文章编号:1001 - 2486(2011)04 - 0080 - 06

# 一种基于核心数据区监视的交叉验证方法*

刘 胜[1],杨焕荣[1],陈书明[1],李 勇[1],侯 申[2]
(1. 国防科技大学 计算机学院,湖南 长沙 410073;
2. 西安通信学院 电子工程系,陕西 西安 710106)

**摘 要**:通常商用体系结构模拟器不开源,仿真过程中无法直接获取中间结果。提出了一种基于核心数据区监视的交叉验证方法,可将体系结构模拟器运行过程中核心数据区的变化情况提取出来,用以支持该模拟器与寄存器传输级代码的模拟结果进行交叉比对,快速精确定位两者的执行差异,提高验证人员的查错效率。实验分析和实际应用表明,本方法比传统方法可将查错效率提高一个数量级以上。

**关键词**:核心数据区;监视;验证;查错
**中图分类号**:TP302.2 **文献标识码**:A

# A Cross-verification Method Based on the Key Data Region Monitoring

*LIU Sheng*[1], *YANG Han-rong*[1], *CHEN Shu-ming*[1], *LI Yong*[1], *HOU Shen*[2]
(1. College of Computer, National Univ. of Defense Technology, Changsha 410073, China;
2. Department of Electronic Engineering, School of Xi'an Communications, Xi'an 71006, China)

**Abstract**: Usually the intermediate results can hardly be obtained in the simulation process from the business architecture (ARC) simulator since its source is non-open. A method based on key data region monitoring was proposed to distill the value in the key region of ARC simulator automatically when it is running. In this way, the results from ARC simulator and RTL simulator can be cross-checked to help the verification engineer debugging the RTL codes quickly and efficiently. Compared with the traditional method, it can speed up the error checking by one order of magnitude according to the experiment and real chip design.

**Key words**: key data region; verification; error checking

验证是芯片开发过程中费时费力,又必不可少的一个重要阶段。根据国际半导体技术路线(ITRS)显示[1],验证阶段所花费的时间一般占整个项目总时间开销的 70%以上。受上市时间的约束,EDA 厂商和验证工程师长期以来一直致力于建立更为高效、快速的验证方法。

处理器设计过程一般采用高级语言(C/C++/System C)等进行规范描述并设计 ARC 模拟器,然后进行 RTL 开发,验证平台是在此过程中统一开发的。但也存在使用不开源的模型作为 ARC 模拟器进行开发的例子,如兼容性设计、对某些处理器级的 IP 核进行验证等。在这些情况下 ARC 模拟器对于验证人员来说是个"黑盒子",验证人员只能通过它得到最终的执行结果,而无法通过修改它构建与 RTL 模拟器交互信息的验证环境。

大型系统设计中,验证和设计人员将面临以下三个方面的压力:(1)RTL 代码模拟数据量大,执行过程复杂。大型 Benchmark 算法运行一般在几百万甚至几亿个 CPU 执行周期以上,波形文件达数十 GB 以上,在如此大的执行轨迹中寻找到出错点比较困难;(2)验证人员对大型的 Benchmark 算法本身不熟悉或者不完全精通,不能依靠算法本身的流程和执行轨迹来查找错误;(3)ARC 模拟器不一定是由自身开发的,验证人员只能通过 ARC 模拟器得到 Benchmark 最终的执行结果和有限的 profile 信息,而不能通过修改或优化 ARC 模拟器,方便地提取所需要的其他执行信息。

本文提出的核心数据区监视的自动化验证方法,能够有效的解决 ARC 模拟器不开源时验证效率低下的问题,加速验证工作的收敛。

* **收稿日期**:2010 - 11 - 08
**基金项目**:"核高基"重大专门项目(2009ZX01034 - 001 - 001 - 006);国家自然科学基金项目(61070036);国家 863 计划基金项目(2009AA011704)
**作者简介**:刘胜(1984—),男,博士生。

## 1　相关研究工作

文献[2]提出范围压缩(scope reduction)的验证方法,范围压缩包括电路压缩(circuit reduction)和验证用例压缩(test case reduction)两种方法,这两种方法也是传统验证中主要采用的方法。

电路压缩的主要思想是暂时去除或取代某些不可能造成错误结果的电路,以加速查错和模拟的时间。验证用例压缩方法通过对验证程序问题规模进行缩减,使错误在较短的时间内重现,方便验证和设计人员进行错误的排查。

上述电路压缩或验证用例压缩方法在某些情况下确实能够帮助验证人员尽快找出错误的根源,但也存在各自的局限性。电路压缩方法中通常不能安全地排除电路单元,因为各个部件之间是相互影响的,某些情况下难以准确压缩电路;验证用例压缩需要验证和设计人员对验证用例非常熟悉,而通常情况下,验证用例采用的是通用的标准 Benchmark,算法复杂多样,熟悉并压缩的工作量很大,并且某些错误只有在某些特定的节拍下才能够暴露,缩小验证用例的规模并不一定能重现错误。

VCS™平台为被验证模块提供了参考模型的协同模拟技术[3]。参考模型和 RTL 模型之间通过一组应用程序编程接口进行通信,协同模拟技术在目前的芯片设计中应用相当广泛[4-6],但这种方法需要验证和设计平台统一开发,不适合 ARC 模拟器不开源的情形。

本文提出的验证方法能够协助验证和设计人员确定错误发生在某一个写核心数据区操作之后(不超过几十个周期),加速开发人员的查错过程,同时不需要验证和设计人员压缩电路或 Benchmark 的算法规模,基本能够在 ARC 模拟器不开源的情况下达到与协同模拟技术[3-6]相同的查错效率。

## 2　核心数据区监视的自动化验证方法

### 2.1　基本定义

KDR(Key Data Region):核心数据区。它是处理器中对 Benchmark 运行正确与否最为敏感的空间。如通用寄存器的内容变化能够迅速反映 CPU 核中的计算单元、控制单元、存储通路是否正确运行,通用寄存器就是 CPU 核的核心数据区域。同理,输入输出缓冲区的内容是外设单元的核心数据区。将 KDR 中数据的个数记为 M,将 KDR 中第 i 个数据的内容记为 KDR _ i。

SP(Sample Point):采样点,在 ARC 模拟时每隔一段时间,记录 KDR 的值。其中每一个记录的时刻记为采样点。

SF(Sample Frequency):采样频率,ARC 模拟时记录的 KDR 频率。

SN(Sample Number):采样点个数,Benchmark 在 ARC 模拟时 KDR 被记录的次数。

State(SP _ i):采样点状态,ARC 模拟器在采样点 SP _ i 时所有 KDR 中数据值的集合。即 State(SP _ i) = {KDR _ 1, KDR _ 2, … ,KDR _ M}。

$T_{ARCSim}$:某个 Benchmark 在 ARC 模拟器模拟所花费的时间。

$T_{RTLSim}$:某个 Benchmark 在 RTL 模拟器模拟所花费的时间。

$T_{ARCInf}$:采集 ARC 模拟信息所花费的额外的时间。

$T_{RTLInf}$:采集 RTL 模拟信息所花费的额外的时间。

$N_{WKDR}$:某个 Benchmark 在执行过程中写 KDR 操作的次数。

### 2.2　主要流程

图 1 是传统的验证方法的流程图。由于 ARC 模拟器并不开源,传统的验证方法将某个 Benchmark 首先在 ARC 模拟器上模拟完毕,再在 RTL 模拟环境中进行模拟,然后将两者的最终执行结果进行对比。若两者的执行结果相同,则认为 RTL 代码通过了该 Benchmark,可以进行下一个 Benchmark 的验证。若两者的执行结果不同,则认

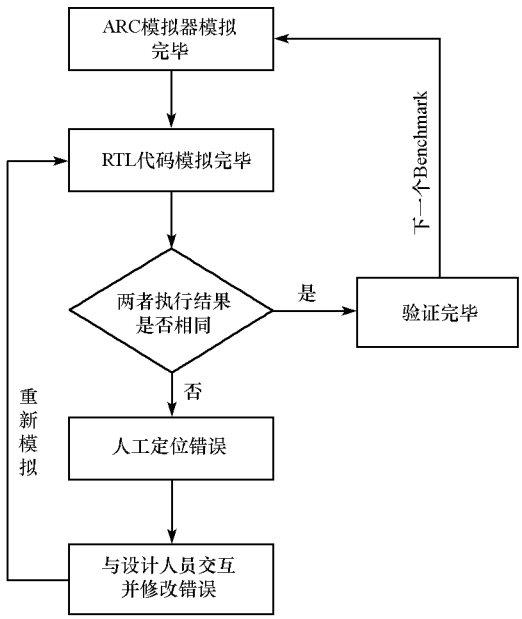


图 1　传统方法的流程图

Fig.1　The flow chart of the traditional method

为 RTL 代码存在错误,需要验证人员从错误的执行结果当中分析 RTL 代码中存在的问题,验证人员受到 RTL 模拟数据量大,执行过程复杂,对大型 Benchmark 算法本身不熟悉等方面问题的困扰,很难高效地查找 RTL 代码中的错误根源。

传统的验证方法存在两个主要的问题,一是必须等到 RTL 模拟器模拟完毕以后才能进行执行结果比对,而 RTL 代码的模拟过程是一个非常漫长的过程,很可能在 RTL 代码开始模拟很短一段时间就已经出错了,这时剩下的模拟是没有意义的。传统验证方法第二个问题是需要验证人员人工进行所有的分析工作,验证人员要在大量的执行节拍中找出出错点,既需要对 Benchmark 本身的精通,也需要靠一定的运气。

图 2 是本文提出方法的验证流程图。首先将某个 Benchmark 在 ARC 模拟器上进行模拟,在模拟的同时监视 ARC 模拟器 KDR 映射的内存的变化,并对其进行采样,生成 ARC 模拟器执行信息。接下来在 RTL 代码进行模拟的时候,通过保存出来的 RTL 执行信息和 ARC 模拟器执行信息进行初级检索。一旦初级检索发现不匹配,立即暂停 RTL 代码的模拟,并通过生成精确同步点、高级检索这两个自动运行的过程,准确到找到 RTL 代码模拟出错的节拍。这样一方面可以在 RTL 代码模拟出现错误之后就暂停其模拟,节省验证时间,另一方面传统方法中依赖验证人员进行人工分析的绝大部分工作都由计算机来完成,大大减轻了验证人员的工作负担,提高了验证效率。

### 2.3 详细步骤

如图 2 所示,本文提出的验证方法将 RTL 模拟器和 ARC 模拟器之间的协调工作分为五个步骤:(1)ARC 模拟器执行信息采集;(2)RTL 模拟器执行信息的采集;(3)初级检索;(4)精确同步点生成;(5)高级检索。下面详细介绍这些步骤和主要的实现方式。

步骤 1 ARC 模拟器执行信息采集

由于本文考虑的情况是 ARC 模拟器并不是自己开发的,设计和验证人员并没有 ARC 模拟器的源代码,因而不能通过修改模拟器源代码来生成一些必要的执行信息。本文采用了内存监视的方法对 ARC 模拟器执行时 KDR 的变化情况进行了采样。主要分为定位 KDR 和采集 State(SP _ i)两个过程。

定位 KDR 在内存中的位置主要基于以下原理:利用计算机程序很容易搜索到一个常量的在内存中的地址,却不容易搜索一个时刻在变化的

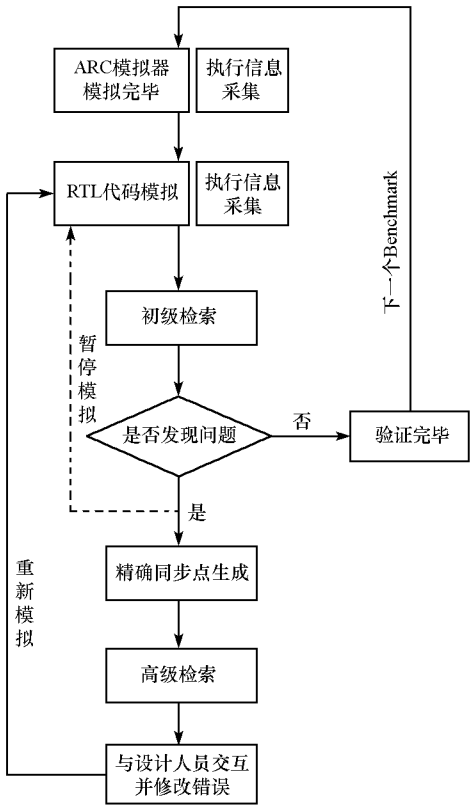


图 2 本文提出方法的流程图

Fig.2 The flow chart of our proposed method

数值在内存中的地址,若一个常量 A 和一个变量 B 在内存中的地址差值(偏移)总是保持不变的话,那么可以通过先查找常量 A,再将其地址与偏移相加,间接地查找到变量 B。定位 KDR 在内存中位置的具体方法如下:(1)编写一个汇编程序对 KDR _ 1 进行写操作,如写入 0x12345678,使用内存查看工具(如 Cheat Engine[7]、Tsearch[8] 等)观察内存,找到 0x12345678 所在的位置,一般情况下,将会有 3 ~ 5 个内存地址的内容为 0x12345678,将这些地址记录下来,记为集合{addr _ A},再对 KDR _ 1 写入另外一个不同的数据,如 0x11112222,使用内存查看工具观察内存,找到 0x11112222 所在的位置,将这些地址也记录下来,记为集合{add _ B},将{addr _ A}和{addr _ B}进行与操作,结果赋给{addr _ A}。重复以上过程直到{addr _ A}中的元素个数为 1,从而得到唯一的一个地址即为 KDR _ 1 在内存中的位置,记为 addr1。(2)使用内存查看工具寻找该 ARC 模拟器的某个字符或数字常量 Constant(如 ARC 模拟器窗口名称的 ACSII 编码)在内存中的位置,记为 addr0。常量 Constant 的选择必须满足的一个条件:它在内存中的地址 addr0 在 ARC 模拟器执行期间与 addr1 的偏移量(offset = addr0 - addr1)保持不变,若不满足该条件就使用其他常量。(3)经过前面

两步,我们得到合适的常量 Constant 以及它在内存中映射地址与 KDR _ 1 在内存中映射地址的偏移量 offset。定位 KDR 在内存中的地址的过程与以上过程相反,即先搜索常量 onstant 的内存映射地址 x,那么 KDR _ 1 在内存中的地址即为 y = x - offset, 这样就实现了通过查找常量 Constant 的方法来找到 KDR _ 1 在内存中的地址, 这里得到的地址 y 将在采集 State(SP _ i)阶段使用。

采集 State(SP _ i),主要是编写一个采样进程,在 ARC 模拟器运行的同时,对 KDR 在内存中映射的内容进行采样,然后保存到文件中。由于 ARC 模拟器中 KDR 一般会被定义为与数组类似的结构(如数组和结构体),并且运行时它们的值在主机内存中是连续存放的,因而,若 KDR _ 1 的地址为 y,则 KDR _ 2 的地址为 y + 4(假设每个 KDR 的长度为 4 个字节),依次类推可得 KDR _ 3, …,KDR _ M 的地址。采样进程的核心是调用内存读取函数 ReadProcessMemory( ),每隔一段时间读取以地址 y 开始的一段内存空间(从 y 到 y + 4 * M)。这样达到的效果是在 SP _ i 记录了 KDR 的 State(SP _ i)。

上述过程的示意如图 3 所示,通过这一过程我们得到了不同 SP 时 KDR 的 State(SP _ i),同时也得到了不同 SP 时 CPU 的 PC 值,这些信息将在初级检索阶段使用。

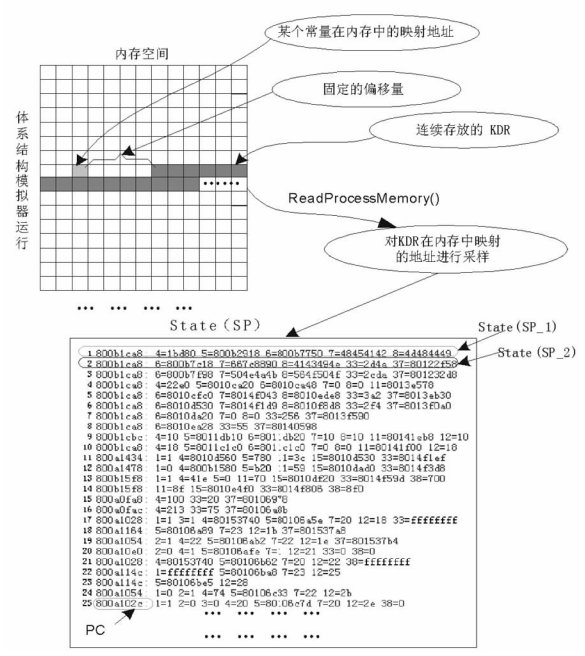


图 3　ARC 模拟器执行信息采集

Fig.3　The process of the information collection of the ARC simulation

步骤 2　RTL 模拟器执行信息的采集

在 RTL 代码中处理器的每个功能部件以 KDR 的写使能作为触发条件,监视功能部件向 KDR 写的过程并生成一系列的 KDR 写操作的集合,每一个 KDR 写操作包括以下信息:当前时间、要写入的 KDR 的编号、写入 KDR 的数值,调用函数 fdisplay( )(假设 RTL 代码使用 verilog 语言描述)将上述这些写 KDR 相关信息保存到 RTL 级代码执行信息文件当中。同时在 RTL 代码中处理器的取指部件以 PC 值发生变化作为触发条件,记录当前 PC 值和当前模拟时间,调用函数 fdisplay( )将 PC 值的相关信息也保存到 RTL 级代码执行信息文件当中。这样在 RTL 级代码模拟的同时,RTL 级代码的执行信息(包括写 KDR 相关信息和 PC 值变化的相关信息)就会按照时间顺序保存在 RTL 级代码执行信息文件当中,以供后续步骤使用。

步骤 3　初级检索

比较前两个阶段中统计的 RTL 模拟器和 ARC 模拟器的执行信息。若同一个 Benchmark 在 RTL 模拟器和软 ARC 模拟器的执行结果不同,那么 ARC 模拟器采集信息中的某一个时刻的 KDR 的数值,必然在 RTL 模拟器的执行信息中不能匹配。

初级检索首先读取 State(SP _ i)与 RTL 代码执行信息进行比对,直到 State(SP _ 1)匹配,然后再从 ARC 模拟器执行信息中的读取 State(SP _ 2)继续与 RTL 级代码执行信息中进行匹配,如此往复。若 RTL 信息中连续有 T 个(本文取 1 万个)写 KDR 操作引起的 KDR 变化的结果均与某一 State(SP _ i)不相同,就认为在该 SP 处出现不匹配,初级检索工作完成。如图 4 所示,在初级检索完成之后,最终会报告给验证人员某个采样点,记为 SP _ A 之前是执行轨迹正确的,在另外某个采样点,记为 SP _ B(B = A + 1)之前,故障出现。一般情况下 SP _ A 和 SP _ B 之间的写 KDR 操作不会超过 5000 次。

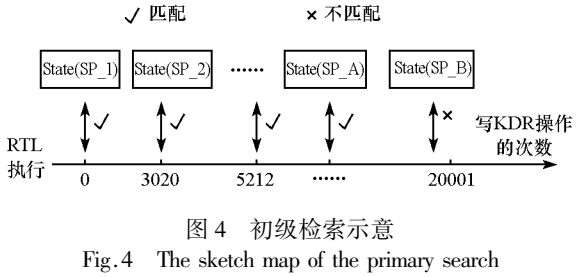


图 4　初级检索示意

Fig.4　The sketch map of the primary search

步骤 4　精确同步点生成

在 RTL 执行信息当中找到 SP _ A 之前的最近一处的 PC 值,统计该 PC 值在 RTL 执行信息统计结果中 SP _ A 之前出现的次数,记为 W。在 ARC 模拟器中的该 PC 值处设断点,并发送运行命令,等待 ARC 模拟器运行暂停后,将 W 减 1,继

续向 ARC 模拟器窗口发送运行命令,如此循环,直至 W 变为零,经过该步骤,能够使 ARC 模拟器执行到和 SP _ A 相对应的时刻。

步骤 5 高级检索

经过精确同步点生成阶段以后,达到了这样一组对应的状态:ARC 模拟器执行到 SP _ A 和 RTL 代码的某次写 KDR 操作对应,同时能够确定在接下来的 1 万次写 KDR 操作中某一次将发生错误。在该阶段采用自动单步运行机制,在 ARC 模拟器外部设计一个控制器,自动控制 ARC 模拟器单步运行,在每一单步运行之后读取 KDR 在内存中映射的值,和 RTL 代码模拟结果进行对比,若相同,就触发一次新的单步运行,若不同就停止比较,认为高级检索阶段完毕,交给验证人员分析。在高级检索阶段完成以后,留给验证人员的工作已经不多,验证或设计人员可充分发挥自己 RTL 代码的熟悉,快速定位存在故障的部件。

需要注意的是,本文提出的验证方法并不是在 RTL 代码模拟完某个 Benchmark 算法之后才开始运行和起作用的,本验证方法在 RTL 代码模拟工作开始一段很短的时间后就可以开始运行,在初级检索阶段检查到模拟已经有错误存在时即触发 RTL 代码模拟暂停,以减少对服务器资源(CPU、内存、硬盘空间)的浪费,并减少验证的时间。另外,本文主要考虑的是 ARC 模拟器是时钟精确时的情况,若 ARC 模拟器不是时钟精确的,步骤 4 ~ 5 将无法进行,但可以进行步骤 1 ~ 3,依然可以加速验证的实施。

## 3 分析及评测

为检验所提出的验证方法的可行性和效率,本文构建了如下试验环境:ARC 模拟器选择 TI 公司 CCS3.1 的 C6415 simulator[9],该软件支持将高级语言如 C/C + + 描述的算法编译为底层芯片支持的线性汇编语言,同时其源代码是不公开的。使用了 Cheat Engine 工具[7]进行内存查看,确定 CCS 中的 KDR 在内存中的映射地址。使用 Seraph 工具[10]控制 CCS 软件的执行,进行 ARC 模拟器信息采集、精确同步点生成和高级检索。使用 Perl 语言实现了初级检索算法。RTL 模拟器选择 Cadence 公司的 NC – Verilog[11], CCS 运行的硬件平台为 Intel P4 2.8GHz CPU、2GB DDR 内存的 Windows 操作系统的 PC 机,NC-Verilog 的运行平台为 4 核 1.2GHz CPU、64G 内存的 Solaris 操作系统的 SUN 服务器。

大型的 Benchmark 算法选取 Xvid-MPEG4 编/解码程序、H.264 编/解码程序、G721 音频编/解码器。其中 Xvid-MPEG4 对 5 帧采用 4:2:0 格式的 176 × 144 的图像进行编解码,H.264 程序选取了视频尺寸 20 × 10,按照 H.264 标准进行编解码,G721 音频编/解码器为 MediaBench 中一个应用程序[12],这些程序在 ARC 模拟器上模拟平均需要几十分钟,RTL 模拟需要 20h 以上,均为比较大的应用程序。

### 3.1 初级检索与高级检索的速率及精确度

本验证方法的核心环节是初级检索和高级检索,通过上述的 Benchmark 算法在本验证方法上的实施,两者的效率和精确度对比结果如表 1 所示。初级检索初步定位某一组写 KDR 操作集合当中存在问题,高级检索精确定位某一次写 KDR 操作出现错误。前者执行速度快但定位精度不高,后者执行速度慢但定位精度高。两者相互配合取得了较好的效果。

表 1 初级检索和高级检索比较

Tab.1 Primary search and advanced search

| | 效率 | 精确度 |
|---|---|---|
| 初级检索 | 2000 个 SP/min | 数千个写 KDR 操作之间 |
| 高级检索 | 200 个执行包/min | 1 ~ 2 个写 KDR 操作之间 |

### 3.2 ARC 模拟器的采样频率分析

在 ARC 模拟器执行信息采集阶段,选取合适 SF 是一个值得权衡的问题。若 SF 过高,则会加大 ARC 模拟器执行的时间,若 SF 过低,则相邻 SP 之间的写 KDR 的数目就会比较大,造成高级检索阶段花费的时间较长。若不考虑 ARC 模拟器模拟和 RTL 模拟之间存在一部分的并行性,同时假设出错点发生在 RTL 模拟的最后阶段,则一个程序从开始模拟到将错误定位到某一个写 KDR 操作花费的时间开销为:

$$T_{total} = T_{ARCSim} + T_{ARCInf} + T_{RTLSim} + T_{RTLInf} + \frac{SN}{V_{primary}} + \frac{N_{WKDR}}{SN \cdot V_{advanced}} \quad (1)$$

对于同一个的验证激励,式(1)中的 $T_{ARCSim}$, $T_{RTLSim}$ 和 $T_{RTLInf}$ 是固定的,和 ARC 模拟器的采样频率无关。因此我们只用关心剩余的时间即可,记为分析时间

$$T_{analysis} = T_{ARCInf} + \frac{SN}{V_{primary}} + \frac{N_{WKDR}}{SN \cdot V_{advanced}} \quad (2)$$

式(2)中的 $V_{primary}$ 和 $V_{advanced}$ 按照 3.1 节得出的数值,也是一个常数。$T_{ARCInf}$ 和 $\frac{SN}{V_{primary}}$ 随着 SF 的增加而增大,$\frac{N_{WKDR}}{SN \cdot V_{advanced}}$ 随着 SF 的增加而减小,这样

必然存在某一个 SF 能够使 $T_{analysis}$ 最小。

选择一个 1/20 帧的 H.264 编码函数作为验证激励,分别选取不同的采样频率,发现 SF 为 25 次/s 能够使对应的 $T_{analysis}$ 最小,因而 SF 为 25 次/s 是一个比较好的选择。

### 3.3　验证环境额外的开销

采用 3.2 节得出的最优 SF(25 次/s)和 2.3 节提出的 ARC 模拟器和 RTL 级模拟信息采集方法,得出如图 5 和图 6 所示的开销对比图,采用本验证方法中 RC 模拟需要的额外的开销为 15% ~ 20%,RTL 级模拟需要的额外开销为 5%左右。由于 ARC 级模拟和 RTL 级模拟可以并行执行,并且 ARC 级模拟所花费的时间一般只有 RTL 级模拟的 5%左右,因此虽然本验证方法使得 ARC 模拟需要的额外的开销为 15% ~ 20%,但从整体考虑,和传统的方法相比,本验证方法需要额外的时间开销只有 5%左右。

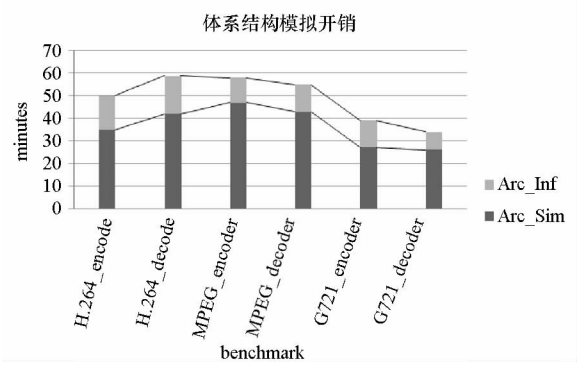


图 5　增加信息采集对 ARC 模拟时间的影响

Fig.5　The extra time cost of the ARC simulation in our method

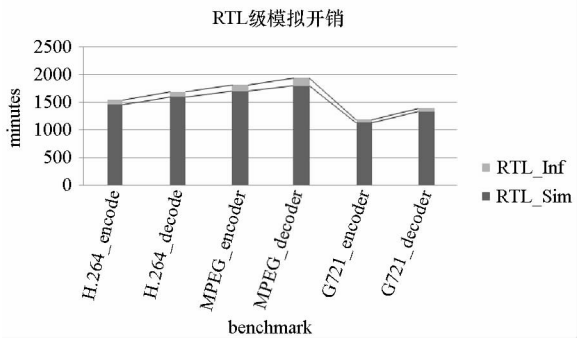


图 6　增加信息采集对 RTL 模拟时间的影响

Fig.6　The extra time cost of the RTL simulation in our method

### 3.4　查错时间分析

由于在芯片设计的模拟验证阶段,传统的验证方法的查错时间受到验证人员的经验、对 Benchmark 的熟悉程度甚至是验证人员的运气的影响,因此定量地比较本验证方法和传统的验证方法的查错效率是没有实际意义的。

本验证方法在我们自主研发的两款银河飞腾系列 DSP 的系统级验证阶段得到了应用,并起到了重要作用。在系统级验证初期未采用本方法时,查找和定位一个错误一般需要数天时间,而应用了本方法后能够在数小时内定位到错误的具体原因,即查错效率提高一个数量级以上。

## 4　总 结

如表 2 所示:与电路压缩、验证用例压缩相比,本验证方法在时间和效率上具有优势;与参考模型的协同模拟方法相对比,本验证方法不需要 ARC 模拟器是同一课题组统一开发的,不需要 ARC 模拟器是开源的。因此本验证方法无论在理论上还是工程实践上都具有一定的价值。

**表 2　本验证方法的特点**

Tab.2　The features of our proposed verification method

| 主要方法 | 是否需要熟悉 Benchmark | ARC 模拟器是否需要开源 | 查错效率 |
|---|---|---|---|
| 电路压缩[2] | 否 | 否 | 低 |
| 验证用例压缩[2] | 是 | 否 | 低 |
| 参考模型的协同模拟[3] | 是 | 是 | 高 |
| 本验证方法 | 否 | 否 | 高 |

## 参 考 文 献:


[1]　International Technology Roadmap for Semiconductors [OL]. [2010 - 06 - 10]http://www.itrs.net.

[2]　Yuyama Y, Aramoto M, RTL/ISS Co-modeling Methodology for Embedded Processor Using SystemC [C]//Proceedings of the International Symposium on Circuits and Systems, 2004(5):305 - 308.

[3]　William K L, Hardware Design Verification: Simulation and Formal Method-Based Approaches [M]. Prentice Hall PTR, 2005: 205 - 206.

[4]　Habibi A, Tahar S, Design for Verification of System C Traction Level Models [C]//Proceeding of Design, Automation and Test in Europe, 2005:560 - 565.

[5]　严迎建,刘明业.片上系统设计中软硬件协同验证方法的研究[J].电子与信息学报,2005:317 - 321.

[6]　鲁芳,柏娜.基于 SystemC 和 Verilog 软硬件协同验证[J].现代电子技术,2009,31(4).

[7]　Cheat Engine [OL]. [2010 - 06 - 10] http: cheatengine. org/download.php.

[8]　Tsearch Tools [OL]. [2010 - 06 - 10] http://www.sai.msu.su/~megera/gist/tsearch.

[9]　Code Composer Studio 3.1 [OL]. [2010 - 06 - 10]http://focus.ti.com/docs.toosw/folders/print/ccstudio.html.

[10]　Seraph Tools [OL]. [2010 - 06 - 10] http://seraphzone.com/newbbs.

[11]　Cadence NC Verilog [OL]. [2010 - 06 - 10] http://cadence.com/us/pages/default.aspx.

[12]　Lee C, Potkonjak M, MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems [C]// Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture, 1997:330 - 335.