

文章编号:1001-9081(2009)08-2136-03

## 基于 PXA3xx 处理器的 NAND 闪存 DMA 方案

史 斌<sup>1</sup>, 丁志刚<sup>2,3</sup>, 张伟宏<sup>2,3</sup>

(1. 上海市计算技术研究所, 上海 200040; 2. 上海计算机软件技术开发中心, 上海 201112;

3. 上海嵌入式系统应用工程技术研究中心, 上海 201112)

(sehrich\_ultra@hotmail.com)

**摘 要:**针对 PXA3xx 处理器的特性, 提出一种 DMA 控制器系统架构方案, 并在此架构上结合嵌入式 Linux 操作系统实现 NAND Flash 的底层驱动程序。重点设计了基于 JFFS2 文件系统的读操作测试方案, 测试结果表明, DMA 方式能够有效降低 CPU 处理负载, 缓解 NAND Flash 性能瓶颈。

**关键词:**PXA3xx 处理器; NAND; 直接存储器存取

**中图分类号:** TP316 **文献标志码:** A

## DMA method for NAND flash based on PXA3xx processor

SHI Bin<sup>1</sup>, DING Zhi-gang<sup>2,3</sup>, ZHANG Wei-hong<sup>2,3</sup>

(1. Shanghai Institute of Computing Technology, Shanghai 200040, China;

2. Shanghai Development Center of Computer Software Technology, Shanghai 201112, China;

3. Shanghai Engineering Technology Research Center of Embedded System Application, Shanghai 201112, China)

**Abstract:** According to the characteristics of PXA3xx processor, a solution of DMA controller system architecture was put forward and the low level NAND flash Direct Memory Access (DMA) driver under Linux was realized. Test method was designed in JFFS2 file system layer for data read. Test results show that DMA could improve the performance of NAND flash read efficiently, and relieve NAND flash performance bottleneck.

**Key words:** PXA3xx processor; NAND; Direct Memory Access (DMA)

### 0 引言

在嵌入式非易失性闪存市场上, NAND Flash 以其大容量数据存储、快速读写擦除以及低价位赢得了广泛的应用。当前智能手机功能不断扩展, 要求有更大的存储空间来存放多媒体数据。IPOD、PMP/MP3、数码相机、存储卡都是 NAND Flash 的重要应用。

NAND Flash 作为外围高速设备, 其读写的速度直接影响到了整个系统的性能, 成为存储设备的瓶颈。本文引入 DMA 方案优化 NAND Flash 的操作效率, 设计编制程序, 并且得出直观测试数据来综合分析 DMA 优化效率。

本平台中的 PXA3xx 是 Monahans L/LV 系列处理器, 基于 ARM 体系<sup>[1]</sup>。由 Mavell 提供的一款最新高性能 XScale 架构芯片, 主频最高可达到 624 MHz, 配置 32KB LI ICache 和 DCache, 16 位 DDR SDRAM 工作在 260 MHz 频率下。相比前身 PXA27x 系列, PXA3xx 拥有更高的主频, 添加对 NAND Flash 的支持, 提升了 DDR 外部接口工作频率, 优化了 DMA 方案<sup>[2]</sup>, 摒弃了 Fly-By 模式。因此, PXA3xx 处理器在 NAND Flash 和 DMA 操作上所做出的增强, 对本文的优化实现是有利的。

NAND Flash 在大容量数据存储的嵌入式产品中有着重要的地位。由于 NAND Flash 硬件设计上的特殊性, 与 NOR Flash 相比, NAND Flash 不支持 XIP, 但是在写数据上比 NOR Flash 要快很多, 在擦除操作上, NOR Flash 一般是秒级, 而 NAND Flash 已达到毫秒级。当前对 NAND Flash 的加强主要

涉及到芯片、Host 接口和控制器三个方面<sup>[3]</sup>。本平台使用了 Micron 的 MT29F1GxxABB 系列 NAND Flash, 由嵌入式 Linux 来驱动<sup>[4]</sup> NAND Flash 的读写等操作。

### 1 NAND Flash 的中断 I/O 方式

中断机构引入后, 外围设备有了反映其状态的能力, 仅当操作正常或异常结束时才中断中央处理器, 实现一定程度的并行操作, 这叫程序中中断方式<sup>[5]</sup>。

在中断 I/O 方式中, 假设进程要求读取 NAND Flash, CPU 首先发出指令启动 NAND Flash, 而该进程则放弃处理器睡眠, 等待 I/O 操作完成, CPU 也调度其他就绪进程占用处理器。NAND Flash 在得到 CPU 的指令后, 陆续将数据放在 FIFO 缓存中, 并向 CPU 发出中断信号, 表示数据准备好了。CPU 是在识别 NAND Flash 中断后执行中断处理程序时, 将 FIFO 缓存中的数据读到内存的指定位置, 一次中断 I/O 读操作完成。CPU 在数据全部传输完成后唤醒该等待数据的进程进入就绪状态。

中断 I/O 方式的缺点是如果外围设备过多以及传送 I/O 缓存太小而传送数据过大时, 会导致中断次数急剧增加, 频繁切换上下文和现场保护, 耗去大量的 CPU 处理时间。如果中断被屏蔽, 软件可以通过轮询查看 NAND Flash 状态寄存器的位来完成程序直接控制 I/O 方式下数据的传输和命令的写入, 然而这样会浪费 CPU 太多的时间等待。由此, 提出了 DMA 方案来进行 NAND Flash 的数据传输。

收稿日期: 2009-03-03; 修回日期: 2009-04-18。 基金项目: 上海市科委重大项目 (08dz1500405)。

作者简介: 史斌 (1983-), 男, 上海人, 硕士研究生, 主要研究方向: 嵌入式 Linux 操作系统; 丁志刚 (1960-), 男, 上海人, 研究员, 主要研究方向: 嵌入式软件、软件测试; 张伟宏 (1975-), 男, 江西上饶人, 工程师, 硕士研究生, 主要研究方向: 嵌入式系统。

## 2 NAND Flash 的 DMA 方式

### 2.1 DMA 控制器架构布局

DMA 控制器 (DMAC) 所处的整个系统架构类似于 AMBA 总线结构的片上系统 SOC 架构<sup>[6]</sup>,如图 1 所示。PXA3xx 处理器内部交换总线 (Memory Switch) 提供 Initiator 与 Completer 之间的数据传输, Agent 是 Initiator 或者 Completer 的总称。CPU XScale Core、DMAC 作为 Initiator 连接到 Memory Switch, Initiators 是可以发起并初始化新的读写数据传输的 Agent。其他如动态内存控制器、Internal SRAM、系统总线则作为 Completers 连接到 Memory Switch 上, Completers 是完成数据传输的 Agent。通过在处理器内部使用交叉结构增加一倍的数据通路, Memory Switch 提升了多个控制器之间的内部数据带宽,同时因为更少的控制器使用单总线,系统的延迟大大降低。NAND Controller 挂载在系统总线上, NAND Flash 通过 NAND Controller 进行操作。大多数的周边设备控制器挂载在 DMAC/Bridge 单元所引出的周边器件总线上,比如 AC'97、I2C、MMC/SD、USB 等<sup>[7]</sup>。

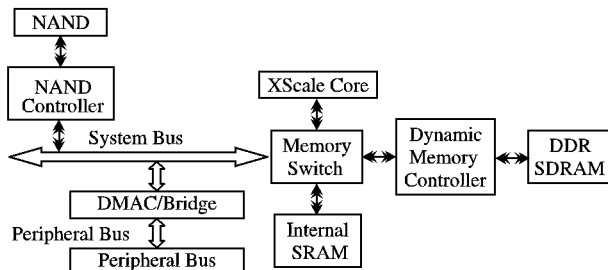


图 1 DMA 系统架构设计

系统总线配置有一个仲裁单元,根据预先设定的优先级来仲裁,以保证在任何时候总线上只有一个 Initiator 启动数据传输。NAND Controller 和 DMA 单元同时连接到系统总线,这样设计是为了 NAND Flash 的读写与 DMA 的控制之间能够有更好的速度和响应。

DMA 单元分为两部分:DMA Controller 和 Bridge。DMA Controller 供总线上各个外围设备使用进行快速数据传输,此时不需要处理器的干涉。Bridge 是桥接电路,可以让处理器通过 Bridge 存取外围设备,这样就绕开了系统 DMA Controller,由 CPU 进行数据操作。DMA 单元通过周边总线连接外围设备控制器,相比系统总线,周边总线架构简单,速度更低。

### 2.2 DMA 编程

DMA 使用 Flow-Through 传输方式,在数据被目标内存缓冲锁存之前,先通过 DMA Controller。本文使用描述符方式 DMA,每个描述符最大 8 191 B 数据传输。DDADR<sub>x</sub> 寄存器保存描述符的地址信息,在 DCSR 寄存器设置 RUN 启动后,处理器自动将 DDADR<sub>x</sub> 中地址所指出的描述符信息分别加载到 DSADR<sub>x</sub>、DTADR<sub>x</sub>、DCMD<sub>x</sub> 三个寄存器中,并且开始 DMA 数据传输,此时系统总线的控制权交由 DMA Controller, CPU 进行其他非系统总线的操作。在 DCMD<sub>x</sub>[LEN] 达到 0 时,传输结束,触发 DMA 完成中断。根据 DDADR<sub>x</sub>[RUN] 位,通道或者停止或者继续从内存中存取一个新的描述符。将 NDCR [DMA\_EN] 设置成 1,表示使能数据和命令 DMA。

DMA 可编程突发数据大小 32 B,可编程外围设备数据宽度 4 B。DMA 每次写入数据大小必须满足 32 B 的整数倍,如果不符合上述要求的话,就要通过软件在写入的数据末尾添加 Padding 信息直到满足 32 B 的整数倍。本文采用的 NAND Flash 每个 Page 是 2 KB + 64 B 的模式,符合条件要求 32 B

的整数倍,因而无需添加 Padding 信息。

### 2.3 嵌入式 Linux 下 NAND DMA 实现

本节详细阐述通过 DMA 方式读取 NAND Flash 一个 Page 的整个过程,读取多个 Page 数据的流程是相类似的。具体 NAND DMA 读操作流程如下:

1) 设置 NAND Controller 基址寄存器和 NAND Flash 时钟,判断 NAND Flash 型号并且初始化,包括 MFP、时序、NAND Controller 模式、相关寄存器。然后读取 NAND Flash ID 号,是否驱动支持匹配,如果没有对应的驱动,则报错退出。

2) 利用 Kernel 函数 kzalloc 分配 mtd\_info、nand\_chip、pxa3xx\_nand\_info 数据结构连续内存空间。定义 DMA 缓存的大小 buf\_len = 2 176 B。

使用 DMA 一致性的内存分配函数 dma\_alloc\_writecombine。最开始 16 B 是数据 DMA 描述符空间,紧接着依次是 16 B 命令 DMA 描述符空间、2 112 B 数据缓存空间和 32 B 命令缓存空间。之后还要初始化完成量 Complete 用于多个执行单元之间的同步,重定向表清零。

3) 使用 pxa\_request\_dma 申请 NAND DATA 和 CMD DMA 通道,并定义相应的 DMA 中断处理函数,保存申请到的 DMA 通道号。另外还要通过 request\_irq 申请 NAND 中断,用于以下两种情况:读数据时 FIFO 缓存中数据准备好;或者写数据时命令已经被写入命令缓存而 FIFO 缓存仍旧是空。

4) 之前的工作基本视为初始化,而后进入的是真正的读取操作了。建立 12 B 的 NAND Flash 命令字段,包含了 CMD1、CMD2、ADDR、命令控制信息、页计数,把命令字段放入命令缓存中。设置命令描述符,包括命令源地址、命令发送的缓存目标地址、突发数据 16 B、命令字长等。在启动命令 DMA 以前,先设置好数据 DMA 描述符,包括源数据缓存地址、目标地址、DMA 突发数据大小 32 B 和数据大小 2 112 B。

5) 启动写命令 DMA,等待完成量 cmd\_completion,进入休眠状态。Completion 是轻量级机制,允许一个线程告知另一个任务已经完成。此时总线的控制权交由 DMA Controller 控制, CPU 调度其他就绪状态的进程运行。

6) 直到命令字段成功写入到 NAND Controller 命令缓存中,触发命令 DMA 中断,进入中断处理函数,使能 RDDREQ 和 DBERR 中断位, RDDREQ 是在数据缓存中已经加载一页的读数据时发生中断, DBERR 是在数据流中发生两位错误时触发中断。完成处理后, CPU 返回或者调度其他进程占用处理器。

7) 在 NAND Controller 收到读命令后,立即与 NAND Flash 接口通信,将命令字段发送给外围 NAND Flash 设备; NAND Flash 在收到命令字段后,将对应的数据发送给 NAND Controller 的数据缓存,此过程无需 CPU 或者 DMA Controller 干预。如果数据传输发生错误,终止退出。

8) 在数据缓存中已经加载了一页的数据时,触发 RDDREQ 中断,表示数据已经准备好了,可以从 NAND Controller 传输到内存中去。

9) 先前数据描述符已经建立,立即启动数据 DMA 传输。中断返回 IRQ\_HANDLED, CPU 在此之后可做其他事。

10) 数据 DMA 传输完毕后,清空数据缓存,触发数据 DMA 中断。NAND Flash 状态设置为 STATE\_READY,唤醒 cmd\_complete 完成量。至此,一次读操作完成。

## 3 应用

为了考察 DMA 方案所带来的影响,本文研究 DMA 与非 DMA (中断 I/O 方式) 在 NAND Flash 上进行数据读取操作的

性能,通过在 JFFS2 文件系统上进行测试,得出数据和结论。

### 3.1 测试设计

/proc/stat 文件跟踪最近一次重新启动后系统统计数据的变化,通过/proc/stat 可以得到 CPU 各模式下的 *jiffies* 数目,这些模式包括用户模式 (User)、低优先级下的用户模式 (Nice)、系统模式 (System)、空闲任务 (Idle)、lowait 等待、Irq 和 Softirq。用户模式下 CPU 负载即是在一定的时间间隔内用户模式的时间占整个运行时间的百分比。在这里,时间是自用户上次读取 *jiffies* 数目到这次读取 *jiffies* 之差与频率的乘积。很自然地想到,计算这个 CPU 负载值时可以利用:

$$CPU\text{Load} = \frac{\Delta jiffies \times HZ}{\Delta time} \quad (1)$$

$\Delta jiffies$  容易得到,只需将当前读取的用户模式 *jiffies* 值 *lastJif* 与  $\Delta time$  时间前读取的用户模式 *jiffies* 值 *prevJif* 相减即可:

$$\Delta jiffies = lastJif - prevJif \quad (2)$$

$\Delta time$  的值,并非使用 PXA3xx 处理器 Timer 的设置来决定,而在本文中使用了总消耗的 *jiffies* 的数目 *totalJif* 与 HZ 的乘积来表示。综上所述,计算 CPU 负载的公式就是:

$$CPU\text{Load} = \frac{(lastJif - prevJif) \times HZ}{totalJif \times HZ} = \frac{lastJif - prevJif}{totalJif} \quad (3)$$

表 1 非 DMA 方式下 JFFS2 读 10 MB 数据 CPU 负载统计

| 方式     | CPU       | 模式         |      |             |            |            |          |          | jiffies/% |
|--------|-----------|------------|------|-------------|------------|------------|----------|----------|-----------|
|        |           | User       | Nice | System      | Idle       | lowait     | Irq      | Softirq  |           |
| 有 SD 读 | 324.4/100 | 51.4/15.84 | 0/0  | 188.4/58.08 | 0/0        | 83.4/25.71 | 1.2/0.37 | 0/0      |           |
| 无 SD 读 | 247.4/100 | 14.6/5.91  | 0/0  | 136.8/55.30 | 95.8/38.72 | 0/0        | 0/0      | 0.2/0.07 |           |

表 2 DMA 方式下 JFFS2 读 10 MB 数据 CPU 负载统计

| 方式     | CPU       | 模式         |      |             |            |            |           |         | jiffies/% |
|--------|-----------|------------|------|-------------|------------|------------|-----------|---------|-----------|
|        |           | User       | Nice | System      | Idle       | lowait     | Irq       | Softirq |           |
| 有 SD 读 | 261.8/100 | 43.8/16.73 | 0/0  | 128.4/49.05 | 0/0        | 70.4/26.89 | 19.2/7.33 | 0/0     |           |
| 无 SD 读 | 194/100   | 14.4/7.42  | 0/0  | 83.2/42.89  | 84.8/43.71 | 0/0        | 11.6/5.98 | 0/0     |           |

表格中,“/”前后的数据分别表示 *jiffies* 数目以及它所占用的总 *jiffies* 的百分比。CPU 表示该次读取操作的总消耗, User、Nice、System、Idle、lowait、Irq 和 Softirq 分别表示在特定模式下的消耗。可以看到:

1) 无 SD 卡后台进程时 DMA 方式下的 Idle 负载是 43.71%,而非 DMA 方式则下降到了 38.72%。可见,DMA 能使 CPU 拥有更多的空闲时间,也表明 DMA 能够降低 CPU 负载。当在 SD 卡后台进程运行时,Idle 变为了零,可见 CPU 此时在全力运作,在相等的条件下,率先完成 NAND Flash 10MB 数据读取任务的方式效率较高,因此,DMA 方式的 261.8 明显优于非 DMA 方式的 324.4。

2) 无论 SD 卡读取后台进程存在否,DMA 方式所消耗的总 *jiffies* 数目比非 DMA 方式少很多。在无 SD 卡读取后台进程时,非 DMA 方式要比 DMA 方式慢 27.53% 的 *jiffies*;而在有 SD 卡读取后台进程时,这个比例也有 23.91%。

3) DMA 方式下,System 负载明显要低于非 DMA 方式,而 User 负载略高,此时 CPU 更少地运行于 System 模式下,而由 DMA 来完成相关的数据传输任务。

4) DMA 的一个显著特点是 Irq *jiffies* 的出现。由于 DMA 的传输需要经过 CPU 的确认和处理,因而两者之间的通信握

本测试是在 JFFS2 文件系统上进行的。首先在后台运行一个 SD 卡的数据读取进程,该进程依次读取 SD 卡上总共 439 MB 的 17 个大小各不相同的文件。之所以选择这些不同大小的文件,是为了让处理器的缓存 Cache 命中率降低,由于 PXA3xx 的 SD 控制器位于周边总线上,使 SD 卡的读取操作都要经过系统总线到周边器件上。经测试,在没有其他进程的条件下读取 SD 卡耗时大约 55 s,这样能够保证,在 NAND Flash 测试读取 10 MB 数据完成以前,SD 卡始终忙于读取;由于 SD 卡读取与 NAND Flash 读取是共用系统总线的(参考图 1 所示),即表示 NAND Flash 的测试是在 CPU 和系统总线充分饱和的状态下进行的。

### 3.2 数据分析

通过 Linux Kernel 中设计的 CONFIG\_MTD\_NAND\_PXA3xx\_DMA 宏开关,分别编译非 DMA 和 DMA 方式下的 zImage,各自加载测试。其中数据都是在相等的条件下,重新启动系统保持稳定后,选取五组重复测试的数据,然后求取平均值得到的,因此具有较好的可靠性。

本文根据非 DMA 与 DMA 方式、后台运行 SD 卡读数据的进程与后台不运行 SD 卡读数据的进程这两个因素组合,形成如表 1 和 2 的数据。

手通过 Irq 方式,Irq 负载大约占到了 6.5% 的比例。在本文设计的程序中,DMA 的两个通道各申请了一个 Irq 中断。

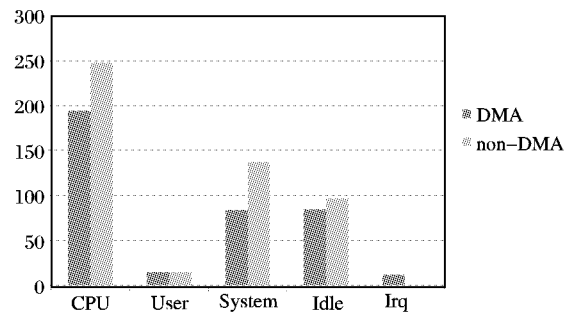


图 2 各 CPU 模式下负载情况柱形图

为了直观反映各 CPU 模式下负载的情况,如图 2 所示的柱形图是在无 SD 卡读取后台进程时测试数据编制的。图中纵坐标的单位是 *jiffies*,横坐标列出各种模式,并对 DMA 与非 DMA 方式进行了比较。总体来看,该图也表明了 DMA 方式的优势。

测试表明:在嵌入式 Linux 的 JFFS2 文件系统层进行 NAND Flash 上数据的读取测试,DMA 由于协助 CPU 完成一些任务,CPU 负载比较低。

(下转第 2142 页)

速缓存缺失率以至于达到了 95%, 这意味着高速缓存和内存的换页频率很高。与此同时从计算时间的数据上可以看出, 随着数据规模的增大, 算法的运行时间增长飞速, 当数据规模达到 8 192 时, 算法的运行时间上升到 4.35 s。而对于优化后的算法而言, 从数据规模为 8 192 来看, 算法的运行时间从 4.350 s 降低到了 0.89 ms, 而这速度的提高完全是由高速缓存缺失率的减少而带来的。原因在于优化后的算法属于高速缓存感知算法, 高速缓存缺失率的大小明显小于原始算法的高速缓存缺失率, 这意味着对于优化后的算法而言, 大量的数据都是直接从高速缓存中读取的, 而原始算法大多需要内存和高速缓存进行数据块传递以后才能够从高速缓存中获得需要的数据, 因此优化后的算法在运算性能上大大高于原始算法。

本组实验将把多线程优化后的高速缓存感知的 Haar 小波变换算法在多核平台上进行性能测试, 所采用的多核平台硬件参数如表 2。本文算法在不同平台上运行所用的线程数由主机的处理器核数决定, 并与处理器核数保持相同。优化后的算法在以上四种平台上的运行时间(单位为 ms)统计如表 3。

表 2 硬件平台参数

| 平台 | CPU                             | 主存/<br>GB | 二级缓<br>存/KB |
|----|---------------------------------|-----------|-------------|
| 单核 | Pentium D CPU 3.00 GHz (单线程模拟)  | 1         | 2×2048      |
| 双核 | Pentium D CPU 3.00 GHz 双核       | 1         | 2×2048      |
| 四核 | Intel Xeon CPU 5110 1.60 GHz 四核 | 4         | 1×4096      |
| 八核 | Intel Xeon CPU 5110 1.60 GHz 八核 | 4         | 2×4096      |

表 3 算法在不同计算平台下运行时间 ms

| 计算规模<br>(阶方阵) | 计算平台 |     |     |     |
|---------------|------|-----|-----|-----|
|               | 单核   | 双核  | 四核  | 八核  |
| 1024          | 16   | 14  | 10  | 6   |
| 2048          | 46   | 31  | 20  | 16  |
| 4096          | 203  | 109 | 78  | 63  |
| 8192          | 890  | 453 | 281 | 230 |

不同规模的数据量在四种平台上产生的加速比曲线如图 5 所示, 可以看出: 对于每条加速曲线来说, 随着多核平台处理器核数的增加, 加速曲线都呈现出增长的趋势, 不同处理器平台上可以进行运算的处理器核数各不相同。对于每条加速曲线来说, 数据规模相同, 由于线程数随着处理器核数的增加而增加, 因此分配到每个线程的数据量随着处理核的增加而变小, 且操作系统分配给该进程的时间片会增加, 因而整个算法在数据规模一定的时候会随着处理器核数的增加而减少时

间, 进而提高加速比。

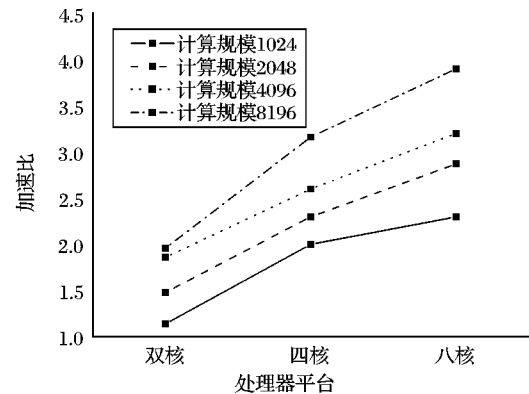


图 5 算法加速比曲线

## 4 结语

本文在多核计算平台的基础上, 充分利用多核平台的计算潜力实现了一种高效的 Haar 小波变换快速算法。本文所提出的 Haar 小波变换计算是一种高速缓存参数相关性感知算法; 同时, 为了进一步提升算法性能, 在分析了算法特点的基础上加入了多线程处理。从实验可得, 在 1 024 到 8 192 数据规模上, 缺失率在 10% 以内, 计算时间降至 1 秒以内。通过对不同多核平台下算法运行性能的比较可以看出, 本文提出的 Haar 小波变换快速算法具有较高的高速缓存利用率且能够在多核平台上发挥较好的计算性能。

## 参考文献:

- [1] VISHWANATH M. The recursive pyramid algorithm for the discrete wavelet transform [J]. IEEE Transactions on Signal Processing, 1994, 42(3): 673-676.
- [2] VISHWANATH M, OWENS R M, IRWIN M J. VLSI architectures for the discrete wavelet transform[J]. IEEE Transactions on Circuits and System-II, 1995, 42(5): 305-316.
- [3] G. MALLAT S. A theory for multiresolution signal decomposition: The wavelet representation [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1989, 11(7): 674-693.
- [4] FRIGO M, LEISERSON C E. Cache-oblivious algorithms [C]// Proceedings of the 40th Annual Symposium on Foundations of Computer Science. Washington, DC: IEEE Computer Society, 1999: 285-297.
- [5] KOWARSHIK M, CHRISTIAN W. An overview of cache optimization techniques and cache-aware numerical algorithms[C]// Algorithms for Memory Hierarchies, LNCS 2625. Berlin: Springer, 2003: 213-232.

(上接第 2138 页)

## 4 结语

从本文测试的数据图表可以看出, DMA 方式能够有效地缓解 CPU 处理压力, 使 CPU 处理其他与数据总线无关的操作, 而由 DMA 负责 NAND Flash 的数据传输机制, 使存取效率有所提高, 一定程度上减轻了 NAND Flash 性能瓶颈问题。对于现今的多通道、多任务、流水线、缓存体系结构的系统来说, DMA 能够实现数据传输和 CPU 运算的并行, 是当前芯片设计中的重要方案。

## 参考文献:

- [1] 杜春雷. ARM 体系结构与编程[M]. 北京: 清华大学出版社, 2003.

- [2] 华清远见嵌入式培训中心. Linux 设备驱动开发详解[M]. 北京: 人民邮电出版社, 2008.
- [3] MIN S L, NAM E H. Current trends in flash memory technology [C]// Asia and South Pacific Conference on Design Automation. [S.l.]: IEEE, 2006: 332-333.
- [4] CORBET J, RUBINI A, KROAH - HARTMAN G. Linux Device Drivers[M]. 3rd ed. [S.l.]: O'Reilly, 2005.
- [5] 孙钟秀, 费翔林, 骆斌, 等. 操作系统教程[M]. 3 版. 北京: 高等教育出版社, 2003.
- [6] 谢勇, 申敏, 郑建宏. AMBA 总线结构中高性能 DMA 控制器的硬件实现[J]. 重庆工学院学报: 计算机与自动化版, 2006, 20(8): 72-74.
- [7] Marvell Corporation. MV-S301374-01, Rev. C [EB/OL]. (2008-03-19) [2009-02-01]. <http://www.marvell.com>.