AN13285
i.MX 8X Using L1 Cache for Cortex-M4 Core

Rev. 0 — 1 June 2021

Application Note

Contents

1 Introduction

The i.MX 8X series takes advantage of the Arm® Cortex®-M4 core with a 16K/16K L1 I/D-Cache. This delivers extremely high performance, regardless of whether the code is executed from the on-chip RAM, external flash, or external memory.

This document introduces the basic technology of the cache system that includes the L1 cache, memory types, attributes, and MPU (Memory Protection Unit) for the Cortex-M4 core embedded into the i.MX 8X series processors. It guides you on how to use the cache to develop applications running correctly and with high performance. It does not describe the details of the cache system. For more detailed information, see the Arm Cortex-M4 processor user's guide.

The software used for the example in this documentation is based on the i.MX 8QXP SDK release with the Arm CMSIS implementation. The development environment is the IAR Embedded Workbench 8.40 IDE. The hardware used to verify the example is the MIMX8QXP-MEK board.

2 Overview

This chapter describes the i.MX 8X Cortex-M4 system architecture with cache-related parts. It describes the L1 cache behavior, Arm Cortex-M4 defined memory types/attributes, and the MPU (Memory Protection Unit) system. This provides an overview of the i.MX 8X Cortex-M4 core cache system and how it affects the application use cases.

2.1 i.MX 8X system architecture (CM4 cache-related)

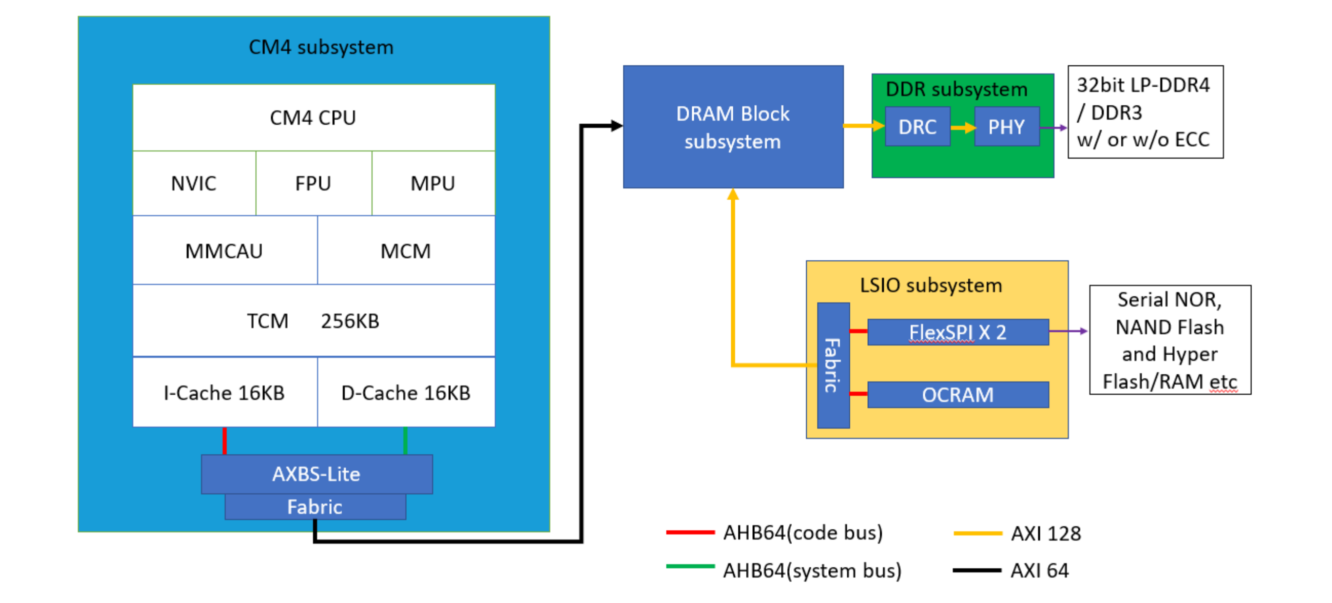

Figure 1. i.MX 8QXP core and system block diagram

The i.MX 8X series implement a CM4 subsystem shown in Figure 1. The L1 I/D-Cache is embedded in the CM4 subsystem. Both the data cache and the instruction cache have a cache line size of 16 bytes. The M4 code and system busses, cached by L1 caches, are connected to the AXBS-Lite. The DB(DRAM Block) subsystem is connected to the bus fabric slave port. The DDR subsystem and the LSIO subsystem (including the OCRAM and two FlexSPIs that support the Serial NOR, NAND Flash, and Hyper Flash/RAM, etc.) are connected to the DB subsystem as well. The CPU core accesses the subsystem through this bus fabric using the L1 cache.

Because the access to the subsystems of those memories can take multiple cycles (especially on the external memory interfaces with multiple wait states), the L1 cache is designed to speed up the read/write operation from/to the memory. This brings a big performance boost.

The TCM is accessed directly by the CPU core and it bypasses the L1 cache. Therefore, it is recommended to put the critical code and data, such as the vector table, into the TCM.

2.2 L1 cache behavior

Any access that is not for a TCM is handled by the appropriate cache controller. If the access is to a non-shared cacheable memory and the cache is enabled, a lookup is performed in the cache and if found (a cache hit), the data is fetched from or written into the cache. When the cache is not enabled and for a non-cacheable or shared memory, the accesses are performed using the AXI bus.

Both caches allocate a memory location to a cache line on a cache miss because of a read. That means that all cacheable locations are Read-Allocate. In addition, the data cache can allocate on a write access if the memory location is marked as Write-Allocate. When a cache line is allocated, the appropriate memory is fetched into a line-fill buffer by the AXI bus before being written to the cache.

The write accesses that hit in the data cache are written into the cache RAMs. If the memory location is marked as Write-Through, the write is also performed on the AXI bus, so that the data stored in the RAM remains coherent with the external memory system. If the memory is Write-Back, the cache line is marked as dirty and the write is only performed on the AXI bus when the line is evicted. When a dirty cache line is evicted, the data is passed to the write buffer in the AXI bus to be written to the external memory system.

2.3 Memory types and attributes

The memory map and the programming of the MPU splits the memory map into regions. Each region has a defined memory type and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

- Normal – The processor can re-order transactions for efficiency or perform speculative reads.
- Device and Strongly-Ordered – The processor preserves the transaction order relative to other transactions to the Device or Strongly-Ordered Memory.

The different ordering requirements for the Device and Strongly-Ordered Memory mean that the external memory system can buffer a write to the Device memory, but it must not buffer a write to the Strongly-Ordered Memory.

The memory attributes include:

- Shareable (S) – For a shareable memory region, the memory system provides data synchronization between bus masters in a system with multiple bus masters, for example, a processor with a DMA controller. For i.MX 8X, shareable means non-cacheable by default.
- Execute Never (XN) – This means that the processor prevents instruction accesses. A fault exception is generated only on execution of an instruction that is executed from an XN region.
- TEX, Cacheable (C), Bufferable (B) – This identifies the memory type and cache policy used by this region of memory.
- Access permission (AP) – These are the access permissions for privileged and unprivileged software. The value can be “No access”, RW, or RO.
- Subregion disable (SRD) - The regions of 256 bytes (or more) are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the MPU_RASR to disable a subregion.

The *memory type, S, TEX, C, B* attributes determine the cache policy that the application should take care of. See the next section for the cache policy.

2.3.1 Cache policy

The TEX/C/B attributes define the memory type and cache policy applied to the region of memory. The list of a full cache policy settings table is in the Arm Cortex-M4 processor user's guide.

Table 1. Cache policy settings

TEX	C	B	S	Memory type	Shareability	Other attributes
0b000	0	0	x[a]	Strongly Ordered	Shareable	-
		1	x[a]	Device	Shareable	-
	1	0	0	Normal	Not shareable	Outer and inner Write-Through. No Write-Allocate.
			1		Shareable	
		1	0	Normal	Not shareable	Outer and inner Write-Back. No Write-Allocate.
			1		Shareable	
0b001	0	0	0	Normal	Not shareable	Outer and inner noncacheable.
			1		Shareable	
		1	x[a]	Reserved encoding		-
	1	0	x[a]	Contact the Arm partner for attributes used.		-
		1	0	Normal	Not shareable	Outer and inner Write-Back. Write and Read Allocate.
			1		Shareable	
0b010	0	0	x[a]	Device	Not shareable	Nonshared Device.
		1	x[a]	Reserved encoding		-
	1	x[a]	x[a]	Reserved encoding		-
0b1BB	A	A	0	Normal	Not shareable	Cached memory, BB = outer policy, AA = inner policy.
			1		Shareable	

[a] The MPU ignores the value of this bit.

Each of the cache policy is as follows:

- Write Allocation (WA) – A cache line is allocated on a write miss. This means that executing a store instruction on the processor might cause a burst read.
- Write-Back (WB) – A write updates the cache only and marks the cache line as dirty. The external memory is updated only when the line is evicted or explicitly cleaned.
- Write-Through (WT) – A write updates both the cache and the external memory system. This does not mark the cache line as dirty.

2.3.2 i.MX8QXP memory map

The default memory map of important regions with memory types and the bus policy is listed in Table 2 and Table 3. The application can use an MPU to configure different memory types and cache policies to overwrite the default ones. The CM4 memory map shown in Table 2 and Table 3 is from the local point of view of the CM4.

2.3.2.1 From M4 code bus

Table 2. i.MX 8QXP code bus setting

Start address	End address	Size	Comment
0x0010_0000	0x1BFF_FFFF	447 MB	NIC maps to SSI outbound. Physically mapped to memory subsystem.
0x1C00_0000	0x1FBF_FFFF	60 MB	NIC maps to SSI outbound. Reserved for M4 code bus mappings.
0x1FC0_0000	0x1FFD_FFFF	3968 KB	Access generates bus error. Reserved for TCML growth.
0x1FFE_0000	0x1FFF_FFFF	128 KB	TCML memory.

2.3.2.2 From M4 system bus

Table 3. i.MX8QXP system bus setting

Start Address	End Address	Size	Comment
0x2000_0000	0x2001_FFFF	128 KB	TCMU memory.
0x2002_0000	0x203F_FFFF	3968 KB	Access generates bus error. Reserved for TCMU growth.
0x2040_0000	0x20FF_FFFF	12 MB	NIC0 maps to SSI outbound. Reserved for M4 system bus mappings.
0x2100_0000	0x211F_FFFF	2 MB	NIC0 maps to SSI outbound with translation. Alias for AP ROM and OCRAM.
0x2120_0000	0x27FF_FFFF	110 MB	NIC0 maps to SSI outbound. Reserved for M4 system bus mappings.
0x2800_0000	0x3FFF_FFFF	384 MB	NIC0 maps to SSI outbound. Physically mapped to smart subsystems.
0x4000_0000	0x40FF_FFFF	16 MB	NIC0 maps to SSI outbound. Reserved for M4 system bus mappings.
0x4100_0000	0x417F_FFFF	8 MB	AIPS-Lite0 peripherals.
0x4180_0000	0x41FF_FFFF	8 MB	Default slave. Reserved for additional slave.
0x4200_0000	0x5FFF_FFFF	480 MB	NIC0 maps to SSI outbound. Physically mapped to subsystems.
0x6000_0000	0x7FFF_FFFF	512 MB	NIC0 maps to SSI outbound. Physically mapped to PCIe mapped I/O.
0x8000_0000	0xDFFF_FFFF	1.5 GB	NIC0 maps to SSI outbound. Physically mapped to DRAM.
0xE000_0000	0xE00F_FFFF	1 MB	M4 PPB (Private Peripheral Bus).
0xE010_0000	0xEFFF_FFFF	255 MB	Access generates bus error. Arm recommends reserving this vendor-specific region.
0xF000_0000	0xFFFF_FFFF	256 MB	Vendor_SYS (Vendor System) region.

2.3.2.3 Commonly used cache policy on i.MX 8X

The NXP SDK MPU settings are usually in the *board.c* file of each example. They are the following:

The MPU region 0 sets the address space from 0x2000000 to 0x3FFFFFFF (512 MB) as non-shared device memory and disables subregions 0 and 1 (from 0x2000000 to 0x27FFFFFF). This has the effect of setting the range from 0x20000000 to 0x27FFFFFF as normal NX-cached memory (Cortex-M4 background memory attributes) and the range from 0x2800000 to 0x3FFFFFFF is set to non-shared device memory.

The MPU region 1 set the address space of 0x80000000 ~ 0xFFFFFFFF to be non-cacheable. Subregions 6 and 7 are disabled to use the background memory attributes for the address space of 0xE0000000 ~ 0xFFFFFFFF, that is non-shareable device memory.

The MPU region 2 sets the text and data sections to be cacheable if the program runs on the DDR. The cacheable area base address should be multiples of its size in the linker file (they can be modified according to the user's requirements).

Table 4. i.MX 8QXP SDK common memory type and cache policy

Region Number	Start Address	Size	XN	AP	TEX	S	C	B	SRD[7..0]	Memory Type
0	0x2000_0000	512 MB	1	RW	2	0	0	0	0b00000011	Device
1	0x8000_0000	2 GB	0	RW	0	0	0	1	0b11000000	Device
2	0x8800_0000	1 MB	0	RW	1	0	1	1	0b00000000	Normal

2.4 MPU (Memory Protection Unit)

The MPU divides the memory map into a few regions and defines the location, size, access permissions, and memory attributes of each region. It supports the following:

- Independent attribute settings for each region
- Overlapping regions
- Export of memory attributes to the system

The memory attributes affect the behavior of memory accesses to the region. The i.MX 8X MPU defines:

- 16 separate memory regions (0-15)
- Background region

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 15 take precedence over the attributes of any region that overlaps region 15. The background region has the same memory access attributes as the default memory map, but it is accessible only from privileged software.

The MPU memory map is unified. This means that instruction accesses and data accesses have the same region settings. If a program accesses a memory location that is prohibited by the MPU, the processor generates a MemManage fault. This causes a fault exception and it might cause a termination of the process in an OS environment.

Typically, the application or embedded OS uses the MPU for memory protection and memory cache policy configurations. See Cache maintenance in SDK driver for how to use the MPU to configure a memory region for a different cache policy.

3 Cache operation

There are the following three types of cache operations:

- **Cache Enable/Disable** – Cache on/off
- **Cache Clean** – Writes the dirty cache lines back to the memory (sometimes called a flush)
- **Cache Invalidate** – Marks the contents in the cache as invalid (basically, a delete operation)

The i.MX 8X SDK provides the below way to perform cache operations.

3.1 Accessing the cache using SDK

The i.MX 8X SDK provides a cache driver (*fsl_cache.h and fsl_cache.c*) for L1 cache operations, configuring the register of the CM4 peripheral as follows:

```
void L1CACHE_EnableCodeCache(void)
void L1CACHE_DisableCodeCache(void)
void L1CACHE_InvalidateCodeCache(void)
void L1CACHE_InvalidateCodeCacheByRange(uint32_t address, uint32_t size_byte)
void L1CACHE_EnableSystemCache(void)
void L1CACHE_DisableSystemCache(void)
void L1CACHE_InvalidateSystemCache(void)
void L1CACHE_InvalidateSystemCacheByRange(uint32_t address, uint32_t size_byte)
void L1CACHE_CleanSystemCacheByRange(uint32_t address, uint32_t size_byte)
void L1CACHE_CleanInvalidateSystemCacheByRange(uint32_t address, void uint32_t size_byte)
```

4 Cache maintenance and data coherency

The cache brings a great performance boost, but the user must pay attention to the cache maintenance for data coherency.

4.1 Typical use case

To get better understanding of the cache maintenance and data coherency, this section describes a typical use case as an example: playback of an audio file stored in the external flash.

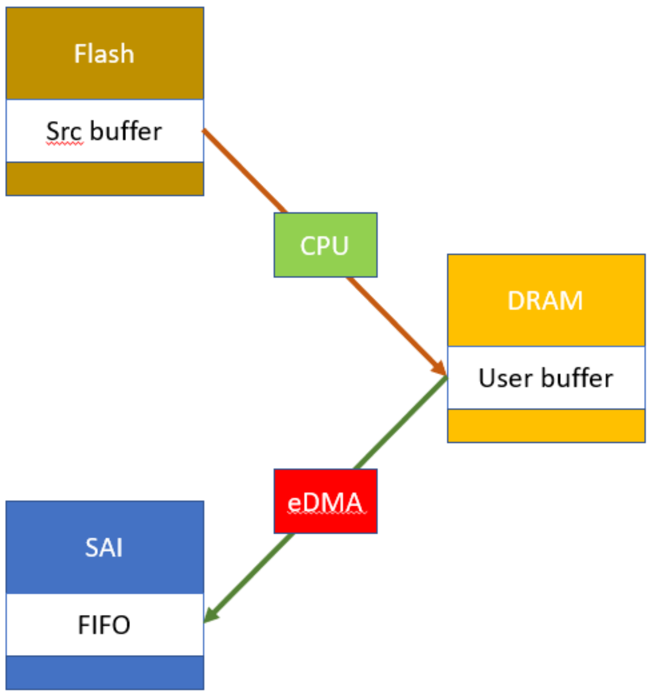

Figure 2. Data flow

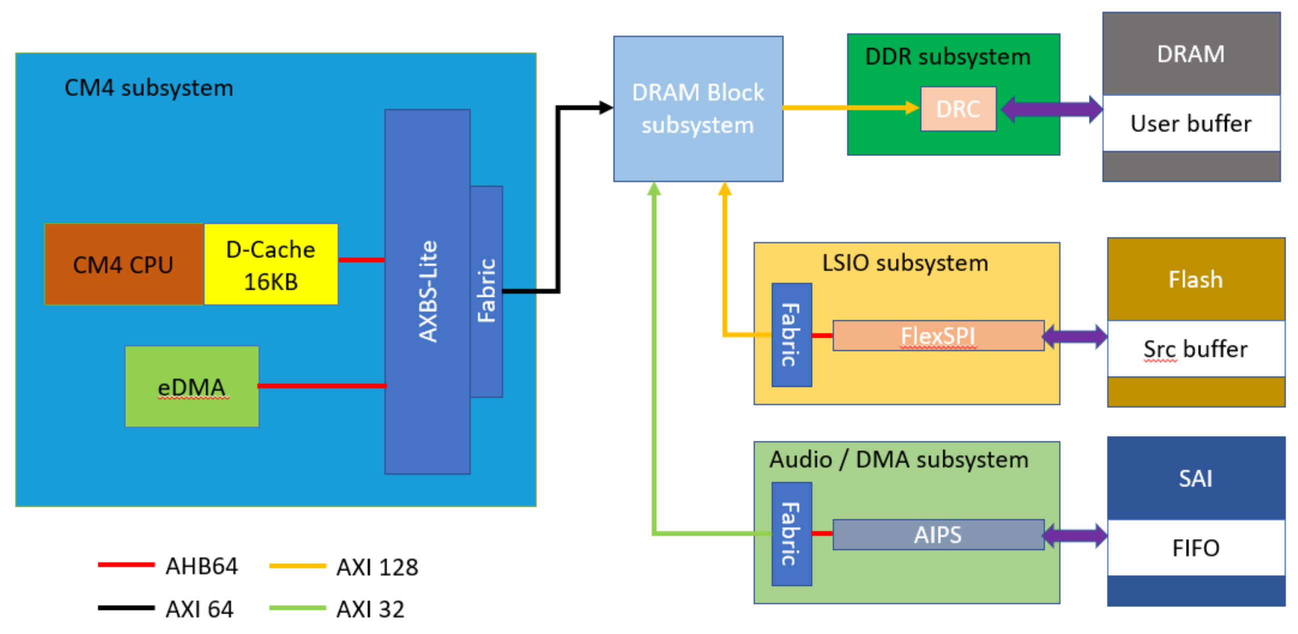

Figure 3. Subsystem interconnection diagram

The CPU reads the audio file content in the SRC buffer through the L1 D-Cache and decodes the PCM frame data written into the DRAM's USER buffer. When the USER buffer is full, the eDMA is started to copy the PCM frame data into the FIFO inside the SAI IP module. The SAI then shifts out the FIFO data to the SAI bus for audio playback. When the CPU writes the frame data to the DRAM with the L1 cache enabled, the data may only be written to the cache, because the default cache policy for the DRAM is Write-Back. The eDMA will then transfer the incorrect data to the SAI FIFO.

To avoid such data coherency issue, here are some solutions:

1. Perform a D-Cache clean operation after the CPU writes data to the DRAM.
2. Configure the DRAM memory region cache policy from Write-Back to Write-Through in the MPU before this write starts.
3. Configure the DRAM memory region cache policy to non-cacheable in the MPU.
4. Configure the DRAM memory region as shareable in the MPU, which means that it is non-cacheable.

For areas where the DMA is used frequently, it is better to configure it as a non-cacheable region, following solutions 3 and 4. However, for some specific application area, a non-cacheable region reduces the flexibility of the memory, so solutions 1 and 2 would be better.

4.2 Cache maintenance in SDK driver

The following drivers in the SDK maintain the data coherency of the cache:

Ethernet

In the ENET, a unified DMA (uDMA) engine is designed. It optimizes the data transfer between the ENET core and the SoC and supports an enhanced buffer descriptor programming model to support the IEEE 1588 functionality.

The user can pass cacheable buffers to this driver. The driver takes care of the data coherency. For other cases that use the DMA, the user should take care of the data coherency using cache operations. See the next section.

4.3 Cache maintenance by application

There are two ways to perform cache maintenance in an application.

4.3.1 Using cacheable buffers

Normally, the buffers on the DRAM can be Cacheable and Write-Back. To use such buffer as a DMA source, the user must perform a DCACHE clean operation before the DMA starts. This ensures that all of the data are committed to the memory from the cache. If a buffer is used as a DMA receive destination, the DCACHE invalidate operation must be done after the DMA completes and before the CPU or other masters read. The buffer address should have the aligned L1 cache line size (16 bytes on the i.MX 8X).

1.linker file

Configure the "symbol __CACHE_REGION_START" and "__CACHE_REGION_SIZE" entries in the linker file.

```
    define symbol m_data_start         = 0x88200000;
    define symbol m_resume_end         = 0x883FFFFF;
    define exported symbol __CACHE_REGION_START   = m_data_start;
    define exported symbol __CACHE_REGION_SIZE    = m_resume_end - m_data_start + 1 ;
```

2.MPU configuration

The SDK *board.c* "BOARD_InitMemory" function will parse these parameters as follows:

```
    extern uint32_t __CACHE_REGION_START[];
    extern uint32_t __CACHE_REGION_SIZE[];
    uint32_t cacheStart = (uint32_t)__CACHE_REGION_START;
    uint32_t size       = (uint32_t)__CACHE_REGION_SIZE;
```

Before the user can configure any regions, the MPU must be disabled by "MPU->CTRL = 0;". To configure the region, "MPU->RBAR" and "MPU->RASR" must be configured separately.

To configure the region number and its base address, the register can be configured as follows:

```
/* Select Region 2 and set its base address to the cacheable region start address. */
MPU->RBAR = (cacheStart & MPU_RBAR_ADDR_Msk) | MPU_RBAR_VALID_Msk | (2 << MPU_RBAR_REGION_Pos);
```

Configure the cacheable, bufferable, and others as follows:

```
        /* Region 2 setting:
         * 1) Enable Instruction Access;
         * 2) AP = 011b, full access;
         * 3) Outer and inner Cacheable, write and read allocate;
         * 4) Region Not Shared;
         * 5) All Sub-Region Enabled;
         * 6) MPU Protection Region size get from linker file;
         * 7) Enable Region 2.
         */MPU->RASR = (0x3 << MPU_RASR_AP_Pos) | (0x1 << MPU_RASR_TEX_Pos) | (0x1 << MPU_RASR_C_Pos) | (0x1
<< MPU_RASR_B_Pos) | ((i - 1) << MPU_RASR_SIZE_Pos) | MPU_RASR_ENABLE_Msk;
```

- SRD means to disable the subregion, which is used in the region overlap case. Just ignore it and set it to 0.
- The size is defined as the region size in bytes = 2(SIZE+1).
- The XN/AP/TEX/S/C/B/SRD parameters are exactly the same as the attributes defined in Memory types and attributes.

4.3.2 Using non-cacheable buffers

Using a non-cacheable buffer would make the process easier, because it can avoid the cache data coherency problem. However, its downside is that the performance of accessing the buffer is not good as with the cacheable ones when the CPU accesses them multiple times.

To make buffers non-cacheable, the user must configure at least one region of memory as a non-cacheable attribute in the MPU and put the buffers into this region by the linker of a toolchain.

The following are the steps to be done in the application:

1. Buffer definition

The SDK provides the following two macros for the application to define the buffers (variable) in the “Noncacheable” section of the program:

```
AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)
AT_NONCACHEABLE_SECTION(var)
```

The first macro defines the buffer (var) with the start address aligned by alignbytes. The second macro defines the buffer (var) with start address aligned automatically by the compiler (4-bytes aligned). Some use cases require the application to explicitly define the buffer aligned with the special bytes. As with the framebuffer for the eLCDIF, 8-bytes aligned is required:

```
AT_NONCACHEABLE_SECTION_ALIGN(static uint8_t buffer[256], 8);
```

2. Linker file

The DDR memory region can be allocated for the CM4 noncacheable data. Add the NCACHE_VAR block (NonCacheable section) with the size for the user buffers' size requirement. Put this "NCACHE_VAR" block into the DDR DATA2 region.

```
define symbol m_data2_start            = 0x88100000;
define symbol m_data2_end              = 0x8FFFFFFF;
define region DATA2_region = mem:[from m_data2_start to m_data2_end];
define block NCACHE_VAR    { section NonCacheable , section NonCacheable.init };
place in DATA2_region      { block NCACHE_VAR };
```

3. MPU configurations

Before the user can configure any regions, the MPU must be disabled by "MPU->CTRL = 0;". To configure the region, the "MPU->RBAR" and "MPU->RASR" entries must be configured seperately.

To configure the region number and its base address, the register can be configured as follows:

```
/* Select Region 0 and set its base address to the M4 code bus start address 0x20000000. */
    MPU->RBAR = (0x20000000U & MPU_RBAR_ADDR_Msk) | MPU_RBAR_VALID_Msk | (0 << MPU_RBAR_REGION_Pos);
```

To configure the cacheable, bufferable, and others:

```
    /* Region 0 setting:
     * 1) Disable Instruction Access;
     * 2) AP = 011b, full access;
     * 3) Non-shared device;
     * 4) Region Not Shared;
     * 5) Sub-Region 0,1 Disabled;
     * 6) MPU Protection Region size = 512M byte;
     * 7) Enable Region 0.
     */
MPU->RASR = (0x1 << MPU_RASR_XN_Pos) | (0x3 << MPU_RASR_AP_Pos) | (0x2 << MPU_RASR_TEX_Pos) |
(0x3 << MPU_RASR_SRD_Pos) | (28 << MPU_RASR_SIZE_Pos) | MPU_RASR_ENABLE_Msk;
```

After configuring all the regions, enable the priviliged default memory map and the MPU as follows:

```
MPU->CTRL = MPU_CTRL_ENABLE_Msk | MPU_CTRL_PRIVDEFENA_Msk
```

If the "MPU_CTRL_PRIVDEFENA_Msk" is not asserted, any memory access to a location not covered by an enabled region causes a fault.

4.3.3 DDR aliasing

For a general case, TCML is used for code, TCMU for data, and DRAM for a non-cacheable buffer and non-cacheable memory. In some cases, the code must run in the DDR.

The i.MX 8X devices include a basic translation block that can be leveraged to alias the DDR memory onto the Cortex-M code bus. This aliasing allows for performance improvements when the Cortex-M code is located in the DDR memory by enabling the CODE BUS and I-Cache. The Cortex-M image can activate the aliasing during the execution by updating the MCM Process ID register (MCM_PID).

If the DDR aliasing is enabled, the DDR region will be aliased in an area originally dedicated to the FlexSPI0 memory-mapped (XIP) space. The application must manage this case.

4.3.3.1 Enabling the DDR aliasing

The DDR aliasing is supported by NXP's SDK. The "__STARTUP_CONFIG_DDR_ALIAS=1" should be defined in the target toolchain. The map file can be checked and the ".text" section should be in the 0x0800_0000 - 0x0FFF_FFFF region.

4.3.3.2 How DDR aliasing works

The MCM_PID has two work mode values, configured in the Reset Handler startup assembly code with a value of 0x7E:

```
Reset_Handler
#ifdef __STARTUP_CONFIG_DDR_ALIAS
LDR     R0, =0xE0080030
MOV     R1, #0x7E
STR     R1, [R0]
ISB
LDR     R0, =alias_start
MOV     PC, R0
alias_start:
#endif
```

Table 5. i.MX8QXP DDR alias MCM_PID configuration

MCM_PID	DDR Region	Size	Alias
0x7E	0x88000000 - 0x8FFFFFFF	128 MB	0x08000000 - 0x0FFFFFFF
0x7F	0x90000000 - 0x97FFFFFF	128 MB	0x10000000 - 0x17FFFFFF
Other	N/A	N/A	No Alias

The linker file will also modify the code address and the relative cache region.

```
if (isdefinedsymbol(__STARTUP_CONFIG_DDR_ALIAS)) {
    define symbol m_interrupts_start       = 0x08000000;
    define symbol m_interrupts_end         = 0x080009FF;
    define symbol m_text_start             = 0x08000A00;
    define symbol m_text_end               = 0x081FFFFF;
} else {
    define symbol m_interrupts_start       = 0x88000000;
    define symbol m_interrupts_end         = 0x880009FF;
    define symbol m_text_start             = 0x88000A00;
    define symbol m_text_end               = 0x881FFFFF;
}
if (isdefinedsymbol(__STARTUP_CONFIG_DDR_ALIAS)) {
    define exported symbol __CACHE_REGION_START   = m_data_start;
    define exported symbol __CACHE_REGION_SIZE    = m_data_end - m_data_start + 1 ;
} else {
    define exported symbol __CACHE_REGION_START   = m_interrupts_start;
    define exported symbol __CACHE_REGION_SIZE    = m_data_end - m_interrupts_start + 1 ;
}
```

NOTE

The SCFW always checks the configuration of the MCM_PID and its I-Cache. In the alias mode, that cache is open by default.

4.3.4 OCRAM aliasing

The M4 can use the OCRAM only if it has access to the OCRAM and OCRAM Alias memory regions and its partition contains the "SC_R_OCRAM" resource.

Consider the fact that there is only one OCRAM memory (256 KB) which has several aliases, so it is the user's task to use this memory in a proper manner to avoid data corruption between code and data. The OCRAM is mapped at "0x0000_0000 -- 0x0001_7FFF (OCRAM Aliasing (lower 96 kb)) and 0x0010_0000 -- 0x0013FFFF (OCRAM)". It can be used for the M4 code to use the benefit brought by the Code Bus and I-Cache of Cortex-M4. It is also mapped at "0x2100_0000 -- 0x2101_7FFF(OCRAM Aliasing (lower 96 kb)) and 0x2110_0000 -- 0x2113FFFF (OCRAM Aliasing)", which can be used for the M4 System Bus and D-Cache of Cortex-M4.

Table 6. i.MX8QXP OCRAM and OCRAM aliasing

Start Address	End Address	Size	Comment
0x0000_0000	0x0001_7FFF	96 KB	OCRAM Alias (lower 96 kb)
0x0010_0000	0x0013_FFFF	256 KB	OCRAM
0x2100_0000	0x2101_7FFF	96 KB	OCRAM Alias (lower 96 kb)
0x2110_0000	0x2113_FFFF	256 KB	OCRAM Alias

On the current architecture of the i.MX 8/8X, the M4 can be booted from the TCM, DDR, and flash (NOR), but it cannot be booted directly from the OCRAM. Note that the OCRAM memory can be used only if the A cores are not used. An example is the U-boot that uses it for SPL booting. To use the OCRAM for the TEXT/code, the OCRAM can be used only after the M4 execution jumps from the previous memory from which it booted. The "L1CACHE_EnableCodeCache" and "L1CACHE_DisableCodeCache()" functions can be used to control the I-Cache for the OCRAM. To enable the D-cache for the OCRAM, the following is the example code to configure the MPU:

```
         /* Region 3 setting:
          * 1) Enable Instruction Access;
          * 2) AP = 011b, full access;
          * 3) Outer and inner Cacheable, write and read allocate;
          * 4) Region Not Shared;
          * 5) All Sub-Region Enabled;
          * 6) MPU Protection Region size 256KB, starting from 0x21100000;
          * 7) Enable Region 3.
          */
MPU->RBAR = (0x21100000 & MPU_RBAR_ADDR_Msk) | MPU_RBAR_VALID_Msk | (3 << MPU_RBAR_REGION_Pos);
MPU->RASR = (0x3 << MPU_RASR_AP_Pos) | (0x1 << MPU_RASR_TEX_Pos) | (0x1 << MPU_RASR_C_Pos) | (0x1 <<
MPU_RASR_B_Pos) | (17 << MPU_RASR_SIZE_Pos) | MPU_RASR_ENABLE_Msk;
```

NOTE

For information on how to use the OCRAM memory for the DATA/TEXT on Cortex-M4, see OCRAM for DATA/TEXT on Cortex-M4.

5 Conclusion

To use the i.MX 8X L1 cache in a correct and efficient way, the following are several recommendations and tips:

- Put the critical code and data into the TCM. This is the fastest way for the CPU to access the code and data.
- Always call the SDK cache driver API to cache operations. This makes sure that the cache is cleaned before it is disabled and invalidated before it is enabled to avoid unpredictable issues.
- If the software is using cacheable memory regions for the DMA source/or destination buffers, the software must trigger a cache clean before starting a DMA operation to ensure that all the data are committed to the subsystem memory. After the DMA transfer completes and when reading the data from the peripheral, the software must perform a cache invalidation before reading the DMA updated memory region. NXP recommends to use non-cacheable regions for DMA buffers. The software can use the MPU to configure a non-cacheable memory region to be used as a shared buffer between the CPU and DMA.

When using the FlexSPI for the external NOR Flash read by the AHB bus, the cacheable memory would cause problems. Because the flash erase and program operation goes through the IP command but not through the AHB bus, a cache invalid operation is needed before the CPU reads the FlexSPI memory map after any erase and program completes.

The DDR aliasing and OCRAM aliasing would help to build a more flexible application and enhance the performance using the I-cache and they are very useful and suggested to be used.

6 References

- ARM Cortex-M4 Devices Generic User Guide (Revision: 1.0,b)
- ARM Cortex-M4 Processor Technical Reference Manual (Revision: r0p1)
- *i.MX 8X Reference Manual*
- *SCU sc_fw_port.pdf*
- Cache (computer) - https://en.wikipedia.org/wiki/Cache_(computing)

7 Revision history

Table 7 summarizes the changes done to this document since the initial release.

Table 7. Revision history

Revision number	Date	Substantive changes
0	1 June 2021	Initial release

Date of release: 1 June 2021

Document identifier: AN13285