使用局部建模的微处理器测试程序自动生成方法

张 良,易江芳,佟 冬,程 旭,王克义
(1. 北京大学深圳研究生院,广东深圳 518055;2. 北京大学微处理器研究开发中心,北京 100871)

摘 要: 模拟仿真方法是当前微处理器功能验证的主要方法,然而在验证工作后期需要耗费大量的时间来检验余下复杂的功能点,验证收敛速度缓慢.针对该问题,本文在覆盖率增长缓慢时,引入结合模型检验引擎的测试程序生成方法.该方法首先采用局部建模策略为处理器构建抽象设计模型,然后使用模型检验引擎读入该模型并产生测试生成指导规则,最后,随机测试生成器依据指导规则产生大量测试程序作为模拟器输入,完成功能验证工作.以北大众志 UniCore32 定点处理器核的功能验证为例评估本文方法,结果表明,使用该方法可以快速完成对未覆盖功能点的验证,加速验证收敛.

关键词: 功能验证; 模型检验; 局部建模; 测试程序生成
中图分类号: TP302 **文献标识码:** A **文章编号:** 0372-2112 (2011) 07-1639-06

Test Program Generation for Microprocessor Verification Using Local Modeling Strategy

ZHANG Liang, YI Jiang-fang, TONG Dong, CHENG Xu, WANG Ke-yi
(1. *ShenZhen Graduate School, Peking University, Shenzhen, Guangdong* 518055, *China*;
2. *Microprocessor Research and Development Center of Peking University, Beijing* 100871, *China*)

Abstract: Simulation is the major technique used for processor verification. In the late of the verification process, simulation requires a lot of expert time and computer resources to verify residual complicated functional points, which slows down the verification progress. This paper introduces a test generation method based on model checking engine to address this problem. First, an abstract microprocessor model focuses on these uncovered functional points is constructed using local modeling strategy. Second, model checker reads this abstract model and automatically produces test generator directives. Finally, these directives guide random test generator to generate test programs that cover the specified functional points. Experiments on verifying PKUnity UniCore32 microprocessor demonstrated that this method spent little time to cover these uncovered coverage tasks and increased the verification efficiency.

Key words: functional verification; model checking; local modeling; test program generation

1 引言

功能验证是现代微处理器设计流程中最耗时的工作,统计显示大约 70% 的开发时间和资源花费在功能验证上[1].如果存在验证漏洞,导致设计错误流入到最终产品中,会给企业带来巨大损失.在当前的微处理器功能验证方法中,形式化验证一般用于规模不大的模块级设计验证[2].模拟仿真验证方法是微处理器功能验证的主要方法.

模拟仿真验证方法的重点是如何生成高品质测试程序[3].在工业界和学术界,开发了许多功能强大的随机测试生成系统,如 IBM 公司的基于模型的测试生成器 Genesys-pro[4],中国科学院计算技术研究所的可配置随机测试程序生成器 CRPG[5],国防科学技术大学的由体系结构描述驱动的基于约束求解的测试生成器 MA^{2}TG[6].随机测试生成系统在验证工作前期可以获得较好的验证效果,但因为无法在指导规则(Directive)和覆盖率模型之间建立确定性联系,导致在验证工作后期覆盖率的提高十分缓慢,影响验证收敛速度[7].

研究如何更有效地利用覆盖率结果指导测试生成的覆盖率驱动的测试生成方法(Coverage Directed Generation, CDG),是目前研究的重点和热点[3].如 Fine 等[8]提出的基于贝叶斯(Bayesian)网络的 CDG 方法,采用贝叶斯网络在覆盖率模型和随机测试生成器之间建立关联,可以推断生成测试激励,但该方法需要较强的专家知识来构建复杂数学模型,且本质上仍然是一种概率生成方法.Koo 等[9]为流水线处理器功能验证提出一种基于抽象模型的自上而下的方法,该方法依据设计规范在较

收稿日期:2010-05-29;修回日期:2010-09-06
基金项目:国家"863"高技术研究发展计划重点项目(No.2006AA010202)

高抽象层次对处理器进行全局建模,但为避免状态空间爆炸问题,只能抽取简单的功能点构建处理器模型,通常用于验证抽象程度较高的功能点.目前对 CDG 方法的研究更偏重于理论探索,在实际的微处理器验证工作中应用较少[10].

本文针对在验证工作后期,使用现有的测试生成方法时间成本较高的问题,在覆盖率增长缓慢时,引入结合模型检验引擎的测试生成方法(Model Checking Engine based Test Generation,MCTG).首先,从这些未覆盖功能点出发,采用局部建模策略为处理器构建抽象 SMV[11]模型;然后,模型检验引擎[12]读入 SMV 模型及未覆盖功能点的形式化描述,并自动产生测试生成指导规则;最后,随机测试生成器依据这些指导规则产生大量测试程序,在模拟仿真环境中模拟执行这些测试程序就可以完成对未覆盖功能点的检验.在北大众志 UniCore32 定点处理器核的功能验证中实际应用的结果表明,使用本文方法可以在较短时间内完成功能验证任务,加速验证收敛.

和本文方法相似的是文献[13,14]中的方法,主要的不同在于:文献[13,14]是面向整个处理器构建全局性的抽象设计模型,并且直接从该抽象模型提取覆盖率模型,如状态覆盖[14],状态迁移覆盖[14]和路径覆盖[13],这导致覆盖率模型定义的越全面、越精细,则要求抽象设计模型越复杂,在覆盖率模型的全面性和缩小抽象设计模型的状态空间之间存在矛盾.而本文是在已经使用随机测试程序生成系统检验了大部分功能点之后才引入 MCTG 方法,此时构建的抽象设计模型只要能满足对少量未覆盖功能点的验证需求即可,因此,很大程度上减小了抽象处理器模型的规模,也有条件对设计细节进行建模而不会遭遇状态空间爆炸问题.

2 现有的模拟仿真验证方法的问题

现有的微处理器模拟仿真验证平台普遍使用随机测试生成器来产生测试程序(如图 1 所示).通过覆盖分析来设定指导规则,进而引导随机测试生成器产生新的测试程序.

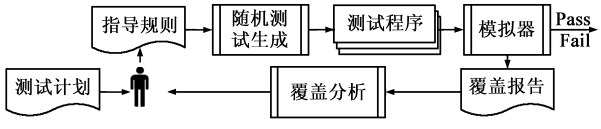

图1 现有的模拟仿真验证平台基本架构

使用该模拟仿真验证平台对 UniCore32 数据相关处理机制进行功能验证,在最初的 55 个小时,覆盖率迅速达到 80.65%,但此后覆盖率增长缓慢,需要再花费 34 个小时才能提高到 84.68%,且很难进一步提高.因此,在验证工作后期如果继续使用该验证平台会很难完成全部功能验证任务,该问题普遍存在于其它商业处理器功能验证中[7].

分析这些未覆盖到功能点,均包含复杂的时序需求和复杂的场景,比如其中一个功能点定义的场景可以描述为:"在周期 T,寄存器读口 A 的操作数来自于数据旁路 2;在周期 $T+1$,寄存器读口 A 的操作数还是来自于数据旁路 2,并且此时 EXE1 级指令引起流水线互锁".能触发该场景的测试程序在模拟执行时,需要在两个时钟周期内出现三次特定的数据相关,并且 UniCore32设计中 EXE1 级指令引起互锁的触发条件很苛刻,所以很难使用随机测试生成器完成该验证任务.

3 本文方法介绍

本文方法的工作流程如图 2 所示.首先使用传统的模拟仿真验证平台来进行功能验证;如果覆盖率达到 100%(从第 2 节可知,很难实现该目标),则验证工作结束,否则判断是否覆盖率增长缓慢(具体的判断标准由验证工程师来确定,如依据覆盖率曲线的斜率,或者在规定时间内覆盖率没有增长);如果覆盖率增长缓慢,则针对未覆盖功能点使用 MCTG 方法来完成后续验证工作.

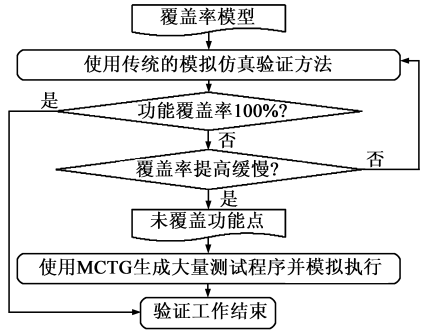

图2 引入MCTG后的处理器验证工作流程

3.1 MCTG 方法工作流程

MCTG 方法的工作流程如图 3 所示,步骤如下:

Step1 针对未覆盖功能点,使用局部建模策略为处理器构建抽象 SMV 模型,记做模型 M;同时为每个未覆盖功能点分别进行形式化描述,构建性质(Property)列表(记为性质集 P).

Step2 选择一个未检验的性质 $p(p\in P)$,调用模型检验引擎搜索模型 M 的状态空间,找到一条使性质 p 成立的状态迁移路径 t.

Step3 分析路径 t,重点分析其中涉及到指令属性及指令间约束的状态变量,生成能被随机测试生成器接受的指导规则,随机测试生成器读取该指导规则并生成测试程序集.

Step4 使用模拟仿真验证平台执行这些测试程序,完成对性质 p 所对应的功能覆盖点的验证.

Step5　重复 Step2 到 Step5,直到完成对所有性质的检验,验证工作结束.

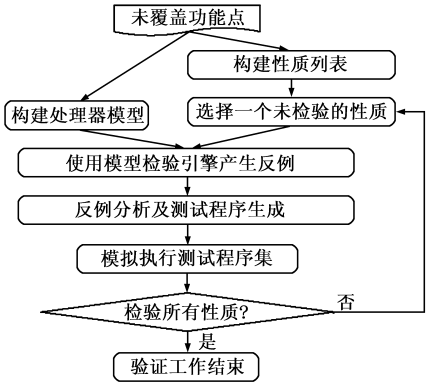

图3　MCTG方法的工作流程

从上述工作流程可以看出,MCTG 方法的主要特点包括:(1)针对未覆盖功能点,使用局部建模策略构建处理器模型,可以显著减小处理器模型的规模.(2)模型检验引擎的主要作用是产生测试生成指导规则,而不是进行完备的模型检验.

3.2　构建处理器模型

基于模型检验引擎的测试程序生成方法,需要一个使用形式化语言描述的抽象设计模型,在模型构建过程中主要面临两个问题[7]:状态空间爆炸和反例分析复杂.现有的方法为整个处理器构建抽象设计模型,这很难避免状态空间爆炸问题;通过提高模型抽象程度的手段可以降低模型复杂度,但会增加反例分析的难度.

本文针对少量未覆盖功能点,使用局部建模策略构建处理器模型,模型只要能满足对未覆盖功能点的验证需求即可,这可以显著缩小处理器模型的规模,是模型检验引擎能够成功应用的关键.

下面详细介绍局部建模策略,为便于理解,结合 UniCore32 处理器数据相关处理机制的功能验证来举例说明(详细的设计结构见第 4 节).局部建模策略包含局部模型抽取和模型精简两个步骤:

(1)局部模型抽取.该过程从 RTL(Register Transfer Level,寄存器传输级)设计代码中抽取与待验证功能点相关联的设计代码,并转化为 SMV 语言描述的模型.首先,利用 RTL 设计的模块划分,找出与覆盖率模型直接相关的设计模块;然后,从这些模块中抽取和待验证覆盖点相关联的设计代码;接着,将影响已抽取代码的其它相关代码扩充进来.上述扩充过程一直持续下去,直到完成对所有 RTL 设计代码的分析,完成局部模型抽取工作.最后,将抽取得到的局部模型转化为 SMV 语言描述的抽象设计模型.

例 1:UniCore32 设计中,数据相关的判断和处理在 bypass_interlock模块中完成.局部模型的抽取就是以 bypass_interlock模块为出发点,首先移除该模块中和待验证功能点无关的设计代码及模块接口信号;然后以这个精简的bypass_interlock模块为基础,将影响该模块的其它模块逐步扩充进来,在扩充过程中只保留对已抽取代码有真正影响的代码.上述扩充过程一直持续下去,直到完成对所有 RTL 设计代码的分析.

(2)模型精简.虽然局部模型抽取很大程度上减小了处理器模型的规模,但仍然会超出模型检验引擎的处理能力(见 4.2 节).MCTG 中模型检验引擎的主要作用是产生用于引导随机测试生成的指导规则,而不是进行完备的模型检验(此时需要保证抽象模型的完整性),因此可以在不影响验证目标的前提下对模型做适当精简.下面介绍 4 条模型精简的指导原则:

时序精简:精简掉不影响待验证功能点的流水级;而对于未精简掉的流水级,则可以对逻辑设计做适当化简.

例 2:假设取指过程正确,则可以精简掉 IF1,IF2 流水级,取出的指令直接进入译码(DEC)阶段;另外,不考虑运算结果及其对寄存器的实际修改,则可以精简掉 WB 流水级.

例 3:如果保留 DEC 流水级,则可以利用译码得到的指令类型信息,更容易地完成反例分析,降低反例分析的难度.同时对译码逻辑做适当化简,如降低指令译码细节,可以降低模型规模.

数据类型拆分:依据设计规范,将模型中的数据类型做更细粒度的划分,并拆分为多个小的数据类型.这样可以对该数据类型做更灵活的处理,比如删除、合并、精简等.

例 4:在局部模型中,指令寄存器使用 32 位二进制类型的变量 *Ir_reg* 来表示.数据类型拆分策略依据设计规范中的指令编码定义,将 *Ir_reg* 变量拆分为 *Rm*, *Rs*, *Rd*, *Rn*, *Opcode* 等多个变量.

数据类型精简:将取值范围较大的数据类型,缩小其取值范围或直接抽象为包含少量数据元素的枚举类型变量,则可以减小模型的状态空间.

例 5:在例 4 中拆分得到的 Rm 等变量,取值范围是 0~31,使用数据类型精简策略将取值范围缩小到 28~31,则可以显著减小模型状态空间.

控制状态机精简:对微处理器设计规范中定义的控制状态机做适当精简.

例 6:UniCore32 处理器在第一次进入 NORMAL 状态时所执行的指令是首条指令,在该状态之前,还包括 5 个连续的 RESET 状态.控制状态机精简策略去掉其中的 4 个 RESET 状态,在一次 RESET 之后,即开始执行首条指令.

通过前面对局部建模策略的详细介绍可以看出,该策略使用如下手段来应对使用模型检验引擎时所遇到的问题:(1)针对状态空间爆炸问题,局部建模策略构建的处理器模型只要能满足对未覆盖功能点的验证需求即可,可以显著减小处理器模型的规模.(2)针对反例分析复杂的问题,因为使用局部建模策略最终可以得到规模很小的处理器模型,所以在建模时有条件保留一些有利于反例分析的逻辑设计,降低反例分析的难度.(3)另外,在功能验证工作后期才开始局部建模工作,此时 RTL 设计已经基本成型并且不会有太大改动,在一定程度上保证了 SMV 模型的稳定性.

3.3 构建性质列表

规范或规范的子集称为性质[1],本文采用计算树逻辑[12](Computation Tree Logic,CTL)进行性质的形式化描述.一个 CTL 公式由两部分组成:(1)路径量词,包括 A(Always,对所有的路径)和 E(Exists,存在一个路径);(2)时态运算符,包括 G(Global),F(Future),X(neXt)和 U(Until).在 CTL 公式中,时态运算符前必须有路径量词.本文中,将所有未覆盖功能点均使用 CTL 公式描述后,即完成性质列表的构建工作.

模型检验引擎读入的性质 p' 是对性质 $p(p\in P)$ 取反(使用符号"~"来表示)并添加路径量词 A 和时态运算符 G 的结果,即:

$$p' = AG(\sim p) \tag{1}$$

在第 2 节中举例说明的未覆盖功能点,其对应的性质 p_1 的 CTL 公式为:

$$p_1 = a\_bp\_2 \ \&\ X(a\_bp\_2 \ \&\ exe1\_il) \tag{2}$$

其中,a_bp_2 和 $exe1\_il$ 是处理器模型中定义的两个变量,a_bp_2 变量为 1 时,表示寄存器读口 A 的操作数来自于数据旁路 2;$exe1\_il$ 变量为 1 时,表示在 EXE1 级发生流水线互锁.根据式(1)和(2),得到性质 p_1' 的 CTL 公式为:

$$p_1' = AG\sim(a\_bp\_2 \ \&\ X(a\_bp\_2 \ \&\ exe1\_il))$$

3.4 使用模型检验引擎产生反例

在处理器模型 M 和性质 p' 准备就绪后,模型检验引擎读入 M 和 p',并在模型 M 的状态空间中搜索是否存在一条使得性质 p' 不成立的状态迁移路径.模型检验引擎的运行结果可能有三种:

(1)在给定时间内模型检验引擎不能给出检测结果.该运行结果表明处理器模型的状态空间过大,需要对处理器模型做进一步精简.

(2)模型检验的结果为 TRUE.即:

$$p' = AG(\sim p) = \text{TRUE}$$

表示在模型 M 中性质($\sim p$)永远成立,即不存在满足性质 p 的状态迁移路径.此时需要转到处理器模型测试步骤以检查模型构建是否正确,也需要检查 RTL 设计是否没有正确实现性质 p 所描述的功能点.

(3)模型检验的结果为 FALSE,并输出一个反例.反例给出的是一条状态迁移路径 t,沿着 t 可以到达使性质 p' 不成立的状态.即:

$$p' = AG(\sim p) = \text{FALSE} \tag{3}$$

根据 $AG(p) = \sim EF(\sim p)$[12]和式(3),推出:

$$EF(p) = \text{TRUE} \tag{4}$$

从式(4)可以看出,反例给出的是一条状态迁移路径 t,沿着 t 可以到达使性质 p 成立的处理器状态.以 3.3 节中的性质 p_1 为例,表 1 是模型检验引擎输出的反例文件的一部分,该文件给出了在每个时钟周期每个变量的精确值.

表 1 反例文件的一部分

变量名	时钟周期				
	1	2	3	4	5
ir_kind	LMUL	LDR	MUL	STR	D_IMM
exe1_v	0	1	1	1	1
mul_a	1	0	0	0	0
ls_p	0	1	0	0	0
exe1_il	0	0	0	0	1
a_bp_2	0	0	0	1	1

3.5 反例分析及测试程序生成

从 3.4 节可知,沿着反例给出的状态迁移路径 t 可以到达使性质 p 成立的处理器状态.所以,如果将状态迁移路径 t 映射到一个汇编测试程序,则在 RTL 模拟仿真环境中执行该测试程序,就可以检验性质 p 的正确性[14].该映射工作是由反例分析及测试程序生成来完成.

反例分析通过分析反例文件,得到指令序列的抽象描述.该抽象描述是对指令序列及指令间约束的通用的、抽象的表达,给出了测试程序中每条指令的属性及指令间约束.通过分析反例文件中每个周期的变量值,如指令类型变量,控制状态机的状态变量,各流水级的有效状态等完成反例分析.图 4 是分析表 1 的反例文件后得到的抽象指令序列,共包含 5 条指令,编号从 1 到 5.在最后一行给出了指令间的约束关系,从该约束关系可以看出指令间存在的数据相关情况.

1: instr_type = *LMUL*, *mul_a* = 1
2: instr_type = *LDR*, *ls_p* = 1, *ls_w* = 0
3: instr_type = = *MUL*, *mul_a* = 0
4: instr_type = *STR*, *ls_p* = 0, *ls_w* = 0
5: instr_type = *D_IMM*

inter_constrain: 4. *Rn* = 1. *Rd* 4. *Rd* = 3. *Rd* 5. *Rn* = 2. *Rd*

图 4 指令序列的抽象描述

在得到抽象指令序列后,可以很容易地针对选用

的随机测试生成器,将抽象指令序列转化为能被该测试生成器接受的指导规则,并生成大量汇编测试程序.图5是随机测试生成器自动生成的一个测试程序片段.

```
MULSLA   R12, R1, R2, R3
LDW      R11, [R4]
MUL      R10, R5, R6
STW      R10, [R12 + ], # 27
ADD      R7, R11, # 34
```

图5　汇编测试程序片段

3.6　处理器模型的测试

模型测试是测试模型行为是否符合RTL设计行为的过程.本文中,处理器模型可以看作是RTL设计的一个缩小规模的形式化模型,只有确保处理器模型行为符合RTL设计行为,才能保证依据该处理器模型针对某个功能点产生的测试程序,在模拟仿真环境中执行时可以真正检验RTL设计是否实现该功能点.

模型测试的过程是:在模拟执行由MCTG方法产生的测试程序时,输出和反例中的状态变量相对应的信号变量的值,通过比较这些信号变量的值与反例中变量的值是否一致,来测试处理器模型是否符合RTL设计的行为.如果发现不一致,则要检查处理器模型的正确性.处理器模型的测试要贯穿验证工作始终.

4　实验及结果分析

本文以北大众志UniCore32定点处理器核数据相关处理机制的功能验证为实验用例.UniCore32是32位单发射RISC处理器,流水线结构包括八个流水级(IF1、IF2、DEC、ISS、EXE1、EXE2、EXE3和WB),数据通路中包含3条数据旁路(Bypassing)路径,如图6所示.

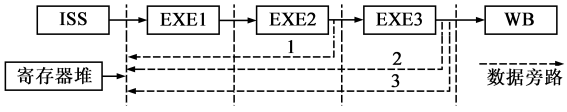

图6　UniCore32的数据旁路路径示意图

表2给出了功能覆盖率模型测试场景的一部分,全部场景共包含248个功能点.

表2　功能覆盖率模型的测试场景

分类号	测试场景
1	寄存器堆的每个读端口,从各个前递路径读取操作数
2	寄存器堆的每个读端口,引起互锁
3	执行级指令引起互锁,同时发射级指令有数据相关
4	C标志位引发的数据相关

引入MCTG方法后的模拟仿真验证平台如图7所示,虚线框部分是本文方法与传统方法(见图1)不同的地方.本文使用内部开发的UniRTG随机测试生成器,并采用Cadence SMV[11]模型检验引擎作为形式化模型检验工具.实验硬件环境为配置双AMD Opteron 1.8GHz处理器和8G物理内存的工作站,操作系统为Red Hat企业版Linux2.4.

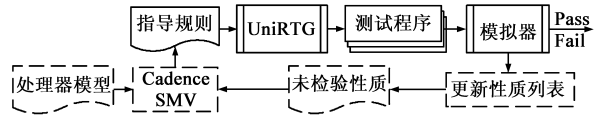

图7　本文的模拟仿真验证平台基本架构

4.1　和传统方法的比较

图8是本文方法和仅使用UniRTG的传统方法在覆盖率进程上的比较.结果显示,传统方法可以快速达到80.65%的覆盖率,但随后覆盖率增长很慢,且达到84.68%后无法继续提高.本文在覆盖率达到80.65%时引入MCTG方法,模型构建需要花费约56个小时,接着在10小时内完成测试程序生成和模拟执行工作.MCTG方法共生成735个汇编测试程序,对未覆盖功能点进行了充分的验证.

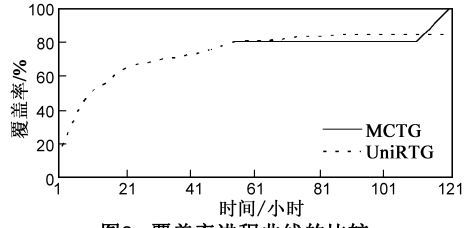

图8　覆盖率进程曲线的比较

传统方法中,对于上述很难使用自动化方法覆盖到的功能点,通常采用人工分析并手写汇编测试程序的方法来解决,但这些未覆盖功能点比较复杂,因此需要耗费大量的时间和人力成本[7].随着现代微处理器设计中逐步引入更复杂的设计结构,人工分析的难度会逐渐增大,无法满足功能验证的需求.另外,MCTG方法可以为每个功能点产生大量满足条件的测试程序,对功能点的验证更充分,而人工分析的方法由于时间成本较高,只能对每个功能点书写少量测试程序.

4.2　局部建模策略的优化效果

局部建模策略的第一步是针对未覆盖功能点抽取出局部处理器模型(M_{init}),虽然此时M_{init}的状态空间已经很小,但仍然会超出模型检验引擎的处理能力,需要做模型精简后才能使用.表3给出了依次使用各模型精简指导原则后,处理器模型的状态变量数、模型检验时的BDD(Binary Decision Diagram)节点数以及模型检验引擎的运行时间("终止",是指在模型检验引擎工作时,物理内存的消耗接近机器的物理内存容量,进程被终止).模型精简指导原则的使用顺序是时序精简(M_1),数据类型拆分(M_2),数据类型精简(M_3)和控制状态精简(M_4).

可以看出,局部模型M_{init}包含161个状态变量(此时,模型的状态数是2^{161}),这超出了模型检验引擎的处理能力,使用全部模型精简指导原则可以精简掉超过

一半的状态变量(共 85 个).从表 3 中还可以看出,数据类型精简指导原则的优化效果最好,可以精简掉 55 个状态变量.

表 3 各模型精简指导原则的优化效果比较

	M_{init}	M_1	M_2	M_3	M_4
状态变量数	161	147	135	80	76
BDD 节点数(10^6)	> 500	> 500	> 500	33.24	8.14
时间(s)	终止	终止	终止	389	88

5 结论

在微处理器功能验证工作后期,现有的自动化验证方法的时间成本较高,人工分析和手写汇编测试程序需要耗费大量的时间和人力成本.针对该问题,本文提出在验证工作后期,使用结合模型检验引擎的 MCTG 方法来自动生成测试程序,并初步实现了该方法的原型系统,成功应用于自主设计的 UniCore32 定点处理器核的实际验证工作中.今后的研究工作将进一步完善该原型系统,使构建处理器模型的过程完全自动化,这可以极大提高本文方法的实用性.另外,进一步完善该系统,将本文方法更广泛地应用于其它实际微处理器的功能验证工作中.

参考文献

[1] WK Lam. Hardware Design Verification. Simulation and Formal Method-Based Approaches [M]. Indiana: Prentice Hall PTR, 2005.

[2] 冯毅,易江芳,刘丹,佟冬,程旭.面向 SoC 系统芯片中跨时钟域设计的模型检验方法[J].电子学报,2008,36(5):886 – 892.
Feng Yi, Yi Jiang-fang, Liu Dan, Tong Dong, Cheng Xu. Model checking on clock domain crossing design of system-on-chip [J]. Acta Electronica Sinica, 2008, 36(5): 886 – 892. (in Chinese)

[3] 沈海华,卫文丽,陈云霁.覆盖率驱动的随机测试生成技术综述[J].计算机辅助设计与图形学学报,2009,21(4):419 – 431.
Shen Hai-hua, Wei Wen-li, Chen Yun-ji. A survey on coverage directed generation technology [J]. Journal of Computer-Aided Design & Computer Graphics, 2009, 21(4): 419 – 431.

[4] Adir A, Almog E, Fournier L, et al. Genesys-pro: innovations in test program generation for functional processor verification [J]. IEEE Design & Test, 2004, 21(2): 84 – 93.

[5] H Shen, L Ma, H Zhang. CRPG: a configurable random test-program generator for microprocessors [A]. Proceedings of IEEE International Symposium on Circuits and Systems [C]. Kobe, 2005. 4171 – 4174.

[6] 朱丹,李暾,郭阳,李思昆.微处理器体系结构级测试程序自动生成技术[J].软件学报,2005,16(12):2172 – 2180.
Zhu Dan, Li Tun, Guo Yang, Li Si-kun. Microprocessor architectural automatic test program generation [J]. Journal of Software, 2005, 16(12): 2172 – 2180. (in Chinese)

[7] A Adir, E Bin, O Peled, A Ziv. Piparazzi: a test program generator for micro-architecture flow verification [A]. Proceedings of the Eighth IEEE International Workshop on High-Level Design Validation and Test Workshop [C]. Washington: IEEE Computer Society, 2003. 23 – 28.

[8] S Fine, A Ziv. Coverage directed test generation for functional verification using Bayesian networks [A]. Proceedings of the 40th Conference on Design Automation, Anaheim [C]. New York: ACM, 2003. 286 – 291.

[9] HM Koo, P Mishra. Functional test generation using property decompositions for validation of pipelined processors [A] Proceedings of the conference on Design, automation and test in Europe [C]. Leuven: European Design and Automation Association, 2006. 1240 – 1245.

[10] A Gluska. Coverage-oriented verification of Banias [A]. Proceedings of Design Automation Conference [C]. New York: ACM, 2003. 280 – 285.

[11] KL McMillan. Getting started with SMV [DB/OL]. http://www.kenmcmil.com/, 1999-03-23/2007-08-07.

[12] 林惠民,张文辉.模型检测:理论、方法与应用[J].电子学报,2002,30(12A):1907 – 1912.
Lin Hui-min, Zhang Wen-hui. Model checking: theories, techniques and applications [J]. Acta Electronica Sinica, 2002, 30(12A): 1907 – 1912. (in Chinese)

[13] J Shen, JA Abraham. An RTL abstraction technique for processor microarchitecture validation and test [J]. Journal of Electronic Testing, Theory and Applications, 2000, 16(1): 67 – 81.

[14] S Ur, Y Yadin. Micro architecture coverage directed generation of test programs [A]. Proceedings of the 36th conference on Design automation [C]. New York: ACM, 1999. 175 – 180.

作者简介

张 良 男,1980 年生于辽宁丹东,北京大学计算机系博士研究生.主要研究方向为计算机系统结构,微处理器设计与验证.
E-mail: zhangliang@mprc.pku.edu.cn

易江芳(通讯作者) 女,1977 年生于四川什邡,北京大学计算机系博士.主要研究方向为软硬件协同设计、芯片验证和测试自动生成. E-mail: yijiangfang@mprc.pku.edu.cn

佟 冬 男,1971 年生于吉林长春,北京大学计算机系副教授.主要研究方向为高性能微处理器、系统芯片、体系结构等.