MT793X IoT SDK for Memory Layout Developer's Guide

Version: 1.0
Release date: 2021-04-08

Version History

Version	Date	Description
0.1	2020-11-11	Initial draft
0.2	2021-03-18	Update 16M Flash Layout
1.0	2021-04-08	Add T/WF FW partition in PSRAM

Table of Contents

Version History **2**
Table of Contents **3**
1 Overview **4**
2 Memory Mapping and Layout for Smart MCU **5**
2.1 Memory Address Mapping for the MT793X 5
2.2 Memory Layout and Configuration for the MT793X 6
2.2.1 Memory Layout 6
2.2.1.1 Load View (Flash) 6
2.2.1.2 Execution View 7
2.2.2 Programming Guide 8
2.2.3 Rules to Adjust the Memory Layout 11
2.2.3.1 Adjusting the Layout for Arm Cortex-M33 Firmware 11
2.2.3.2 Adjusting the NVDM Buffer 11
Exhibit 1 Terms and Conditions **13**

List of Figures

Figure 2-1 MT793X Memory Address Mapping 6
Figure 2-2 Load View of the MT793X Flash Memory Layout 7
Figure 2-3 Execution View of MT793X Flash/SRAM/PSRAM/TCM 8

List of Tables

Table 2-1 Memory Attribute Table 9

1 Overview

This document provides details of the memory layout design and configuration of Mediatek MT793X IoT development platform for RTOS.
Each memory layout has two types of views, the load view and the execution view. The design concept is described based on the two views:

- The load view describes a memory region and section of each image in terms of the address where it is located before the image is processed.
- The execution view describes a memory region and section of each image in terms of the address where it is located during the image execution.

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

2 Memory Mapping and Layout for Smart MCU

2.1 Memory Address Mapping for the MT793X

The MT793X chipsets support serval types of physical memories included **Serial Flash**, Pseudo Static Random Access Memory (**PSRAM**), Static Random Access Memory (**SYSRAM**) and Tightly Coupled Memory (**TCM**) . The memory layouts are designed based on the type of memory.

The virtual memory on the MT793X is provided for cacheable memory and is implemented based on the memory mapping mechanism of the Arm Cortex-M33 processor. The addresses ranging from ***0x0010_8000*** to ***0x0011_7FFF*** are mapped to the **TCM** addresses. The virtual addresses ranging from ***0x1800_0000*** to ***0x1FFF_FFFF*** are mapped to the **Flash** addresses ranging from ***0x9000_0000*** to ***0x9FFF_FFFF.*** The virtual addresses ranging from ***0x1000_0000*** to ***0x17FF_FFFF*** are mapped to the **PSRAM** addresses ranging from ***0xA000_0000*** to ***0xAFFF_FFFF.*** The virtual addresses ranging from ***0x0800_0000*** to ***0x0FFF_FFFF*** are mapped to the **SYSRAM** addresses ranging from ***0x8000_0000*** to ***0x8FFF_FFFF***, as shown in Figure 2-1. The virtual address memory region is used as cacheable memory if cache is enabled. All read-write (RW) data is stored in this region by default.

To convert address mapping between virtual memory and physical memory for SRAM/PSRAM, use the definition API at "***memory_map.h***"

```
//converts the virtual address to physical address
#define HAL_CACHE_VIRTUAL_TO_PHYSICAL(address) \
...
//converts the physical address to virtual address.
#define HAL_CACHE_PHYSICAL_TO_VIRTUAL(address) \
...
```

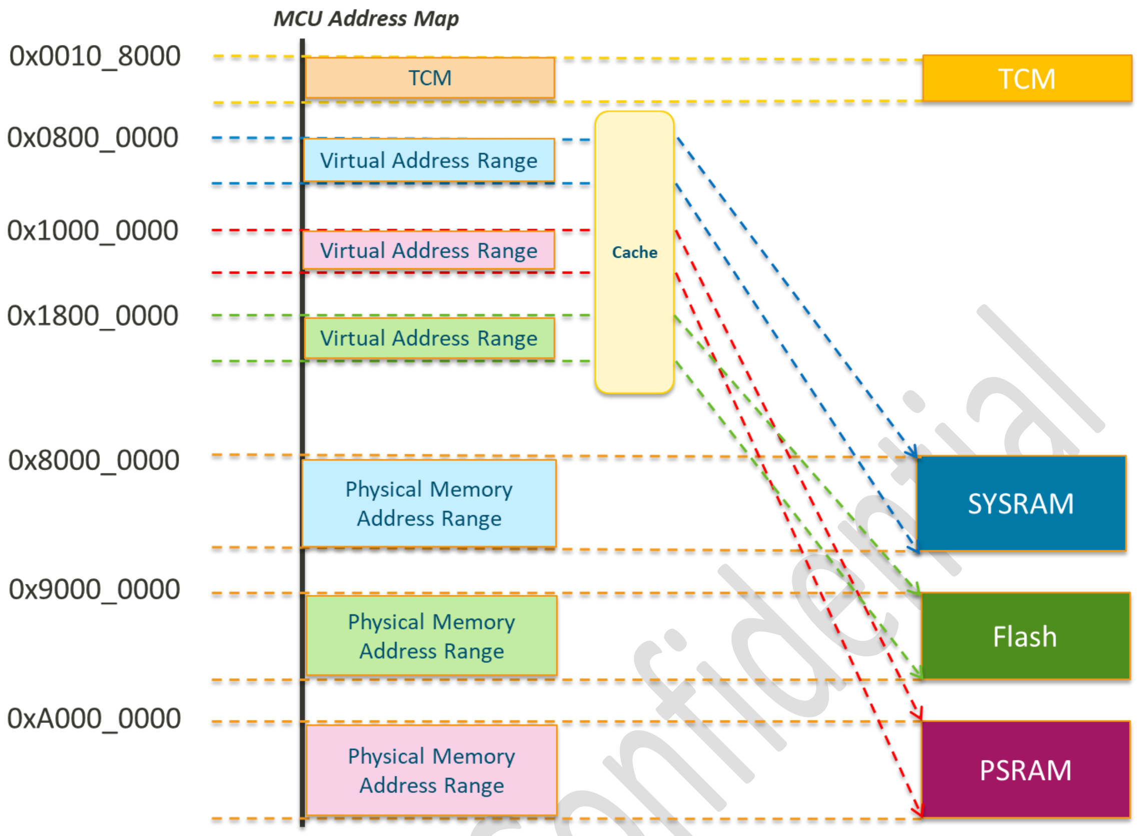

Figure 2-1 MT793X Memory Address Mapping

2.2 Memory Layout and Configuration for the MT793X

2.2.1 Memory Layout

2.2.1.1 Load View (Flash)

The MT793X demo SDK has 8-MB/16-MB serial flash memory. The load view on the 8-MB/16-MB flash memory is shown in Figure 2-2.

- Bootloader. The bootloader binary is always located at the very beginning of the flash memory. The size of the bootloader is not configurable and currently is fixed to 64KB.
- TFM. Arm Trusted Firmware-M provides a reference implementation of secure world software for Armv8-M.
- ARM Cortex-M33 firmware. This section of the memory is reserved for the RTOS binary, ARM CMSIS, HAL drivers and customer.
- FOTA. This section of the memory is reserved for Firmware Over-The-Air (FOTA) upgrade.

- NVDM. The section of the memory is reserved for the NVDM module to save system/module configured value.
- BT. This section of the memory is reserved for Bluetooth firmware.
- WiFi. This section of the memory is reserved for Wi-Fi firmware/Patch/PwrTbl.

The start address and the maximum size of each binary and reserved buffer are configurable; Please refer to Section 2.2.3, "Rule to Adjust the Memory Layout" for more details.

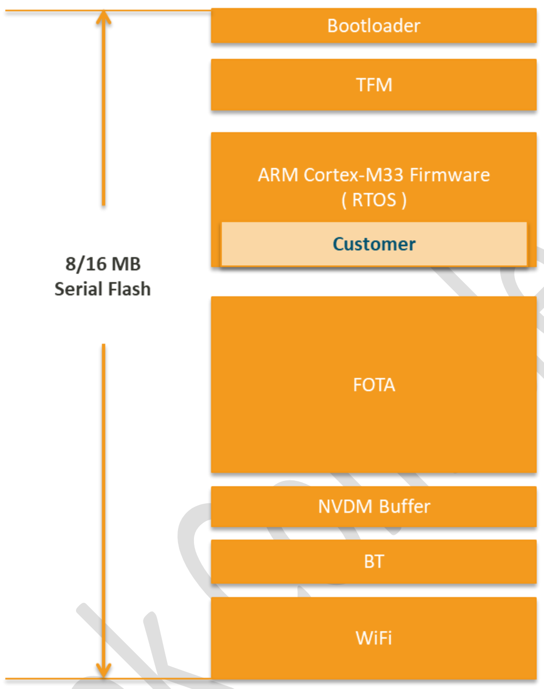

Figure 2-2 Load View of the MT793X Flash Memory Layout

2.2.1.2 Execution View

This section describes the execution view (Please refer to Figure 2-3 for the MT793X) at runtime.

- **Serial Flash**. The code and RO data are located in the flash memory during runtime.
- **SRAM**
  - Cacheable code, RW data and ZI data
  - Non-cacheable code, RW data and ZI data
- **PSRAM**
  - BT firmware, Wi-Fi firmware (Note: Execute these firmwares from their N10 respectively)
  - Cacheable code, RW data and ZI data
  - Non-cacheable code, RW data and ZI data

- **TCM**. Some critical and high-performance code and data can be stored in the TCM. Please refer to Section 2.2.2 Programming Guide to learn how to put code or data to the TCM.
  - Code and RO data.
  - RW data, ZI data.
  - System stack.
- When the system starts up, the **Reset_Handler (at statup_mt7933.s)** moves code and data from flash to each sector where they belong to.

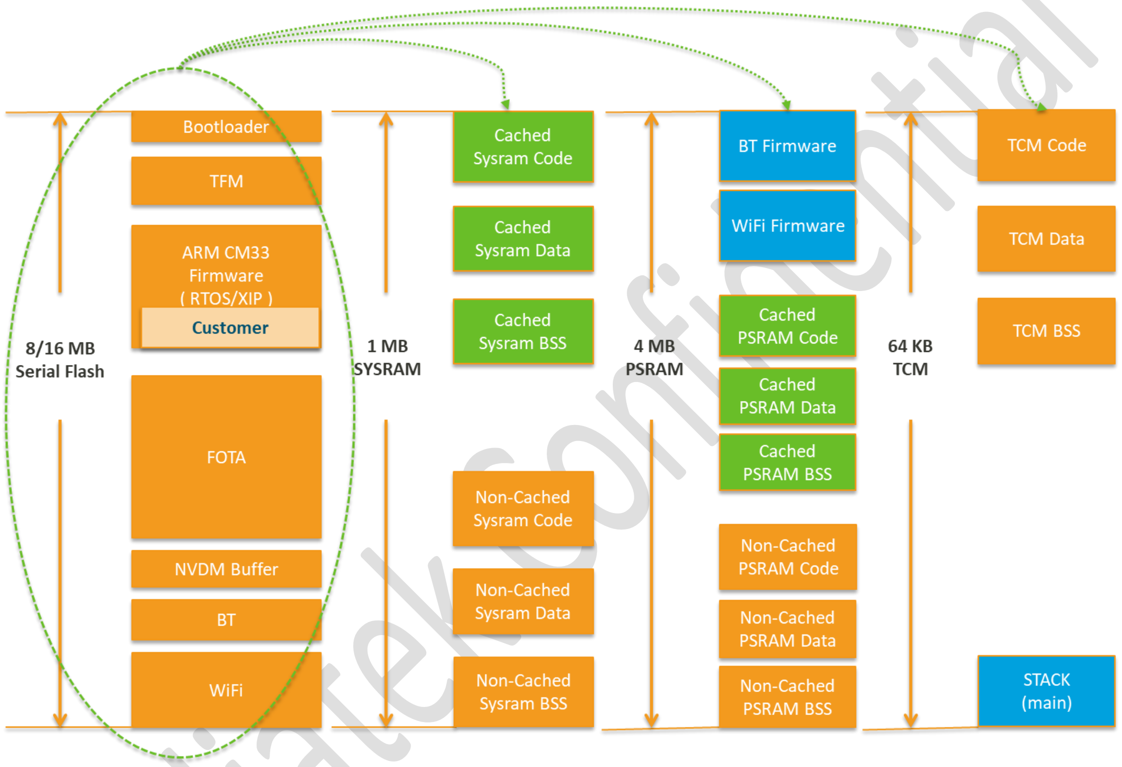

Figure 2-3 Execution View of MT793X Flash/SRAM/PSRAM/TCM

2.2.2 Programming Guide

This programming guide is based on the memory layout described in Section 2.2.1.2 Execution View. The following configurations are recommended for placing the code successfully to the desired memory location during runtime. Please reference Table 2-1 Memory Attribute Table for each memory location and include "memory_attribute.h" before using it.

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 modification is required.

2) Place the code, data and stack to TCM at runtime.

By default, the system stack is placed in the TCM; no modification is required. To run the code or access RO data in the TCM with better performance, specify the attribute explicitly in your code.

3) Place the code or RO data to the SRAM/PSRAM at runtime.

To run the code or access RO data in the SRAM/PSRAM with better performance, specify the attribute explicitly in your code, as shown in the example below.
The attribute is defined as –

```
#define    ATTR_TEXT_IN_RAM                                __attribute__
((__section__(".ram_code")))
```

		SRAM	PSRAM	TCM
Code	Cached	ATTR_TEXT_IN_SYSRAM	ATTR_TEXT_IN_RAM	ATTR_TEXT_IN_TCM
	Non-Cached	ATTR_TEXT_IN_NONCACHED_SYSRAM	ATTR_TEXT_IN_NONCACHED_RAM	
Data	Cached	ATTR_RODATA_IN_SYSRAM ATTR_RWDATA_IN_SYSRAM	ATTR_RODATA_IN_RAM ATTR_RWDATA_IN_RAM	ATTR_RWDATA_IN_TCM
	Non-Cached	ATTR_RODATA_IN_NONCACHED_SYSRAM ATTR_RWDATA_IN_NONCACHED_SYSRAM	ATTR_RODATA_IN_NONCACHED_RAM ATTR_RWDATA_IN_NONCACHED_RAM	
BSS	Cached	ATTR_ZIDATA_IN_SYSRAM	ATTR_ZIDATA_IN_RAM	ATTR_ZIDATA_IN_TCM
	Non-Cached	ATTR_ZIDATA_IN_NONCACHED_SYSRAM	ATTR_ZIDATA_IN_NONCACHED_RAM	

Table 2-1 Memory Attribute Table

```
//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 is placed in the Serial Flash instead of the SRAM/PSRAM.

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

4) Place RW data or ZI data to 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 is placed in the cacheable memory instead of the non-cacheable memory.

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

//ZI data
int b;
```

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

By default, RW data and ZI data are placed in the cacheable memory; no modification is required.

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 is 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 better performance, you should 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 is placed in the SRAM instead of the TCM.

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

2.2.3 Rules to Adjust the Memory Layout

The memory layout can be customized to fit the application requirements. However, the bootloader is not configurable. The rest of the memory layout can be adjusted as follows.
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 4 KB.
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.

2.2.3.1 Adjusting the Layout for Arm Cortex-M33 Firmware

To adjust the memory assigned to Arm Cortex-M33 firmware:

1) Modify the ROM_RTOS length and starting address in the `mt7933_flash.ld` linker script under the GCC folder of the project.

```
MEMORY
{
    ...
    ROM_RTOS(rx)           : ORIGIN = 0x18044000, LENGTH = 2616K
    ...
}
```

2) Rebuild the bootloader and the Arm Cortex-M33 firmware.

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

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

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 qfn_sdk_demo of mt7933_hdk, use the command:

```
./build.sh mt7933_hdk qfn_sdk_demo
```

3) Make sure the length of ROM region does not exceed the flash size.

2.2.3.2 Adjusting the NVDM Buffer

To adjust the NVDM buffer layout:

1) Modify size of the Arm Cortex-M33 firmware if needed. Please refer to 2.2.3.1 Adjusting the Layout for Arm Cortex-M33 Firmware.
2) Modify the ROM_NVDM length and starting address in the `mt7933_flash.ld`

```
MEMORY
{
    ...
    ROM_NVDM (rx) : ORIGIN = 0x187F0000, LENGTH = 64K
    ...
}
```

Exhibit 1 Terms and Conditions