恩智浦半导体
应用笔记

文档编号：AN12042
第1版，2017年12月

使用i.MXRT的L1缓存

1. 介绍

i.MXRT系列充分利用了带有32K/32K L1 I/D-Cache（指令/数据缓存）的Arm Cortex-M7核，无论是在片上RAM、外部Flash（闪存）还是外部存储器中执行代码，都能提供极高的性能。

本文档介绍了缓存系统的基本技术，包括L1缓存、存储器类型及属性和MPU系统（内存保护单元），指导用户如何使用缓存来开发一个以正确且高性能的方式运行的应用。本文不深入阐述缓存系统的细节，更多详细信息，请参见《Arm Cortex-M7处理器用户指南》（Arm Cortex-M7 Processor User Guide）。

本文档中的示例使用的软件基于i.MXRT1050的SDK版本，以Arm的CMSIS（Arm Cortex™微控制器软件接口标准）实现。开发环境采用IAR Embedded Workbench 8.11。用于验证示例的硬件是MIMXRT1050 EVK板。

目录

2. 概述

本章节介绍了i.MXRT系统架构与缓存相关的部分，讨论了L1缓存的行为模式、Arm Cortex-M7定义的存储器类型/属性以及MPU（内存保护单元）系统，对i.MXRT缓存系统以及它们如何影响应用的实例进行了概述。

2.1. i.MXRT系统架构（与缓存相关的部分）

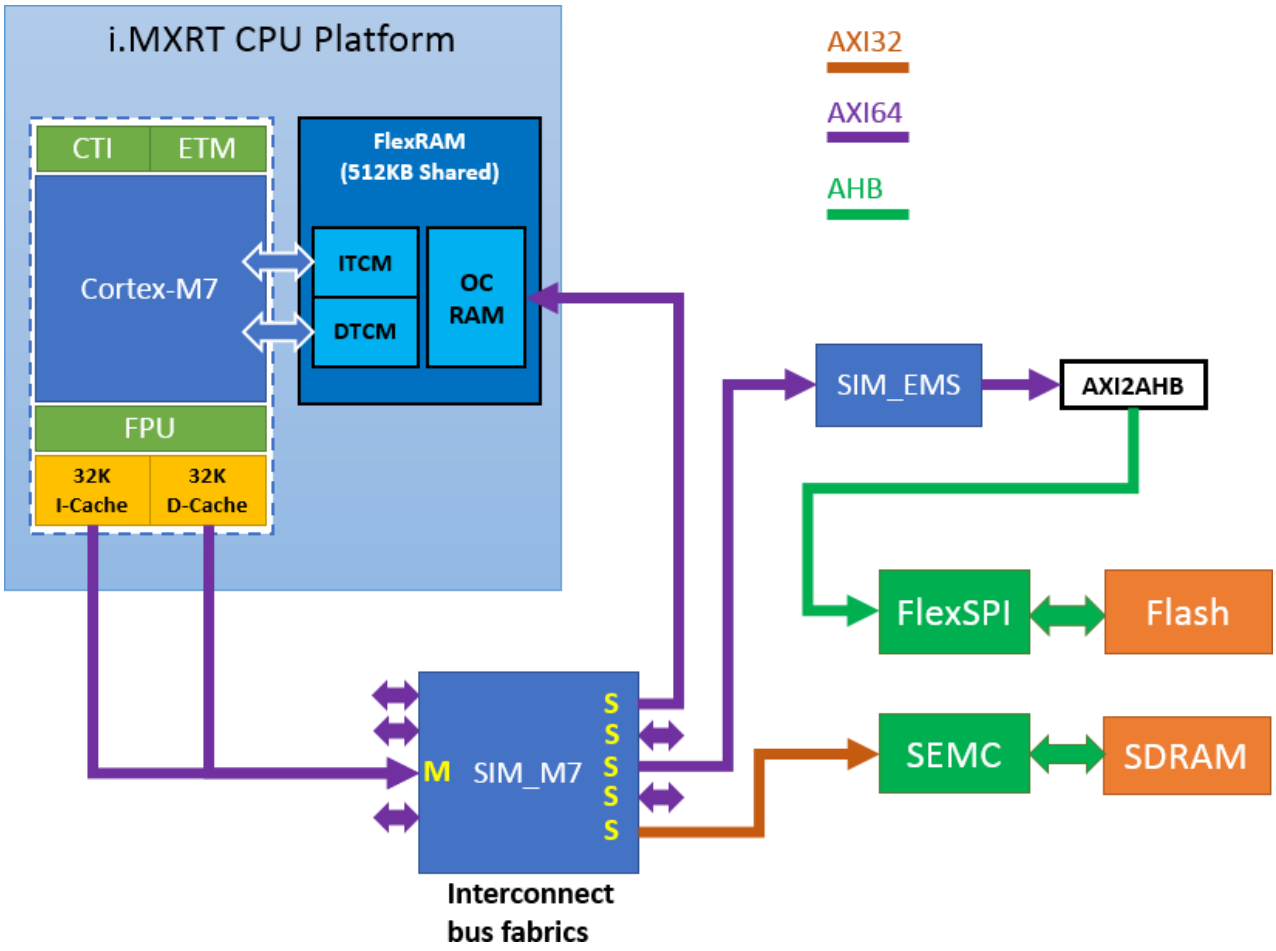

图1. i.MXRT1050内核和系统框图

i.MXRT系列实现了如图1所示的CPU内核。该CPU内核平台嵌入了L1 I/D-Cache（指令/数据缓存），其中的数据缓存为4路组相联，指令缓存为2路组相联，缓存行大小为32字节。它通过AXI总线与SIM_M7总线构架的主端口相连。内部/外部存储器的子系统，如OCRAM（配置为OCRAM的FlexRAM存储区）、FlexSPI（串行NOR、NAND Flash和Hyper Flash/RAM等）和SEMC（SDRAM、PNOR Flash、NAND Flash等）则连接到总线构架的从端口。CPU内核经L1缓存通过这个总线构架访问这些子系统。

由于对这些存储器子系统的访问可能需要多个周期（特别是对于有多个等待状态的外部存储器接口），因此采用了L1缓存以加速对这些存储器的读/写操作，这带来了极大的性能提升。

I/DTCM（配置为TCM（紧耦合存储器）的FlexRAM存储区）是绕过L1缓存，由CPU内核直接访问的。因此，建议将关键代码和数据（例如向量表）放入TCM中。

2.2. L1缓存操作

任何一个不是对于TCM的访问均由相应的缓存控制器处理。如果访问的是非共享的可缓存存储区，并且启用了缓存，则会在缓存中执行查找功能，如果找到了（即缓存命中），则从缓存中获取数据或将数据写入缓存。当缓存未启用，或者访问的是不可缓存或共享的存储区时，将使用AXI总线执行访问。

对于由读取操作导致的缓存未命中，两个缓存都会给缓存行分配一个存储区位置，也就是说，所有的可缓存位置都是为读取分配的（Read-Allocate）。此外，如果一个存储区位置被标记为写入分配（Write-Allocate），则该数据缓存可以在写入访问时进行分配。当一个缓存行被分配后，相应的存储区会在写入缓存之前由AXI总线提取到一个行填充缓冲区中。

命中了数据缓存的写入访问将会写入缓存RAM中。如果该存储区位置被标记为透写（Write-Through），则该写入操作也会在AXI总线上执行，以便存储在RAM中的数据与外部存储器系统存储的数据保持一致。如果该存储区为回写（Write-Back），则该缓存行被标记为脏（dirty），且只有在该缓存行被逐出时才会在AXI总线上执行写入操作。当一个脏缓存行被逐出时，数据将被传递到AXI总线中的写入缓冲区，以写入外部存储器系统。

2.3. 存储区的类型和属性

存储区映射和MPU的编程可将存储区映射划分为多个区域。每个区域都有一个定义好的存储类型，并且某些区域还具有其他的存储属性。存储区的类型和属性决定了对该区域进行访问的行为模式。

存储区的类型包括：

- **Normal（普通存储区）**——处理器可以重新排序事务以提高效率，或执行推测性的读取。
- **Device（设备存储区）和Strongly-Ordered（强序存储区）**——对于其他设备存储区或强序存储区相关的事务，处理器保留其事务顺序。

设备存储区和强序存储区的排序要求不同，这意味着外部存储器系统可以缓冲对设备存储区的写入，但不能缓冲对强序存储区的写入。

存储区的属性包括：

- **可共享（Shareable，S）——**对于一个可共享的存储区，存储系统可在一个具有多个总线主设备（例如一个带DMA控制器的处理器）的系统中提供数据同步。对于i.MXRT，在默认情况下可共享意味着不可缓存（**non-cacheable**）。
- **不可执行（Execute Never，XN）——**表示处理器会阻止指令访问。若在XN区域执行指令，就会产成故障异常。
- **TEX、可缓存（Cacheable，C）、可缓冲（Bufferable，B）——**标识该存储区使用的存储类型和缓存策略。
- **访问权限（Access permission，AP）**——有特权和无特权软件的访问权限。值可以是“无访问权限（No access）”、可读写（RW）、只读（RO）。

***这些存储类型，S、TEX、C、B*等属性决定了应用应注意的缓存策略。**请参见下一节的缓存策略。

2.3.1. 缓存策略

TEX/C/B属性定义了应用于存储区的存储类型和缓存策略，这里列出了这些位的常用组合：

表1. 缓存策略的设置

TEX	C（可缓存）	B（可缓冲）	存储区类型	缓存策略
0b000	0	0	强序存储区	不可缓存
	0	1	设备存储区	不可缓存
	1	0	普通存储其	透写、非写分配
	1	1	普通存储区	回写、非写分配
0b001	0	0	普通存储区	不可缓存
	1	1	普通存储区	回写、写分配

> 当**可共享（Shareable）位置1时**，无论TEX/C/B的值是什么，**缓存策略都固定为不可缓存（Non-cacheable）**。完整的缓存策略设置表可在《**Arm Cortex-M7处理器用户指南**》中找到。

各缓存策略的描述如下：

- **写入分配（Write allocation, WA）**——在写入未命中时分配一个缓存行，这意味着在处理器上执行存储指令可能会导致一个突发读取操作的发生。
- **回写（Write-back, WB）**——写入操作仅更新缓存并将缓存行标记为脏（dirty）。仅当该缓存行被逐出或显式清除时，外部存储器才会被更新。
- **透写（Write-through, WT）**——写入操作会将缓存和外部存储系统都更新，不会将缓存行标记为脏（dirty）。

2.3.2. i.MXRT1050的存储区映射

表2列出了重要区域的默认存储区映射及其存储类型和缓存策略。应用程序可以使用MPU来配置不同的存储类型和缓存策略并覆盖默认值。

表2. i.MXRT1050的存储区映射及其存储类型和缓存策略

开始值	结束值	大小	模块	存储类型	缓存策略
C000_0000	DFFF_FFFF	512MB	SEMC3	设备存储区	不可缓存
A000_0000	BFFF_FFFF	512MB	SEMC2	设备存储区	不可缓存
9000_0000	9FFF_FFFF	256MB	SEMC1	普通存储区	可缓存/透写（非写分配）
8000_0000	8FFF_FFFF	256MB	SEMC0	普通存储区	可缓存/透写（非写分配）
7FC0_0000	7FFF_FFFF	4MB	FlexSPI RX FIFO	普通存储区	可缓存/回写/写分配
7F80_0000	7FBF_FFFF	4MB	FlexSPI TX FIFO	普通存储区	可缓存/回写/写分配
6000_0000	7F7F_FFFF	504MB	FlexSPI/FlexSPI密文	普通存储区	可缓存/回写/写分配
2020_0000	2027_FFFF	512KB	OCRAM	普通存储区	可缓存/回写/写分配
2000_0000	2007_FFFF	512KB	DTCM	普通存储区	-
0000_0000	0007_FFFF	512KB	ITCM	普通存储区	-

DTCM/ITCM是一个紧耦合存储区，内核可以直接访问（不涉及缓存）。

应用程序使用哪一个SEMC存储区是由电路板设计中的片选（Chip-Select）决定的。例如，SDRAM使用SEMC_CS0，则通过SEMC0存储区访问SDRAM。

2.4. MPU（内存保护单元）

内存保护单元（MPU）将存储区映射划分为几个区域，并定义了每个区域的位置、大小、访问权限和存储属性，它支持：

- 每个区域的独立属性设置
- 重叠区域
- 将存储属性导出到系统

存储属性会影响对该区域的存储访问操作。i.MXRT的MPU定义了：

- 16个独立的存储区（0-15）
- 一个背景区域

当存储区域重叠时，对一个存储区的访问会受到最大编号区域的属性的影响。例如，区域15的属性优先于与区域15重叠的任何区域的属性。背景区域具有与默认存储区映射相同的存储区访问属性，但只能由有特权软件进行访问。

MPU的存储区映射是统一的。这意味着指令访问和数据访问具有相同的区域设置。如果一个程序访问了MPU禁止的存储区位置，处理器会产生一个MemManage故障，这会导致一个故障异常，并可能导致一个操作系统环境中的进程终止。

通常，应用程序或嵌入式操作系统使用MPU进行存储区保护和存储区缓存策略的配置。有关如何使用MPU为不同的缓存策略配置存储区的信息，请参见第4.2节。

2.5. 硬件L1 I-cache（指令缓存）的预取

推测性获取可能是由Cortex-M7中的分支预测功能引起的，当启用了分支预测器时，内核会尝试获取当前执行点之前的内容，而当分支预测器禁用时，内核仍会执行少量预测（后向直接分支将被预测为已采用，前向直接分支会被预测为未采用），因此即使禁用了分支预测，内核也有很小的可能去获取一些不期望获取的位置。

注意

推测性获取会对普通存储区（Normal Memory）执行，但对强序存储区（Strongly Ordered）或设备存储区（Device）没有影响，且永远不会对XN存储器执行推测性指令获取（请参见第2.3节的存储类型）。

3. 缓存操作

缓存操作有三种类型：

- **缓存启用/禁用（Cache Enable/Disable）**——开/关缓存
- **缓存清除（Cache Clean）**——将脏（dirty）缓存行写回到存储区（有时称为刷新）
- **缓存失效（Cache Invalidate）**——将缓存中的内容标记为无效（基本上是一个删除操作）

i.MXRT SDK提供了两种方式来执行缓存操作

3.1. 使用CMSIS访问缓存

表3. CMSIS缓存函数

CMSIS函数	描述
void SCB_EnableICache (void)	使指令缓存失效，之后启用指令缓存
void SCB_DisableICache (void)	禁用指令缓存，并使其内容失效
void SCB_InvalidateICache (void)	使指令缓存失效
void SCB_EnableDCache (void)	使数据缓存失效，之后启用数据缓存
void SCB_DisableDCache (void)	禁用数据缓存，之后清除其内容并使其失效
void SCB_InvalidateDCache (void)	使数据缓存失效
void SCB_CleanDCache (void)	清除数据缓存
void SCB_CleanInvalidateDCache (void)	清除数据缓存并使其失效

void SCB_InvalidateDCache_by_Addr (uint32_t *addr, int32_t dsize)	按地址使数据缓存失效。 @地址必须与32字节边界对齐 @数据大小以字节为单位
void SCB_CleanDCache_by_Addr (uint32_t *addr, int32_t dsize)	按地址清除数据缓存 @地址必须与32字节边界对齐 @数据大小以字节为单位
void SCB_CleanInvalidateDCache_by_Addr (uint32_t *addr, int32_t dsize)	按地址清除并使数据缓存失效 @地址必须与32字节边界对齐 @数据大小以字节为单位

3.2. 使用SDK访问缓存

i.MXRT SDK为L1缓存的操作提供了一个缓存驱动程序，此程序封装了那些CMSIS缓存函数：

```
void L1CACHE_DisableICache(void)
void L1CACHE_InvalidateICache(void)
void L1CACHE_InvalidateICacheByRange(uint32_t address, uint32_t size_byte)
void L1CACHE_EnableDCache(void)
void L1CACHE_DisableDCache(void)
void L1CACHE_InvalidateDCache(void)
void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)
void L1CACHE_CleanDCacheByRange(uint32_t address, uint32_t size_byte)
void L1CACHE_CleanInvalidateDCacheByRange(uint32_t address, void uint32_t size_byte)
```

更多详细信息，请参见SDK参考手册。

4. 缓存的维护和数据一致性

缓存可大幅地提升系统性能，但用户必须注意缓存的维护来确保数据的一致性。

4.1. 典型用例

为了更好地理解缓存的维护和数据一致性，本节以一个典型实例为例：播放存储在外部Flash（闪存）中的一个音频文件。子系统互联和程序数据流如下：

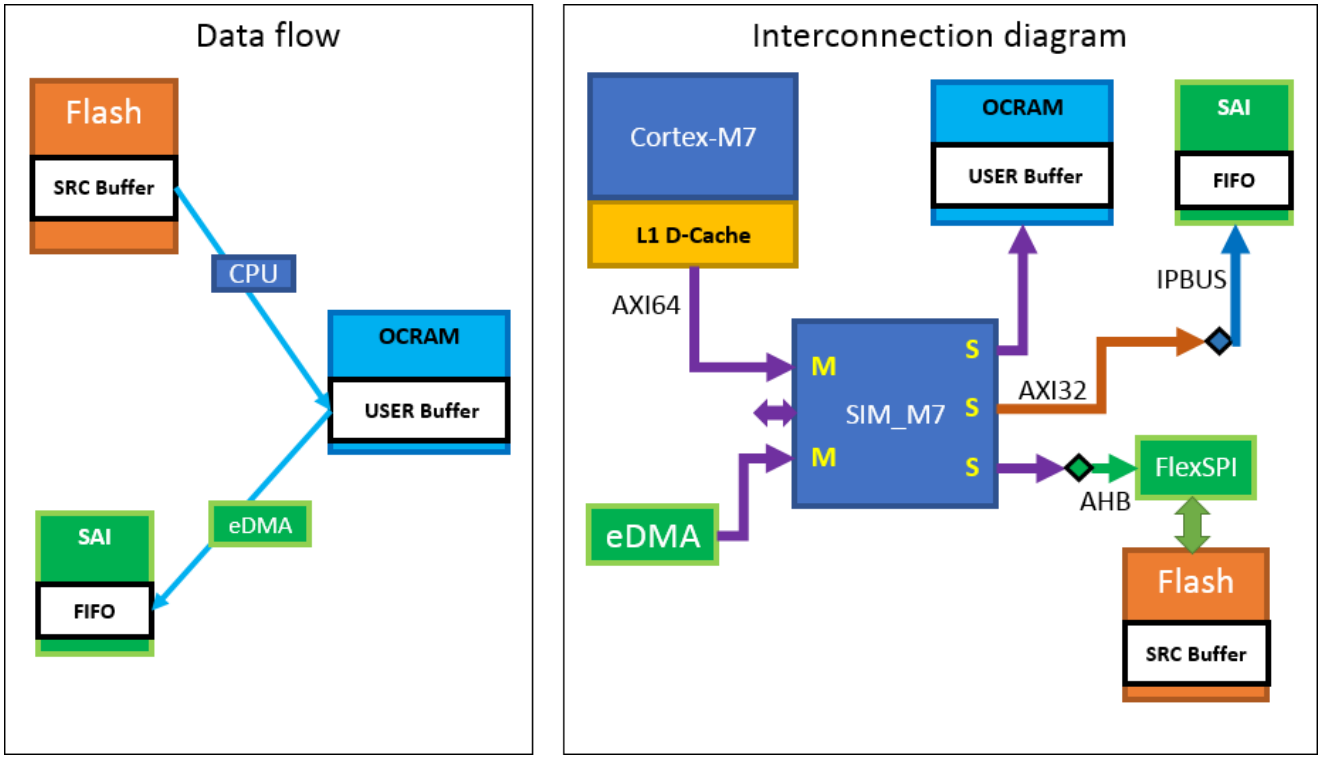

图2. 数据流和子系统框图

CPU通过L1数据缓存（D-Cache）读取SRC缓冲区中的音频文件内容，并对PCM帧数据进行解码，写入OCRAM的用户缓冲区。当用户缓冲区满时，启动eDMA，将PCM帧数据复制到SAI IP模块内的FIFO中。之后SAI将FIFO内的数据移出到SAI总线进行音频播放。当CPU启用L1缓存将帧数据写入OCRAM时，数据可能只被写入缓存，因为OCRAM的默认缓存策略是回写（Write-Back）。那么eDMA传输到SAI FIFO的数据是不正确的，随之会产生数据一致性的问题。

为了避免此类数据一致性的问题，有以下解决方案：

1. 在CPU将数据写入OCRAM后执行D-Cache清除操作。
2. 在本次写入开始之前，在MPU中将OCRAM存储区的缓存策略从回写（Write-Back）更改为透写（Write-Through）。
3. 在MPU中将OCRAM存储区的缓存策略配置为不可缓存。
4. 在MPU中将OCRAM存储区配置为可共享，这意味着不可缓存。

4.2. SDK驱动程序中的缓存维护

SDK中的以下驱动程序可维护缓存的数据一致性：

1. **以太网**

以太网中设计了统一的DMA（uDMA）引擎，优化了以太网内核与SoC之间的数据传输，并支持增强型缓冲描述符编程模型，以支持IEEE1588的功能。

2. **uSDHC**

   在SD主机控制器标准中，设计了一种新的DMA传输算法，称为ADMA（高级DMA）。

   用户可以将可缓存的缓冲内容传递给这些驱动程序，驱动程序负责处理数据一致性。对于使用 DMA的其他情况，用户应通过缓存操作来处理数据的一致性。请参见下一节。

4.3. 应用中的缓存维护

有两种方法可在应用程序中进行缓存维护。

4.3.1. 使用可缓存的缓冲区

通常，OCRAM和SDRAM上的缓冲区是可缓存和回写的。它可以在栈、静态段中或从堆中分配。要使用此类缓冲区作为DMA的源，用户必须在DMA启动前执行一个DCACHE清除操作，这可确保缓存中的所有数据都写入到了存储器中。如果缓冲区被用作DMA的接收目标，则必须在DMA完成后且CPU或其他主设备读取之前执行一个DCACHE失效操作。缓冲区的地址应与L1缓存行的大小对应（在i.MXRT中为32字节）。

4.3.2. 使用不可缓存的缓冲区

使用不可缓存的缓冲区会使事情更容易，可避免出现缓存数据的一致性问题。但副作用是，如果CPU多次访问这些缓冲区，则性能就不如访问可缓存缓冲区。

要让缓冲区不可缓存，用户必须在MPU中将至少一个存储区配置为不可缓存的属性，并通过工具链的链接器将缓冲区放入该区域。

在应用中要执行的步骤：

1. **缓冲区的定义**

   SDK为应用程序提供了以下两个宏，用于在程序的“Noncacheable（不可缓存）”段定义缓冲区（变量）：

```
AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)
AT NONCACHEABLE  SECTION(var)
```

第一个宏定义了起始地址与alignbytes对齐的缓冲区（变量）。第二个宏定义了起始地址由编译器自动对齐的缓冲区（变量），通常为4字节对齐。某些用例需要应用程序来明确定义与特殊字节对齐的缓冲区。例如，eLCDIF的帧缓冲区需要8字节对齐：

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

2. 链接器文件

添加NCACHE_VAR块（不可缓存的段），其大小应符合用户缓冲区的要求。将此NCACHE_VAR块放入SDRAM区域，例如SDRAM的最开头。

```
define symbol m_sdram_start           = 0x80000000;
define symbol m_sdram_end             = 0x80FFFFFF;

define region SDRAM_region = mem:[from m_sdram_start to m_sdram_end];

define block NCACHE_VAR with size = 2*1024*1024, alignment = 1024*1024 { section NonCacheable };


place in SDRAM_region                 { first block NCACHE_VAR };
```

3. MPU的配置

在用户可配置任何区域之前，必须先通过ARM_MPU_Disable()函数禁用MPU。要配置这些区域，ARM_MPU_SetRegionEx()函数将第一个参数作为Region Number（区域编号），第二个参数作为要配置的存储器的基地址。第三个参数是Region Attributes（区域属性）和Size（大小），宏定义如下：

ARM_MPU_RASR（XN、AP、TEX、S、C、B、SRD、Size）

- XN/AP/TEX/S/C/B参数的属性与第2.2节中的定义完全相同
- SRD表示禁用Subregion（子区域），用于区域重叠的情况。这里只需将其忽略并设置为0。
- 大小定义为：***区域大小字节数*** = $2^{(SIZE+1)}$

传递给ARM_MPU_SetRegionEx()的区域基地址必须根据其大小的字节数来对齐。ARM_MPU_SetRegionEx()可以多次调用来配置那些具有唯一区域号的不同存储区。MPU最多支持16个区域。在存储区配置完成后，调用ARM_MPU_Enable()来启用MPU。ARM_MPU_Enable()的参数设置为0x4，从而允许使用默认存储区映射作为背景区域。

例如，将SDRAM（从0x80000000开始）的前2MB区域配置为不可缓存：

```
ARM_MPU_Disable();

// Configure for a non-cache region in beginning of SDRAM (2MB)
ARM_MPU_SetRegionEx(0, 0x80000000,
                ARM_MPU_RASR(1, ARM_MPU_AP_FULL, 0x1, 0, 0, 0, 0,
                                ARM_MPU_REGION_SIZE_2MB));

ARM_MPU_Enable(0x4);
```

5. 限制推测性预取

由于Cortex-M7支持推测性预取功能，可以随时对具有普通存储属性的存储区位置进行推测性访问，如果预取发生在无效地址上，就会产生总线故障，因此必须避免这一问题的发生。

由于推测性获取永远不会对强序存储区或设备存储区执行（参见第2.3节存储类型），因此可以采用配置MPU的方法来限制其操作。

需要考虑以下的MPU配置：

- 将所使用的存储区配置为普通存储区（Normal Memory）

  I.MXRT为一些设备保留了存储区地址，需要将所使用的这些地址空间配置为普通存储区（Normal Memory）。例如，对于SDRAM，需要将有效的地址空间配置为普通存储区（Normal Memory），且MPU的配置如下：

```
/* Region 7 setting */
ARM_MPU_SetRegion ( ARM_MPU_RBAR ( 7 , 0x80000000U ) ,
                            ARM_MPU_RASR(0, ARM_MPU_AP_FULL, 0, 0, 1, 1, 0,
                          ARM_MPU_REGION_SIZE_32MB) ) ;
```

- 将所有未使用的地址空间配置为设备存储区（Device Memory）。

  由于并非所有存储区都用于同一个应用，因此要将所有未使用的存储区配置为设备存储区（Device Memory），例如，如果板上没有安装外部闪存（通过FlexSPI接口），则需要将相应的地址区域配置为设备存储区（Device Memory）类型，如下所示。

```
/* Region 2 setting */
ARM_MPU_SetRegion ( ARM_MPU_RBAR ( 2 , 0x60000000U ) ,
                            ARM_MPU_RASR(0, ARM_MPU_AP_FULL, 2, 0, 0, 0, 0,
                          ARM_MPU_REGION_SIZE_512MB) ) ;
```

- 需要确保所有的存储区空间都在MPU上得到正确的配置。

  请根据应用的需求配置合适的存储属性，检查已使用/未使用的存储区及有效的地址空间，分配正确的存储类型，以避免由推测性预取引起的问题，另外I.MXRT SDK

还提供了MPU的初始化驱动程序（BOARD_ConfigMPU(void)），这是一个配置MPU的示例。

6. 结论

总之，要以一个正确且高效的方式使用i.MXRT的L1缓存，有以下几个建议和提示：

- 将关键代码和数据（如向量表）放入TCM中，这是CPU访问代码和数据最快的方式。
- 始终调用CMSIS缓存函数或SDK缓存驱动程序API来进行缓存操作。这样可确保缓存在禁用之前被清除，并在启用之前失效，以避免不可预测的问题。
- 如果软件使用可缓存的存储区作为DMA源/或目标缓冲区，则它必须在启动DMA操作之前触发缓存清除操作，以确保所有数据都提交到了子系统存储区。在DMA传输完成后，当从外设读取数据时，软件必须在读取DMA更新的存储区之前执行缓存失效操作。
- 始终建议以不可缓存的区域作为DMA缓冲区。软件可以使用MPU配置一个不可缓存的存储区作为CPU和DMA之间的共享缓冲区。例如：
  - eLCDIF显示器的帧缓冲区
  - PXP通道的输入和输出缓冲区
- 当使用FlexSPI通过AHB总线读取外部NOR Flash时，可缓存的存储区会引起问题。因为Flash的擦除和烧录操作是通过IP命令进行的，而不是通过AHB总线进行的。因此在任何擦除和烧录操作完成后，及CPU读取FlexSPI存储区映射之前，都需要进行缓存失效操作。

7. 参考资料

- 《Arm Cortex-M7处理器用户指南》（修订版：r1p1）
- 《Arm Cortex-M7处理器技术参考手册》（修订版：r1p1）
- 《i.MX RT1050处理器参考手册》
- 《Kinetis SDK v.2.2 API参考手册》（在SDK发布包中）
- 《缓存（计算机）》——https://en.wikipedia.org/wiki/Cache_(computing)

8. 修订历史

表4. 修订历史

版本号	日期	实质性变更
第0版	2017年8月	初版发布
第1版	2017年12月	添加第5章和2.5章

How to Reach Us:

Home Page:
nxp.com.cn

Web Support:
nxp.com.cn/support

Document Number: AN12042
Rev. 1
12/2017