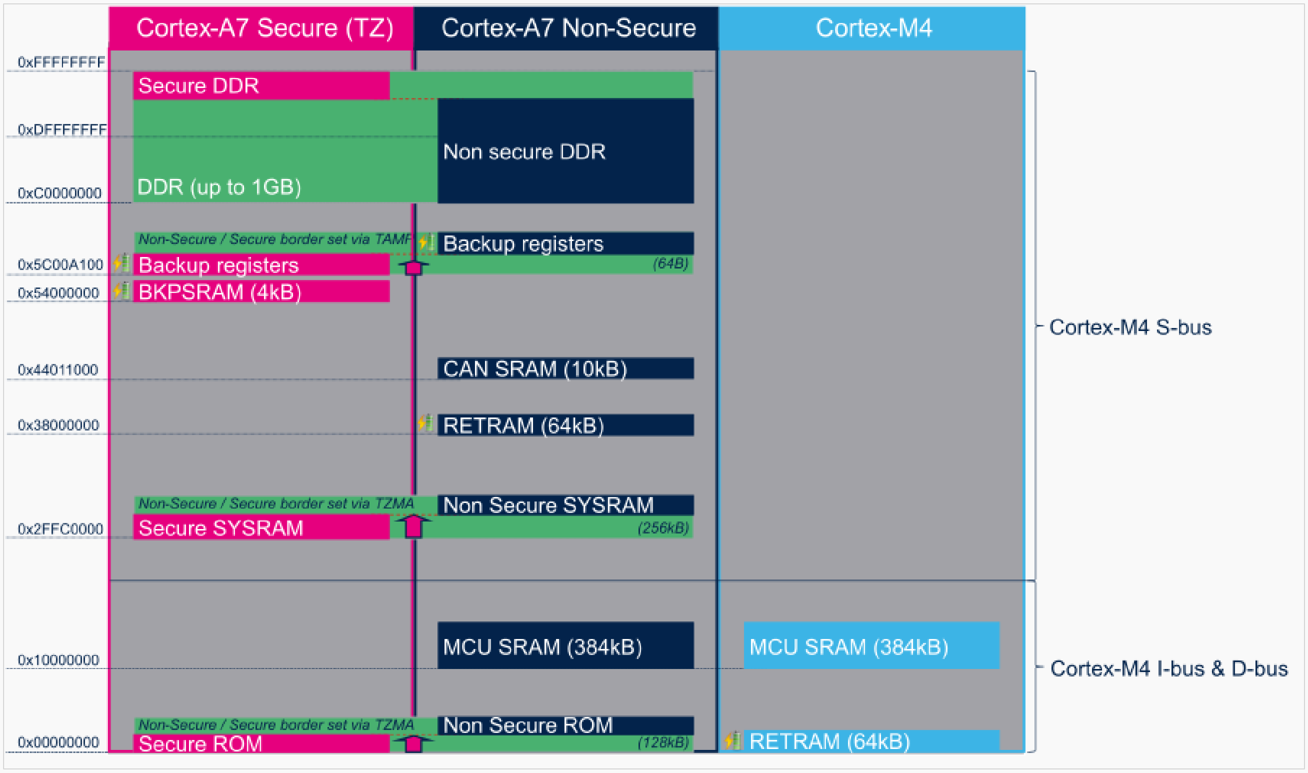
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

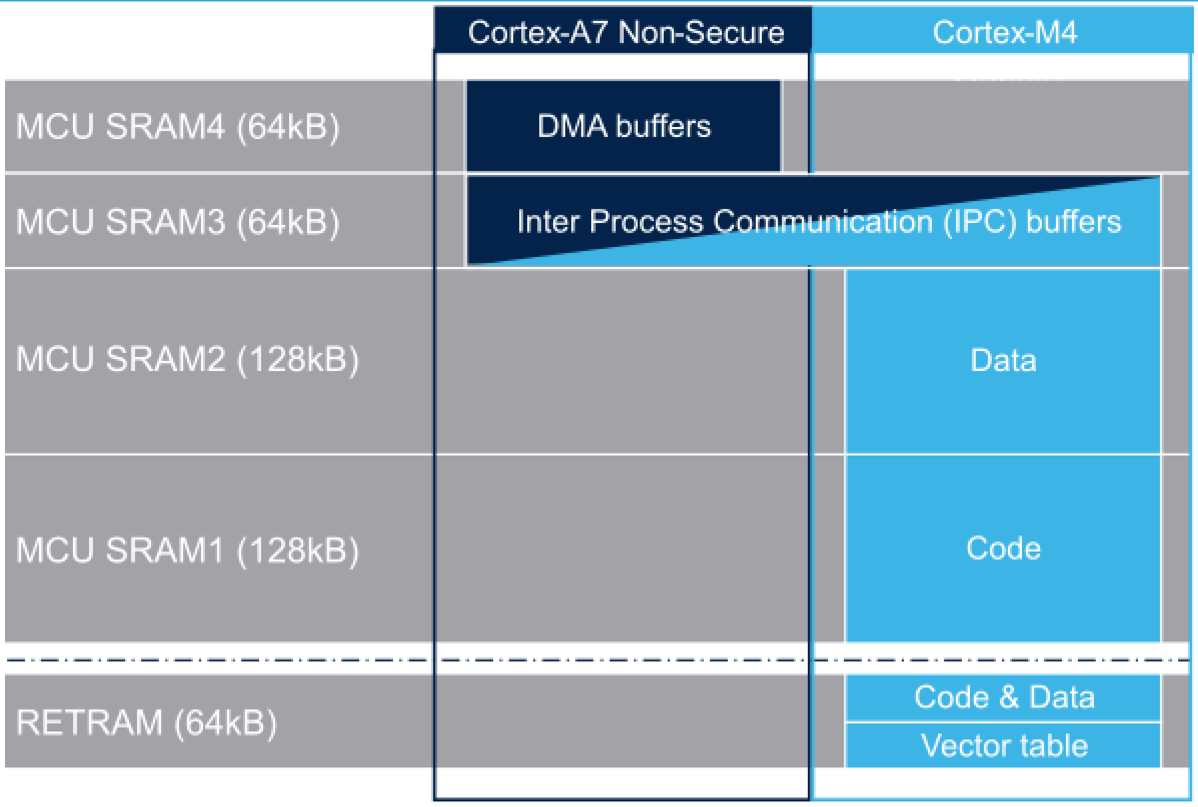
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 MCU SRAM3

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                         <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 MCU SRAM1 & SRAM2

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

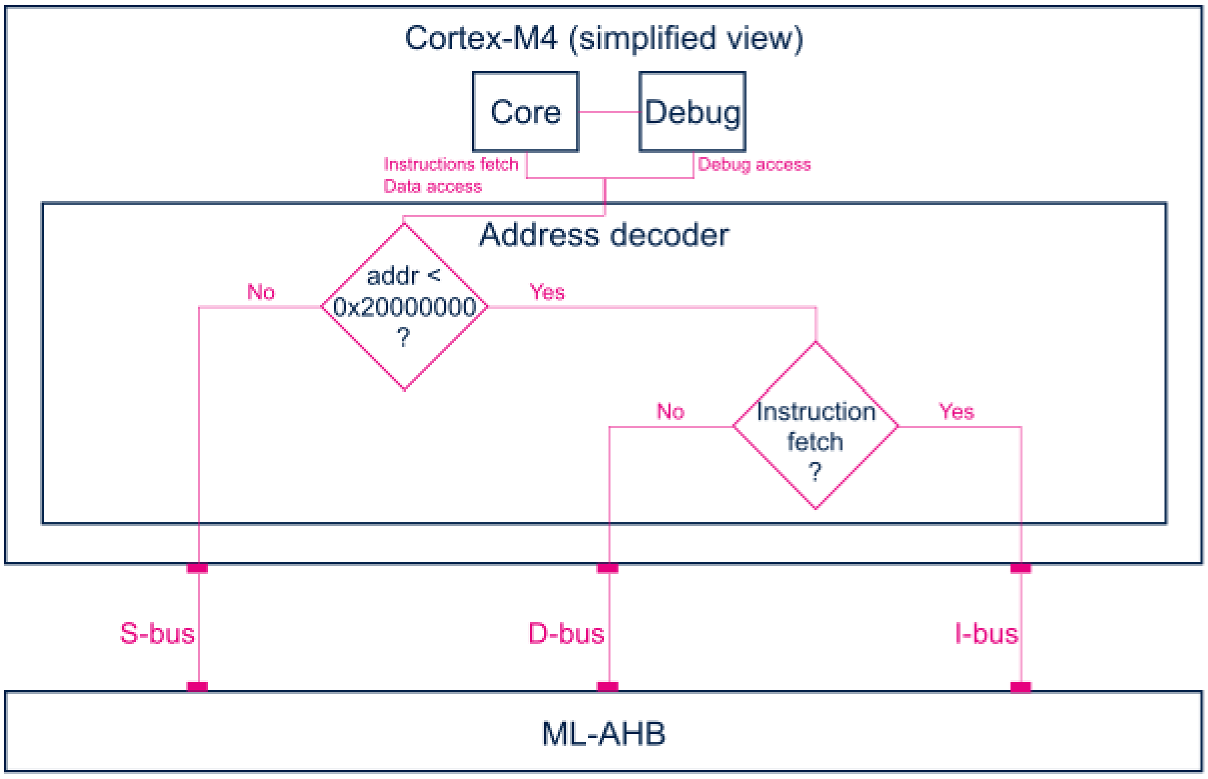
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



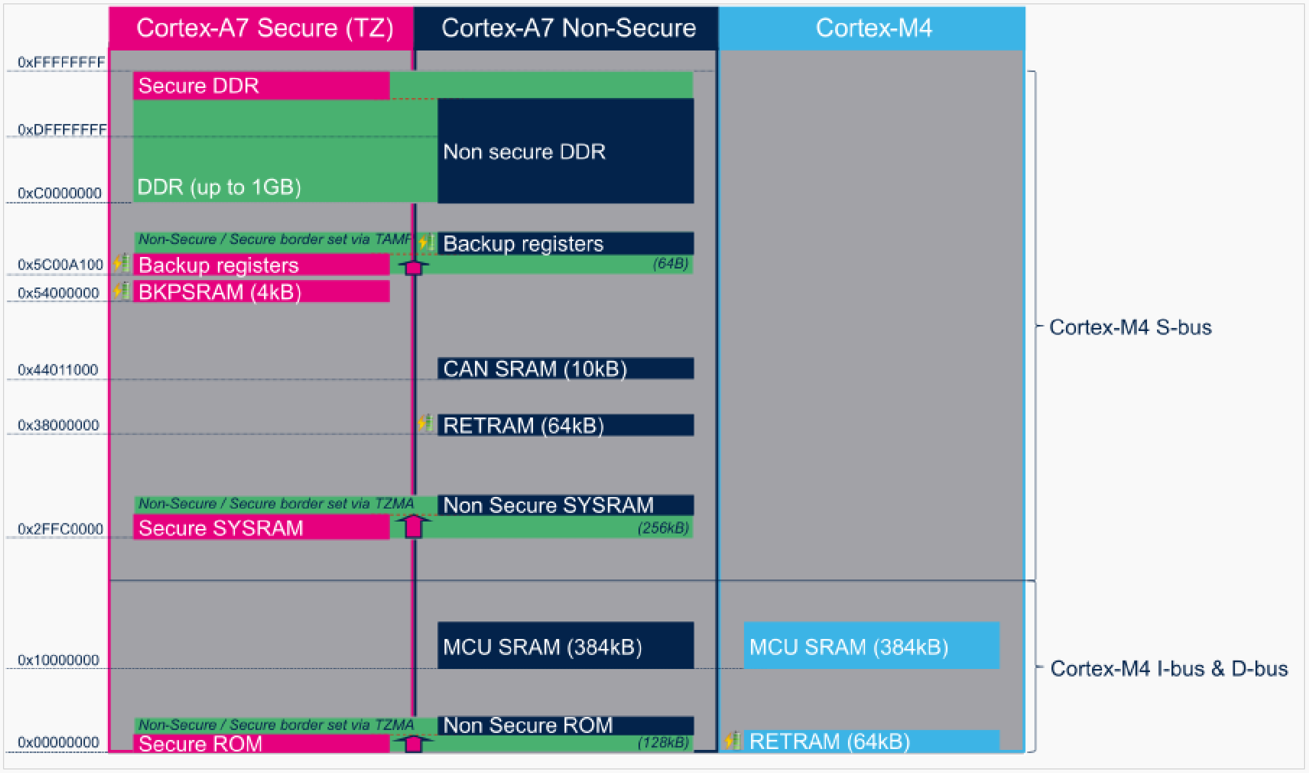
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

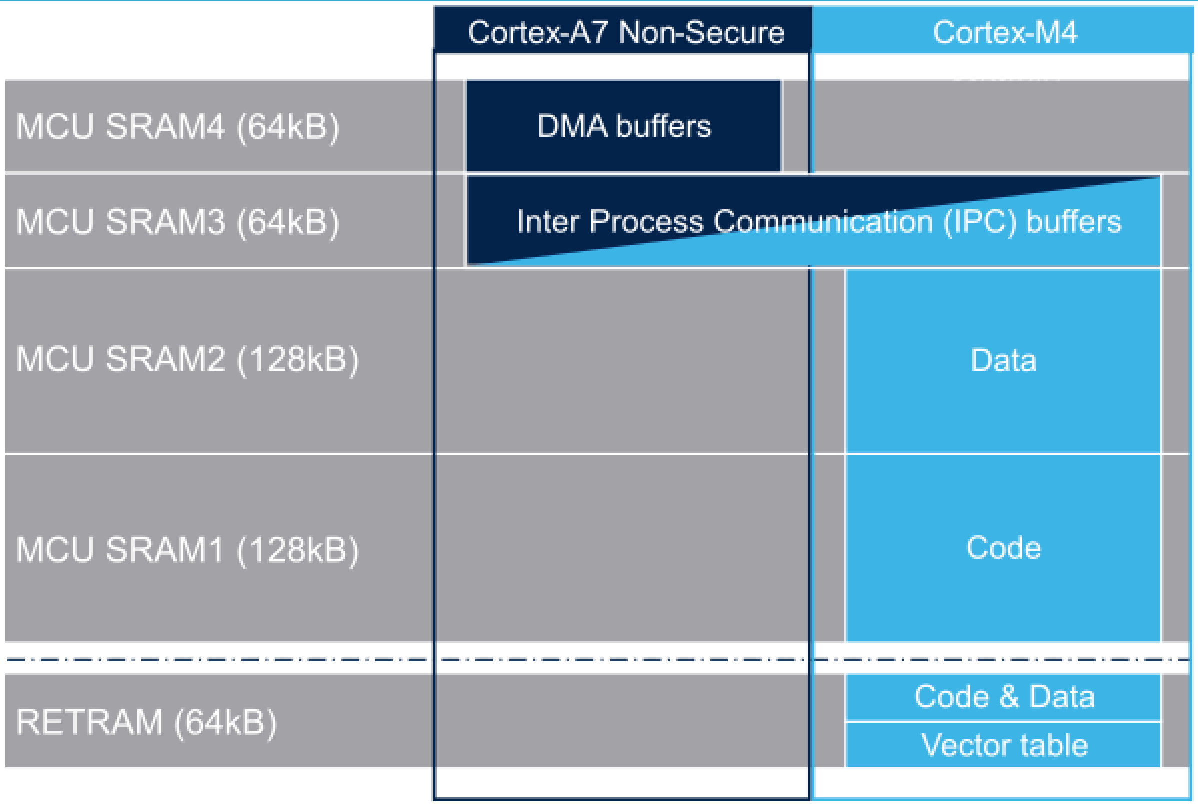
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 MCU SRAM3

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                        <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                        <0x30000000 0x30000000 0x60000>,
                        <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 MCU SRAM1 & SRAM2

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4 
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

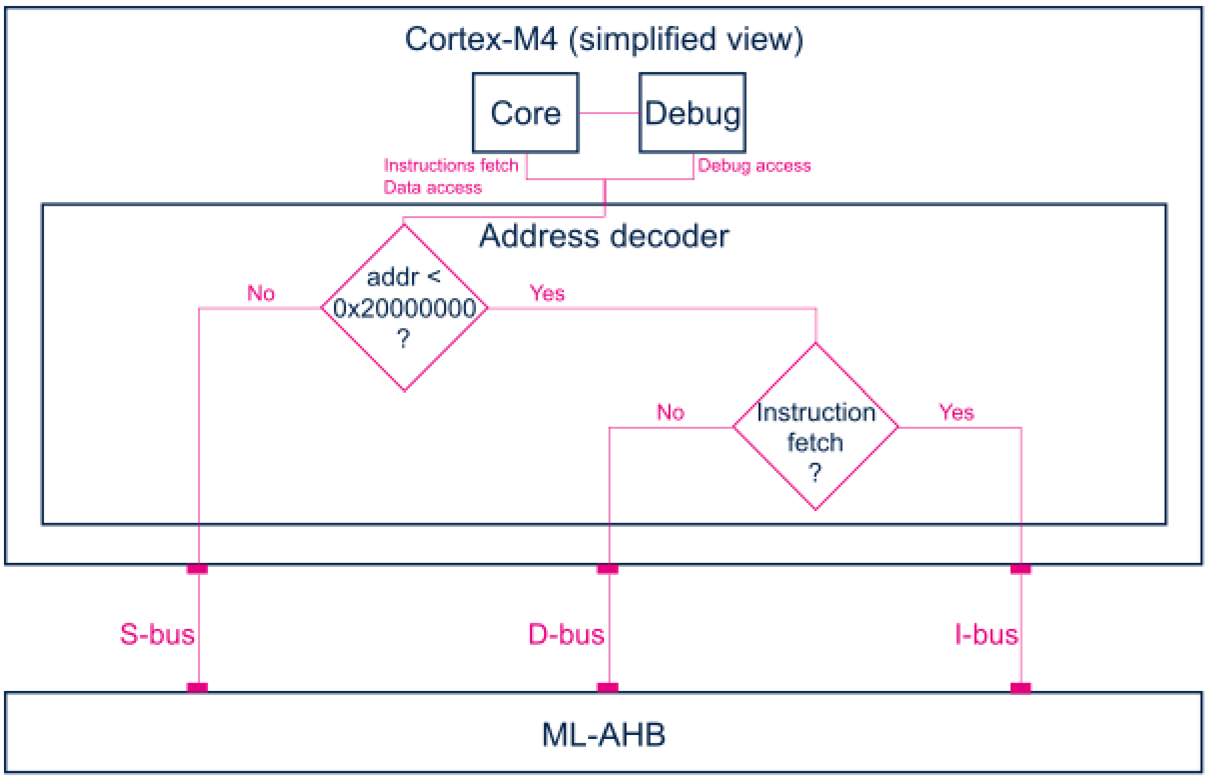
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



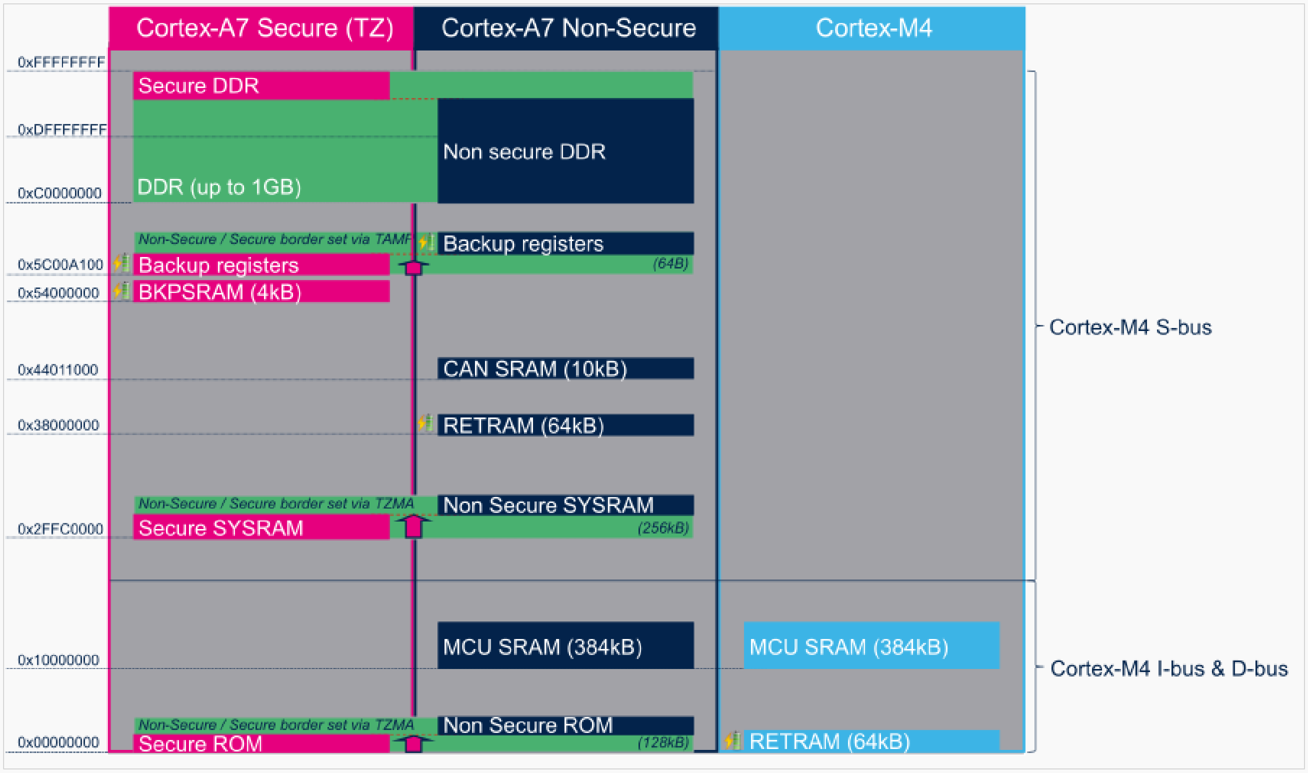
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

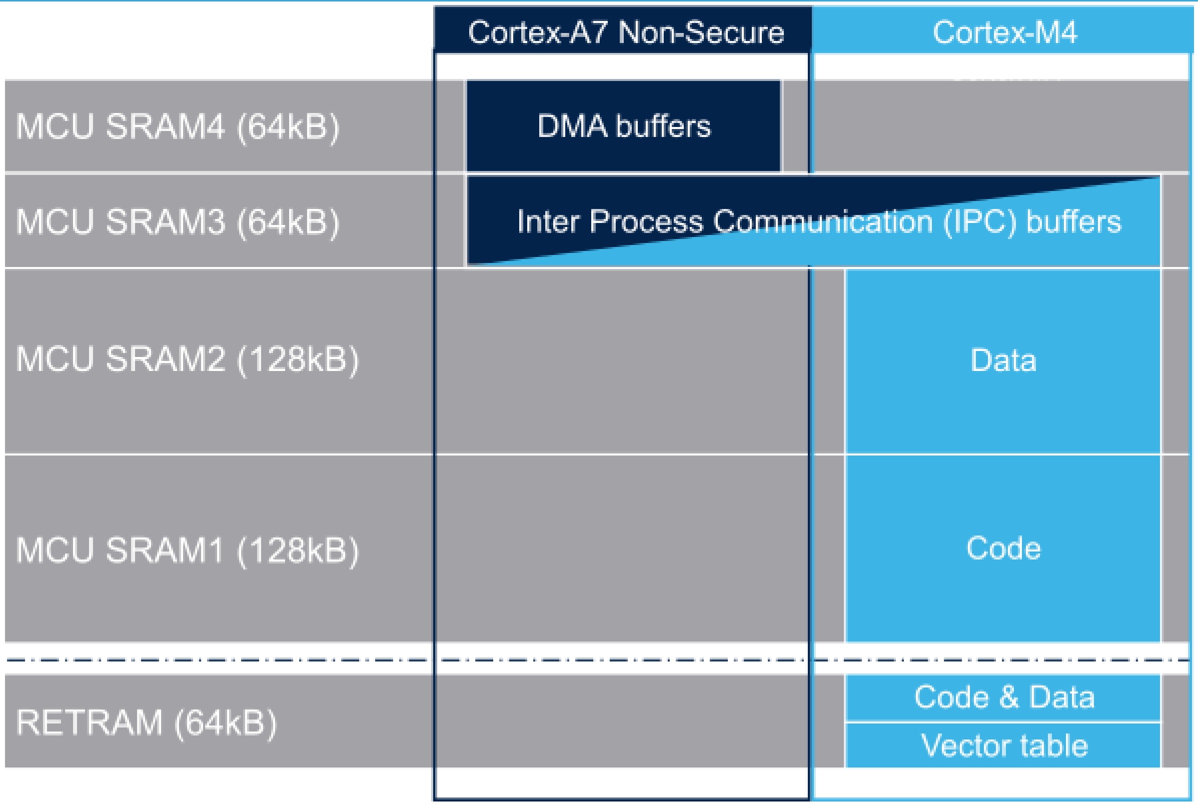
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                   <0x30000000 0x30000000 0x60000>,
                   <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 *MCU SRAM3*

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                 vdev0vring0: vdev0vring0@10040000 {
                         compatible = "shared-dma-pool";
                         reg = <0x10040000 0x2000>;
                         no-map;
                 };

                 vdev0vring1: vdev0vring1@10042000 {
                         compatible = "shared-dma-pool";
                         reg = <0x10042000 0x2000>;
                         no-map;
                 };

                 vdev0buffer: vdev0buffer@10044000 {
                         compatible = "shared-dma-pool";
                         reg = <0x10044000 0x4000>;
                         no-map;
                 };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                         <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 *MCU SRAM1 & SRAM2*

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                 <0x30000000 0x30000000 0x60000>,
                 <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

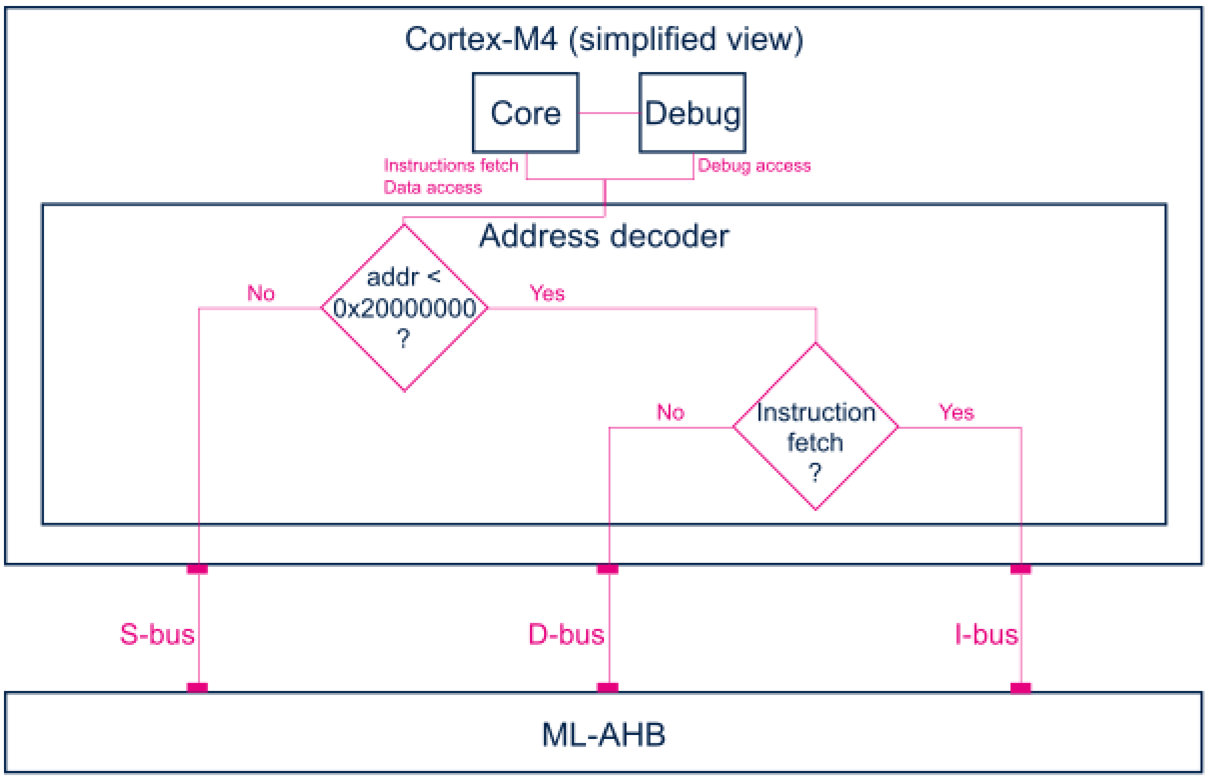
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



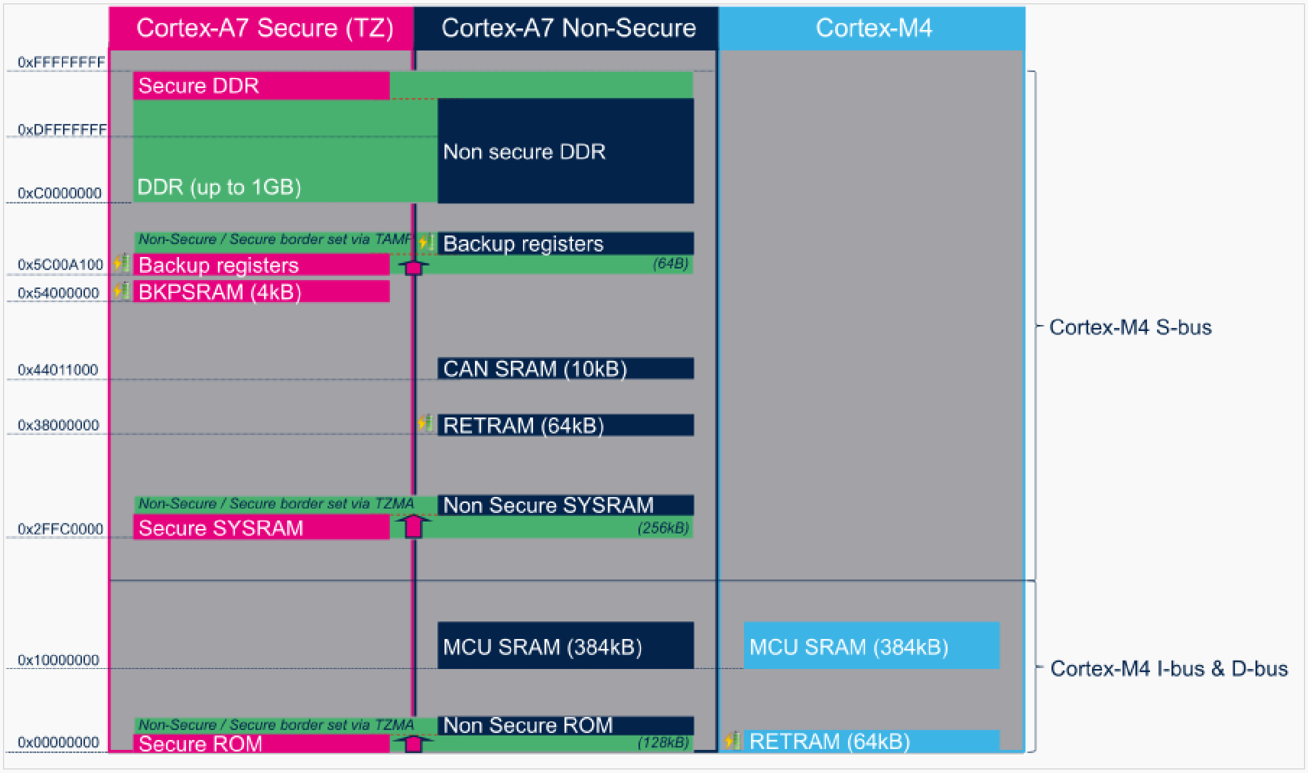
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

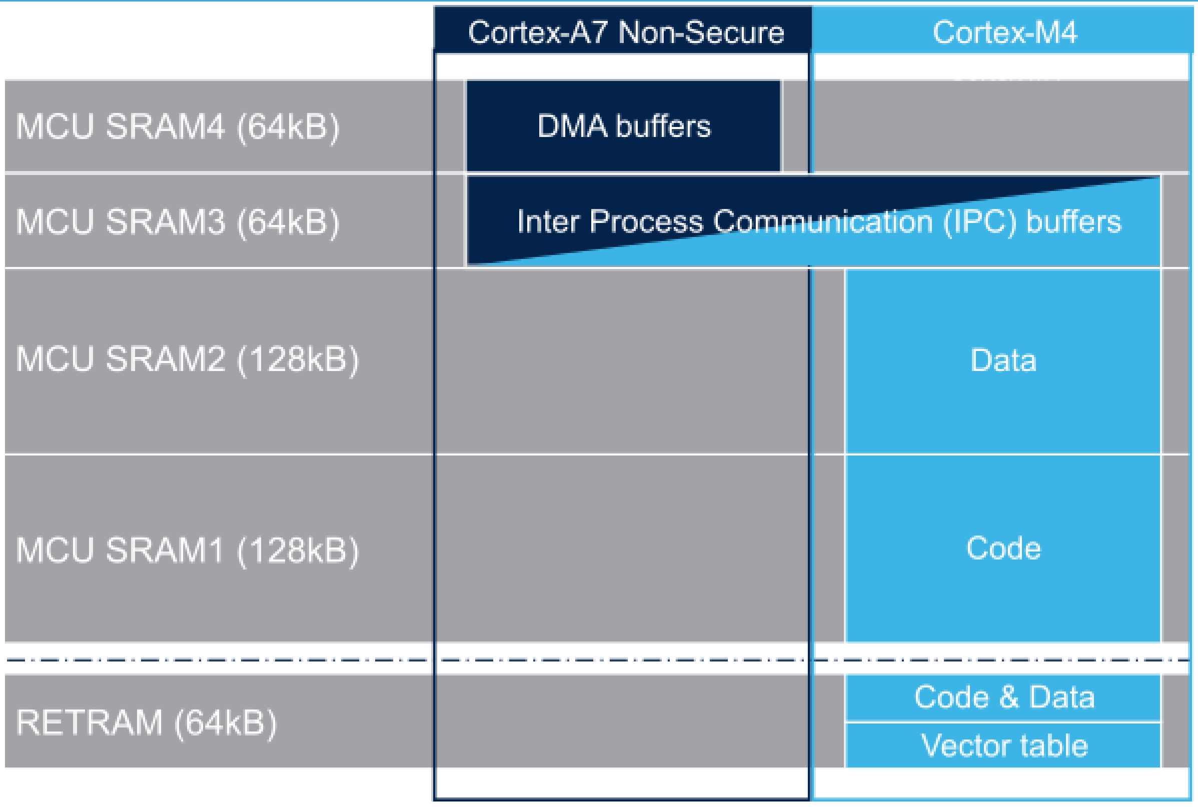
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                   <0x30000000 0x30000000 0x60000>,
                   <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 MCU SRAM3

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                        <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                        <0x30000000 0x30000000 0x60000>,
                        <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 MCU SRAM1 & SRAM2

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                 <0x30000000 0x30000000 0x60000>,
                 <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

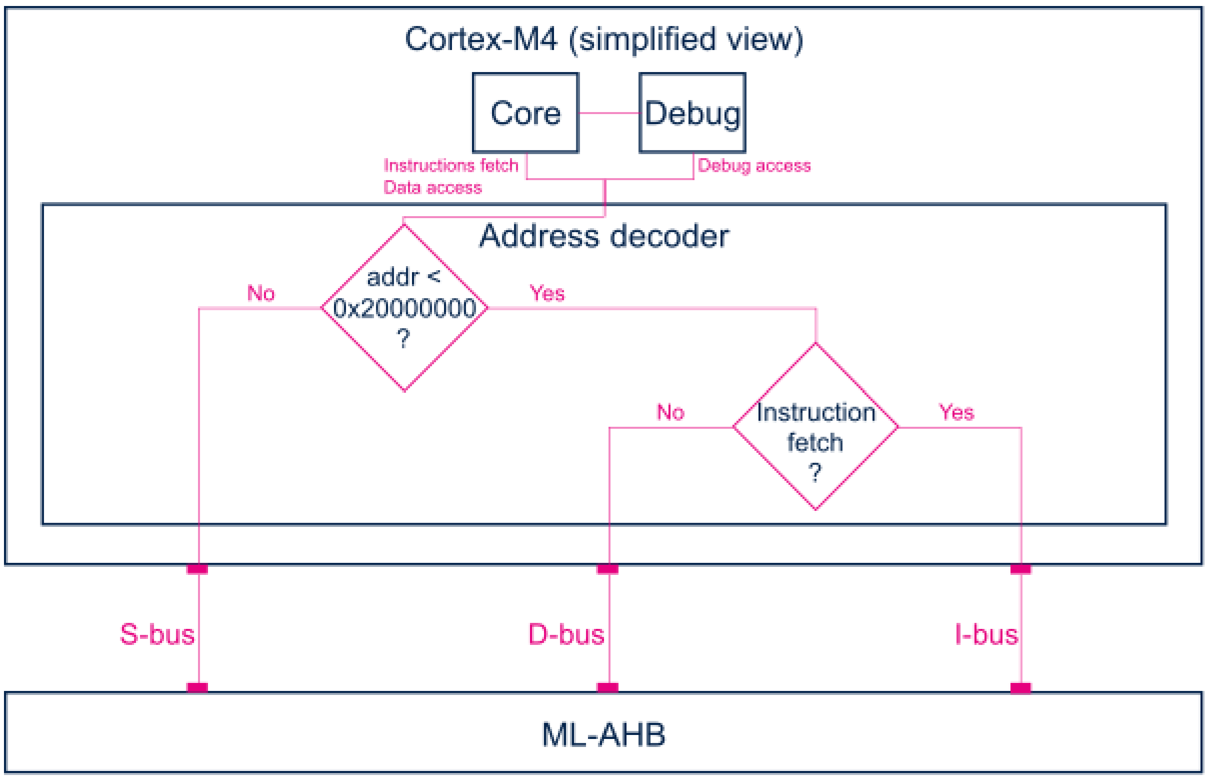
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



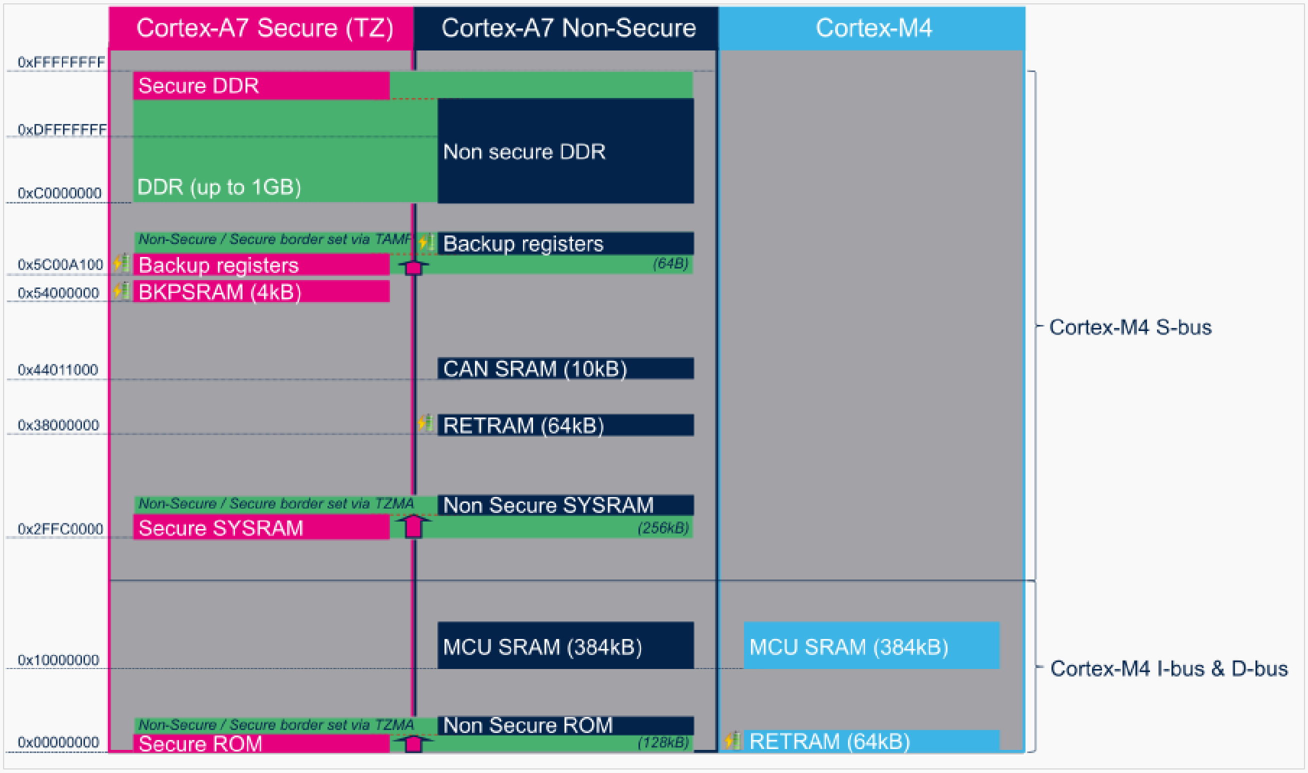
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

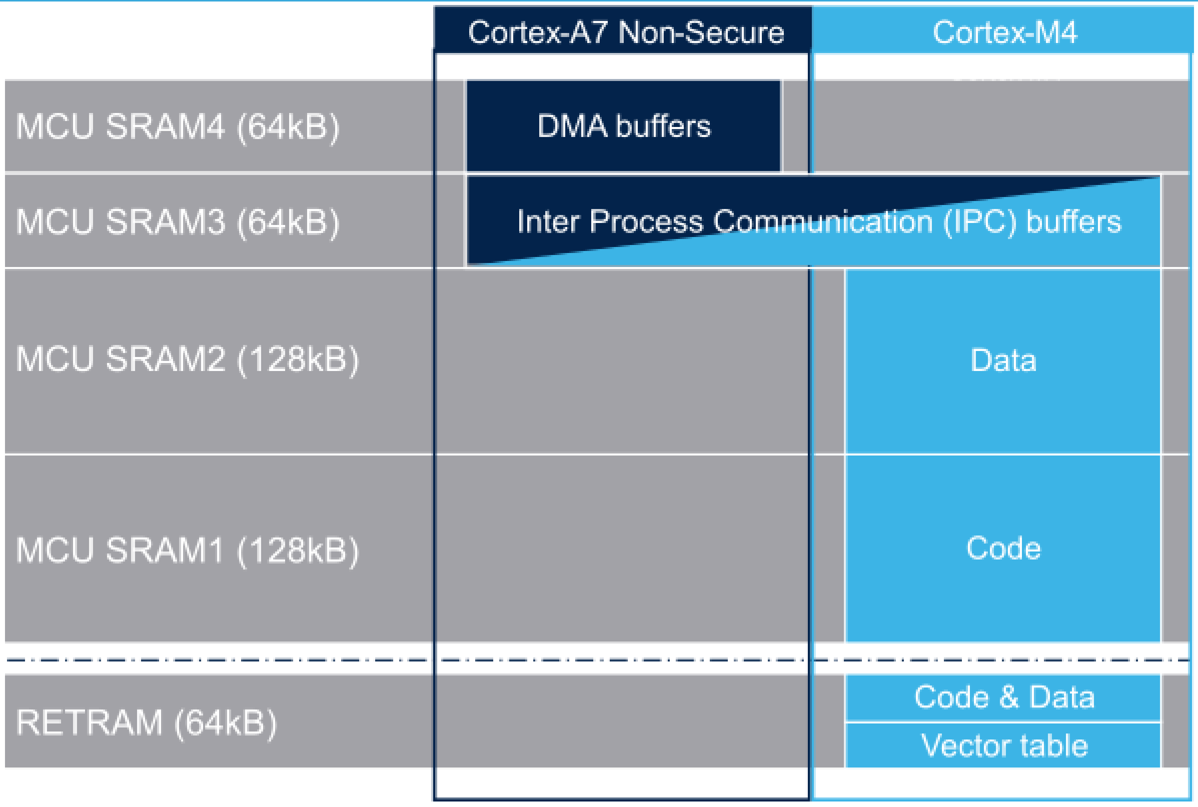
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 MCU SRAM3

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                        <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                        <0x30000000 0x30000000 0x60000>,
                        <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 MCU SRAM1 & SRAM2

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

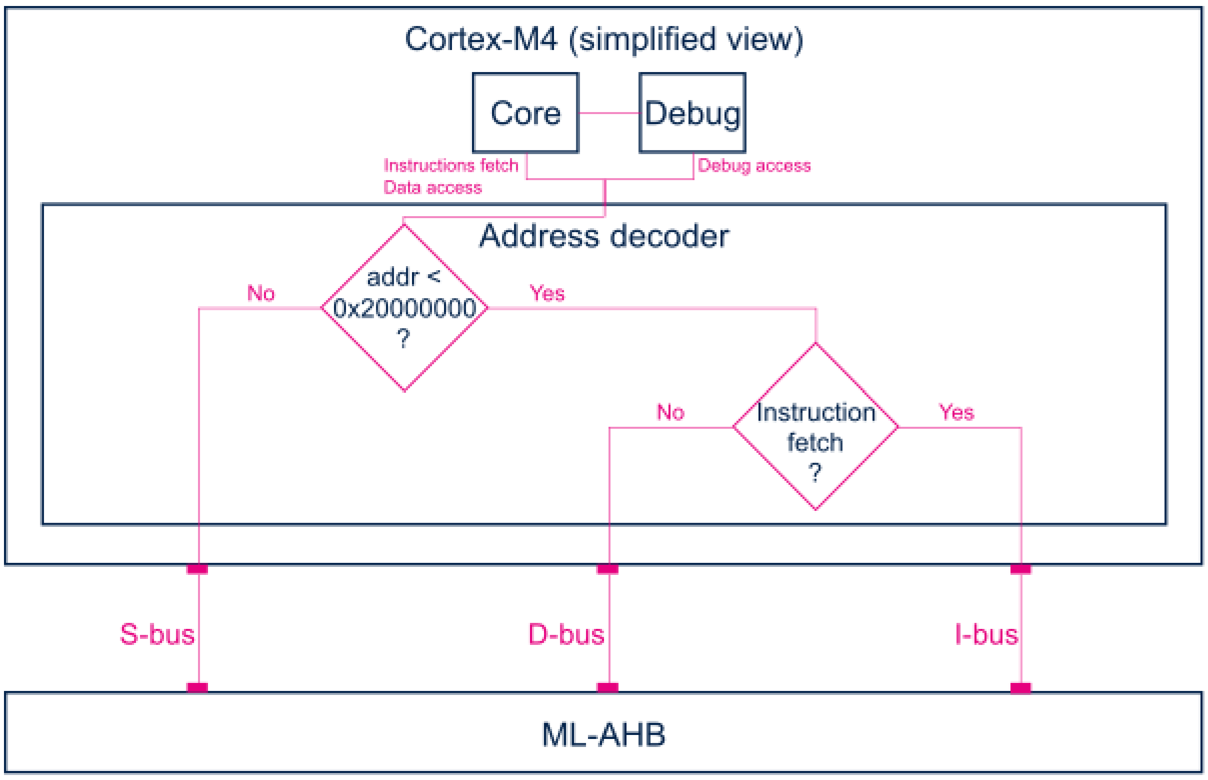
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



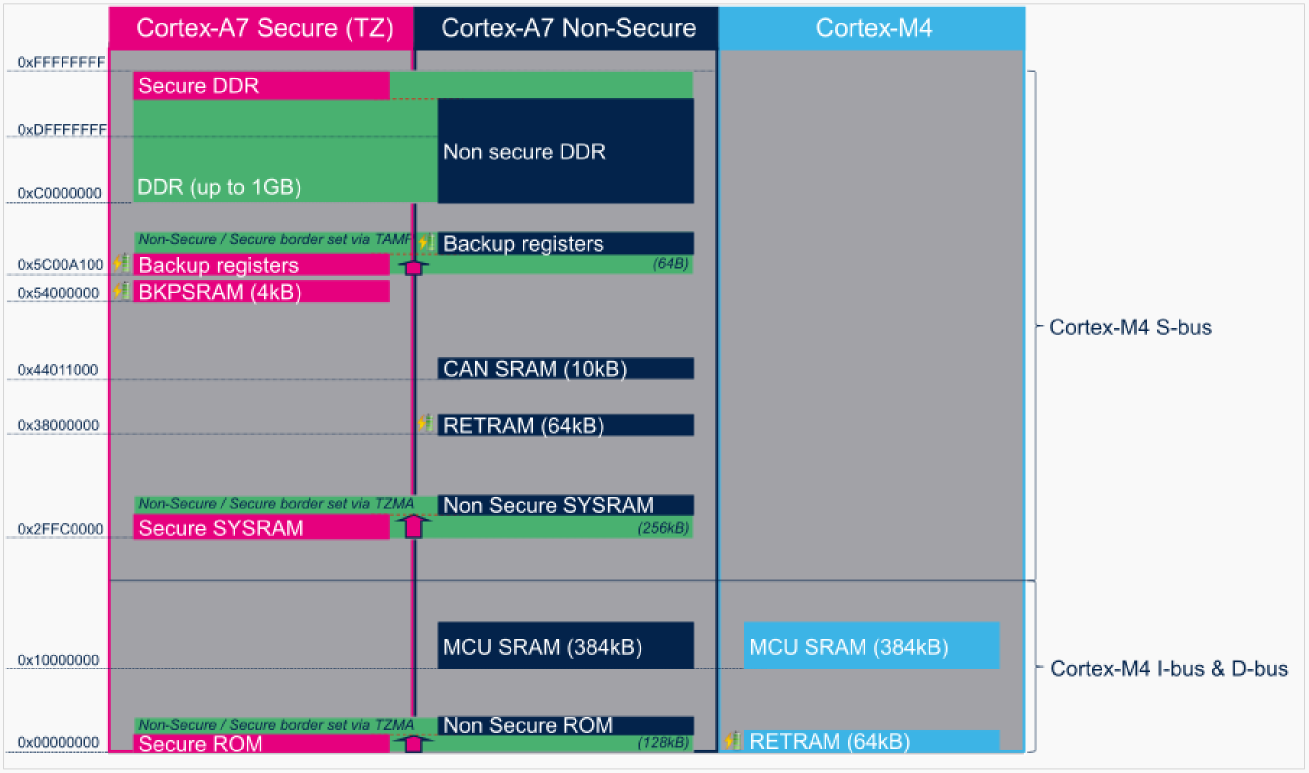
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

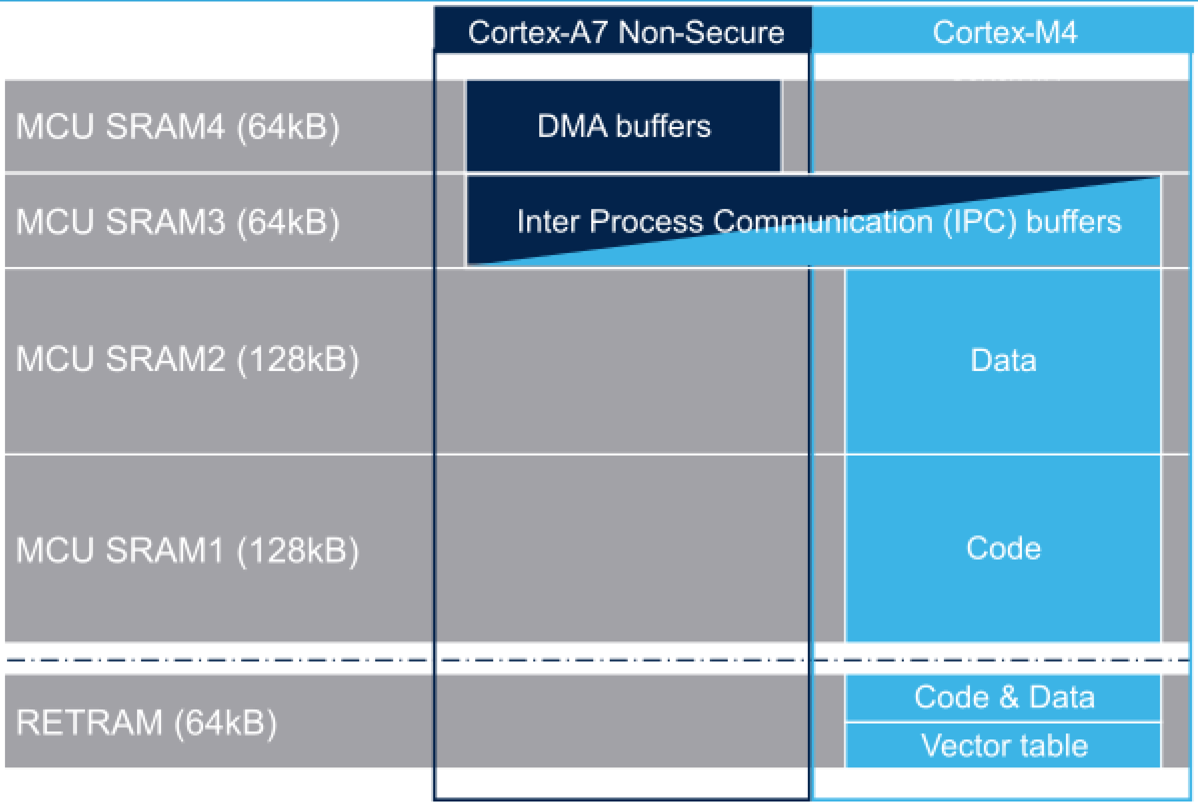
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 MCU SRAM3

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                        <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                        <0x30000000 0x30000000 0x60000>,
                        <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 MCU SRAM1 & SRAM2

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4 
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

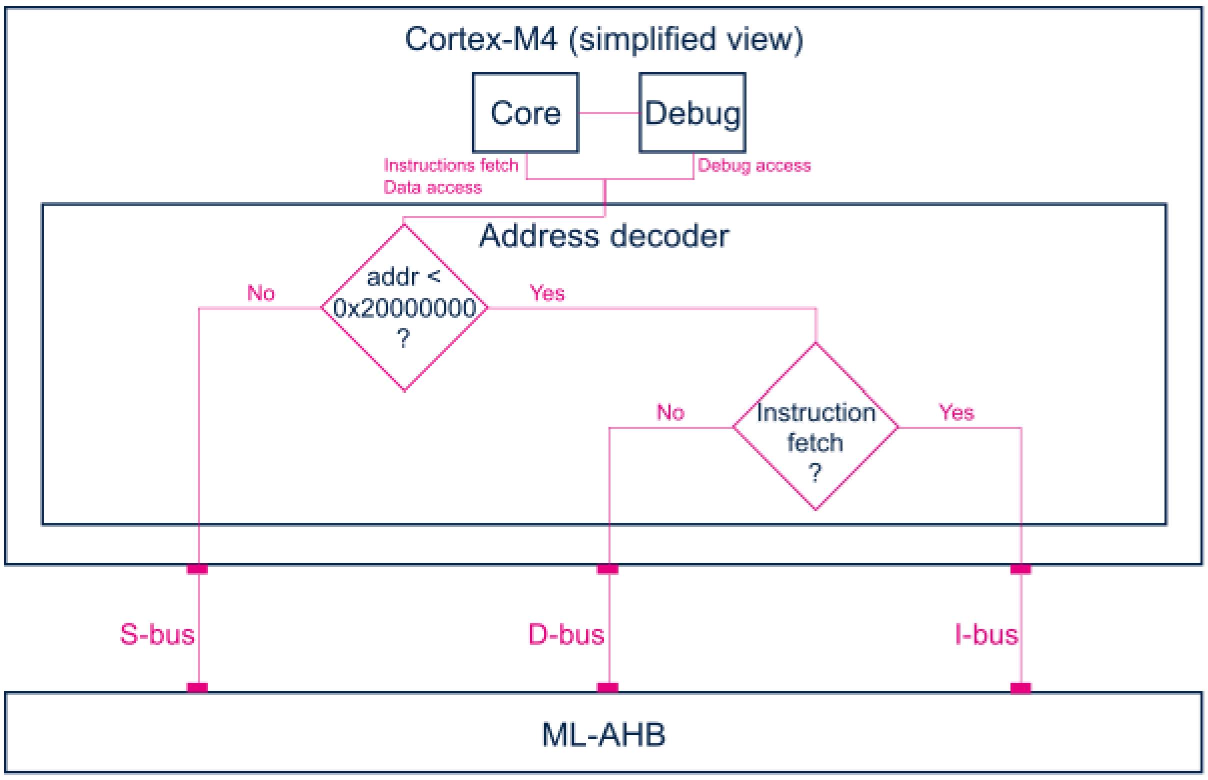
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



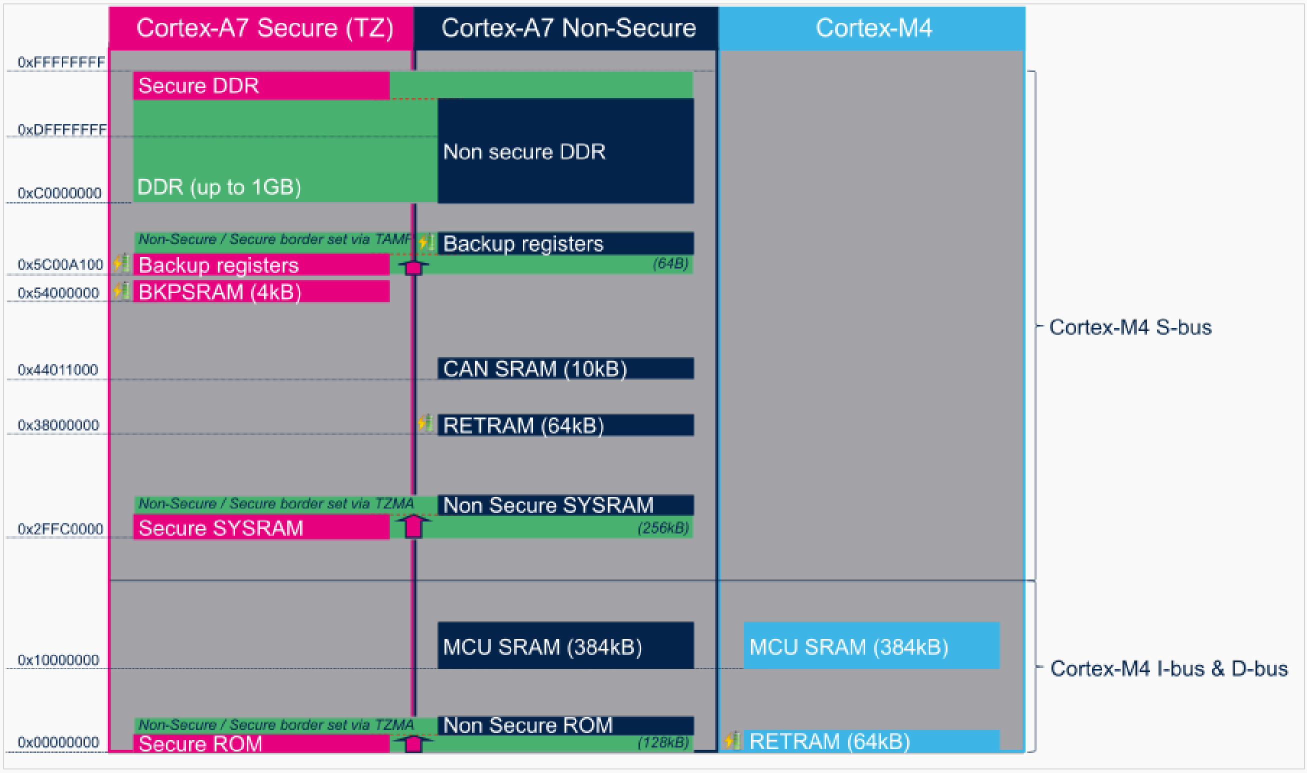
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

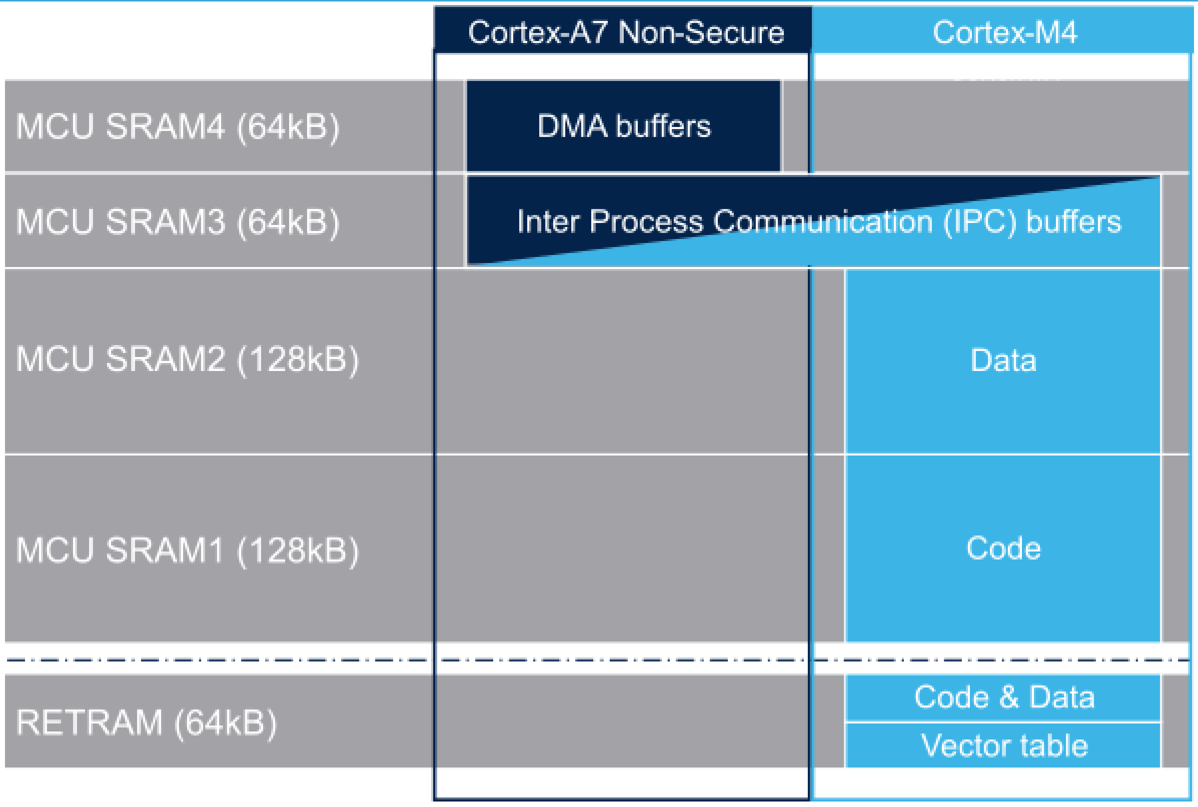
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 *MCU SRAM3*

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                         <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 *MCU SRAM1 & SRAM2*

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                 <0x30000000 0x30000000 0x60000>,
                 <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

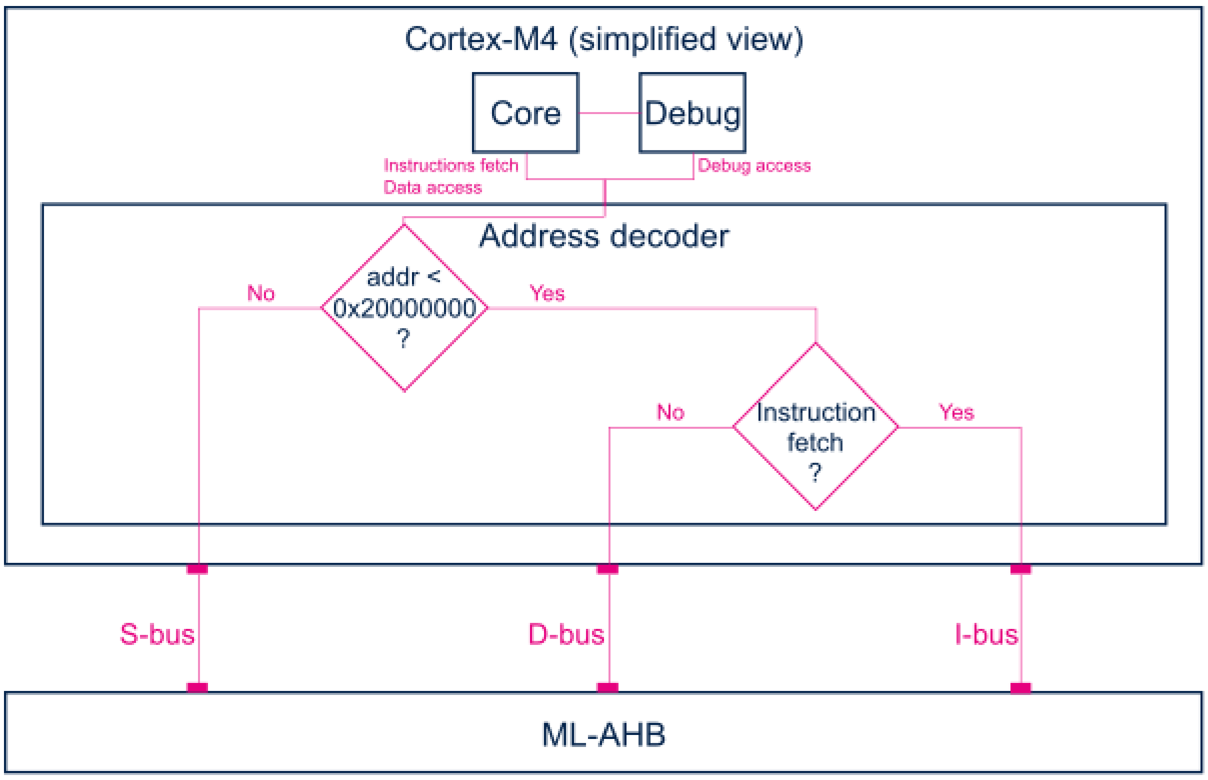
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



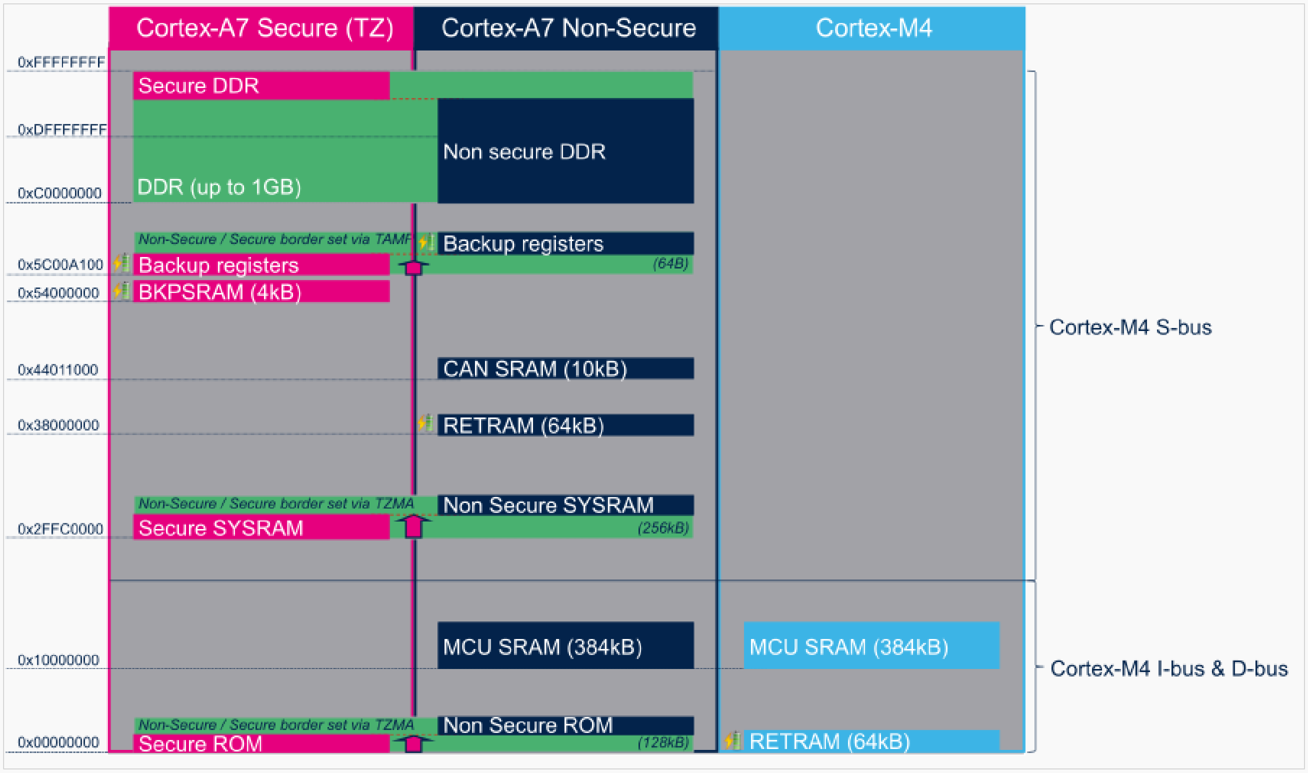
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

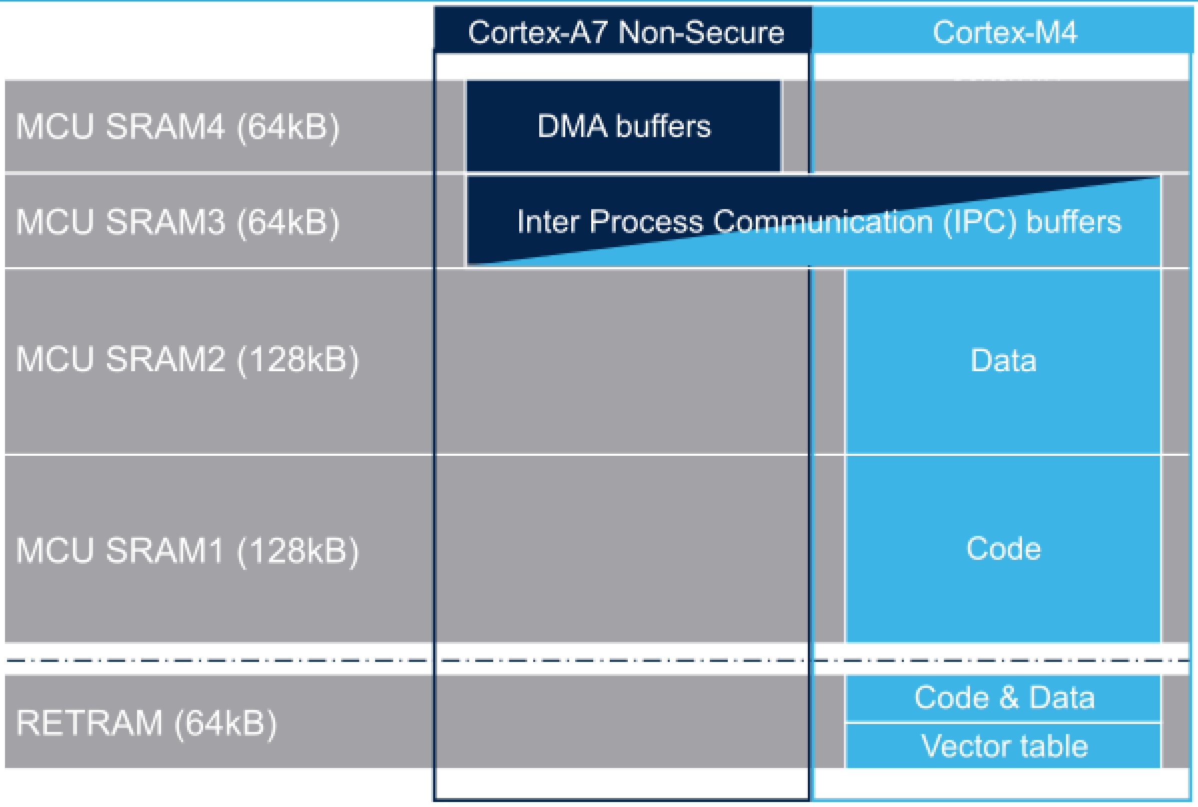
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 MCU SRAM3

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                        <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                        <0x30000000 0x30000000 0x60000>,
                        <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 MCU SRAM1 & SRAM2

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4 
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

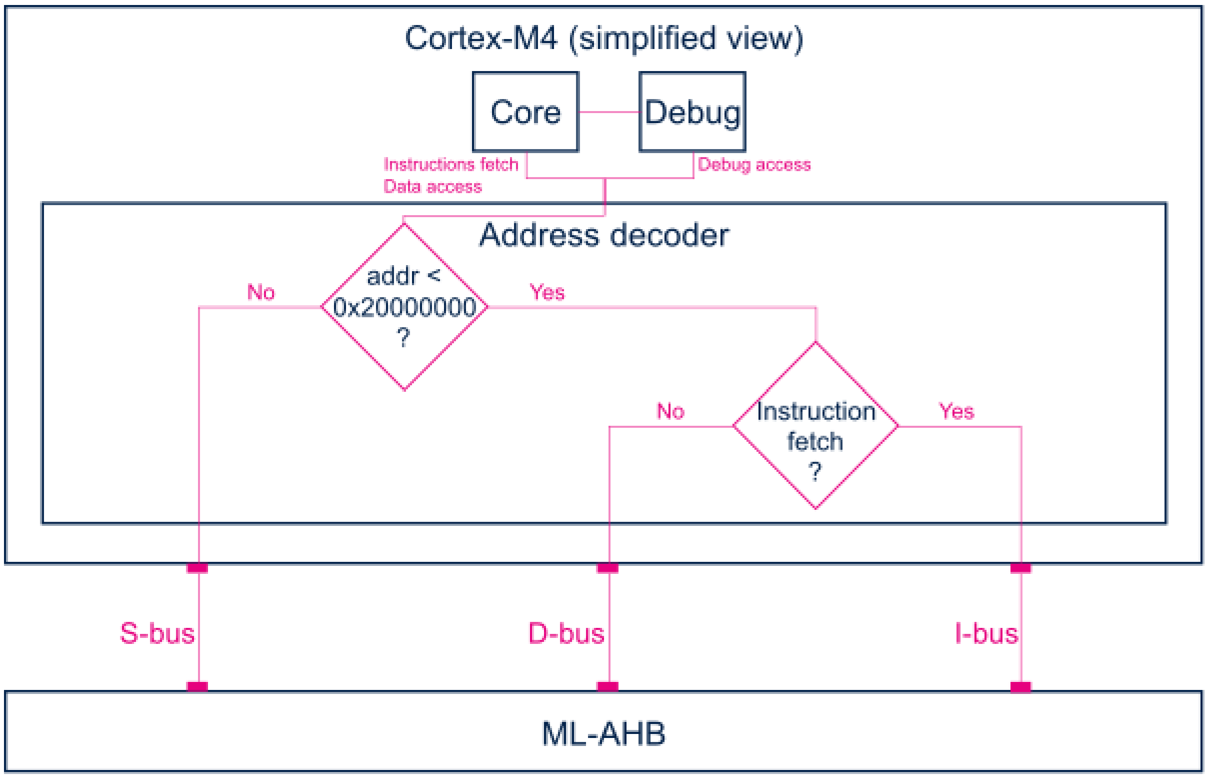
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



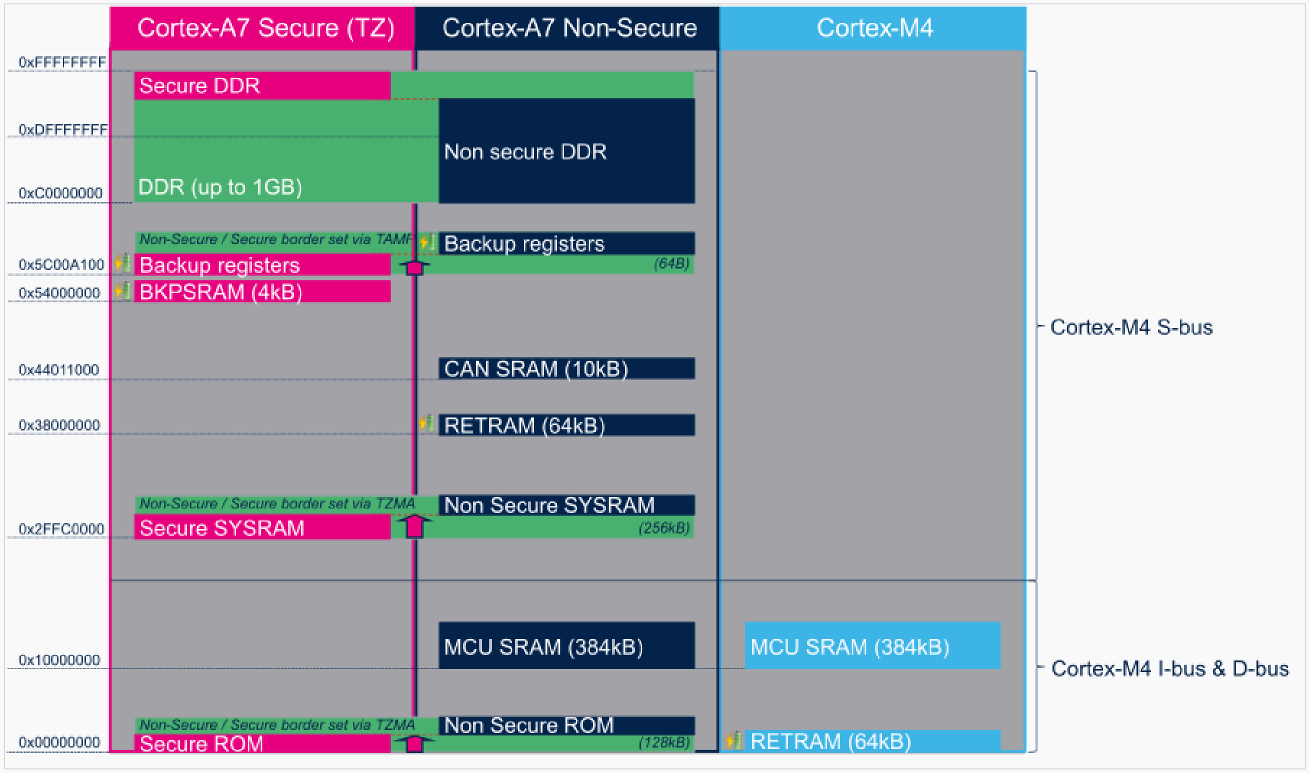
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

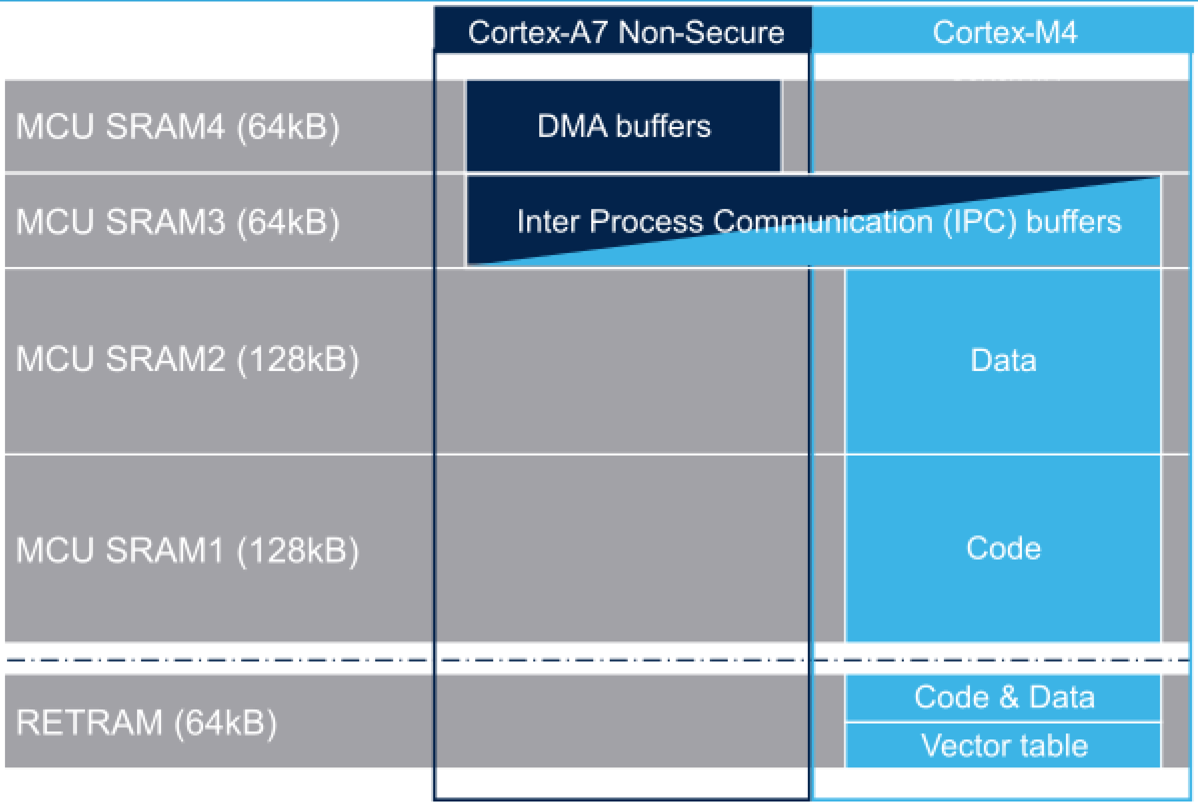
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 MCU SRAM3

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                         <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 MCU SRAM1 & SRAM2

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                 <0x30000000 0x30000000 0x60000>,
                 <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive

Contents

1 Overview

This article shows the default memory mapping defined by STMicroelectronics in STM32MPU Embedded Software. It uses a subset of all memory regions that are exposed at hardware level: customers may use other memory regions or aliases that are not shown here but are described in the STM32MP15 reference manuals.

2 Arm core characteristics

The integration of Arm®Cortex® cores sets some constraints on the device memory mapping: the main ones are listed in this article.

2.1 Reset address

Arm®Cortex® cores start running from address 0x00000000 on reset, which is why this address respectively points to:

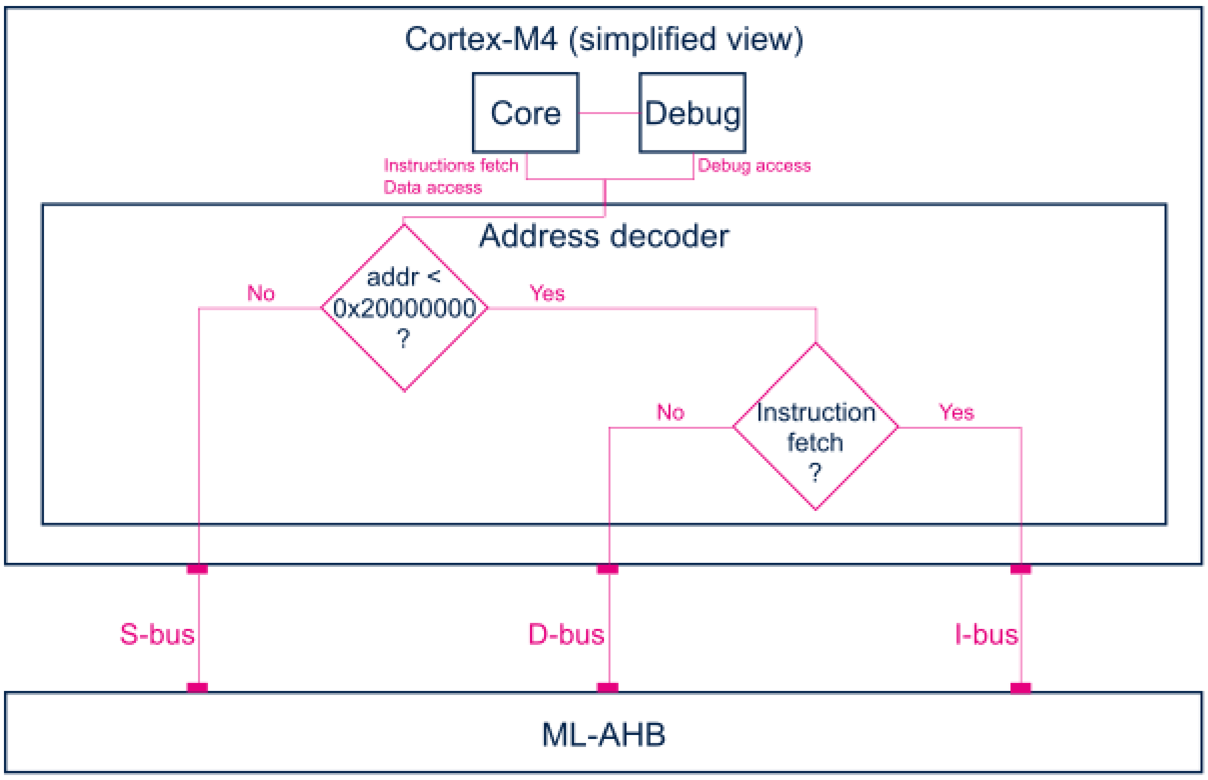
- The ROM code on the Cortex-A7 side. This read-only memory embeds the boot code that is executed when the platform boots (and executes the boot chain) or wakes up from low power STANDBY mode.
- The Retention RAM on the Cortex-M4 side. This needs to be loaded by the Cortex-A7 before releasing the Cortex-M4 reset (in the RCC) and getting it running. This is done by Linux coprocessor management, by default.

Note: since the Cortex-A7 has its ROM code mapped at address 0x00000000, it uses a hardware alias to access the retention RAM at address 0x38000000

2.2 Cortex-M4 multiple ports

The Cortex-M4 is connected to the interconnect (ML-AHB) via three ports, listed below and shown in the following figure:

- I-bus is used to fetch code instructions in the 0x00000000--0x1FFFFFFF address range
- D-bus is used to read/write data in the 0x00000000--0x1FFFFFFF address range
- S-bus is used for all accesses in the 0x20000000--0xFFFFFFFF address range; all STM32MP15 internal peripherals registers are mapped in this range.



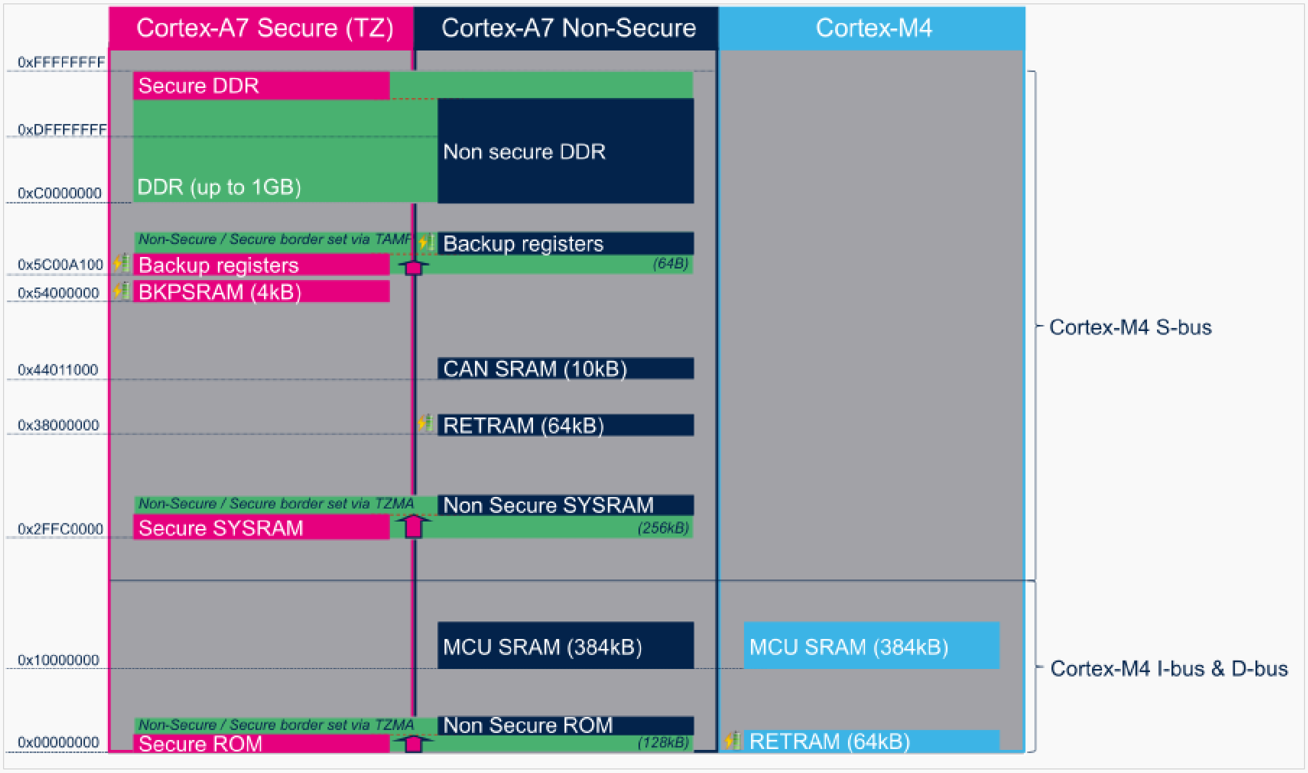
Balancing the Cortex-M4 firmware accesses among those ports allows tuning of the system performance, which is why the MCU SRAM is defined in the first address range (from 0x10000000), but is also visible in the second range (from 0x30000000) in the STM32MP15 reference manuals.

Nevertheless, it is important to notice that the Cortex-M4 embedded in the STM32MP15 only allows hardware breakpoints to be set on the address range covered by the I-bus. Thus any code accessed via the S-bus has to be debugged through software breakpoints.

3 Memory mapping

3.1 Overall memory mapping

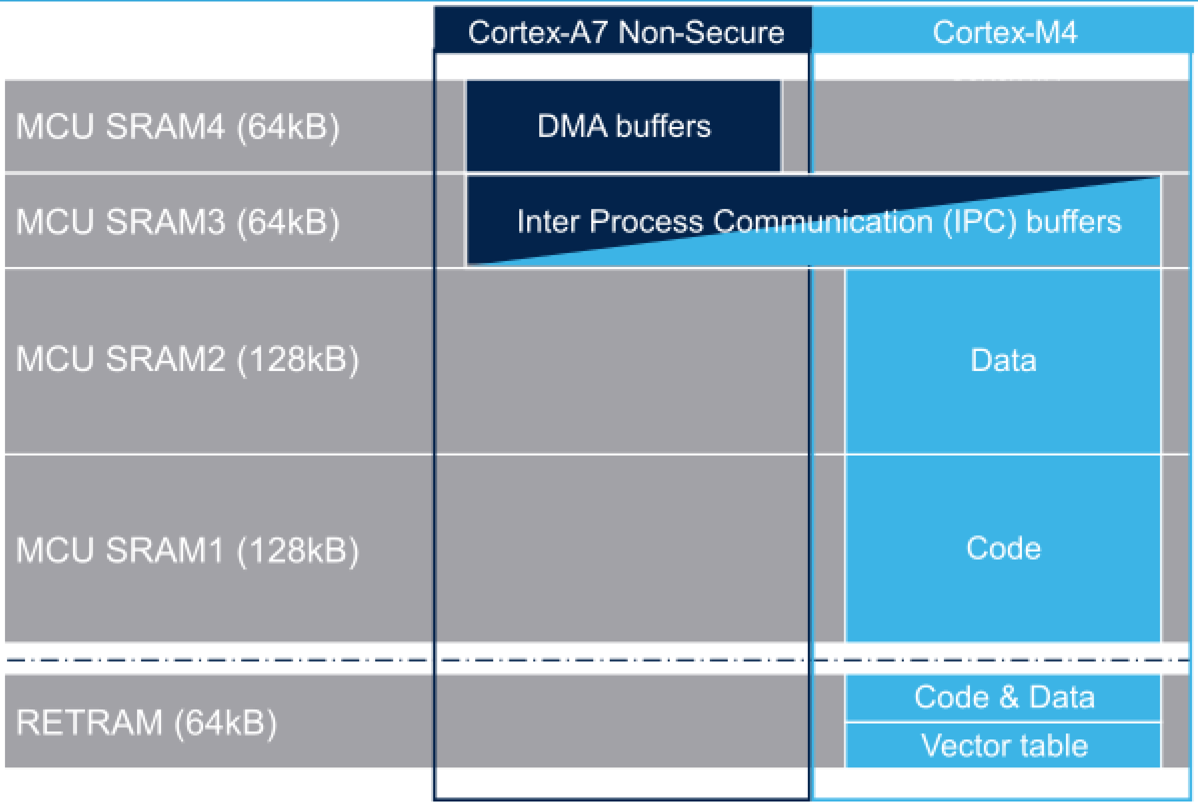
The memory mapping below is a subset of all regions that are exposed at hardware level: it shows the default configuration used in OpenSTLinux but the customer may choose a different mapping to take advantage of other address ranges defined in STM32MP15 reference manuals.



STM32MP15 memory mapping

3.2 Zoom in the Cortex-A7/Cortex-M4 shared memory

The figure below is a zoom of the RAM areas that are shared between the Cortex-A7 non-secure and the Cortex-M4. This mapping is STMicroelectronics' default implementation that can be freely adapted by customers to fit to other needs.



3.3 Possible customization of shared RAM memory

As described in MCU SRAM internal memory, ST has defined a memory mapping to be able to enable all the possible use cases in parallel but this can be customized depending on customer use cases.

3.3.1 Overview

- **RETRAM** is not used except for the vector table which must be stored at address 0. The rest of this memory section can be used for any purpose.
- **MCU SRAM1** (Code) and **SRAM2** (Data) sizes can be tuned depending on user needs to better use physical area of 256KB.
- **MCU SRAM3** (IPC Buffers) can be used for other purpose if IPC is not used
- **MCU SRAM4** (DMA) can be used for other purpose if DMA1 and/or DMA2 is not used in chained mode (using MDMA)

3.3.2 How to do that in practice

The memory usage is defined in both Cortex contexts:

- on Cortex-A7: in Linux device tree, using Reserved_memory mechanism
- on Cortex-M4: in the linker script

To ensure the consistency of the system, both memory declarations have to be updated according to the expected configuration.

3.3.2.1 RETRAM

By default the RETRAM is reserved for the Cortex-M4 firmware and only the vector table uses it. The base address is fixed, the vector table section of the M4 firmware needs to be at this place but other sections may be added on top.

- Linux Device tree

  "retram" memory region declaration (no update needed):

```
retram: retram@0x38000000 {
        compatible = "shared-dma-pool";
        reg = <0x38000000 0x10000>;
        no-map;
};
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        ranges =  <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  In STM32Cube linker script definition (.ld): keep vector table "m_interrupts" but any new section can be added on top to use RETRAM free memory space:

```
MEMORY
{
    m_interrupts (RX)   : ORIGIN = 0x00000000, LENGTH = 0x00000298
    m_any_section .... : ..................
}
```

3.3.2.2 MCU SRAM4

By default the MCU SRAM4 is reserved for DMA chaining for Linux features. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings declarations:

```
sram: sram@10050000 {
        compatible = "mmio-sram";
        reg = <0x10050000 0x10000>;
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x10050000 0x10000>;

        dma_pool: dma_pool@0 {
                reg = <0x0 0x10000>;
                pool;
        };
};
```

```
&dma1 {
        sram = <&dma_pool>;
};

&dma2 {
        sram = <&dma_pool>;
};
```

3.3.2.3 MCU SRAM3

By default the MCU SRAM3 is reserved for the IPC. It can be freed for some other purposes by removing following:

- Linux device tree

  Remove followings memory region declarations:

```
                vdev0vring0: vdev0vring0@10040000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10040000 0x2000>;
                        no-map;
                };

                vdev0vring1: vdev0vring1@10042000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10042000 0x2000>;
                        no-map;
                };

                vdev0buffer: vdev0buffer@10044000 {
                        compatible = "shared-dma-pool";
                        reg = <0x10044000 0x4000>;
                        no-map;
                };
```

  Remove associated reference in m4_rproc node:

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>,
                        <&vdev0vring0>, <&vdev0vring1>, <&vdev0buffer;
        ranges =  <0x00000000 0x38000000 0x10000>,
                        <0x30000000 0x30000000 0x60000>,
                        <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware:

  Must be aligned with STM32Cube linker script definition (.ld), remove "m_ipc_shm":

```
MEMORY
{
    m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

3.3.2.4 MCU SRAM1 & SRAM2

By default the MCU SRAM1 & SRAM2 are reserved for the STM32Cube firmware. This can be optimized depending on the firmware needs.

- Linux device tree

  Only one section is declared for STM32Cube firmware code and data in Linux device tree

  Notice that the MCURAM is aliased so accessible at addresses 0x10000000 or 0x30000000. In consequence mcuram and mcuram2 memory sections definitions have to be coherent.

```
        mcuram: mcuram@ 0x30000000 {
                compatible = "shared-dma-pool";
                reg =  <0x30000000 0x40000>; /* define memory base and size for Cortex-M4
firmware*/
                no-map;
        };
        mcuram2: mcuram2@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>; /* define memory base and size for Cortex-M4 
firmware*/
                no-map;
        };
```

No update needed for m4_proc node.

```
&m4_rproc {
        memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;

        ranges = <0x00000000 0x38000000 0x10000>,
                         <0x30000000 0x30000000 0x60000>,
                         <0x10000000 0x10000000 0x60000>;
```

- Cortex-M4 STM32Cube firmware

  The memory mapping you have defined on Linux side (freed for M4 usage) needs to be consistent with STM32Cube linker script definition (.ld):

```
MEMORY
{
 m_interrupts (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000298
 m_text       (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00020000
 m_data       (RW)  : ORIGIN = 0x10020000, LENGTH = 0x00020000
 m_ipc_shm    (RW)  : ORIGIN = 0x10040000, LENGTH = 0x00008000
}
```

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.

Read Only Memory

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.)

Advanced High-performance Bus

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.)

Inter-Processor Communication

Direct Memory Access

Receive