SoC 高级综合验证研究进展

胡健¹⁾, 胡永扬¹⁾, 王观武¹⁾, 陈桂林¹⁾, 杨海涛¹⁾, 康云¹⁾, 王康¹⁾, 李思昆²⁾

¹⁾ (国防科学技术大学第六十三研究所 南京 210000)

²⁾ (国防科学技术大学计算机学院 长沙 410000)

(hujian198681@126.com)

摘 要: 针对近年来片上系统(system on chip, SoC)高级综合验证领域的工作, 首先分析了高级综合验证的难点, 然后根据应用领域将算法分为 3 类: 高级综合前端验证算法、高级综合调度验证算法和高级综合后端验证算法。同时分析了各类算法的优缺点和现有算法的主要技术手段; 最后讨论了 SoC 高级综合验证算法目前面临的映射关系缺失、状态空间爆炸和复杂的数据结构等挑战, 并对该领域今后的研究方向进行了展望。

关键词: 高级综合; 带数据通路的有限状态机; 互模拟关系; 路径验证; 形式化方法

中图法分类号: TP391.7

DOI: 10.3724/SP.J.1089.2021.18394

A Survey of Verification for High-level Synthesis

Hu Jian¹⁾, Hu Yongyang¹⁾, Wang Guanwu¹⁾, Chen Guilin¹⁾, Yang Haitao¹⁾, Kang Yun¹⁾, Wang Kang¹⁾, and Li Sikun²⁾

¹⁾ (Sixty-third Institute, National University of Defense Technology, Nanjing 210000)

²⁾ (School of Computer, National University of Defense Technology, Changsha 410000)

Abstract: For the recent research in the verification of high-level synthesis for SoC, this paper analyzes the difficulties on formal verification for high-level synthesis, and classifies the recent research works to 3 classes according to the algorithm types. The algorithms are classified to high-level synthesis verification algorithms for front-end, high-level synthesis verification algorithms for scheduling and high-level synthesis verification algorithms for back-end. Then, the advantages, disadvantages and the used techniques of the existing algorithms are analyzed. Finally, the existing challenges including lack of mapping information, state explosion and complex data structure and future research directions in formal verification for high-level synthesis are discussed.

Key words: high-level synthesis; finite state machine with datapath; bisimulation; path verification; formal method

目前, 片上系统(system on chip, SoC)设计的复杂度越来越高, 大量设计已无法直接从寄存器传输级(register transfer level, RTL)开始, 对高层行

为和体系结构模型的需求越来越迫切, 系统级功能验证已经成为影响 SoC 设计效率和质量的最重要的环节。在电子系统设计流程中, 设计人员采用自

收稿日期: 2020-05-20; 修回日期: 2020-08-23. 基金项目: 国家自然科学基金(61902421). 胡健(1986—), 男, 博士, 助理研究员, 主要研究方向为形式化验证、高级综合、高层等价性检验; 胡永扬(1977—), 男, 硕士, 高级工程师, 主要研究方向为形式化技术和 ASIC 设计; 王观武(1986—), 男, 博士, 工程师, 主要研究方向为粗粒度可重构体系结构、高级编译; 陈桂林(1994—), 男, 硕士, 工程师, 主要研究方向为神经网络加速器; 杨海涛(1977—), 男, 硕士, 副研究员, 主要研究方向为微处理器设计与验证; 康云(1980—), 女, 硕士, 工程师, 主要研究方向为并行模拟、电子 CAD; 王康(1990—), 男, 硕士, 工程师, 主要研究方向为电子设计自动化; 李思昆(1941—), 男, 学士, 教授, 博士生导师, CCF 高级会员, 主要研究方向为电子设计自动化、SoC 设计方法学和虚拟现实。

顶向下的方式利用高级设计语言(C/C++/SystemC)进行系统级建模并逐步精化到 RTL(Verilog/VHDL). 其转换过程通过高级综合工具实现.

高级综合是将电路设计规范的算法级或行为级描述在一定的约束条件下转化为电路结构描述的方法和过程^[1]. 高级综合过程可以分为高级综合前端、高级综合调度和高级综合后端 3 个阶段. 高级综合前端进行常数传播、循环展开、公共子式消除和死代码消除等编译优化, 目的是降低产生代码的复杂性, 最大化数据局部性, 将设计转换为更适宜资源分配和控制综合的形式; 高级综合调度将进行代码移动, 操作调度, 将操作分配到特定的时钟周期; 高级综合后端进行硬件资源绑定和控制综合, 根据资源分配产生 RTL 代码并进一步优化时序和功耗^[2]. 目前主流的高级综合工具(如 SPARK, AutoESL^[3])都是庞大的软件系统, 高级综合过程以及高级综合工具因其复杂性, 极易引入错误. 因此, 保证 SoC 在高级综合的各个阶段功能的一致性至关重要. 高级综合过程中各个层次之间存在巨大的时序与结构差异, 其功能验证存在诸多困难和迫切需要解决的科学问题, 主要包括以下几方面.

(1) 高级综合前端进行的编译优化、死代码删除、公共子式消除等优化操作, 导致部分代码映射关系缺失, 循环展开导致验证规模变大, 验证难度较大.

(2) 高级综合调度中的非均匀代码移动(推测、反向推测和安全推测等)以及路径的合并和分割将导致设计控制结构改变, 进一步增加验证的难度.

(3) 高级综合调度中跨越循环体的代码调度会导致映射关系消失, 非定次的循环无法进行循环展开, 跨越非定次循环的代码调度成为验证中最难的部分.

(4) 高级综合后端缺乏中间表达(intermediate representation, IR)与 RTL 代码的映射关系, 直接进行功能等价性验证非常困难, 需要设计间的映射关系.

高级综合验证已成为电子设计自动化和计算机辅助设计中非常活跃的研究领域, 本文首先分析高级综合的层次划分; 然后从算法类型归类角度对各种高级综合验证方法进行概述评论, 同时分析各类算法的优缺点和现有算法的主要技术手段; 最后讨论 SoC 高级综合验证方法目前面临的挑战, 并对该领域今后的研究方向进行展望.

1 高级综合过程

高级综合通常根据硬件资源和时序需求将系统级描述转换为 RTL 描述^[2]. 系统级描述和 RTL 描述之间的主要区别在于实现的详细程度. 系统级描述主要描述设计的功能, 较少涉及具体设计细节, 能够进行快速的设计验证; RTL 为时钟周期精确, 需要描述每个时钟的操作. RTL 包括了设计的所有细节, 其模拟速度比系统级慢几个数量级. 从系统级描述到 RTL 描述的高级综合通常需要进行以下步骤^[4].

(1) 编译转换. 将高层设计描述转换为控制数据流图(control/data flow graph, CDFG), 使其更适合资源绑定和控制综合.

(2) 代码优化. 进行代码优化, 降低代码复杂性, 最大化数据局部性, 包括死代码消除、常数传播、公共子式消除和循环展开等.

(3) 调度. 将操作分配给控制步骤, 即同步硬件设计的时钟周期. 调度结果需要满足时序和资源的要求, 每一个操作都会映射到一个硬件资源.

(4) 分配. 计算高级综合需要的最小功能单元和寄存器数量, 从部件库里选择实现功能的硬件单元, 包括算术运算单元、存储(寄存器和存储器)和互连资源(总线、多路复用器等).

(5) 绑定. 绑定操作、数据对象(变量和数据结构)、数据传输到功能单元、存储元素(寄存器和存储块)和互联. 绑定分为功能单元绑定、存储元素绑定和互联绑定.

(6) 代码产生. 根据 IR 和功能单元的映射关系, 产生 RTL 代码.

目前, 已存在许多商用的高级综合工具, 它们通常需要对输入的编程语言进行扩展或限制. 这些工具包括 Mentor Graphics 公司的 CatapultC, ForteDesign Systems 公司的 Cynthesizer, Synopsys 公司的 Synphony HLS, Cadence 公司的 C-to-silicon, C 到 Verilog 的编译器 C-to-Verilog, AutoESL 公司的 AutoPilot 等.

2 高级综合验证层次

如图 1 所示, 高级综合过程分为高级综合前端, 高级综合调度和高级综合后端. 高级综合前端进行代码转换和优化, 将系统级代码转换为控制流图(control flow graph, CFG)和数据流图(data flow graph, DFG), 最终得到 CDFG. 同时会进行代码优

化. 高级综合调度会根据时序和资源要求, 将操作分配给控制步骤. 高级综合后端进行资源分配和绑定, 根据映射关系产生 RTL 代码. 高级综合的每个阶段都会对上一层级表达进行优化转换并产生下一层级表达. 因此, 验证高级综合每个阶段前后设计的功能等价性至关重要. 根据高级综合层次划分, 验证层次可分为高级综合前端验证、高级综合调度验证和高级综合后端验证. 下面根据高级综合验证层次来阐述近年高级综合验证的研究进展.

图1 高级综合验证层次划分

3 高级综合前端验证算法

3.1 高级综合前端代码优化方法

高级综合前端进行代码转换和代码优化, 包括死代码消除、常数传播、公共子式消除和循环展开等. 死代码消除^[5]是一种编译最优化技术, 它的用途是移除对程序运行结果没有任何影响的代码. 移除这类代码不但可以缩小程序, 还可以避免程序在运行中执行不相关的运算行为, 缩短运行时间. 常数传播使用控制流和数据流信息确定变量在程序中的特定点必须具有特定的常量值. 公共子式消除技术通过使用先前计算的值来消除重新计算相同表达式的过程. 循环展开通过减少迭代次数和复制循环主体来减少循环开销. 设计经过

一系列高级综合前端优化, 得到优化后的 IR. 高级综合前端验证就是验证前端优化后 IR 与系统级设计的功能一致性.

3.2 综合前端验证算法

文献[6]使用有限状态机对模型进行建模, 利用符号模拟产生系统级与编译前端优化后表达的所有状态序列, 根据状态序列最弱前置条件选择和比较相容状态序列的等价性, 若所有相容状态序列等价, 则设计等价. 该方法产生了设计的所有可能路径, 存在路径爆炸的问题, 无法处理大规模设计的验证. 为了解决上述方法路径爆炸的问题, 文献[7]使用了函数模块化推理技术, 先验证局部模块的等价性, 再将等价模块替换为无解释函数符号; 同时使用了循环分割技术将循环分割为循环入口、循环体和循环出口, 进行 3 次验证, 即可验证循环结构, 避免了循环展开导致的路径爆炸. 但是该方法要求被验证设计拥有相同的循环结构, 不能进行部分循环展开. 文献[8-10]通过确定潜在等价点的方法对设计进行划分, 使用符号模拟对设计进行验证, 增强了算法的处理能力. 文献[11-15]只验证代码的文本差异区域, 减少扩展和验证次数, 提高了验证效率.

4 高级综合调度验证算法

4.1 高级综合代码调度技术

高级综合调度根据数据相关性以及时序、面积和延迟等约束将操作分配到特定时钟周期. 调度通常会改变设计的控制结构, 破坏设计的映射关系, 增加验证难度. 调度主要分为时间约束的调度和资源约束的调度. 时间约束的调度根据规定的时间约束分配最少的硬件资源, 资源约束的调度根据硬件资源约束使用最少的时钟周期进行调度. 代码移动是高级综合调度过程常用的方法, 其主要目的为减少计算的数量和最小化临时变量的生存周期. 图2所示为高级综合调度中常用的代码移动技术, 数字代表不同的移动技术, 下面分别对各种技术进行解释.

(1) 向下复制/向上复制. 它将操作从分支之前和之后的基本块移至所有分支基本块, 如图2中的虚线①和虚线②所示.

(2) 向上合并/向下合并. 如果所有的分支基本块都有相同的操作, 将这些操作从所有的分支基本块移至分支前和之后的基本块, 如图2中的虚线③和虚线④所示.

图2 高级综合代码调度技术

(3) 有效移动. 将操作从分支之前的基本块移至分支之后的基本块或从分支之后的基本块移至分支之前的基本块(如果这些操作与所有的分支基本块无关), 如图2中虚线⑤所示。

(4) 推测. 将不改变整体输出行为的代码, 从部分分支基本块移至分支状态之前的基本块, 如图2中虚线⑥所示。

(5) 反向推测. 将不改变整体输出行为的代码, 从分支状态之前的基本块移至部分分支基本块, 如图2中虚线⑦所示。

(6) 跨循环移动. 将代码从循环体之前的基本块移至循环体之后的基本块, 如图2中虚线⑧所示。

a. 系统级模型

b. 高级综合后模型

图3 计算最大公约数算法的FSMD模型

本文根据高级综合调度验证算法的特点, 将验证算法分为基于互模拟的验证算法、基于路径的验证算法和基于符号值传播的验证算法。基于互模拟的验证算法需要验证设计的对应割点存在互模拟关系; 基于路径的验证算法将设计划分为有限的路径集合, 验证设计对应路径的等价性; 基于符号值传播的验证算法将不等价的符号值传播到后续路径, 直到所有变量等价或最终路径不等价。

4.2 基于互模拟的验证算法

4.2.1 互模拟关系定义

互模拟验证算法中, 使用带数据通路的有限状态机^[16](finite state machine with datapath, FSMD)表示调度前后设计。FSMD是一种高效表达硬件设计和计算机程序的抽象形式, 定义为一个有序元组 $\langle Q, q_0, I, O, V, f, h \rangle$ 。其中, Q 为有限状态集合; q_0 为初始状态; I 为输入变量集合; O 为输出变量集合; V 为存储变量集合; $f: Q \times S \rightarrow Q$ 为迁移函数, S 为作用在 $I \cup U$ 的谓词, 表示状态表达式; $h: Q \times S \rightarrow U$ 为输出和存储变量的更新函数, U 为对输出和存储变量的赋值。

FSMD为确定的有限状态机, 用 M 表示, 其迁移边上的表达式为 s/h , 对于任意的状态 q , 如果迁移条件 $S_1 = S_2$, 则 $f(s_1, q) = f(s_2, q), h(s, q) = h(s, q)$ 。图3所示为计算最大公约数算法的FSMD模型。FSMD是能够提取被测设计的功能模型, 消除时序与代码结构差异, 降低验证难度; 被测设计的等价性验证转换为有限状态机的等价性验证。

定义 1. 互模拟路径. 从状态 q_i 和 q_j 的模拟路径 p 为一个状态的序列. 其中, $q_i, q_j \in Q$, $\langle q_i = q_1 \xrightarrow{c_1} q_2 \xrightarrow{c_2} \cdots \xrightarrow{c_{n-2}} q_{n-1} \xrightarrow{c_{n-1}} q_n = q_j \rangle$, $\longrightarrow$ 表示状态的迁移. $\forall l, 1 \leq l \leq n-1, \exists c_l \in S$ 且 $f(q_l, c_l) = q_{l+1}$. FSMD 的所有路径定义为 $\mathcal{S}_1$.

定义 2. 验证关系. $M_0 = \langle Q_0, q_{00}, I_0, O_0, V_0, f_0, h_0 \rangle$ 与 $M_1 = \langle Q_1, q_{10}, I_1, O_1, V_1, f_1, h_1 \rangle$ 的验证关系为一个元组 $\langle q_{0i}, q_{1j}, \varphi \rangle$. 其中, $q_{0i} \in Q_0$, $q_{1j} \in Q_1$, φ 是状态 q_{0i} 和 q_{1j} 上变量的谓词, $\varphi(\sigma_{0i}, \sigma_{1j}) = \text{TRUE}$ 表示 φ 在 $(\sigma_{0i}, \sigma_{1j}) \in (V_0 \times V_1)$ 上满足.

定义 3. 模拟关系. $M_0 = \langle Q_0, q_{00}, I_0, O_0, V_0, f_0, h_0 \rangle$ 与 $M_1 = \langle Q_1, q_{10}, I_1, O_1, V_1, f_1, h_1 \rangle$ 的验证关系 $R \langle q_{00}, q_{10}, \text{TRUE} \rangle$ 满足条件

$$\begin{aligned} R \langle q_{00}, q_{10}, \text{TRUE} \rangle: \\ \forall (q_1, q_2, q'_1, \sigma_1, \sigma_2, \sigma'_1, \varphi, \eta_1) \in \\ (Q_1, Q_2, Q'_1, V_1, V_2, V'_1, \phi, \mathcal{S}_1) \\ \left[\langle q_1, \sigma_1 \rangle \xrightarrow{\eta_1} \langle q'_1, \sigma'_1 \rangle \wedge R \langle q_1, q_2, \varphi \rangle \wedge \varphi(\sigma_1, \sigma_2) = \right. \\ \left. \text{TRUE} \right] \Rightarrow \exists (q'_2, \sigma'_2, \varphi', \eta_2) \in \\ (Q_2 \times V_2 \times \phi \times \mathcal{S}_2) \varphi' \left[\langle q_2, \sigma_2 \rangle \xrightarrow{\eta_2} \langle q'_2, \sigma'_2 \rangle \wedge \right. \\ \left. R \langle q'_1, q'_2, \varphi' \rangle \wedge \varphi'(\sigma'_1, \sigma'_2) = \text{TRUE} \wedge \eta_1 \equiv \eta_2 \right]. \end{aligned}$$

则称为模拟关系.

定义 4. 互模拟关系. 验证关系 R 是 M_0 和 M_1 的互模拟关系, 当且仅当 R 是 M_0 和 M_1 的模拟关系且 $R^{-1} = \{ \langle q_2, q_1, \varphi \rangle \mid R \langle q_1, q_2, \varphi \rangle \}$ 是 M_1 和 M_0 的模拟关系.

模拟关系和互模拟关系是包含了额外条件的验证关系, 本质上验证了 FSMD 对应状态的等价性. 因此, 得到互模拟关系即可得到设计的等价关系.

定义 5. 如果 M_0 和 M_1 存在互模拟关系, 则 M_0 与 M_1 等价, 表示为 $M_0 \simeq M_1$.

4.2.2 互模拟验证算法

互模拟验证算法可以确定具有相同分支结构的迁移系统的功能一致性, 并能进行状态机的逐步互模拟^[17]. 本质上, 如果迁移系统 T_0 的每一步都可以与迁移系统 T_1 中的一步(或多步)匹配, 则迁移系统 T_0 可以模拟迁移系统 T_1 . 互模拟等价性表示迁移系统进行相互逐步模拟的可能性. 基于互模拟的验证算法已经被应用于多个领域, 如标记迁移系统^[18]、并行系统^[19]、时序系统^[20]、结构图^[21]和概率过程^[22-23].

文献[24]提出了一种使用标记片段(labelled segments, LS)的符号模拟来自动验证编译调度. 该算法使用标记片段语言(language of labelled segments, LSS)对设计进行统一描述, 验证调度前后设计描述的互模拟关系. 如果每一个 LS 输出一致, 则 2 个设计功能一致. 文献[25]提出的互模拟验证过程包含推理和检验 2 个步骤. 推理算法通过前向扫描设计代码获取一系列的约束, 检验算法验证约束的正确性. 文献[26]改进了文献[25]中的方法, 提出利用通用求解器代替特定求解器和简化算法进行约束验证, 增加了算法的模块性和求解能力. 文献[27-28]提出了基于 PRES+^[29]模型的高级综合验证算法, 它是一种高效的硬件设计表达方式, 能够有效地建立设计的并行行为. 该算法使用 PRES+模型表示硬件设计, 利用割点划分 PRES+模型并验证对应割点的互模拟关系. 文献[30]实现了文献[27-28]的算法, 提出了基于 Eclipse 插件的高级综合验证工具, 该工具能够验证 PRES+模型的功能等价性. 文献[31]进一步改进了该工具, 使其能够处理并行程序的同步转换.

文献[32]提出了高级综合工具 SPARK^[33]调度前后设计的等价性验证算法. 该算法需要前向扫描被测设计并产生约束条件, 再使用迭代算法增强约束条件, 当约束条件到达固定点时验证模拟关系. 该算法需要在每一次迭代时验证关系的等价性, 导致大量求解器运算, 同时该算法的迭代过程不一定终止. 文献[34]重新定义了互模拟关系并在其基础上提出了新的互模拟验证算法, 它无需迭代, 能够极大地减少求解器验证次数, 提高验证效率.

以上互模拟算法都无法验证非结构保持的代码移动, 如基于路径的调度器^[35-36]产生的路径合并等操作. 文献[37]提出了解决非结构保持转换的互模拟等价性验证算法, 该算法基于 FSMD 构建层次图, 从层次图获取非结构保持的转换信息, 然后建立互模拟关系, 最后使用最弱前置条件^[38]计算方法验证互模拟关系的等价性. 文献[39]提出使用共享值图(shared value graph, SVG)的方法验证互模拟关系的等价性, 该方法首先将原始描述与调度后的描述转换为 CFG 与 DFG, 并插入割点^[40], 利用 SVG 验证割点的等价性, SVG 可以一次性验证循环结构, 无需进行固定点迭代, 有效地解决了循环结构的验证问题. 基于互模拟的方法可以解决代码循环移动问题, 但验证非结构保持的调度需要确定被测设计进行的调度操作, 并根据这些

信息建立互模拟关系,这种调度信息在实际综合过程中很难得到.目前,验证非结构保持的调度仍然是互模拟方法需要进一步研究的方向.

4.3 基于路径等价的验证算法

4.3.1 路径等价和 FSM D 等价

定义 6. 路径条件 R_α . 路径 α 的执行条件 R_α 为路径 α 的最弱前置条件. 其表示为存储变量集合 V 和输入集合 I 中变量的逻辑表达式, 当路径 α 执行时, R_α 在路径 α 的初态被满足.

定义 7. 路径数据转换 r_α . 路径 α 的数据转换 r_α 为元组 $\langle s_\alpha, o_\alpha \rangle$. s_α 表示路径 α 的存储变量转换, 是存储变量集合 V 和输入集合 I 的代数表达式 $\langle e_i \rangle$, 其中, e_i 表示从初始状态执行路径 α 后变量 v_i 的值; o_α 表示路径 α 的输出列表.

定义 8. FSM D 的路径覆盖. $P = \{p_0, p_1, p_2, \dots, p_k\}$ 为一个有限的路径集合, 如果对 M 中任意一条从初态到终态的路径都能表示为 P 中路径的连接, 则 P 为 M 的路径覆盖.

定义 9. FSM D 路径等价. p_0 为 M_0 的路径, p_1 为 M_1 的路径, R_0 和 s_0 表示路径 p_0 的执行条件和数据转换, R_1 和 s_1 表示路径 p_1 的执行条件和数据转换, 如果 $R_0 \equiv R_1$ 且 $s_0 \equiv s_1$, 则路径 p_0 和路径 p_1 等价, 表示为 $p_0 \approx p_1$.

定义 10. FSM D 状态机包含. 令 $P_0 = \{p_{00}, p_{01}, p_{02}, \dots, p_{0k}\}$ 为 M_0 的路径覆盖集合, 如果存在状态机的路径集合 $P_1 = \{p_{10}, p_{11}, p_{12}, \dots, p_{1k}\}$, 其中, $p_{0i} \approx p_{1i}$, $0 \leq i \leq k$; 则 M_1 包含 M_0 , 表示为 $M_1 \supseteq M_0$.

定义 11. FSM D 状态机等价. 如果 M_0 包含 M_1 , 且 M_1 包含 M_0 , 则 M_1 和 M_0 等价, 表示为 $M_0 \approx M_1$.

4.3.2 验证算法

目前国际上已有许多基于路径等价的验证算法, 文献[41]在 FSM D 中引入割点并构造 FSM D 的路径覆盖, 寻找调度后 FSM D 的对应潜在等价路径, 但该方法只能验证 FSM D 结构保持的调度算法, 需要割点之间有双射关系. 对于割点之间无对应关系的调度方法, 文献[42-43]在 FSM D 中插入割点, 采用一种探测比较的方式验证 FSM D 间路径的等价性. 如果路径不等价, 则对路径进行扩展并重新进行比较; 如果路径等价, 则进行下一条路径的验证. 对调度前后差异较大的 FSM D, 该算法需要多次进行路径的扩展和比较, 验证效率较低.

文献[44]将调度前的 IR 使用 CDFG 建模, 调度后的设计使用状态迁移划分(state transition partition, STP)建模, 通过验证 CDFG 和 STP 的功能一致性得到设计的功能一致性, 该方法能够验证各种模式的调度和协议, 并能够处理大规模的设计.

上述方法只能够处理基本块内的代码调度^[45-46], 不能处理基本块之间的代码移动^[47]. 文献[48]改进了文献[41]算法, 提出一种验证推测调度的算法, 该算法使用高阶逻辑表达正确性条件, 并使用验证环境 PVS^[49]验证 FSM D 关键路径等价性, 可以解决跨基本块的代码移动, 但无法验证非均匀的代码移动. 为了验证非均匀的移动代码, 文献[50]提出使用有效变量和可用语句的方法解决推测调度和公共子式消除的高级综合验证算法. 文献[51]能够自动产生数据流中需要被验证属性的 CTL 公式(变量是否会在后续路径中使用), 并使用模型检验工具 NuSMV^[52]进行验证. 文献[50-51]解决了文献[43]中存在的错误否定问题. 文献[53]发现了基于互模拟关系的验证算法和基于路径等价的验证算法的关系, 提出了从基于路径等价的 FSM D 中提取互模拟关系的理论和算法. 文献[54]解决了循环结构的验证问题, 能够避免错误的等价判断. 文献[55]能够根据不等价的设计产生反例, 进一步定位错误.

4.4 基于符号值传播的验证算法

4.4.1 符号值传播的定义

定义 12. 传播向量. 在 $M_0 = \langle Q_0, q_{00}, I_0, O_0, V_0, f_0, h_0 \rangle$ 和 $M_1 = \langle Q_1, q_{10}, I_1, O_1, V_1, f_1, h_1 \rangle$ 中, 路径 β 的传播向量为 $\langle C, \langle e_1, e_2, \dots, e_k \rangle \rangle$, $k = |V_0 \cup V_1|$. 其中, C 表示在路径 β 的开始状态需要满足的条件; $\langle e_1, e_2, \dots, e_k \rangle$ 表示值向量, $e_i (1 \leq i \leq k)$ 表示变量 $v_i \in V_0 \cup V_1$ 在路径 β 的终态得到的符号值.

定义 13. $R_\beta(\bar{v})$ 和 $s_\beta(\bar{v})$ 表示路径 β 的执行条件和数据转换. 路径 β 的开始状态 β^s 的传播向量 $\bar{v}_{\beta^s} = \langle c_1, \bar{e} \rangle$, 其执行条件和路径转换为 $c_1(\bar{v}) \wedge R_\beta(\bar{v}) \{ \bar{e} / \bar{v} \}$ 和 $s_\beta(\bar{v}) \{ \bar{e} / \bar{v} \}$; 其中, $\{ \bar{e} / \bar{v} \}$ 称为替换, $\tau \{ \bar{e} / \bar{v} \}$ 表示 τ 中出现的每一个变量 $v_j \in \bar{v}$ 被对应表达 $e_j \in \bar{e}$ 同时替换.

定义 14. 基于传播向量的路径等价与条件等价. M_0 的路径 $\beta: q_{0,s} \Rightarrow q_{0,f}$, 其传播向量为 $\langle C_\beta, \bar{v}_{0,s} \rangle$, 等价于 M_1 的路径 $\alpha: q_{1,s} \Rightarrow q_{1,f}$, 其传播向量为 $\langle C_\alpha, \bar{v}_{1,s} \rangle$, 记为 $\beta \approx \alpha$; 如果 $C_\beta \wedge R_\beta \{ \bar{v}_{0,s} / \bar{v} \} \equiv$

$C_\alpha \wedge R_\alpha \{\bar{v}_{1,s} / \bar{v}\}, s_\beta \{\bar{v}_{0,s} / \bar{v}\} \equiv s_\alpha \{\bar{v}_{1,s} / \bar{v}\}$. 如果 $q_{0,f} \neq q_{0,0}, q_{1,f} \neq q_{1,0}$; 其中, $q_{0,0}$ 和 $q_{1,0}$ 为 M_0 和 M_1 的 reset 状态, 并且从状态 $q_{0,f}$ 发出的任意路径 β' , 其传播向量 $\langle C_\beta \wedge R_\beta \{\bar{v}_{0,s} / \bar{v}\}, s_\beta \{\bar{v}_{0,s} / \bar{v}\} \rangle$, 存在从状态 $q_{1,f}$ 发出的路径 α' , 其传播向量为 $\langle C_\alpha \wedge R_\alpha \{\bar{v}_{1,s} / \bar{v}\}, s_\beta \{\bar{v}_{1,s} / \bar{v}\} \rangle$, 且 $\beta' \approx \alpha'$ 或 $\beta' \approx_c \alpha'$, 则路径 β 条件等价于路径 α , 记为 $\beta \approx_c \alpha$.

4.4.2 验证算法

根据路径覆盖的定义, 路径不能跨越循环体扩展^[40]. 因此, 上述方法无法解决跨循环结构的代码移动验证问题. 为了验证跨循环体的代码调度, 文献[56]提出了基于符号值传播的高级综合调度验证算法, 将未匹配的变量值传播到所有后续路径, 直到发现所有变量匹配或最终路径不匹配. 该算法首先验证跨循环体移动的代码是否是循环不变量, 如果不是, 则输出不等价; 否则通过将不匹配的活跃变量值传播到所有后继路径来验证跨循环代码移动. 文献[57]改进了文献[56]算法, 能够进一步处理推测调度和动态循环调度^[36]. 文献[58]针对文献[57]中跨循环体调度验证存在错误否定的问题, 提出了增强型符号值传播算法, 其检测循环是否会执行至少一次, 忽略不会执行的路径, 避免产生错误的等价路径, 提高验证准确性. 针对文献[57]的错误否定问题, 文献[59]提出了一种利用测试矢量生成技术和模拟技术产生对应深度路径的验证方法, 通过模拟实际的测试矢量排除无需进行验证的路径, 解决错误否定问题. 文献[60-61]针对增强型符号值传播算法中设计不等价的情况, 分别使用限界模型验证器^[62]和 SMT^[63]求解器根据违反路径产生对应的反例; 利用反例重现错误情况, 指导设计调试. 文献[64]针对符号值传播算法中正规化技术处理能力的局限性(不能处理位向量和用户定义数据类型), 提出了利用 SMT 求解器验证对应路径等价性的方法, 提高了路径的验证能力. 为了解决高层模型中数组的验证问题, 文献[65]在文献[58]的基础上利用带数组数据的有限状态机(finite state machines with datapath having arrays, FSMDA)^[66]对存在数组的设计进行建模, 并利用文献[58]算法验证 FSMDA 的等价性.

5 高级综合后端验证算法

高级综合后端将调度后的设计转换为 RTL 代

码, 后端验证主要验证调度后的 IR 与 RTL 的功能一致性. 高级综合后端验证比前端验证和调度过程简单, 可以利用高级综合工具绑定阶段提供的映射信息得到对应的割点并验证割点的等价性.

文献[67]利用行为级到 RTL 的变量和状态的绑定信息产生必要的重写规则, 基于重写规则进行验证, 其方法已被集成至高级综合工具. 文献[68]利用验证高级综合后端资源绑定阶段的映射信息验证调度后设计的 FSMD 和 RTL FSMD 之间的功能等价性. 由于绑定阶段不涉及代码移动, 该方法只需验证对应状态的迁移等价. 文献[69]使用带时钟的控制数据流图(clocked control/data flow graph, CCDFG)表示调度后设计与 RTL 设计, 根据高级综合工具提供的映射关系, 使用符号模拟和定理证明器验证 CCDFG 的等价性. 文献[70]优化了文献[69]方法, 使用割点技术解决循环的验证, 避免循环固定点的计算, 使用模块化技术减小验证规模, 最后利用符号模拟和 SMT 求解器验证对应等价点的等价性. 上述方法要求在被测设计之间存在等价操作或模块, 但是由于代码优化和全局变量的存在, 使等价的操作和模块在实际的综合过程中极少存在. 因此, 文献[71]进一步改进了上述方法, 使用扩展标记解决全局变量问题, 使用宽松等价算法解决局部不等价问题, 该算法能够处理大规模设计.

6 高级综合验证算法的分析与比较

本文对高级综合前端验证、调度验证和后端验证 3 个层次的特点和难点进行了分析和总结, 如表 1 所示.

同时, 将高级综合验证方法归结为基于文本差异的符号模拟算法、基于互模拟的验证算法、基于路径等价的验证算法和基于符号值传播的验证算法. 表 2 对 4 类算法的特点以及各自的优势与局限性进行了分析.

7 主要挑战和研究趋势

通过分析近年来高级综合验证算法的发展, 本文认为高级综合验证仍然面临以下挑战.

(1) 高级综合前端进行的无用代码删除、公共子式消除和循环展开等优化操作, 导致部分代码映射关系缺失, 循环展开导致验证规模变大, 目前的高级综合前端验证算法还未能较好地解决.

表 1 高级综合验证层次分析

高级综合验证层次	验证特点	验证难点
高级综合前端验证	高级综合前端将进行代码转换和代码优化, 包括死代码消除、常数传播、公共子式消除和循环展开等。前端验证主要验证优化前系统级设计与优化后 IR 的功能一致性	优化导致部分映射关系缺失, 增加验证难度; 循环结构验证困难; 循环展开导致验证规模变大
高级综合调度验证	代码移动, 将操作分配给控制步骤, 同步硬件设计的时钟周期。调度验证主要验证高级综合调度前后 IR 的功能一致性	调度导致设计控制结构改变, 其中基于路径的调度、非均匀调度和跨循环体代码调度验证难度最大
高级综合后端验证	将操作映射到分配的硬件单元, 根据 IR 和功能单元映射关系, 产生 RTL 代码。后端验证主要验证调度后 IR 与 RTL 代码的功能一致性	验证方法直接, 但是需要从高级综合工具获取设计之间的映射关系

表 2 4 种高级综合验证算法分析与比较

高级综合验证算法	特点	优点	缺点
基于文本差异的符号模拟算法	验证 2 个代码的差异, 扩展并验证不等价的代码, 直到所有差异都等价或最终设计不等价	方法简单, 能够快速地进行验证	对于文本差异较大的设计, 需要多次扩展和验证, 验证复杂度较高
基于互模拟的验证算法	验证对应设计存在互模拟关系, 互模拟关系表示对应结构的等价性, 验证过程包含推测和验证 2 个步骤	能够验证循环移动 ^[72] 和结构保持的调度	需要设计存在映射关系, 无法对非结构保持调度进行验证, 验证不一定终止
基于路径的验证算法	验证状态机路径覆盖中的每一条路径在另一个状态机中存在对应的等价路径, 反之亦然	能够验证非结构化保持调度, 验证过程一定终止	不能验证循环移动和跨循环体的调度, 路径扩展导致验证效率较低
基于符号值传播的验证算法	将未匹配的变量值传播到所有后续路径, 直到发现变量匹配或最终路径不匹配	能够验证均匀代码移动, 非均匀代码移动, 跨循环体代码移动	存在错误否定问题, 无法处理数组结构

(2) 高级综合编译调度过程会根据数据依赖关系和时序要求进行代码调度, 其中跨越循环体的代码调度会改变设计控制流, 导致映射关系进一步消失, 成为等价性检验最难的部分, 还需要进一步研究。

(3) 形式化方法存在状态空间爆炸等问题, 同时求解器能力还需进一步提高, 目前的形式化方法还无法验证工业界大规模设计。

(4) 高层系统级模型中存在的不定次循环、数组、指针和复杂的结构, 学术界和业界都还没有找到很好的验证方法。

在未来高级综合验证研究中, 本文认为存在以下研究趋势。

(1) 随着高级综合工具的发展, 其处理能力和范围将逐渐提高, 高级综合工具的运用将会更加广泛。因此, 高级综合验证将成为电子设计自动化和计算机辅助设计领域的研究热点。

(2) 高级综合调度是高级综合过程最复杂最易错的阶段, 其验证过程最为困难, 研究高级综合调度前后设计的功能一致性将成为高级综合验证研究的热点。

(3) 基于互模拟的验证方法能够验证循环移动, 但算法不一定终止; 基于路径的验证方法能够验证非结构保持的调度, 但不能处理跨循环体移

动; 基于符号值传播的方法能够验证跨循环体的调度, 但存在错误否定问题。研究结合 3 种方法优点的统一算法框架可以成为研究方向。

(4) 对调度前后的设计, 设计之间映射关系缺失, 包括机器学习、深度学习和模拟等启发式方法能够被用于产生设计之间的映射关系, 启发式方法在高级综合验证中的应用将成为研究热点。

从目前最新的文献[73]可以看出, 高级综合验证, 特别是高级综合调度验证仍然是目前电子设计自动化方向的研究热点之一。解决不同层次的模型抽象、化简、时序匹配、割点识别、映射关系提取、不定次循环、数组、指针和复杂的堆存储结构的验证将是未来亟待解决的问题。

8 结 语

高层综合验证在 SoC 设计和验证领域具有非常重要的理论和实践意义。本文首先分析了高级综合的含义、高级综合验证的层次划分和高级综合验证的难点; 然后根据高级综合验证层次概述分析了高级综合前端验证算法, 高级综合调度验证算法和高级综合后端验证算法, 分析了各类算法的特点、优势与局限性; 总结了目前高级综合验证还将面临的挑战, 包括高级综合前后设计映射关

系缺失、设计状态空间爆炸以及高层模型中不定次循环和复杂结构的验证等难点。同时分析了今后高级综合验证的研究趋势,未来将会在高级综合调度验证,高级综合工具的研究以及包括机器学习等启发式方法的应用等方面进行展开。最后,本文对 SoC 高级综合验证的总结与分析能够为 SoC 系统级功能验证方法和验证自动化技术提供研究基础和可行的解决方案。

参考文献(References):

- [1] Gupta S, Gupta R K, Dutt N D, *et al.* Spark: a parallelizing approach to the high-level synthesis of digital circuits[M]. Heidelberg: Springer, 2004
- [2] de Micheli G. Synthesis and optimization of digital circuits[M]. New York: ACM Press, 1994
- [3] Zhang Z R, Fan Y P, Jiang W, *et al.* AutoPilot: a platform-based ESL synthesis system[M]. Heidelberg: Springer, 2008: 99-112
- [4] Koch D, Hannig F, Ziener D. FPGAs for software programmers[M]. Heidelberg: Springer, 2016
- [5] Aho A V, Sethi R, Ullman J D. Compilers, principles, techniques and tools[M]. New Jersey: Addison Wesley Publishing Company, 1986
- [6] Yang Z K, Hao K C, Cong K, *et al.* Equivalence checking for compiler transformations in behavioral synthesis[C] // Proceedings of the 31st International Conference on Computer Design. Los Alamitos: IEEE Computer Society Press, 2013: 491-494
- [7] Yang Z K, Hao K C, Cong K, *et al.* Scalable certification framework for behavioral synthesis front-end[C] // Proceedings of the 51st ACM/EDAC/IEEE Design Automation Conference. Los Alamitos: IEEE Computer Society Press, 2014: 1-6
- [8] Shankar S, Fujita M. Rule-based approaches for equivalence checking of SpecC programs[C] // Proceedings of the 6th International Conference on Formal Methods and Models for Co-Design. Los Alamitos: IEEE Computer Society Press, 2008: 39-48
- [9] Yoshida H, Fujita M. Improving the accuracy of rule-based equivalence checking of system-level design descriptions by identifying potential internal equivalences[C] // Proceedings of the 10th International Symposium on Quality of Electronic Design. Los Alamitos: IEEE Computer Society Press, 2009: 366-370.
- [10] Hu J, Wang G W, Chen G L, *et al.* Equivalence checking between system-level descriptions by identifying potential cut-points[C] // Proceedings of International Conferences on Communications, Signal Processing, and Systems. Heidelberg: Springer, 2019: 1328-1335
- [11] Matsumoto T, Saito H, Fujita M. An equivalence checking method for C descriptions based on symbolic simulation with textual differences [J]. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2005, 88 (12): 3315-3323
- [12] Matsumoto T, Saito H, Fujita M. Equivalence checking of C programs by locally performing symbolic simulation on dependence graphs[C] // Proceedings of the 7th International Symposium on Quality Electronic Design. Los Alamitos: IEEE Computer Society Press, 2006: 370-375
- [13] Horwitz S, Reps T, Binkley D. Interprocedural slicing using dependence graphs[J]. ACM Transactions on Programming Languages and Systems, 1990, 12 (1): 26-60
- [14] Matsumoto T, Nishihara T, Kojima Y, *et al.* Equivalence checking of high-level designs based on symbolic simulation[C] // Proceedings of International Conference on Communications, Circuits and Systems. Los Alamitos: IEEE Computer Society Press, 2009: 1129-1133
- [15] Nishihara T, Ando D, Matsumoto T, *et al.* ExSDG: unified dependence graph representation of hardware design from system level down to RTL for formal analysis and verification[C] // Proceedings of the International Workshop of Logic and Synthesis. Tokyo: CiNii, 2007: 83-90
- [16] Gajski D D, Dutt N D, Wu A C H, *et al.* High-level synthesis: introduction to chip and system design[M]. Norwell: Kluwer Academic Publishers, 1992
- [17] Milner R. A calculus of communicating systems[M]. Heidelberg: Springer, 1980
- [18] Fernandez J C, Mounier L. "On the fly" verification of behavioural equivalences and preorders[C] // Proceedings of International Conference on Computer Aided Verification. Heidelberg: Springer, 1991: 181-191
- [19] de Nicola R, Fantechi A, Gnesi S, *et al.* An action based framework for verifying logical and behavioural properties of concurrent systems[C] // Proceedings of International Conference on Computer Aided Verification. Heidelberg: Springer, 1991: 37-47
- [20] Tripakis S, Yovine S. Analysis of timed systems using time-abstracting bisimulations[J]. Formal Methods in System Design, 2001, 18(1): 25-68
- [21] Dovier A, Piazza C, Policriti A. An efficient algorithm for computing bisimulation equivalence[J]. Theoretical Computer Science, 2004, 311(1-3): 221-256
- [22] Baier C. Polynomial time algorithms for testing probabilistic bisimulation and simulation[C] // Proceedings of International Conference on Computer Aided Verification. Heidelberg: Springer, 1996: 50-61
- [23] Baier C, Engelen B, Majster-Cederbaum M. Deciding bisimilarity and similarity for probabilistic processes[J]. Journal of Computer and System Sciences, 2000, 60(1): 187-231
- [24] Eveking H, Hinrichsen H, Ritter G. Automatic verification of scheduling results in high-level synthesis[C] // Proceedings of International Conference on Design, Automation and Test in Europe. Los Alamitos: IEEE Computer Society Press, 1999: 260-265
- [25] Necula G C. Translation validation for an optimizing compiler[C] // Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation. New York: ACM Press, 2000: 83-94
- [26] Kundu S, Lerner S, Gupta R. Validating high-level synthesis[C] // Proceedings of the 20th International Conference on Computer Aided Verification. New York: ACM Press, 2008: 459-472

- [27] Bandyopadhyay S, Sarkar D, Banerjee K, *et al.* A path-based equivalence checking method for petri net based models of programs[C] //Proceedings of the 10th International Joint Conference on Software Technologies. Los Alamitos: IEEE Computer Society, 2015: 319-329
- [28] Bandyopadhyay S, Sarkar D, Mandal C. Poster: an efficient equivalence checking method for petri net based models of programs[C] //Proceedings of the 37th IEEE/ACM International Conference on Software Engineering. Los Alamitos: IEEE Computer Society, 2015: 827-828
- [29] Cortes L A, Eles P, Peng Z B. Verification of embedded systems using a petri net-based representation[C] //Proceedings of the 13th International Symposium on System Synthesis. Los Alamitos: IEEE Computer Society, 2000: 149-155
- [30] Bandyopadhyay S, Sarkar S, Sarkar D, *et al.* SamaTulyata: an efficient path based equivalence checking tool[C] //Proceedings of International Symposium on Automated Technology for Verification and Analysis. Heidelberg: Springer, 2017: 109-116
- [31] Bandyopadhyay S, Sarkar D, Mandal C. SamaTulyataOne: a path based equivalence checker[C] //Proceedings of the 12th Innovations on Software Engineering Conference. New York: ACM Press, 2019: Article No.21
- [32] Kundu S, Lerner S, Gupta R K. Translation validation of high-level synthesis[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2010, 29(4): 566-579
- [33] Gupta S, Dutt N, Gupta R, *et al.* SPARK: a high-level synthesis framework for applying parallelizing compiler transformations[C] //Proceedings of the 16th International Conference on VLSI Design. Los Alamitos: IEEE Computer Society Press, 2003: 461-466
- [34] Li T, Guo Y, Liu W W, *et al.* Efficient translation validation of high-level synthesis[C] //Proceedings of International Symposium on Quality Electronic Design. Los Alamitos: IEEE Computer Society Press, 2013: 516-523
- [35] Camposano R. Path-based scheduling for synthesis[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1991, 10(1): 85-93
- [36] Rahmouni M, Jerraya A A. Formulation and evaluation of scheduling techniques for control flow graphs[C] //Proceedings of European Design Automation Conference. Los Alamitos: IEEE Computer Society Press, 1995: 386-391
- [37] Li T, Guo Y, Liu W W, *et al.* Translation validation of scheduling in high level synthesis[C] //Proceedings of the 23rd ACM International Conference on Great lakes Symposium on VLSI. New York: ACM Press, 2013: 101-106
- [38] Hoare C A R. An axiomatic basis for computer programming[J]. Communications of the ACM, 1969, 12(10): 576-580
- [39] Li T, Hu J, Guo Y, *et al.* Equivalence checking of scheduling in high-level synthesis[C] //Proceedings of the 16th International Symposium on Quality Electronic Design. Los Alamitos: IEEE Computer Society Press, 2015: 257-262
- [40] Floyd R W. Assigning meaning to programs[M] //Colburn T R, Fetzer J H, Rankin T L. Program Verification. Studies in Cognitive Systems. Heidelberg: Springer, 1993: 65-81
- [41] Kim Y, Kopuri S, Mansouri N. Automated formal verification of scheduling process using finite state machines with datapath[C] //Proceedings of the 5th International Symposium on Quality Electronic Design. Los Alamitos: IEEE Computer Society Press, 2004: 110-115
- [42] Karfa C, Mandal C, Sarkar D, *et al.* A formal verification method of scheduling in high-level synthesis[C] //Proceedings of the 7th International Symposium on Quality Electronic Design. Los Alamitos: IEEE Computer Society Press, 2006: 71-78
- [43] Karfa C, Sarkar D, Mandal C, *et al.* An equivalence-checking method for scheduling verification in high-level synthesis[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2008, 27(3): 556-569
- [44] Yang Z K, Hao K C, Cong K, *et al.* Validating scheduling transformation for behavioral synthesis[C] //Proceedings of International Conference on Design, Automation, and Test in Europe. Los Alamitos: IEEE Computer Society Press, 2016: 1652-1657
- [45] Jain R, Mujumdar A, Sharma A, *et al.* Empirical evaluation of some high-level synthesis scheduling heuristics[C] //Proceedings of the 28th ACM/IEEE Design Automation Conference. New York: ACM Press, 1991: 686-689
- [46] Lee J H, Hsu Y C, Lin Y L. A new integer linear programming formulation for the scheduling problem in data path synthesis[C] //Proceedings of the IEEE International Conference on Computer Aided Design. Los Alamitos: IEEE Computer Society Press, 1989: 20-23
- [47] Gupta S, Savoiu N, Dutt D, *et al.* Using global code motions to improve the quality of results for high-level synthesis[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2004, 23(2): 302-312
- [48] Kim Y, Mansouri N. Automated formal verification of scheduling with speculative code motions[C] //Proceedings of the 18th ACM Great Lakes Symposium on VLSI. New York: ACM Press, 2008: 95-100
- [49] Owre S, Shankar N, Rushby J, *et al.* PVS language reference[M]. New York: SRI International, 1999
- [50] Lee C H, Shih C H, Huang J D, *et al.* Equivalence checking of scheduling with speculative code transformations in high-level synthesis[C] //Proceedings of the 16th Asia and South Pacific Design Automation Conference. New York: ACM Press, 2011: 497-502
- [51] Karfa C, Mandal C, Sarkar D. Formal verification of code motion techniques using data-flow-driven equivalence checking[J]. ACM Transactions on Design Automation of Electronic Systems, 2012, 17(3): Article No.30
- [52] Cimatti A, Clarke E, Giunchiglia F, *et al.* NuSMV: a new symbolic model checker[J]. International Journal on Software Tools for Technology Transfer, 2000, 2(4): 410-425
- [53] Banerjee K, Sarkar D, Mandal C. Deriving bisimulation relations from path extension based equivalence checkers[J]. IEEE Transactions on Software Engineering, 2017, 43(10): 946-953
- [54] Chouksey R, Karfa C, Bhaduri P. Translation validation of code motion transformations involving loops[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 38(7): 1378-1382
- [55] Chouksey R, Karfa C, Banerjee K, *et al.* A counter-example generation procedure for path based equivalence checkers[J]. IET Software, 2019, 13(4): 280-285
- [56] Banerjee K, Karfa C, Sarkar D, *et al.* A value propaga-

- tion-based equivalence checking method for verification of code motion techniques[C] //Proceedings of the International Symposium on Electronic System Design. New York: ACM Press, 2012: 67-71
- [57] Banerjee K, Karfa C, Sarkar D, *et al.* Verification of code motion techniques using value propagation[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2014, 33(8): 1180-1193
- [58] Chouksey R, Karfa C, Bhaduri P. Translation validation of loop invariant code optimizations involving false computations[C] // Proceedings of International Conference on VLSI Design and Test. Heidelberg: Springer, 2017: 767-778
- [59] Hu J, Wang G W, Chen G L, *et al.* Equivalence checking of scheduling in high-level synthesis using deep state sequences[J]. IEEE Access, 2019, 7: 183435-183443
- [60] Chouksey R, Karfa C, Bhaduri P. Formal verification of optimizing transformations during high-level synthesis[C] // Proceedings of the 12th Innovations on Software Engineering Conference. New York: ACM Press, 2019: Article No.27
- [61] Chouksey R, Karfa C, Banerjee K, *et al.* Counter-example generation procedure for path-based equivalence checkers[J]. IET Software, 2019, 13(4): 280-285
- [62] Clarke E, Kroening D, Lerda F. A tool for checking ANSI-C programs[C] //Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Heidelberg: Springer, 2004: 168-176
- [63] de Moura L, Bjørner N. Z3: an efficient SMT solver[C] // Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Heidelberg: Springer, 2008: 337-340
- [64] Banerjee K, Mandal C, Sarkar D. Extending the scope of translation validation by augmenting path-based equivalence checkers with SMT solvers[C] //Proceedings of the 18th International Symposium on VLSI Design and Test. Los Alamitos: IEEE Computer Society Press, 2014: 1-6
- [65] Banerjee K, Mandal C, Sarkar D. A translation validation framework for symbolic value propagation-based equivalence checking of FSMdAs[C] //Proceedings of the 15th IEEE International Working Conference on Source Code Analysis and Manipulation. Los Alamitos: IEEE Computer Society Press, 2015: 247-252
- [66] Banerjee K, Sarkar D, Mandal C. Extending the FSMd framework for validating code motions of array-handling programs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2014, 33(12): 2015-2019
- [67] Mansouri N, Vemuri R. A methodology for automated verification of synthesized RTL designs and its integration with a high-level synthesis tool[C] //Proceedings of International Conference on Formal Methods in Computer-Aided Design. Heidelberg: Springer, 1998: 204-211
- [68] Borriore D, Dushina J, Pierre L. A compositional model for the functional verification of high-level synthesis results[J]. IEEE Transactions on Very Large-Scale Integration Systems, 2000, 8(5): 526-530
- [69] Ray S, Hao K C, Chen Y, *et al.* Formal verification for high-assurance behavioral synthesis[C] //Proceedings of International Symposium on Automated Technology for Verification and Analysis. Heidelberg: Springer, 2009: 337-351
- [70] Hao K C, Xie F, Ray S, *et al.* Optimizing equivalence checking for behavioral synthesis[C] //Proceedings of the Design, Automation Test in Europe Conference Exhibition. Los Alamitos: IEEE Computer Society Press, 2010: 1500-1505
- [71] Yang Z K, Hao K C, Ray S, *et al.* Handling design and implementation optimizations in equivalence checking for behavioral synthesis[C] //Proceedings of the 50th ACM/EDAC/IEEE Design Automation Conference. New York: ACM Press, 2013: 1-6
- [72] Darte A, Huard G. Loop shifting for loop compaction[J]. International Journal of Parallel Programming, 2000, 28(5): 499-534
- [73] Chouksey R, Karfa C. Verification of scheduling of conditional behaviors in high-level synthesis[J]. IEEE Transactions on Very Large Scale Integration Systems, 2020, 28(7): 1638-1651