

基于模型检测的硬件木马检测技术研究

张启智, 赵毅强, 高雅, 马浩诚

(天津大学微电子学院, 天津 300110)

摘要: 硬件木马对原始电路的恶意篡改, 已成为集成电路面临的核心安全威胁。为了保障集成电路的安全可信, 研究人员提出了诸多硬件木马检测方法。其中, 模型检测作为一种形式化验证方法, 在设计阶段可有效检测出硬件木马。首先, 阐述了模型检测的工作原理和应用流程; 其次, 介绍了基于模型检测的硬件木马检测技术的研究进展; 最后, 指出了当前该技术所面临的瓶颈, 并讨论了潜在的研究方向。

关键词: 硬件木马; 模型检测; 模型构建; 属性声明

中图分类号: TN407

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2021029

Survey on model checking based hardware Trojan detection technology

ZHANG Qizhi, ZHAO Yiqiang, GAO Ya, MA Haocheng

School of MicroElectronics, Tianjin University, Tianjin 300110, China

Abstract: Hardware Trojan is malicious tampering to the original circuit, which has become the most important security threat of integrated circuit. In order to ensure the safety and reliability of ICs, many hardware Trojan detection methods are proposed. As one of the formal verification methods, model checking can effectively detect the hardware Trojan in the design phase. Firstly, the working principle and process of model checking were described. Secondly, the research progress of hardware Trojan detection technology based on model checking was introduced. Finally, the bottlenecks faced by the current technology were pointed out and the potential research direction was discussed.

Keywords: hardware Trojan, model checking, model establishing, property declaration

收稿日期: 2020-11-16; 修回日期: 2021-01-21

通信作者: 赵毅强, yq_zhao@tju.edu.cn

基金项目: 国家自然科学基金 (61832018)

Foundation Item: The National Natural Science Foundation of China(61832018)

论文引用格式: 张启智, 赵毅强, 高雅, 等. 基于模型检测的硬件木马检测技术研究[J]. 网络与信息安全学报, 2021, 7(2): 57-63.

ZHANG Q Z, ZHAO Y Q, GAO Y, et al. Survey on model checking based hardware Trojan detection technology[J]. Chinese Journal of Network and Information Security, 2021, 7(2): 57-63.

1 引言

近年来,人工智能、5G 等新兴技术飞速发展,“万物互联”的时代逐渐成为现实。集成电路作为现代信息技术的硬件基础,其功能多样性与应用广泛性迅速提升,已经成为支撑经济社会发展的战略性、基础性和先导性产业。由于集成电路产品的先进性和复杂性,同时为了更合理地配置资源,简化电路设计流程,缩短上市时间,电路设计者通常会大量复用将第三方 IP 核 (intellectual property core)。然而,由于第三方机构的不可控性,第三方 IP 核可能存在植入硬件木马的风险,严重威胁集成电路的安全可信。

硬件木马是指对原始集成电路的恶意篡改,如冗余模块或恶意逻辑^[1]。一般情况下,硬件木马会保持静默状态,只有在满足特定情况时才会被激活,执行攻击者预先设定的恶意行为,这对集成电路的安全造成了严重的威胁。被广泛应用于生产实践中的硬件木马检测技术大多只适用于硅后阶段,如逻辑测试、侧信道分析等。基于逻辑测试的硬件木马检测技术通过在芯片的输入端口施加特定的测试激励,检测输出端口的信号是否与预期的输出一致,进而判断电路中是否存在硬件木马。但是,一些触发结构较为复杂的时序性硬件木马难以被测试激励触发,有些硬件木马即使被触发也不会改变电路的输出逻辑,使逻辑检测技术在实际硬件木马检测中的使用大大受限^[2]。基于侧信道信息的硬件木马检测技术关注硬件木马的植入给电路运行参数带来的影响^[3]。早在 2007 年, Agrawal 等^[1]建立了芯片的功耗指纹模型,提出了利用电路的功耗信息来实现硬件木马的检测。之后,电磁、时延等多种侧信道信息被应用到木马检测中。但基于侧信道信息的硬件木马检测技术容易受到检测环境与工艺偏差的影响,导致难以完全识别出硬件木马。综上所述,硅后的硬件木马检测技术受到诸多限制的影响,木马检测的准确率不高;同时,由于芯片已经制造完成,即使检测出硬件木马,也难以对其进行针对性的防御与修正。

模型检测^[4]是一种可以同时面向软件和硬件的自动化验证技术,其原理是通过对系统状态空

间进行穷尽搜索来判断系统是否满足某些性质。模型检测作为形式化方法的一种,可以在硅前阶段对芯片设计是否满足规定的安全属性进行验证,在芯片投入生产之前对其中是否被植入了硬件木马进行检测。同时,模型检测可以在系统不满足性质规约时给出反例,有助于实现硬件木马的定位并进行针对性的修改。因此,近年来基于模型检测的硬件木马检测技术逐渐成为相关领域研究的重点。

本文对基于模型检测的硬件木马检测技术的应用流程进行了细致的归纳与梳理,对较为常用的模型检测工具进行了调研,对现有的电路形式化模型构建方法与面向木马检测的安全属性声明策略进行了详细的总结。

2 模型检测技术概述

2.1 基本原理与应用流程

模型检测是一种将给定的模型中所有可达状态与行为进行遍历搜索,进而验证其是否满足某些性质的方法。模型检测具有两个突出的优点:一是模型检测的自动化程度非常高,整个验证的过程全部为自动进行,它并不要求验证人员对数学知识有非常专业的掌握;二是如果模型检测得到的结果反馈表示所验证的模型或系统不能完全满足属性的约束或者不符合某些性质,那么模型检测器将生成一个反例,告诉验证人员具体哪些逻辑或哪些行为不满足要求的属性。这种机制对于验证人员了解系统安全性的漏洞,以及集成电路中是否存在硬件木马有相当大的帮助;同时,给验证人员排除恶意电路指明了方向。

利用模型检测技术来进行电路中硬件木马的检测一般分为以下 3 步:电路模型的构建、安全属性的声明以及验证^[5]。模型检测的应用流程如图 1 所示。首先是电路模型的构建,也就是将电路设计转换为模型检测器可以读取的形式。在多数情况下,模型构建是电路描述语言向形式化语言的转译。然而,在有些情况下,受限于验证时间和计算机内存,需要对模型进行化简或省略与验证目标无关的电路。其次是安全属性的声明,验证者需要首先声明希望电路满足的性质,这种性质一般以某种逻辑的形式进行表达。最后是验

证, 验证的过程一般是由解析器自动化实现的, 但在实际的验证过程中需要一定程度的人为参与, 如对未通过验证的结果进行分析, 进而提出对应的安全性设计需求。

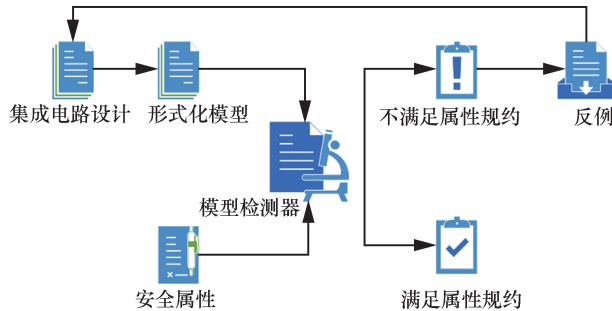


图1 模型检测的应用流程
Figure 1 Application process of model checking

2.2 常用检测工具

2.2.1 NuSMV

NuSMV (new symbolic model verifier)^[6]是由 Carnegie Mellon 大学基于 SMV (symbolic model verifier)^[7]开发的, 以有序二叉判定图为基础对有限状态系统是否满足基于 CTL 逻辑描述的属性约束进行验证的模型检测器。经过长时间的使用和更新, NuSMV 已经是一个较为成熟的可以同时适用于软硬件验证的模型检测器^[8-10]。

NuSMV 是一个开放式 (这是其最重要的特征) 的系统, 它的内部由解析器、实例化模块、编码模块、模型检测模块等组成。每个模块实现一组功能, 并通过精准定义的接口与其他模块进行交互, 这种设计方式方便用户根据自己的需要进行调整和修改。

NuSMV 兼容的程序代码在结构组成上与硬件描述语言类似, 由模块组成。每个程序必须有一个主模块, 主模块中定义系统的主要功能以及各个模块之间的关系, 同时可以对其余模块进行多次实例化。这种编程方式与硬件设计的代码描述方式高度契合, 因此, 在面向硬件安全的模型检测研究中, NuSMV 的使用非常广泛^[11-13]。NuSMV 代码结构如图 2 所示。

2.2.2 UPPAAL

UPPAAL 是由瑞典的 Uppsala 大学和丹麦的 Aalborg 大学联合开发用于实时系统验证的

工具^[14], 它在通信协议和多媒体领域得到了成功的应用。UPPAAL 模型检测器包括 3 部分: 编辑器、模拟器和验证器。编辑器是建立时间自动机模型的部分; 模拟器是对编辑器中建立的模型进行仿真模拟的部分; 验证器是编写安全属性并对其进行验证的模块, 在这一模块中, 用户可以利用查询语言对想要验证的安全属性进行编写。UPPAAL 的基本结构如图 3 所示。

```

MODULE main
VAR
    sender:process Sender (s2r_in,r2s_out) ;
    s2r_in:S2RData;
    s2r_out:S2RData;
    receiver:process Receiver (r2s_in,s2r_out) ;
    res_in:R2SData;
    r2s_out: R2S Data;
    r2s:process R2S (r2s_in,r2s_out) ;
    s2r:process S2R (s2r_in,s2r_out) ;
    SPEC AG AF (sender.state=get) —— & (AF receiver.state=deliver)
  
```

图2 NuSMV 代码结构
Figure 2 The code structure of NuSMV

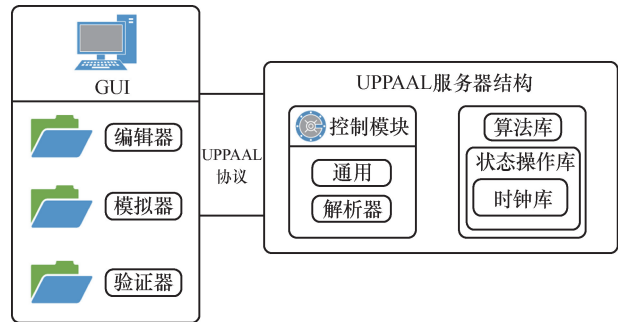


图3 UPPAAL 的基本结构
Figure 3 The fundamental structure of UPPAAL

在 UPPAAL 中, 一个系统被建模为由多个时间自动机并行组成的网络。系统的每个状态在 UPPAAL 中被定义为位置、时钟变量以及有界变量的集合。每个时间自动机通过激活边界或者触发与其他自动机的同步信号来进入下一个状态。

虽然相比于一般的模型检测器, UPPAAL 的性能与效率均较为优异^[15], 但受限于电路模型建立的复杂程度以及工具本身难以满足电路中常见数据类型的声明要求, UPPAAL 在硬件安全验证领域的应用并不如 NuSMV 广泛^[16-18]。

3 不同种类硬件木马的模型检测技术

在利用模型检测技术进行硬件木马检测的过程中, 需要对不同种类的硬件木马根据其功能或结构特点有针对性地展开验证。硬件木马根据其功能可分为 4 类: 降低性能型、改变功能型、拒绝服务型、信息泄露型^[19]。下面将详细介绍针对各种类型硬件木马的模型检测方法。

3.1 降低性能型硬件木马的模型检测技术

降低性能型硬件木马是指通过改变电路的制造参数(如使电路的主频下降、低温下系统瘫痪、电磁辐射变强和电量消耗加速等)来降低电路性能的木马。对于这种类型的硬件木马, 验证者在进行模型检测时, 通常考虑对木马可能攻击的某些电路参数设定阈值, 基于不能超过预设阈值这一特性声明电路应当满足的安全属性。

Hasan 等^[20]对包含两个主机和一个从机的高级处理器总线结构(AMBA)进行建模, 其中从机中被植入创造死锁场景, 使从机优先于主机占用总线以降低处理器性能的硬件木马。对于此类木马, 基于计算树逻辑(CTL, computation tree logic)描述了如下安全属性: 对于全部状态路径, 主机的总线使用优先级要高于从机。在得到反例后, 基于反例描述的死锁情况运用博弈论的理论对木马植入和检测的复杂程度进行分析, 并给出了针对性的木马检测结构设计意见。

文献[21-22]基于电路运行过程中产生的侧信道信息进行了电路模型的构建, 明确了电路中存在的各种逻辑门在真值表的各个状态下产生的功耗信息以及逻辑门自身的时延信息, 通过统计电路中所有可能的状态得到电路功耗与时延信息的上下限, 基于侧信道信息的阈值构建电路应当满足的安全属性。研究者在基准电路中插入冗余的逻辑门作为硬件木马, 按照上述方式声明安全属性, 在模型检测器中进行验证, 成功检测出了植入的硬件木马, 如式(1)、式(2)所示。

$$G(\text{pwr_max} \geq \text{gate1.pwr} + \text{gate2.pwr} + \dots + \text{gaten.pwr} \geq \text{pwr_min}) \quad (1)$$

$$G(\text{del_}(\text{path_}i)\text{max} \geq \text{gate1}(i).\text{del} + \text{gate2}(i).\text{del} + \dots + \text{gatek}(i).\text{del} \geq \text{del_}(\text{path_}i)\text{min}) \quad (2)$$

上述安全属性用自然语言描述为: 电路中所有逻辑门的功耗之和应大于或等于电路总功耗的最小值且小于或等于电路总功耗的最大值; 电路中某条通路上所有逻辑门的时延之和应大于或等于通路时延的最小值, 且小于或等于通路时延的最大值。

3.2 改变功能型硬件木马的模型检测技术

改变功能型硬件木马是通过增加、删减或者修改电路逻辑, 在特定时刻篡改原始电路的功能, 使电路产生错误输出甚至使电路瘫痪的木马。针对此类硬件木马的模型检测技术往往需要将安全属性定义为待验证电路功能与正确功能完全一致或者对于特定的攻击场景将安全属性定义为只能执行特定指令。

Rathmair 等^[23]采用模型检测的技术对植入在内存控制器中的硬件木马进行检测。该研究针对的硬件木马在地址信号高于某个预设阈值时, 交换地址信息的某两位数值, 使外部无法访问到正确的内存中的信息。针对这种改变功能型的木马, 研究者首先将电路转译为形式化模型, 进而提出了针对性的安全属性:

$$!EF(\text{state} = \text{writeMem1} \ \& \ \text{Address1} \neq \text{Address}) \quad (4)$$

用自然语言描述为存储器接口的地址不得与输入的 16 位地址不同。如果出现输入存储器接口的地址信息与预设地址信息不同的情况, 模型检测器就会报告验证失败并生成反例。

Zhang 等^[24]针对难以通过电路运行参数差别进行微小木马展开模型检测。该微型木马可以篡改电路的真值表, 影响电路的正常输出。研究者将原始电路和可疑电路组成可疑电路对, 并统一对原始电路和可疑电路进行模型构建, 采用 CTL 逻辑对可疑电路对进行安全属性的约束:

$$G((u_1 = v_1) \ \& \ (u_2 = v_2) \ \& \ \dots \ (u_i = v_i)) \quad (5)$$

用自然语言表述为对于任意相同的输入变量, 可疑电路对的输出总是相同的。这是一种白名单的验证策略, 也就是说只允许出现与母本电路真值表相同的情况。如果属性验证没有通过, 即可得知可疑电路与母本电路并不完全一致, 进而检测出硬件木马的存在。

Zhao 等^[25]同样采用白名单的验证策略对改变功能型硬件木马进行检测, 基于电路的逻辑运算结果建立安全属性; Krieg 等^[26]通过对恶意行为进行描述的方式来建立安全属性, 然而这种属性建立方式存在一个问题, 即如果属性通过验证则可以说明电路中存在恶意逻辑, 但如果属性未能通过验证却不能保证电路中不存在其他恶意逻辑。

3.3 拒绝服务型硬件木马的模型检测技术

拒绝服务型硬件木马在特定条件下可以使系统暂时性拒绝响应、状态跳转紊乱甚至系统永久性瘫痪等, 常常被植入在总线、微处理器等硬件场景中。对于此类硬件木马, 可以采用以下两种验证策略: ①用形式化语义描述电路的正常功能, 查看模型检测器是否报告反例; ②对于可能的木马攻击方式, 针对性地设计安全属性进行验证, 这种验证策略需要对木马的攻击方式以及可能的攻击位置有充分的认识。

高洪博^[27]在 2013 年提出了一种基于模型检测的指令诱发型硬件木马的检测技术, 在以微处理器为核心的交通灯控制系统中植入了设计的木马^[28], 其功能由特定的指令序列触发, 触发后导致系统工作紊乱, 无法正常工作。研究者基于 CTL 对系统应当产生的正常功能进行描述, 验证发现存在不符合属性约束的反例, 同时对固件代码进行逆向, 可以得到产生反例的代码序列, 检测并定位到硬件木马, 交通灯控制系统正常显示序列的 CTL 语言描述如式(6)所示。

$$\begin{aligned} &AG((p_1 = \text{TRUE}) \rightarrow A[p_1 \cup p_2]) \& \\ &AG((p_2 = \text{TRUE}) \rightarrow A[p_2 \cup p_3]) \& \\ &AG((p_3 = \text{TRUE}) \rightarrow A[p_3 \cup p_4]) \& \\ &AG((p_4 = \text{TRUE}) \rightarrow A[p_4 \cup p_1]) \end{aligned} \quad (6)$$

其中, p_1 为处理器控制指示灯显示‘→’; p_2 为处理器控制指示灯显示‘←’; p_3 为处理器控制指示灯显示‘↑’; p_4 为处理器控制指示灯显示‘↓’。

He 等^[29]考虑了具有主机 A 与主机 B 的总线结构, 拒绝服务型硬件木马被植入在主机 B 中。正常工作模式下, 两个主机拥有相同的优先级, 不会出现某个主机长时间占用总线的情况。而主机 B 中植入的硬件木马能够修改并调高主机 B 的优先等级, 此时如果主机 B 不断请求总线的使用

权限, 主机 A 将无法访问总线参与通信, 影响 SoC 的正常功能, 发生拒绝服务攻击。研究者首先提取了电路的有限状态机模型, 对电路进行了完备的形式化描述。随后提出了针对该类木马攻击的电路安全属性(总线的使用权限不能被某个主机长期占用, 尤其当其他主机也在请求总线的使用权限时)。研究者采用了递进式的验证方法, 首先验证是否存在主机 B 长期占用总线的情况, 进而验证在这种情况下主机 A 是否尝试取得总线的使用权限。在 UPPAAL 解析器中对上述安全属性进行验证, 成功检测出了拒绝服务型硬件木马。

3.4 信息泄露型硬件木马的模型检测技术

信息泄露型硬件木马是母本电路中的冗余电路, 借助电路的输出端口、扫描测试端口、射频发射模块以及电路耗散的侧信道信息等泄露电路内部的私密信息。对于这一类硬件木马, 验证人员通常通过“敏感信息不能被已知模块以外的模块获取”这一安全属性进行验证。

He 等^[29]针对总线结构中植入在从机的读取总线信息的硬件木马, 提出了由主机存储给特定从机的信息不应该被泄露给总线上其他从机的安全属性, 首先通过状态存在性的约束确定了电路的有限状态机模型中确实存在信息泄露, 进而通过状态可达性的约束确定了硬件木马被植入的位置, 成功检测出了信息泄露型硬件木马。

Rajendran 等^[30]在检测信息泄露型硬件木马时, 首先考虑安全属性, 即存在一种输入的情况能够使密钥被输出。尽管这一安全属性存在诸多缺陷, 如需要检测所有状态情况、无法检测分段输出的密钥等, 它仍然是检测信息泄露型木马的基础属性。

4 结束语

近年来, 基于模型检测的硬件木马检测技术逐渐成为硬件安全领域研究的热点。对于面向硬件木马检测的模型检测技术来说, 状态爆炸问题始终是限制其发展的最主要的瓶颈^[31]。对于一个具有 n 个逻辑门的电路来说, 其所具有的电路状态数目为 2^n 。而如今超大规模集成电路已经被广泛生产和应用, 想要对具有数百万的逻辑门的电路系统进行完备的模型建立与验证十分具有挑战性。

在学术界,缓解状态爆炸的主要方法有4种:组合推理、抽象技术、对称、归纳。由于电路设计具有模块化的特点,面向硬件安全的模型检测研究者通常采用组合推理的方式来缓解电路建模的状态爆炸问题,即对电路进行分模块建模或者对于特定电路模块而非全部电路进行建模与验证^[29-32]。而分模块建模的方法必须要考虑的问题是分模块建模的情况下仍然需要保证模型能够完备地描述全部电路状态,针对特定模块进行建模则需要对硬件木马的攻击模式有相当全面的了解和认识。这些问题在一定程度上限制了基于模型检测的硬件木马检测技术的实际应用。

基于上述研究现状,对电路模型的约简可作为未来的一个研究方向。例如,硬件电路中拥有许多重复的单元,如存储单元等,在建模过程中可以通过特定的技术手段对这些重复单元对应的状态机进行化简,进而达到减少验证开销的目的。

参考文献:

- [1] AGRAWAL D, BAKTIR S, KARAKOYUNLU D, et al. Trojan detection using IC fingerprinting[C]//IEEE Symposium on Security and Privacy (SP '07), 2007: 296-310.
- [2] FLOTTE M, DUPUIS S, BA P, et al. On the limitations of logic testing for detecting hardware Trojans Horseshoe[C]//10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), Naples. 2015: 1-5.
- [3] HELY D, MARTIN J, DARIO G, et al. Experiences in side channel and testing based hardware Trojan detection[C]//IEEE 31st VLSI Test Symposium (VTS). 2013: 1-4.
- [4] CLARKE E M, GRUMBERG, PELED D, et al. Model checking[M]. Cambridge: MIT press, 1999.
- [5] BARTOLETTI M, DEGANI P, FERRARI G L, et al. Model checking usage policies[C]//Trustworthy Global Computing. 2009: 19-35.
- [6] ALESSANDRO C, EDMUND C, FAUSTO G, et al. NUSMV: a new symbolic model checker[J]. STTT 2, 2000: 410-425.
- [7] MCMILLAN K L. Symbolic model checking: an approach to the state explosion problem[M]. Kluwer Academic, 1993.
- [8] SZWED P. Efficiency of formal verification of ArchiMate business processes with NuSMV model checker[C]//Federated Conference on Computer Science and Information Systems (FedCSIS), Lodz. 2015: 427-436.
- [9] XU N, MA Z, JIANG J, et al. Model checking instance based on NuSMV[C]//IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCOM/IOP/SCI). 2018: 2052-2056.
- [10] SEBASTIANI R, ROVERI Marco, PISTORE M, et al. NuSMV2: an open source tool for symbolic model checking[C]//International Conference on Computer Aided Verification. 2002.
- [11] HASAN SR, KAMHOUA CA, KWIAT KA, et al. Translating circuit behavior manifestations of hardware Trojans using model checkers into run-time Trojan detection monitors[C]//IEEE Asian Hardware-Oriented Security and Trust (AsianHOST). 2016: 1-6.
- [12] DESAI K, STEVENS K. S, O'LEARY J, et al. Symbolic verification of timed asynchronous hardware protocols[C]//IEEE Computer Society Annual Symposium on VLSI (ISVLSI). 2013: 47-52.
- [13] PAKONEN A, BUZHINSKY I. Verification of fault tolerant safety I&C systems using model checking[C]//International Conference on Industrial Technology. 2019.
- [14] BEHRMANN G, DAVID A, LARSEN K G A. Tutorial on UPPAAL[C]//Proc of Int school on Formal Methods for the Design of Computer Communication & Software Systems. 2004: 200-236.
- [15] AAAMIR N, FAROOQUE A, AMJAD A, et al. Comparison of model checking tools using timed automata - PRISM and UPPAAL[C]//IEEE International Conference on Computer and Communication Engineering Technology (CCET). 2018: 248-253.
- [16] YAN X, LI Y, LI X. Real-time simulation of automotive systems based on UPPAAL[C]//Proceedings of 2017 IEEE 8th International Conference on Software Engineering and Service Science. 2017: 193-196.
- [17] CHAUDHTY Y. A. K, HAMMED M. Formal verification of cloud based distributed System using UPPAAL[C]//International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT). 2019: 1-4.
- [18] YAN X, WANG L, CHE X, et al. Source code testing for automotive software based on UPPAAL model[C]//IEEE International Conference on Software Engineering & Service Science, 2014: 95-98.
- [19] LI H, LIU Q, ZHANG J, et al. A Survey of hardware Trojan detection, diagnosis and prevention[C]//IEEE International Conference on Computer-aided Design & Computer Graphics. 2016.
- [20] HASAN S. R, KAMHOUA C. A, KWIAT K. A, et al. A novel framework to introduce hardware Trojan monitors using model checking based counterexamples: inspired by game theory[C]//2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS). 2018: 853-856.
- [21] LODHI F K, ABBASI I H, KAMBOH A M, et al. Formal verification of gate-level multiple side channel parameters to detect hardware Trojans[C]//Communications in Computer and Information Science. 2016.
- [22] HAFEEZ A I, FAIQ K, OSMAN H, et al. McSeVIC: a model checking based framework for security vulnerability analysis of in-

egrated circuits[J]. IEEE Access, 2018, 6:1.

- [23] RATHMAIR M, SCHUPFER F, KRIEG C. Applied formal methods for hardware Trojan detection[C]//IEEE International Symposium on Circuits & Systems. 2014: 169-172.
- [24] ZHANG Y, YU L, LI H, et al. Small Trojan testing using bounded model checking[C]//IEEE International Test Conference. 2018.
- [25] ZHAO J F. Case study: discovering hardware Trojans based on model checking[C]//The 8th International Conference. 2018: 64-68.
- [26] KRIEG C, RATHMAIR M, SCHUPFER F. A process for the detection of design-level hardware Trojans using verification methods[C]//High Performance Computing & Communications, IEEE Intl Symp on Cyberspace Safety & Security, IEEE Intl Conf on Embedded Software & Syst. 2015.
- [27] 高洪博. 指令诱发型硬件木马检测技术研究[D]. 郑州: 信息工程大学, 2013.
GAO H B. Research on detection techniques of instruction-triggered hardware Trojan horse[D]. Zhengzhou: Information Engineering University, 2013.
- [28] GAO H, LI Q, ZHU Y, et al. Code-controlled hardware Trojan horse[C]//International Conference on Information Computing & Applications. 2012: 171-178.
- [29] HE J J, GUO X, MEADE T, et al. SoC interconnection protection through formal verification[J]. Integration, 2019, 64: 143-151.
- [30] RAJENDRAN J, DHANDAYUTHAPANY A M, VEDULA V, et al. Formal security verification of third party intellectual property cores for information leakage[C]//IEEE International Conference on VLSI Design & International Conference on Embedded Systems. 2016: 547-552.
- [31] CLARKE E M, KLIEBER W, MILO N, et al. Model checking and the state explosion problem[J]. Lecture Notes in Computer Science, 2011: 1-30.
- [32] CRUZ J, FARAHMANDI F, AHMED A, et al. Hardware Trojan detection using ATPG and model checking[C]//International Conference on Vlsi Design & International Conference on Embedded Systems. 2018: 91-96.

[作者简介]



张启智（1998—），男，山东菏泽人，天津大学博士生，主要研究方向为集成电路设计与硬件形式化验证。



赵毅强（1964—），男，河北辛集人，博士，天津大学教授、博士生导师，主要研究方向为集成电路安全、混合信号集成电路设计、光电检测与成像系统设计、传感器系统设计。



高雅（1998—），女，辽宁东港人，天津大学硕士生，主要研究方向为数字电路安全。



马浩诚（1996—），男，河北沧州人，天津大学博士生，主要研究方向为集成电路设计与硬件安全。