# 多核处理器 S698P – SOC 的数据一致性

梁宝玉[1] 颜 军[1] 侯 雄[2] 宋征宇[2]
1. 欧比特(珠海)软件工程有限公司, 珠海 519080
2. 北京航天自动控制研究所, 北京 100854

**摘 要** 在多核和多处理器系统中,数据一致性是一个非常重要和关键的问题。影响数据一致性的设计主要包括 3 个方面:处理器体系结构,Cache 算法和软件设计。本文介绍了 S698P – SOC 多核处理器的体系结构,Cache 算法和数据一致性保持机制,讨论了一般情况下的数据一致性问题,介绍了硬件对软件数据一致性设计的支持,并给出了软件设计时的设计要点。相关的技术在工程实践中已经得到验证,获得了良好的效果。

**关键词** S698P – SOC; 并行处理; Cache; 数据一致性; AMBA 总线
**中图分类号**: TP332 **文献标识码**: A
**文章编号**: 1006-3242(2008)05-0082-05

## Data Consistency of S698P-SOC

LIANG Baoyu[1] YAN Jun[1] HOU Xiong[2] SONG Zhengyu[2]
1. Orbita Software Engineering Inc., Zhuhai 519080, China
2. Beijing Aerospace Automatic Control Institute, Beijing 100854, China

**Abstract** *Data consistency is essential in design of multi – cores and multi-processor, the following three aspects influenced the design of data consistency: processor architecture, cache arithmetic and software design. This article mainly introduces the* S698P-SOC *system structure*, cache *arithmetic, data consistency, and the hardware holding updata consistency, also provides the key process about design software.*

**Key words** S698P – SOC; SMP; *Cache*; *Data Consistency*; AMBA *bus*

由于性能上的需求,当前嵌入式系统已经不满足于使用单核处理器系统。许多设计者开始考虑多核处理。但多核处理系统在带来性能提升的同时,也面临着数据一致性的问题。如果一个多核、多任务系统无法保持数据一致性,结果将是灾难性的。由于嵌入式系统环境苛刻,有些还无法使用操作系统,而作为嵌入式操作系统,很难完全屏蔽硬件,将接口做到抽象和统一,工程师需要直接面对底层进行编程,这个问题就显得尤为突出。

本文将以 S698P – SOC 处理器为例,介绍在多核处理器系统中,处理器层面和软件层面如何保持数据一致性。

## 1 S698P – SOC 简介

S698P – SOC 是一种高性能四核处理器,采用

**收稿日期**: 2007-04-23

**作者简介**: **梁宝玉**(1974 – ),男,山东高密人,硕士,研究方向为高可靠 SOC 设计; **颜 军**(1962 – ),男,山东高密人,博士,研究方向为智能控制、模糊控制、高可靠嵌入式控制器、SOC 芯片的设计及产业化; **侯 雄**(1967 – ),男,湖北人,博士,研究员,研究方向为飞行器导航、制导与控制和信息技术; **宋征宇**(1969 – ),男,江苏人,研究员,研究方向为飞行器导航、制导与控制。

并行架构,每个处理器都可以独立运行。4 个处理器通过 AMBA 总线互联,并共享存贮器和 I/O 等外设。整个系统架构如图 1 所示。

S698P - SOC 的核心频率为 400MHz,外部总线频率最高可达 166MHz。为了解决高速运行的 CPU 和外部缓慢的存贮器之间的矛盾,处理器加入了 cache,常用的数据可以保存在 cache 中,可以极大地提高系统的运行速度。在多处理器系统中,cache 还能有效降低总线带宽占用,减少总线冲突,进一步提高系统的整体性能。S698P - SOC 采用本地 cache 的结构,每个处理器都有自己独立的数据 cache 和指令 cahce,使用 LRU 算法进行数据调度,采用透写策略(write-through)更新数据。在地址映射时,采用的是全相联映像法。

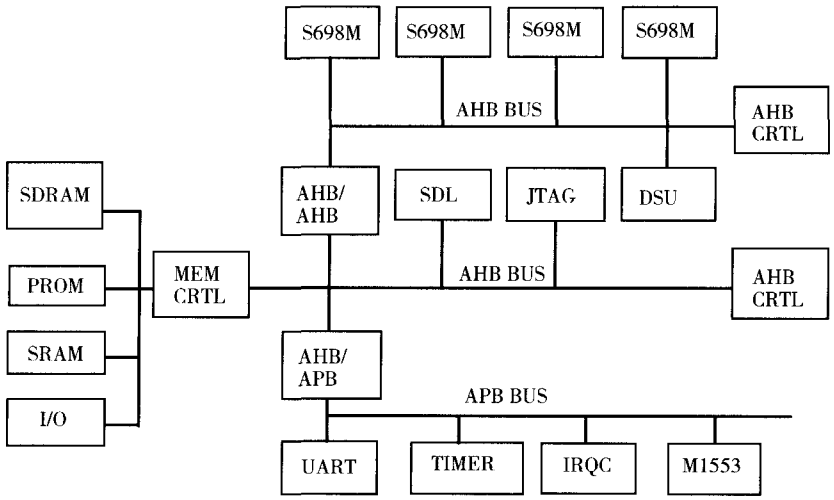


图 1　S698P - SOC 总体架构图

S698P - SOC 的 cache 与整数单元及片内总线之间的关系如图 2 所示。

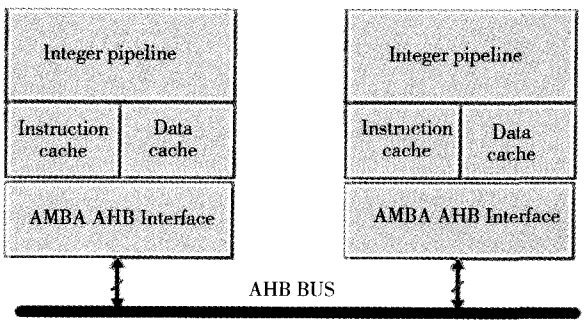


图 2　S698P - SOC IU 架构图

上图画出了 2 个 IU(Integer Unit)单元。IU 单元共有 7 级管线(pipeline),它通过指令 cache 请求指令,数据 cache 请求数据。cache 通过 AMBA AHB 接口和总线相连,所有的请求都由总线控制器统一处理。存贮器是属于 AHB 总线的一个设备。数据 cache 引入了 AHB 总线信号,对 AHB 总线上的读写操作进行监视。

## 2　数据不一致问题

cache 的存在和多个处理器共享存贮器及 I/O 空间,会带来数据一致性的问题。其具体表现在以下几个方面。

### 2.1　数据丢失

在下表中,$T1$,$T2$,$T3$,$T4$ 表示时间顺序(下同)

| CPU | $T1$ | $T2$ | $T3$ | $T4$ |
|---|---|---|---|---|
| 1 | | $X=40$ | $X=X-30$ | |
| 2 | $X=40$ | | | $X=X-20$ |

假设 CPU1 和 CPU2 都读取 $X(X=40)$,然后分别把 $X$ 减去 30 和 20。CPU1 在 $T3$ 把改后的 $X(X=10)$写入存贮器。随后,CPU2 在 $T4$ 把改后的 $X(X=20)$写入存贮器。于是对 CPU1 而言,它的修改在 $T4$ 处丢失了。

### 2.2　数据脏读

| CPU | $T1$ | $T2$ | $T3$ | $T4$ |
|---|---|---|---|---|
| 1 | $X=40$ | $X=X+30$ | | $X=X-30$ |
| 2 | | | $X=70$ | |

CPU1 在 $T2$ 把 $X$ 增加了 30，并更新了 cache，CPU2 在 $T3$ 读出 $X=70$，但 CPU1 在 $T4$ 时把 $X$ 减去了 30，数据仍维持 $X=40$，但 CPU2 已把数据 $X=70$ 取走。

2.3 不能重复读

| CPU | $T1$ | $T2$ | $T3$ | $T4$ | $T5$ | $T6$ |
|---|---|---|---|---|---|---|
| 1 | $X=40$ | | $Y=30$<br>$X+Y=70$ | | | $Z=30$<br>$X+Y+Z=100$ |
| 2 | | $X=40$ | | $X=X+20$ | 写入存贮器 | $X=X-20$ |

CPU1，CPU2 分别读取 $X=40$ 后，在 $T3$ 时 CPU1 取出 $Y=30$ 并计算 $X+Y=70$。在 $T4$ 时 CPU2 把 $X$ 增加 20，并于 $T5$ 把 $X=60$ 写入存贮器。在 $T6$ 时，CPU1 取出 $Z(Z=30)$ 并继续计算 $X+Y+Z=100$。但如果 CPU1 为了进行校验而把 $X,Y,Z$ 重读一次再进行计算，却出现 $X+Y+Z=120$！($X$ 已增加 20)。

## 3 S698P－SOC 中的数据一致性算法

3.1 现有的数据一致性算法

a. 软件法

硬件不提供数据一致性的方法，整个算法完全由编译器和操作系统决定，当需要进行数据一致性时，软件刷新缓存，并保证关键数据只有一个副本在进行操作。软件解决方案可以使硬件设计简单，面积减少。但是，软件通常采用比较保守的方法进行数据一致性操作，会带来不必要的开销。通常，这个过程也要比硬件慢。

b. 目录法

目录法是将内存中每个共享数据块设置一定的目录项，用于记录那些 cache 含有该数据块的拷贝。当某个处理机对私有 cache 进行数据更新操作时，系统根据 cache 目录的内容将所有其他存有相同内容的 cache 拷贝置为无效。一般采用 3 种形式：全映射目录、有限目录和链式目录。这 3 种策略只是在目录的组织形式和采用的数据结构不同，全映射目录因为需要映射的目录项很多，将占用大量的共享存储空间；有限目录虽然有所改进，但由于采用了固定的指针数，装入 cache 的数据受到了限制；由于链式数据结构固有的查找性能缺陷，致使链式目录在目录项很多时降低系统的性能。

c. 总线监听法

总线监听法是通过总线监听机制实现 cache 与内存之间的数据一致性，一般采用写无效(Write-Invalidate)和写更新(Write-Update)两种策略。写无效策略的基本思想是当私有 cache 的某个数据块更新时，同时将其他 cache 中含有该数据块的数据均置为无效。该策略中数据块的大小对 cache 的性能影响很大，如果数据块太大，当某个处理机正在更新一个数据块而其他多个处理机发生阵发式“读缺失”时，就有可能导致总线的流量大大增加，甚至出现争用总线的现象，影响系统的整体性能，cache 结构带来的好处将大打折扣；写更新策略的实现方法是当某个处理机在更新私有 cache 的同时，将更新后的数据块发送给所有相关的 cache，并用新的数据覆盖原来的数据，而不管这些 cache 是否会读取这些数据。这种不加选择的复制造成了多余的开销，并且，当数据块较大或频繁的数据更新时将占用很长的总线时间或导致总线通信拥挤，轻者导致系统性能降低，重者造成数据丢失。另外，由于此策略采用广播更新，可能有的处理机进程不再使用该数据，这样就产生了无效通信。

以上 3 种方法都有自己的优点和缺点。考虑到嵌入式系统一般存贮空间小，可能没有存放目录表的空间，而软件法性能较低，所以 S698P－SOC 采用了总线监听法。

3.2 S698P－SOC 的数据 cache 结构

数据 cache 由 2 部分构成。一部分用来存贮缓存的标志(TAG)等信息，另一部分用来保存缓存的数据。

存储缓存标志等信息的 RAM 称为 TAG RAM。每个 TAG 的结构定义如下：

| 31 | 10 | 9 | 8 | 7 | 0 |
|---|---|---|---|---|---|
| ATAG | LRR | LOCK | | VALID | |

各个区域的含义：

[31:11]：地址 Tag (ATAG) － 主存贮器的地址信息

[10]：LRR － 使用 LRR 算法更新数据。“0”表示不使用

[9:8]：LOCK － 当置位时，这条 cache line 被锁定

[7:0]：Valid (V) － 当置位时，表示当前 cache line 的数据无效

主存贮器地址的高 21 位保存在 ATAG 中，低

11 位则决定了数据在 cache 中的位置。一条 TAG 对应了一块 cache 区域,这块区域称为一条 cahce line。

为了提高效率,S698P - SOC 使用了“line fill”算法。即当请求一个数据而这个数据不在 cahce 中时,cache 控制器会一直请求数据,直到填满当前的 cache line。这个过程可以被别的数据请求打断。

### 3.3　S698P - SOC 采用的数据一致性算法

如果 CPU 需要数据,cache 首先检查数据是否已经在自己的缓存中,如果是(称为缓存命中),则直接返回数据。如果不是,则从主存中调入这个数据到自己的 cache 中,同时传输给 CPU。

S698P - SOC 中的数据 cache 采用透写策略(write - through)更新数据。当 CPU 把修改后的数据写入到 cache 中时,cache 在更新自己缓存的同时,也通过 AHB 总线把数据写到主存贮器中。

S698P - SOC 使用基于“写无效”的数据一致性策略。所有的数据 cache 都会监听地址和数据总线,当总线上存在一个写操作时,cache 控制器会检查这个地址是否命中自己的缓存,如果命中,则使自己相应的缓存区域无效。当 CPU 需要这个数据时,cache 就会从主存中读取,从而可以使自己的数据和主存中的数据保持一致。

如果 cache 使这个缓存区域无效时,CPU 正好需要这个数据,则 cache 会把当前数据总线上的数据返回给 CPU,并更新自己的缓存。

为了提高效率,cache 使用双口 RAM 保存地址 TAG,CPU 访问 cache 和数据侦听从 2 个不同的端口读取 TAG 数据,它们可以同时进行。

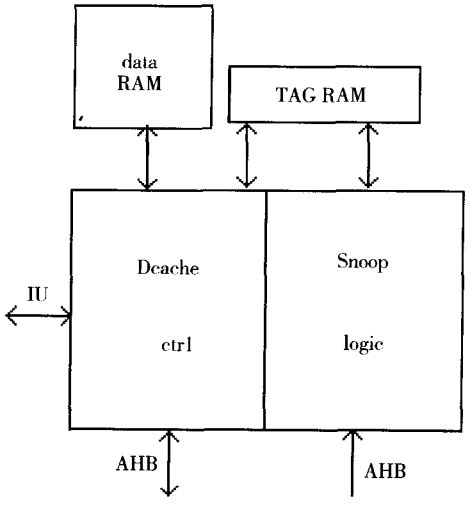


图 3　S698P - SOC 数据 cache 逻辑结构图

数据侦听是否有效是软件可控的。这个标志位在 cache 控制寄存器的 23 位。整个 cache 控制寄存器如下所示:

| 31 | 23 | 22 | 21 | | 16 | 15 | 14 | | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DS | FD | FI | | IB | IP | DP | | DF | IF | DCS | | ICS | |

各个区域的含义如下:

[23]: DS - 数据 cache 侦听允许,‘1’有效

[22]: FD - 刷新数据 cache

[21]: FI - 刷新指令 cache

[16]: IB - 指令猝发预取

[15]: IP - 指令 cache 刷新标志

[14]: DP - 指令 cache 刷新标志

[5]: DF - 中断时冻结数据 cache

[4]: IF - 中断时冻结指令 cache

[3:2]: DCS - 数据 cache 状态,共有 3 种:有效,冻结,无效

[1:0]: ICS - 指令 cache 状态,共有 3 种:有效,冻结,无效

当使能数据 cache 时,如果需要数据侦听功能,则需要同时使能 23(DS)位。

S698P - SOC 使用的“写无效”数据一致性算法,由于地址比较是在 cache 内部进行的,不占用总线带宽,cache 也不会立即更新这个数据,而是采取了需要才更新的策略,减少了无效的操作和等待时间,减少了和 CPU 请求的冲突,数据存贮器也无需使用昂贵的双口 RAM,不失为一种高效、经济的数据一致性解决方案。

## 4　硬件和软件设计

数据一致性操作是靠 CPU 和软件共同完成的。

### 4.1　硬件设计

在 S698P - SOC 中,每个 CPU 的 cache 控制器中都有一个 DS(Data cache Snoop)位,如果设置了这个位,则数据 cache 会进入数据侦听过程,直到再次禁止它。如果用户打算使用 S698P - SOC 中的 2 个或以上的核,并使能了数据 cache,同时有数据一致性的需求,则需要在每个 CPU 的启动代码中都加入允许这个标志位的代码,否则,没有设置这个位的 CPU 不会进行数据侦听工作。软件中允许 cache 并设置数据侦听标志的语句是:set 0x81000f, %g1。

### 4.2　软件设计

在系统启动完成后,用户在应用软件中如果需要操作共享的数据,需要遵循以下几条原则。

1) 先锁定,再读取

对于共享数据的操作,用户应该首先向系统请求锁定这些数据,然后进行操作。当操作完成后,更新数据并解锁。这样,即使是这些数据中间是在寄存器中操作,也可以保证最终结果是正确的。

S698P - SOC 中的数据一致性算法可以保证所有锁的状态在所有的 cache 和主存中都是一致的。

S698P - SOC 对数据加解锁可以只用 4 条指令,3 个时钟周期完成,示例代码如下:

```
get_sem: ! 加锁
set mpsem, %o1
set 0, %o0
retl
swapa [%o1] 1, %o0 ! 请求锁,上锁标志写入到主存贮器
ret_sem: ! 解锁
set 1, %o0
set mpsem, %o1
retl
st %o0, [%o1] ! 释放锁
```

以上加、解锁操作的关键指令都只有一条,不可被打断,并保证了锁的最终状态仅存在于 cache 和主存中,保证了一致性。软件无须屏蔽中断,无须做上下文保护,使用起来非常方便。

2)先缓冲,再操作

对于 I/O 数据,由于 I/O 数据一般都具有不可复现性,这次读的数据和下一次读的数据很难保证是一样的,软件需要维持一个 I/O 缓冲区,以保证当前步骤的所有操作都使用同一次的数据。当都完成这次操作后,才进行下一次 I/O 读写。

3) 使用存贮器变量

在 c 语言编程中,对于共享的变量,在声明时可以合理使用 volatile 关键字,使变量每次都从缓存或存贮器中读取,以免变量被缓冲在寄存器中,引起莫名其妙的问题,例如数据修改丢失等。需要注意的是,对于 volatile 关键字不可滥用,否则会导致软件效率低下。另外,对于特定的编译器,也可以通过设置编译参数,改变编译器的编译行为,生成正确的代码。

只要在软件和硬件设计时注意以上几点,就可以有效的解决数据一致性问题。

## 5 总 结

在多处理器系统中,数据一致性已经难以通过硬件或软件单独完成。S698P - SOC 并行多处理器中使用"透写"和"写无效"技术,实现了数据一致性算法,可以保持多个 CPU 之间 cahce 和主存贮器的数据一致。用户通过适当的软件设计,可以容易地实现多核、多任务的数据一致性。相关的技术在工程实践中已经得到验证,获得了良好的效果。

## 参 考 文 献


[1] Zhao Y, Hu C, Wang S, and Zhang S. An Extended OpenMP Targeting on the Hybrid Architecture of SMP-Cluster[M], Advances in Computer Science and Technology, 2006.

[2] Byoungro S, Anwar M. G, and Youfeng W. Optimizing Data Parallel Operations on Many-core Platforms[D]. Intel Corporation, 2004.

[3] Sabot G W. The Paralation Model: Architecture - independent Parallel Programming[M]. The MIT Press, 1989.

[4] 郑伟民. 汤忠. 计算机系统结构[M]. 清华大学出版社,2001.

[5] 杜贵然,周兴铭. Trace Cache 及 Trace 处理器技术[J]. 计算机工程与科学,2001,23(1):39 - 43.

[6] 唐志敏. 分布存储并行系统中的共享存储编程环境讲义[D]. 中科院计算机研究所,2003.

[7] 季振洲. 并行处理与体系结构讲义[M]. 哈尔滨工业大学,2005.

[8] 刘广忠. 基于外部共享 Cache 的多处理机 Cache 一致性协议[J]. 河北工程技术高等专科学校学报,2006,2:1 - 3,10.