第35卷 第22期 计 算 机 工 程 2009年11月
Vol.35 No.22 Computer Engineering November 2009
·开发研究与设计技术· 文章编号：1000—3428(2009)22—0253—03 文献标识码：A 中图分类号：TP301

SOCDMMU中核心部件的软件实现方法

梁东凯，张发存
(西安理工大学计算机科学与工程学院，西安 710048)

摘 要：针对传统系统级芯片动态内存管理单元(SOCDMMU)，提出用软件方法实现SOCDMMU中的核心部件。该方法分析了核心硬部件的功能，以软件方程的形式进行算法抽象，并在改进后实现算法。测试结果表明，经过该方法实现的SOCDMMU流片能进行算法的更新，且随着片上内存资源的增大，管理效率能同步提高。

关键词：系统级芯片动态内存管理单元；多处理器；内存管理

Software Implementation Method of Core Part in SOCDMMU

LIANG Dong-kai, ZHANG Fa-cun
(College of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048)

【Abstract】 Aiming at traditional System On Chip Dynamic Memory Manage Unit(SOCDMMU), this paper proposes realizing the core parts of SOCDMMU with the software method. This method analyzes the function of core parts, abstracts the algorithm in the form of software equation, and realizes the algorithm after improvement. Test result indicates the SOCDMMU realized by this method can carry on the algorithm renewal and as the memory resources of the piece increase, management efficiency can be enhance synchronously.

【Key words】 System On Chip Dynamic Memory Manage Unit(SOCDMMU); multiprocessor; memory management

1 概述

片上全局内存资源的管理一直是多处理器设计方面的关键环节，其管理效率低下会导致内存的访问、存储效率无法与高速的处理器相匹配[1-2]。传统的管理方式有静态和动态，但有其各自的不足和应用局限，如大量内存碎片和高最坏情况执行时间等。系统级芯片动态内存管理单元(System on Chip Dynamic Memory Manage Unit, SOCDMMU)位于各PE与全局内存之间，采用全新的管理思想——二级管理体系对全局和PE本地内存资源进行管理，与未采用SOCDMMU相比，其平均管理效率提高440%[3]，应用前景广阔。

体系结构中控制部分的最终实现有硬连线和微程序2种方式[4]，SOCDMMU采用硬件电路实现。SOCDMMU中的核心部分，如实现片上全局内存块资源分配的部件——分配单元，其算法复杂，工作时需多次数据通路执行，硬件实现代价较高，且随着门数的增加会导致信号的传播延迟。针对该问题，采用软件方式来实现，节约了片上面积，也方便算法的改进。

2 SOCDMMU简述

SOCDMMU是由文献[3]提出的嵌入式实时多处理器系统级芯片动态内存管理单元，采用硬件方式实现，该单元位于PE和片上全局内存之间，负责片上全局内存单元在各PE之间的分配和回收，并以快速可靠的方式来执行。SOCDMMU将全局内存单元分成一个个固定大小的块G_block，一个或多个G_block又组成一页，每页由唯一的页ID标识。每一个PE通过内存总线连接到SOCDMMU上，使PE对G_block的请求必须通过SOCDMMU，SOCDMMU能够将PE地址(虚拟地址)转换为物理地址，无论对于连续的还是非连续的G_block，PE本身也可以将分配好的G_block映射到PE地址空间范围内的任何位置。SOCDMMU被映射到每一个PE的I/O空间中的某一位置，可执行3类指令：G_allocate命令(G_alloc_ex, G_alloc_ro, G_alloc_rw)，G_deallocate命令和转移指令。SOCDMMU允许PE地址空间压缩，转移命令用来在PE地址空间内将已经分配好的G_block向另一个位置实施重映射。

2.1 指令及其执行周期

PE通过已映射到自己空间内的SOCDMMU地址或I/O端口向SOCDMMU发送命令(向端口或向映射地址写数据)和接收命令执行状态(从端口或映射地址读数据)。表1为SOCDMMU的指令及执行时需要的指令周期。

表1 指令及执行周期

命令	周期数
G_alloc_ex	4
G_alloc_rw	4
G_alloc_ro	4
G_delloc	4
最坏情况执行时间	4*x*(依据PE数)

2.2 SOCDMMU体系结构

图1描述了SOCDMMU的体系结构。每个PE都有自己的命令和状态寄存器，当PE拥有的本地内存资源无法满足其需求时，就要通过SOCDMMU申请全局内存资源。PE首先将命令写入对应的命令寄存器，发送到请求处理单元，当请求处理单元同时接收到多个请求时，根据其自身的优先算法来决定哪个PE的请求先被SOCDMMU执行。SOCDMMU一次可以处理一个请求，收到请求处理单元发来的请求后即

作者简介：梁东凯(1985－)，男，硕士，主研方向：嵌入式计算机系统结构；张发存，副教授、博士

收稿日期：2009-05-29 **E-mail：**liangdongkai1985@yahoo.com.cn

开始工作，完成 G_block 的分配或释放，然后将该 G_block 地址通过 PE 总线交给 PE，最后更新部分状态寄存器，保存分配或回收信息到特定硬件单元。

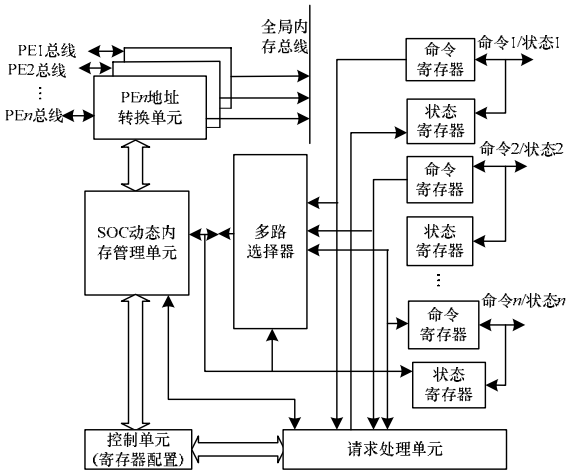

图 1　SOCDMMU 体系结构框图

2.3　二级管理体系

二级存储管理体系是在 SOCDMMU 基础上，针对片上全局内存以及各 PE 拥有本地 RAM 的情况下，为了提高管理效率而引出的管理体系[5]。其中，对片上全局内存块 G_block 在各 PE 之间的分配属于第 2 级；而每个 PE 对运行在自身的进程、线程之间分配本地内存单元属于第 1 级。在典型情况下，当一个进程需要一块内存区时，该进程向实时操作系统(Real-Time Operating System, RTOS)发出请求，如果当前 PE 有足够的、尚未使用的内存空间时，那么 RTOS 将立即执行简单分配，该情况属于第 1 级，如果 PE 没有足够的、空闲的内存块，那么将向 SOCDMMU 请求更多的资源，该情况属于第 2 级的工作。

3　SOCDMMU 的硬件组成

3.1　SOCDMMU 硬件原理

SOCDMMU 硬件结构如图 2 所示。

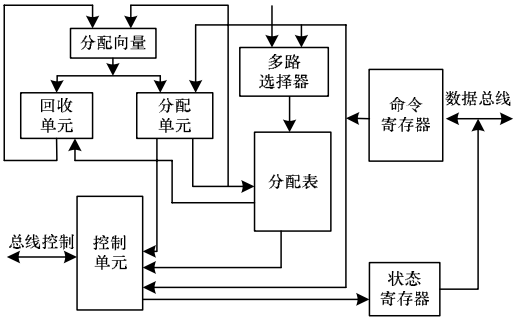

图 2　SOCDMMU 硬件结构

SOCDMMU 由分配向量、分配单元、回收单元、分配表、命令寄存器和状态寄存器等组成，各个部分主要功能如下：

(1)分配向量是一个总位数同 G_block 数相等的位向量，位值“ 0 ”表示对应的 G_block 尚未被分配，“ 1 ”表示对应的 G_block 已经被分配，当接收到新的分配请求时，该 G_block 不能被使用。

(2)分配表用来存储已分配的 G_block 信息，分配表中每项对应一个特定的 G_block，包含以下信息：分配模式(ex, ro, rw)得到该 G_block 的 PE 地址和该 G_block 所在的页 ID。

(3)分配单元负责将空闲 G_block 分配给 PE，即完成 G_allocate 操作，其算法通过硬件逻辑电路实现，输入为分配向量 *in*[*n*-1:0]和由控制单元产生的需求空间大小，输出 *out*[*n*-1:0]用来更新分配向量、分配表，以及将分配信息反馈给控制单元。

(4)回收单元完成 G_block 的回收，即完成 G_dealloc 操作。通过分配向量得到需要回收的 G_block，然后根据分配表来查询所在页，回收完毕后，更新分配向量和分配表，即将对应分配向量位置 0 并且在分配表中删除对应项。

3.2　分配单元

SOCDMMU 主要完成片上全局内存块的管理，即分配和回收。回收单元相对简单，无需复杂算法和电路逻辑[2]。分配单元完成 G_allocate(G_alloc_ex, G_alloc_ro, G_alloc_rw)操作，是整个 SOCDMMU 最核心的部分。图 3 为分配单元的门级电路逻辑关系。首先对输入的分配向量进行“ 非 ”操作，然后通过一个全减器(FS)完成 $o=x-y$，其中，x 为 *size*(每经过一个 FS 后 *size* 值减 1)；y 为经过“ 非 ”操作的 *in* 值。当全减器的输出 $b_{out}\geq 0$ 且对应的 *in*=0 时，将对应的输出 *out* 置 1，当某一个全减器的输出 $b_{out}<0$ 时，表示已经分配完毕，将从当前位开始向前的所有 *out* 值都置 0。分配单元的输出对分配向量更新时由逻辑“ 或 ”完成，即 *in*’[*n*-1:0]=*in*[*n*-1:0]∪*out*[*n*-1:0]，其中，*in*’[*n*-1:0]为更新后的分配向量。

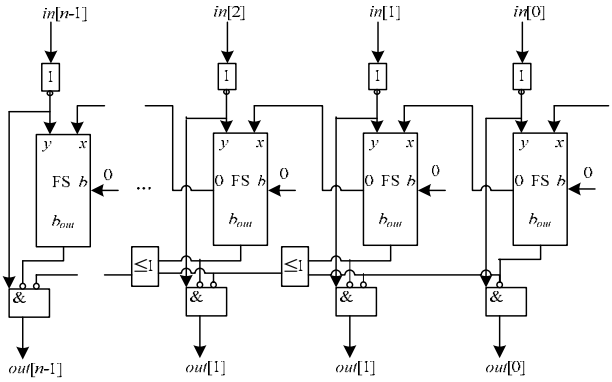

图 3　分配单元的电路逻辑关系

由于硬件实施占用较多片上面积(较多门数)以及传播延迟，因此，在该硬件算法下，其效率难以随着 *n* 数目的增加同步增加。

4　软件实现

4.1　算法分析

通过对分配单元逻辑关系的深入分析，本文对整个算法做如下抽象：

$$\begin{cases} out[x]=\begin{cases}1,(size-(!in[x]))\geq 0,\ in[x]=0\\ 0,(size-(!in[x])<0\end{cases}\\ size=size-1\end{cases}$$

可见，算法并不复杂，由于上述算法不能保证随着 *n* 数目的增大效率的同步提高，因此不妨将 *n* 位输入 *in*[*n*-1,0]分成 *k* 段，每段有 *n*/*k*=*m* 位，原有算法做如下更改：

(1)统计每段 *m* 位中值为 0 的个数 *zc*。

(2)当 *size*＞0 时(表明尚未分配完毕)，进行到第 *x* 段时(0≤x≤n-1)要进行如下处理：

当 *size*-*zc*>0 时，表明所有 *x* 段空闲空间分配完毕但仍不能满足需求，此时将 *x* 段每位 *in* 值取反作为 *x* 段的输出。然后进入(*x*+1)段处理，此时需分配的空间已经减少为 *size*=*size*-*zc*。

当 *size-zc*≤0 时，说明 *x* 段中空闲空间可以满足 *size* 需求，此时在 *size* 减少为 0 前，每遇到一个 *in* 值为 0，将其取反作为输出，每做一次，需分配的空间 *size* 减少 1。

(3)所有段已经处理完但 *size* 尚未减少到 0，说明需求远大于当前内存能提供的空间，此时应当作为例外处理。

(4)*size* 减少到 0 但 *x* 段尚未处理完，此时将当前位开始到 *in*[*n*-1]的所有位全部置 0。

(5)经过以上情况处理后，将 *out*[*n*-1,0]作为输出返回。

本文对修改后的算法做如下抽象总结：

$$\begin{cases} zc = zc+1;\ \ in[y]=0,\ x\times m \le y \le ((x+1)\times m-1) \\ out[i]=\begin{cases} !in[i];\ \ size-zc>0 \\ 1;\ \ (size-zc\le 0,\ in[i]=0, \\ \quad x\times m\le i\le((x+1)\times m-1),\ size>0) \\ 0;\ \ size\le 0 \end{cases} \\ size = size-zc;\ \ out[i]=!in[i] \\ size = size-1;\ \ out[i]=1 \end{cases}$$

4.2 算法实现

上文已抽象出详细的算法，因此，可以容易地进行软件实现，为了保证程序的执行效率，将 *k* 定为一个常量。图 4 为软件实现的流程。

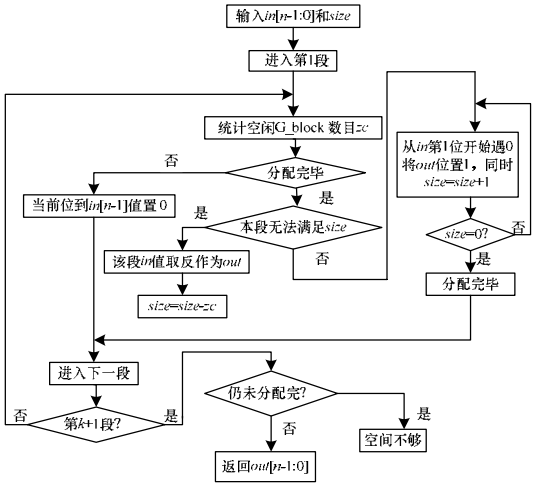

图 4　软件流程

5　测试及结果分析

测试模型由 4 个 ARM9TDMI 核组成，每个核包含 64 KB 的 L1 级指令和数据 Cache。所有 PE 共享一个全局总线，总线的另一端是 16 MB 的 SRAM 作为全局共享内存，假定指令从全局内存中取第 1 个字需要花 5 个周期。测试过程分 3 个阶段：(1)为未使用 SOCDMMU 的情况下，测试采用 glibc 库中 malloc()对 G_block 分配的执行时间；(2)使用 SOCDMMU，对 G_allocate 指令进行测试；(3)将 SOCDMMU 中的分配单元部分进行软件替换后进行相同测试。测试结果如表 2 所示。

表 2　测试后的结果对比　　周期

	平均情况执行时间	最坏情况执行时间
malloc()	215	1 700
纯硬件实现时的 G_allocate	29	212
部分软件实现时的 G_allocate	33	237

采用纯硬件情况和部分软件情况下，其加速比分别为未使用 SOCDMMU 时的 7.4 倍和 6.5 倍左右。可见，分配单元采用软件替换后，其效率略低于纯硬件电路情况，但仍远高于未使用 SOCDMMU 的情况。

6　结束语

为了方便地对算法进行更新和增加新的特性，在较小影响性能的情况下，通过软件的方式来实现 SOCDMMU 中的核心部分可以达到此目的。本文通过分析 SOCDMMU 功能部件和整个模块中分配单元的电路逻辑，抽象出其算法并对此算法进行一定改进，并用软件方式实现该部分的功能。经过仿真测试的数据表明，其效率稍低于纯粹的电路实现，但远高于不使用 SOCDMMU 的情况，同时，软件方式也节省了采用硬件分配单元时占用的片上面积。

此外，回收单元与分配表也可以使用软件来取代，但它们本身逻辑简单，在数据通路中可一次执行完毕，且占用少量时钟周期，因此，仍采用硬件方式为宜。

参考文献

[1] Rowen C. Engineering the Complex SOC Fast, Flexible Design with Configurable Processors[M]. New Jersey, USA: Prentice Hall, 2004.

[2] Wolf W. High-performance Embeded Computing Architecures, Applications and Methodologies[M]. San Fransisco, USA: Morgan Kaufmann, 2007.

[3] Shalan M, Mooney V J. Hardware Support for Real-time Embeded Multiprocessor System-on-a-chip Memory Management[J]. Hardware/Software Codesign, 2002, 17(3): 79-84.

[4] Patterson D A, Hennessy J L. Computer Organization and Design the Hardware/Software Interface[M]. 3rd ed. San Fransisco, USA: Morgan Kaufmann, 2007.

[5] Mohamed A S. Dynamic Memory Management for Embedded Real-time Multiprocessor System-on-a-chip[D]. Atlanta, Georgia, USA: Georgia Institute of Technolog, 2003.

编辑　陆燕菲

(上接第 252 页)

4　结束语

本文采用模糊聚类方法实现了对彩色图像中颜色区域的提取，实验结果证明了该方法的有效性。在以后的工作中，将依据图像的颜色、灰度和形状等多种信息进行融合处理，以实现更多的应用及更高的精度。

参考文献

[1] 林开颜, 吴军辉, 徐立鸿. 彩色图像分割方法综述[J]. 中国图象图形学报, 2005, 10(1): 1-10.

[2] Barbeau S, Labrador M, Winters P, et al. Location API 2.0 for J2ME-A New Standard in Location for Java-enabled Mobile Phones[J]. Computer Communications, 2008, 31(6): 1091-1103.

[3] Busin L, Vandenbroucke N, Macaire L. Color Spaces and Image Segmentation[J]. Advances in Imaging and Electron Physics, 2008, 151(2): 65-168.

[4] Zadeh L A. Similarity Relations and Fuzzy Orderings[J]. Information Sciences, 1971, 3(2): 177-200.

编辑　陆燕菲