# MediaTek LinkIt™ Development Platform for RTOS Memory Layout Developer's Guide

Version: 1.0

Release date: 8 September 2017

# Document Revision History

| Revision | Date | Description |
|---|---|---|
| 1.0 | 8 Sep 2017 | Added MT2625 memory layout description |

# Table of Contents

# Lists of Tables and Figures

# 1. Overview

This document provides details on the memory layout design and configuration of MediaTek LinkIt™ development platform for real-time operating system (RTOS). The platform includes:

- MT2625 chipsets：MT2625D

Each memory layout has two types of views, load view and an execution view. The design concept will be described based on the two views:

- Load view describes a memory region and section of each image in terms of the address it is located at before the image is processed.
- Execution view describes a memory region and section of each image in terms of the address it is located at during the image execution.

Different toolchains have different layout configuration files. The GCC toolchain uses a linker script, the ARMCC toolchain uses a scatter file. The memory layout configuration will be described separately for each toolchain.

# 2. Memory Layout and Configuration for MT2625

MT2625 supports five types of physical memory – Serial Flash, Pseudo Static Random Access Memory (PSRAM), System Random Access Memory (SYSRAM), RTC Retention SRAM (RETSRAM) and Tightly Coupled Memory (TCM). The memory layouts are designed based on the five types of memory.

***The virtual memory on the MT2625 is provided for cacheable memory and is implemented based on the memory mapping mechanism of ARM Cortex-M4 MCU. The virtual address range from 0x10000000 to 0x14000000 is mapped to the PSRAM address range from 0x00000000 to 0x04000000, as shown in Figure 1 MT2625 virtual memory mapping***

. The virtual memory region (`0x10000000` ~ `0x14000000`) is used as cacheable memory. All read-write (RW) data is stored in this region, by default. RW data is stored in the first virtual memory region, by default.

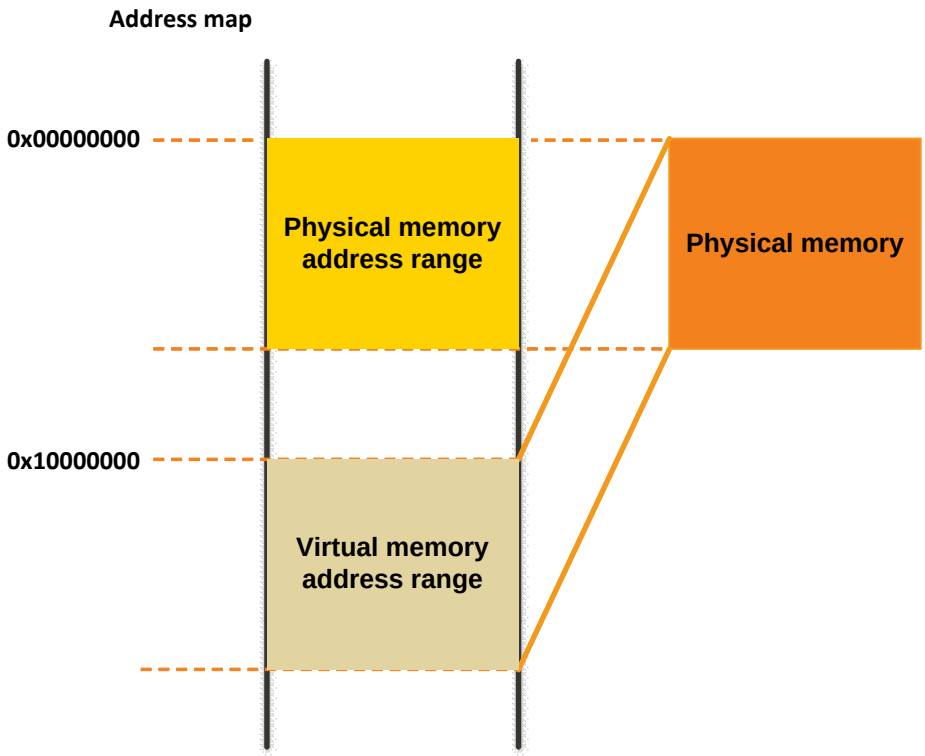


***Figure 1 MT2625 virtual memory mapping***

The memory layout can be defined with FOTA and without FOTA.

This section guides you through:

- Types of the memory layout
- Programming guide
- Memory layout adjustment with a
  - Linker script
  - Scatter file

- IAR configuration file

## 2.1. Memory layout without FOTA

### 2.1.1. Load view

***MT2625 has 4MB internal serial flash memory. The load view on the flash memory with disabled FOTA for MT2625 is shown in Figure 2 Load view of the MT2625 memory layout without FOTA***

2.

- Header 1. Always located at the very beginning of the flash memory and reserved for bootloader security information. The size of the Header 1 is not configurable and is fixed to 4kB.
- Header 2. Reserved for RTOS binary security information. The size of the Header 2 is not configurable and is fixed to 4kB.
- Bootloader. The size of the bootloader is not configurable and is fixed to 64kB size.
- ARM Cortex-M4 firmware. This section of the memory is reserved for the RTOS binary and modem (MD) firmware. Note, that RTOS firmware and MD firmware are isolated in the flash, but are provided as a single binary to the user. MD firmware follows the RTOS firmware.
- The end of the flash is a reserved buffer for NVDM buffer, the size of the NVDM buffer is configurable.

The start address and the maximum size of each binary and reserved buffer are configurable, see section 2.4, "*Memory layout adjustment with a linker script*", for more details.

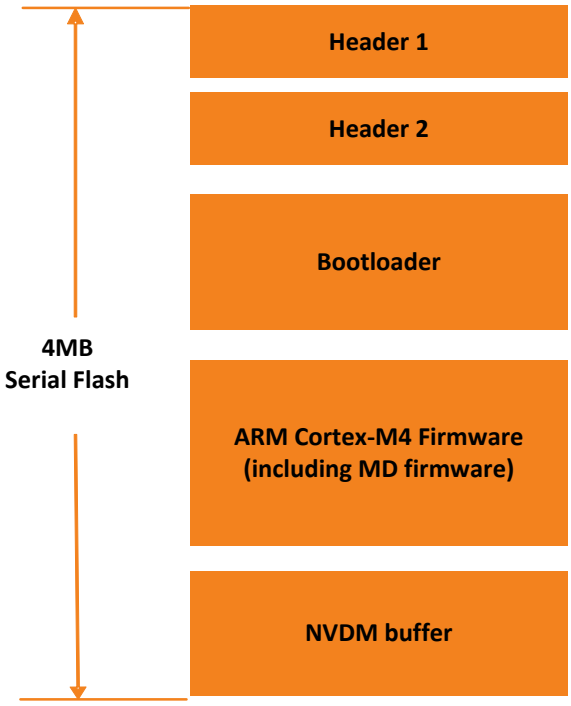


***Figure 2 Load view of the MT2625 memory layout without FOTA***

For more information about FOTA, refer to MediaTek LinkIt Development Platform for RTOS Firmware Update Developer's Guide located under SDK/doc folder.

For more information about NVDM, refer to MediaTek LinkIt Development Platform for RTOS API reference guide.

### 2.1.2. Execution view

***Execution view describes where the code and data are located during the program runtime, as shown in Figure 2 Load view of the MT2625 memory layout without FOTA***

3 for MT2625. The execution view is based on the Serial Flash, PSRAM, SYSRAM, RERSRAM and TCM, as described below:

- Serial Flash. The code and read-only (RO) data are located in the flash memory during runtime.
- PSRAM.
  - PSRAM code and RO data. The PSRAM code and RO data is cacheable.
  - Cacheable RW data and ZI data.
  - Non-cacheable RW data and ZI data.
  - MD PSRAM code and MD RO data. The MD PSRAM code and MD RO data is cacheable.
  - MD Cacheable RW data and ZI data.
  - MD Non-cacheable RW data and ZI data.
- SYSRAM.
  - SYSRAM RO data. The RO data is non-cacheable.
  - SYSRAM RW data and ZI data. The SYSRAM RW data and ZI data is non-cacheable.
- TCM. Some critical and high-performance code or data can be stored into the TCM. See section 2.3, "*Programming guide*" to learn how to put code or data to the TCM.
  - Vector table, single bank code and some high-performance code and data are stored at the beginning of TCM.
  - Code and RO data.
  - RW data and ZI data.
  - The system stack.
  - MD code and MD RO data.
  - MD RW data and MD ZI data.
- RETSRAM.
  - RETSRAM RO data. The RO data is cacheable
  - RETSRAM RW data. The RW data is cacheable
  - RETSRAM ZI data. The ZI data is cacheable

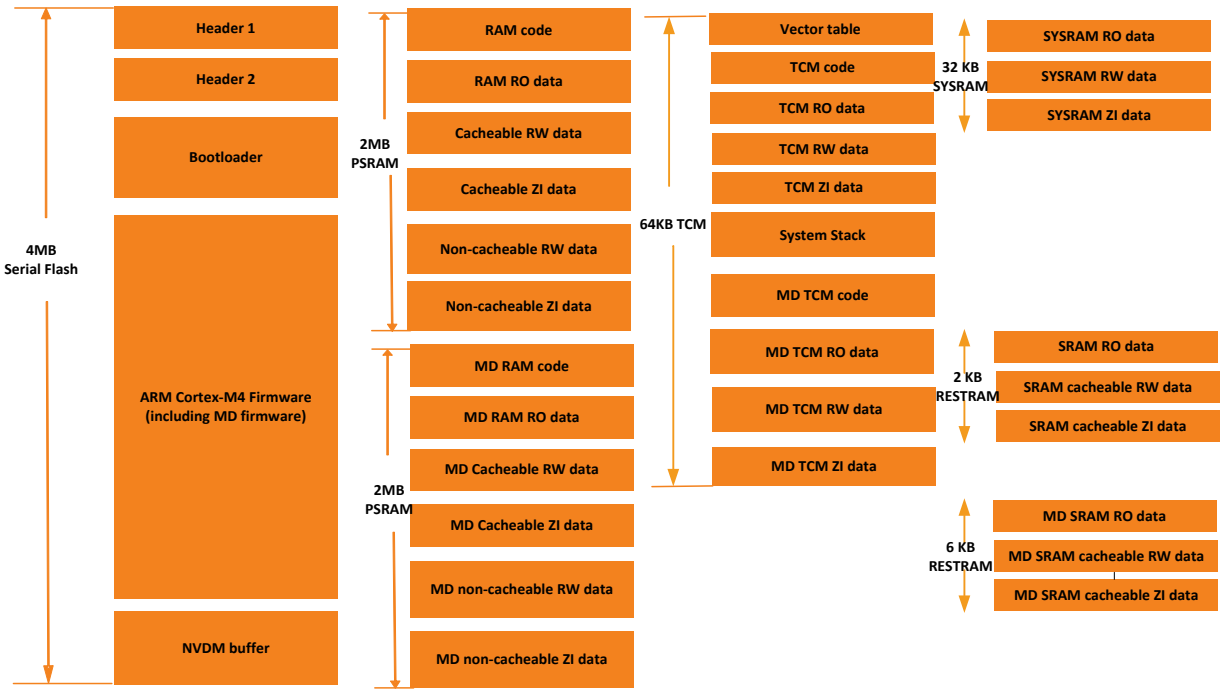


***Figure 3 Execution view of the MT2625 memory layout without FOTA***

## 2.2. Memory layout with FOTA of full binary update

### 2.2.1. Load view

The flash memory layout's load view with enabled FOTA is shown in *Figure 4 Load view of the MT2625 memory layout with full binary FOTA* for MT2625. A FOTA buffer is added for temporary storage of the binary that will be used to update the current ARM Cortex-M4 firmware. The start address and maximum size of each binary and the reserved space of certain memory layouts are configurable, see section 2.4, "*Memory layout adjustment with a linker script*", for more details. To enable FOTA, refer to the MediaTek LinkIt Development Platform for RTOS Firmware Update Developer's Guide located under the SDK /doc folder.

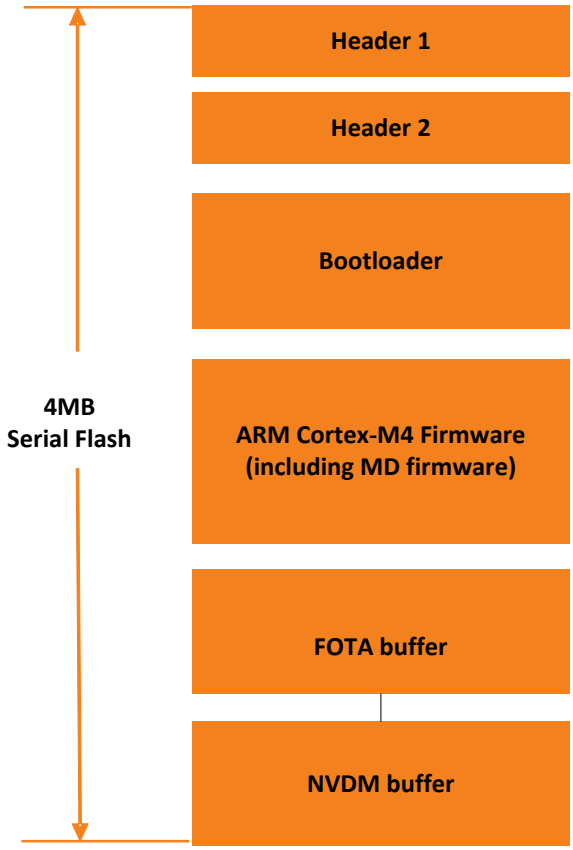


***Figure 4 Load view of the MT2625 memory layout with full binary FOTA***

### 2.2.2. Execution view

Execution view describes where the code and data are located during the program runtime, as shown in *Figure 5 Execution view of the MT2625 memory layout with full binary FOTA* for MT2625. The execution view is based on the Serial Flash, PSRAM, SYSRAM, RERSRAM and TCM, as described below:

- Serial Flash. The code and read-only (RO) data are located in the flash memory during runtime.
- PSRAM.
  - PSRAM code and RO data. The PSRAM code and RO data is cacheable.
  - Cacheable RW data and ZI data.
  - Non-cacheable RW data and ZI data.
  - MD PSRAM code and MD RO data. The MD PSRAM code and MD RO data is cacheable.
  - MD cacheable RW data and ZI data.
  - MD non-cacheable RW data and ZI data.
- SYSRAM.
  - SYSRAM RO data. The RO data is non-cacheable.
  - SYSRAM RW data and ZI data. The SYSRAM RW data and ZI data is non-cacheable.
- TCM. Some critical and high-performance code or data can be stored into the TCM. See section 2.3, "*Programming guide*" to learn how to put code or data to the TCM.

- Vector table, single bank code, and some high-performance code and data are stored at the beginning of TCM.
- Code and RO data.
- RW data and ZI data.
- The system stack.
- MD code and MD RO data.
- MD RW data and MD ZI data.

- RETSRAM.
  - RETSRAM RO data. The RO data is cacheable
  - RETSRAM RW data. The RW data is cacheable
  - RETSRAM ZI data. The ZI data is cacheable

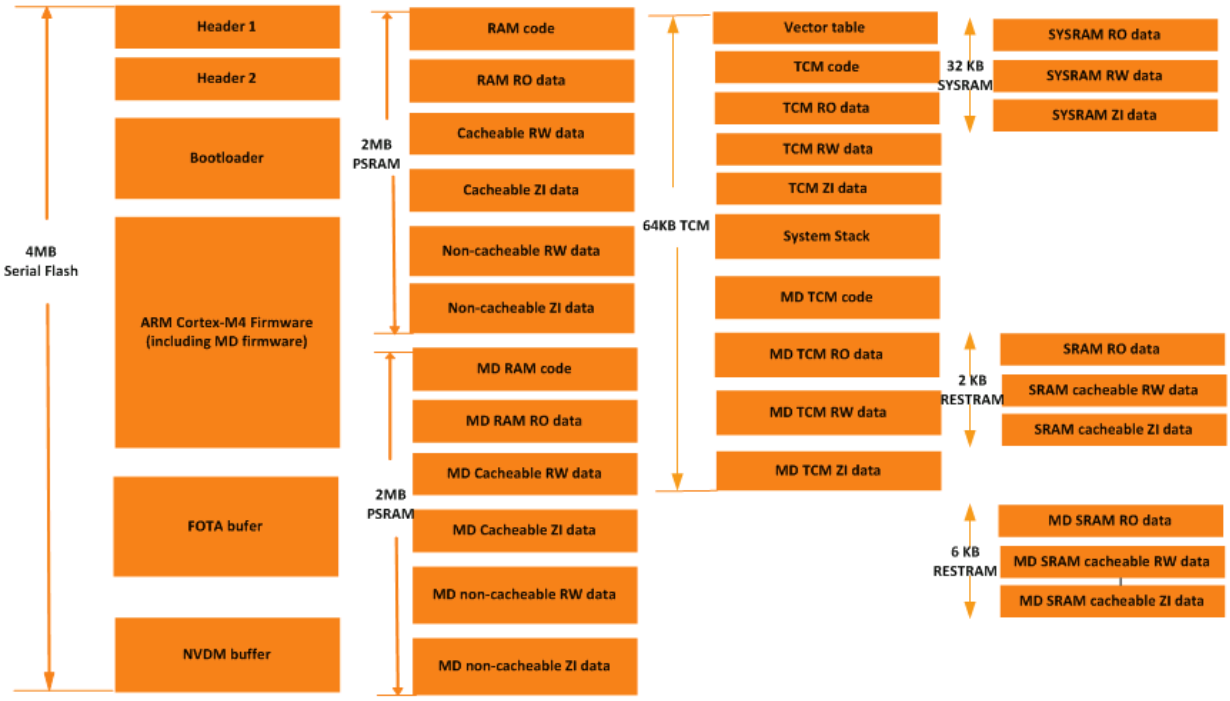


***Figure 5 Execution view of the MT2625 memory layout with full binary FOTA***

## 2.3. Programming guide

This programming guide is based on the memory layout described in section 2.1.2, "*Execution view*".

Examples to place the code successfully to required memory location during runtime are shown below.

1) Place the code or RO data to the Serial Flash at runtime.

By default, the code or RO data is placed in the flash, execute in place (XIP), no need to modify.

1) Place the code or RO data to the PSRAM at runtime.

To run the code or access RO data in the PSRAM with higher efficiency, specify the attribute explicitly in your code, as shown in the example below.

```
//code
ATTR_TEXT_IN_RAM int func(int par)
{
   int s;
   s = par;
   //....
}
//RO data
ATTR_RODATA_IN_RAM const int b = 8;
```

For comparison, if the attribute is not explicitly defined, during the function call the code will be placed in the Serial Flash instead of the PSRAM.

```
//code
int func(int par)
{
   int s;
   s = par;
   //....
}
//RO data
const int b = 8;
```

2) Place RW data or ZI data to PSRAM non-cacheable memory at runtime.

To access RW data and ZI data in the non-cacheable memory with special purpose, such as direct memory access (DMA) buffer, specify the attribute explicitly in your code, as shown in the example below.

```
//RW data
ATTR_RWDATA_IN_NONCACHED_RAM int b = 8;
//ZI data
ATTR_ZIDATA_IN_NONCACHED_RAM int b;
```

For comparison, if the attribute is not explicitly defined, the data will be placed in the PSRAM cacheable memory instead of the non-cacheable memory.

```
//RW data
int b = 8;

//ZI data
int b;
```

3) Place RW data or ZI data to PSRAM cacheable memory at runtime.

By default, RW data/ZI data are placed in the cacheable memory, no need to modify.

4) Place the RO data to the SYSRAM at runtime.

To access RO data in the SYSRAM with better performance, specify the attribute explicitly in your code, as shown in the example below.

```
//RO data
ATTR_RODATA_IN_SYSRAM const int b = 8;
```

For comparison, if the attribute is not explicitly defined, during the function call the code will be placed in the Serial Flash instead of the SYSRAM.

```
//RO data
const int b = 8;
```

5) Place RW data or ZI data to SYSRAM cacheable memory at runtime.

To access RW data and ZI data in the SYSRAM cacheable memory, specify the attribute explicitly in your code, as shown in the example below.

```
//RW data
ATTR_RWDATA_IN_CACHED_SYSRAM int b = 8;
//ZI data
ATTR_ZIDATA_IN_CACHED_SYSRAM int b;
```

For comparison, if the attribute is not explicitly defined, the data will be placed in the PSRAM cacheable memory instead of the non-cacheable memory, because RW and ZI are default in PSRAM cacheable memory.

```
//RW data
int b = 8;

//ZI data
int b;
```

6) Place code or RO data to the TCM at runtime.

To run the code or access RO data in the TCM with better performance, specify the attribute explicitly in your code, as shown in the example below.

```
//code
ATTR_TEXT_IN_TCM int func(int par)
{
   int s;
   s = par;
   //....
}
//RO data
ATTR_RODATA_IN_TCM const int b = 8;
```

For comparison, if the attribute is not explicitly defined, during the function call the code will be placed in the Serial Flash instead of the TCM.

```
//code
int func(int par)
{
   int s;
   s = par;
   //....
}
//RO data
const int b = 8;
```

7) Put RW data/ZI data to TCM at runtime.

To access RW data and ZI data in the TCM with higher efficiency, specify the attribute explicitly in your code, as shown in the example below.

```
//rw-data
ATTR_RWDATA_IN_TCM int b = 8;
```

```
//zi-data
ATTR_ZIDATA_IN_TCM int b;
```

For comparison, if the attribute is not explicitly defined, the data will be placed in the PSRAM instead of the TCM.

```
//RW data
int b = 8;
//ZI data
int b;
```

8) Place code or RO data to the RETSRAM at runtime.

To run the code or access RO data in the RETSRAM with higher efficiency, specify the attribute explicitly in your code, as shown in the example below.

```
//RO data
ATTR_RODATA_IN_RETSRAM const int b = 8;
```

For comparison, if the attribute is not explicitly defined, during the function call the code will be placed in the Serial Flash instead of the RETSRAM.

```
//RO data
const int b = 8;
```

9) Put RW data/ZI data to RETSRAM at runtime.

To access RW data and ZI data in the RETSRAM with higher efficiency, specify the attribute explicitly in your code, as shown in the example below.

```
//rw-data
ATTR_RWDATA_IN_RETSRAM int b = 8;
//zi-data
ATTR_ZIDATA_IN_RETSRAM int b;
```

For comparison, if the attribute is not explicitly defined, the data will be placed in the PSRAM instead of the RETSRAM.

```
//RW data
int b = 8;
//ZI data
int b;
```

10) Place the MD code or RO data to the PSRAM at runtime.

To run the code or access MD RO data in the PSRAM with better performance, specify the attribute explicitly in your code, as shown in the example below.

```
//code
ATTR_MD_TEXT_IN_RAM int func(int par)
{
   int s;
   s = par;
   //....
}
//RO data
ATTR_MD_RODATA_IN_RAM const int b = 8;
```

For comparison, if the attribute is not explicitly defined, during the function call the code will be placed in the Serial Flash instead of the PSRAM.

```
//code
int func(int par)
{
   int s;
   s = par;
   //....
}
//RO data
const int b = 8;
```

11) Place RW data or ZI data to MD PSRAM non-cacheable memory at runtime.

To access RW data and ZI data in the non-cacheable memory with special purpose, such as direct memory access (DMA) buffer, specify the attribute explicitly in your code, as shown in the example below.

```
//RW data
ATTR_MD_RWDATA_IN_NONCACHED_RAM int b = 8;
//ZI data
ATTR_MD_ZIDATA_IN_NONCACHED_RAM int b;
```

For comparison, if the attribute is not explicitly defined, the data will be placed in the PSRAM cacheable memory instead of the non-cacheable memory.

```
//RW data
int b = 8;

//ZI data
int b;
```

12) Place RW data or ZI data to MD PSRAM cacheable memory at runtime.

```
//RW data
ATTR_MD_RWDATA_IN_RAM int b = 8;
//ZI data
ATTR_MD_ZIDATA_IN_RAM int b;
```

13) Place code or RO data to the MD TCM at runtime.

To run the code or access RO data in the MD TCM with higher efficiency, specify the attribute explicitly in your code, as shown in the example below.

```
//code
ATTR_MD_TEXT_IN_TCM int func(int par)
{
   int s;
   s = par;
   //....
}
//RO data
ATTR_MD_RODATA_IN_TCM const int b = 8;
```

For comparison, if the attribute is not explicitly defined, during the function call the code will be placed in the Serial Flash instead of the TCM.

```
//code
int func(int par)
{
   int s;
   s = par;
   //....
}
//RO data
const int b = 8;
```

14) Put RW data/ZI data to MD TCM at runtime.

To access RW data and ZI data in the TCM with higher efficiency, specify the attribute explicitly in your code, as shown in the example below.

```
//rw-data
ATTR_MD_RWDATA_IN_TCM int b = 8;
//zi-data
ATTR_MD_ZIDATA_IN_TCM int b;
```

For comparison, if the attribute is not explicitly defined, the data will be placed in the PSRAM instead of the TCM.

```
//RW data
int b = 8;
//ZI data
int b;
```

15) Place code or RO data to the MD RETSRAM at runtime.

To run the code or access RO data in the MD RETSRAM with better performance, specify the attribute explicitly in your code, as shown in the example below.

```
//RO data
ATTR_MD_RODATA_IN_RETSRAM const int b = 8;
```

For comparison, if the attribute is not explicitly defined, during the function call the code will be placed in the Serial Flash instead of the MD RETSRAM.

```
//RO data
const int b = 8;
```

16) Put RW data/ZI data to MD RETSRAM at runtime.

To access RW data and ZI data in the MD RETSRAM with higher efficiency, specify the attribute explicitly in your code, as shown in the example below.

```
//rw-data
ATTR_MD_RWDATA_IN_RETSRAM int b = 8;
//zi-data
ATTR_MD_ZIDATA_IN_RETSRAM int b;
```

For comparison, if the attribute is not explicitly defined, the data will be placed in the PSRAM instead of the MD RETSRAM.

```
//RW data
int b = 8;
//ZI data
```

```
  int b;
```

## 2.4. Memory layout adjustment with a linker script

The memory layout can be configured with different toolchains. When the code is built based on the GCC toolchain, the memory layout description file called a linker script is required. When the code is built based on ARMCC toolchain, the memory layout description file called a scatter file is used.

This section describes how to use the linker script provided by MediaTek and how to configure it when building code with the GCC toolchain. The scatter file is introduced in section 2.5, "Memory layout adjustment with a scatter file".

### 2.4.1. Types of linker scripts

Two types of linker scripts are provided:

- Template linker script – every application linker script should be based on the template linker script.
- Application linker script – every application has its particular linker script. This linker script is passed to the linker at linking stage.

#### 2.4.1.1. Template linker script

Template linker scripts are based on the memory layout see section 2.1 and 2.2. If the memory layout is modified, the linker script should also be modified manually. It's recommended to use the layout and linker scripts provided by MediaTek as a reference for your customizations.

The template linker scripts are located under `/driver/CMSIS/Device/MTK/<chip>/linkerscript/GCC/` folder.

The folder includes:

- `default`. This folder contains a template linker script to build a project without FOTA memory layout, see section 2.1, "*Memory layout without FOTA*".
- `full_bin_fota`. This folder contains a template linker script to build a project with full binary FOTA memory layout, see section 2.2, "*Memory layout with FOTA of full binary update*"
- `sysram.` This folder contains a template linker script to enable RAM debugging. To place all your code into SYSRAM, use this linker script as a reference.

#### 2.4.1.2. Application linker script

The application linker script is located under `/project/<board>/apps/<project>/GCC/` folder. Each application has its own linker script and each linker script can have a different memory layout configuration based on the application requirements.

### 2.4.2. How to use the linker script

To create a new linker script file for your application:

- Clone a linker script from the template folder.
- Create a new linker script manually. The memory layout in this case should also be user-defined to match your linker script.

#### 2.4.2.1. Cloning the linker script

To clone a linker script from the template:

1) Specify the memory layout feature for your application development, such as without FOTA, full binary FOTA see section 2.1 and 2.2.
2) Copy the template linker script from template folder to your application project's folder, see section 2.4.1, "*Types of linker scripts*".
3) Memory layout without FOTA.

Copy `/driver/CMSIS/Device/MTK/<chip>/linkerscript/GCC/default` to `/project/<board>/apps/<project>/GCC/`.

4) Memory layout with FOTA full binary update.

Copy `/driver/CMSIS/Device/MTK/<chip>/linkerscript/GCC/full_bin_fota` to `/project/<board>/apps/<project>/GCC/`.

5) Modify the linker script according to the application requirements.

### 2.4.3. Rules to adjust the memory layout

The memory layout can be customized to fit the application requirements. However, the sections for Header 1, Header 2 and bootloader are not configurable. The rest of the memory layout can be adjusted.

Common rules for different memory layout adjustment settings are described below.

1) The address and size must be block aligned. The default block size is 4kB and is defined in `driver/chip/<chip>/inc/flash_opt_gen.h` header file.
2) To configure the size or the address, make sure there is no overlap between two adjacent memory regions. The total size of all the regions should not exceed the physical flash size.

#### 2.4.3.1. Adjusting the layout for ARM Cortex-M4 firmware

To adjust the memory assigned to ARM Cortex-M4 firmware:

1) Modify the ROM_RTOS length and starting address in the mt2625_`flash.ld` linker script under the GCC folder of the project.

```
MEMORY
{
    ...
    ROM_RTOS(rx)          : ORIGIN = 0x08012000, LENGTH = 2344K
    ...
}
```

2) Modify the macro definitions for RTOS_BASE and RTOS_LENGTH in `project/<board>/apps/<application>/inc/memory_map.h` header file.
3) Rebuild the bootloader and the ARM Cortex-M4 firmware.

Execute the following command under the root folder of the SDK.

```
./build.sh project_board example_name BL
```

The `project_board` is the project folder of a specific hardware board and `example_name` is the name of the example. For example, to build the `hal_adc` of `mt2625_hdk`, the command will be:

```
./build.sh mt2625_hdk hal_adc BL
```

4) Make sure the length of ROM region doesn't exceed the flash size of the system and for MT2625 the internal flash is 4MB.

#### 2.4.3.2. Adjusting the memory layout with FOTA full binary update

1) Modify ARM Cortex-M4 firmware size if needed see section 2.4.3.1, "*Adjusting the layout for ARM Cortex-M4 firmware*".
2) Modify the `ROM_FOTA_RESERVED` length and starting address in the `flash.ld` linker script under the GCC folder of a project.

```
MEMORY
{
    ...
    ROM_FOTA_RESERVED(rx) : ORIGIN = 0x0825C000, LENGTH = 1612K
    ...
}
```

3) Modify the macro definitions for `FOTA_RESERVED_BASE` and `FOTA_RESERVED_LENGTH` in `project/<board>/apps/<application>/inc/memory_map.h` header file.

Note, refer to the SDK Firmware Upgrade Developer's Guide located under SDK /doc folder, for more details about how to adjust the FOTA buffer.

#### 2.4.3.3. Adjusting the NVDM buffer

To adjust the NVDM buffer layout:

1) Modify size of the ARM Cortex-M4 firmware if needed see section 2.4.3.1, "*Adjusting the layout for ARM Cortex-M4 firmware*".
2) Modify FOTA buffer size if needed see section 2.4.3.1, "*Adjusting the layout for ARM Cortex-M4 firmware*".
3) Modify the `ROM_NVDM_RESERVED` length and starting address in the `flash.ld`, if no FOTA or full binary FOTA feature is enabled.

```
MEMORY
{
    ...
    ROM_NVDM_RESERVED(rx) : ORIGIN = 0x080EF000, LENGTH = 64K
    ...
}
```

4) Modify the macro definitions for `ROM_NVDM_BASE`, `ROM_NVDM_LENGTH` in `project\<board>\apps\<application>\inc\memory_map.h` header file.

Note, to adjust the NVDM buffer, refer to the NVDM module of HAL in the MediaTek LinkIt development platform for RTOS API reference.

## 2.5. Memory layout adjustment with a scatter file

### 2.5.1. Types of scatter files

Two types of scatter files are provided:

- Template scatter file – every application scatter file should be based on the template scatter file.
- Application scatter file – every application has its particular scatter file. This scatter file will be passed to the linker during linking stage.

#### 2.5.1.1. Template scatter file

Template scatter files are based on the memory layout. If the memory layout is modified, the scatter file should also be modified manually. It's recommended to use the layout and scatter files provided by MediaTek as a reference for your customizations.

The template scatter files are located under `/driver/CMSIS/Device/MTK/<chip>/linkerscript/RVCT/` folder. The folder includes:

- `default`. This folder contains a template scatter file to build a project without FOTA memory layout, see section 2.1, "*Memory layout without FOTA*".
- `full_bin_fota`. This folder contains a template scatter file to build a project with full binary FOTA memory layout, see section 2.2, "*Memory layout with FOTA of full binary update*"
- `sysram.` This folder contains a template scatter file to enable RAM debugging. To place all your code into SYSRAM, you can use this scatter file as a reference.

#### 2.5.1.2. Application scatter file

The application scatter file is located under `/project/<board>/apps/<project>/MDK-ARM/` folder. Each application has its own scatter file and each scatter file can have a different memory layout configuration based on the application requirements.

### 2.5.2. How to use the scatter file

To create a new scatter file for your application:

- Clone a scatter file from the MDK-ARM folder of the template folder.
- Create a new scatter file manually. The memory layout in this case should also be user-defined to match your scatter file.

#### 2.5.2.1. Cloning the scatter file

To clone a scatter file from the template:

1) Specify the memory layout feature for your application development, such as without FOTA, full binary FOTA see section 2.1 and 2.2.
2) Copy the template scatter file from template folder to your application project's folder see section 2.5.1, "*Types of scatter files*".
3) Memory layout without FOTA.

Copy `/driver/CMSIS/Device/MTK/<chip>/linkerscript/rvct/default` to `/project/<board>/apps/<project>/MDK-ARM/`.

4) Memory layout with FOTA full binary update.

Copy `/driver/CMSIS/Device/MTK/<chip>/linkerscript/rvct/full_bin_fota` to `/project/<board>/apps/<project>/MDK-ARM/`.

5) Memory layout with RAM debugging.

Copy `/driver/CMSIS/Device/MTK/<chip>/linkerscript/rvct/ram` to `/project/<board>/apps/<project>/MDK-ARM/`.

6) Modify the scatter file according to the application requirements.

### 2.5.3. How to configure the scatter file

The configuration is the same; see section 2.4.3, "*Rules to adjust the memory layout*".

## 2.6. Memory layout adjustment with an IAR configuration file

### 2.6.1. Types of scatter files

Two types of scatter files are provided:

- Template scatter file – every application scatter file should be based on the template scatter file.
- Application scatter file – every application has its particular scatter file. This scatter file will be passed to the linker during linking stage.

#### 2.6.1.1. Template scatter file

Template scatter files are based on the memory layout, see section 2.1, "*Memory layout without FOTA*" and 2.2, "*Memory layout with FOTA of full binary update*". If the memory layout is modified, the scatter file should also be modified manually. It's recommended to use the layout and scatter files provided by MediaTek as a reference for your customizations.

The template scatter files are located under `/driver/CMSIS/Device/MTK/<chip>/linkerscript/IAR/` folder. The folder includes:

- `default`. This folder contains a template scatter file to build a project without FOTA memory layout, see section 2.1, "*Memory layout without FOTA*".
- `full_bin_fota`. This folder contains a template scatter file to build a project with full binary FOTA memory layout, see section 2.2, "*Memory layout with FOTA of full binary update*"
- `sysram.` This folder contains a template scatter file to enable RAM debugging. To place all your code into SYSRAM, you can use this scatter file as a reference.

#### 2.6.1.2. Application scatter file

The application scatter file is located under `/project/<board>/apps/<project>/EWARM/` folder. Each application has its own scatter file and each scatter file can have a different memory layout configuration based on the application requirements.

### 2.6.2. How to use the scatter file

To create a new scatter file for your application:

- Clone a scatter file from the EWARM folder of the template folder.

- Create a new scatter file manually. The memory layout in this case should also be user-defined to match your scatter file.

#### 2.6.2.1. Cloning the scatter file

To clone a scatter file from the template:

1) Specify the memory layout feature for your application development, such as without FOTA, full binary FOTA.
2) Copy the template scatter file from template folder to your application project's folder see section 2.5.1, "Types of scatter files".
3) Memory layout without FOTA.

Copy `/driver/CMSIS/Device/MTK/<chip>/linkerscript/iar/default` to `/project/<board>/apps/<project>/EWARM/`.

4) Memory layout with FOTA full binary update.

Copy `/driver/CMSIS/Device/MTK/<chip>/linkerscript/iar/full_bin_fota` to `/project/<board>/apps/<project>/EWARM/`.

5) Memory layout with RAM debugging.

Copy `/driver/CMSIS/Device/MTK/<chip>/linkerscript/iar/ram` to `/project/<board>/apps/<project>/EWARM/`.

6) Modify the scatter file according to the application requirements.

### 2.6.3. How to configure the scatter file

The configuration is the same; see section 2.4.3, "*Rules to adjust the memory layout*".