计算机研究与发展 ISSN 1000-1239/CN 11-1777/TP
Journal of Computer Research and Development 49(1): 21-34, 2012

超大规模集成电路可调试性设计综述

钱 诚[1,2,3] 沈海华[1,2] 陈天石[1,2,3] 陈云霁[1,2]
[1](中国科学院计算机系统结构重点实验室 北京 100190)
[2](中国科学院计算技术研究所微处理器技术研究中心 北京 100190)
[3](中国科学院研究生院 北京 100049)
(qiancheng@ict.ac.cn)

Survey of Design-for-Debug of VLSI

Qian Cheng[1,2,3], Shen Haihua[1,2], Chen Tianshi[1,2,3], and Chen Yunji[1,2]
[1](*Key Laboratory of Computer System and Architecture*, *Chinese Academy of Sciences*, *Beijing* 100190)
[2](*Research Center of Microprocessor Technology*, *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)
[3](*Graduate University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Design-for-debug (DFD) has become an important feature of modern VLSI. On the one hand, traditional pre-silicon verification methods are not sufficient to enssure the quality of modern complex VLSI designs, thus employing DFD to facilitate post-silicon verification has attracted wide interests from both academia and industry; on the other hand, debugging parallel program is a world-wide difficult problem, which cries out for DFD hardware supports. In this paper, we analyze the existing structures of DFD comprehensively and introduce different fields of DFD for debugging hardware and software. These fields contain various kinds of DFD infrastructures, such as the DFD infrastructure for the pipe line of processor, the system-on-chips (SOC) and the networks on multi-cores processor. We also introduce the recent researches on how to design the DFD infrastructures with certain processor architecture and how to use the DFD infrastructures to solve the debug problems in these different fields. The topologic of the whole infrastructure, the hardware design of components, the methods of analyzing signals, the compressing and storing signals, the using of DFD infrastructure in debugging parallel program and post-silicon debugging are all introduced. Furthermore, some possible future directions of DFD research are given based on the state-of-the-art literatures and industrial requirements.

Key words debug; verification; post-silicon verification; parallel program debugging; design-for-debug

摘 要 随着硬件复杂度的不断提高和并行软件调试的需求不断增长,可调试性设计已经成为集成电路设计中的重要内容.一方面,仅靠传统的硅前验证已经无法保证现代超大规模复杂集成电路设计验证的质量,因此作为硅后验证重要支撑技术的可调试性设计日渐成为大规模集成电路设计领域的研究热点.另一方面,并行程序的调试非常困难,很多细微的 bug 无法直接用传统的单步、断点等方法进行调试,

收稿日期:2009-07-17;**修回日期**:2010-11-09
基金项目:"核高基"国家科技重大专项基金项目(2009EX01028-002-003,2009EX01029-001-003);国家自然科学基金项目(60921002,61003064,61050002,61070025,61100163,61133004,61173001)

如果没有专门的硬件支持，需要耗费极大的人力和物力.全面分析了现有的可调试性设计，在此基础上归纳总结了可调试性设计技术的主要研究方向并介绍了各个方向的研究进展，深入探讨了可调试性结构设计研究中的热点问题及其产生根源，给出了可调试性结构设计领域的发展趋势.

关键词 调试；验证；硅后验证；并行程序调试；可调试性设计

中图法分类号 TP302

1 可调试性设计简介

随着硬件和软件技术的发展，复杂度越来越高，导致保证硬件和软件设计正确的重要步骤——调试的任务越来越艰巨.具体表现为，一方面，硬件集成度的提高，新的体系结构的出现，尤其是多核系统的出现，导致单独依靠传统的硅前验证(pre-silicon verification)无法保证硬件设计的正确性.最近的数据显示，仍然有许多错误在芯片流片后甚至投入市场后被发现，为了使这些错误在芯片推向市场前得到解决，这些流片后存在的问题需要通过硅后验证手段来发现[1].另一方面，在硬件集成度、复杂度提高的同时，软件的集成度、复杂度也随之提高，尤其是随多核而越来越普及的并行技术、多线程技术导致现在软件调试的复杂度大大提高，仅依靠传统的纯软件调试方法(如单步、断点等)要耗费大量的人力物力，这就急迫地需要硬件支持，从而通过软硬件协作来完成并行程序、多线程程序的调试.鉴于上述两点，硬件的可调试性设计已经成为硬件设计中一个非常重要的内容.

作为可调试性设计研究的一个重要动力，硅后验证越受重视所占芯片开发开销比重越大.根据最近几年的统计，平均每款芯片需要35%的开发时间用于硅后验证，Intel公司用于硅后验证的成本接近设计的1/3[1].为了提高硅后验证的调试效率，目前学术界和工业界有很多不同角度的研究，可调试性设计(design-for-debug)是其中的一个热点.传统可调试性设计包括种类繁多的方法和工具，例如约束检查(assertion)、扫描链(scan chain)等，这些工具和方法可以给调试带来极大便利，将其用于硅后验证是一种很自然的想法.在硅后验证的平台调试和实际系统的调试中，如果事先加入可调试性设计，可以帮助硅后验证更好地了解设计内部的状态，更有效地定位错误，提高硅后验证的效率.和各种物理探针类方法不同，将可调试性设计模块加入电路中是可调试性设计的重要特点之一，而通过对可调试性模块的精心设计，使错误能被很容易地精确定位，同时获得各种灵活的调试功能，这是可调试性设计的研究目标.

可调试性设计研究的另一个主要动力并行程序调试也在蓬勃发展.可调试性设计可以为并行程序调试提供强大的硬件辅助功能.目前这方面的研究热点主要有两个：一个是解决并行程序调试的确定性重放(deterministic replay)方法，二是用于硅后验证的数据竞争冲突检测(data race detection).通过加入可调试性设计可以将上述两种问题的解决速度提高5～10倍.

本文介绍了可调试性设计系统包括的4个组成部分，以及最新的可调试设计系统的结构设计，在此基础上进一步介绍了可调试性设计的热点应用，即保证并行程序正确的确定性重放、数据冲突检测硬件辅助可调试性设计.对于可调试性设计和应用热点，皆指出了面临的主要问题、解决问题的关键所在以及现在最新的技术进展.最后，给出了当今可调式性设计的总结和发展趋势展望.

本文的剩余部分组织结构如下，由于可调试性设计的结构设计和功能需求是可调试性设计的基础和最重要部分，因此，本文首先花了比较大的篇幅对其进行介绍.第2节介绍了可调试性设计的基本概念；第3节介绍了可调试性设计的几种主要结构；第4节探讨了可调试性设计结构中的数据采集、存储、压缩和导出；然后我们在第5节介绍了可调试性设计用于解决并行程序调试问题的确定性重放设计，以及用于解决数据冲突验证的硬件自动检测设计这两个热门应用.最后一节对可调试性设计进行了总结，并在此基础上对可调试性设计发展趋势进行了展望.

2 可调试性设计解决的问题以及包含的4个组成部分

2.1 硅后调试面临的问题

硅后调试问题的基础是有效率地从实际芯片中

获取运行状态,目前有两种方法:一种是探针类方法,通过机械或者激光的方式[2-4](如 LVP(laser voltage probe)激光电压探测仪技术、EMMI(emission microscope)微光显微镜等光电检测技术)探测芯片内部信号,可以采集一个或一组具有精确时钟周期的信号,并能重构成高抽象层次的信号;另一种是通过芯片的引脚利用内部的观测逻辑将状态输出.前者适合对底层具体信号进行分析,后者适合对高层抽象信号进行分析,对于边界条件下的调试以及非逻辑错误往往需要两者结合使用[5-6].

对于硅后调试问题,一次调试可以分为 5 步操作,即控制错误、分离错误、定位错误、扩展错误、解决错误,其中开销最大的两步是控制错误和分离错误.不同于硅前验证向量的侧重理论推导和仿真[7-8],硅后验证需要面对现实中的不确定因素,不仅要面对验证平台提供的验证向量(如 Intel 的奔腾 M 测试平台提供的种类齐全的硅后验证向量[9]、shmoo plotting 测试中的电压温度对),还要面对各种真实环境下的可重现问题和不可重现问题(如启动操作系统的调试过程、验证平台的电气故障),需要通过验证向量的可控来使错误可控、可重现、可捕捉.分离错误是在控制错误的基础上,利用已有的信息和手段将错误隔离到设计中的一个小范围,例如特定的部件和时钟周期.

硅后调试需要解决的问题主要是在受限的硅后环境下解决调试问题的控制错误和分离错误.

2.2　可调试性设计包含的内容

目前成熟的 DFT(design for test)设计都可以作为 DFD 设计的基本功能加以使用.例如 IEEE 的 TAP(test access port)标准[10]提供了一种最普遍的 DFD 功能,即扫描链访问、断点设置、时钟控制等.此外,后台调试模块(background debug mode, BDM)[11]设计也提供了低层调试功能,通过将 CPU 停机进行单个时钟的调试.

上述调试标准提供了实用的调试功能,目前被很多处理器采用,但是,随着处理器设计日趋复杂,可调试性设计研究变得越来越重要,不断有新的功能更强大的结构被提出,概括而言,可调试性设计系统主要包括 4 部分:确定设计需要加入的结构、确定需要采集的信息、存储和导出信息、设计分析信息的软硬件系统.下面我们将结合最新的研究进展,详述可调试性设计研究的 4 个方面.

2.2.1　确定需要加入的结构

可调试性设计为了提高设计内部状态的可视性和精确定位错误,需要具备如下功能:监测信号、分析信号、接收外部控制信号、建立用于传递信号的网络、存储和导出信号以及处理导出的信息.

基于以上功能需要,可调试性设计一般包含如下几种结构:

Probe——用于获取信号并传递给信号分析模块;

Monitor——内部包含各种状态机,对信号进行分析和处理,生成需要记录的信息,产生内部的控制信息或事件;

Controller——接收内外的控制信息,控制整个调试结构的运行;

Memory——存储需要记录的信息;

Net——将各种调试性设计模块联接在一起的网络,各种控制信号、事件、观测信号在其中传递.

以上结构在不同的可调试性设计实现中可以根据需要进行组合(如图 1 所示),产生各种具有针对性功能的模块:

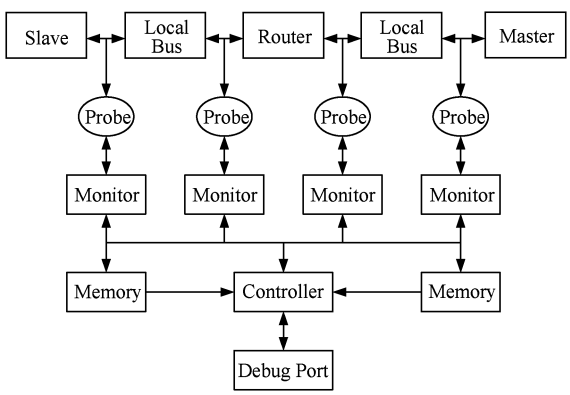


Fig. 1　An infrastructure of DFD.

图 1　可调试性设计结构示例

2.2.2　确定需要采集的信息

对于硅后调试而言,需要获得的信号越多硬件或时间开销越大.对于每一个具体的设计需要在提高信号可见性和降低开销之间取得平衡.

确定采集哪些信息:可调试性设计所采集的信息应该是能反应设计状态,能够用于分离和定位错误的必要信息.根据分离错误的需要只采集尽量少的必要的信息是最基本的设计出发点,而采集信号的开销和调试方法对错误的覆盖率是成正比的.在实际的工程中,确定可调试性平台开发效费比的工作不可忽视[12].

确定采集的时间窗口等配置内容:设置采集的数据窗口主要通过内外控制信号来判断.外部控制信号对于不可重复性错误和不确定性出现的错误

很难起作用；内部控制信号主要通过监测到的信号来判断系统的当前状况，因此通过内部控制信号判断错误是否发生，需要错误捕捉机制对可能发生的异常状态产生正确而及时的反应. 内部控制信号的产生机制越全面复杂开销就越大.

2.2.3 存储和导出数据

硅后调试信息一般是通过寄存器和专用的存储部件保存下来，再通过扫描或专用的存储部件构成的调试通道导出.

扫描：扫描的实现方式比较简单，获得的接口支持也很广泛，而且也能获取特殊的状态信息[13]，但是对于需要保存大量数据的调试性设计，扫描提供的带宽过小，对实时性导出的限制非常严格，而且仅用扫描链和内建自测试(built-in self-test，BIST)能够达到的测试质量是有限的，文献[14]给出了其能达到的测试效率. 因此高性能的可调试性系统一般用扫描链结合专用存储 buffer 来记录和导出数据.

专用的存储部件：利用 buffer 或者 cache 来保存要存储的数据[15-17]，在控制信号的驱动下，可以开始、暂停、停止. 为了尽可能多地记录数据，各种压缩算法和实时处理方法也被引入和使用.

2.2.4 分析信息

记录的信息需要进一步分析. 一般来说记录的数据抽象层次越低，需要越多的工作来处理数据，使其更易被观察. 对于记录的数据，分析信息的方法主要来源于对已知错误分析方法的总结、对不符合规范的行为划分的方法以及选取覆盖广泛的判断错误方法等.

一种错误判断逻辑能够检测的错误种类越多(如判断死锁)，越需要更多的辅助工作来进一步精确定位错误，但是优点是监测机制的相对设计开销小、可移植性强.

目前各大芯片开发公司都有专门的对调试信息进行分析的软硬件平台.

3 可调试性设计系统的结构设计

可调试性设计可以应用于系统的各个层次，从逻辑门到总线，从处理器核到 SoC 系统都可以加入可调试性设计逻辑. 下面通过对各种可调试性设计结构的介绍来阐述可调试性设计的特点和发展趋势.

3.1 简单的可调试性设计

shmoo plotting[18]测试是一种对 IC 设计的工作环境向量进行严格测试的工序，用于确定 IC 设计的正常工作环境范围. 从 20 世纪 70 年代被提出后，其测试手段一直随自动测试设备(automatic test equipment，ATE)工具的进步而改变. 可调试性设计在 shmoo plotting 测试中的使用是提高硅后调试效率的一种典型应用.

文献[19]介绍了一种用于测试存储部件正常工作电压频率对的可调试性设计. 对于待测试的存储部件，流片后的功能不一定遵守设计规范，对比逻辑设计和实际电路，两者的正常工作范围并不一致，在电路中加入可调试性设计可以观测实际电路状态，寻找问题所在.

如图 2 所示，对存储部件进行 shmoo 测试的可调试性设计的组成包括控制接口、实现状态机的控制寄存器、存放访问指令的数组、可控的时钟输入、记录出错后被恢复信息的容错存储，以及计数、读写控制等功能.

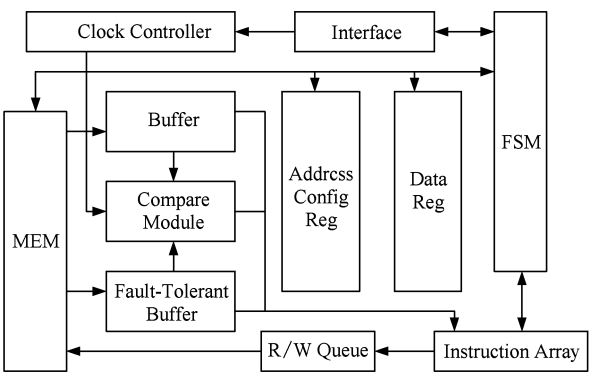


Fig. 2 DFD for memory shmoo.

图 2 对存储部件进行 shmoo 测试的可调试性设计

在上述设计中，可调试性模块和待测的存储部件各自独立，环境向量可控(包括时钟频率可控、数据内容和地址可控)，测试结果可存储，实现了解决调试问题前两步的前提条件，只要将测试结果导出就可以得到整个测试过程的状态.

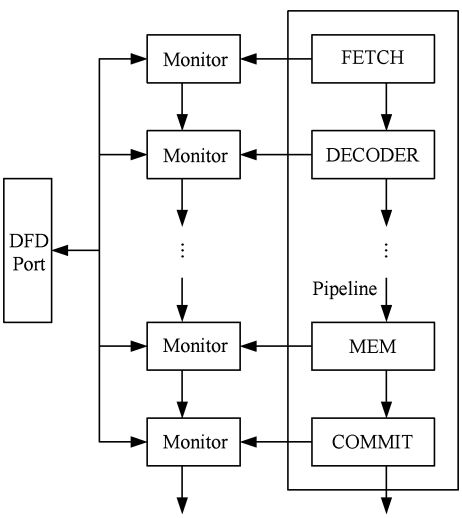
这个应用很好地体现了可调试性设计在提高系统状态观测性方面的功能. 但是，很多情况下，我们不仅需要泛泛地了解系统的内部状态，更关注特定情况下的系统状态，让可调试性设计最大化地提炼出有用信息，这对复杂电路结构调试来说非常重要.

3.2 用于处理器流水级的可调试性设计模型

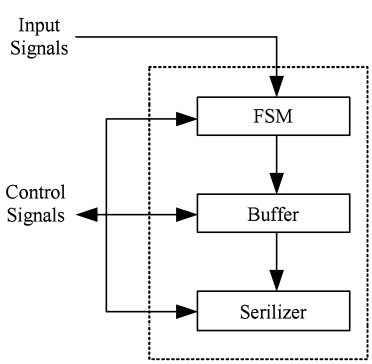
处理器核是大型系统的核心模块，面临的调试问题具有一定的代表性. 对处理器流水级可调试性问题的分析思路有很多种，有的通过增加超时计数器、寄存器文件错误检测码(error check code，ECC)、

指令代码奇偶校验等方法来提高流水级运行时的健壮性,增加容错率[20],有的通过加入可配置的支持硬件对流水级中出现的硬件设计问题进行规避解决[21],而文献[22]介绍的一种对流水级中指令的trace进行记录,判断流水级状态是否正确,进而自动定位错误的可调试性设计结构则最具代表性,充分运用可调试性设计给硅后调试带来的改进.

如图3(a),(b)所示,处理器流水级的可调试性设计结构由控制模块(controller)、分布于各流水级各功能部件的监测部件(monitor)、联接各调试性模块的流水级网络、各监测模块内保存指令足迹(trace)的存储缓冲区(buffer)以及对各存储缓冲区中数据进行导出处理的整理模块(serilizer)组成.



(a) DFD for pipeline



(b) Monitor

Fig. 3　DFD for pipeline and the sketch map of monitor.

图3　处理器流水级的可调试性结构和监测部件的结构

在上述系统中,网络传递信息如下:1)各指令的标识ID,用于唯一表示一条在流水级中的指令,该ID和指令在流水级中的传输同步;2)外部的控制信息,用来控制记录的运行、暂停和结束;3)各monitor对信息进行分析后产生的触发事件,用来判断内部控制信号是否成立.

信息采集和存储的过程如图3(b)所示:每个功能部件的monitor有一个状态机,当没有新的ID传送过来时,时间计数器记下有多少拍空转.当新的ID进入时,时间信息被记录入buffer中,同时指令ID和各功能部件提供的内容一起存入buffer中.考虑到各功能部件需要观测的状态不同,为避免重复记录,各功能部件提供的内容并不统一,而是根据需要有所不同.例如,取指模块只记录pc,译码模块记录指令包含的操作数个数等等.当controller发出停止并导出trace的命令后,先将取指部件的buffer中记录的trace整理成1份指令执行顺序文件,然后按时间顺序将提交部件中已经提交的指令在各部件中的trace整理一遍,再将等待提交的指令按时间顺序,将其在各部件中的状态整理一遍,其余信息忽略掉.

信息分析:monitor检查5种情况来判断错误是否发生,分别是向量错、运算错、非正常例外、段分配错、死锁,其中段分配错和死锁从导致错误的指令执行到错误显示出来需要经过很多时钟节拍,而buffer的大小是有限的,如果采用判断出错再停止,可能buffer中记录的信息已经无法定位到正确的错误点,因此,需要采用预判断措施.当段分配错发生时,先发生tlb_miss,可在此暂停buffer的记录,当错误真正发生后再导出buffer的数据,反之则继续执行记录,考虑到发生死锁前必然会发生指令未能及时提交的状况,可在一个取指周期的间隔后未能发生指令提交的情况下暂停记录.

对于导出的trace,利用数据依赖分析、控制流分析、存取分析、译码分析等方法,使用软件自动定位错误.统计的结果为75%的错误得到精确的定位,21%的错误除了精确定位还伴有冗余定位,4%的错误没有检测出来.

这项研究对于探索调试的自动化作出了贡献,其结构具有普遍适用性,其对错误的分析方法也取得了良好的覆盖率,尽管很多环境下自动判断错误的机制达不到这样的精确程度,但是其提出的暂停结构对于判断一些未知的错误是一种有效的思路.

3.3　用于片上网络的可调试性设计结构

目前,多核芯片中使用复杂的片上网络已成趋势[23].对于片上网络的调试,不管是要检查数据是否符合协议还是其他错误情况,都需要了解片上网络当前的运行状态.片上网络调试性设计的特点是其中传输的内容往往需要在高级的抽象层次观察,

对传输内容进行分析往往要考虑不同协议下的传输，基于这样的特点，以通信为中心的模型能比较好地描述片上网络的状态；而另一些情况下，对片上网络中传输的内容需要分析其相关性，这时基于一致性协议的模型更适用. 在实际工作中根据片上网络调试的侧重点不同选择采用不同的模型.

3.3.1 以通信为中心的片上网络可调试性设计模型

文献[24-25]介绍了片上网络的通信模型. 一种多 master 到单 slave 的映射网络如图 4(a)所示，它包括 N 个 slave 网络接口，一个 1:N 的 shell 以实现单对多的映射，一个 master 网络接口；图 4(b)为单 slave 到多 master 的映射网络；图 4(c)为一个基本的多 master 到多 slave 的片上网络：

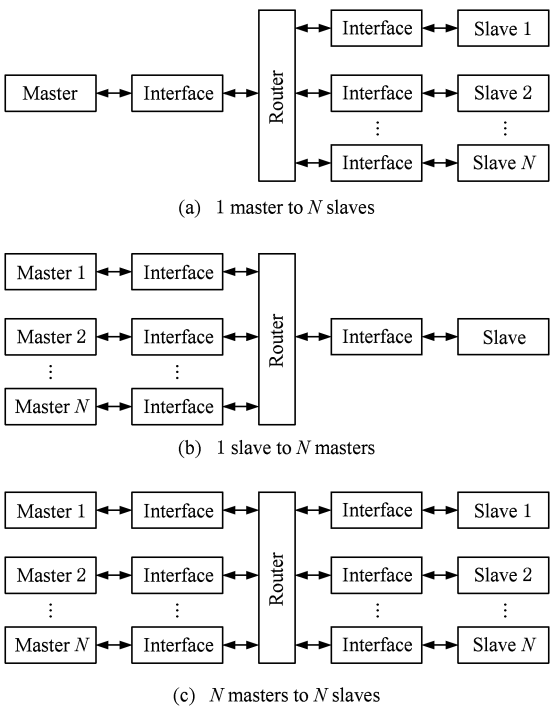


Fig. 4 Model of networks on chip.

图 4 片上网络通信模型

文献[26]介绍了一种对片上网络加入调试性设计的方法. 如图 5 所示，每一个 master 和 slave 是发送和接收通信的模块，需要加入通信控制 monitor，它有两个功能：监视 master 和 slave 端的通信情况，根据控制信号改变 master 和 slave 端的请求和应答. 每一个路由逻辑一般包含 N:N 的映射关系，对其中每一个 N:1 或 1:N 的映射，都需要一个 monitor 来监控其中的映射关系，并将分析的数据和结果存储到 buffer 或其他模块. 每一个 1:1 的 master 和 slave 之间的通道可以根据选择加入一个 probe 来观测数据的传输情况，以提高片上网络的可视性. 一个控制模块用来控制信号的获取和新的激励的输入.

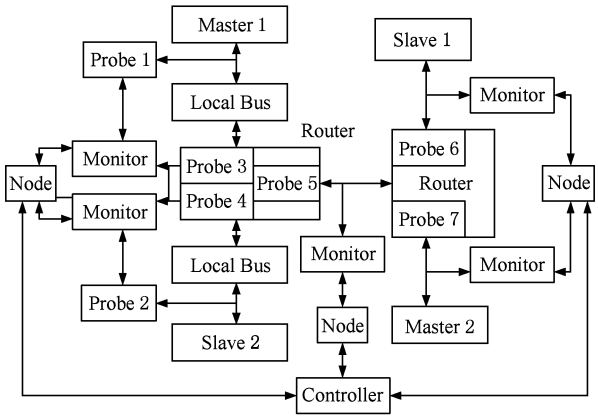


Fig. 5 DFD for networks on chip.

图 5 一种片上网络通信模型的可调试性设计

文献[27]提出了片上网络的多粒度调试方法，充分使用了可调试性设计结构的通信特征，特别针对片上网络这一特殊的结构，将网络中的通信划分为连接层、通道层、传输层、消息层、消息元素层、时钟层，提高调试信息的抽象层次，与门级电路中的单 bit 信号相比更容易理解和分析.

对于片上网络的每一个 N:N 的路由模块，需要加入 $2N$ 个 monitor，用于记录和分析每一个 N:1 或 1:N 映射 shell 的状态，这些 monitor 判断其监测的网络，将生成的事件(例如信号的请求或应答握手)发送给事件 node 模块. 事件 node 是每个路由模块对应的一个组成事件交互网络的部件，接收本路由模块中所有 monitor 的事件并和其他事件 node 产生事件信息的交互，它拥有一个事件驱动的状态机处理各种情况.

控制模块通过内部事件和外部控制信号产生控制信息并发送给调试网络中的各 monitor、事件 node，probe.

片上网络的状态保存于各网络接口的寄存器中，各网络接口中的状态可以通过扫描的方式导出.

文献[24]提出的调试模型对于片上网络的可调试性设计具有普遍适用性，包含观测、记录、修改测试向量几种功能，而文献[25]提出的多粒度调试方法是对文献[24]方法的进一步细化. 然而，以通信为中心的调试方法侧重点是对片上网络运行状态的观测和改变，需要进一步分析来对片上网络中的错误进行分离，而采用扫描方式导出数据也需要高效的错误定位方法辅助. 因此，在实际工作环境中，采用 buffer 或实时导出的方式可能更加适用.

3.3.2　以检验一致性为中心的结构模型

以通信为中心的调试方法对于观测片上网络运行状态提供了一种有效的方法,但是在某些特定的环境下,可能需要对信息进行更加有针对性的分析,文献[28]介绍了一种用于调试共享存储网络中的一致性错误的结构,利用 cache 的存储空间,当处于调试模式时,其中一些区域被用来记录对存储地址的访问序列,通过访问序列可以知道各处理器对同一地址的操作顺序,结合 cache 需要维护的一致性协议来判定是否存在错误访问.

3.4　用于 SoC 的可调试性设计结构

SoC 的可调试性设计有其独特的实时性、复杂性的特点,种类繁多的协议,不同特性的设备,使得可调试性设计往往有很强的针对性. 文献[29]提出了一种可配置的调试性结构,为了能够使可调试性模块可重用,其设计包括如下组件:

probe network——观测信号并传输到指定部件的网络,可以跨越时钟域;

debug monitor——对信号进行分析的模块,可以跨越时钟域;

wrapper——输出等于输入信号或控制信号传入的激励信号;

tracer——存储设备;

capstim——事先存入激励向量,输出为存储的激励,输入为其他模块给它的反馈.

在设计时,对各可调试性组件可以从接口到内部状态机进行配置. 该结构对信息的采集提出了 4 种配置方式,assertion 判断(将 assertion 判断逻辑加入 monitor 中)、event 分析(对于复杂的情况以及需要提高观测数据的抽象层次时,在 monitor 中加入事件分析的组合逻辑)、激励捕获(对于指定的激励输入,获取执行结果,通过添加 capstim 模块来实现)、假设性实验(采用添加 wrapper 的方式来置换想要改变的信号,观测执行结果). 对于每一个实际系统,配置需要采集信号时,根据上述特点,可以采集有限状态机信号、传输的重要表征信号(特别是提供更高层次的抽象信息,如果待测设计中没有这类信号,可以通过 monitor 中加入的组合逻辑生成)、assertion 的输入信号、需要进入 wrapper 的信号.

采用可配置的调试性设计对于越来越庞大和复杂的系统是一个重要趋势. 随着集成电路设计工艺的不断更新、功能的日趋复杂,为每一个新的设计重新进行可调试性设计将是一个艰巨而费时的工作,开发可调试性系统时进行可配置性的设计可以节省大量的不必要开销.

3.5　几种实际系统的可调试性设计

目前可调试性设计逐渐成为芯片设计中必不可少的部分,各种商业芯片更是采用尽可能高效的可调试性设计.

文献[9,30]介绍了 Intel 的 Pentium M 处理器的验证平台和 Pentium pro 处理器的验证经验. 该平台通过一块 FPGA 将验证向量实时导入处理器,通过中断等方式切换验证向量的替换和验证处理过程. 验证向量来源于真实环境下导致处理器错误的大量实例. 平台提供时钟控制到 shmoo 变量(电压和时钟频率)控制等多种支持,同时对于系统各部分的状态提供大量的存储支持. 一旦验证向量产生错误的结果,验证平台切换到错误处理程序,对记录的 3 种信息(状态信息、同步信息、恢复现场信息)进行分析和处理,随后对异常情况的判断、错误出现时间和部件的锁定工作由软件完成,效率非常高.

文献[31]介绍了用于摩托罗拉 MPC7410 及其改进型和 DSP56621 数字信号处理器的可调试性设计. 这是一种最基本的扫描链结构,通过对芯片内部的寄存器插入扫描链,采用 jtag(Joint Test Action Group,IEEE 1149.1)接口来控制数据的导入导出,不具有实时分析的能力,只提供有限的观测功能.

4　可调试性设计中数据的采集、存储、导出、分析

4.1　数据的采集

为了高效地提取有用信息,可调试性设计模块必须对采集的信息进行判断和处理,例如,对存储单元的可调试性设计,根据访问地址判断是否记录其附带的数据,对流水级各部件返回的状态信息,判断是否停止记录转入错误处理程序等. 运用先进的理论分析,将复杂的调试方法转化为高效可实现的方法,可以减少需要采集的信号[32-33],这是提高采集效率的重要方法.

在信号处理的可配置性和自动化方面,文献[34]进行了一些探索,概括为两个基本问题:SAT 问题[35-36]和 QBF 推导[37]. 当输入信号组不变,对特定的组合逻辑需要通过 SAT 分析来确定信号组满足组合逻辑的全部子集,复杂度为 NP. 实际环境中一个可调试性设计会提供若干个固定的寄存器,对于

SAT 分析得到的所有子集，通过 QBF 推导，找到满足条件的寄存器设置值，使得 SAT 分析得到的子集可以经过这几个固定的寄存器代表的判断逻辑，得到等价于特定组合逻辑的结果. 当需要改变对采集信号的处理方法时，只需要将新的组合逻辑通过 SAT 分析和 QBF 推导得到等价的配置值，就可以达到仅通过改变固定的可配置硬件和固定的输入取得可变的分析逻辑的效果.

4.2 数据的压缩存储

由于 buffer 或 cache 存储容量的有限性，以及实时导出对带宽的要求，常常需要对记录数据进行压缩处理，压缩的方法有两类：

1)利用数据的逻辑特性进行压缩. 由于采集的原始数据往往是低层次的抽象，通过组合逻辑可以产生更高层次的抽象数据. 例如通过记录待测的运算部件提供的余数码(residue code)这一高层次的抽象取代了对运算结果的记录，通过记录通信在通道中的状态来取代对低粒度信息的记录. 采用高抽象数据的优点是直观，更容易被分析软件直接利用，缺点是依赖于事先确定的组合逻辑对错误情况的覆盖率，会损失一些信息，而这可能导致一些情况无法定位错误，必须依赖多粒度的配置或其他存储方式的辅助.

2)利用不同环境下数据的规律性进行压缩. 例如流水级中 pc 值变化为等加数列，可默认不记录. 文献[38]介绍了一种对于变化不大的多媒体数据进行差值压缩的可调试性设计，将存储数据分为基地址、偏移量、数据信息、连续 0 数据 4 种类型，由于大多数数据为全 0 信息以及非 0 信息之间极小的偏移量，因此可以达到 90%左右的压缩率.

在 SoC 环境中，压缩是一种普遍采用的方法，而实时性压缩也是经常需要满足的要求. 在各种压缩源中，相同来源的一组数据往往具有内在的联系性，例如同一种语言的文字并不是杂乱无章的字母或文字组合. 目前对于这种特性的压缩源，采用字典压缩算法或字典压缩混合其他压缩算法比较有效[39]. 但随之而来的问题是这些高效的压缩算法需要记录初始值或经常改变字典，而有限的存储空间在进行覆盖存储时会破坏压缩的数据. 在实时环境下，启动一次压缩流程需要初始化硬件，实时压缩需要解决这一限制. 文献[40]提出了一种实时压缩存储结构，并对循环 buffer 进行改造以满足压缩算法特性的需要. 利用两套压缩硬件轮转工作，共用一个存储区的方法保证了实时压缩能正确初始化. 为了不使循环 buffer 的覆盖特性破坏压缩数据，将存储区间划分成若干区域，每个区域重启一次压缩算法，保存初始值，用很小的代价保存了大多数数据，使得字典压缩算法的优势得以利用.

文献[41]提出了一种有损压缩方法来确定错误范围. 对于采集的信息进行每一层压缩时，仅保留必要的信号以作为确定下一层的划分依据，这样的有损压缩——如对总线上的信号只记录握手信息，可以用来定位大时间范围内的罕见错误，对于可重复的罕见错误，通过有损压缩可以在很短时间内定位到一个小的范围.

4.3 数据的导出

可调试性设计需要导出的数据目前有两种：一种是各种寄存器记录的设计内部状态；另一种是可调试性设计存储模块保存的压缩数据或各种 trace.

数据的导出也有两种方式：一种是通过高速的调试通道，直接将压缩数据或各种 trace 通过外部读取命令送到外部存储介质上，这种方式需要专门的测试平台，优点是速度快，对实时调试支持好，缺点是目前没有高效的通用规范，以及调试系统要保证调试数据导出通道不受待测设计错误的影响；另一种使用广泛的数据导出方法是使用扫描链，优点是有通用的 jtag 接口，简便易用，可以通过内建自测试存储模块作为 buffer 来保存数据，然后通过 jtag 接口扫描出去. 由于有统一的标准，目前有很多支持的验证平台，例如文献[42]. 但是扫描链方式缺点是速度比较慢，对于实时的需求有限制.

4.4 数据的分析

对于可调试性设计导出的数据一般采用软件来推导和定位错误区域[43-45]，例如结合指令在流水级内部的状态，通过对错误类型的分析，定位错误到相应的部件. 这一分析过程使用软件来确定有无数据依赖、向量分配错误、指令跳转错误等一系列问题. 当软件发现其中存在错误时，数据往往已经隔离到单个部件和单个周期，提高了调试的效率. 在对片上网络的分析中，可以通过记录片上网络的各节点状态，分析网络的堵塞情况，进而将错误定位到特定的一次通信. 在对访存操作的分析中，可以通过软件分析记录下来的访存操作的序，将错误定位到某个处理器对某个地址的一次访存操作.

对错误的修复，既可以修改 rtl 设计，也可以通过工艺设计来弥补，采用如硅后调整、PARsyn 自

动修复、闲置单元插入(spare cell)等方法.通过分析错误对系统的影响,可以在新的设计中采用相应的容错措施,以提高设计的健壮性[46-47].

由于数据之间的相互关联,一次记录可以提取出非常多的信息.文献[48-49]研究了如何从有限的数据中得到最大化的信息,以提高信息的可视性.这些研究为相应 eda 工具的开发提供了一些可操作的方法.

5　可调试性设计在硅后调试和验证中的热点应用

5.1　可调试性设计解决并行软件调试的确定性重放方法

5.1.1　确定性重放方法基本概念

随着多核处理器的出现,并行程序应用越来越广泛,但随之而来的一个问题就是并行程序的调试相对于传统的串行程序更加复杂.现在依靠传统的断点等方法调试手段已经无法保证并行程序的正确性,为了更好地调试程序,保证程序的正确性,确定性重放已经成为一种重要方法.

多核并行环境十分复杂,不确定因素很多,经常导致并行程序运行出现致命错误.为了重现不确定因素,确定性重放方法就是通过在程序运行时录制必要信息,在必要时进行重放.

由于重放只有在特定的情况下(如出错后)才需要,因此,其效率要求相对来说不需要很高.而录制往往要求能够在实际发布的产品中运行,这就要求录制带来的额外消耗要低,录制文件要小.

因此确定性重放的可调试性设计面对的的主要问题是如何解决有效录制不确定信息,这些不确定信息主要包括各种 IO,DMA 操作;程序中遇到的中断、陷阱;多核系统的多线程并行程序中对共享存储访存的访存顺序.

一般的程序中 IO、中断、DMA 等不确定信息相对不频繁,因此录制它们成本较低.而对于并行程序,各线程的 CPU 核间共享存储访问却很频繁,因此,高效录制的关键之处就是如何高效地录制共享存储的访存顺序.

录制这些信息有软件(如 Microsoft 的 iDNA[50],Intel 的 pinSEL[51])和硬件两种方法,但软件方法运行效率的极低,有研究表明,这些软件方法导致程序运行比原始的程序要慢 10 倍.因此,近年来,研究者开始考虑利用硬件录制这些信息,从而降低运行时的消耗.其基本的录制结构如图 6 所示:

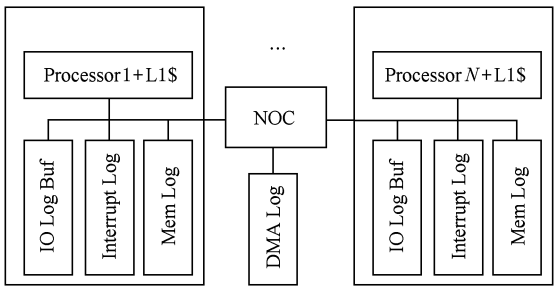


Fig. 6　An infrastructure of deterministically replaying.

图 6　常见的确定性重放基本录制结构

下面将介绍确定性重放算法利用可调试性设计的结构采集、存储,以及重放数据的研究进展.

5.1.2　确定性重放的可调试性设计

较早(2003 年)提出的确定性重放可调试性设计是 Xu 等人的 FDR[52].该设计是在缓存模型为基于目录(directory-based)、顺序一致性(sequentially consistent, SC)的多核 CPU 系统上实现的.对于录制的关键所在——访存竞争结果的录制则是通过如下方法来提高效率.在顺序一致性模型的系统中,每个线程的所有指令形成一个序列(称 total order),这种序列可以通过一个叫 Lamport Clock[53]来表示,且对访存满足"每次读出来的结果都是最近一次写进去的(a load get value last stored to an address)"[54].为了用尽量少的 log 记录这些指令访存顺序,FDR 先把前述顺序关系用有向弧表示,然后对有向弧形成的有向图进行化简,化简的主要依据是这种访存顺序的传递性(如 $a<b$,$b<c$,$a<c$ 依据传递性只需记录 $a<b$,$b<c$).采用这种方法可以大大减少访存记录但不会造成重放所需要的信息不足.最终,该方法取得了比较小的 log(25MB/second/processor)以及 overhead(小于 2%).不过,该算法有不足之处,首先,该算法并不是记录程序运行后所有的信息,而是只能记录最近的一段信息(大约 1s),从而不能从程序执行处开始重放;其次,该算法所基于的模型是顺序一致性模型,在这种模型下并行程序运行效率比较低.

在 2006 年,针对上述缺点,Xu 等人又对 FDR 进行了改进,改进后的算法为 RTR[55].在 RTR 中,它将访存顺序依赖具有可向量化(vectorizable)的特性进一步减少记录访存顺序依赖.在此之前的 2005 年,Narayanasamy 等人的 BugNet[56] 也对 FDR 进行了优化.

在 2008 年，又有两种方法在 ISCA 会议上提出：它们分别是 Montesinos 等人的 Delorean 等人[57]和 Hower 等人的 Rerun[58].

在 Delorean 的设计中，引入了一个基于分块(chunk-based)的概念. 通过对指令进行分块，只记录块内的访存顺序依赖，从而进一步减小了 log 的大小以及硬件消耗，该设计的一个关键就是如何进行分块. 在此之前的 2006 年，Narayanasamy 等人的 Strata[59]算法也采用了类似的分层(stratum)方法.

而在 Rerun 的设计中，则采用了和前面各种算法不一样的思路，上述算法都是通过记录关键的访存依赖来实现重放，而在 Rerun 的设计中，则是记录那些时间段和其他线程没有访存冲突(即不存在访存依赖). 将这些无冲突的程序区间分成原子块，然后记录这些原子块之间的执行顺序依赖.

以上的方法都是关注如何将访存依赖用尽可能少的 log 记录. 而在 2009 年，Lee 等人则提出一种不用记录访存依赖的方法[60]. 他们通过深入分析并行程序，提出对 load-based 的程序，只需要单独记录每个线程的所有输入，即可实现每个进程依赖重放并分析出访存依赖；并实现了一个离线的基于 Yices[61] SMT 的重放器，不过，不记录访存依赖是以牺牲重放效率为代价的. 而在 ASPLOS 2009 会议上，Montesinos 等人提出的 capo[62]则关注如何能用更“通用”的硬件、更少的硬件设计改动，以及如何定义一个好的软硬件接口来记录和重放.

5.2 可调试性设计用于硅后验证的数据冲突检测

5.2.1 数据冲突检测的基本概念

对于多核系统下的多线程程序，对共享存储进行访存同步是一个非常重要的机制. 当几个线程共同访问一个共享变量时，至少有一个是写时就会产生数据冲突(data race)，程序就可能执行错误.

由于数据冲突的存在直接影响到程序执行是否正确. 因此，对数据冲突进行检测很有必要，而这种检测又分为静态检测和动态检测. 静态检测主要通过软件(如编译器)实现，在此不讨论. 动态检测又分为软件方法和硬件方法，和确定性重放一样，纯软件的方法意味着程序运行的极其低效(如 Eraser[63]对有些程序有 10～30 倍的额外消耗)，因此很有必要用硬件辅助来检测数据冲突.

对于利用硬件检测数据冲突，最关键的问题就是如何用尽可能少(最好尽量少更改现有硬件的架构，如缓存机制)的硬件资源实现不存在漏报(false negative)和尽量减少误报(false positive).

为了用尽量少的硬件资源实现比较理想的检测效果，近年来，国际学术界提出了两类方法：一类是基于因果关系(happens-before)关系；另一类是基于类似锁协议的方法(lockset approach). 但在 2009 年，又有些新的方法提出：Shantanu Gupta 等人提出用事务内存[68]；Nistor 等人的 Light64[69]则是通过对执行的历史记录进行 Hash，通过比较 Hash 值来判断是否存在冲突. 下面我们将简要介绍这些算法和相应可调试性设计模型的研究情况.

5.2.2 数据冲突检测的可调试性设计

2003 年 Prvulovic 等人提出一种叫 ReNact[64]的数据冲突检测设计. 该设计主要基于 TLS (thread-level speculation)机制，通过一个类似于一种有序的“向量”时钟(vector clocks)，通过这些时钟的比较来判断是否出现数据竞争. 为了防止比较的数目过大，该设计将执行分成一个个叫 Epoch 的块，只比较块内的“向量”时钟. 要求分块的机制必须要有很好的设计，保证块间没有访存冲突. 该算法的硬件消耗比较大，不够理想，对线程数较多的程序效果运行 Overhead 也比较大.

在 2006 的 HPCA 会议上，Prvulovic 提出了一种 Overhead 极小的设计 CORD[66]，这是一个基于 happens-before 的方法，CORD 通过指令执行的逻辑时间(logical time)和存储地址时间戳(timestamp)的比较来判断是否存在冲突. 为了减少比较次数，对于一个读操作只会和写操作比较，因为只有写操作才可能和读操作发生数据冲突. 该方法由于是以 cache 行为单位加时间戳的，所以可能由于伪共享(fasle sharing)而导致误报(false postive).

在 CORD 出现的第 2 年，即 2007 年，同样在 HPCA 会议上，Zhou 等人提出了一个基于锁协议(lockset)的方法 HARD[65]. HARD 使用一个 Bloom 过滤器(Bloom filters)的过滤结果(相当于一种 Hash 运算结果)来表示共享变量的锁持有者和申请者，然后通过锁算法来判断潜在的数据冲突. 该算法为了保存申请者，对每条 cache 行增加了一个 16 b 的区域，因此它需要对 cache 设计进行更改. 同 CORD[66]的方法一样，cache line 的伪共享也会造成误报.

在 2009 年的 ISCA 会议上，Muzahid 等人提出了一种基于 Address Signature 和 Happens-Before 的方法 SigRace[67]. 该方法解决了之前的方法要修改 cache 的结构和一致性协议的缺点. 并且支持出错后通过重新执行(re-Execution)，从而进行程序自

我修复.而在2009年另一个会议(IEEE International Symposium on Parallel & Distributed Processing)上,Gupta等人提出一种基于事务内存(transactional memory)的数据竞争检测方法RaceTM[68],该方法通过重用现有已有的硬件设计实现,不需要专门的硬件支持,但它需要依赖于事物存储模型.

同样在2009年,Nistor等人提出一种比较新颖的办法Light64[69].他们观察到,当两个线程存在交叉访问且存在访存顺序翻转时,程序执行的历史记录会产生偏移(deviation).Light64通过硬件把这些历史记录进行Hash得到一个64 b的Hash值,将这Hash值和因果关系(happens-before)状态图的Hash值比较,判断是否出现数据冲突.

6　可调试性设计的总结和趋势展望

可调试性设计由于其对待验证设计的依赖性,从结构设计到验证手段都不尽相同,但是其设计的指导思想,即提高设计状态的观测性和调试效率是相同的.本文总结的几种可调试性设计和相应的设计方法具有普遍性.

随着集成电路设计的进步,对可调试性设计的研究越来越深入,最近几年在大规模集成电路可调试性设计方面的顶级会议发表的相关研究文章日渐增多,可调试性设计的研究逐渐向系统化的方向进步,并且对并行软件调试支持也开始出现,学术界提出了一些具有通用意义的模型,一些工作对如何依靠可调试性设计来提高调试效率甚至达到自动化,展开了开创性的研究并取得了一定进展.

纵观近年来大规模集成电路的可调试设计研究,今后该领域的进一步研究可能会主要集中在如下几个方向:

1) 可调试性设计验证平台.由于电路功能越来越先进,设计越来越复杂,对验证向量的要求也越来越复杂和灵活,这就要求验证平台对真实环境的模拟更全面,验证向量更精确且可重现.而对错误控制的追求必然导致各种更加精密和全面的错误重现手段的涌现,推动可调试性设计的进一步完善.调试自动化的研究.对隔离错误的需求必然导致各种自动化分析算法和流程的出现,以及相应结构设计上的支持,将各种调试方法和可调试性设计的结构相结合,提高调试的自动化水平,这是提高调试工作生产率的必然要求.

2) 确定性重放以及数据冲突检测的可重用设计.这方面研究的意义主要体现在两个方面:一是确定性重放和数据检测又比较大的相通之处,如对共享存储导致的依赖和竞争,可以通过统一设计一个机制来完成这两个任务;另一方面是如何更好地利用已有的硬件资源,尽量减少硬件结构的改动,在重用已有硬件设计基础上辅助并行软件调试.

3) 各种高效的硬件结构.开销和复杂度的要求也会使得结构设计向简单高效化发展,相应地,围绕设计的外部处理软硬件也会向全面、高效化发展.

4) 系统的设计理论.规范化一直是电路设计的准则之一,对可调试性设计的移植性、易维护性、正确性的要求也需要有规范的指导,对各种规范化原则的研究也将成为可调试性设计的发展趋势之一.

参　考　文　献

[1] Josephson D. The good, the bad, and the ugly of silicon debug [C] //Proc of Design Automation Conference 2006. New York: ACM, 2006: 3-6

[2] Paniccia M, Eiles T M, Rao V R M, et al. Novel optical probing technique for flip chip packaged microprocessors [C] //Proc of Int Test Conf 1998. Piscataway, NJ: IEEE, 1998: 740-747

[3] Rowlette J A, Eiles T M. Critical timing analysis in microprocessors using near-IR laser assisted device alteration (LADA)[C] //Proc of Int Test Conf 2003. Piscataway, NJ: IEEE, 2003: 264-273

[4] Knebel D, Sanda P, Manus M M C, et al. Diagnosis and characterization of timing-related defects by time-dependent light emission [C] //Proc of Int Test Conf 1998. Piscataway, NJ: IEEE, 1998: 733-739

[5] Livengood R H, Medeiros D. Design for (physical) debug for silicon microsurgery and probing of flip-chip packaged integrated circuits [C] //Proc of Int Test Conf 1999. Piscataway, NJ: IEEE, 1999: 877-882

[6] Josephson D D. The manic depression of microprocessor debug [C] //Proc of Int Test Conf 2002. Piscataway, NJ: IEEE, 2002: 657-663

[7] Beers R. Pre-RTL formal verification: An Intel experience [C] //Proc of Design Automation Conference 2008. New York: ACM, 2008: 806-811

[8] Ng K. Challenges in using system-level models for RTL verification [C] //Proc of Design Automation Conf 2008. New York: ACM, 2008: 812-815

[9] Silas I, Frumkin I, Hazan E, et al. System-level validation of the Intel pentium M processor [J] Intel Technical Journal, 2003, 7(2): 1-9

[10] Joint Test ActionGroup (JTAG). IEEE Standard Test Access Port and Boundary-Scan Architecture [S]. Piscataway, NJ: IEEE, 1990

[11] P&E. BDM Multilink Rev B Technical Summary Document #PE3306, Version1.2 [EB/OL]. (2008-08-06) [2009-06-01]. http://www.pemicro.com

[12] Kinra A. Searching for common ground between low cost and high performance ATE systems [C] //Proc of Int Test Conf 2001. Piscataway, NJ: IEEE, 2001: 1152

[13] Yang B, Wu K, Karri R. Secure scan: A design-for-test architecture for crypto chips [C] //Proc of Design Automation Conf 2005. New York: ACM, 2005: 2287-2293

[14] Pyron C. Scan and BIST can almost achieve test quality levels [C] //Proc of Int Test Conf 2002. New York: IEEE, 2002: 1196

[15] Shimizu K, Gupta S, Koyama T, et al. Verification of the cell broadband engine/spl trade/processor [C] //Proc of Design Automation Conf 2006. New York: ACM, 2006: 338-343

[16] Kao C, Huang I, Lin C. An embedded multi-resolution AMBA trace analyzer for microprocessor-based SoC integration [C] //Proc of Int Test Conf 2007. Piscataway, NJ: IEEE, 2007: 477-482

[17] Caty O, Dahlgren P, Bayraktaroglu I. Microprocessor silicon debug based on failure propagation tracing [C] //Proc of Int Test Conf 2005. Piscataway, NJ: IEEE, 2005: 284-293

[18] Baker B, Beers J V. Shmoo plotting: The black art of IC testing [J]. IEEE Design and Test of Computers, 1997, 14(3): 90-97

[19] Huott W, McManus M, Knebel D, et al. The attack of the 'Holey Shmoos': A case study of advanced DFD and picosecond imaging analysis (PICA) [C] //Proc of Int Test Conf 1999. Piscataway, NJ: IEEE, 1999: 883-891

[20] Wang N J, Quek J, Rafacz T M, et al. Characterizing the effects of transient faults on a high-performance processor pipeline [C] //Proc of Int Conf on Dependable Systems and Networks 2004. Piscataway, NJ: IEEE, 2004: 61-72

[21] Sarangi S, Narayanasamy S, Carneal B, et al. Patching processor design errors with programmable hardware [J]. IEEE Micro, 2007, 27(1): 12-25

[22] Park S, Mitra S. IFRA: Instruction footprint recording and analysis for post-silicon bug localization in processors [C] //Proc of Design Automation Conf 2008. New York: ACM, 2008: 373-378

[23] Wang Huandong, Gao Xiang, Chen Yunji, et al. Interconnection of Godson-3 multi-core processor [J]. Journal of Computer Research and Development, 2008, 45(12): 6-16 (in Chinese)
(王焕东，高翔，陈云霁，等. 龙芯3号互联系统的设计与实现[J]. 计算机研究与发展，2008，45(12)：6-16)

[24] Goossens K, Vermeulen B, Steeden R, et al. Transaction-based communication-centric debug [C] //Proc of Int Symp on Networks-on-Chips 2007. Piscataway, NJ: IEEE, 2007: 95-106

[25] Vermeulen B, Goossens K, Steeden R, et al. Communication-centric SoC debug using transactions [C] //Proc of European Test Symp 2007. Piscataway, NJ: IEEE, 2007: 69-76

[26] Abramovici M, Goossens K, Vermeulen B, et al. You can catch more bugs with transaction level honey [C] //Proc of Int Conf on Hardware-Software Codesign and System Synthesis 2008. New York: ACM, 2008: 121-124

[27] Vermeulen B, Goossens K, Umrani S. Debugging Distributed-shared-memory communication at multiple granularities in networks on chip [C] //Proc of Int Symp on Networks-on-Chips 2008. Piscataway, NJ: IEEE, 2008: 3-12

[28] DeOrio A, Wagner I, Bertacco V. Dacota: Post-silicon validation of the memory subsystem in multi-core designs [C] //Proc of Int Symp on High-Performance Computer Architecture 2009. Piscataway, NJ: IEEE 2009: 405-416

[29] Abramovici M, Bradley P, Dwarakanath K N, et al. A reconfigurable design-for-debug infrastructure for SoCs [C] //Proc of Design Automation Conf 2006. New York: ACM, 2006: 7-12

[30] Carbine A, Feltham D. Pentium pro processor design for test and debug [J]. IEEE Design and Test of Computers, 1998, 15(3): 77-82

[31] Pyron C, Bangalore R, Belete D, et al. Silicon symptoms to solutions: Applying design for debug techniques [C] //Proc of Int Test Conf 2002. Piscataway, NJ: IEEE, 2002: 664-672

[32] Chen Y, Chen T, Hu W. Global clock, physical time order and pending period analysis in multiprocessor systems [EB/OL]. [2009-09-03]. http://arxiv.org/abs/0903.496

[33] Chen Y, Lü Y, Hu W, et al. Fast complete memory consistency verification [C] //Proc of Int Symp on High-Performance Computer Architecture 2009. Piscataway, NJ: IEEE, 2009: 381-392

[34] Ko H F, Nicolici N. On automated trigger event generation in post-silicon validation [C] //Proc of Design, Automation, and Test in Europe 2008. Piscataway, NJ: IEEE, 2008: 256-259

[35] Jin H, Somenzi F. Prime clauses for fast enumeration of satisfying assignments to boolean circuits [C] //Proc of Design Automation Conf 2005. New York: ACM, 2005: 750-753

[36] Chen Yunji, Zhang Jian, Shen Haihua, et al. ASAT-based arithmetic circuit bug-hunting method [J]. Chinese Journal of Computers, 2007, 30(12): 2082-2089 (in Chinese)
(陈云霁，张健，沈海华，等. 一种基于SAT的运算电路查错方法[J]. 计算机学报，2007，30(12)：2082-2089)

[37] Sabharwal A, Ansótegui C, Gomes C P, et al. QBF modeling: Exploiting player symmetry for simplicity and efficiency [C] //Proc of Int Conf on Theory and Applications of Satisfiability Testing 2006. Piscataway, NJ: IEEE, 2006: 382-395

[38] Hsieh M, Huang C. An embedded infrastructure of debug and trace interface for the DSP platform [C] //Proc of Design Automation Conf 2008. New York: ACM, 2008: 866-871

[39] Ziv J, Lempel A. A universal algorithm forsequential data compression [J]. IEEE Trans on Information Theory, 1977, 23(3): 337-343

[40] Lin Y, Shiue W, Huang I. A multi-resolution AHB bus tracer for real-time compression of forward/backward traces in a circular buffer [C] //Proc of Design Automation Conf 2008. New York: ACM, 2008: 862-865

[41] Anis E, Nicolici N. Interactive presentation: Low cost debug architecture using lossy compression for silicon debug [C] //Proc of Design, Automation, and Test in Europe 2007. Piscataway, NJ: IEEE, 2007: 225-230

[42] Shen H, Chen Y, Huang J. EmGen: An automatic test-program generation tool for embedded IP cores [C] //Proc of Int Conf on Embedded Software and Systems 2004. Piscataway, NJ: IEEE, 2004: 528-535

[43] Oh N, Mitra S, McCluskey E J. ED4I: Error detection by diverse data and duplicated instructions [J] IEEE Trans on Computers, 2002, 51(2): 180-199

[44] Dahlgren P, Dickinson P, Parulkar I. Latch divergence in microprocessor failure analysis [C] //Proc of Int Test Conf 2003. Piscataway, NJ: IEEE, 2003: 755-763

[45] Chen K, Malik S, Patra P. Runtime validation of memory ordering using constraint graph checking [C] //Proc of Int Symp on High-Performance Computer Architecture 2008. Piscataway, NJ: IEEE, 2008: 415-426

[46] Mitra S, Karnik T, Seifert N, et al. Logic soft errors in sub-65nm technologies design and CAD challenges [C] //Proc of Design Automation Conf 2005. New York: ACM, 2005: 2-4

[47] Sanda P N, Kellington J W, Kudva P, et al. Soft-error resilience of the IBM POWER6 processor [J]. IBM Journal of Research and Development, 2008, 52(3): 275-286

[48] Nanshi N, Somenzi F. Guiding simulation with increasingly refined abstract traces [C] //Proc of Design Automation Conf 2006. New York: ACM, 2006: 737-742

[49] Ko H F, Nicolici N. Automated trace signals identification and state restoration for improving observability in post-silicon validation [C] //Proc of Design, Automation, and Test in Europe 2008. Piscataway, NJ: IEEE, 2008: 1298-1303

[50] Bhansali S, Chen W, Jong S D, et al. Framework for instruction-level tracing and analysis of programs [C] //Proc of Int Conf on Virtual Execution Environments 2006. New York: ACM, 2006: 154-163

[51] Pereira C, Patil H, Calder B. Reproducible simulation of multi-threaded workloads for architecture design exploration [C] //Proc of IEEE Int Symp on Workload Characterization 2008. Piscataway, NJ: IEEE, 2008: 173-182

[52] Xu M, Bodik R, Hill M. A flight data recorder for enabling full-system multiprocessor deterministic replay [C] //Proc of Int Symp on Computer Architecture 2003. Piscataway, NJ: IEEE, 2003: 122-135

[53] Lamport L. Time, clocks and the ordering of events in a distributed system [J]. Communications of the ACM, 1978, 21(7): 558-565

[54] Lamport L. How to make a multiprocessor computer that correctly executes multiprocess programs [J]. IEEE Trans on Computers, 1979, 28(9): 690-691

[55] Xu M, Hill M D, Bodik R. A regulated transitive reduction (rtr) for longer memory race recording [C] //Proc of Int Conf on Architectural Support for Programming Languages and Operating Systems 2006. New York: ACM, 2006: 49-60

[56] Narayanasamy S, Pokam G, Calder B. BugNet: Continuously recording program execution for deterministic replay debugging [C] //Proc of Int Symp on Computer Architecture 2005. Piscataway, NJ: IEEE, 2005: 284-295

[57] Montesinos P, Ceze L, Torrellas J. Delorean: Recording and deterministically replaying shared-memory multiprocessor execution efficiently [C] //Proc of Int Symp on Computer Architecture 2008. Piscataway, NJ: IEEE, 2008: 289-300

[58] Hower D R, Hill M D. Rerun: Exploiting episodes for lightweight memory race recording [C] //Proc of Int Symp on Computer Architecture 2008. Piscataway, NJ: IEEE, 2008: 265-276

[59] Narayanasamy S, Pereira S, Calder B. Recording shared memory dependencies using strata [C] //Proc of Architectural Support for Programming Languages and Operating Systems 2006. New York: ACM, 2006: 229-240

[60] Lee D, Narayanasamy S, Said M, et al. Offline symblolic analysis for multi-processor execution replay [C] //Proc of IEEE/ACM Int Symp on Microarchitecture 2009. New York: ACM, 2009: 564-575

[61] Dutertre B, Moura L. A fast linear-arithmetic solver for DPLL (T) [C] //Proc of Computer-Aided Verification Conf 2006. Piscataway, NJ: IEEE, 2006: 81-94

[62] Montesinos P, Hicks M, King S T, et al. Capo: A software-hardware interface for practical deterministic multiprocessor replay [C] //Proc of Architectural Support for Programming Languages and Operating Systems 2009. New York: ACM, 2009: 73-84

[63] Savage S, Burrows M, Nelson G, et al. Eraser: A dynamic data race detector for multithreaded programs [J]. ACM Trans on Computer Systems, 1997, 15(4): 391-411

[64] Prvulovic M, Torrellas J. ReEnact: Using thread-level speculation mechanisms to debug data races in multithreaded codes [C] //Proc of Int Symp on Computer Architecture 2003. Piscataway, NJ: IEEE, 2003: 110-121

[65] Zhou P, Teodorescu R, Zhou Y. HARD: Hardware-assisted lockset-based race detection [C] //Proc of Int Symp High Performance Computer Architecture 2007. Piscataway, NJ: IEEE, 2007: 121-132

[66] Prvulovic M. CORD: Cost-effective (and nearly overhead-free) order-recording and data race detection [C] //Proc of Int Symp High-Performance Computer Architecture 2006. Piscataway, NJ: IEEE, 2006: 232-243

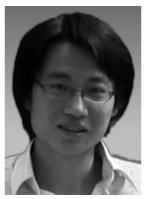
[67] Muzahid A, Suárez D, Qi S, et al. SigRace: Signature-based data race detection [C] //Proc of Int Symp on Computer Architecture 2009. New York: ACM, 2009: 337-348

[68] Gupta S, Sultan F, Cadambi S R, et al. Using hardware transactional memory for data race detection [C] //Proc of IEEE Int Symp on Parallel & Distributed Processing 2009. Piscataway, NJ: IEEE, 2009: 1-11

[69] Nistor A, Marinov D, Torrellas J. Light64: Lightweight hardware support for race detection during systematic testing of parallel programs [C] //Proc of IEEE/ACM Int Symp on Microarchitecture 2009. New York: ACM, 2009: 541-552

Qian Cheng, born in 1985. Received his PhD degree in computer science from the Institute of Computing Technology (ICT), the Chinese Academy of Sciences (CAS), Beijing, China. Assistant engineer at ICT. His current research interests include micro architecture and post-silicon verification.

Shen Haihua, born in 1971. Received her PhD degree from Tsinghua Univesity in 2002. Associate professor, and master supervisor. Senior member of China Computer Federation. Her current research interests include computer architecture, microprocessor's design and verification.



Chen Tianshi, born in 1985. Received his PhD degree in computer science from the University of Science and Technology of China in 2010. Currently assistant professor at the Institute of Computing Technology, the Chinese Academy of Sciences, Beijing, China. His research interests include parallel computing and evolutionary computation.

Chen Yunji, born in 1983. Received his PhD degree in computer science from the Institute of Computing Technology, the Chinese Academy of Sciences, Beijing, China, in 2007. Since 2007, associate professor at ICT. His main research interests include parallel computing, micro architecture, and hardware verification.