TI C64x+ DSP CACHE 一致性分析与维护

德州仪器 DSP 技术应用工程师 宋洋

Abstract

CACHE is widely used to fill in the gap between Core and low speed memory in order to get better system performance. When applying CACHE in system, the inconsistencies of data in different memory hierarchy need to be avoided. This article analyzes the scenarios of CACHE coherency on TI C64x+ platform and also describes how to maintain them appropriately.

摘要

在各种数字信号处理系统中，CACHE 被广泛用于弥补 Core 与存储器之间的速度差异。在 CACHE 的使用过程中，存在不同类型存储器之间数据是否一致的问题。本文着重分析 TI 高性能 C64x+ DSP 系列中各级 CACHE 之间数据一致性问题以及如何进行一致性维护。

1. 概述

CACHE 作为 Core 和低速存储器之间的桥梁，基于代码和数据的时间和空间相关性，以块为单位由硬件控制器自动加载 Core 所需要的代码和数据。如果所有程序和数据的存取都由 Core 完成，基于 CACHE 的运行机制，Core 始终能够得到存储器中最新的数据。但是当有其它可以更改存储器内容的部件存在时，例如不需要 Core 干预的直接数据存取（DMA）引擎，就可能出现由于 CACHE 的存在而导致 Core 或者 DMA 不能够得到最新数据的现象，也就是 CACHE 一致性的问题。

2. C64x+ 存储器组织结构

TI 对高性能 C64x 核进行了改进，使其性能大大提升，称之为 C64x+DSP 核。基于 C64x+ 核开发的 DSP 芯片，所有部件都以交换网络（SCR）为核心连接起来。SCR 上的部件分为两类：Master 和 Slave。Master 包括 Core、EDMA 以及串行高速 IO（sRIO），EMAC 等外设。Master 可以直接通过 SCR 发起到 Slave 的数据传输。Slave 包括每一个 Core 的内存，DDR2 外存以及其它不能直接发起数据传输的外设，Slave 之间的数据传输，需要通过 DMA 协助完成。各款基于 C64x+DSP 的数据手册上详细描述了 SCR 的配置和 Master、Slave 的情况。

C64x+系统的存储器框图如图 1 所示。存储器被分成了三级：第一级是 L1，包含数据存储器（L1D）和代码存储器(L1P)；第二级是代码和数据共用存储器（L2）；第三级是外

部存储器，主要是 DDR2 存储器。L1P、L1D 和 L2 的 CACHE 功能分别由相应的 L1P 控制器、L1D 控制器和 L2 控制器完成。表 1 总结了 C64x+平台上可用的 CACHE 情况。

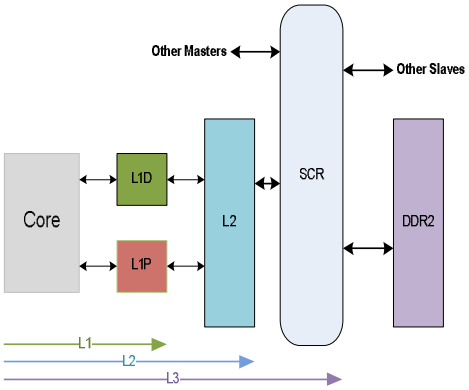

图 1 C64x+存储器框图

表 1 C64x+ CACHE 特性

	类型	大小	Line 大小
L1P	代码；直接映射；	最大 32K 字节	32 字节
L1D	数据；2 路； 读分配	最大 32K 字节	64 字节
L2	代码、数据；4 路；读写分配	最大 256K 字节	128 字节

C64x+平台上 L1P 用来存储或者缓存代码；L1D 用来存储或者缓存数据。L1P 和 L1D 大小都是 32K 字节，可以分别配置 0K、4KB、8KB、16KB 或者 32KB 作为 CACHE，其余作为代码或者数据 RAM。作为 CACHE 的部分，用来缓存 L2 和 DDR2 的数据或代码。作为 RAM 的部分，可以存储关键的代码或者数据使得 Core 能够以很高的速度访问。

C64x+平台上 L2 存储器可用于存储代码和数据。L2 上最大可以分配 256K 字节 CACHE 来缓存 DDR2 中的数据或代码。L2 中其余部分作为 RAM 存储代码和数据。

图 2 描述了 Core 访问存储器内容的操作流程。在这个访问流程中，Core 对于存储器的访问总是先从离 Core 最近的一级存储器开始，如果命中，Core 可以直接得到代码/数据，否则代码/数据会被加载到前几级的 CACHE 中，从而 Core 可以得到要处理的代码/数据。在这个动态访问过程中，各级 CACHE 中的内容和下一级存储器中的内容可能存在不一致，这种瞬态的不一致不会造成问题。但是，如果 Core 或者其它 Master 不能得到另外一方对存储器内容更新后的内容，就会出现 CACHE 一致性问题。

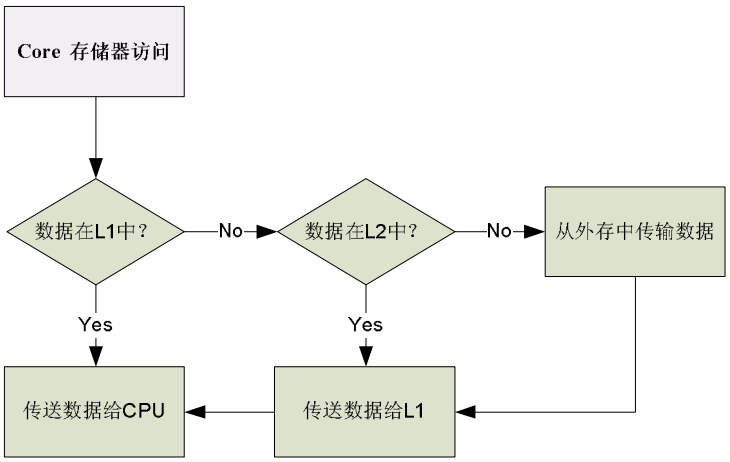

图 2 Core 访问存储器流程

3. CACHE 一致性问题分析

在任何时刻，Core 或者其它 Master 访问存储器中数据时，由于 CACHE 的存在造成不能够得到最近更新过的数据，就会出现 CACHE 一致性问题。

在一个特定的时间范围内，各级 CACHE 和它的下一级存储器中的内容不一致是正常的。因为 CACHE 的作用是在一段时间内将低速存储器中的内容自动搬运到高速的 CACHE 中重复使用。当 CACHE 中的空间被后续的数据占用的时候，才将 CACHE 中的内容进行失效或者回写的操作。在失效或者回写之前，CACHE 中的内容可能与物理存储器中的内容是不一致的。这种临时性的不一致是正常的，上述 CACHE 一致性问题的描述不包含此类正常情况。

CACHE 的引入是为了提高 Core 存取数据的效率，所以出现 CACHE 一致性问题一定与 Core 对存储器的访问有关。Core 对存储器的访问分为两类：

1. Core 读代码或者数据；
2. Core 写代码或者数据。

据此，CACHE 的一致性问题分为两个大类：Core 读一致性问题和 Core 写一致性问题。在下面两个小节中，分别描述了这两种情况的模型：

3.1 Core 读一致性模型

图 3 给出了 Core 读一致性的模型。在这个模型中，CACHE 一致性问题的存在取决于图中虚线箭头指示的第二步操作能否在 Core 从 CACHE 中重新读数据之前完成。如果不能，则会造成 Core 读取的数据不是其它 Master 更新后的数据，而是原来 CACHE 中的内容，从而导致一致性的问题。

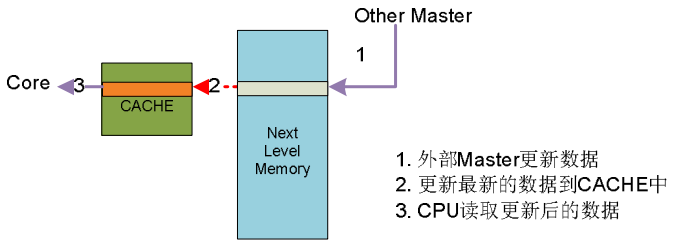

图 3 Core 读一致性模型

L1P CACHE 对 L2 内存或者 DDR2 外存中的代码进行缓存。当 Core 第一次对 L2 或者 DDR2 中的代码进行读操作的时候，由于代码不在 L1P CACHE 中，CAHCE 硬件会将 L2 或者 DDR2 中的代码读到 L1P CACHE 中。Core 可以得到最新的代码，不存在一致性的问题。此后，如果其它 Master 更新 L2 或者 DDR2 中的代码，然后 Core 再次读取此部分代码时，会发现相应的代码已经存在 L1P CACHE 中，此时 Core 会直接从 L1P CACHE 中读取代码。由于 Core 不能得到最新的代码，就出现了 Core 读一致性的问题。

L1D Core 读一致性问题的原理和 L1P 相同，只是 L1D 缓存的是 L2 或者 DDR2 中的数据。

L2 CACHE 对 DDR2 中的代码/数据进行缓存，当 Core 第一次对 DDR2 中的代码/数据进行读操作，这时代码/数据不在 L2 CACHE 中，需要进行 L2 CACHE 的加载，Core 可以得到最新的代码/数据。之后，其它 Master 对 DDR2 中的代码/数据进行更改，Core 重读此部分代码/数据的时候，Core 读到的是 L2 CACHE 中的内容而不是 DDR2 中最新的代码/数据，因此也存在 Core 读一致性的问题。

3.2 Core 写一致性模型

图 4 给出了 Core 写一致性的模型。在这个模型中，CACHE 一致性问题的存在取决于图中虚线箭头指示的第二步操作能否在其它 Master 从存储器中读数据之前完成。如果不能，会造成其它 Master 从存储器中读到的数据是原来的数据而不是 Core 更新过的数据，从而导致一致性的问题。

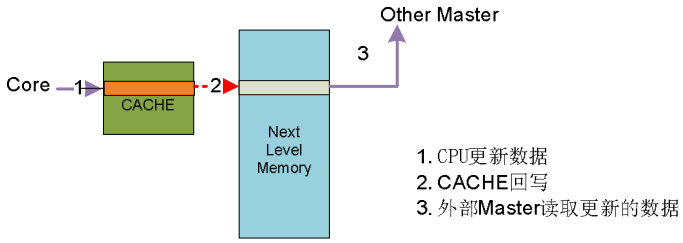

图 4 Core 写一致性模型

当 Core 对 L2 或者 DDR2 中的代码/数据进行写操作的时候，如果代码/数据已经在 L1 CACHE 中，新的代码/数据会被更新到 L1 CACHE 中。当其它 Master 从 L2 或者 DDR2 中读代码/数据的时候，会直接从 L2 或者 DDR2 中读取相应的内容，如果 L1 CACHE 中新的代码/数据未被更新到 L2 或者 DDR2 中，则其它 Master 读取的不是更新后的内容，就会出现 Core 写一致性的问题。

同样，Core 更新过的代码/数据有可能只是缓存在 L2 CACHE 中，其它 Master 从 DDR2 中读取的内容不是更新后的内容，同样会出现 Core 写一致性的问题。

3.3 C64x+一致性分析

在 C64x+上的 CACHE 一致性问题，需要根据放置代码/数据的相应位置进行分析。由于在 C64x+平台上，L1P、L1D 和 L2 内存既可以作为 CACHE 又可以作为存储器使用，因此，在分析一致性问题的时候，需要考虑以下几种情况：

Case1. 代码在 L1P 存储器中；

Case2. 代码在 L2 存储器中；

Case3. 代码在 DDR2 存储器中；

Case4. 数据在 L1D 存储器中；

Case5. 数据在 L2 存储器中；

Case6. 数据在 DDR2 存储器中。

对于 Case1，由于代码直接在 L1P 存储器中，不需要进行 CACHE，所以不会存在一致性的问题。

对于 Case2 和 Case3，涉及到 L1P CACHE，存在代码的更新能否被 Core 读到的问题。代码的更新分成两种情况：一是 Core 在运行过程中对代码进行修改；二是其它 Master 对代码的修改。这两种情况下，都会存在 CACHE 读一致性问题，需要由软件来维护。

对于 Case4，数据直接在 L1D 存储器中， Core 始终能够读到其它 Master 更新到 L1D 内存中的内容，Core 写过的数据也能够被其它 Master 直接从 L1D 内存中读到。所以不会存在一致性的问题。

对于 Case5，数据在 L2 存储器，按照上面的分析，会存在 CACHE 读和写一致性的问题。在 C64x+平台上这种情况下的一致性问题会由硬件自动维护。

对于 Case6，也会存在 CACHE 读和写一致性的问题，这种情况需要软件进行 CACHE 一致性的维护。

4. C64x+ CACHE 一致性维护操作

出现 CACHE 一致性问题时，为了保证 Core 或者其它 Master 在进行数据操作的时候能够得到最新的数据，需要进行 CACHE 的一致性维护操作。CACHE 一致性问题维护在设计中，有两种处理方式：硬件自动维护和应用程序进行维护。

下面具体分析以上几种情况在 C64x+平台上如何进行 CACHE 一致性问题处理：

4.1 硬件维护的 CACHE 一致性

在 C64x+平台上，硬件会对 Case5 的情况自动进行数据一致性维护。分析需要分为读写两类操作进行，图 5 和图 6 分别描述了 Core 对 L2 上的数据进行读和写的情况。

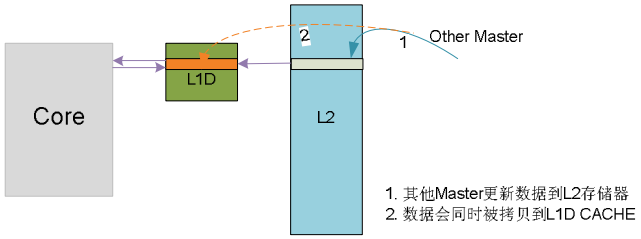

图 5 Core 读 L2 数据的情况

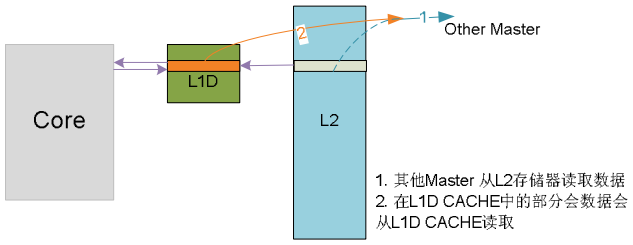

图 6 Core 写 L2 数据的情况

其它 Master 要对 L2 中的内容进行更新操作时，L2 控制器会根据被更新数据的地址判断相应的地址是否在 L1D CACHE 中，如果在 L1D CACHE 中，硬件会自动将更新的数据拷贝一份到 L1D CACHE 中。当 Core 重新对 L2 中的这部分数据进行处理的时候，如果要读取的数据已经在 L1D CACHE 中，Core 可以直接从 L1D CACHE 中得到更新过的数据。如果要读取的数据不在 L1D CACHE 中，L1D 控制器会自动从 L2 加载数据，Core 也可以得到更新后的数据。过程如图 5 中的 1 和 2 所示，这样就可以解决一致性的问题。

其它 Master 要对 L2 中的内容进行读操作的时候，L2 控制器会判断要读取的数据地址是否在 L1D CACHE 中，对于在 L1D CACHE 中的数据，硬件会自动从 L1D CACHE 中读取最新的数据。对于不在 L1D CACHE 中的数据，说明 L2 中的数据已经是最新的数据，可以直接从 L2 中读取。通过这样的处理，可以保证其它 Master 读到 Core 更新后的数据，从而可以解决一致性的问题。过程如图 6 中的 1 和 2 所示。

4.2 软件维护的 CACHE 一致性

在 C64x+平台上，Case2、Case3 和 Case6 的情况需要软件进行的一致性维护操作以保证 Core 或者其它 Master 可以得到最新的数据。

4.2.1 C64x+软件一致性维护实现

C64x+平台上由软件控制的一致性维护操作包含三种：CACHE 数据失效、CACHE 数据回写和 CACHE 数据回写并失效。启动维护操作需要配置相应的基地址和计数寄存器，当计数寄存器中的值变为 0 时表示操作完成。TI 提供的芯片支持库中也提供了相应的 API 来完成相应的功能。各种操作涉及的各级 CACHE 的一致性操作控制寄存器列在表 2 中。

表格中主要涉及以下几类寄存器的操作：

- WB：全局回写寄存器
- INV：全局失效寄存器
- WBINV：全局回写并失效寄存器

- IBAR: 部分失效基地址寄存器
- IWC：部分失效计数寄存器
- WBAR：部分回写基地址寄存器
- WWC：部分回写计数寄存器
- WIBAR：部分回写并失效基地址寄存器
- WIWC：部分回写并失效计数寄存器

表 2 C64x+ CACHE 一致性维护寄存器

	L1P CACHE	**L1D CACHE**	**L2 CACHE**
全局失效	L1PINV	L1DINV	L2INV
全局回写	不支持	L1DWB	L2WB
全局回写并失效	不支持	L1DWBINV	L2WBINV
部分失效	LIPIBAR L1PIWC	L1DIBAR L1DIWC	L2IBAR L2IWC
部分回写	不支持	L1DWBAR L1DWWC	L2WBAR L2WWC
部分回写并失效	不支持	L1DWIBAR L1DWIWC	L2WIBAR L2WIWC

例如，需要对 L2 CACHE 进行部分回写操作，需要将回写的 DDR2 的地址配置到 L2WBAR，同时将需要回写的数据 32-bit 长度写到 L2 的计数寄存器 L2WWC 中，当 L2WWC 中的值变为 0 之后，表示回写操作已经完成。

4.2.2 代码 CACHE 一致性

图 7 中描述了其它 Master 对 L2 中代码进行修改的情况。这种情况下，当 Core 第一次执行此部分代码时，这部分代码会被加载到 L1P 中。之后如果被其它 Master 修改，Core 仍会从 L1P 中读取原来的代码而不是更新后的代码。因此需要软件进行图中 2 指示的操作。软件不需要进行代码的搬移，只要在 Core 重新执行此部分代码之前将 L1P 中此部分内容失效。当 Core 再次执行此部分代码的时候，会按照 CACHE 的正常机制进行此部分代码的重新加载，从而保证 Core 可以读取到更新后的代码。操作顺序如下：

1. 其它 Master 对 L2 中的代码进行更新
2. Core 在执行此段代码之前对 L1P 中的此段地址进行失效操作

图 8 描述的是其它 Master 对 DDR2 中代码进行修改的情况。这种情况下，需要在 Core 重新执行此部分代码前，将 L1P 和 L2 CACHE 中的相应内容进行失效以保证 Core 执行时可以将最新的代码加载到 L2 和 L1P CACHE 中。操作顺序如下：

1. 其它 Master 对 DDR2 中的代码进行更新
2. Core 在执行此段代码之前对 L2 中的此段地址进行失效操作
3. Core 在执行此段代码之前对 L1P 中的此段地址进行失效操作

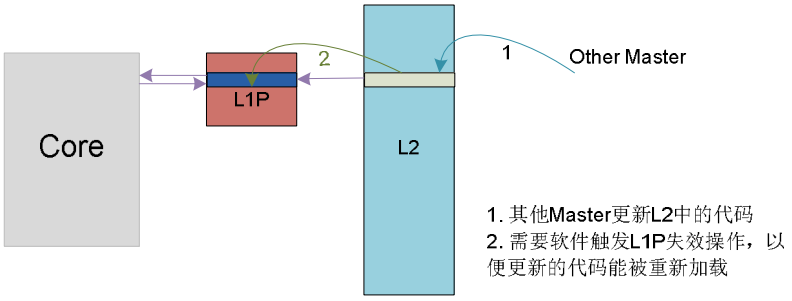

图 7 其它 Master 修改 L2 代码的情况

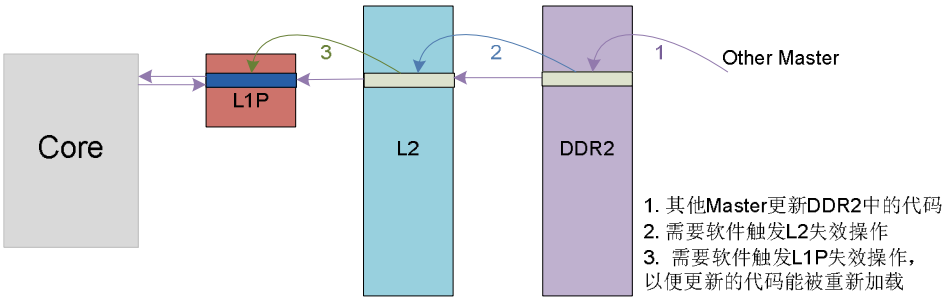

图 8 其它 Master 修改 DDR2 代码的情况

Core 对修改代码会转换为对存储器的写操作，由于 L1D 只对读不命中的情况才分配 CACHE，所操作的代码一定不在 L1D CACHE 中，更新的代码会被直接写到 L2 中，如果修改的是 DDR2 中的代码，数据可能会被更新到 L2 CACHE 中。之后的所有操作与上述两种情况的处理相同。

4.2.3 数据 CACHE 一致性

对于数据部分的一致性维护，需要由软件维护的情况是 Case6。包括 Core 对 DDR2 的读取和写两种情况。图 9 和图 10 分别描述了这两种情况。

图 9 描述的是 Core 读取 DDR2 中数据进行处理的情况。当其它 Master 对此部分数据进行更新之后，在 Core 重新读取之前，为了保证 DDR2 中的数据会被重新加载到 L1D 和 L2 中，需要将 L1D 和 L2 CACHE 中与此部分数据对应的内容失效。操作顺序如下：

1. 其它 Master 对 DDR2 中的数据进行更新
2. Core 在读此段数据之前对 L2 中的此段地址进行失效操作
3. Core 在读此段数据之前对 L1D 中的此段地址进行失效操作

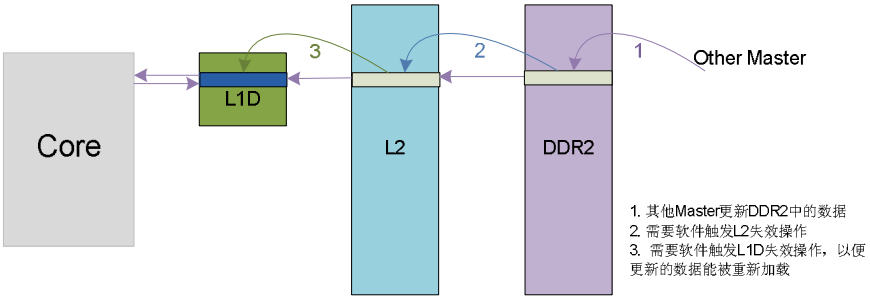

图 9 Core 对 DDR2 上的数据读的情况

图 10 描述了 Core 对 DDR2 中数据更新的情况，更新的数据可能被保存在 L1D 或者

L2 CACHE 中。为了保证其它 Master 能正确读取此部分数据，需要将 L1D 或者 L2 CACHE 中更新的此部分数据回写到 DDR2。操作顺序如下：

1. 将 L1D CACHE 中的此部分数据进行回写
2. 将 L2 CACHE 中的此部分数据进行回写
3. 其它 Master 可以从 DDR2 中读到更新后的数据

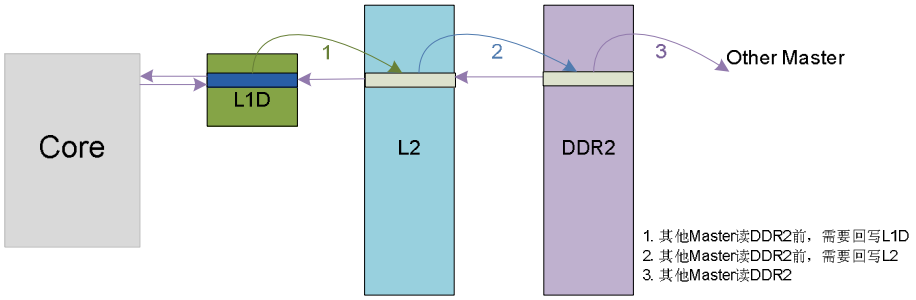

图 10 Core 对 DDR2 上的数据写的情况

结论

CACHE 一致性问题是 DSP 应用中常见的问题，TI C64x+ DSP 是业界高性能信号处理平台，具有优良的 CACHE 性能。C64x+平台 CACHE 一致性问题的维护操作情况总结如下：

表 3 C64x+平台 CACHE 一致性问题的维护操作

	是否存在 CACHE 一致性问题	硬件/软件维护
代码在 L1P 存储器中	否	不需要
代码在 L2 存储器中	是	软件
代码在 DDR2 存储器中	是	软件
数据在 L1D 存储器中	否	不需要
数据在 L2 存储器中	是	硬件
数据在 DDR2 存储器中	是	软件

C64x+平台上 CACHE 一致性问题维护可以归纳为以下两点：

1. 代码部分的一致性问题需要由软件来维护；
2. 只有当 Core 和其它 Master 共同需要访问的数据缓冲区在外部存储器中的时候，数据 CACHE 一致性问题才需要由软件来进行维护。其它情况下，数据 CACHE 一致性都会由硬件自动完成。