第 35 卷 第 3 期 计 算 机 工 程 2009 年 2 月
Vol.35 No.3 Computer Engineering February 2009
·软件技术与数据库· 文章编号：1000—3428(2009)03—0086—03 文献标识码：A 中图分类号：TP311

基于验证库的微处理器指令集验证方法

龚令侃，王玉艳，章建雄
(华东计算技术研究所，上海 200233)

摘 要：指令集作为微处理器软件和硬件的分界线在计算机体系结构中占有重要地位。测试程序自动生成(RTPG)是微处理器指令集验证的主要方法之一。该文比较目前主流的 RTPG 技术和验证策略，提出基于验证库的随机测试程序生成工具。使用通用脚本语言开发验证库和测试程序模板，针对不同验证阶段生成高质量的测试程序。测试结果表明，该方法实现简单，能达到较好的验证效果。

关键词：微处理器；指令集验证；随机测试程序生成；验证库

Library-based Verification Methodology for Instruction Set Validation of Microprocessor

GONG Ling-kan, WANG Yu-yan, ZHANG Jian-xiong
(East China Institute of Computer Technology, Shanghai 200233)

【Abstract】Instruction set, as the interface between software and hardware, plays an important role in computer architecture. Random Test Program Generation(RTPG) is one of the efficient ways to verify an instruction set. After comparing some existing RTPG technologies and verification strategies, this paper proposes a library-based RTPG tool for instruction set verification. By developing both library and test template using a general purpose script language, it is capable of generating high quality test in different test stage. Results of test show that it is easy implemented, and on the meantime, it can get a satisfying verification result.

【Key words】microprocessor; instruction set verification; Random Test Program Generation(RTPG); library-based verification

1 概述

随着微处理器硬件规模和复杂度的增长，功能验证已成为设计流程的瓶颈。指令集体系结构(Instruction Set Architecture, ISA)划分了整个计算机系统的软件和硬件，是处理器的核心，指令集的验证自然也成为微处理器验证最重要的一部分。

测试程序的自动生成(Random Test Program Generation, RTPG)是一种有效的验证处理器指令集的方法。RTPG 算法可分为伪随机和半形式化 2 种[1]。伪随机方法的主要代表如 GENESYS[2]和 MA2TG[3]等系统，它们定义某种语言为指令建立模型(约束限制)，再通过求解一个约束满足问题(Constraint Satisfy Problem, CSP)生成高质量的测试程序。这种方法的缺点是工具的开发工作比较费时，约束求解的算法和建模语言的定义都比较复杂。半形式化方法主要针对流水线及控制部件，通过遍历状态机生成测试程序，如 BITG 系统[4]，该方法理论上能取得 100%的覆盖率，但由于状态空间爆炸问题，半形式化验证只能用于模块级的验证，实际上，目前工业界验证仍以仿真验证为主[5]。

功能验证的策略主要分为软件自测试和基于比较的验证 2 种方法。在传统的软件自测试中，验证人员使用汇编语言编写测试程序，程序完成寄存器初值的设置、待测指令的执行及结果检测等所有功能。这种方法简单实用，测试程序可在任何平台(仿真验证，FPGA 或真实处理器)上运行，平台间“移植”的开销小。但开发维护困难，在汇编程序中需要大量初始化和结果检测的代码，而真正用于测试激励的待测指令只是一小部分；基于比较的验证需要一个作为比较的标准模型，验证人员将测试程序同时送到待测模型和标准模型执行，动态比较执行结果，查找待测模块错误，并统计功能覆盖率等各种数据。国内有很多这方面的研究[5-6]，这种方法验证效果较好，但比须搭建一个复杂的验证平台，而标准模型本身的验证工作也比较困难。

本文提出一种新的测试程序自动生成技术——基于验证库的微处理器指令集验证方法，验证库为测试程序的开发封装了“平坦”的环境，验证人员通过调用库函数编写测试模板生成测试程序。相比伪随机方法，它更简单灵活，同时又能完成半形式化方法无法完成的全芯片级的验证；它发挥了软件自测方法“可移植”的优点，使验证人员不必再为低级的汇编代码编程而苦恼。

2 验证计划

在不同的微处理器功能验证阶段，须针对不同的验证目的，制定验证计划并开发测试程序。

(1)体系结构级的验证(Architecture Verification Plan, AVP)，主要验证单条指令的基本功能。测试程序关注的重点主要是操作数的选择，合适的操作数能激发更多的指令的功能。为减小调试成本，该阶段以确定性的测试程序为主。

(2)实现级的验证(Implementation Verification Plan, IVP)，这个阶段主要验证与微体系结构相关的处理器指令特性，如流水线、分支预测等，验证重点是指令的组合序列，旨在激发处理器内部的各种状态，操作数的选取并不重要，因此，

作者简介：龚令侃(1983－)，男，硕士研究生，主研方向：计算机系统结构，数字系统设计；王玉艳，高级工程师；章建雄，研究员
收稿日期：2008-06-13 **E-mail：**glk47@hotmail.com

可使用随机生成的测试程序验证。

(3)移植一些应用程序或通用操作系统等，本文不加以讨论。

3 基于验证库的 RTPG 技术

基于验证库的 RTPG 技术本质上只是一种通用脚本语言的程序设计，无论验证库还是测试程序模板都用同样的通用脚本语言编写。测试模板通过调用库函数和脚本语言自带的函数生成汇编语言源程序，同时具有汇编语言和高级语言的编程风格。这样省去了模板语言的定义及其编译器的开发工作，只需使用通用脚本解释器即可自动生成测试程序。软件自测试的验证策略简化了库函数的设计难度。基于验证库的 RTPG 生成的是汇编语言源代码(文本文件)，可用通用的汇编器编译成可执行文件，加载到任何平台运行，任何一种脚本语言都能完成本文研究使用 Perl 脚本语言。

3.1 生成引擎的工作原理

测试程序生成引擎结构如图 1 所示。测试开发工程师用脚本语言编写测试模板，调用验证库的函数。

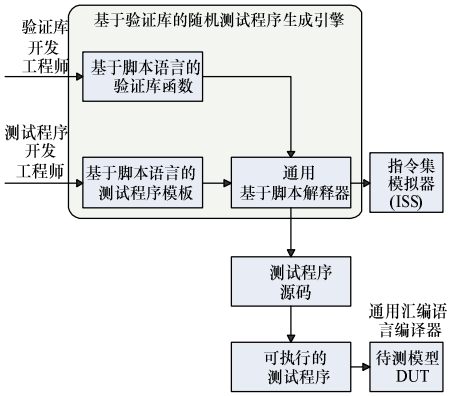

图 1 基于验证库的 RTPG 引擎结构

基于软件自测试的测试程序主要分为处理器初始化、待测指令的执行和结果检测 3 个部分，测试模板只显示定义前 2 个部分，经脚本解释器解释执行后，生成只有初始化和待测指令的汇编程序，称为部分测试程序(Partial Test Program, PTP)。

测试模板的第 3 部分是一条启动仿真的库函数，它将 PTP 送往指令集模拟器执行，并返回执行结果。这些的返回值撤是生成结果检测汇编指令的依据，PRPG 引擎将这最后生成的这部分汇编指令同 PTP 合在一起，输出完整的基于软件自测试的测试程序。

3.2 验证库

验证库函数应尽可能封装脚本语言本身的特性，使测试开发工程师可忽略脚本语言的细节而专注于测试程序本身。基于上述考虑，库函数与测试程序的 3 个部分相对应，分为以下几类：(1)生成初始化代码的函数；(2)执行待测指令的函数；(3)启动仿真器并生成检测代码的函数；(4)提供控制测试程序生成的接口函数。

处理器初始化的测试模板示例如下：

```
# RTPG processor initialization data structure
  %rtpg_reg =   (
    "r8"     => 0x7fff,
    "r9"     => 0x0001,
    "reg"    => rand_data(),
  );
  %rtpg_mem =   (
    "address1" => "ea"     => 0x00010000,
                  "pa"     => rand_addr(),
                  "data" => rand_data(),
  );
  rtpg_init_sys();
…
```

其中，%*rtpg_reg* 和%*rtpg_men* 是 2 个全局变量，通过对其赋值，然后调用 *rtpg_init_sys*()函数即可产生初始化处理器的指令序列。*rand_addr*()和 *rand_data* 是产生随机数据和地址的库函数。

执行类库函数根据不同的指令类型设计。算术运算指令和访存指令如下：

```
#RTPG instruction under test library function example
#RTPG control library function example
  Switch(rtpg_ctrl_selc(2)){
case 1{
    rtpg_exec_arth("ADD reg r8 ?");}
case 2{
    rtpg_exec_ldst("LOAD reg address1")}}
…
```

控制类库函数增加了模板开发的灵活性，包括指令选择、循环和条件控制等。*rtpg_ctrl_selc*()是选择控制函数，它将随机选择一个库函数执行，即随机生成加法或访存。该类函数随机选取该类指令中的某一条，组合成一条完整的指令。*rtpg_exec_arth*() 将产生一条加法指令，它的源操作数是确定的寄存器“r8”和随机的寄存器“?”，目的操作数是一个特定的寄存器 reg，它可能已定义过，也可能会被后续的指令使用。示例的 *rtpg_exec_ldst*()则可随机产生取数指令，并根据指令的寻址模式拆分地址“address1”，生成符合要求的基地址和偏移。结果检测代码通过调用指令集模拟器计算预期结果全自动生成，测试开发人员不用关注。

3.3 嵌入式指令集模拟器

基于验证库 RTPG 的另一个关键技术是嵌入式的指令集模拟器，这是个普通的指令集模拟器。它的输入是部分测试程序 PTP，输出是处理器状态，包括所有寄存器的值和相关内存的值，脚本读入这些结果，取出关注的寄存器和内存值，输出检测指令序列。

4 随机测试程序的生成

RTPG 工具将测试程序的开发转为测试模板的开发。根据不同阶段验证目标的不同，测试模板开发的重点也有所变化。

4.1 AVP 测试程序

如上文所述，AVP 主要验证单条指令的基本功能，测试程序重点关注的是操作数的选择。

AVP 测试模板及其产生的程序如下：

```
//====File:example_avp.pl=======================
Rtpg_example_avp($instr, $data1, $data2){
%rtpg_reg=(                                          #line1
  "r8"   =>$data1,                                   #line2
  "r9"=>$data2,                                      #line3
  );
rtpg_init_sys();                                     #line4
rtpg_exec_arth("$instr r10 r8 r9");                  #line5
rtpg_chck_result(rtpg_begn_sim());                   #line6
}
```

```
Rtpg_example_avp("add",0x7fffffff, 0x00000001);

//=========File: example_avp.s================
LI32        r8, 0x7fffffff                          #line1
LI32        r9, 0x0001                              #line2

Add             r10, r8, r9                         #line3
CHECKPSW    OVERFLOW                                #line4
CHECKREG    r10, 0x80000000                         #line5
```

测试模板分为模板定义和模板调用 2 个部分，模板定义的前 4 行产生初始化的指令，然后执行待测指令(第 5 行)，最后启动仿真，并将返回结果用于产生检测运算结果和机器状态字 PSW 的指令(第 6 行)。调用时把待测指令和操作数的值传给 rtpg_example_avp()，产生相应的汇编程序，该程序测试了有溢出的加法。

在 AVP 阶段，通过显示定义待测指令操作数，验证人员可根据设计规范穷尽所有功能点，达到验证单条指令的目的。

4.2 IVP 测试程序

IVP 主要验证与微体系结构相关的处理器指令特性，重点关注指令的组合。IVP 测试模板及其产生的程序如下:

```
//=========File: example_ivp.pl=======================
 rtpg_example_ivp($load, $instr1, $instr2, $store){
  rtpg_exec_ldst("$load    reg    addr");                  #line1
  switch (rtpg_ctrl_selc()) {                             #line2
case 1 {                                                  #line3
          rtpg_exec_arth("$instr1 reg reg ?");        }
       case 2 {                                          #line 4
          rtpg_exec_arth("$instr2 reg reg ?"); }}
    rtpg_exec_ldst("$store reg    addr");                 #line 5
    rtpg_chck_result( rtpg_begn_sim() );                  #line 6
 }
 rtpg_example_ivp("LOAD", "add", "mul", "STORE");

// ================ File: example_ivp.s ===============
  SETMEM    0x4000,0x51fc    #line1: 0x4000: 0x51fc
  LI32          r8,    0x3ffc       #line2: r8:    base addr
  LI32          r20, 0x1004    #line3: r20: oprand

  load        r5,    4(r8)     #line4: r5    <- mem(0x4000)
  add          r20, r5,r20    #line5: r20 <- r5 + r20
  store       r10, 4(r8)     #line6: mem(0x4000) <- (r10)
  CHECKREG r20,0x6200         #line 7: r20: result
 CHECKMEM 0x4000,0x6200 #line 8: 0x4000: 0x6200
```

该模板是“取-算-存”指令组合的测试模板。由于操作数的值不受关注，模板没有显示定义寄存器和内存的初始值，而是由执行类库函数根据处理器特性随机选取。第 1 行~第 5 行定义了“取数-运算-存数”的指令序列，并且允许在 2 条运算指令($instr1 和$instr2)中随机选择一条执行。相比 AVP 测试模板，IVP 测试模板的指令参数化，在模板调用的时候，传递指令类型(如 LOAD 或 STORE)或某条具体的指令(如 add 或 mul)，以突出 IVP 阶段“关注指令组合”的验证目标。

5 微处理器验证结果

本文针对嵌入式微处理器开发验证库、编写测试模板并用自动生成的测试程序对处理器指令集进行功能验证。各阶段的仿真验证结果如表 1 所示。

表 1 各阶段仿真验证结果

功能	基于验证库的 RTPG		基于 Linux
	AVP	IVP	
工具开发/移植时间/天	15	15	15
测试程序开发时间/天	20	30	*N/A*
调试时间/周	4	14	8
指令条数/($\times 10^7$)	0.5	8.0	3.0
发现错误比例/(%)	31	65	4

设计是否成熟一般可通过比较各个阶段发现的设计错误体现，一个理想的验证工具应能帮助验证人员及早发现设计错误。表 1 列出 3 个验证阶段的结果比较，其中第 3 个阶段是在 FPGA 上移植了经裁减的 Linux 操作系统，并将它作为软验证的最后环节。可见，基于验证库的随机测试程序能覆盖 96%的设计错误，而工具的开发成本仅为 15 人/天。

表 2 比较了各个阶段发现错误的分类统计结果。

表 2 发现错误比较 (%)

单元	基于验证库的 RTPG	基于 Linux 的验证
浮点单元	99	1
整数运算单元	100	0
访存单元	95	5
取指、分支预测单元	91	9
译码单元	88	12
完成单元	98	2

由表 2 可见，基于验证库的 RTPG 对运算单元的覆盖比较出色。在访存方面，由于验证库地址分解算法的局限性，导致验证效果下降。对于取指和分支预测的验证，由于测试模板的指令序列是半随机而不是完全随机的，无法达到较高的覆盖。译码、完成单元遗漏的设计错误则是由于“影子寄存器”问题造成。“影子寄存器”问题主要存在于软件自测试的验证策略中，由于软件自测试只能比较程序运行的最终结果，无法验证存放中间结果的寄存器，导致对“寄存器重命名”等技术的验证上存在缺陷，最终影响了译码和完成单元的验证结果。

6 结束语

本文介绍的基于验证库的随机测试程序生成工具是对微处理器指令集验证方法的一次有益探索，它使用通用脚本语言开发的验证库和测试程序模板，针对不同验证阶段生成高质量的测试程序。实验结果表明，该工具的验证效果良好。进一步研究方向是针对目前库函数的局限性问题，主要包括全随机指令序列、访存地址分解算法和影子寄存器等。

参考文献

[1] 梁 磊. 基于约束求解的微处理器功能验证程序自动生成技术研究[D]. 长沙: 国防科技大学, 2004.

[2] Adir A. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification[J]. Design & Test of Computers, 2004, 21(2): 84-93.

[3] 朱 丹. 微处理器体系结构级测试程序自动生成关键技术研究[D]. 长沙: 国防科技大学, 2004.

[4] Utamaphehai N, Blanton R D, Shen J P. A Buffer-oriented Methodology for Microarchitecture Validation[J]. The Journal of Electronic Testing, 2000, 16(1/2): 49-65.

[5] 张 珩, 沈海华. 龙芯 2 号微处理器的功能验证[J]. 计算机研究与发展, 2006, 43(6): 974-979.

[6] 张山刚. 微处理器验证平台的实现[D]. 西安: 西北工业大学, 2005.