STM32F7 MPU Cache 浅析

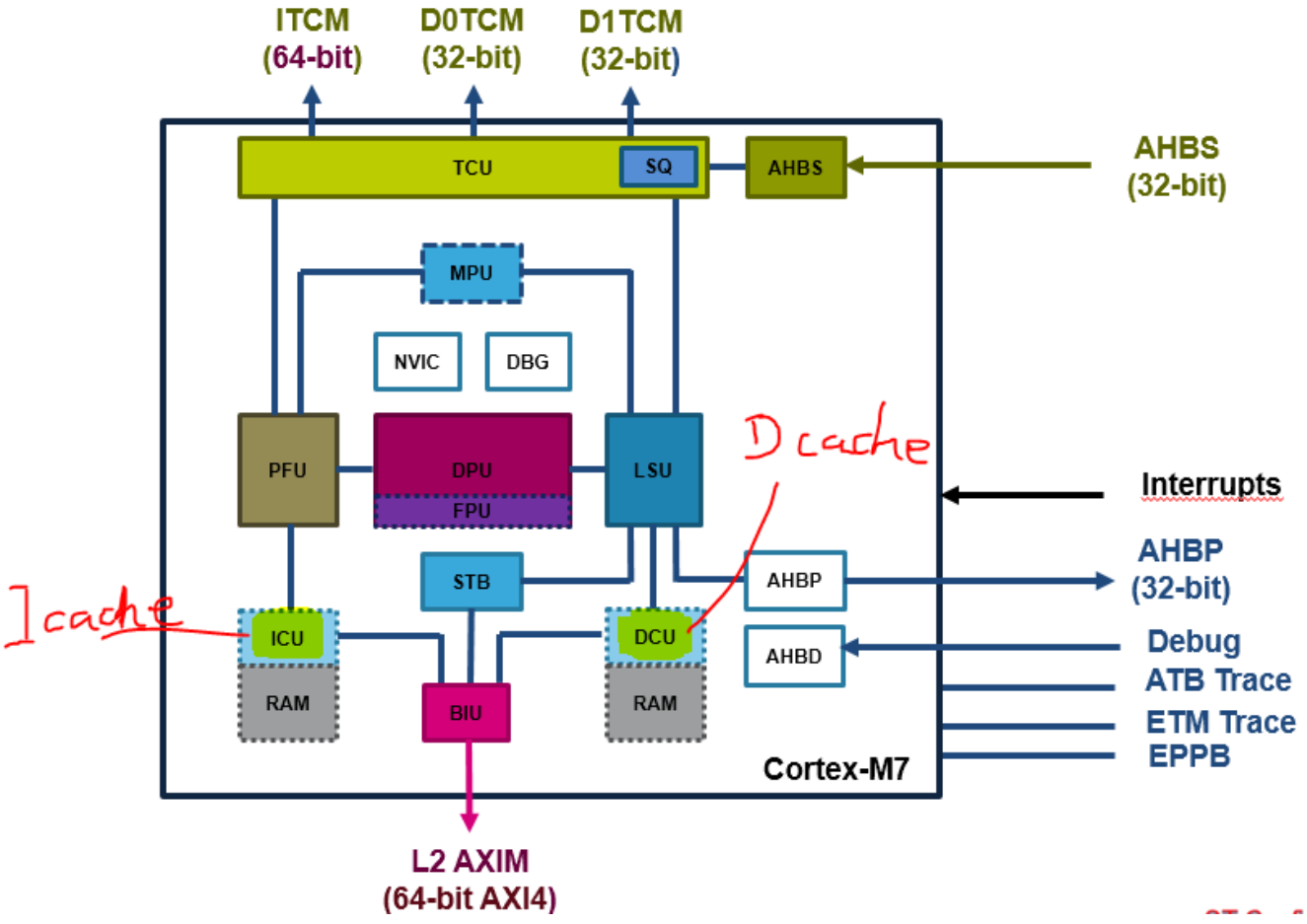
前言

本文会从结构，原理以及应用方面对 MPU 和 Cache 进行分析，主要目的是希望读者对 Cache 有基本的了解，在具体的实际应用中，使用带有一级 cache 的 MCU 时，避免常见的错误。

Cache 介绍

Cache 及其原理

Cache，高速缓存，一般指的是 L1 cache，即和 Core 紧耦合的，从 STM32F7 系列开始，基于 ARM cortex-M7 内核，增加了对 L1 Cache 的支持。

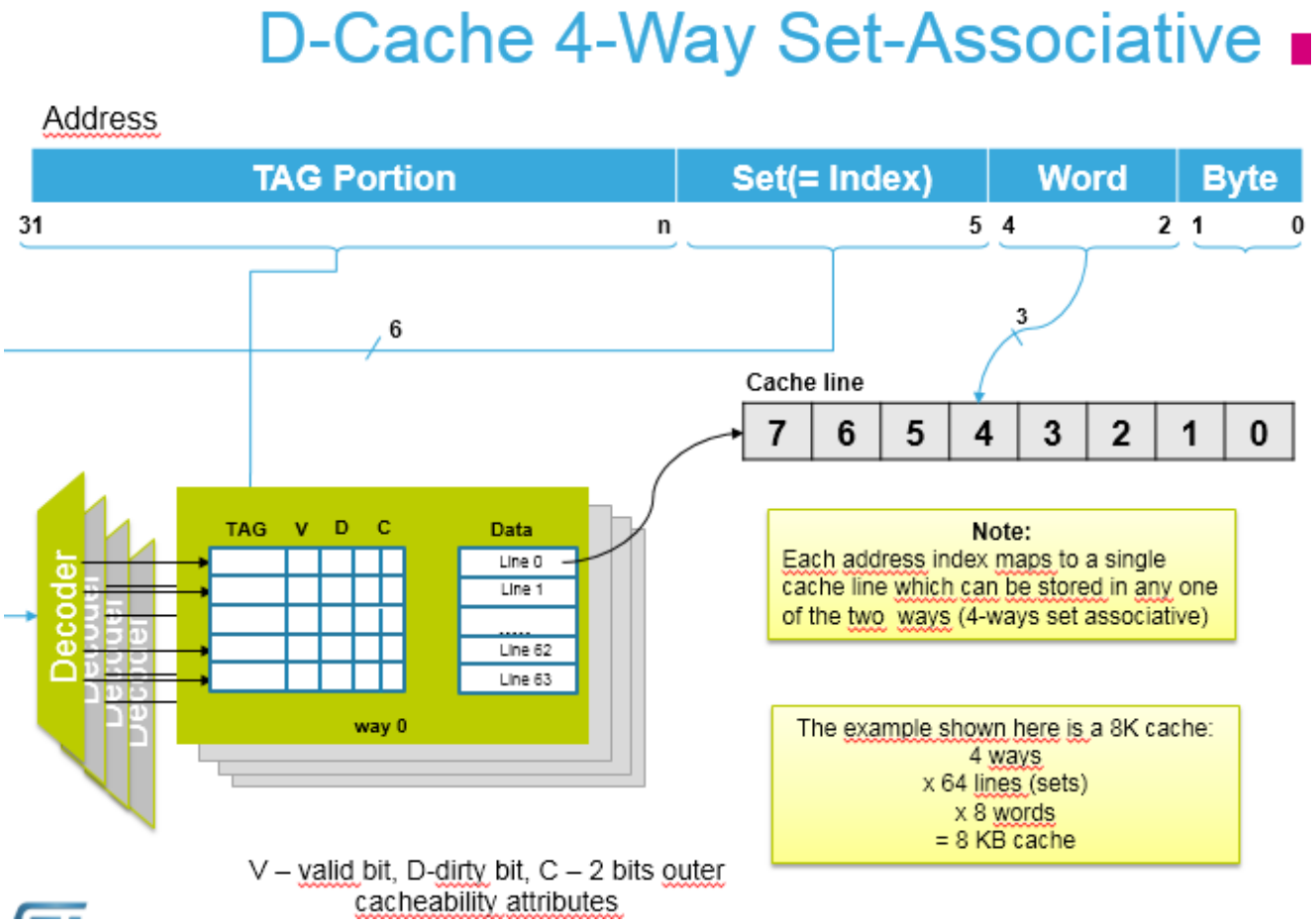


从这张图片可以看出来，无论是指令 Cache(I-cache)还是数据 Cache(D-cache)，一旦使能之后，就分别与 Core 的 prefetch unit(I-cache)和 load-store unit(D-cache)相连，以 D-cache 为例，因为

直接与 LSU 相连，访问速度会比直接访问 SRAM 或外部 RAM 快很多，只要保证 cache 有足够高的命中率(由 cache 策略保证)，尽量少的 cache miss，读/写的速度会有比较大的提高。

结构及策略

同样这里以 D-cache 为例，看一下 D-cache 的构成：



包括 Address 和 cache-line，Address 表明其地址，对应一条包含 32bytes 的 cache-line：

读数据时，当地址命中时即 cache-hit，便可以直接从 cache line 中取出相应的数据，反之，当遍历了 address 都没有找到，就会产生 cache-miss，这时便会从实际的内存单元(如 SRAM)中取出相应的数据，并更新到某一条 cache-line 中并修改相应的 cache-line 信息；

写数据时，就有点不同了，包括 write-through 策略和 write-back 策略，当使用 write-though 策略时，更新 cache-line 的同时，同样会更新其对应的实际物理地址的区域，当采用 write-back 策略时，更新 cache-line 的同时，并不是马上去更新其对应的实际物理地址的内容，而是在其认为合适或者所有的 cache-line 都 dirty 的时候才会去更新，当然，也可以通过软件让其强制更新，即 clean 的动作，这一块会在后面的 cache 一致性问题上也会有体现；

同样，对于为什么将 cache 拆分为 2-way 或是 4-way，这和 cache 自身的策略如查找算法等相关，由于本文侧重讨论 cache 的应用相关问题，所以关于 cache 本身的策略这里不再详述。

Cache 及 MPU 属性

这里需要注意的是，cache 一般是配合 MPU(memory protection unit)一起使用的，首先需要通过 MPU 配置相应 memory 的属性(normal, strongly-ordered, device, XN etc.)，如下表所示：

Address	Name	Memory Type	XN ?	Cache	Description
0x0000 0000 0x1FFF FFFF	Code	Normal		WT	Typically ROM or flash memory. Memory required from address 0x0 to support the vector table for system boot code on reset
0x2000 0000 0x3FFF FFFF	SRAM	Normal	-	WBWA	SRAM region typically used for on-chip RAM
0x4000 0000 0x5FFF FFFF	Peripheral	Device	XN	-	On chip peripheral address space
0x6000 0000 0x7FFF FFFF	RAM	Normal	-	WBWA	Memory with write-back, write allocate cache attribute for L2/L3 cache support
0x8000 0000 0x9FFF FFFF	RAM	Normal	-	WT	Memory with write-through cache attribute
0xA000 0000 0xBFFF FFFF	Device	Device, Shareable	XN	-	Shared device space
0xC000 0000 0xDFFF FFFF	Device	Device, non-shareable	XN	-	Non-shared device space
0xE000 0000 0xE00F FFFF	PPB	Strongly-Ordered	XN	-	1MB region reserved as the PPB. This supports key resources, including the System Control Space and debug features.
0xE010 0000 0xFFFF FFFF	Vendor_SYS	Device	XN	-	Vendor system region

选取几个有特点的作为示例：

0~0x1FFF_FFFF: flash 空间，属性为 normal，cache 的属性为 Write-through，即更新 cache 的同时，将数据同时写入相应的物理地址空间

0x2000_0000~0x3FFF_FFFF: SRAM 空间，属性为 normal，cache 的属性为 write-back，即仅更新 cache，在合适的时候(由 cache 策略决定或者软件强制更新)将数据更新到相应的 SRAM 空间

0x4000_0000~0x5FFF_FFFF: 芯片内部的外设空间，属性为 device，这一版是外设寄存器所处的位置，对其读写的过程中不会经过 cache

XN 的意思是 Execute-Never，其含义为如果相应的地址空间是 XN，是绝不允许执行代码的。

Cache 相关函数及作用

这里以 core_cm7.h 里对 cache 封装的函数为例

(C:\Program Files (x86)\IAR Systems\Embedded Workbench 7.2\arm\CMSIS\Include)

SCB_EnableDCache

使能 D-cache

SCB_DisableDCache

禁用 D-cache

SCB_EnableICache

使能 I-cache

SCB_DisableICache

禁用 I-cache

SCB_CleanDCache

Clean 所有的 cache-line，即将 dirty 的 cache-line 全部写到 cache line 对应的真实的物理地址中

所谓的 drity 属性，即写操作时，更新了相应的 cache-line，但是没有更新到真实的物理地址，

而这个 clean 的动作，就是将 cache 中的内容更新到真实的物理地址中

SCB_CleanDCache_by_Addr

根据地址信息 clean 其对应的 cache-line

SCB_InvalidateDCache

无效 D-cache，D-cache 被 invalidate 之后，当有 Host(如 core，DMA 等)读取数据时，会忽略相应的 cache-line 中的内容（因为被 validate 了），从真实的物理地址中去获取相应的数据

SCB_InvalidateDCache_by_Addr

根据地址信息无效其对应的 cache-line

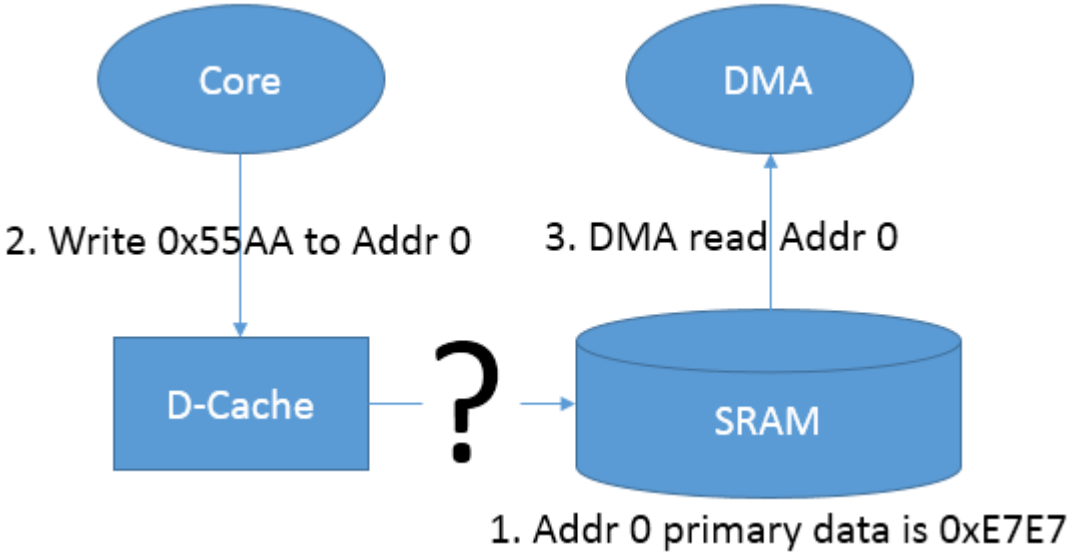
Cache 一致性问题

所谓的 Cache 一致性问题，主要指的是由于 D-cache 存在时，表现在有多个 Host（典型的如 MCU 的 core，DMA 等）访问同一块内存时，由于数据会缓存在 D-cache 中而没有更新实际的物理内存。

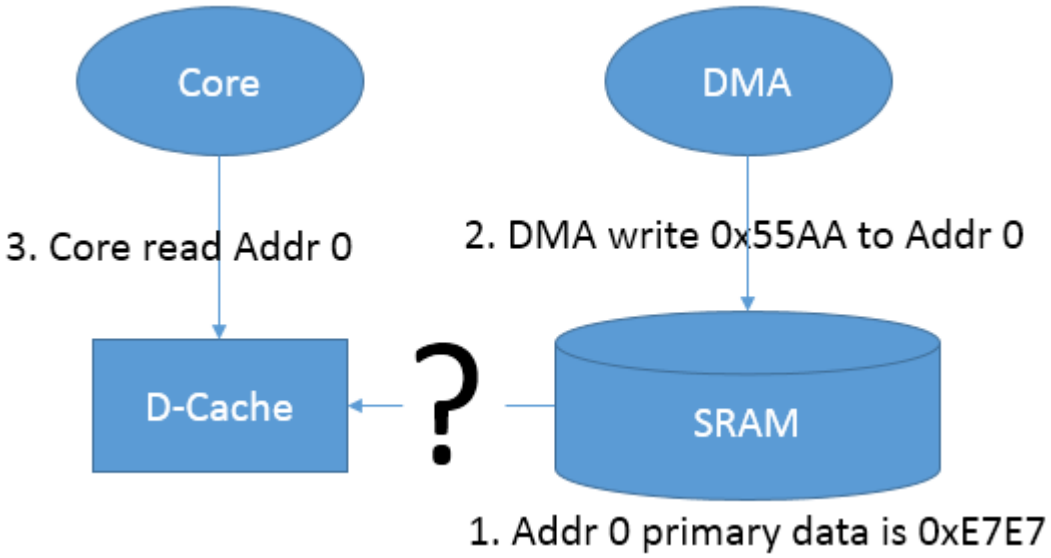
第一种情况是当有写物理内存的指令时，core 会先去更新相应的 cache-line(Write-back 策略)，在没有 clean 的情况下，会导致其对应的实际物理内存中的数据并没有被更新，如果这个时候有其它的 Host（如 DMA）访问这段内存时，就会出现问题（由于实际物理内存并未被更新，和 D-cache 中的不一致），这就是所谓的 cache 一致性的问题！

第二种情况是 DMA 更新了某段物理内存（DMA 和 cache 直接没有直接通道），而这个时候 Core 再读取这段内存的时候，由于相对应地址的 cache-line 没有被 invalidate，导致 Core 读到的是 cache-line 中的数据，而非被 DMA 更新过的实际物理内存的数据，下面这张图比较清晰的展示了上述两个过程：

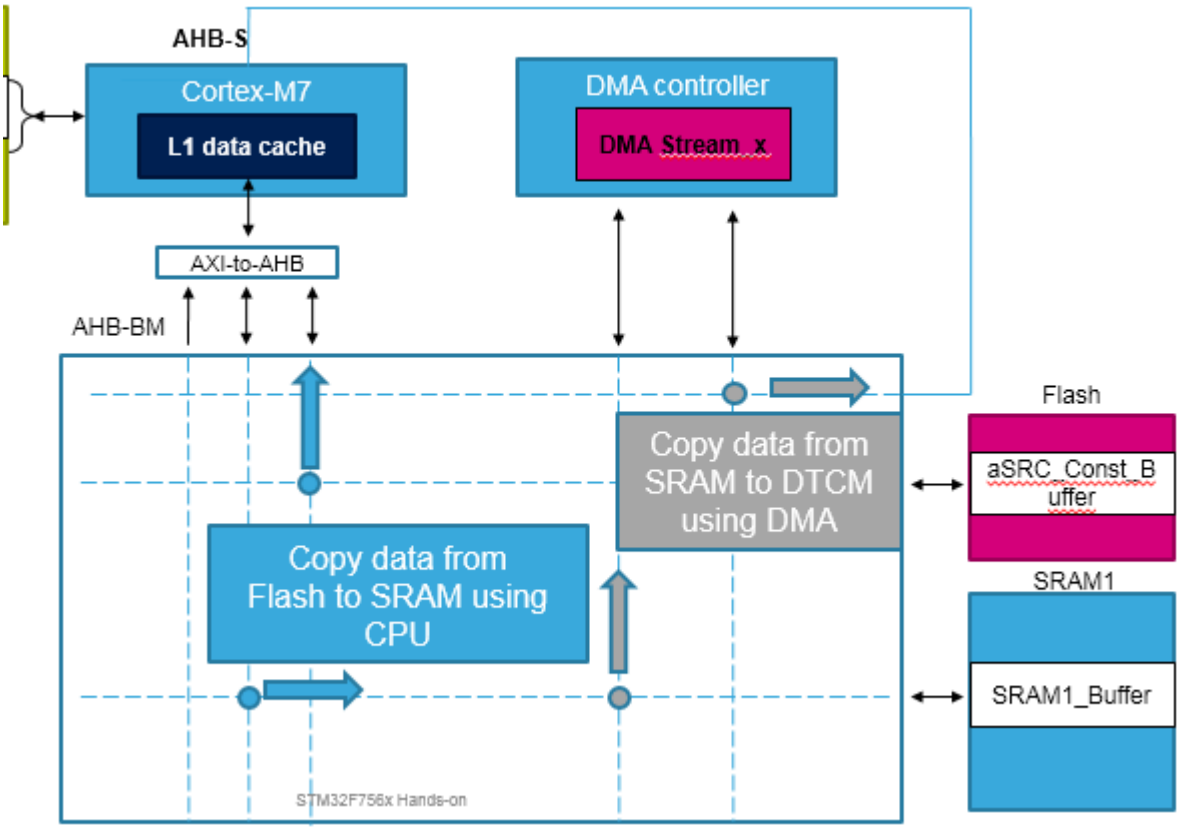
- 第一种情况



- 第二种情况



下面以一个实例来分析 cache 一致性问题，展示的是上面的第一种情况，如下图所示：



先看一下这个例程数据的传输流程和路径：

- SRAM1_Buffer 先全部写入 0x55
- Core 将 Flash 中的 Const_Buffer 写入 SRAM1_Buffer（这里会先经过 d-cache）

- 配置 DMA，将 SRAM1_Buffer 中的数据通过 DMA 写入另一段内存 DTCM_Buffer
- 比较 DTCM_Buffer 中的数据和 Flash 中的 Const_Buffer 数据，看是否一致

代码示例如下：

- MPU 对 memory 的配置 - - - - step1

```
static void MPU_Config(void)
{
  MPU_Region_InitTypeDef MPU_InitStruct;

  /* Disable the MPU */
  HAL_MPU_Disable();

  /* Configure the MPU attributes as WT for SRAM */
  MPU_InitStruct.Enable = MPU_REGION_ENABLE;
  MPU_InitStruct.BaseAddress = 0x20010000;
  MPU_InitStruct.Size = MPU_REGION_SIZE_256KB;
  MPU_InitStruct.AccessPermission = MPU_REGION_FULL_ACCESS;
  MPU_InitStruct.IsBufferable = MPU_ACCESS_NOT_BUFFERABLE;
  MPU_InitStruct.IsCacheable = MPU_ACCESS_CACHEABLE;
  MPU_InitStruct.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
  MPU_InitStruct.Number = MPU_REGION_NUMBER0;
  MPU_InitStruct.TypeExtField = MPU_TEX_LEVEL0;
  MPU_InitStruct.SubRegionDisable = 0x00;
  MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_ENABLE;

  HAL_MPU_ConfigRegion(&MPU_InitStruct);

  /* Enable the MPU */
  HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);
}
```

重点介绍一下高亮部分的配置：

- BaseAddress 为要配置的存储空间的起始地址；
- Size 为要配置的存储空间的大小
- IsCacheable 表明这段存储空间是否可以 cache
- IsBufferable 表明使能 cache 之后，策略是 write-through 还是 write-back(bufferable)

这里需要特别注意的 1 点：配置的 BaseAddress 需要被 Size 整除，以上述配置为例，即 0x20010000 除以 256K 需要是整数！

- 使能 Cache - - - - step2

```
static void CPU_CACHE_Enable(void)
{
  /* Enable I-Cache */
  SCB_EnableICache();

  /* Enable D-Cache */
  SCB_EnableDCache();
}
```

- 初始化 SRAM1_Buffer 为 0x55 ---- step3
- Copy Flash 中的 Const_Buffer 到 SRAM1_Buffer ---- step4
- 配置 DMA，将 SRAM1_Buffer 写入 DTCM_Buffer ---- step5
- 比较 DTCM_Buffer 和 Const_Buffer，看是否一致 ---- step6

从结果上看，最后一步比较的结果并不一致，原因比较简单，由于设定的 WB 策略，所以在 step4 的时候，数据会暂存在 D-cache 当中，并没有更新到 SRAM1_Buffer，所以当 SRAM1_Buffer 被 DMA 拷到 DTCM_Buffer 中的时候，有一部分可能还是初始值，导致最终的比较不一样，而解决的方法有以下几个：

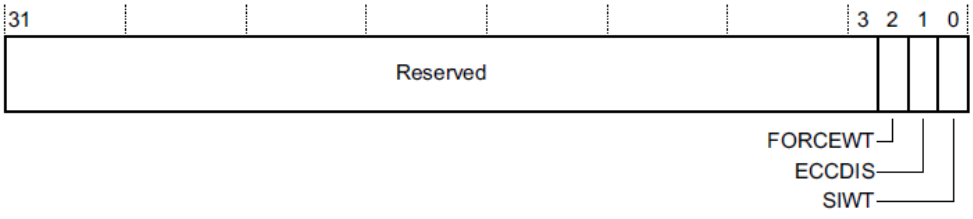
1. MPU 配置的代码，将属性改为 MPU_ACCESS_BUFFERABLE，即使用 write-though 策略
2. 通过 cache 控制寄存器，将所有 cacheable 的空间全部强制 write-though

3.3.8 L1 Cache Control Register

The CM7_CACR characteristics are:

Purpose	Controls the L1 ECC and the L1 cache coherency usage model.
Usage Constraints	Accessible in privileged mode only.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 3-1 on page 3-3.

Figure 3-8 shows the CM7_CACR bit assignments.



Bits	Name	Type	Function
[31:3]	-	-	Reserved, RAZ/WI.
[2]	FORCEWT	RW	Enables Force Write-Through in the data cache: 0 Disables Force Write-Through. 1 Enables Force Write-Through. All Cacheable memory regions are treated as Write-Through. This bit is RAZ/WI if the data cache is excluded. If the data cache is included the reset value of FORCEWT is 0.

3. 将 dirty cache-line 更新到真实的物理地址中

   在 step5 操作之前，先调用 SCB_CleanDCache 或 SCB_CleanDCache_by_Addr 将相应 cache-line 中的数据写入 SRAM1_Buffer，就解决了这个问题！

这是最常用的方法，在实际的开发过程中，为了提高性能，一般都会使能 cache，同时将其配置为 WB 策略，这就需要开发者在使用时特别小心！同样如之前的第二种情况，需要先调用 SCB_InvalidateDCache 或 SCB_InvalidateDCache_by_Addr 去 Invalidate 相应的 cache-line，这样当 core 在读取时，会忽略 D-cache 中的内容，去真实的物理地址读取对应的数据！