avionicstech@avic.com

DOI:10.12175/j.issn.1006-141X.2022.04.08

基于机载多核弱序存储模型的共享内存驱动软件设计方法研究

徐　光[1]，曲国远[2]，吉沛琦[2]

（1. 海装驻上海地区第十军事代表室，上海 200233；
2. 中国航空无线电电子研究所，上海 200233）

[摘　要] 随着航空电子系统承载的应用日趋复杂，飞机对机载计算机的计算力和功耗比要求不断提升，这也推动了嵌入式多核处理器的加速应用和普及。多核处理器在航空电子设备的深入应用，随之而来的是运行其上的软件复杂度急剧上升，面向应用的航电系统设计面临挑战。多核处理器平台由于需要面对并行、指令乱序、资源共享冲突等问题，而目前国内大多数机载嵌入式软件和驱动仍然是基于单核处理器设计和实现的，影响最大的是在机载嵌入式实时操作系统环境下的驱动软件，因此需要充分考虑多核带来的各方面影响，尤其是需要兼顾共享内存等资源的使用冲突和实时高效要求。本文结合机载航电多核处理平台的特点，提出了一种基于机载多核弱序存储模型的共享内存驱动软件设计方法，并基于该方法设计了 FC 总线驱动和 MBI 总线驱动，项目应用结果表明，设计的驱动程序在多核处理器平台上数据传输正确，验证了方法的正确性和有效性。

[关键词] 嵌入式多核处理器；RTOS；驱动软件

[中图分类号] TP316　　[文献标识码] A　　[文章编号] 1006-141X(2022)04-0052-06

Research on Design Method of Shared Memory Driver Software Based on Airborne Multi-core Weak-Order Memory Model

XU Guang[1], QU Guo-yuan[2], JI Pei-qi[2]

(1. NO. 10 Military Representatives Office of Navy in Shanghai Area, Shanghai 200233, China;
2. China National Aeronautical Radio Electronics Research Institute, Shanghai 200233, China)

Abstaract: With the increasingly complex application of avionics systems, the requirements for computing power and power consumption ratio of aircraft to airborne computers are constantly improving, which also promotes the accelerated application and popularization of embedded multi-core processors. With the deep application of multi-core processors in avionics, the complexity of software running on them has risen sharply, and the design of application-oriented avionics systems is facing challenges. Because multi-core processor platform needs to face problems such as parallelism, instruction disorder, resource sharing conflict, and so on, most of the domestic airborne embedded software and drivers are still designed and implemented based on a single core processor, and the driver software in

收稿日期：2022-11-15

引用格式：徐光，曲国远，吉沛琦.基于机载多核弱序存储模型的共享内存驱动软件设计方法研究[J]. 航空电子技术,2022,53(4):52-57.

the environment of airborne embedded real-time operating system has the greatest impact, so it is necessary to fully consider the impact of multi-core in all aspects. In particular, it is necessary to take into account the use conflict of shared memory and other resources and the requirements for real-time efficiency. Combining the characteristics of airborne avionics multi-core processing platform, a shared memory driver software design method is proposed based on the airborne multi-core weak order storage model, and a FC bus driver and an MBI bus driver are designed based on this method. The project application results show that the designed driver is correct in data transmission on the multi-core processor platform, which verifies the correctness and effectiveness of the method.

Key words: embedded multi-core processor; RTOS; driver software

随着我国航空电子机载设备的高速发展，在航电的核心设备中已经开始大量应用嵌入式多核处理器，较为典型的如 PowerPC 架构下 T1040 系列、T2080 系列，ARM 架构下的 IMX、LS1026、Xilinx MPSOC、国产飞腾（双核、四核、八核）处理器、海思处理器以及 MIPS 架构的龙芯等等。随之而来的是运行其上的设计难度在显著提升 [1]，从而出现了基于多核的安全性分析方面的标准和准则 [4]。多核处理器平台由于需要面对并行、指令乱序、资源共享冲突等问题，而目前国内大多数机载嵌入式软件和驱动仍然基于单核处理器设计和实现，影响最大的是在机载嵌入式实时操作系统环境下的驱动软件，因此需要充分考虑多核带来的各方面影响，尤其是需要兼顾共享内存等资源的使用冲突和实时高效要求。

在机载嵌入式平台中无论是底层软件还是应用软件，共享内存是较为常见的一种软件模块间、软件硬件间交互方式。而在多核处理器平台中，共享内存数据交互和访问较之单核处理器有所不同。这与多核处理器架构本身的内存模型息息相关。本文针对嵌入式实时操作系统驱动开发，以 ARMv8 内核为例，研究 ARM 的内存模型，并据此阐述基于嵌入式实时操作系统（RTOS：Real Time Operating System）下的共享内存驱动开发方法。

1　ARMv8 内存模型及影响分析

ARMv8 内核采用的是弱序（Weakly Ordered）内存模型，这种模型允许内存的使用者（写）和观察者（读）以不同顺序完成对内存操作。基本原理是 [2]：

- 提供一种与高级编程语言（如 C、Java）对应的弱 order 内存模型；
- 大多数内存中避免涉及使用 multi-copy atomicity；
- 提供指令和内存屏障，以避免 multi-copy atomicity 引起的缺陷；
- 在指令执行顺序构建中使用地址、数据和控制指令的依赖关系，从而避免程序设计者或者编译器在常见场景中过多使用屏障和其他相关特殊指令。

在 ARMv8 架构的内存模型下，对内存的真实访问顺序和程序的 load store 操作顺序并不完全一致，这种顺序的不一致是处理器在执行访问指令时动态完成和优化的，并不是编译器对软件代码的优化，这会导致操作系统、驱动等底层软件在执行数据内容前后相关的代码时，产生数据内容不确定的错误和问题。下面是一些需要考虑使用内存屏障指令的典型场景。

- 在 CPU 的多个不同内核之间共享数据；
- 执行和外设相关的操作，例如 DMA 操作修改内存管理的策略，例如上下文切换、请求缺页以及修改页表等；
- 修改存储指令的内存区域，例如自修改代码的场景。

2　内存屏障指令分析

ARMv8 指令集提供了屏障指令，用于在特定点强制访问顺序和访问完成。其中包括 DMB、DSB 和 ISB 三个较为常用的指令。

数据存储屏障（DMB：Data Memory Barrier）指令：仅当所有在它前面的存储器访问操作都执行完毕后，才提交（commit）在它后面的访问指令。DMB 指令保证的是 DMB 指令之前的所有内存访问指令和 DMB 指令之后的所有内存访问指令的执行顺序。

数据同步屏障（DSB：Data Synchronization Barrier）指令：比 DMB 指令要严格一些，仅当所有在它前面的内存访问指令都执行完毕后，才会执行在它后面的指令，即任何指令都要等待 DSB 指令前面的内存访问指令完成。位于此指令前的所有缓存（如分支预测和 TLB 维护）操作需要全部完成。

指令同步屏障（ISB：Instruction Synchronization Barrier）指令：确保所有在 ISB 指令之后的指令都从指令高速缓存或内存中重新预取。

2.1 DMB 指令分析

DMB 指令影响的是数据访问序列。DMB 指令并不能确保 ARMv8 的所有指令必须在某个时刻一定执行完，它只能限制 DMB 指令前后的内存访问指令的执行顺序。数据访问指令包括普通的加载操作 load 和存储操作 store，也包括数据高速缓存 data cache 维护指令。

DMB 指令可用来确保 DMB 指令之前的数据访问可以被 DMB 后面的数据访问指令观察到。所谓“观察到”指的是先执行完 A 指令，然后执行 B 指令，于是 B 指令可以观察到 A 指令的执行结果。如果 B 指令先于 A 指令执行，那么 B 指令没有办法观察到 A 指令的执行结果。

所以，DMB 指令强调的是内存屏障前后数据访问指令的访问次序。这里有两个要点：一个是数据访问指令，另一个是保证访问的次序。

举例，处理器执行如下指令：

```
1 ldr x0, [x1]
2 str x2, [x3]
```

LDR 指令读取 x1 地址的值，STR 指令把 x2 的值写入 x3 地址。如果这两条指令没有数据依赖或者地址依赖关系，那么处理器可以先执行 STR 指令或者先执行 LDR 指令，从最终结果没有任何区别。这种情况下，并不需要使用 DMB 指令。

但是，在如下场景下，会产生数据依赖。

```
1 ldr xm, [xn]
2 str xm, [xy]
```

从 xn 地址读取内容到 xm 地址中，然后把 xm 地址中的值写入 xy 地址中，那么 xm 为这两条指令就有数据依赖。

2.2 DSB 指令分析

DSB 指令相比 DMB 指令更加严格。在 DSB 后面的任何指令需要满足以下 2 个条件才能开始执行。

一是 DSB 指令前面的所有数据访问指令（内存访问指令）必须执行完；

二是 DSB 指令前面的高速缓存、分支预测、TLB 等维护指令也必须执行完。

需要注意的是，上述两个条件满足之后才能执行 DSB 指令后面的指令。DSB 指令后面的指令指的是任意指令。与 DMB 指令相比，DSB 指令规定了 DSB 指令在什么条件下才能执行，而 DMB 指令仅仅约束屏障前后的数据访问指令的执行次序。

如下所示，CPU 执行 3 条指令。ADD 指令必须要等待 DSB 指令执行完才能开始执行，它不能重排到 LDR 指令前面。如果把 DSB 指令换成 DMB 指令，那么 ADD 指令可以重排到 LDR 指令前面。

```
1 ldr x0, [x1]
2 dsb ish
3 add x2, x3, x4
```

2.3 ISB 指令分析

ISB 指令会冲刷流水线，然后从指令高速缓存或者内存中重新预取指令。可在更改上下文操作中使用，包括高速缓存、TLB、分支预测等维护操作以及改变系统控制寄存器等操作。使用 ISB 确保在 ISB 之前执行的上下文更改操作的效果对在 ISB 指令之后获取的指令是可见的。更改上下文操作的效果仅仅在上下文同步事件（context synchronization event）之后能看到。上下文同步事件包括：发生一个异常（exception）；从一个异常返回；执行了 ISB 指令。

举例如下，处理器执行如下代码来打开 FPU 功能。把 cpacr_el1 的 Bit[21:20] 设置为 0x3，即可以打开浮点运算单元。但在打开之后，马上执行一条 FADD 指令，可能会导致 CPU 异常。因为 FADD 这条指令可能已经在流水线里，并且有可能会提前执行，即打开浮点运算单元之前就提前执行了，所以出现错误了。

解决办法就是插入一条 ISB 指令。这里的 ISB 指令是为了保证前面打开 FPU 的设置已经完成，才从指令高速缓存里预取 FADD 这条指令。

```
//打开FPU
mrs  x1, cpacr_el1
orr x1, x1 #(0x3 << 20)
msr cpacr_el1, x1

isb

fadd   s0, s1, s2
```

综上可以总结一条原则，当程序执行过程中有很强的时间顺序约束，也就是说前一条代码与后一条代码在指令上或者数据上。

3　基于共享内存的 RTOS 驱动设计

在机载嵌入式处理器平台上，共享内存是驱动软件中软件模块间、软件与硬件交互的一种常见方式，该方式设计简单灵活、平台耦合性低、执行效率高 [3]。本文以机载高速总线驱动和多线程共享内存数据交互 2 个场景为例，阐述设计方法。

3.1　机载高速总线驱动共享内存设计

在机载嵌入式平台上，常规的机载高速总线通信方式是处理器 PCIe 控制器外部连接机载高速总线子卡，处理器上运行的驱动再内存中开辟专用的共享内存空间，将需要发送的数据或者读取的数据通过 PCEe 与机载高速总线子卡完成交互。

机载高速总线驱动的数据发送流程一般如下所示，如图 1 所示。

- 处理器将需要发送的数据写入到内存中已预先初始化过的发送缓冲区（该区域一般已禁 Cache）；
- 处理器控制机载高速总线子卡启动 DMA 传输；
- 机载高速总线子卡从内存中搬移数据帧并发送。

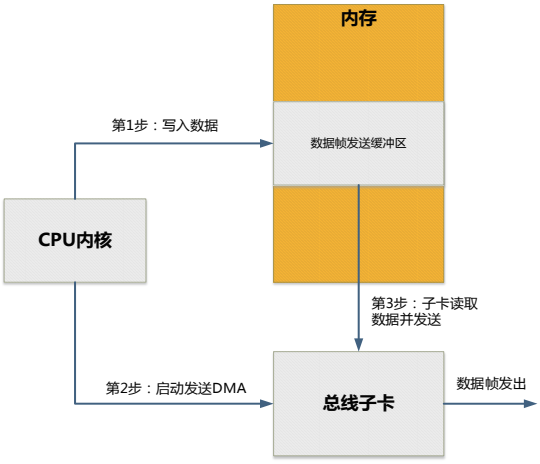

图 1　机载高速总线数据发送

机载高速总线驱动的数据接收流程一般如下所示，如图 2 所示：

- 机载高速总线子卡将接收到的数据帧写入内存中实现划分的接收缓冲区（该区域一般禁止 Cache）；
- 机载高速总线子卡触发中断信号通知 CPU 内核；
- CPU 处理接收中断，从内存中读取数据并提交给应用软件。

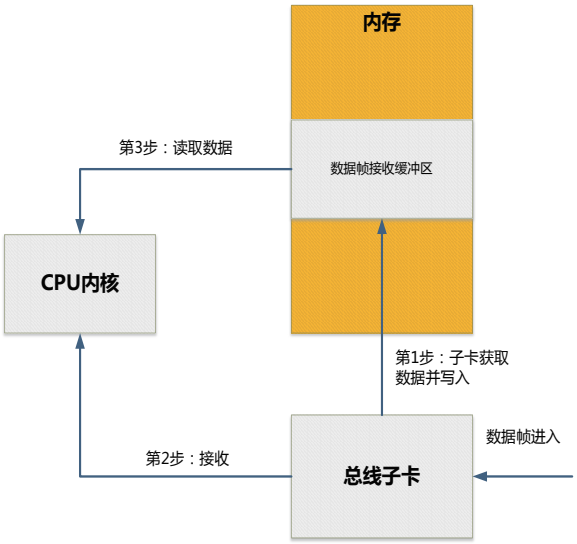

图 2　机载高速总线数据接收

机载高速总线驱动在单核操作系统中，考虑了缓存一致性问题后，不存在问题。但是，如果在 ARMv8 多核处理器平台上，那么发送或者接收到的数据可能存在错误的问题。这个问题的原因，是由于前章节中阐述的处理器内存模型导致的。

上述问题可以通过增加内存屏障指令来解决。具体解决方法如下：

- 在发送流程中，CPU 内核将数据写入发送缓冲区后，使用内存屏障写操作 DSB/DMB，确保前序写入操作指令全部执行完成，然后启动 DMA 操作。
- 在接收流程中，拷贝接收缓存区数据并清除读取的标志后，执行内存屏障写操作 DSB/DMB，确保读的数据和操作的标识全部执行完成，然后再使能下次读取。

3.2　多线程共享内存设计

机载嵌入式平台中，底层驱动或者应用会利用共享内存完成 2 个线程之间的数据交互。一般流程如图 3 所示。

线程 1 向共享内存区域写入数据并更新标识，线程 2 读取标识并在满足条件时读取数据。

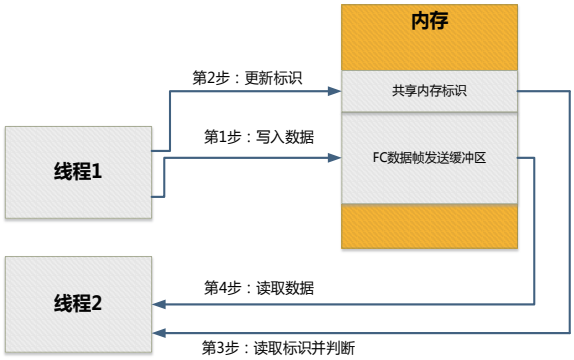

图 3 多线程共享内存交互

上述多线程共享内存机制如果在 ARMv8 多核处理器平台上运行，可能存在数据或者标识读取错误的问题。

上述问题可通过增加内存屏障指令来解决。具体解决方法如下：

- 在线程 1 执行写入操作后，使用内存屏障写操作 DSB/DMB，确保前序写入操作指令全部执行完成。
- 在线程 2 执行读取操作前，执行内存屏障写操作 DSB/DMB，确保前序的数据和操作的标识全部写操作已执行完成。

4 应用验证

FC 总线是典型的一种高速总线，其架构基本上都是处理器模块通过 PCIe 与 FC 子卡连接，在处理器模块上提供驱动，实现与 FC 子卡的交互，从而达到与外部节点的通信。

为了提升效率，FC 子卡会如图 1 和图 2 所示，通过 DMA 完成数据的搬移。因此，当在多核中使用时，需要通过内存屏障指令确保处理器操作内存空间与 FC 子卡操作内存的顺序不会被处理器内核打乱。

MBI 总线是机载设备应用成熟、广泛的一种网络。在以往的架构下，主要是处理器模块与 MBI 子卡通过 PCIe 连接，通过内存双口实现处理器的 MBI 网络通信。

而在多核平台下，MBI 子卡已经被协议芯片所替换，处理器平台通过 SPI 高速接口与 MBI 协议芯片连接。MBI 子卡的功能迁移到了多核处理器上，与应用运行在同一个处理器平台上。在这种情况下，可以在多核处理器平台构建一个线程并在独立的处理器内核上负责实现 MBI 协议芯片控制，其他核的应用通过共享内存实现 MBI 数据的收发和管理。如图 4 所示。在这种情况下，也需要内存屏障指令来确保上述 2 个线程之间数据交互的正确性。

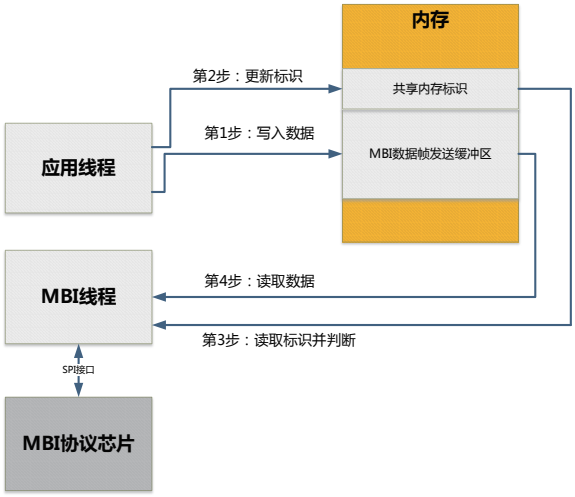

图 4 MBI 多线程共享内存交互

上述 FC 总线驱动和 MBI 总线驱动，在基于飞腾四核处理器平台上进行了项目应用和验证具体验证。验证方法如下所示：

（1）将 FC/MBI 外部总线接收和发送回环，以实现自发自首；

（2）然后在内存中申请长度为2048 字节的数组，填充有规律的数据作为发送数据；

（3）申请长度 2048 字节的数组并清零，作为接收缓冲区；

（4）调用 FC 驱动发送函数发送数据，并调用接收函数接收；

（5）对比接收数据与发送数据是否相同，并计算执行时间；

（6）重复执行 （2）~（4）步 1000 次。

采用上述测试方法验证增加内存屏障设计方法前后的正确性和性能，结果如表 1 所示。从实际验证数据可以看出，通过本论文提出的设计方法虽然增加了一定的时间开销（FC 驱动平均执行时间增加了 3.2μs，MBI 驱动平均执行时间增加了 2.6μs），但是能够有效确保数据传递的正确性，并且未对整体性能造成影响。

表 1 验证结果

驱动	测试项	无内存屏障	增加内存屏障
FC 总线驱动	测试正确次数	1	100
	平均执行时间	10.1μs	13.3μs
	最大执行时间	12.3μs	14.6μs
	最小执行时间	9.1μs	12.4μs
MBI 总线驱动	测试正确次数	0	100
	平均执行时间	17μs	19.6μs
	最大执行时间	19.7μs	21.1μs
	最小执行时间	16.3μs	18.7μs

5　结论

本文针对机载嵌入式多核处理器平台，分析了 ARMv8 典型的内存模型以及内存屏障的概念和作用。以机载嵌入式平台常见的共享内存机制为例，针对机载高速总线驱动和多线程通信 2 种场景，提出了一种基于机载多核弱序存储模型的共享内存驱动软件设计方法，并基于该方法设计了 FC 总线驱动和 MBI 总线驱动，项目应用结果表明，设计的驱动程序在多核处理器平台上数据传输正常，验证了方法的正确性和有效性。提出的驱动软件设计方法，可为机载多核计算平台的共享内存底层软件设计提供参考。

参 考 文 献

[1]周霆 , 朱晓宁 , 任晓瑞 . 一种多核分区操作系统的绑定组调度方法 [J]. 航空计算技术 , 2022-10: 85-88.

[2]Arm Limited.Arm® Architecture reference manual armv8, for armv8-a architecture profile[M]. 2019.7: 2268-2276.

[3] 朱素霞 , 季振洲 , 李东 . 面向多核处理器的内存竞争记录研究综述 [J]. 智能计算机与应用 , 2013: 53-59.

[4]Certification Authorities Software Team. Certification authorities software team position paper CAST-32A: multicore processors[M]. 2016-11.