软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(6):2453−2476 [doi: 10.13328/j.cnki.jos.007328] [CSTR: 32375.14.jos.007328]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

智能化芯片设计程序测试研究综述*

李晓鹏[1], 闫 明[1], 樊兴宇[1], 唐振韬[2], 开昱雄[2], 郝建业[1,2], 袁明轩[2], 陈俊洁[1]

[1](天津大学 智能与计算学部, 天津 300350)

[2](华为技术有限公司 诺亚方舟实验室, 广东 深圳 518129)

通信作者: 陈俊洁, E-mail: junjiechen@tju.edu.cn

摘 要: 在当今智能化的时代背景下, 芯片作为智能电子设备的核心组件, 在人工智能、物联网、5G 通信等诸多领域发挥着关键作用, 保障芯片的正确性、安全性和可靠性至关重要. 在芯片的开发流程中, 开发人员首先需要利用硬件描述语言, 将芯片设计实现成软件形式 (即芯片设计程序), 然后再进行物理设计并最终流片 (即生产制造). 作为芯片设计制造的基础, 芯片设计程序的质量直接影响了芯片的质量. 因此, 针对芯片设计程序的测试具有重要研究意义. 早期的芯片设计程序测试方法主要依赖开发人员人工设计测试用例来测试芯片设计程序, 往往需要大量的人工成本和时间代价. 随着芯片设计程序复杂度的日益增长, 诸多基于仿真的自动化芯片设计程序测试方法被提出, 提升了芯片设计程序测试效率及有效性. 近年来, 越来越多的研究者致力于将机器学习、深度学习和大语言模型 (LLM) 等智能化方法应用于芯片设计程序测试领域. 调研 88 篇智能化芯片设计程序测试相关的学术论文, 从测试输入生成、测试预言构造及测试执行优化这 3 个角度对智能化芯片设计程序测试已有成果进行整理归纳, 重点梳理芯片设计程序测试方法从机器学习阶段、深度学习阶段到大语言模型阶段的演化, 探讨不同阶段方法在提高测试效率和覆盖率、降低测试成本等方面的潜力. 同时, 介绍芯片设计程序测试领域的研究数据集和工具, 并展望未来的发展方向和挑战.

关键词: 芯片设计程序测试; 大语言模型; 测试用例生成

中图法分类号: TP311

中文引用格式: 李晓鹏, 闫明, 樊兴宇, 唐振韬, 开昱雄, 郝建业, 袁明轩, 陈俊洁. 智能化芯片设计程序测试研究综述. 软件学报, 2025, 36(6): 2453–2476. http://www.jos.org.cn/1000-9825/7328.htm

英文引用格式: Li XP, Yan M, Fan XY, Tang ZT, Kai SX, Hao JY, Yuan MX, Chen JJ. Survey on Testing of Intelligent Chip Design Program. Ruan Jian Xue Bao/Journal of Software, 2025, 36(6): 2453–2476 (in Chinese). http://www.jos.org.cn/1000-9825/7328.htm

Survey on Testing of Intelligent Chip Design Program

LI Xiao-Peng[1], YAN Ming[1], FAN Xing-Yu[1], TANG Zhen-Tao[2], KAI Shi-Xiong[2], HAO Jian-Ye[1,2], YUAN Ming-Xuan[2], CHEN Jun-Jie[1]

[1](College of Intelligence and Computing, Tianjin University, Tianjin 300350, China)

[2](Noah's Ark Laboratory, Huawei Technologies Co. Ltd., Shenzhen 518129, China)

Abstract: In the current intelligent era, chips, serving as the core components of intelligent electronic devices, play a critical role in multiple fields such as artificial intelligence, the Internet of Things, and 5G communication. It is of great significance to ensure the correctness, security, and reliability of chips. During the chip development process, developers first need to implement the chip design into a software form (i.e., chip design programs) by using hardware description languages, and then conduct physical design and finally tape-out (i.e., production and manufacturing). As the basis of chip design and manufacturing, the quality of the chip design program directly

* 基金项目: 国家自然科学基金 (62322208); 华为高校合作项目
本文由"大模型下的软件质量保障"专题特约编辑王赞教授、王莹副教授、陈碧欢副教授、姚远副教授、张敏灵教授推荐.
收稿时间: 2024-08-26; 修改时间: 2024-10-14; 采用时间: 2024-11-25; jos 在线出版时间: 2024-12-10
CNKI 网络首发时间: 2025-02-26

impacts the quality of the chips. Therefore, the testing of chip design programs is of important research significance. The early testing methods for chip design programs mainly depend on the test cases manually designed by developers to test the chip design programs, often requiring a large amount of manual cost and time. With the increasing complexity of chip design programs, various simulation-based automated testing methods have been proposed, improving the efficiency and effectiveness of chip design program testing. In recent years, more and more researchers have been committed to applying intelligent methods such as machine learning, deep learning, and large language models (LLMs) to the field of chip design program testing. This study surveys 88 academic papers related to intelligent chip design program testing, and sorts and summarizes the existing achievements in intelligent chip design program testing from three perspectives: test input generation, test oracle construction, and test execution optimization. It focuses on the evolution of chip design program testing methods from the machine learning stage to the deep learning stage and then to the large language model stage, exploring the potential of different stages' methods in improving testing efficiency and coverage, as well as reducing testing costs. Additionally, it introduces research datasets and tools in the field of chip design program testing and envisions future development directions and challenges.

Key words: testing of chip design program; large language model (LLM); test case generation

芯片, 通常亦被称为集成电路 (integrated circuit, IC), 是现代电子设备的重要组成部分, 其支撑着顶层应用程序的正常运行. 在当今智能化的背景下, 芯片已被应用于诸多应用场景中, 并起到关键的作用. 在 5G 通信应用中, 芯片确保了不同设备通信过程中数据的高速传输和低时延的可靠连接; 在自动驾驶场景下, 芯片支持着顶层系统的实时信息感知及智能化决策; 在大语言模型场景下, 芯片支持着人工智能算法及神经网络的高速复杂运算. 高性能的复杂芯片现如今已成为推动创新和实现技术落地的重要驱动力. 因此, 保证芯片的正确性、安全性和可靠性至关重要.

通常来说, 芯片的制造过程涵盖规格说明、架构设计、逻辑设计、物理设计及流片 (生产制造) 这 5 个主要环节. 芯片开发人员首先需要针对芯片功能需求编写芯片设计规格说明, 设计芯片的整体功能架构及模块组成. 在逻辑设计阶段, 开发人员使用硬件描述语言将芯片设计实现成软件形式 (即芯片设计程序), 基于芯片设计程序生成门级网表 (gate-level netlist), 用于指导后续电路的物理设计及芯片布局布线, 并最终将芯片设计流片制造. 由于芯片一经流片生产无法修改, 芯片制造中任意环节的错误都会带来难以承受的经济损失, 甚至带来灾难性的后果. 例如, Intel 公司 1994 年发布的 Pentium 处理器存在浮点除法运算相关的设计缺陷, 导致某些除法运算结果不准确. 最终 Intel 召回了受影响的处理器, 该缺陷导致了高达 4.75 亿美元的经济损失. 作为芯片制造的早期环节, 芯片设计 (design under test, DUT) 程序的正确性从根本上影响了芯片的质量. 因此, 在软件层面保障芯片质量 (即保障芯片设计程序质量) 至关重要.

芯片设计程序的质量保证方法主要包括两类: (1) 基于形式化的验证方法, 此类方法依赖数学工具、模型构造来确保芯片设计程序的正确性; (2) 基于仿真的测试方法, 此类方法生成芯片设计程序的输入, 借助数字仿真工具动态执行程序来模拟芯片真实使用时的行为, 确保芯片设计程序的正确性. 随着芯片设计程序复杂性与日俱增, 基于形式化的验证方法面临效率较低、计算资源需求高等诸多问题. 因此, 研究人员逐渐聚焦于基于仿真的芯片设计程序测试方法, 生成有效的测试输入, 测试芯片设计程序的正确性. 随着机器学习技术、深度学习技术乃至大语言模型技术的快速发展, 越来越多的研究人员提出了更多智能化的芯片设计程序测试技术, 以此提升芯片设计程序的测试效率及充分性.

目前, 已有部分研究者从不同角度对芯片设计程序质量保证方法进行了总结. Wu 等人[1]从形式化验证和测试两方面对基于机器学习的质量保障方法在芯片设计程序方面的应用进行了总结. Jayasena 等人[2]从不同技术路线角度出发, 对芯片设计程序定向测试生成研究进行总结. Ismail 等人[3]调研了机器学习方法在芯片设计程序测试输入生成、测试覆盖收集、错误检测和定位及数字仿真建模等方面的应用. Ioannides 等人[4]对机器学习方法在基于覆盖率引导的芯片设计程序测试输入生成方面的应用进行了总结. 与已有综述文章相比, 本文对基于仿真的芯片设计程序智能化测试方法研究进展进行归纳总结, 涵盖芯片设计程序测试输入生成、不同测试预言构造和测试执行优化这 3 大关键测试环节, 同时梳理已有智能化测试方法在机器学习、深度学习以及大语言模型等不同技术阶段的发展演化过程; 整理 40 个芯片设计程序测试数据集和 19 个测试工具, 并针对该领域未来研究方向进行展望,

为该领域的研究者提供参考.

本文所提芯片设计程序智能化测试研究框架如图 1 所示. 芯片设计程序智能化测试输入生成涵盖测试度量指标和基于覆盖结果反馈、基于输入特征提取和基于语义结构分析的测试输入生成方法; 测试预言构造包括基于差分测试 (参考模型) 和基于蜕变测试两种方式, 以此判断不同输入下芯片设计程序行为是否符合预期; 测试执行优化是指针对已有测试输入, 通过智能化的测试用例选择及测试排序方法, 确定测试集合内容及测试执行顺序, 以此提升整体测试流程效率; 测试数据集和工具涵盖现阶段智能化测试研究中常用的芯片设计程序项目及开源的测试工具等.

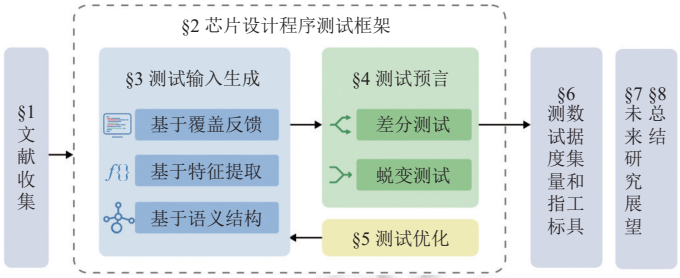


图 1 芯片设计程序智能化测试研究框架

本文第 1 节对文献收集和筛选的过程和标准进行介绍. 第 2 节对芯片设计程序测试流程和框架进行介绍. 第 3 节对芯片设计程序测试输入生成方法进行分类, 并总结不同方法类型在机器学习、深度学习以及大语言模型不同阶段的演化. 第 4 节对芯片设计程序测试过程中常见测试预言进行归纳. 第 5 节介绍当前芯片设计程序智能化测试领域的测试优化方法. 第 6 节对芯片设计程序测试领域开源数据集和测试工具进行总结. 第 7 节展望智能化芯片设计程序测试未来的发展方向和挑战. 第 8 节对全文进行总结, 归纳梳理智能化芯片设计程序测试方法的应用现状和综述结论.

1 文献收集

本文开展了系统化的文献收集和整理过程, 确保对芯片设计程序智能化测试的发展和现状进行全面归纳总结. 需要说明的是, 在软件工程领域, 验证一词通常代指针对软件的形式化验证, 软件测试则是指通过动态生成输入的方式来揭露软件缺陷; 而在芯片设计程序领域, 基于形式化和基于仿真的方式均被统称为验证[5,6]. 本文综合考虑上述两种概念习惯, 使用验证 (verification) 和测试 (testing) 两种关键词对论文进行检索收集, 同时沿用软件工程领域概念用法对所调研论文的技术路线及方案进行阐述. 具体来说, 本文首先以“chip design testing/verification”“hardware testing/verification”“functional verification”“functional coverage”“芯片 (设计) 测试/验证”等作为关键词, 在 Google Scholar、ACM Digital Library、DBLP、IEEE Xplore、CNKI 等重要搜索引擎和数据库中进行检索. 在检索到的文献中, 本文筛选了和智能化芯片设计程序测试研究相关的文献, 重点关注采用机器学习方法、深度学习方法和大语言模型方法等智能化芯片设计程序测试方法. 对于采用非智能化的传统测试方法和形式化验证方法的文献, 本文则筛选了部分较有代表性的文献于第 2 节简要介绍, 并不作为主要关注文献而进行详细综述. 此外, 部分检索到的文献仅关注芯片设计而非测试, 或仅关注芯片物理设计 (硬件层面) 测试, 由于本文主要关注芯片设计程序 (软件层面) 测试, 因此该类别的文献不在本文综述范畴.

通过对检索到的文献进行筛选, 并根据文献的引用和被引进行补充, 最终确定了 88 篇与本文研究问题相关的文献. 关于芯片设计程序测试问题, 在软件工程领域、计算机体系结构领域和芯片 (硬件) 验证领域都有较多研究, 因此本文选择的文献来源不仅包括软件工程领域的会议 (如 ICSE、ISSTA 等), 还包括计算机体系结构领域的会议或期刊 (如 DAC、TCAD、DATE、ICCAD、ASP-DAC 等), 以及芯片 (硬件) 验证领域的会议或期刊 (如 DVCon U.S. 等). 调研的文献在不同年份的发表数量分布如图 2 所示, 其中有 26 篇来自 CCF-A/B 类会议或期刊, 10 篇来自 CCF-C 类会议或期刊, 35 篇来自芯片设计和验证领域会议或期刊, 13 篇来自 arXiv.

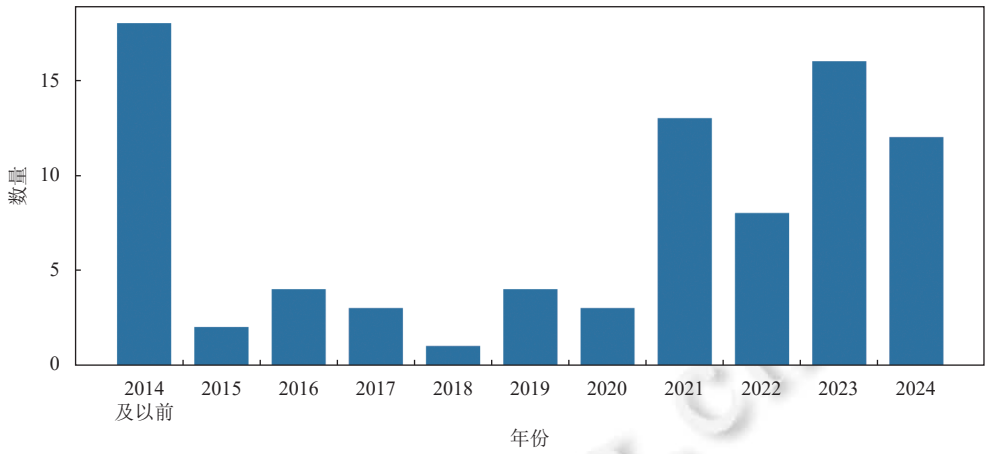


图 2 不同年份的文献数量分布

2 芯片设计程序测试框架

如图 3 所示, 芯片设计制造过程主要可以分为 5 个阶段: 规格说明、架构设计、逻辑设计、物理设计和流片制造. 其中, 在规格说明和架构设计阶段, 芯片设计人员使用高抽象层次的事务级模型来描述芯片的功能和行为, 主要关注系统的架构设计和事务级别功能的验证. 在逻辑设计阶段, 芯片开发人员将芯片设计的事务级模型转化为更详细的软件形式 (即芯片设计程序), 称为寄存器传输级别 (register-transfer level, RTL) 设计. 芯片逻辑设计是芯片设计的核心阶段, 其详细描述了芯片的功能模块和数据路径, 主要关注寄存器之间的数据传输及运算. 开发人员通常使用诸如 Verilog、SystemVerilog、VHDL 和 Chisel[7]等硬件描述语言 (hardware description language, HDL) 来编写芯片设计程序源代码, 实现对芯片设计的详细描述, 并进行功能仿真和测试, 确保芯片设计程序符合规格要求. 在物理设计和流片制造阶段, 开发人员将芯片设计程序转化为实际的逻辑门和连线布局, 最终制造为实际可用的芯片产品.

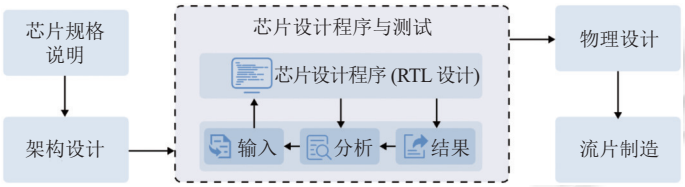


图 3 芯片设计和制造的一般流程

由于芯片制品一经流片无法再做任何修改, 因此保障芯片设计制造各个环节的正确性至关重要. 作为芯片生产周期的前置环节, 芯片设计的正确性从根本上影响了芯片的质量. 现阶段, 已有大量针对芯片设计的质量保证工作被提出. 在此类工作中, 芯片设计描述通常以 HDL 软件代码形式存在 (称为芯片设计程序). 因此, 多数传统软件质量保证方法可被迁移到芯片设计代码质量保证中, 检验芯片设计的正确性和可靠性.

在芯片设计程序质量保障方法中, 早期方法主要依赖开发人员人工设计测试用例来测试芯片设计程序, 往往需要大量的人工成本和时间代价, 且测试的准确率和充分性较低. 随着芯片设计程序的复杂度日益增长, 此类方法难以有效测试更大规模的芯片设计程序. 测试人员开始将自动化方法应用于芯片设计程序测试过程, 典型方法有基于形式化的验证方法和基于仿真的测试方法等. 基于形式化的验证方法的核心思想是使用数学方法验证芯片设计程序的正确性, 而无需依赖测试用例的构建, 通过逻辑推理所有可能的设计状态来验证设计的正确性. Grimm 等人[8]于 2018 年对芯片设计程序的形式化验证方法做出了总结, 并对比了不同测试方法的效率和准确性. 芯片设计

程序的形式化验证方法主要有定理证明 (theorem proving)[9]、等价性检查 (equivalence checking)[10]、模型检查 (model checking)[11,12]和符号执行 (symbolic execution)[13,14]等. 形式化验证方法虽然无需构建和维护大量测试用例, 但需要更多专业领域知识和人工成本, 且在大规模芯片设计程序上的资源开销较大、时间成本较高.

与基于形式化验证的方法不同, 基于仿真的测试方法通常使用构建的测试输入对芯片设计程序进行仿真, 模拟芯片在实际使用场景中的行为, 检查芯片的仿真结果来判断其功能是否符合规范检测以及揭示芯片设计程序的缺陷, 测试效率更高, 是目前工业界常用的芯片设计程序测试方法. 具体来说, 仿真测试可被细分为功能仿真和时序仿真, 其中功能仿真通过模拟芯片的输入事件或指令来检测芯片设计程序功能是否符合预期, 常用于处理器和片上系统 (system on chip, SoC) 设计的质量保证; 时序仿真用于验证芯片设计程序时序行为的正确性, 包括时钟频率、延迟和建立保持时间等. 传统的基于仿真的芯片设计程序测试方法可被分为两类: 1) 基于随机采样的测试生成; 2) 基于覆盖率引导的测试输入生成. 基于随机采样的测试输入生成方法可进一步细分成纯随机测试输入生成以及约束随机测试生成 (constraint random verification, CRV), 其中前者对芯片设计程序输入进行随机生成而不添加任何限制, 后者则在随机生成的基础上增加约束条件, 实现更有针对性的测试用例, 提高测试输入的合法性和有效性. 目前诸如 Synopsys VCS[15]等商业的电子设计自动化 (electronic design automation, EDA) 软件均对随机仿真测试方法提供了支持, 许多特定领域的开源软件也提供了随机测试生成的解决方案, 如针对 RISC-V 指令集处理器测试的随机指令生成器 RISCV-DV[16]. 目前, 诸多基于覆盖率引导的测试生成 (coverage-directed generation, CDG) 方法被提出, 以此来生成测试充分性更高的芯片设计程序测试输入[17−19].

此外, 许多研究采用传统软件测试方法 (如模糊测试方法) 基于覆盖率指标进行测试输入生成. 例如, Laeufer 等人[20]将模糊测试技术应用于芯片设计程序测试中, 以多路复用器作为覆盖指标, 首次提出了基于变异的覆盖率引导芯片设计程序测试的方法 RFUZZ, 涵盖位翻转、增减随机数、重写、删除和复制等诸多变异算子. Canakci 等人[21]提出了一种针对芯片设计程序的特定目标进行定向覆盖的方法 DirectFuzz, 首次将定向灰盒测试引入芯片设计程序测试, 其只考虑处理器设计的部分组件, 约减覆盖目标范围, 相比关注整体处理器设计的 RFUZZ 提高了测试效率. Trippel 等人[22]将软件模糊器及软件模型应用到芯片设计程序模糊测试场景中, 通过将芯片设计转换为软件模型, 使用软件模糊器跟踪芯片设计程序覆盖范围, 并将模糊器生成的测试用例转换为适合芯片设计程序执行的形式来有效地对芯片设计程序进行仿真. Hur 等人[23]提出了寄存器覆盖指标, 并以此为引导目标提出了一种模糊测试的方法 DifuzzRTL, 重点关注芯片设计程序中寄存器的状态转换. Xu 等人[24]提出了一种利用寄存器覆盖率等运行时信息对测试输入进行动态变异的方法 MorFuzz, 通过设计测试输入模板、运行时对输入指令进行变异、按策略对待测设计和参考模型进行状态同步等方式, 提高了测试效率和覆盖率. Canakci 等人[25]提出了控制状态寄存器 (control and status register, CSR) 覆盖指标, 并提出了一种将其用于指导模糊器生成测试的方法 ProcessorFuzz, 核心思想是监测 CSR 状态的转换来发现新的处理器状态, 并使模糊器生成可以探索新的处理器状态的测试输入, 提高测试效率.

近年来, 随着智能化技术的发展, 机器学习方法 (贝叶斯网络、遗传算法等)、深度学习方法 (人工神经网络、循环神经网络等) 和大语言模型方法被应用于芯片设计程序测试领域, 在提高测试效率和覆盖率、优化测试过程等任务上表现出较好的性能. 本文主要关注这些基于智能化技术的芯片设计程序测试方法, 梳理已有智能化测试方法研究近况及演化, 并为未来研究提供展望.

3 测试输入生成

测试输入是芯片设计程序测试的重要组成部分, 其质量直接决定了芯片设计程序测试的充分性和有效性. 本文根据测试输入生成方法依据的关注目标和生成策略, 将测试输入生成方法分为基于覆盖结果反馈的测试生成、基于输入特征提取的测试生成和基于语义结构分析的测试生成这 3 类, 分别依据芯片设计程序测试的覆盖率结果、测试输入的分布和属性特征、芯片设计程序和规约的语义结构来生成测试输入. 本节将从这 3 方面来归纳总结芯片设计程序智能化测试生成研究相关进展.

3.1 基于覆盖结果反馈的测试生成

覆盖率是度量芯片设计程序测试充分性的重要指标. 随机测试生成方法往往无法对芯片设计程序进行全面的覆盖, 传统的利用覆盖结果反馈的测试输入生成方法通常使用覆盖率分析工具来分析芯片设计程序中尚未覆盖的情况, 并采用形式化方法构造测试输入[26], 或依据规则生成[27], 这类方法往往依赖大量专家知识和人工经验. 因此, 众多研究者开始将智能化方法应用于覆盖率引导的测试生成过程, 用智能化方法在覆盖率结果和测试输入之间搭建桥梁. 本节首先介绍常用的测试覆盖指标 (第 3.1.1 节), 并以此为基础, 展开对不同基于覆盖引导的测试输入生成方法的讨论 (第 3.1.2–3.1.4 节), 并根据是否采用神经网络、是否采用大语言模型, 将不同方法归类为机器学习方法、深度学习方法和大语言模型方法. 研究者通常使用贝叶斯网络对测试输入和覆盖结果进行建模, 使用遗传算法和粒子群优化算法等启发式搜索方法提高生成测试的效率; 使用强化学习和深度神经网络结合, 提高覆盖范围, 加速覆盖率收敛; 此外, 部分研究人员基于大语言模型学习芯片设计程序结构信息和覆盖率信息, 利用大语言模型的推理和生成能力, 生成测试输入.

3.1.1 测试覆盖指标

测试覆盖指标用于评估测试的充分性, 并可用于指导测试输入生成[28]. 除了和传统软件测试类似的行覆盖、条件覆盖、状态机覆盖等指标 (统称为代码覆盖) 之外, 功能覆盖和断言覆盖等指标[6]也被经常用于衡量芯片设计程序测试的充分性.

- 代码覆盖. 和传统软件测试类似, 代码覆盖用于度量芯片设计程序代码被测试用例执行的程度, 帮助发现未执行的代码[29]. 行覆盖用于度量代码中每条语句是否被执行; 条件覆盖用于度量每个布尔条件的可能值是否被赋值; 路径覆盖用于度量代码中所有可能的路径是否被遍历; 分支覆盖用于度量每个条件分支是否被执行; 有限状态机 (finite state machine, FSM) 覆盖用于度量基于芯片设计程序表征的有限状态机中所有状态和转移是否都被测试. 另外, 切换覆盖 (toggle coverage) 用于度量代码中每个信号的高低电平切换情况, 确保信号在测试过程中经历了高低电平的变化.
- 功能覆盖. 代码覆盖依赖于对芯片设计程序代码的分析, 比较容易收集, 但不足以用来指导测试生成. 达到较高的代码覆盖率并不能说明测试是充分的, 因此功能覆盖同样需要考虑在内. 功能覆盖用于度量芯片设计程序功能被测试用例覆盖的程度, 与代码覆盖不同, 功能覆盖不能直接获得, 它依赖于特定的芯片设计程序信息, 需要开发人员手动设置, 且需要使用 HDL 的特殊结构 (如 SystemVerilog 中的 coverage groups/points)[30]. 通常情况下, 芯片设计程序功能覆盖由覆盖组 (coverage groups) 和覆盖点 (coverage points) 组成[6]. 覆盖组用于组织和管理一组相关的覆盖点, 定义了要度量的功能和行为, 提供了一个结构化的方法来收集和管理覆盖数据, 便于分析和报告. 覆盖点是覆盖组中的基本元素, 表示芯片设计程序中某个特定信号、状态或条件需要被覆盖, 每个覆盖点用于捕获特定变量或表达式的值, 并统计对应信号和条件的各种状态, 以验证芯片设计程序在这些条件下的功能和行为. 在覆盖点内部可以定义覆盖仓 (bins), 通过对其指定关注的值或值范围来统计信号或条件的特定值或值范围出现的次数, 获取更详细的覆盖数据, 帮助分析哪些条件被充分测试.
- 断言覆盖. 为了评估芯片功能测试的充分性, 除了功能覆盖指标外, 还可以使用断言覆盖. 断言覆盖通过判断芯片设计程序代码中断言语句 (assertions) 是否被触发, 来评估芯片设计程序的行为是否符合预期. 其中, 断言触发覆盖用于判断每条断言语句是否被触发; 断言成功/失败覆盖用于判断断言触发时是否成功或失败, 对应断言成立与否. 断言覆盖和功能覆盖类似, 均需要开发人员手动设置, 因此这两种覆盖指标的收集难度较大, 但对于指导测试生成的重要性比代码覆盖更高. Witharana 等人[31]对基于断言的芯片设计程序测试做出了详细总结, 包括自动断言生成方法和触发断言的测试生成方法等.

此外, 部分研究工作定义了多路复用器覆盖[20,21]、寄存器覆盖[23,24]等更加细粒度的度量指标, 这些指标在“指导测试生成的作用”和“容易收集”之间做出了权衡. 总之, 通过综合评估测试输入在代码覆盖、功能覆盖和断言覆盖等指标上的表现, 测试人员可以定量衡量针对芯片设计程序的执行、功能和行为方面测试的充分性和有效性, 保障芯片设计程序的质量和可靠性.

3.1.2 机器学习方法

早期研究工作多基于贝叶斯网络等机器学习方法表示和计算测试输入和覆盖结果的关系, 并用来指导从覆盖结果到测试输入生成的反馈循环. Fine 等人[32]提出了一种覆盖率引导的定向测试生成方法, 通过使用贝叶斯网络表示测试输入和覆盖结果的关系, 并用于指导测试输入生成, 提高了测试覆盖率, 有效解决了传统测试方法的不足. 诸多后续研究工作在贝叶斯网络的基础上进行了改进. Fine 等人[33]将单一覆盖事件通过添加额外覆盖目标组合为结构化覆盖模型 (称为虚拟覆盖模型), 将难覆盖的目标分解为易覆盖的子目标并组合, 提高了贝叶斯网络在难覆盖点上的覆盖能力. Baras 等人[34]利用特征选择提取测试输入子集、利用结构学习自动构建网络结构、利用参数学习调整节点参数, 改进了基于贝叶斯网络的覆盖率引导测试生成过程, 减少了对领域知识的需求, 提高了从覆盖数据到测试生成器的反馈循环的自动化.

遗传算法等启发式搜索算法常被用于提高测试输入生成过程的效率和覆盖率, 其基本思想是以覆盖结果作为适应度函数, 通过交叉变异产生测试输入, 实现测试覆盖率最大化. Squillero 等人[35]将遗传算法应用于微处理器的测试生成任务, 提出了一种用于生成汇编程序的进化方法 MicroGP. 该方法由进化核心、指令库和外部评估器组成, 其中进化核心适应并产生种群个体, 使用自适应机制、动态算子概率、动态算子强度和可变种群规模; 指令库用于将个体映射到有效的汇编语言程序; 外部评估器模拟汇编器, 并向进化核心提供反馈. Ioannides 等人[36]将 MicroGP 应用于复杂的工业级芯片设计程序中, 在成熟的工业芯片设计程序上评估了遗传算法对测试生成的促进作用. 实验结果表明, 将遗传算法和随机生成结合可以有效提高测试生成的质量和效率. 为了对复杂芯片设计程序进行充分测试, Wang 等人[37]将遗传算法和随机约束机制结合, 增强覆盖率引导的测试生成过程的灵活性和实用性. 该方法将测试输入进行编码, 用实数表示测试输入的参数; 将覆盖点在给定范围内被完全覆盖次数的倒数作为测试输入的适应度值, 该过程使用轮盘赌选择算法; 在交叉和变异操作后, 增加检查步骤验证新生成的测试输入是否合法. 该方法在真实的 PCIE 系统上的实验结果表明, 遗传算法与随机生成方法相比, 可以有效提升测试覆盖率. Imková等人[38]将遗传算法优化器引入处理器设计程序测试的 UVM 环境, 并讨论了通过微调参数对算法性能的影响. 在简单的 ALU 和复杂的 RISC 指令集处理器上的实验结果表明, 该方法与传统随机方法相比, 可以更快达到覆盖率收敛, 且需要的测试输入更少. Martinez-Cruz 等人[39]将具有重新初始化机制的二进制粒子群优化算法 (binary particle swarm optimization with reinitialization, BPSOr) 应用于芯片设计程序测试生成任务, 将基于仿真的测试和启发式算法进行结合, 生成测试向量序列来测试芯片设计程序行为. 该方法通过配置和初始化参数, 使用 BPSOr 算法生成测试序列, 输入到待测设计并根据覆盖率计算适应度函数, 作为反馈信息生成新的测试序列. 与遗传算法相比, 该方法需要更小的种群规模, 且在一些场景下, 该方法获得了比遗传算法和随机生成方法更好的效果.

此外, 部分研究者从约束优化的角度出发, 通过优化测试输入生成的约束来提高生成输入的质量. Roy 等人[40]将贝叶斯优化和梯度增强回归树方法应用于动态优化随机约束并自动执行约束随机测试, 通过离线或在线的方式间接影响测试平台的输入生成, 提高了芯片设计程序测试的覆盖率. Huang 等人[41]关注约束随机测试的参数优化问题, 提出了一种自动配置可调节测试参数的方法——智能回归规划器 (SRP), 以更好地探索输入空间, 并加速覆盖范围的收敛. 该方法使用贝叶斯优化算法进行参数更新, 使用夜间回归的覆盖范围作为反馈. 实验结果表明, 该方法可以提高回归测试的覆盖率和覆盖收敛的速度. Ismail 等人[42]对比研究了不同机器学习模型对芯片设计程序测试输入的预测精度, 包括人工神经网络、深度神经网络、支持向量回归和决策树. 研究表明, 机器学习模型的应用可以加快覆盖率收敛, 决策树具有更高的精度和更短的训练时间, 更适合作为用于功能测试的机器学习模型.

除单一的测试覆盖度量指标外, 部分方法重点关注测试输入生成方法揭示芯片设计程序缺陷的能力. Elver 等人[43]关注芯片设计程序的内存一致性问题, 将遗传编程应用于内存一致性验证过程. 该研究提出了新的测试输入生成框架, 将覆盖率作为适应度函数, 将非确定性的内存操作作为交叉函数. 实验结果表明, 该方法与随机生成测试输入相比, 可以发现更多设计错误. Pan 等人[44]关注芯片设计程序被恶意植入硬件木马的问题, 将强化学习技术应用于木马检测, 提出了一种木马检测方法. 该方法首先通过对设计的静态分析和动态仿真, 得到中间结果信息作为强化学习模型的输入, 随后对模型进行训练并生成测试输入, 不断改进以提高测试输入的覆盖率, 最终将训练好的强化学习模型用于测试生成. 实验结果表明, 该方法可以减少测试生成时间, 并提高对硬件木马的检测率. Bhargav

等人[45]将遗传算法应用于芯片设计程序的测试平台 (testbench) 生成任务, 利用覆盖率作为指标, 为测试平台的增强过程提供反馈信息, 最终通过增强的测试平台可以检测芯片设计程序中的错误. Chen 等人[46]关注芯片设计程序模糊测试的效率问题, 提出了一种使用粒子群优化来调度变异算子并动态生成测试输出程序的方法 PSOFuzz. 该方法使用改进的粒子群优化算法计算最优解, 选择探索芯片设计程序中新区域所需的变异算子, 采用基于粒子群优化的种子生成提高效率. 实验结果表明, 该方法与基于仿真的硬件模糊器相比, 提高了芯片设计程序漏洞检测速度和覆盖率.

3.1.3 深度学习方法

随着深度学习技术的发展, 许多研究人员使用深度神经网络来学习测试输入和覆盖结果的关系, 并将其用于指导测试输入生成. Fajcik 等人[47]将循环神经网络应用于芯片设计程序测试的处理器验证, 提出了一种通过循环神经网络动态改变随机测试生成约束的方法. 该方法基于芯片设计程序的覆盖率反馈, 和伪随机生成器进行结合, 通过改变生成器的约束来提高生成测试输入的质量. Wang 等人[48]提出了一种基于人工神经网络 (artificial neural network, ANN) 的覆盖率引导测试生成方法. 该方法利用 ANN 提取事务的特征, 通过标记测试输入和其触发的断言来对 ANN 进行训练, 通过学习历史数据和触发断言的关系来筛选更有可能触发目标断言的测试输入, 从而加速测试过程. Cristescu 等人[49,50]使用 ANN 对测试输入和覆盖结果进行建模, 将训练好的 ANN 模型与覆盖率引导的测试生成过程结合, 根据反馈的覆盖率结果使用 ANN 定向生成测试输入, 提高下一次迭代的覆盖率.

部分研究人员将神经网络和强化学习结合, 采用与环境的交互的方式来学习如何生成测试输入, 以最大化覆盖率. Pfeifer 等人[51]将定向测试生成任务建模为决策过程, 将强化学习方法作为决策策略来最大化覆盖范围. 该方法基于约束随机测试生成方法, 采用循环神经网络对设计环境和覆盖率结果进行学习, 生成测试输入以提高芯片设计程序测试的覆盖范围. Hughes 等人[52]将强化学习技术应用于约束随机测试过程. 该方法将输入数据和覆盖率结果用于模型训练, 使用训练后的强化学习模型对输入对应的覆盖点进行预测, 生成测试输入. Choi 等人[53]提出了一种基于深度神经网络和强化学习的测试生成方法, 使用 K 近邻搜索、迁移学习和回放缓存等方法, 探索更大的输入空间. 该方法将覆盖率作为目标函数, 使用策略梯度和贝叶斯优化方法生成和优化测试序列. Halim 等人[54]将强化学习技术应用于芯片设计程序测试生成任务, 结合了 Q 学习和蒙特卡罗方法, 指导芯片设计程序测试输入生成. Ohana[55]提出了一种基于深度强化学习的测试生成方法, 将深度 Q 网络 (deep Q-network, DQN) 与覆盖率引导的约束随机测试相结合, 通过使用 DQN 学习测试过程的状态和奖励值, 并生成测试序列实现最大的奖励值. 生成的测试序列加速了覆盖收敛, 提高了覆盖率, 并减少了人工成本. Tweehuysen 等人[56]提出了一种基于 Actor-Critic 算法的测试生成技术, 将芯片设计程序测试平台、约束和覆盖点分别视为强化学习方法中的环境、动作和状态, 直接生成测试输入的约束值.

3.1.4 大语言模型方法

近年来, 许多研究者利用大语言模型强大的推理能力和生成能力, 来学习芯片设计程序结构和覆盖率信息, 并通过提示词工程技术改善大语言模型的推理和生成表现. Zhang 等人[57]提出了一种评估大语言模型在芯片设计程序测试任务上的表现的框架 LLM4DV. 该框架由测试生成代理 (大语言模型)、DUT 和覆盖率监测器组成, 通过引入提示词模板, 利用覆盖率信息, 以交互方式得到大语言模型生成的测试输入, 并通过对提示词的改进来增强大语言模型的性能. 该方法考虑了大语言模型输入长度限制, 使用未覆盖点采样和最佳迭代消息采样方法, 在减小输入长度的同时提高输入信息的关键性. 为了防止大语言模型重复历史错误, 该方法使用对话重启和最佳迭代消息缓冲区重置方法, 在“有效忘记历史错误”和“重启对话后更快地了解任务”之间做出权衡. 改进提示词后的大语言模型在 3 个设计的 DUT 模块上分别取得了 98.94%、86.19% 和 5.61% 的覆盖率. 实验结果表明, 大语言模型在处理简单的 DUT 场景上的效率较高, 在复杂任务设置中效率较低, 但表现均优于约束随机测试生成方法.

Xiao 等人[58]探讨了使用大语言模型进行领域特定架构 (domain specific architecture, DSA) 测试时遇到的挑战和机遇, 提出了基于大语言模型的工作流程, 研究了大语言模型在传统处理器和新兴 DSA 的功能测试时的能力和局限性. 该方法通过收集覆盖率信息, 结合提示词生成器生成提示词, 输入到大语言模型并得到 C 语言或汇编语言形式的测试程序, 随后被编译生成二进制文件作为 DUT 的输入. 最后根据收集到的 DUT 覆盖信息进行下一次

迭代测试. 该工作流程在 DUT 上取得了 89% 和 91% 的覆盖率结果, 表明大语言模型驱动的芯片设计程序测试生成方法是比较有前景的研究方向.

此外, 部分研究人员将大语言模型与传统机器学习方法结合, 提高芯片设计程序漏洞检测的能力. Rostami 等人[59]研究了大语言模型对芯片设计程序模糊测试的促进能力. 该研究将大语言模型和模糊测试结合, 提出了一种基于机器学习和大语言模型的硬件模糊器 ChatFuzz, 利用大语言模型对处理器设计进行理解, 利用强化学习进行覆盖率引导的测试输入生成. 该方法使用基于 GPT2 模型的无监督训练来学习芯片设计程序的内部结构; 在基于近端策略优化 (proximal policy optimization, PPO) 的强化学习训练中使用反汇编程序作为评分代理, 对生成器模型进行优化; 在基于 PPO 的强化学习过程中使用基于仿真的覆盖率信息作为奖励函数, 提高测试覆盖率. 实验结果表明, 该方法在更短的时间内可以达到更高的条件覆盖率, 并能够识别处理器的错误行为, 表明了大语言模型在探索处理器漏洞方面的有效性, 为芯片设计程序安全和测试提供了更快速和全面的方法.

3.2 基于输入特征提取的测试生成

部分研究人员利用智能化方法提取测试输入特征来指导测试生成, 以解决在大规模的复杂芯片设计程序上进行覆盖率反馈过程困难且低效的问题. 这种方法的思路是提取测试输入的特征进行评估, 利用数据挖掘的方法, 如考虑测试输入的新颖性, 降低测试输入的相似程度, 通过利用提取的输入特征, 生成的测试输入往往会探索到更广泛的测试空间. 研究者通常使用支持向量机、决策树、规则学习、分类回归树、随机森林等监督学习方法对测试输入特征进行分类和选择, 使用子组发现等数据挖掘技术发现具有特殊特征的测试输入; 使用神经网络学习测试输入和复杂特征的关系, 使用注意力机制捕捉测试输入依赖关系, 使用生成对抗网络生成高质量的测试输入; 使用大语言模型学习历史输入特征, 并进行定向测试输入生成.

3.2.1 机器学习方法

在芯片设计程序测试中, 诸多研究利用决策树学习芯片设计程序测试输入的复杂特征, 并进一步生成高质量的测试输入. Katz 等人[60]提出了一种自动获取微架构信息并与测试生成器结合的方法, 该方法从仿真的历史信息中提取微架构数据, 包括生成输入的指令信息 (操作码、操作数、地址和值等) 和处理器状态, 并输入决策树进行学习. 通过学习过程得到测试生成规则, 最后将规则集成到测试生成器中, 产生更高质量的测试输入. 在大量测试输入用例中, 新颖测试用例是和大部分测试用例相似性较低的测试, 一般会覆盖不同的覆盖点. Chen 等人[61]通过将新颖测试输入和非新颖测试输入作为样本提供给学习引擎 (决策树或子组发现算法), 以提取描述新颖测试的结构特征的规则, 随后利用合理的规则作为约束对测试生成器进行增强, 生成更接近新颖测试的输入.

部分研究人员使用子组发现、规则学习算法来学习测试输入特征, 以发现具有特殊特征的输入, 并用于引导定向测试输入生成. Chen 等人[62]提出了一种从约束随机模拟数据中提取知识的方法, 该方法通过提取新颖测试输入的特殊属性规则作为特征, 并通过学习这些特征来指导约束随机测试生成, 以达到未覆盖的事件. 该方法采用了 CN2-SD 算法作为规则搜索引擎, 将分类规则学习算法和子组发现算法结合, 以同时实现预测性和描述性归纳. 实验结果表明, 通过利用从约束随机仿真过程提取的知识, 该方法与随机生成方法相比, 可以触发更多难以触发的断言. Hsieh 等人[63]提出了一种从设计文档和仿真数据中自动提取特征作为支持规则学习的可行方法. 该方法同时考虑了设计文档的信息和历史仿真的数据, 从文档中进行文本挖掘, 提取与设计信号相关的单词, 随后将单词映射到实际的设计信号; 同时对仿真数据进行收集和处理, 结合第一步得到的文本特征, 应用规则学习算法来归纳断言覆盖点的规则. 不同的规则学习方法均适用于此过程, 如子组发现、规则学习和分类回归树算法等. 在基于商业芯片设计程序的一组断言规则覆盖预测的实验结果表明, 70% 以上的断言可以获得 100% 准确的预测.

3.2.2 深度学习方法

部分研究工作使用深度神经网络来学习测试输入和复杂特征的关系. Miyamoto 等人[64]关注覆盖率引导的测试过程中存在难覆盖点的问题, 提出了一种通过深度学习技术学习仿真模式特征的方法, 并根据重构误差找到覆盖某个难以覆盖的点的仿真模式. 该方法首先采用多个随机模式进行仿真, 得到难覆盖点, 并对深度神经网络进行预训练来学习覆盖目标覆盖点的仿真模式的特征; 随后随机生成仿真模式, 并提供给多层神经网络, 计算它们的重

构误差并排序; 最后选择排名高于某个阈值的模式, 并将其用于仿真. 该方法的特点是不需要指定学习时要关注的特征, 而是直接对输入模式进行处理, 因此减少了人工成本. 实验结果表明, 该方法可以找到有效的仿真模式, 且与随机生成方法相比, 可以提高在难覆盖点上的覆盖次数和效率. Ambalakkat 等人[65]使用线性回归和人工神经网络等方法对输入和输出的复杂关系进行建模, 优化随机约束, 影响测试输入生成, 提高测试输入的质量.

Wang 等人[66]将注意力机制和强化学习算法应用于约束随机测试过程的边界案例覆盖问题, 通过将约束随机测试和机器学习算法结合, 提高测试的效率. 该方法将测试过程分为两阶段——约束选择阶段和输入生成阶段. 在约束选择阶段, 约束随机过程的约束范围集合重构为一系列子范围集, 利用 Transformer 的自注意力机制选择更好的子范围集; 在输入生成阶段, 通过强化学习算法用选定的子范围集生成有效测试输入. 实验结果表明, 该方法可以提高测试覆盖边界案例的命中率.

Yan 等人[67]关注芯片设计程序测试中难以覆盖的功能点, 基于随机森林和生成对抗网络, 提出了一种针对这些“最后一公里”功能点的覆盖方法. 该方法首先使用随机森林进行输入特征提取, 学习测试输入的相关变量和“最后一公里”功能覆盖点之间的关系, 减少搜索空间; 为了解决芯片设计程序中变量间的复杂约束问题, 采用生成对抗网络来生成满足变量间复杂约束的有效测试输入. 在真实工业芯片设计程序上的实验结果表明, 该方法与最先进的基于深度学习和遗传算法的芯片设计程序测试生成技术相比, 在相同数量的测试输入下, 达到了更高的“最后一公里”功能覆盖率, 并节省了更多时间.

3.2.3 大语言模型方法

研究者在使用大语言模型生成测试输入时, 通常在提示词中包含历史输入信息, 让大语言模型学习历史输入特征, 并结合当前生成任务进行推理生成. Zhang 等人[57]在 LLM4DV 测试输入生成框架中考虑历史对话中的关键信息, 并按策略对历史对话进行采样, 让大语言模型在输入长度范围内尽可能多地理解输入特征和当前任务. 同时考虑大语言模型生成错误测试输入的特征, 按策略对历史信息进行清除, 提高大语言模型对测试输入的理解效果. 通过大语言模型对特征的学习和改进, 有效提高了其生成测试输入的质量.

3.3 基于语义结构分析的测试生成

考虑到提取输入特征需要首先获得大量测试输入作为样本, 一些新颖性检测的方法还需要具有不同类别标签的样本用于模型训练, 另一种生成高质量测试输入的方法是对芯片设计程序代码或规约的语义结构进行分析. 传统的语义结构分析方法依赖于大量专家知识和人工经验对芯片设计程序或文档进行分析, 因此许多研究也致力于将智能化方法应用于对芯片设计程序的语义结构分析过程, 如将芯片设计程序的功能测试问题抽象为状态转换图的可达性问题等, 并结合测试生成方法生成有效的测试输入. 研究者通常使用聚类和决策树对芯片设计程序结构信息进行分类; 使用自然语言处理方法对芯片设计程序代码或规约进行理解, 使用图神经网络对芯片设计程序结构信息进行表征; 使用大语言模型对芯片设计程序结构进行理解.

3.3.1 机器学习方法

部分研究者使用聚类方法对芯片设计程序结构中的相似属性进行分类, 实现对芯片设计程序结构的表征. Shyam 等人[68]提出了一种用芯片设计程序结构导出的距离函数来指导仿真测试过程的方法 Guido, 该方法通过对芯片设计程序的精确可达性分析计算得到距离函数, 以此控制随机输入生成器生成测试输入, 引导随机测试过程向目标发展. 实验结果表明, 该方法的运行时间相比随机仿真测试更短, 且质量更高. Chen 等人[69]提出了一种利用芯片设计程序结构中不同属性之间的相似性进行测试生成的方法, 该方法对相似属性进行聚类, 并通过高效的学习技术, 在类似的测试输入之间共享知识来减少特定属性的总体测试生成时间. 实验结果表明, 该方法可以减少总体测试生成时间.

此外, 还有一些研究者使用决策树对芯片设计程序结构进行学习, 生成断言来检测芯片设计程序存在的缺陷. Vasudevan 等人[70]提出了一种使用数据挖掘和芯片设计程序静态分析的自动生成断言方法 GoldMine, 该方法通过对芯片设计程序进行静态分析得到设计约束, 连同仿真数据一起输入至基于决策树的监督学习算法, 最终产生一组候选断言. 实验结果表明, 该方法可以在芯片设计程序中生成复杂的高覆盖率的断言, 减少人工成本. Liu 等

人[71]提出了一种使用芯片设计程序源代码的静态和动态分析来发现词级特征的方法, 该方法使用底层学习算法发现的词级特征来生成词级断言, 并非类似 GoldMine 方法生成的位级断言, 提高了生成断言的表示性和可读性. 该方法首先识别词级目标, 然后识别词级特征, 随后使用决策树进行断言生成. 实验结果表明, 词级断言可以检测更多设计错误. Sheridan 等人[72]在 GoldMine 方法的基础上做了改进, 将决策树算法改为基于覆盖引导的挖掘算法, 提高了输入空间的覆盖范围. Hanafy 等人[73]提出了一种新的挖掘技术, 将广度优先决策树应用于从仿真轨迹数据中提取设计属性, 包括芯片设计程序断言等信息.

3.3.2 深度学习方法

除了从芯片设计程序代码中挖掘芯片设计程序的语义结构, 还有研究从设计规约出发, 运用自然语言处理的方法进行分析, 并生成符合设计规约的断言, 检测芯片设计程序中的错误. Soeken 等人[74]提出了一种将自然语言规约转换为形式化断言的方法, 使用自然语言处理技术, 首先根据抽象级别分割自然语言, 然后根据语句相似性划分为聚类, 使用同一个模板来翻译相同聚类中的断言, 减少了人工成本. Harris 等人[75]提出了一种新的学习算法 GLAsT, 创建了一个自定义的形式语法, 可以捕捉规范的写作风格和句子结构, 将规约语句转换为 SystemVerilog 断言 (SVA). Gulliya 等人[76]使用机器学习算法识别自然语言描述中的模式, 提出了一种自动捕获相对可寻址寄存器和序列的设计意图的方法, 实现寄存器自动化和生成 SystemVerilog 断言. Parthasarathy 等人[77]提出了一种将自然语言转换为 SystemVerilog 断言的方法 SpecToSVA, 包括目标语句检测、语句分析、深度神经网络和模型训练过程, 具有合理的精度. Aditi 等人[78]提出了一种结合机器学习和形式分析的混合方法, 从自然语言规约中自动生成 SystemVerilog 断言, 其中形式分析用于解析输入文本, 机器学习用于将输入文本转换为形式逻辑和断言, 通过混合方式减少了机器学习算法的负担, 并提高了生成断言的准确性.

Vasudevan 等人[79]将图神经网络应用于芯片设计程序的测试流程, 提出了一种学习芯片设计程序语义抽象的方法 Design2Vec. 该方法用于学习捕获芯片设计程序语义的连续表示, 将芯片设计代码表示为控制数据流图. 基于这种表示方法, 使用图神经网络进行表示学习, 预测芯片设计程序在测试输入上进行仿真的状态. 最后, 将这种表示方法应用于预测芯片设计程序的覆盖点, 以及测试输入的生成. 实验结果表明, 该方法具有更好的性能, 并且可以覆盖通过人工方法很难覆盖的覆盖点.

3.3.3 大语言模型方法

大语言模型在文本和代码理解、高级推理和程序综合等任务上表现出极大的优势, 因此众多研究者致力于利用大语言模型理解芯片设计程序规约或代码, 以此来增强测试输入生成过程. 现阶段研究人员主要将大语言模型应用于理解芯片设计程序并生成断言、调试并检测芯片设计程序缺陷, 或构建领域大语言模型并用于通用芯片设计程序测试任务等.

Liu 等人[80]探索大语言模型在工业芯片设计程序中的应用, 构建了芯片设计领域大语言模型 ChipNeMo, 用于聊天机器人助手、EDA 脚本生成和错误总结分析. 通过和基础 LLaMA 2 的对比实验表明, 语言模型的领域自适应预训练可以在领域相关的下游任务中带来卓越的性能, 而不会降低通用能力. ChipNeMo-70B 在工程助理聊天机器人和 EDA 脚本生成两个任务上优于功能强大的 GPT-4, 并在错误总结和分析方面也表现出有竞争力的性能. Fu 等人[81]开发了一个专为半导体行业硬件领域定制的大语言模型 Hardware Phi-1.5B, 适用于芯片设计程序测试过程. 该工作开发了一个专门的分层数据集, 并在中型数据集上进行预训练, 提高了芯片设计和验证任务的性能.

在生成芯片设计程序断言并检测设计缺陷方面, Orenes-Vera 等人[82]探讨了大语言模型用于生成断言的能力, 设计了一个评估断言的正确性和完整性的框架, 并评估了大语言模型生成断言的能力. 研究表明, GPT-4 可以将 Verilog 翻译为 SVA, 而且在一定程度上可以捕捉到一些设计意图. Sun 等人[83]提出了一种利用大语言模型来自动化生成断言的框架 nl2sva, 从自然语言的通用设计属性中生成特定的 SVA, 以及实现模型检查器和人机循环, 以交互的方式向测试者和底层大语言模型提供反馈, 方便调试设计. Kande 等人[84]关注基于断言的芯片设计程序测试, 利用芯片设计程序文件的代码注释等信息, 将大语言模型应用于断言生成任务, 研究了使用大语言模型生成断言的能力, 设计了一个生成不同提示的评估框架, 并创建了一个基准套件. Fang 等人[85]提出了一个用于完整规范文

件的自动断言生成框架 AssertLLM, 该框架将复杂任务分为 3 个阶段, 包含 3 个定制的大语言模型, 用于提取结构规范、映射信号定义和生成断言, 并提供了一个评估断言生成能力的开源基准. 实验结果表明, AssertLLM 生成的断言中有 89% 在语法和功能上都是准确的. Paria 等人[86]提出了一种利用大语言模型的知识库来识别芯片设计程序中安全漏洞的框架 DIVAS, 首先根据芯片设计规范通过大语言模型生成查询语句, 随后利用大语言模型的上下文理解能力来识别与芯片设计程序相关的 CWE, 利用大语言模型根据相关 CWE 列表创建安全断言, 以确定现有芯片设计程序是否容易受到这些特定漏洞的影响, 并将断言转换为安全策略, 表示芯片设计程序的安全需求.

此外, 部分研究人员使用大语言模型对芯片设计程序代码进行调试, 以检测芯片设计程序代码的错误行为. Tsai 等人[87]提出了一种使用大语言模型交互式调试代码的框架 RTLFixer, 通过采用检索增强生成 (RAG) 和 ReAct 提示, 使大语言模型能够充当自主代理, 通过反馈交互式调试代码, 降低了大语言模型直接生成代码的错误率. 在解决语法错误方面, 该框架可以成功纠正调试数据集中约 98.5% 的编译错误. 在 VerilogEval-Machine 和 VerilogEval-Human 的基准测试中的 pass@1 成功率分别提高了 32.3% 和 10.1%. Xu 等人[88]关注大语言模型对芯片设计程序代码的调试表现, 提出了一种新的框架 MEIC, 利用大语言模型来克服其在芯片设计程序代码调试中的局限性, 识别和纠正语法和函数错误, 同时构建了一个包括常见芯片设计程序编程错误的开源数据集. Ma 等人[89]将大语言模型作为芯片设计程序代码阅读器, 集成到覆盖率引导的测试生成过程. 大语言模型可以较为准确地掌握芯片设计程序代码的逻辑, 从而产生能够到达未探索分支的测试输入. 为了充分利用大语言模型对代码的理解能力, 该方法引入了两个解释器模块, 用于将中间数据进行格式化. 其中, 覆盖率解释器模块将初始仿真器覆盖率报告格式化为大语言模型容易理解的格式, DUT 解释器模块 DUT 代码使用自然语言描述并丰富信息. 解释器增强了大语言模型对测试意图和 DUT 功能的理解. 实验结果表明, 虽然大语言模型在简单和中等规模的 DUT 上的性能优于随机方法, 但其性能会随着 DUT 复杂度的上升而下降.

在芯片设计程序缺陷检测方面, Saha 等人[90]研究了大语言模型用于芯片安全验证的可行性, 展示了大语言模型在漏洞插入、安全评估、安全验证、对策制定等场景中的不同能力, 总结了将大语言模型集成到芯片安全框架中的潜力和挑战. Meng 等人[91]提出了一种基于自然语言处理的芯片安全属性生成器 NSPG, 利用芯片设计文档来提取安全属性, 包括一种硬件安全特定语言模型 HS-BERT 和一种数据修改技术, 以改进自动安全属性生成. 实验结果表明, 该方法可以有效提取安全属性, 并比当时流行的 ChatGPT 提供了更好的性能. Fu 等人[92]利用版本控制数据来收集开源芯片设计程序缺陷及其修复步骤的数据集, 并将大语言模型在数据集上微调, 使得大语言模型能够识别和修复芯片设计程序中的错误. Ahmad 等人[93]关注大语言模型在硬件设计程序安全缺陷的自动检测和修复任务的应用, 构建了一个硬件安全漏洞领域语料库, 设计了一个评估大语言模型修复特定硬件设计程序错误的框架, 支持对不同提示工程技术的探索. 该框架由包含错误设计的源代码、定位错误的检测器、大语言模型驱动的修复错误的生成器和检测修复是否成功的评估器组成. 其中, 提示词生成器接受修复错误的命令和错误定位的信息, 生成提示词并输入到大语言模型, 得到输出的修复结果. 研究表明, 当选择合适的参数和提示词时, 大语言模型可以成功修复语料库中的芯片设计程序错误, 且表现优于 CirFix[94]. Tarek 等人[95]提出了一个基于大语言模型检测芯片设计程序中的安全漏洞的框架 SoCureLLM, 增强了大语言模型的适用性和在大规模设计上的能力. 在该框架中, 总结大语言模型用于理解芯片设计程序代码信息、检测大语言模型用于检查芯片设计程序代码是否存在安全策略违规、查找大语言模型用于定位芯片设计程序代码的潜在攻击点、生成大语言模型用于制定安全策略. 实验结果表明, 该框架性能优于最先进的安全验证方法. Saha 等人[96]构建了一个包含 10 000 个易受攻击的有限状态机设计的数据集, 包含了 SecRT-LLM 框架生成的 16 个不同的安全弱点和漏洞. 分析了大语言模型在漏洞插入、检测和缓解方面的能力.

4 测试预言

与传统软件测试类似, 测试预言常被用于判断给定的输入下芯片设计程序行为是否符合预期 (芯片规格设计)[6]. 本节详细介绍了对芯片设计程序中常用的两种测试预言概念及相关工作, 即参考模型 (差分测试) 和蜕变测试.

4.1 参考模型

参考模型 (即传统软件测试中的差分测试) 通常被用于判断待测芯片设计程序的正确性. 具体来说, 开发人员在测试 DUT 的正确性之前, 需要依据 DUT 的规格说明, 实现与 DUT 功能完全等价的模型作为参考模型. 而后, 开发人员生成测试输入并将测试输入同步传输至 DUT 与参考模型中进行协同仿真. 通过比较二者之间的状态和行为来判断 DUT 是否符合规约[6], 实现对 DUT 功能正确性的检测.

在处理器测试领域, 通常使用指令集仿真器 (instruction set simulator, ISS) 作为参考模型. ISS 是一种用于模拟处理器指令集架构 (instruction set architecture, ISA) 的软件, 执行与处理器相同的指令集, 运行于软件环境 (而非物理硬件). ISS 可以模拟处理器的行为, 包括指令执行、寄存器操作、内存访问等. 在处理器设计程序测试过程中, ISS 常用于检测处理器设计是否符合预期. 图 4 介绍了基于参考模型的处理器设计程序测试的通用流程. 以开放指令集架构 RISC-V 为例, 在测试基于 RISC-V 架构的处理器设计正确性时, 研究人员通常使用 Spike[97]等 RISC-V ISS 与 DUT 协同仿真, 记录相同输入下的输出, 比较最终结果并进行差异调试.

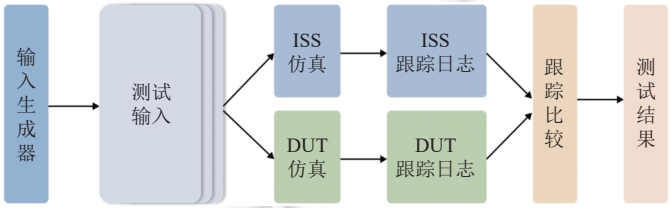


图 4 基于参考模型的处理器设计程序测试一般流程

参考模型在芯片设计程序的功能测试和回归测试中得到了广泛应用, 通过对跨平台测试、系统级测试和实时结果反馈的支持, 其在复杂芯片设计程序的测试过程中起到了重要作用, 有助于保障芯片设计程序的正确性和鲁棒性. Herdt 等人[98]设计了一种高效的协同仿真测试平台, 使待测设计和指令集仿真器执行相同的指令, 通过检测二者的状态是否匹配来判断设计是否存在错误. Bruns 等人[99]提出了一种基于无限随机指令流的处理器设计程序测试框架, 将生成器生成的指令由待测处理器设计和指令集仿真器执行, 使用比较器来比较二者的寄存器值, 以发现二者的差异. 此外, Bruns 等人[14]将符号执行技术应用于芯片设计程序测试的处理器验证, 利用指令集仿真器作为参考模型, 向指令集仿真器和待测处理器设计提供相同指令, 并在执行后比较寄存器值, 能够检测处理器设计的错误. Hur 等人[23]提出了芯片设计程序模糊测试框架 DifuzzRTL, 其中使用相同测试输入对指令集仿真器和待测芯片设计程序共同仿真, 并在每次运行后交叉检测二者的执行结果, 以此判断待测设计是否正确. Xu 等人[24]提出了另一种芯片设计程序模糊测试框架 MorFuzz, 使用相同的运行时动态变异的输入指令流对指令集仿真器和待测处理器设计进行仿真, 比较二者状态并进行错误检测和同步. Canakci 等人[25]在模糊测试框架 ProcessorFuzz 中同样使用相同测试输入对待测设计和指令集仿真器进行仿真, 最终通过比较二者状态来检测潜在的错误.

4.2 蜕变测试

在芯片设计程序测试中, 蜕变测试通常被用于参考模型缺失的场景中, 为芯片设计程序测试方法提供测试预言. 蜕变测试通过构造等价的蜕变关系对待测程序的输入进行转换, 程序应该对转换前后的输入产生相同或者满足指定变化关系的输出, 通过检测软件的输出变化是否符合预期检测芯片设计程序的正确性. 图 5 展示了芯片设计程序蜕变测试的通用流程: 首先根据源 (source) 测试输入和蜕变关系来生成后续 (follow-up) 测试输入, 随后对待测设计程序执行源测试输入和后续测试输入, 最后通过检查蜕变关系得到测试结果.

Liu[100]探讨了将蜕变测试应用于硬件容错的可行性, 提出了两种蜕变测试方法: 时间冗余和硬件冗余, 其中时间冗余通过对操作数进行移位并重新计算, 比较两次执行结果是否符合移位关系; 硬件冗余通过对测试输入进行修改, 使用相同测试单元的副本同时仿真, 比较二者输出是否符合蜕变关系. Hassan 等人[101]提出了一种基于蜕变测试的芯片设计程序测试方法, 构造了一组蜕变关系, 描述了连续执行的输入和输出的关系. 该方法在工业级芯片设计程序上验证了有效性, 展示了蜕变测试无需参考模型即可发现重要设计错误的能力. Riese 等人[102]将蜕变测

试应用于芯片设计程序的处理器测试领域, 考虑了 RISC-V 指令集架构的可用性, 并提供了为 RISC-V 指令集架构定制的蜕变规则. 该工作提出了一种高效的动态蜕变测试框架, 将蜕变规则与 ISS 结合, 通过变异分析来衡量蜕变规则的质量. 实验结果证明了所提出的蜕变规则的有效性. Hazott 等人[103]将蜕变规则应用于嵌入式系统的虚拟原型 (virtual prototype, VP) 测试过程, 提出了一个自动化蜕变测试框架, 并成功检测到设计程序错误.

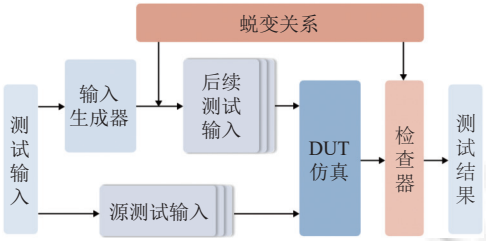


图 5 基于蜕变测试的芯片设计程序测试一般流程

5 测试优化

传统软件中, 测试优化是提升测试效率、缩短测试开发周期、确保高质量软件快速交付的重要环节. 在芯片设计程序测试中, 已有部分研究工作提出针对芯片设计程序的测试优化工作, 对测试输入的执行优先级进行排序以及测试集合进行约减, 以此提升芯片设计程序测试效率, 同时保证测试有效性和充分性. 本节梳理归纳了现阶段智能化芯片设计程序测试优化方法.

现阶段部分研究人员通过采用无监督和监督学习的方法对新颖测试进行分类和检测, 并进行测试选择和优先级排序, 优化测试过程. 新颖性检测是识别大部分测试中具有不常见特征的少量测试, 通过选择这些新颖的测试, 可以比相似的测试贡献更大的覆盖范围.

5.1 基于无监督学习的方法

一些研究者使用支持向量机从大量随机测试中对测试输入进行分类和选择, 优化测试过程. Guzey 等人[104]将支持向量机应用于测试集合约减、选择和优先级排序任务, 提出了一种基于无监督支持向量分析的测试模型生成方法, 并用此模型来选择可能执行未测试功能的测试输入. 在测试集合约减上, 使用支持向量模型过滤掉冗余测试, 减少了仿真需要的时间; 在测试优先级排序上, 首先通过约束随机生成器产生若干测试用例, 选择部分测试用例训练支持向量模型, 随后通过首先应用异常值的方法对其余测试用例进行排序, 在更短的时间内提高了测试覆盖率; 在测试集合的选择上, 使用支持向量模型识别集合中异常值的数量, 选择数量高的测试集合进行仿真, 实验结果表明, 异常值更高的测试集合会带来更高的覆盖率结果. Chang 等人[105]提出了一种基于新颖性检测的在线功能测试选择方法. 该方法首先通过随机或约束的方式生成测试程序集合, 随后使用基于图的编码衡量测试程序之间的相似性, 其中测试输入的新颖程度用图编辑距离来衡量, 随后使用支持向量机从测试程序中选择新颖的测试输入, 最终提高覆盖率.

部分研究者使用聚类方法对测试输入进行优先级排序. Wang 等人[106]关注芯片设计程序的功能覆盖率提升问题, 提出了一种基于聚类算法的测试优先级排序方法, 该方法的核心是增强 K-means 算法, 首先对随机生成的大量测试用例进行仿真, 得到覆盖率数据库; 随后再生成 n 个测试用例, 转换为特征向量, 用上一步的覆盖率数据库对 n 个特征向量进行覆盖率预测, 根据预测结果, 使用增强 K-means 算法将 n 个测试用例划分为 k 个簇; 最后根据预测的覆盖率结果对测试集进行优先级排序, 因此在仿真过程中可以更早执行可以明显提高覆盖率指标的测试, 提高测试效率. Choi 等人[107]提出了一种基于聚类的测试用例优先级排序的方法, 核心思想是将仿真日志提取为测试用例的特征, 包括任务、功能和错误信息, 并根据自定义权重或优先级算法对测试用例进行优先级排序, 尽早发现不同或重要的错误, 提高错误检测的速度.

此外, 还有部分研究者使用其他机器学习方法进行测试用例约减、选择和排序. Liang 等人[108]将基于机器学

习的异常检测方法应用于测试选择, 首先在测试池中进行快速仿真得到覆盖率结果并提取测试用例表征, 随后在表征空间中应用基于孤立森林的异常检测方法, 选择可能执行不常见覆盖事件的新颖测试, 最终将选择的测试进行仿真, 减少了覆盖率收敛所需时间.

在使用深度学习方法对测试用例进行选择方面, Blackmore 等人[109]关注芯片设计程序测试输入冗余的问题, 提出了一种基于无监督学习的新颖测试输入选择方法, 使用神经网络构建自动编码器, 并在大量测试用例上进行训练, 随后对测试输入进行预测并选择若干异常值用例, 选择这些用例重训练, 重复此过程至达到一定水平. 通过这种方式选择的新颖测试用例更有可能增加覆盖范围.

5.2 基于监督学习的方法

一些研究者采用监督学习方法对芯片设计程序测试过程进行优化. Masamba 等人[110]将监督学习方法应用于覆盖率引导的测试选择任务, 将覆盖率引导的测试选择和新颖性驱动的测试过程结合, 提出了一种混合智能测试方法. 在新颖性驱动的测试过程中, 使用新颖性检测来识别潜在有用的测试用例并确定其优先级, 首先创建大量测试输入作为候选, 随后学习这些测试的相似性模型, 并根据新生成的测试和先前仿真的测试的差异进行排序, 优先选择最不同的测试进行仿真, 加快覆盖收敛. Parthasarathy 等人[111]提出了一种利用机器学习方法从芯片设计程序回归测试集合中选择测试子集的方法, 通过学习芯片设计程序代码和测试用例的特征, 以估计测试用例揭露芯片设计程序更改引入错误的可能性, 选择揭错能力更强的测试用例执行, 提高测试效率.

此外, 部分研究者采用基于神经网络的监督学习方法对测试用例进行选择. Zheng 等人[112]提出了一种基于神经网络的新颖测试选择方法, 用于在基于仿真的芯片设计程序测试过程中选择新颖测试, 加速覆盖率收敛. 该方法建立了一个测试选择循环, 首先使用约束随机生成器生成随机测试向量并进行编码, 随后输入到测试选择器选择新颖测试, 将选择的测试输入执行仿真过程, 结合仿真信息和覆盖率信息得到训练数据, 通过动态新颖度学习对测试选择器进行重训练. 其中, 测试选择器是深度神经网络结构, 用于衡量测试向量的新颖度分数, 在测试选择循环中使用仿真信息进行训练. 实验结果表明, 该方法与随机测试选择方法相比, 可以减少仿真时间和提高覆盖率, 更快实现功能覆盖率收敛. Singh 等人[113]使用深度神经网络加速复杂数据路径的测试过程, 将测试输入和断言结果进行训练, 对于每个测试用例, 利用深度神经网络预测是否命中某断言, 并根据该断言所在断言组是否已覆盖, 将测试用例分类有效和冗余测试. 通过丢弃冗余测试输入并收集有效测试输入, 提高了处理器设计程序测试的效率.

6 芯片设计程序测试覆盖度量指标、测试数据集和开源工具

为了对芯片设计程序测试领域研究人员提供更加细粒度的参考, 本节对目前芯片设计程序测试领域覆盖度量指标的使用、测试数据集和开源工具进行归纳梳理. 具体来说, 本节在表 1 中罗列了第 3.1.1 节所提 3 种测试覆盖指标在论文中的使用情况, 以此来凸显当前芯片设计程序测试研究领域在测试覆盖方面的主要关注点; 此外, 本文从 88 篇调研的论文和 GitHub 开源项目中整理了 40 个芯片设计程序测试中较为常用的数据集和 18 个工具, 列举了数据集和工具的名称和类别等信息, 并提供了仓库地址, 希望为未来研究人员开展相应研究提供良好的资料支持.

表 1 芯片设计程序测试覆盖度量指标的使用

覆盖度量指标	文献
代码覆盖	[22,35,36,38,41,43,45,46,51,53,58,59,61,64,70,79,89,102,104−106,111]
功能覆盖	[32−34,37−39,41,42,47,49,50,52,54−57,65–67,69,99,106,108−110,112]
断言覆盖	[31,48,62,63,71−73,82,113]
其他覆盖	[20,21,23−25,32,33,38,40,44,60]

6.1 芯片设计程序测试数据集

在芯片设计程序测试中, 数据集通常是以 Verilog、SystemVerilog (SV)、VHDL 或 Chisel 等硬件描述语言编

写的程序, 根据其规模大小和实现功能的不同, 可以分为设计程序库和处理器设计程序. 表 2 整理了芯片设计程序测试开源数据集.

表 2 芯片设计程序测试开源数据集

名称	类别	语言	地址
OpenCores	设计程序库	Verilog/VHDL	https://opencores.org
OpenTitan IP	设计程序库	SV	https://github.com/lowRISC/opentitan/tree/master/hw/ip
FPGA Peripherals	设计程序库	Verilog/SV	https://github.com/FPGAwars/FPGA-peripherals
LLM-for-Vulnerability-Insertion-and-Detection	设计程序库	Verilog	https://github.com/HWSecurityTeam/LLM-for-Vulnerability-Insertion-and-Detection
LLM4DV	设计程序库	SV	https://github.com/ZixiBenZhang/ml4dv
MEIC	设计程序库	Verilog	https://anonymous.4open.science/r/Verilog-Auto-Debug-6E7F
CirFix	设计程序库	Verilog	https://github.com/hammad-a/verilog_repair
Open Source RTL Design	设计程序库	Verilog	https://github.com/embedded-explorer/Open-Source-RTL-Design
SymbiYosys	设计程序库	Verilog	https://github.com/YosysHQ/sby/tree/main/docs/examples
WB2AXIP	设计程序库	Verilog	https://github.com/ZipCPU/wb2axip
Hackaday Supercon 2019 Badge	设计程序库	Verilog	https://github.com/Spritetm/hadbadge2019_fpgasoc
hardware-bugbase	设计程序库	Verilog/SV	https://github.com/efeslab/hardware-bugbase
UberDDR3	设计程序库	Verilog/SV	https://github.com/AngeloJacobo/UberDDR3
formal_hw_verification	设计程序库	VHDL	https://github.com/tmeissner/formal_hw_verification
hdl-benchmarks	设计程序库	Verilog/VHDL	https://github.com/ispras/hdl-benchmarks
BaseJump STL	设计程序库	SV	https://github.com/bespoke-silicon-group/basejump_stl
yosys tests	设计程序库	Verilog	https://github.com/YosysHQ/yosys/tree/main/tests/hana
uts-sv	设计程序库	Verilog	https://github.com/SymbiFlow/utd-sv
ivtest	设计程序库	Verilog	https://github.com/steveicarus/ivtest
hdlConvertor tests	设计程序库	Verilog/SV	https://github.com/Nic30/hdlConvertor/tree/master/tests
Rocket Chip	处理器设计程序	Scala	https://github.com/chipsalliance/rocket-chip
BOOM	处理器设计程序	Scala	https://github.com/riscv-boom/riscv-boom
Ibex	处理器设计程序	SV	https://github.com/lowRISC/ibex
CVA6	处理器设计程序	SV	https://github.com/openhwgroup/cva6
CV32E40P	处理器设计程序	SV	https://github.com/openhwgroup/cv32e40p
CV32E40X	处理器设计程序	SV	https://github.com/openhwgroup/cv32e40x
CV32E40S	处理器设计程序	SV	https://github.com/openhwgroup/cv32e40s
PicoRV32	处理器设计程序	Verilog	https://github.com/YosysHQ/picorv32
VeeR EH1	处理器设计程序	SV	https://github.com/chipsalliance/Cores-VeeR-EH1
VeeR EH2	处理器设计程序	SV	https://github.com/chipsalliance/Cores-VeeR-EH2
VeeR EL2	处理器设计程序	SV	https://github.com/chipsalliance/Cores-VeeR-EL2
Caliptra	处理器设计程序	SV	https://github.com/chipsalliance/caliptra-rtl
Zip CPU	处理器设计程序	Verilog	https://github.com/ZipCPU/zipcpu
mor1kx	处理器设计程序	Verilog	https://github.com/openrisc/mor1kx
Human Resource Machine CPU	处理器设计程序	Verilog	https://github.com/adumont/hrm-cpu
BlackParrot	处理器设计程序	SV	https://github.com/black-parrot/black-parrot
FX68K	处理器设计程序	SV	https://github.com/ijor/fx68k
RSD	处理器设计程序	SV	https://github.com/rsd-devel/rsd
SCR1	处理器设计程序	SV	https://github.com/syntacore/scr1
tnoc	处理器设计程序	SV	https://github.com/taichi-ishitani/tnoc

6.2 芯片设计程序测试工具

表 3 整理了当前芯片设计程序测试领域常用的开源工具, 包括芯片设计程序仿真器、测试输入生成器和模糊

测试工具等.

表 3 芯片设计程序测试开源工具

名称	类别	地址
Verilator	仿真器	https://www.veripool.org/verilator
GHDL	仿真器	https://github.com/ghdl/ghdl
cocotb	协同仿真库	https://github.com/cocotb/cocotb
VUnit	单元测试框架	https://github.com/VUnit/vunit
iverilog	HDL编译工具	https://github.com/steveicarus/iverilog
Yosys	RTL综合工具	https://github.com/YosysHQ/yosys
apio	RTL验证工具	https://github.com/FPGAwars/apio
EDAUtils	RTL测试工具	https://www.edautils.com
RISCV-DV	随机指令生成器	https://github.com/chipsalliance/riscv-dv
DatagenDV	测试程序生成器	https://github.com/microsoft/datagenDV
LLM4DV	LLM基准框架	https://github.com/ZixiBenZhang/ml4dv
MEIC	LLM基准框架	https://anonymous.4open.science/r/Verilog-Auto-Debug-6E7F
CirFix	缺陷修复框架	https://github.com/hammad-a/verilog_repair
Mantra	变异测试工具	https://github.com/wndif/Mantra
riscv-formal	形式验证框架	https://github.com/YosysHQ/riscv-formal
Icicle	模糊测试工具	https://github.com/icicle-emu/icicle
DifuzzRTL	模糊测试工具	https://github.com/compsec-snu/difuzz-rtl
MorFuzz	模糊测试工具	https://github.com/sycuricon/MorFuzz
PreSiFuzz	模糊测试工具	https://github.com/IntelLabs/PreSiFuzz

7 未来研究展望

现阶段, 已有智能化测试方法在提高芯片设计程序测试覆盖率和效率、优化测试过程等方面取得了一定研究进展, 但该方向仍存在一些挑战. 本文对该领域存在的主要挑战进行概括, 希望未来研究对这些挑战提出更好的解决方案, 不断完善芯片设计程序的智能化测试方法和流程, 提高芯片设计程序的质量保证.

第一, 尽管本文在第 5 节列出了一些可用的开源数据集, 但由于工业芯片设计程序的复杂性, 这些开源数据集并不能涵盖各种复杂度的芯片设计和应用场景, 比如专用集成电路 (ASIC) 设计、高性能计算 (HPC) 芯片设计和复杂的多核处理器设计等. 因此, 在未来研究中, 可以开发更多样化、复杂度高的芯片设计程序, 有助于未来研究在不同应用场景下的性能评估.

第二, 对于复杂芯片设计程序测试的研究, 将结构分析和仿真测试结合的方法展现出较好的效果. 大语言模型因其较高的推理和生成能力, 多被用于理解芯片设计程序代码任务. 由于大语言模型上下文理解能力和输入长度的限制, 无法直接用于理解完整的复杂芯片设计程序, 因此如何高效地利用大语言模型理解复杂芯片设计程序是一项有待进一步研究的问题. 一些研究[95]将芯片设计程序进行分割并使用大语言模型对代码进行总结, 来提高大语言模型对芯片整体设计的理解, 但在此基础上保证整体测试效率和大语言模型理解结果的准确性仍值得进一步研究.

第三, 目前已有许多研究使用大语言模型根据对芯片设计程序代码的理解生成断言, 进而检测芯片设计程序可能存在的缺陷. 但目前关注大语言模型和芯片设计程序功能覆盖问题的研究相对较少, 相比代码覆盖和断言覆盖, 较少有研究使用大语言模型生成芯片设计程序功能覆盖组和覆盖点. 如何有效地借助大语言模型提高芯片设计程序功能覆盖, 是一个值得研究的方向.

第四, 不同芯片设计程序的输入形式往往不同, 如简单设计模块的输入激励、处理器设计的指令序列、复杂工业芯片设计程序的自定义配置参数等, 因此不同芯片设计程序测试方法一般无法通用地适用于所有类型的芯片

设计程序, 导致了测试工具和方法的多样性和专用性. 未来研究可以致力于开发通用性更强的测试框架, 探索标准化测试流程和统一的测试接口, 减少测试方法实现之间的差异, 提高测试效率.

8 总 结

芯片设计程序作为芯片设计和制造流程中最关键的部分, 以软件的形式表征了芯片设计的内部结构, 对于芯片设计程序的质量保证将影响芯片最终生产的质量. 本文对芯片设计程序的智能化测试研究进行了系统的归纳总结, 主要关注机器学习、深度学习和大语言模型等智能化方法在芯片设计程序测试领域的提出、发展、完善和广泛应用的演化过程, 并从测试输入生成、测试预言构造和测试执行优化这 3 个方面对该领域进行了详细的综述. 表 4 对本文综述的智能化芯片设计程序测试方法进行了归纳梳理, 详细列举了不同智能化测试方法涉及的技术和相关文献, 以供研究者参考.

表 4 智能化芯片设计程序测试方法

方法	技术	文献
机器学习方法	聚类	[68,69]
	贝叶斯网络	[32−34]
	遗传算法	[35−38,43,45]
	粒子群优化算法	[39,46]
	贝叶斯优化	[40,41]
	决策树	[42,60,61,70−73]
	规则学习	[62,63]
	强化学习	[44]
深度学习方法	人工神经网络	[42,48−50,64,65]
	循环神经网络	[47,51]
	图神经网络	[79]
	深度强化学习	[52−56]
	Transformer	[66]
	生成对抗网络	[67]
大语言模型方法	模型预训练	[80,81,90]
	模型微调	[59,88,90,92]
	提示词工程	[57,58,82−87,89,90,92,93,95,96]
	检索增强生成	[86,87,90]

References:

[1] Wu N, Li YJ, Yang H, Chen HQ, Dai S, Hao C, Yu CX, Xie Y. Survey of machine learning for software-assisted hardware design verification: Past, present, and prospect. ACM Trans. on Design Automation of Electronic Systems, 2024, 29(4): 59. [doi: 10.1145/3661308]

[2] Jayasena A, Mishra P. Directed test generation for hardware validation: A survey. ACM Computing Surveys, 2024, 56(5): 132. [doi: 10.1145/3638046]

[3] Ismail KA, El Ghany MAA. Survey on machine learning algorithms enhancing the functional verification process. Electronics, 2021, 10(21): 2688. [doi: 10.3390/electronics10212688]

[4] Ioannides C, Eder KI. Coverage-directed test generation automated by machine learning—A review. ACM Trans. on Design Automation of Electronic Systems (TODAES), 2012, 17(1): 7. [doi: 10.1145/2071356.2071363]

[5] Bergeron J. Writing Testbenches: Functional Verification of HDL Models. 2nd ed., New York: Springer, 2003.

[6] Piziali A. Functional Verification Coverage Measurement and Analysis. New York: Springer, 2008.

[7] Bachrach J, Vo H, Richards B, Lee Y, Waterman A, Avižienis R, Wawrzynek J, Asanović K. Chisel: Constructing hardware in a Scala embedded language. In: Proc. of the 49th Annual Design Automation Conf. San Francisco: ACM, 2012. 1216–1225. [doi: 10.1145/2228360.2228584]

[8] Grimm T, Lettnin D, Hübner M. A survey on formal verification techniques for safety-critical systems-on-chip. Electronics, 2018, 7(6): 81. [doi: 10.3390/electronics7060081]

[9] Kropf T, Wunderlich H. A common approach to test generation and hardware verification based on temporal logic. In: Proc. of the 1991 Int'l Test Conf. Nashville: IEEE, 1991. 57. [doi: 10.1109/TEST.1991.519494]

[10] Bhadra J, Krishnamurthy N, Abadir MS. Enhanced equivalence checking: Toward a solidarity of functional verification and manufacturing test generation. IEEE Design and Test of Computers, 2004, 21(6): 494–502. [doi: 10.1109/MDT.2004.87]

[11] Yao GY, Zhang N, Tian C, Duan ZH, Liu LM, Sun FJ. Formal method of functional verification for chip development. Ruan Jian Xue Bao/Journal of Software, 2021, 32(6): 1799–1817 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6250.htm [doi: 10.13328/j.cnki.jos.006250]

[12] Cruz J, Farahmandi F, Ahmed A, Mishra P. Hardware Trojan detection using ATPG and model checking. In: Proc. of the 31st Int'l Conf. on VLSI Design and the 17th Int'l Conf. on Embedded Systems. Pune: IEEE, 2018. 91–96. [doi: 10.1109/VLSID.2018.43]

[13] Tang SB, Zhu JC, Gao YF, Zhou J, Mu DJ, Hu W. Verifying RISC-V privilege transition integrity through symbolic execution. In: Proc. of the 32nd IEEE Asian Test Symp. Beijing: IEEE, 2023. 1–6. [doi: 10.1109/ATS59501.2023.10317946]

[14] Bruns N, Herdt V, Drechsler R. Processor verification using symbolic execution: A RISC-V case-study. In: Proc. of the 2023 Design, Automation & Test in Europe Conf. & Exhibition. Antwerp: IEEE, 2023. 1–6. [doi: 10.23919/DATE56975.2023.10137202]

[15] Synopsys, Inc. VCS functional verification solution. 2024. https://www.synopsys.com/verification/simulation/vcs.html

[16] Chipsalliance/RISCV-DV. CHIPS Alliance. 2024. https://github.com/chipsalliance/riscv-dv

[17] Tasiran S, Fallah F, Chinnery DG, Weber SJ, Keutzer K. A functional validation technique: Biased-random simulation guided by observability-based coverage. In: Proc. of the 2001 IEEE Int'l Conf. on Computer Design: VLSI in Computers and Processors. Austin: IEEE, 2001. 82–88. [doi: 10.1109/ICCD.2001.955007]

[18] Guo Y, Qu WX, Li T, Li SK. Coverage driven test generation framework for RTL functional verification. In: Proc. of the 10th IEEE Int'l Conf. on Computer-aided Design and Computer Graphics. Beijing: IEEE, 2007. 321–326. [doi: 10.1109/CADCG.2007.4407902]

[19] Benjamin M, Geist D, Hartman A, Wolfsthal Y, Mas G, Smeets R. A study in coverage-driven test generation. In: Proc. of the 36th Annual ACM/IEEE Design Automation Conf. New Orleans: IEEE, 1999. 970–975. [doi: 10.1109/DAC.1999.782237]

[20] Laeufer K, Koenig J, Kim D, Bachrach J, Sen K. RFUZZ: Coverage-directed fuzz testing of RTL on FPGAs. In: Proc. of the 2018 IEEE/ACM Int'l Conf. on Computer-aided Design. San Diego: IEEE, 2018. 1–8. [doi: 10.1145/3240765.3240842]

[21] Canakci S, Delshadtehrani L, Eris F, Taylor MB, Egele M, Joshi A. DirectFuzz: Automated test generation for RTL designs using directed graybox fuzzing. In: Proc. of the 58th ACM/IEEE Design Automation Conf. San Francisco: IEEE, 2021. 529–534. [doi: 10.1109/DAC18074.2021.9586289]

[22] Trippel T, Shin KG, Kelly G, Rizzo D, Hicks M, Tech V. Fuzzing hardware like software. In: Proc. of the 31st USENIX Security Symp. Boston: USENIX Association, 2022. 3237–3254.

[23] Hur J, Song S, Kwon D, Baek E, Kim J, Lee B. DifuzzRTL: Differential fuzz testing to find CPU bugs. In: Proc. of the 2021 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2021. 1286–1303. [doi: 10.1109/SP40001.2021.00103]

[24] Xu JY, Liu YY, He SR, Lin HR, Zhou YJ, Wang C. MorFuzz: Fuzzing processor via runtime instruction morphing enhanced synchronizable co-simulation. In: Proc. of the 32nd USENIX Security Symp. Anaheim: USENIX Association, 2023. 1307–1324.

[25] Canakci S, Rajapaksha C, Delshadtehrani L, Nataraja A, Taylor MB, Egele M, Joshi A. ProcessorFuzz: Processor fuzzing with control and status registers guidance. In: Proc. of the 2023 IEEE Int'l Symp. on Hardware Oriented Security and Trust. San Jose: IEEE, 2023. 1–12. [doi: 10.1109/HOST55118.2023.10133714]

[26] Ur S, Yadin Y. Micro architecture coverage directed generation of test programs. In: Proc. of the 1999 Design Automation Conf. New Orleans: IEEE, 1999. 175–180. [doi: 10.1109/DAC.1999.781305]

[27] Nativ G, Mittennaier S, Ur S, Ziv A. Cost evaluation of coverage directed test generation for the IBM mainframe. In: Proc. of the 2001 Int'l Test Conf. Baltimore: IEEE, 2001. 793–802. [doi: 10.1109/TEST.2001.966701]

[28] Tasiran S, Keutzer K. Coverage metrics for functional validation of hardware designs. IEEE Design & Test of Computers, 2001, 18(4): 36–45. [doi: 10.1109/54.936247]

[29] Jou JY, Liu CNJ. Coverage analysis techniques for HDL design validation. 1999. https://www.researchgate.net/publication/266883269_Coverage_Analysis_Techniques_for_HDL_Design_Validation

[30] Mehta AB. ASIC/SoC Functional Design Verification. Cham: Springer, 2018.

[31] Witharana H, Lyu YD, Charles S, Mishra P. A survey on assertion-based hardware verification. ACM Computing Surveys (CSUR), 2022, 54(11s): 225. [doi: 10.1145/3510578]

[32] Fine S, Ziv A. Coverage directed test generation for functional verification using Bayesian networks. In: Proc. of the 40th Annual

Design Automation Conf. Anaheim: ACM, 2003. 286–291. [doi: 10.1145/775832.775907]

[33] Fine S, Fournier L, Ziv A. Using Bayesian networks and virtual coverage to hit hard-to-reach events. Int'l Journal on Software Tools for Technology Transfer, 2009, 11(4): 291–305. [doi: 10.1007/s10009-009-0119-0]

[34] Baras D, Fournier L, Ziv A. Automatic boosting of cross-product coverage using Bayesian networks. In: Proc. of the 4th Int'l Haifa Verification Conf. on Hardware and Software: Verification and Testing. Berlin: Springer, 2009. 53–67. [doi: 10.1007/978-3-642-01702-5_10]

[35] Squillero G. MicroGP—An evolutionary assembly program generator. Genetic Programming and Evolvable Machines, 2005, 6(3): 247–263. [doi: 10.1007/s10710-005-2985-x]

[36] Ioannides C, Barrett G, Eder K. Feedback-based coverage directed test generation: An industrial evaluation. In: Proc. of the 6th Int'l Conf. on Hardware and Software: Verification and Testing. Springer, 2010. 112–128. [doi: 10.5555/1987082.1987095]

[37] Wang JW, Liu ZG, Wang SL, Liu Y, Li YF, Yang H. Coverage-directed stimulus generation using a genetic algorithm. In: Proc. of the 2013 Int'l SoC Design Conf. Busan: IEEE, 2013. 298–301. [doi: 10.1109/ISOCC.2013.6864032]

[38] Imková M, Kotásek Z. Automation and optimization of coverage-driven verification. In: Proc. of the 2015 Euromicro Conf. on Digital System Design. Madeira: IEEE, 2015. 87–94. [doi: 10.1109/DSD.2015.34]

[39] Martínez-Cruz A, Barrón-Fernández R, Molina-Lozano H, Ramírez-Salinas MA, Villa-Vargas LA, Cortés-Antonio P, Cheng KT. An automatic functional coverage for digital systems through a binary particle swarm optimization algorithm with a reinitialization mechanism. Journal of Electronic Testing, 2017, 33(4): 431–447. [doi: 10.1007/s10836-017-5665-x]

[40] Roy R, Benipal MS, Godil S. Dynamically optimized test generation using machine learning. 2021. https://dvcon-proceedings.org/document/dynamically-optimized-test-generation-using-machine-learning/

[41] Huang QJ, Shojaei H, Zyda F, Nazi A, Vasudevan S, Chatterjee S, Ho R. Test parameter tuning with blackbox optimization: A simple yet effective way to improve coverage. 2022. https://dvcon-proceedings.org/wp-content/uploads/Test-Parameter-Tuning-with-Blackbox-Optimization-A-Simple-Yet-Effective-Way-to-Improve-Coverage-1.pdf

[42] Ismail KA, Abd El Ghany MA. High performance machine learning models for functional verification of hardware designs. In: Proc. of the 3rd Novel Intelligent and Leading Emerging Sciences Conf. Giza: IEEE, 2021. 15–18. [doi: 10.1109/NILES53778.2021.9600502]

[43] Elver M, Nagarajan V. McVerSi: A test generation framework for fast memory consistency verification in simulation. In: Proc. of the 2016 IEEE Int'l Symp. on High Performance Computer Architecture. Barcelona: IEEE, 2016. 618–630. [doi: 10.1109/HPCA.2016.7446099]

[44] Pan ZX, Mishra P. Automated test generation for hardware Trojan detection using reinforcement learning. In: Proc. of the 26th Asia and South Pacific Design Automation Conf. Tokyo: ACM, 2021. 408–413. [doi: 10.1145/3394885.3431595]

[45] Bhargav H, Vs V, Kumar B, Singh V. Enhancing testbench quality via genetic algorithm. In: Proc. of the 2021 IEEE Int'l Midwest Symp. on Circuits and Systems. Lansing: IEEE, 2021. 652–656. [doi: 10.1109/MWSCAS47672.2021.9531876]

[46] Chen C, Gohil V, Kande R, Sadeghi AR, Rajendran J. PSOFuzz: Fuzzing processors with particle swarm optimization. In: Proc. of the 2023 IEEE/ACM Int'l Conf. on Computer Aided Design. San Francisco: IEEE, 2023. 1–9. [doi: 10.1109/ICCAD57390.2023.10323913]

[47] Fajcik M, Smrz P, Zachariasova M. Automation of processor verification using recurrent neural networks. In: Proc. of the 18th Int'l Workshop on Microprocessor and SoC Test and Verification. Austin: IEEE, 2017. 15–20. [doi: 10.1109/MTV.2017.15]

[48] Wang FC, Zhu HB, Popli P, Xiao Y, Bodgan P, Nazarian S. Accelerating coverage directed test generation for functional verification: A neural network-based framework. In: Proc. of the 2018 Great Lakes Symp. on VLSI. Chicago: Association for Computing Machinery, 2018. 207–212. [doi: 10.1145/3194554.3194561]

[49] Cristescu MC, Bob C. Flexible framework for stimuli redundancy reduction in functional verification using artificial neural networks. In: Proc. of the 2021 Int'l Symp. on Signals, Circuits and Systems. Iasi: IEEE, 2021. 1–4. [doi: 10.1109/ISSCS52333.2021.9497443]

[50] Cristescu MC, Ciupitu D. Stimuli redundancy reduction for nonlinear functional verification coverage models using artificial neural networks. In: Proc. of the 2021 Int'l Semiconductor Conf. Romania: IEEE, 2021. 217–220. [doi: 10.1109/CAS52836.2021.9604141]

[51] Pfeifer N, Zimpel BV, Andrade GAG, dos Santos LCV. A reinforcement learning approach to directed test generation for shared memory verification. In: Proc. of the 2020 Design, Automation & Test in Europe Conf. Exhibition. Grenoble: IEEE, 2020. 538–543. [doi: 10.23919/DATE48585.2020.9116198]

[52] Hughes W, Srinivasan S, Suvarna R, Kulkarni M. Optimizing design verification using machine learning: Doing better than random. arXiv:1909.13168, 2019.

[53] Choi H, Huh I, Kim S, Ko J, Jeong C, Son H, Kwon K, Chai J, Park Y, Jeong J, Kim DS, Choi JY. Application of deep reinforcement learning to dynamic verification of DRAM designs. In: Proc. of the 58th ACM/IEEE Design Automation Conf. San Francisco: IEEE, 2021. 523–528. [doi: 10.1109/DAC18074.2021.9586282]

[54] Halim YM, Ismail KA, Abd El Ghany MA, Ibrahim SA, Halim YM. Reinforcement-learning based method for accelerating functional coverage closure of traffic light controller dynamic digital design. In: Proc. of the 32nd Int'l Conf. on Computer Theory and Applications. Alexandria: IEEE, 2022. 44–50. [doi: 10.1109/ICCTA58027.2022.10206069]

[55] Ohana E. Closing functional coverage with deep reinforcement learning: A compression encoder example. 2023. https://dvcon-proceedings.org/wp-content/uploads/1046-Closing-Functional-Coverage-With-Deep-Reinforcement-Learning-A-Compression-Encoder-Example-1.pdf

[56] Tweehuysen SL, Adriaans GLA, Gomony M. Stimuli generation for IC design verification using reinforcement learning with an Actor-Critic model. In: Proc. of the 2023 IEEE European Test Symp. Venezia: IEEE, 2023. 1–4. [doi: 10.1109/ETS56758.2023.10174129]

[57] Zhang ZX, Chadwick G, McNally H, Zhao YR, Mullins R. LLM4DV: Using large language models for hardware test stimuli generation. In: Proc. of the 37th Conf. and Workshop on Neural Information Processing Systems. 2023.

[58] Xiao C, Deng YF, Yang ZJ, Chen RZ, Wang H, Zhao JY, Dai HD, Wang L, Tang YH, Xu WX. LLM-based processor verification: A case study for neuromorphic processor. In: Proc. of the 2024 Design, Automation & Test in Europe Conf. & Exhibition. Valencia: IEEE, 2024. 1–6. [doi: 10.23919/DATE58400.2024.10546707]

[59] Rostami M, Chilese M, Zeitouni S, Kande R, Rajendran J, Sadeghi AR. Beyond random inputs: A novel ML-based hardware fuzzing. In: Proc. of the 2024 Design, Automation & Test in Europe Conf. & Exhibition. Valencia: IEEE, 2024. 1–6. [doi: 10.23919/DATE58400.2024.10546625]

[60] Katz Y, Rimon M, Ziv A, Shaked G. Learning microarchitectural behaviors to improve stimuli generation quality. In: Proc. of the 48th Design Automation Conf. San Diego: ACM, 2011. 848–853. [doi: 10.1145/2024724.2024914]

[61] Wen C, Wang LC, Bhadra J, Abadir MS. Novel test analysis to improve structural coverage—A commercial experiment. In: Proc. of the 2013 Int'l Symp. on VLSI Design, Automation, and Test. Hsinchu: IEEE, 2013. 1–4. [doi: 10.1109/VLDI-DAT.2013.6533851]

[62] Chen W, Wang LC, Bhadra J, Abadir M. Simulation knowledge extraction and reuse in constrained random processor verification. In: Proc. of the 50th Annual Design Automation Conf. Austin: ACM, 2013. 120. [doi: 10.1145/2463209.2488881]

[63] Hsieh KK, Siatkowski S, Wang LC, Chen W, Bhadra J. Feature extraction from design documents to enable rule learning for improving assertion coverage. In: Proc. of the 22nd Asia and South Pacific Design Automation Conf. Chiba: IEEE, 2017. 51–56. [doi: 10.1109/ASPDAC.2017.7858295]

[64] Miyamoto M, Hamaguchi K. Finding effective simulation patterns for coverage-driven verification using deep learning. In: Proc. of the 20th Workshop on Synthesis And System Integration of Mixed Information Technologies. 2016. 335–340.

[65] Ambalakkat SM, Nelson E. Simulation runtime optimization of constrained random verification using machine learning algorithms. 2017. https://dvcon-proceedings.org/wp-content/uploads/simulation-runtime-optimization-of-constrained-random-verification-using-machine-learning-algorithms.pdf

[66] Wang CA, Tseng CH, Tsai CC, Lee TY, Chen YH, Yeh CH, Yeh CS, Lai CT. Two stage framework for corner case stimuli generation using Transformer and reinforcement learning. 2022. https://dvcon-proceedings.org/wp-content/uploads/Two-stage-framework-for-corner-case-stimuli-generation-Using-Transformer-and-Reinforcement-Learning-1.pdf

[67] Yan M, Chen JJ, Mao HY, Jiang JJ, Hao JY, Li XJ, Tian Z, Chen ZC, Li D, Xian ZK, Guo YW, Liu WL, Wang B, Sun YF, Cui YS. Achieving last-mile functional coverage in testing chip design software implementations. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice. Melbourne: IEEE, 2023. 343–354. [doi: 10.1109/ICSE-SEIP58684.2023.00037]

[68] Shyam S, Bertacco V. Distance-guided hybrid verification with GUIDO. In: Proc. of the 2006 Design Automation & Test in Europe Conf. Munich: IEEE, 2006. 1–6. [doi: 10.1109/DATE.2006.244050]

[69] Chen MS, Mishra P. Functional test generation using efficient property clustering and learning techniques. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2010, 29(3): 396–404. [doi: 10.1109/TCAD.2010.2041846]

[70] Vasudevan S, Sheridan D, Patel S, Tcheng D, Tuohy B, Johnson D. GoldMine: Automatic assertion generation using data mining and static analysis. In: Proc. of the 2010 Design, Automation & Test in Europe Conf. & Exhibition. Dresden: IEEE, 2010. 626–629. [doi: 10.1109/DATE.2010.5457129]

[71] Liu LY, Lin CH, Vasudevan S. Word level feature discovery to enhance quality of assertion mining. In: Proc. of the 2012 IEEE/ACM Int'l Conf. on Computer-aided Design. San Jose: ACM, 2012. 210–217. [doi: 10.1145/2429384.2429424]

[72] Sheridan D, Liu LY, Kim H, Vasudevan S. A coverage guided mining approach for automatic generation of succinct assertions. In: Proc. of the 27th Int'l Conf. on VLSI Design and the 13th Int'l Conf. on Embedded Systems. Mumbai: IEEE, 2014. 68–73. [doi: 10.1109/VLSID.2014.19]

[73] Hanafy M, Said H, Wahba AM. Complete properties extraction from simulation traces for assertions auto-generation. In: Proc. of the

24th IEEE North Atlantic Test Workshop. Johnson City: IEEE, 2015. 1–6. [doi: 10.1109/NATW.2015.8]

[74] Soeken M, Harris CB, Abdessaied N, Harris IG, Drechsler R. Automating the translation of assertions using natural language processing techniques. In: Proc. of the 2014 Forum on Specification and Design Languages. Munich: IEEE, 2014. 1–8. [doi: 10.1109/FDL.2014.7119356]

[75] Harris CB, Harris IG. GLAsT: Learning formal grammars to translate natural language specifications into hardware assertions. In: Proc. of the 2016 Design, Automation & Test in Europe Conf. & Exhibition. Dresden: IEEE, 2016. 966–971.

[76] Gulliya N, Bora A, Chaudhary N, Kaur A. Using machine learning in register automation and verification. 2019. https://dvcon-proceedings.org/wp-content/uploads/using-machine-learning-in-register-automation-and-verification.pdf

[77] Parthasarathy G, Nanda S, Choudhary P, Patil P. SpecToSVA: Circuit specification document to SystemVerilog assertion translation. 2021. https://document-intelligence.github.io/DI-2021/files/di-2021_final_19.pdf

[78] Aditi F, Hsiao MS. Hybrid rule-based and machine learning system for assertion generation from natural language specifications. In: Proc. of the 31st IEEE Asian Test Symp. Taichung: IEEE, 2022. 126–131. [doi: 10.1109/ATS56056.2022.00034]

[79] Vasudevan S, Jiang WJ, Bieber D, Singh R, Shojaei H, Ho R, Sutton C. Learning semantic representations to verify hardware designs. In: Proc. of the 35th Int'l Conf. on Neural Information Processing Systems. Curran Associates Inc., 2021. 1799.

[80] Liu MJ, Ene TD, Kirby R, *et al*. ChipNeMo: Domain-adapted LLMs for chip design. arXiv:2311.00176, 2024.

[81] Fu WM, Li SJ, Zhao YF, Ma HC, Dutta R, Zhang X, Yang KC, Jin YE, Guo XL. Hardware Phi-1.5B: A large language model encodes hardware domain specific knowledge. In: Proc. of the 29th Asia and South Pacific Design Automation Conf. Incheon: IEEE, 2024. 349–354. [doi: 10.1109/ASP-DAC58780.2024.10473927]

[82] Orenes-Vera M, Martonosi M, Wentzlaff D. Using LLMs to facilitate formal verification of RTL. arXiv:2309.09437, 2023.

[83] Sun CY, Hahn C, Trippel C. Towards improving verification productivity with circuit-aware translation of natural language to SystemVerilog assertions. 2023. https://openreview.net/pdf?id=FKH8qCuM44

[84] Kande R, Pearce H, Tan B, Dolan-Gavitt B, Thakur S, Karri R, Rajendran J. LLM-assisted generation of hardware assertions. arXiv:2306.14027v1, 2023.

[85] Fang WJ, Li MM, Li M, Yan ZY, Liu S, Xie ZY, Zhang HC. AssertLLM: Generating and evaluating hardware verification assertions from design specifications via multi-LLMs. arXiv:2402.00386v3, 2024.

[86] Paria S, Dasgupta A, Bhunia S. DIVAS: An LLM-based end-to-end framework for SoC security analysis and policy-based protection. arXiv:2308.06932, 2023.

[87] Tsai Y, Liu MJ, Ren HX. RTLFixer: Automatically fixing RTL syntax errors with large language model. In: Proc. of the 61st ACM/IEEE Design Automation Conf. San Francisco: ACM, 2024. 53. [doi: 10.1145/3649329.3657353]

[88] Xu K, Sun JL, Hu YC, Fang XW, Shan WW, Wang X, Jiang Z. MEIC: Re-thinking RTL debug automation using LLMs. arXiv:2405.06840, 2024.

[89] Ma RY, Yang YX, Liu ZQ, Zhang JX, Li M, Huang JH, Luo GJ. VerilogReader: LLM-aided hardware test generation. In: Proc. of the 2024 IEEE LLM Aided Design Workshop. San Jose: IEEE, 2024. 1–5. [doi: 10.1109/LAD62341.2024.10691801]

[90] Saha D, Tarek S, Yahyaei K, Saha SK, Zhou JB, Tehranipoor M, Farahmandi F. LLM for SoC security: A paradigm shift. IEEE Access, 2024, 12: 155498–155521. [doi: 10.1109/ACCESS.2024.3427369]

[91] Meng XY, Srivastava A, Arunachalam A, Ray A, Silva PH, Psiakis R, Makris Y, Basu K. Unlocking hardware security assurance: The potential of LLMs. arXiv:2308.11042, 2023.

[92] Fu WM, Yang KC, Dutta RG, Guo XL, Qu G. LLM4SecHW: Leveraging domain-specific large language model for hardware debugging. In: Proc. of the 2023 Asian Hardware Oriented Security and Trust Symp. Tianjin: IEEE, 2023. 1–6. [doi: 10.1109/AsianHOST59942.2023.10409307]

[93] Ahmad B, Thakur S, Tan B, Karri R, Pearce H. Fixing hardware security bugs with large language models. arXiv:2302.01215, 2023.

[94] Ahmad H, Huang Y, Weimer W. CirFix: Automatically repairing defects in hardware design code. In: Proc. of the 27th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Lausanne: ACM, 2022. 990–1003. [doi: 10.1145/3503222.3507763]

[95] Tarek S, Saha D, Saha SK, Tehranipoor M, Farahmandi F. SoCureLLM: An LLM-driven approach for large-scale system-on-chip security verification and policy generation. 2024. https://eprint.iacr.org/2024/983.pdf

[96] Saha D, Yahyaei K, Kumar Saha S, Tehranipoor M, Farahmandi F. Empowering hardware security with LLM: The development of a vulnerable hardware database. In: Proc. of the 2024 IEEE Int'l Symp. on Hardware Oriented Security and Trust. Tysons Corner: IEEE, 2024. 233–243. [doi: 10.1109/HOST55342.2024.10545393]

[97] RISCV-software-src/RISCV-ISA-sim. RISC-V Software. 2024. https://github.com/riscv-software-src/riscv-isa-sim

[98] Herdt V, Große D, Jentzsch E, Drechsler R. Efficient cross-level testing for processor verification: A RISC-V case-study. In: Proc. of the 2020 Forum for Specification and Design Languages. Kiel: IEEE, 2020. 1–7. [doi: 10.1109/FDL50818.2020.9232941]

[99] Bruns N, Herdt V, Jentzsch E, Drechsler R. Cross-level processor verification via endless randomized instruction stream generation with coverage-guided aging. In: Proc. of the 2022 Design, Automation & Test in Europe Conf. & Exhibition. Antwerp: IEEE, 2022. 1123–1126. [doi: 10.23919/DATE54114.2022.9774771]

[100] Liu J. Metamorphic testing and its application on hardware fault-tolerance [Ph.D. Thesis]. Madison: University of Wisconsin, 2011.

[101] Hassan M, Große D, Drechsler R. System-level verification of linear and non-linear behaviors of RF amplifiers using metamorphic relations. In: Proc. of the 26th Asia and South Pacific Design Automation Conf. Tokyo: ACM, 2021. 761–766. [doi: 10.1145/3394885.3431592]

[102] Riese F, Herdt V, Große D, Drechsler R. Metamorphic testing for processor verification: A RISC-V case study at the instruction level. In: Proc. of the 29th IFIP/IEEE Int'l Conf. on Very Large Scale Integration. Singapore: IEEE, 2021. 1–6. [doi: 10.1109/VLSI-SoC53125.2021.9606997]

[103] Hazott C, Stögmüller F, Große D. Verifying embedded graphics libraries leveraging virtual prototypes and metamorphic testing. In: Proc. of the 29th Asia and South Pacific Design Automation Conf. Incheon: IEEE, 2024. 275–281. [doi: 10.1109/ASP-DAC58780.2024.10473799]

[104] Guzey O, Wang LC, Levitt JR, Foster H. Increasing the efficiency of simulation-based functional verification through unsupervised support vector analysis. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2010, 29(1): 138–148. [doi: 10.1109/TCAD.2009.2034347]

[105] Chang PH, Drmanac D, Wang LC. Online selection of effective functional test programs based on novelty detection. In: Proc. of the 2010 IEEE/ACM Int'l Conf. on Computer-aided Design. San Jose: IEEE, 2010. 762–769. [doi: 10.1109/ICCAD.2010.5653868]

[106] Wang SP, Huang K. Improving the efficiency of functional verification based on test prioritization. Microprocessors and Microsystems, 2016, 41: 1–11. [doi: 10.1016/j.micpro.2015.12.001]

[107] Choi J, Noh S, Hong S, Jang H, Yim S, Choi SB. Finding a needle in a haystack: A novel log analysis method with test clustering in distributed systems. 2022. https://dvcon-proceedings.org/wp-content/uploads/Finding-a-Needle-in-a-Haystack-A-Novel-Log-Analysis-Method-with-Test-Clustering-in-Distributed-System-1.pdf

[108] Liang RJ, Pinckney N, Chai YJ, Ren HX, Khailany B. Late breaking results: Test selection for RTL coverage by unsupervised learning from fast functional simulation. In: Proc. of the 60th ACM/IEEE Design Automation Conf. San Francisco: IEEE, 2023. 1–2. [doi: 10.1109/DAC56929.2023.10247936]

[109] Blackmore T, Hodson R, Schaal S. Novelty-driven verification: Using machine learning to identify novel stimuli and close coverage. 2021. https://dvcon-proceedings.org/document/novelty-driven-verification-using-machine-learning-to-identify-novel-stimuli-and-close-coverage/

[110] Masamba N, Eder K, Blackmore T. Hybrid intelligent testing in simulation-based verification. In: Proc. of the 2022 IEEE Int'l Conf. on Artificial Intelligence Testing. Newark: IEEE, 2022. 26–33. [doi: 10.1109/AITest55621.2022.00013]

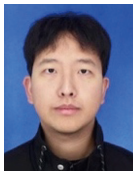
[111] Parthasarathy G, Rushdi A, Choudhary P, Nanda S, Evans M, Gunasekara H, Rajakumar S. RTL regression test selection using machine learning. In: Proc. of the 27th Asia and South Pacific Design Automation Conf. Taipei: IEEE, 2022. 281–287. [doi: 10.1109/ASP-DAC52403.2022.9712550]

[112] Zheng X, Eder K, Blackmore T. Using neural networks for novelty-based test selection to accelerate functional coverage closure. In: Proc. of the 2023 IEEE Int'l Conf. on Artificial Intelligence Testing. Athens: IEEE, 2023. 114–121. [doi: 10.1109/AITest58265.2023.00026]

[113] Singh K, Gupta R, Gupta V, Fayyazi A, Pedram M, Nazarian S. A hybrid framework for functional verification using reinforcement learning and deep learning. In: Proc. of the 2019 on Great Lakes Symp. on VLSI. Tysons Corner: Association for Computing Machinery, 2019. 367–370. [doi: 10.1145/3299874.3318039]

附中文参考文献:

[11] 姚广宇, 张南, 田聪, 段振华, 刘灵敏, 孙风津. 芯片开发功能验证的形式化方法. 软件学报, 2021, 32(6): 1799–1817. http://www.jos.org.cn/1000-9825/6250.htm [doi: 10.13328/j.cnki.jos.006250]



李晓鹏(2002－), 男, 硕士生, CCF 学生会员, 主要研究领域为软件测试, 芯片设计程序测试.

闫明(1996－), 男, 博士生, 主要研究领域为深度学习系统测试, 芯片设计程序测试.

樊兴宇(2001－), 男, 硕士生, 主要研究领域为编译器测试, 基于大语言模型的电子设计自动化.

唐振韬(1992－), 男, 博士, 主要研究领域为深度强化学习, EDA 物理布局优化.

开昱雄(1994－), 男, 硕士, 主要研究领域为强化学习, EDA 物理设计.

郝建业(1986－), 男, 博士, CCF 专业会员, 主要研究领域为深度强化学习, 多智能体系统.

袁明轩(1980－), 男, 博士, CCF 专业会员, 主要研究领域为学习优化, AI4EDA, AI 求解器.

陈俊洁(1992－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件分析与测试.