第36卷 第15期 计 算 机 工 程 2010年8月
Vol.36 No.15 Computer Engineering August 2010
·开发研究与设计技术· 文章编号：1000—3428(2010)15—0280—03 文献标识码：A 中图分类号：TP303

微处理器存储管理单元的功能验证方法

李 智[1]，周大钧[2]，龚令侃[1]
(1. 华东计算技术研究所，上海 200233；2. 山东轻工业学院现代教育中心，济南 250353)

摘 要：在分析存储管理单元(MMU)验证方法的基础上设计一种验证专用操作系统(VPOS)。采用静态存储管理、静态用例调度和伪中断处理等技术，为编写可执行、可控制的MMU测试程序提供了软件平台。验证表明，基于VPOS的仿真能在早期的软仿真阶段覆盖94%的MMU设计错误，在FPGA验证中覆盖剩余的错误，保证了移植通用操作系统一次成功。

关键词：功能验证；存储管理单元；验证专用操作系统；静态存储管理；静态用例调度；伪中断处理

Functional Verification Method for Memory Management Unit of Microprocessors

LI Zhi[1], ZHOU Da-jun[2], GONG Ling-kan[1]
(1. East China Institute of Computer Technology, Shanghai 200233;
2. Modern Educational Technology Center, Shandong Institute of Light Industry, Jinan 250353)

【Abstract】This paper puts forward a Verification Purpose Operating System(VPOS) based on the verification for Memory Management Unit(MMU), which is capable of static memory management, static case scheduling and pseudo interrupt handling. Designed for hardware verification only, VPOS provides an executable and controllable platform for MMU test-program development. Verification practice shows that VPOS based simulation can cover 94% of the MMU bugs very early in the project, which releases the debugging pressure on later FPGA verification by porting General Purpose Operating System(GPOS). Therefore, it guarantees the one time success of transplanting general operating system.

【Key words】functional verification; Memory Management Unit(MMU); Verification Purpose Operating System(VPOS); static memory management; static case scheduling; Pseudo Interrupt Handling(PIH)

1 概述

存储管理单元(Memory Management Unit, MMU)是多任务系统中微处理器的关键功能部件之一，主要负责完成有效地址到物理地址的转换、存储保护以及访存的中断处理。在微处理器设计过程中，MMU 设计验证相对困难，需要寻求新技术和新方法，以保证验证的充分和高效。

微处理器功能验证方法主要有形式验证和仿真验证2类[1-2]。形式验证虽然能100%证明设计的正确性，在发现错误时给出反例，但由于状态空间爆炸的问题，只用于规模不大的模块级验证[2-3]。目前微处理器的功能验证仍然以仿真验证为主[4-5]。验证人员开发一些测试程序，同时送到待测模型和标准模型执行，动态比较发现待测模块的错误。

测试程序来源之一是随机生成的指令流 RTPG(Random Test Program Generation)[2]。IBM 的 Genesys-Pro 及 Obsidian 的 RAVEN 等都提供了基于随机指令的微处理器验证平台，验证人员通过该平台初始化系统寄存器和存储器，为测试程序配置 MMU 控制寄存器和页表项，再由工具生成随机指令流的 MMU 测试程序[3]。

但是这样的测试程序不具备“可执行性”，即不能在真实的处理器上运行，因为随机指令流显然无法完成处理器资源的初始化[3]。测试程序的“可执行性”是保证它们本身正确性的保障；另一方面，可执行的测试程序也可以在 FPGA 平台上运行，对于一些复杂冗长的测试，FPGA 的高性能可以大大缩短验证时间。

测试程序另一主要来源是系统软件，如通用操作系统(General Purpose OS, GPOS)，移植系统软件可以充分验证MMU，但“可控性”差，难以根据验证需要配置地址转换机制，事实上，通用操作系统更适合做设计签署(Design Sign-off)，以提高对设计在真实应用中的信心，而不是做以调试错误为目的的验证。

由此可见，MMU 的验证需要一种能像 RTPG 那样灵活配置处理器状态、又能像 GPOS 那样执行的测试程序。本文提出的以验证专用操作系统(Verification Purpose OS, VPOS)为平台的验证方法可以把现有方法的优点结合起来。VPOS 负责为测试程序初始化处理器状态，使在 VPOS 上开发的测试程序是“可执行的”；简化了调度、存储管理和中断处理算法，又使 VPOS“灵活可控”。在该平台上，验证人员能高效地开发各种测试程序(包括 RTPG)。在微处理器设计过程中，采用基于 VPOS 的验证方法完成了 MMU 软仿真。

2 验证专用操作系统VPOS

基于 VPOS 的验证平台如图 1 所示。编译好的 VPOS 和测试程序分别被载入到内存和闪存中。模仿真实操作系统，测试首先进入 VPOS，由它为测试程序初始化处理器，调度并复制测试程序到内存，然后转入测试程序执行。当发生中

作者简介：李 智(1985－)，男，硕士研究生，主研方向：计算机系统结构，数字系统设计；周大钧，讲师；龚令侃，硕士
收稿日期：2010-02-24 **E-mail：**zhil@mail.ustc.edu.cn

断或者程序正常退出后，控制权又将回到 VPOS，处理中断或为后继的测试程序初始化处理器。

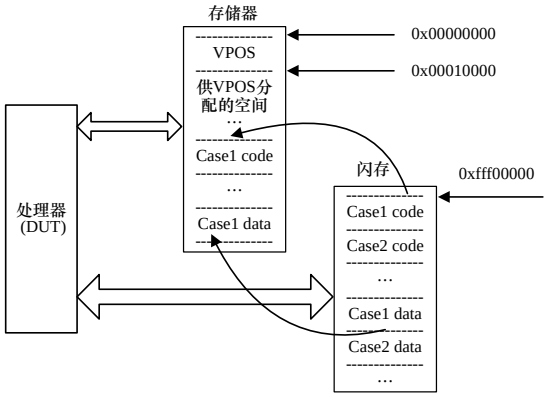

图 1　基于 VPOS 的验证平台

与 GPOS 的算法相比，VPOS 使用简化了的算法为测试程序提供服务，包括静态用例调度、静态存储管理和伪中断处理(Pseudo Interrupt Handling, PIH)。整个操作系统由 3 个部分组成。

(1)中断处理程序

VPOS 的大部分中断处理程序(Interrupt Service Routine, ISR)使用伪中断处理。它们不进行常规中断服务，而是由软件检测中断相关的寄存器是否被正确设置。VPOS 用 PIH 验证各种访存中断，如违反保护和操作数未对齐。

另一些特殊的中断不能使用 PIH，如系统调用、缺页中断。测试程序需要它们完成静态用例调度和页表查找。

(2)初始化部分

初始化部分负责测试程序运行环境的初始化。VPOS 初始化部分采用静态存储管理算法实现。

(3)用例管理数据

用例管理数据(Case Control Block, CCB)是一系列预定义的数据结构。它们描述了调度上下文和内存管理信息。每个测试程序都有一条 CCB 记录和一个 CCB 入口。VPOS 利用它们进行静态用例调度和静态存储管理。

2.1　用例管理数据

一个简单的 CCB 示例代码如下所示：

```
// ==== File: VPOS_CCB.s ==================
// ==== EA=0x3000 Case Managment Data =====
section VPOS_CCB
_case_CCB_start:
// Case 1 ===================
// CCB_length ===============
.long /*line 1: CCB_length*/                    0d60
// Memory Allocation Address ===
.long /*line 2: CaseDataFlaStart Addr*/              0xfff40000
.long /*line 3: CaseData MemStart Addr*/    0x00080000
.long /*line 4: CaseText FlaStart Addr*/       0xfff20000
.long /*line 5: CaseText MemStart Addr*/    0x00020000
// Scenario Fabrication ========
.long /*line 6: Machine State*/    (MSR_DR | MSR_PR)
.long /*line 7: Case Entry Point*/ 0x00020000
// MMU Register Configure =====
.long /*line 8: MMU register0 */  …
.long /*line 9: MMU register1 */  …
.long /*line 10:Num*/      0x00000001
// PTE 0x40000 => 0x80000 ====
// PTE Supervisor Only ========
.long /*line 11: EA */      0x00040000
.long /*line 12: PA */      0x00080000
.long /*line 14: Prot */    PTE_Supervisor_Only
// Additional Pointer ==========
.long /*line 15: Null */    CCB_Null
```

CCB 示例包括 CCB 大小、内存静态分配、上下文伪造、MMU 控制寄存器和页表项配置等内容：

(1)CCB 大小。每个测试程序 CCB 记录的大小是可变的，VPOS 通过它找到测试程序 CCB 记录的入口。示例 CCB 占用了 60 Byte 的空间。

(2)内存静态分配信息。VPOS 根据 CCB 把测试程序的数据段从起始地址为 0xfff40000 的闪存静态分配到起始地址为 0x00080000 的内存中。

(3)MMU 控制寄存器和页表项。MMU 控制寄存器与体系结构相关，页表项的配置需要与内存静态分配对应。示例 CCB 记录了一个从有效地址 0x40000 到物理地址 0x80000 的页表项。

(4)上下文伪造。模仿 GPOS 进程调度的原理，VPOS 的 CCB 记录了伪造上下文信息以完成用例调度。示例记录了测试程序的机器状态字(Machine State)和程序入口地址(Entry Point)。

(5)指向其他参数的指针。

2.2　VPOS中的静态算法

CCB 的引入使得静态调度和内存管理成为了可能。所谓的“静态”，就是在验证人员开发测试程序的同时为测试程序调度用例和管理内存。

静态调度采用非抢占式的顺序调度策略，静态存储管理采用独占物理内存的方法。通过读取各个测试程序的 CCB，VPOS 可以获得静态算法所需的全部信息。静态用例调度和静态存储管理的程序流程如图 2、图 3 所示。

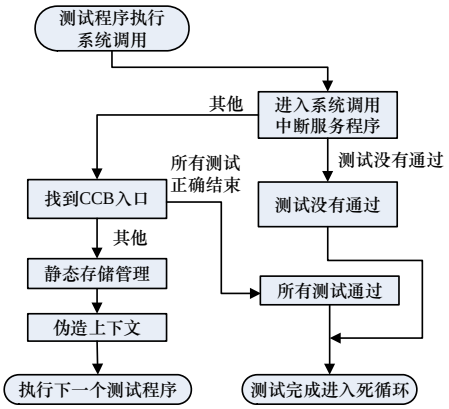

图 2　静态用例调度的程序流程

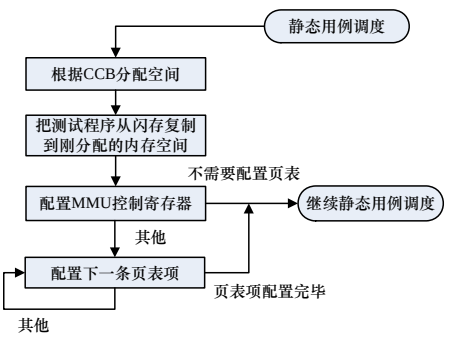

图 3　静态存储管理的程序流程

当测试程序正常退出后，通过系统调用中断进入 VPOS 内核。系统调用中断服务将检查测试程序的结果正确与否，并决定是否继续下一个测试。如果正确，VPOS 将找到下个测试的 CCB 入口，静态分配内存，伪造上下文，然后转入测试程序执行。如果错误，或者所有测试程序都已执行完毕，VPOS 会进入相应的后处理程序。

静态存储管理通过读取 CCB 的记录分配内存空间、复制测试程序的代码和数据、配置 MMU 寄存器和页表项。待所有页表项都配置完毕后，VPOS 继续执行静态用例调度。

2.3 伪中断处理

VPOS 采用伪中断处理技术验证访存中断。伪中断处理的程序流程如图 4 所示。测试程序主体把中断相关控制寄存器的预期值存入通用寄存器(GPR)，然后执行一条会触发中断的指令。如果中断没有正确产生，程序将继续执行进入“陷阱区”，无条件跳转到出错处理处；否则，程序将进入中断 ISR 执行。

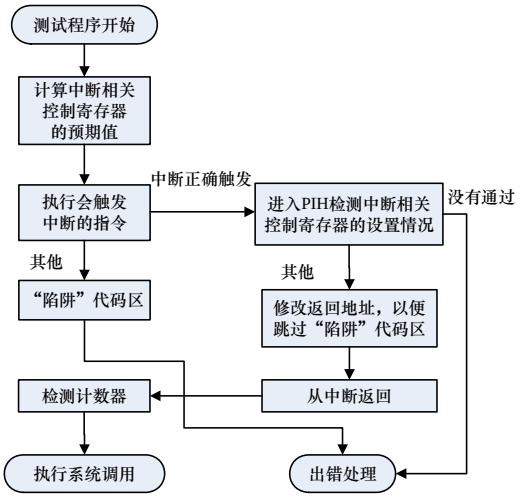

图 4 伪中断处理流程

违反存储保护的 PIH 中断服务程序示例代码如下所示：

```
// ==== File: VPOS_PIH.s ============================
// ==== EA=0x0300 Pseudo Interrupt Handler =====
.section VPOS_pih
// line 1-2:Check the interrupted PC register
//       Expected value of the interrupted PC
//       should be put into R13 in main program
   R3 <- (Register containing interrupted PC)
   CHECKREG R3,R13
// line 3-4:Check the offending EA register
   R4 <- (Register containing offending EA)
   CHECKREG R4,R14
// line 5: Add counter
   Counter <- Counter +1
// line 6: Set new return address in PC. The return
//     address should be put into R23 in main program
   PC <- R23
```

通常的处理器都会有一个专用的寄存器存放中断指令的地址(interrupted PC)和指令访问的存储器有效地址(offending EA)。第 1 行~第 4 行分别从寄存器中读出这些值，和 GPR 内的预期值比较。然后，ISR 会修改计数器以备统计结果；重新设置中断返回地址以越过程序主体的“陷阱区”；最后从中断返回。

3 基于VPOS的MMU验证

VPOS 为 MMU 测试程序的开发提供了灵活可控的平台。在该平台下，针对自主设计的嵌入式处理器 MMU 特性，开发了大量 MMU 测试程序，完成了 MMU 的仿真验证。

3.1 基于VPOS的测试程序设计

基于 VPOS 的测试程序设计包括 2 个环节。首先，验证人员需要定义测试程序的 CCB，为测试程序静态定义内存管理和上下文。验证人员可以灵活控制测试程序的运行环境以验证某个特定的功能点。其次是测试程序主体的开发。测试程序完全独立于 VPOS 的算法，可以使用各种已有的技术(如 RTPG)生成测试程序。

如下所示的示例测试程序代码与 CCB 和 PIH 结合，能完成违反访存保护的中断。程序主体把中断指令的地址(INT_PC)、违规有效地址(OFF_EA)和 PIH 返回地址(RETURN_PC)放入通用寄存器，然后触发中断。经过伪中断处理后，程序检查计数器设置并执行系统调用，提交 VPOS 处理。

```
// ==== File: VPOS_case.s ==========================
// ==== EA=0x00020000 Case Code ====
.text
// line 1-2: Prepare expected value for PIH
//           Load the expected value of the interrupted PC
//           and the offending EA to R13,R14 respectively
R13 <-   IMM(INT_PC)
R14 <-   IMM(OFF_EA)
// line 3: Prepare return address for PIH
//          Load the return address to R23
R23 <-   IMM(RETURN_PC)
INT_PC:
// line 4: This instruction will invoke the interrupt
R8   <-   MEM(OFF_EA)
// line 5: Trap
PC   <-   IMM(Error_Handling_Entry_Address)
RETURN_PC:
// line 6-7: Return from PIH, check counter and
//           system call
CHECKREG Counter
System Call
// ==== EA=0x00040000 Case Data =====
.data
OFF_EA: .long 0x00fedcba,0x98000000
```

3.2 验证结果分析

基于 VPOS 开发了大量 MMU 测试程序。表 1 列出了主要的测试程序集。验证人员根据设计规范编写测试程序和相应的 CCB，覆盖 MMU 的各个功能点。每个程序集包括仿真验证和 FPGA 验证 2 个子测试程序集。仿真验证的测试一般比较短小，测试单个的 MMU 功能；FPGA 验证的测试程序往往比较复杂，VPOS 根据 CCB 构造各个 MMU 场景的排列组合，并执行测试程序。

表 1 主要 MMU 测试程序集

主要测试程序集合	主要测试点	CCB 设计重点
mmu_sim	基本 MMU 流程；基本地址转换	配置部分 PTE
mmu_align	操作数不对齐中断	配置全部 PTE
mmu_le	小端模式下 MMU 中断	机器状态设成小端
mmu_misc	一些特殊指令引起的 MMU 中断	特殊的 MMU 配置
mmu_prot	违反保护机制	配置单个的 PTE 保护
mmu_cross	跨保护边界访存	配置跨边界的 PTE 保护

（下转第 285 页）