Swarup Bhunia · Mark M. Tehranipoor
Editors

The Hardware Trojan War

Attacks, Myths, and Defenses

Springer

The Hardware Trojan War

Swarup Bhunia • Mark M. Tehranipoor
Editors

The Hardware Trojan War

Attacks, Myths, and Defenses

Springer

Editors
Swarup Bhunia
Department of Electrical &
Computer Engineering
University of Florida
Gainesville, FL, USA

Mark M. Tehranipoor
Department of Electrical &
Computer Engineering
University of Florida
Gainesville, FL, USA

ISBN 978-3-319-68510-6 ISBN 978-3-319-68511-3 (eBook)
https://doi.org/10.1007/978-3-319-68511-3

Library of Congress Control Number: 2017958447

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Acknowledgments

Editing this book has been a uniquely fulfilling experience. The project has met several obstacles and challenges on its way. Eventually, we overcame them all and came up with a content of the book which is both high-quality and comprehensive. We are proud to be able to publish the first book to our knowledge on this topic of emerging significance. We believe malicious modification of electronic hardware, aka hardware Trojan attack, which is covered by this book, would not only remain a pressing security concern for the entire electronics industry in the coming years – but evolve into a major field of research and innovation in the broader field of hardware security as well. With that, we hope this book would keep serving its purpose as a valuable reference on this important topic.

The book greatly benefited from a number of people in many ways. Several students in FICS Research Lab, at the University of Florida, in particular, Atul Prasad Deb Nath, greatly helped in the compilation process. Of course, the project would not be successful without the contributions of many researchers and experts in the field of hardware security and trust. We would like to gratefully acknowledge the valuable contributions from all the chapter contributors.

This work has been supported in part by research grants from the National Science Foundation, the Semiconductor Research Corporation, and Cisco. Any opinions, findings, conclusions, or recommendations presented in this book are only those of the authors and contributors and do not necessarily reflect the views of the National Science Foundation, the Semiconductor Research Corporation, or Cisco.

Contents

Part I
Hardware Trojan Preliminaries

Chapter 1
Introduction

Swarup Bhunia, Atul Prasad Deb Nath, and Mark M. Tehranipoor

Security of a computer system has been traditionally related to the security of the software or the information being processed. The underlying hardware used for information processing has been considered as the proverbial "root of trust" or trust anchors. Malicious hardware modifications – e.g., hardware Trojan attacks, which brought to light just a decade ago – are potent attacks on electronic hardware that violates the fundamental assumption of the hardware root of trust. These attacks, in the form of malicious modifications of electronic hardware at different stages of its life cycle, pose major security concerns in the electronics industry. An adversary can mount such an attack with an objective to cause in-field operational failure potentially leading to catastrophic consequences or to leak secret information from a chip – e.g., the key in a cryptographic chip, to an unauthorized party. The global economic trend that encourages increased reliance on untrusted entities in the hardware design and fabrication process is rapidly enhancing the vulnerability to such attacks. This book ventures an all-inclusive approach to illustrate the threats originating from hardware Trojan attacks, present different Trojan attack models with elaborate description on the types and scenarios, and describe existing trust metrics and various forms of protection approaches, defenses, and future research directions. The research progress on hardware Trojan is remarkable in academia and industrial arena. The research activities in this area have grown significantly over the decade with new vulnerabilities and solutions frequently reported by researchers across the globe.

Hardware Trojan attacks have emerged as a major security concern for electronic hardware at all levels of abstractions – hardware intellectual property (IP) blocks, integrated circuits (ICs), printed circuit boards (PCBs), and systems. These attacks

S. Bhunia (✉) • A.P. Deb Nath • M.M. Tehranipoor
Department of Electrical & Computer Engineering, University of Florida, Gainesville, FL, USA
e-mail: swarup@ece.ufl.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_1

relate to malicious modifications of a hardware during design or fabrication in an untrusted design house or foundry, which involve untrusted people, design tools, or components. Such modifications can give rise to an undesired functional behavior of a hardware, or degrade performance, or degrade performance, or provide covert channels or back doors through which sensitive information can be leaked – thus leading to violation of the hardware root of trust. An adversary is expected to make a Trojan stealthy in nature that evades detection through the conventional post-manufacturing test but manifests during long hours of field operation. There is growing interest in this topic in both academia and industry with a significant increase in research activities across the globe on all topics related to hardware Trojan attacks. This book, for the first time, provides comprehensive coverage on hardware Trojan attacks, highlighting the evolution of the threat, the challenges, and diverse array of defense approaches. It tries to debunk the myths associated with hardware Trojan attacks and presents practical attack space in the scope of current business models and practices. It covers the threat of hardware Trojan attacks; presents attack models, types, and scenarios; discusses trust metrics; describes different forms of protection approaches, both proactive and reactive; and, finally, describes emerging attack modes, defenses, and future research pathways.

1.1 Why a New Book

Current industrial practices such as increased reliance on third-party hardware IP and automation tools in the design flow as well as outsourcing of design/fabrication steps to external parties due to economic reasons are rapidly increasing the vulnerability to Trojan attacks. With growing interest in this topic in industry and academia alike, there is a critical need for a book on this important topic. Proposing a new book in this area will impact the field in the following ways:

1. There is no comprehensive resource to provide researchers, students, and practitioners with fundamentals of hardware Trojan attacks – spanning attack models/instances and all sorts of protection approaches. This will be the first and only book in the market on this exciting topic of emerging significance in the broad area of hardware security.
2. The book will include a description on trust benchmarks and trust metrics, which will provide valuable resources for researchers in this area.
3. A trained cybersecurity workforce for private and government sectors is needed to fully understand the risks of hardware Trojan attacks and evaluate, implement, and improve upon protection methods against these attacks. This book is written in a manner that makes its content accessible to both beginners (e.g., undergraduate students) and advanced researchers and practitioners in order to fill this need.

4. The coverage on Trojan vulnerabilities at different hardware (IC and PCB) design and fabrication steps will make manufacturers and government policy makers aware of how different steps involve Trojan threat and how appropriate protection method can be implemented.
5. This book contains information on topics that are underexplored in the current hardware security literature including hardware Trojan threats for emerging nanoscale devices.

1.2 Benefits to the Readers

The book is expected to provide the following benefits to its readers:

1. Readers will get the most comprehensive coverage on hardware Trojan attacks based on decade-long research on this significant topic.
2. Readers will become familiar with all types of Trojan attacks – including Trojan taxonomy, attack surfaces, threat models, and realistic attack instances.
3. Readers will understand Trojan vulnerabilities at levels of hardware abstraction – from IP to system and all types of hardware (e.g., digital, mixed-signal) and corresponding defense approaches.
4. Readers will gain insight on emerging threats associated with future nanoscale technology nodes, globally distributed supply chain, and complexity of modern hardware as well as challenges/opportunities developing effective countermeasures.
5. Readers will gain knowledge on procedure for hardware trust evaluation, trust benchmarks, and trust metrics.

1.3 About the Topic

Over the past decade, different possible forms of Trojan attacks have been reported, and the adversarial model and attack surface have evolved [1–6]. A Trojan attack, in its simplest form, can be mounted by adding a circuit block that will trigger anomalous functional or parametric behavior of a design. Figure 1.1 shows a simplified block diagram of a hardware Trojan, which causes malfunction (by flipping signal S) when triggered – i.e., when the activation condition realized by a trigger logic is true [1]. These circuit blocks can implement a Boolean function that would create a trigger condition when specific internal nodes reach a suitable state (e.g., memory write control asserted and two most significant bits of data set to logic value "10"). The trigger condition can be digital or analog (e.g., state of a node capacitance or operating temperature). There can also be Trojans which are always on [6] – i.e., where the trigger condition does not depend on the circuit or environmental state. The other part of a Trojan is the malicious effect or payload that

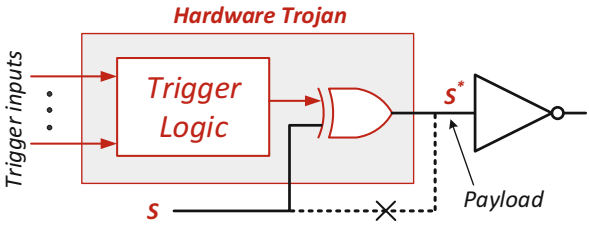


Fig. 1.1 General structure of a hardware Trojan in a design [1]

determines how the design deviates from its desired behavior. The payload can alter the internal circuit state at specific nodes leading to disruptive functional behavior when the trigger condition is satisfied. It may also cause an undesired change in parametric behavior (e.g., creating a current spike).

Hardware Trojan attack model assumes untrusted design or fabrication process, which means they involve either untrusted design components (e.g., IPs or design tools) or rogue designer or entities (e.g., fabrication facilities). Malicious alteration of a design can be caused by one or combination of these elements. Figure 1.2 shows the life cycle of an IC from design conception process, which leads to the creation of the design specification, to deployment. The stages, in general, have different vulnerabilities with respect to Trojan attacks, and the level of vulnerabilities depends on a company's business model and design practices. For example, fabless companies that rely on untrusted foundry would have vulnerabilities coming from the untrusted foundry. Similarly, a company which relies on third-party IP would be vulnerable to IP-level Trojans. The stages marked in red – chip fabrication and printed circuit board (PCB) fabrication and system integration – are typically untrusted since these stages involve third-party fabrication facilities which need to have access to the entire design of chip or PCB. An adversary in these facilities may alter the design in many ways to incorporate a Trojan. Other stages marked in yellow can either be trusted or untrusted depending on business models and practices followed by individual companies.

Current economic trend plays a major role in enhancing the vulnerability to Trojan attacks [1, 3]. IC design and manufacturing practices increasingly rely on untrusted parties and entities in the IC life cycle. Economic reasons dictate that most of the modern ICs are manufactured in offshore fabrication facilities. Moreover, modern IC design often involves IP cores supplied by third-party vendors, outsourced designs, and test services as well as electronic design automation (EDA) software tools supplied by different vendors. Such a business model has, to a large extent, relinquished the control that IC design houses had over the design and manufacture of ICs making them vulnerable to malicious inclusions. Figure 1.3 illustrates the attack surface and different forms of Trojan attacks that can be mounted in key stages of a system-on-chip (SoC) life cycle. It also shows how existing research in this field tries to mitigate the threat of Trojan attacks at these stages. Depending on the stage and form of Trojans, various testing or design solutions have been proposed in open literature. The figure gives a snapshot of

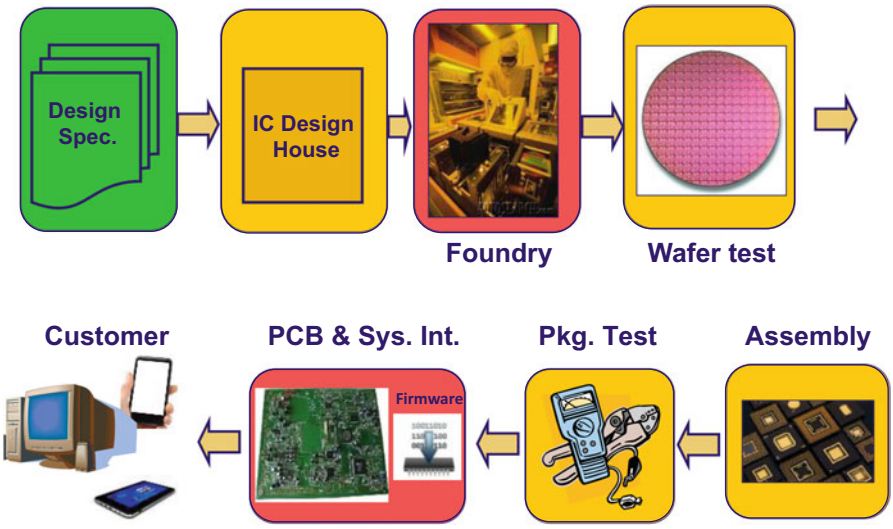


Fig. 1.2 IC life cycle from design to deployment and stages vulnerable to Trojan attacks. The stages marked in *red* are typically highly vulnerable, and the stages marked in *yellow* are less vulnerable. Vulnerability of these stages largely depends on business model of specific company

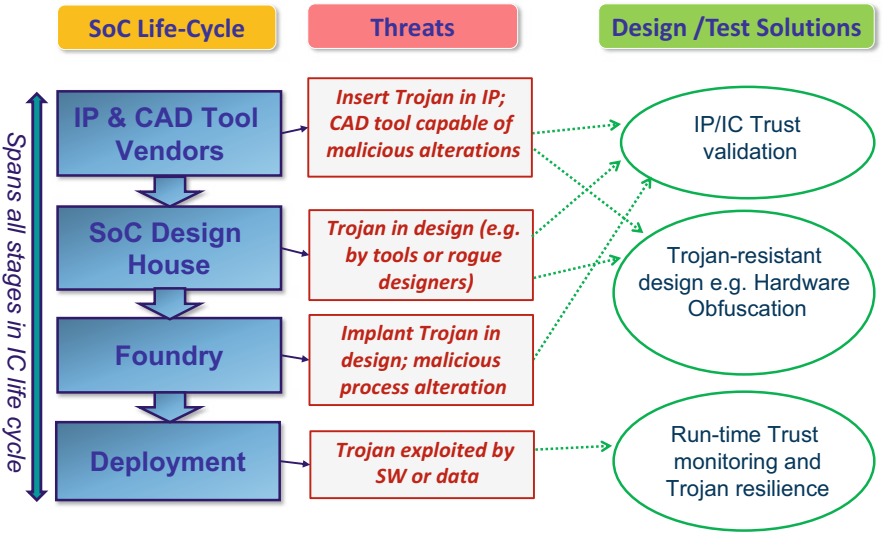


Fig. 1.3 Hardware Trojan attacks by different parties at different stages of SoC life cycle

various design/test techniques for protecting against Trojan attacks. Latter chapters elaborate on both these attacks and protection approaches.

Ideally, any undesired modification made to an IC should be detectable by pre-silicon verification/simulation and post-silicon testing. However, pre-silicon

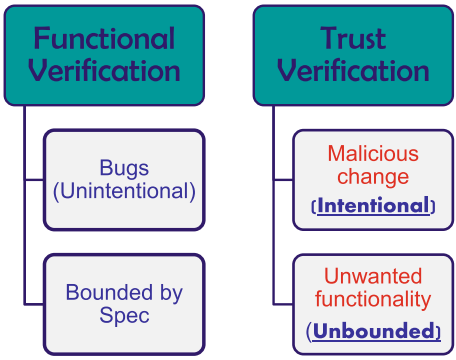


Fig. 1.4 Major distinctions between traditional functional verification and hardware trust verification

verification or simulation requires a completely specified golden model of the entire IC. This might not be always available, especially for IP-based designs where IPs can come from third-party vendors or for chips which are acquired from an untrusted supply chain by a system integrator. Besides, a large multi-module design is usually not amenable to exhaustive functional verification for detecting malicious design modification [1]. Figure 1.4 shows the distinctions between traditional functional verification and hardware trust verification. Traditional post-manufacturing testing is not suitable for detecting hardware Trojans. This is due to the stealthy nature of hardware Trojans and the vast spectrum of possible Trojan instances an adversary can exploit. Unlike a bug or fault, a Trojan is deliberately inserted into a design, and it causes unwanted functionality outside the specification. While traditional functional verification is bounded by the specification – i.e., it verifies that if a design meets its intended functional behavior, but does not check the unbounded space of what the design does outside this specification. The Chaps. 3, 4, 5, and 6, which illustrate Trojan attacks, give a number of examples on how a Trojan causes additional functional or parametric behavior beyond its desired specification.

Post fabrication, the design can be verified either through destructive de-packaging and reverse engineering of the IC or by comparing its functionality or circuit characteristics with a golden version of the IC [4–5]. However, state-of-the-art approaches do not allow destructive verification of ICs to be either cost-effective or scalable. Moreover, it is possible for the adversary to insert Trojans in only some ICs on a wafer, not the entire population, which limits the usefulness of a destructive approach.

1.4 The Content of the Book

This book is an attempt to bridge across the research and practice in hardware Trojan and is conceived as an authoritative reference for researchers interested in all aspects of Trojan attacks. The book includes 16 chapters focusing on diverse aspects of

hardware Trojan attacks and protection approaches. The book is divided into six parts. Next, we introduce briefly the content of each of the chapters.

Part I of the book focuses on hardware Trojan preliminaries including the history and evolution, definitions, taxonomies, and industrial perspectives on current security trend.

Chapter 2 on "Introduction to Hardware Trojans" by Jason Vosatka provides a comprehensive overview of hardware Trojans. The study elaborately describes the definitions of hardware Trojans and how they vary from bugs, defects, or software Trojans. A detailed depiction of the comparison and misconceptions with Trojans are delineated in this chapter. It illustrates the trends, trade-offs, and threats posed by malicious hardware Trojans. A comprehensive taxonomy is presented to provide insights about offensive strategies of Trojan attacks. The primary categories in the taxonomy include classifications based on insertion phase, abstraction level, activation mechanism, effects, and location. A classification of different Trojan attack models is provided based on 3PIP vendors, SoC developers, and chip foundries. The defensive strategies against Trojan attacks are elaborated by providing a taxonomy of Trojan countermeasures. The primary countermeasure techniques for Trojan detection like design-for-trust (DFT) and split manufacturing are also discussed in this study.

Part II (Chaps. 3, 4, 5, and 6) of the book contains threat analysis of hardware Trojans and discusses preventive and assistive approaches to make Trojan insertion difficult and the detection process more efficient.

Chapter 3 on "Hardware Trojan Attacks in SoC and NoC" by Rajesh JS, et al. discusses the emerging hardware Trojan attacks and countermeasures employed in SoC security, with a special focus on Trojans in the network on chip (NoC). The work highlights the challenges and vulnerabilities associated with the current practices of SoC security assurance. Security assurance of system-on-chips has become a burning issue with the rapid growth of digital footprints by internet of things) in the context of untrusted SoC operation in mobile, embedded systems, and high-performance computing. The study provides an overview of the most conspicuous threat model along with some promising system level solutions proposed for SoC security assurance. It classifies the attacks on NoC into several categories including information leakage attacks, attacks causing network interface malfunctions, denial of service attacks, and fault injection-based denial of service attacks. The work also includes a study on the less explored territory of untrusted malicious NoC 3PIPs with sample attack models and solutions.

Chapter 4 on "Hardware IP Trust" by Mainak Banga and Michael S. Hsiao provides a high-level overview of the security and trust issues related to third-party IPs (3PIPs), potential vulnerabilities due to hardware Trojans, and possible countermeasures to prevent Trojan attacks. A taxonomy of hardware Trojans based on physical, action, and activation characteristics is illustrated in the work. The major categories of the taxonomy include classification by types, sizes, distribution, activation method, function, and specification of the Trojans. The chapter classifies the mitigation techniques broadly into two categories, i.e., preventive and detection techniques which encompass approaches like design-for-trust and obfuscation-

based methods, as well as delay and power-based detection methods. For IP-level Trojan detection, the work describes suspect signal-guided sequential equivalence checking and introduces proof-carrying code as a case study of prevention technique.

Chapter 5 on "Hardware Trojans in Analog, Mixed-Signal and RF ICs" by Angelos Antonopoulos et al. explores the threats arising from hardware Trojans in analog, mixed-signal, and RF ICs. Compared to the research progress made over the last decade in detecting and preventing hardware Trojan attacks in digital integrated circuits, their analog/mixed-signal (AMS) and radio-frequency (RF) counterparts are still largely unexplored in terms of threat modeling and vulnerability analysis. The security of AMS and RF IC, however, is crucial to most contemporary computing systems given the widespread use of analog functionality, i.e., physical interfaces, sensors, actuators, wireless communications, etc. The chapter provides an in-depth description of the attack models and defense mechanisms for hardware Trojans in wireless cryptographic ICs, RF transmissions below the noise floor, and AMS ICs. Apart from Trojan attacks, the work sheds light on issues related to IC/IP piracy and counterfeiting, vulnerability analysis, and associated countermeasures like IP watermarking, counterfeit protection strategies, and split manufacturing.

Part III (Chaps. 6, 7, 8, and 9) of the book illustrates functional testing-based approaches for Trojan detection including statistical and directed testing, formal verification, and golden-free trust verification methods.

Chapter 6 on "Hardware Trojans and Piracy of PCBs" by Anirudh Iyengar and Swaroop Ghosh focuses the security issues of printed circuit boards (PCBs) rising from Trojans and piracy due to the soaring complexity of modern printed circuit boards and high reliance on third-party entities. The chapter studies the possible attack models in scenarios where PCBs are assumed both trusted and untrusted and further analyzes the attacks based on the motives of the adversary, i.e., either causing malfunction or leaking secret information. While illustrating attack instances, the issues of trusted and untrusted design houses are taken into consideration. Preventive countermeasures like hardening via secure interfaces and several other methods to design secure PCBs are discussed in the chapter. Novel PUF structures like star-coil PUF and arbiter-coil PUF are discussed in the work to address PCB authentication challenges. Additionally, analysis of various sources of variations in PCBs and qualitative study of the quality metrics to evaluate the PCB PUFs are presented in the work.

Chapter 7 on "Logic Testing for Hardware Trojan Detection" by Vidya Govindan and Rajat Subhra Chakraborty discusses the challenges of conventional post-manufacturing testing, test generation algorithms, and test coverage metrics and introduces a test pattern generation technique termed multiple excitation of rare occurrence (MERO). MERO is an effective logic testing algorithm for relatively small (e.g., less than ten two-input NAND gate equivalent) hardware Trojan detection. The probability of inserted Trojans getting triggered and detected by logic testing can be maximized by employing MERO while attaining a significant reduction in the number of vectors compared to weighted random pattern-based test generation. The work further discusses an extension of MERO by exploiting

the combined strength of genetic algorithm and Boolean satisfiability and proposes an automatic test pattern generation (ATPG) scheme for the detection of Trojans dependent on rare input triggering conditions.

Chapter 8 on "Formal Approaches to Hardware Trust Verification" by Farimah Farahmandi et al. elaborates hardware trust verification by formal methods. Due to the stealthy nature of hardware Trojans, simulation-based validations are not effective for the detection of rarely triggered malicious circuitry like Trojans. This chapter discusses the advantages of formal validation approaches in proving the correct functionality of a design to detect surreptitious hardware Trojans. Though scalability issues can limit the capability of formal approaches to validate digital circuitry in many cases, there are several formal techniques like satisfiability solvers, model checkers, theorem provers, and symbolic algebras that are efficiently scalable to validate larger designs. The chapter provides an in-depth overview of various formal approaches for verification of hardware security and trust. With soaring importance and complexity of designing trustworthy SoC architectures, the formal approaches are expected to gain further research attention in the future.

Chapter 9 on "Golden-free Trojan Detection" by Azadeh Davoodi discusses the challenges associated with golden-dependent Trojan detection methods and introduces golden-free IC authentication process. The fundamental limitations of golden-dependent Trojan detection include the rare possibility of finding a golden IC and identifying its standard features. Golden-free IC authentication process, however, is a viable approach to Trojan detection that can eliminate the golden IC from the authentication loop. The chapter provides a general overview of existing techniques for golden-free Trojan detection and describes one technique in detail, as a case study for self-authentication of a chip based on assistance from custom design-dependent on-chip sensors. For the sensor-assisted self-authentication framework, the work depicts three scenarios, i.e., when the sensor is inserted in the design paths, inserted in the sensors, and inserted in both. The primary idea behind sensor-assisted self-authentication is to use on-chip delay information from the sensors to make an accurate prediction about the expected path delays. A higher number of design paths increase the probability of Trojan detection with an overhead of longer testing time.

Part IV (Chaps. 10 and 11) of the book encompasses studies on reverse engineering and side-channel analysis-based Trojan detection methods.

Chapter 10 on "Detecting Hardware Trojans using Delay Analysis" by Jim Plusquellic and Fareena Saqib surveys different delay-based techniques to detect hardware Trojans by exploiting precise analog testing. The delay-based Trojan detection methods can be characterized by the Heisenberg principle or observer effect which states that any attempt to measure or monitor a system leads to a change in its behavior. The chapter provides a comprehensive survey on path delay-based testing methods that can detect subtle changes in delay caused by Trojan connections and gate insertions which are basically the triggers and payloads of the hardware Trojans, respectively. A high-level depiction of Trojan insertion strategies is provided in this chapter with an emphasis on detection of layout or GDSII Trojans. The constraints of detection methods are also described with an elaborate discussion

on analyzing side-channel signals, e.g., power and delay information. The work also introduces multiple-parameter-based side-channel method.

Chapter 11 on "Reverse-Engineering Based Hardware Trojan Detection" by Chongxi Bao et al. depicts the fundamentals of IC reverse engineering and describes how reverse engineering techniques can be employed to detect hardware Trojans. The chapter introduces a machine learning approach to differentiate between Trojan-free and Trojan-inserted ICs. The work includes training and testing support vector machine (SVM) with features extracted from IC images obtained from a scanning electron microscope. Experimental results demonstrate very high accuracy of SVM-based reverse engineering technique in detecting Trojan-free and Trojan-infected ICs. The chapter also studies security-aware design strategies for Trojan prevention as these are crucial aspects of developing secure hardware platforms. A novel design-for-security approach is proposed in this work where the standard cells with higher sensitivity to hardware Trojan insertion are selected to synthesize the layout of the chip. The approach helps to attain significant improvement in the fraction of Trojan detection with an acceptable overhead.

Part V of the book (Chaps. 12, 13, and 14) describes the preventive approaches introduced at the design phase of the product to thwart Trojan attacks.

Chapter 12 on "Hardware Obfuscation Methods for Hardware Trojan Prevention and Detection" by Qiaoyan Yu et al. focuses on the application of hardware obfuscation methods during IC design to prevent the outsourced chip from being tampered or reverse engineered by malicious adversaries. The chapter discusses the idea of introducing hardware obfuscation at different levels of abstraction, i.e., device, circuit, gate, and register transfer levels. The work also provides an overview of approaches of hardware obfuscation in FPGAs and PCBs. The obfuscated hardware design makes it more difficult for attackers to understand the original circuit functionality. Moreover, the obfuscation process can be incorporated with authentication techniques to generate special signatures for Trojan detection.

Chapter 13 on "Deterrent Approaches against Hardware Trojan Insertion" by Qihang Shi, Domenic Forte, and Mark M. Tehranipoor introduces several deterrent approaches employed at the design phase of ICs to prevent hardware Trojan insertion. Contrary to conventional countermeasures of hardware Trojan, deterrent approaches prevent Trojan insertions via design level modifications eliminating the requirements of future Trojan detection. The chapter mainly discusses three primary deterrent approaches, i.e., monitoring approach, obstructive approach, and hybrid approach. The monitoring approach includes measuring and classifying side-channel signatures. The obstructive approach obstructs the adversary's access to necessary information required to insert a Trojan. The hybrid approach combines both monitoring and obstructive approaches to prevent Trojan insertion. The chapter describes the archetypal techniques for each of the approaches and illustrates their strengths and weaknesses on a relative basis. While the advantages and disadvantages of the approaches are product and application specific, the development of a holistic approach is yet subject to future research.

Chapter 14 on "Hardware Trojan Attacks in FPGA and Protection Approaches" by Vinayaka Jyothi and Jeyavijayan (JV) Rajendran discusses potential Trojan attacks on FPGAs and the possible countermeasures available in the literature to thwart such security risks. The chapter categorizes the threats originating from untrusted foundries and adversaries in the FPGA supply chain and presents a taxonomy tailored for FPGA-specific Trojans. It describes different types of Trojans that can be inserted into the FPGA fabric at various phases of the life cycle. The chapter provides an in-depth discussion on the primary classes of Trojans in FPGA and demonstrates several Trojan insertion methodologies. The existing countermeasures to prevent Trojans in FPGA are also described in the chapter. The countermeasures focus on verifying the physical fabric of the FPGA by taking account of spatially correlated intra-die process variations. The delay or voltage changes stemming from inconsistent physical characteristics, especially the ones in the close-by region, can aid in the Trojan detection in FPGA fabric.

Part VI of the book highlights the emerging trend, industrial practices, and new attack instances of hardware Trojan.

Chapter 15 on "Hardware Trust in Industrial SoC Designs: Practice and Challenges" by Sandip Ray focuses on security and trust assurance techniques and validation mechanisms employed in contemporary industrial practices and explores their complexities and limitations. It provides a brief introduction to the spectrum of trust vulnerability sources and discusses the security issues arising from the distributed nature of modern SoC design flow. The current practices of trust validation and design implementation are described based on adversarial and deployment methods. The study illustrates a comprehensive overview of the techniques from software/design targets to those involving the PCB and the platform. At present, it is challenging to perform a systematic analysis of the vulnerability sources due to the diversity of attacks and complexity of devices. Moreover, the widely varying level of expertise required for assurance activities from architecture to RTL/software models and to silicon implementation attacks poses further difficulties to trust assurance. To eliminate the gross inadequacy in current standards, a cohesive assurance approach is required that can seamlessly move toward various abstraction levels and provide comprehensive solutions for modern SoC design security issues.

Chapter 16 provides a summary of the chapter contents and describes future research directions on hardware Trojan attacks. It enlists specific topic areas which are likely to see growing number of research activities in both academia and industry.

It has been pleasure and honor for the editors to edit this material, and we hope the broad coverage on malicious modifications of electronic hardware bridges an important gap in the current offerings on the hardware security space. We believe the content of the book will provide a valuable reference on hardware Trojan issues and solutions to a diverse readership including students, researchers, and industry practitioners. Of course, it is impossible for any book on the topic to be exhaustive like most areas of hardware security. Nevertheless, we hope that the book will help to introduce the issue of hardware trust to its readers. Furthermore, it will provide a

flavor of the current state of the research in this area and emerging challenges across different sectors that need to be addressed to achieve the goal of understanding hardware trust issues and developing trustworthy computing systems.

References

1. S. Bhunia, M. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan attacks: Threat analysis and countermeasures. Proc. IEEE **102**(8), 1229–1247 (2014)
2. S. Ghosh, A. Basak, S. Bhunia, How secure are printed circuit boards against Trojan attacks? IEEE Des & Test **32**(2), 7–16 (2014)
3. K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, M. Tehranipoor, Hardware Trojans: Lessons learned after one decade of research. ACM Trans Des Autom Electron Syst (TODAES) **22**(1), 1–23 (2016)
4. S. Narasimhan, D. Dongdong, R.S. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, S. Bhunia, Hardware Trojan detection by multiple-parameter Side-Channel analysis. IEEE Trans. Comput. **62**(11), 2183–2195 (2012)
5. R.S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, S. Bhunia, MERO: A statistical approach for hardware Trojan detection, in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Lecture Notes in Computer Science, ed. by C. Clavier, K. Gaj (Eds), vol. 5747, (Springer, Berlin, 2009)
6. L. Lin, W. Burleson, C. Paar, MOLES: Malicious off-chip leakage enabled by side-channels. ICCAD, 117–122 (2009)

Chapter 2
Introduction to Hardware Trojans

Jason Vosatka

2.1 Overview of Hardware Trojans

Hardware Trojans are malicious modifications to the intended functionality of a hardware circuit [9, 17, 36, 37]. These modifications (i.e., tamperings) are undesired and unknown to the hardware designer and can have devastating effects on the electronic system. Trojans have three key characteristics: malicious intention, evasion of detection, and rarity of activation [6]. The intent of a Trojan is always the same: perform an unintended action to compromise the confidentiality, integrity, or authentication of the underlying hardware.

This compromise may be in the form of a shortened operational lifetime of the hardware (e.g., 5 years instead of 20 years) or complete failure of the system upon the Trojan's activation. It may allow an attacker to gain unauthorized access into the hardware (i.e., remote access through a backdoor) or lead to leakage of information (e.g., cryptographic keys for secure data communication). Hardware Trojans may manifest from software Trojans inside of pirated software tool suites during the synthesis portion of the design flow or be inserted as a result of collusion between multiple parties at different stages of the hardware's life cycle [45]. Trojans can also be designed with the sole intention to damage or destroy the brand reputation of a company, which may result in bankruptcy of the company and a competitive advantage for the adversary.

J. Vosatka (✉)
Florida Institute for Cybersecurity Research (FICS Research), Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, 32611, USA
e-mail: jvosatka@ufl.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_2

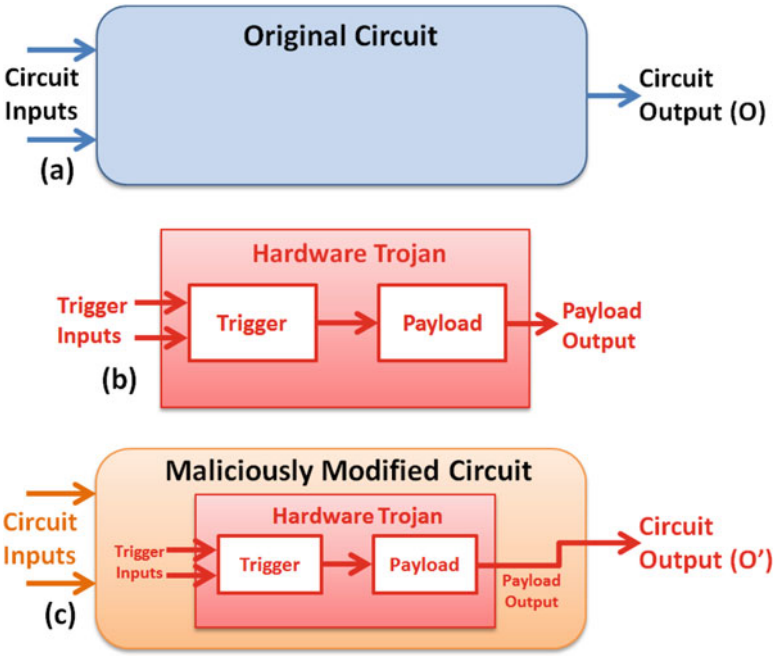


Fig. 2.1 Block diagram showing (**a**) original circuit, (**b**) simplified hardware Trojan, and (**c**) hardware Trojan inserted into the circuit, which inverts O to O' upon activation

Figure 2.1a shows a simplified block diagram of an original circuit, Fig. 2.1b shows the hardware Trojan with trigger and payload, and Fig. 2.1c shows the Trojan inserted into the circuit. When the Trojan activates in this example, its payload delivers a malfunction in the form of inverting the output of the circuit (i.e., changing the O to O').

Historically, hardware has been considered the *root of trust*, and the software or firmware that runs on top of the hardware has been untrusted until proven otherwise [6, 37]. However, much research has been performed over the last decade, and security researchers are aware of numerous Trojan models, attacks, countermeasures, as well as the threats to security and trust of the underlying hardware. Every entity (e.g., person, design house, foundry, supply chain) that is involved with the design, fabrication, testing, packaging, and delivery of an integrated circuit (IC) could be considered a potential adversary as they have the opportunity to tamper with the IC at multiple points in the design cycle. Therefore, the need exists to design "for security" in addition to specified functionality of the hardware.

A hardware Trojan is not a design or manufacturing fault. A fault (e.g., SA0, SA1, path delay) is an unintentional error or failure during these processes, and its location for activation is usually known by the designer. A Trojan, in any form, is an intentional insertion by an adversary, and its location for activation is unknown to

the designer. Although the intentions of a hardware Trojan are similar to its nefarious counterpart, the software Trojan, the hardware Trojan cannot be removed once the IC is fabricated, whereas the software Trojan can be eradicated post-deployment [6].

Trojans are activated by a specific mechanism, called a trigger, and deliver a specific function, called a payload. They can be small or large in size with respect to the rest of the circuit, ranging from just a few transistors to thousands of gates in a multimillion transistor SoC design [6]. Trojans exist in many forms and are most commonly triggered by a sequential or combinational digital circuitry (or a hybrid combination of both), but can also be triggered by analog stimuli. The payload can be digital or analog with each being specifically crafted to deliver malicious consequences upon activation.

Trojans are often undetectable to conventional pre-silicon and post-silicon manufacturing testing processes such as informal and formal verification [47]. This is due to test coverage for the hardware solely focusing on the specific functionality of the circuit, as well as exhaustive testing of all possible functions being both expensive and time-consuming. Trojans are intended to be stealthy and are inserted into rare internal nodes of the circuit, which reduces the likelihood of activation during normal testing. These nodes are often outside the scope (i.e., outliers or corner cases) of the circuits' intended functionality and are not activated during conventional test and verification methods. Unconventional methods do exist for detection, but these often require access to a verified and authentic (i.e., golden) IC or model for comparisons, and these methods may be performed as destructive (e.g., physical reverse engineering) or nondestructive (e.g., side-channel) testing methodologies [6, 37].

2.2 Trends, Tradeoffs, and Threats of Trojans

2.2.1 Semiconductor Design Flow

Over the last decade, there has been a shift in the global manufacturing model and design flow of companies that produce semiconductor ICs such as application-specific integrated circuits (ASIC), field-programmable gate arrays (FPGA), and system-on-chips (SoCs). These new trends are driven by various economic factors including monetary costs, time-to-market demands, and the increased complexity of semiconductors. Historically, hardware design companies performed the entire design flow as a trusted entity from "cradle-to-grave" including defining specifications, generating schematics and netlists, fabrication, testing, packaging, and delivery to the supply chain marketplace. This design flow was commonly referred to as a "vertical model." However, companies have been adopting a "horizontal model" in which they outsource certain steps of the design flow to untrusted entities and offshore foundries. Today, the majority of hardware design companies have fully adopted the horizontal model and now are rapidly moving toward a "fabless

model" in which they outsource all hardware fabrication. Many system integrators outsource both the hardware design and fabrication in order to maximize service to their customers and profits for their company.

The reliance on untrusted foundries reduces monetary costs for design companies as it eliminates the need to build and maintain a multibillion dollar fabrication facility. Using offshore foundries also allows design companies to have access to state-of-the-art fabrication technologies and reduce the risk of fabrication errors. By incorporating third-party intellectual property (3PIP) from untrusted entities into the hardware design, companies are able to decrease the time-to-market delivery and maximize the profit window of their product. These business decisions allow design companies to leverage the economies of scale created by the untrusted entities [39, 45].

However, these tradeoffs decrease the level of security and trust of the hardware, thus violating the traditional hardware *root of trust* philosophy. Relying on untrusted entities reduces control of the hardware design, thereby increasing the likelihood of a vulnerability being introduced during the design life cycle and supply chain distribution [45, 47]. These vulnerabilities exist in various forms including IP piracy, counterfeiting, cloning, overproduction, and hardware Trojans [6, 30, 37, 39, 40].

The modern horizontal model for semiconductor IC design and fabrication flow is shown in Fig. 2.2. Typically, a trusted entity (e.g., design house) is responsible for the specification, register-transfer level (RTL) design, netlist generation, and layout. An untrusted entity (e.g., foundry) is responsible for wafer fabrication, assembly, and testing. However, this generality is not always the case as demonstrated by the seven attack models described in Sect. 2.4.5.

2.2.2 Adversaries and Attacks

2.2.2.1 Adversarial Threats

Any entity that is involved with the design, fabrication, testing, packaging, or supply chain of an IC has the potential to be an adversary. Adversaries may be people, design houses, foundries, or even electronic design automation (EDA) or computer-aided design (CAD) software tools. What is common with all adversaries is the opportunity to tamper with the design (e.g., insert a Trojan) anytime the design is outside of the control of the rightful owner. These opportunities occur at the many stages of the IC's life cycle as shown in Figs. 2.2 and 2.3. The motivation of each adversary varies as does their design of the Trojan. Adversaries are motivated by many factors including monetary gains, increased market space in the supply chain, personal or political vendettas, tarnishing a competitor's reputation, or even the sole purpose of disrupting critical infrastructure through operational failures or leakage of sensitive information. Adversarial actions result in specially crafted malicious modifications to the original circuit, which are designed to achieve a specific nefarious goal.

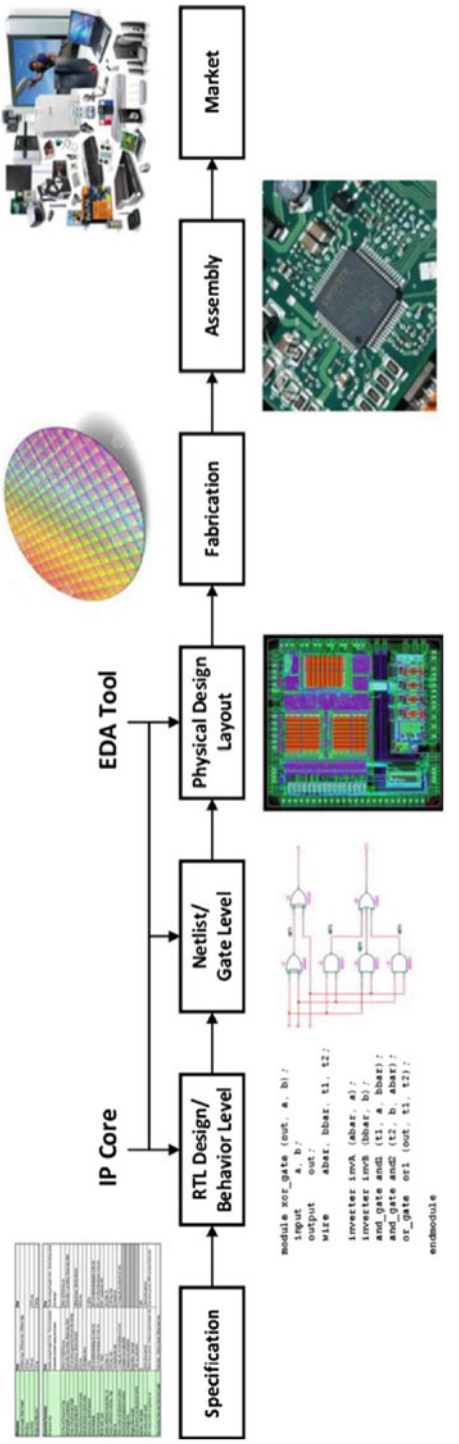


Fig. 2.2 Semiconductor IC design and fabrication flow [45]

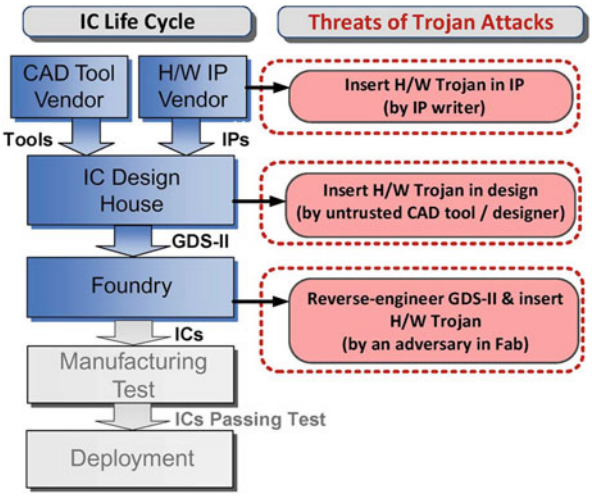


Fig. 2.3 Vulnerable stages in the integrated circuit (IC) life cycle [6]

2.2.2.2 Attack Surfaces

The goal of a hardware Trojan is as unique as its adversary. Trojans may be inserted into ICs including control circuitry, memory modules, sensors, and input/output drivers. They may also be inserted into embedded system processors allowing for software backdoors or inserted into cryptographic engines in SoCs to weaken, bypass, or disable the security features of the system [6]. Three examples of adversarial attacks that may occur during the design process are an untrusted third-party intellectual property (3PIP) vendor introducing a Trojan into the IP core of an SoC, an untrusted designer inserting a Trojan into unused cells of an IC, or even a trusted designer inadvertently inserting a hardware Trojan through the use of untrusted third-party EDA software tools.

Another example of attack surface is when an adversary at an untrusted foundry inserts a Trojan into the lithography mask during the fabrication process of the semiconductor wafer. This Trojan-infected wafer is then assembled into an IC and returned to the design company. Unless the design company has specific Trojan detection or prevention mechanisms for testing, such as a *golden IC* or *golden model*, the infected IC will enter the supply chain for integration into an unknowingly compromised electronic system. A *golden IC* or *golden model* is considered to be trustworthy as it has been verified to be fabricated exactly per the design specifications (i.e., nothing more, nothing less) and to be Trojan-free.

Furthermore, an adversary at a untrusted foundry or an untrusted design house could reverse engineer the entire IC design for the purpose of cloning (i.e., creating illegal copies) the IC. The adversary could insert a Trojan into the cloned IC and release the infected IC directly into the supply chain, thus bypassing any Trojan detection mechanism of the legitimate design owner. In this situation, there is no

golden model for verification, so integrators have to rely on other approaches such as self-referencing and side-channel analysis methods for Trojan detection [12, 23, 25].

Figure 2.3 shows examples of Trojan attacks during several vulnerable stages of the IC life cycle.

2.3 Comparisons and Misconceptions with Trojan Attacks

2.3.1 Trojans Compared with Bugs or Defects

Trojans should not be considered design bugs or manufacturing defects as this generalization is simply not accurate. Recall that *a hardware Trojan is an intentional and malicious modification of a circuit that is designed to alter the circuit's behavior in order to accomplish a specific objective*. A design bug is an unintentional problem (i.e., error) that is unknowingly introduced into the circuit during its design and development phases. A manufacturing defect is an unintentional physical phenomenon (e.g., imperfection) that occurs during the circuit's fabrication, assembly, and testing phases. Both design bugs and manufacturing defects can cause flaws, failures, or faults in the final assembled IC or electronic system.

Although design bugs and manufacturing defects lead to incorrect results or unexpected behaviors, these outcomes are detectable during conventional test and validation methods. That is, bugs and defects are discoverable with functional or structural testing as well as pre-silicon or post-silicon verifications. Similar to Trojan taxonomies and attack models (discussed in Sect. 2.4), companies also use models to detect bugs and defects. These models are typically bounded by the specification of the design and will support testing and verification only within the intended design of the circuit. For example, stuck-at-one (SA1), stuck-at-zero (SA0), opens, shorts, and path delay faults are all detectable based upon specification models, and their specific activation locations are usually identifiable within the circuit.

Trojans are similar to bugs and defect in that they all can lead to unwanted functionality. However, Trojans deliver an unwanted functionality that is not bounded by the specification of the design. As a result, Trojans are often undetectable with standard testing and validation practices. Since conventional models typically do not check for any functionality outside of the defined specification, adversaries often attempt to hide Trojans in internal circuit nodes that are difficult to reach, control, and observe during testing [38]. Trojans can also be designed to activate after a rare and arbitrary set of complex conditions have occurred [6, 37]. Further details of Trojan detection strategies are explained in Sect. 2.5.

Figure 2.4 illustrates the comparisons between hardware Trojans, design bugs, and manufacturing defects.

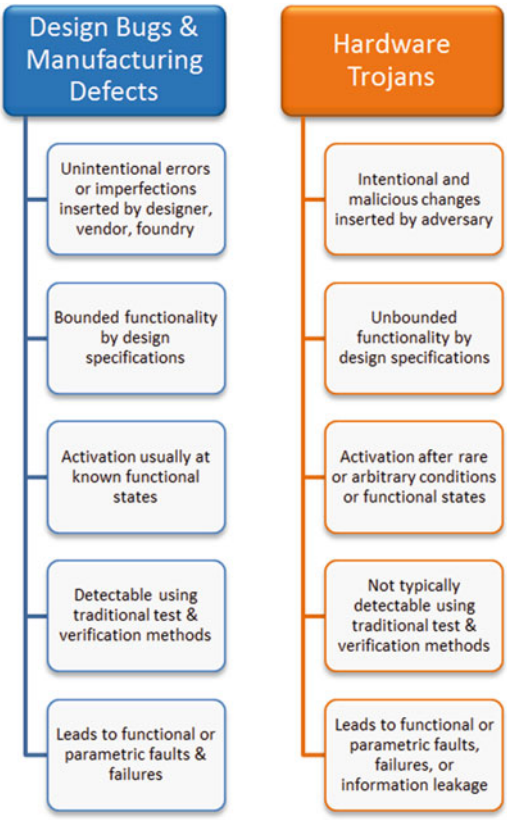


Fig. 2.4 Hardware Trojans compared with bugs and defects

2.3.2 Hardware Trojans Compared with Software Trojans

Trojans, whether software or hardware, share the same three key characteristics: malicious intention, evasion of detection, and rarity of activation. They also share a similar abstraction of two main components: a trigger and a payload. A software Trojan is commonly known as a computer program that contains specially crafted malicious code (i.e., payload) designed to cause harm to a targeted system once triggered. The malicious objective of software Trojans can include privilege escalation on the targeted system, leakage of sensitive user information including credentials and passwords, as well as data corruption, unauthorized encryption, and denial-of-service (DoS) attacks. Software Trojans are designed to remain stealthy, require specific events to occur for activation, and require special programs to detect and remove them. Most software Trojan detection programs include run-time monitoring, a concept that has been extended to hardware security (Sect. 2.5.1.2) [6].

Although the malicious intention of a hardware Trojan is similar to its nefarious counterpart, there are notable differences between these two Trojans. For example, software Trojans are hidden inside software code and are activated during program

Software Trojans	Hardware Trojans
Hidden inside software code and activates after specific conditions are met during program execution	Hidden inside physical hardware and activates after specific conditions are met during hardware operation
Spreads from user-to-user or adversary-to-user, through computer activities (e.g. file sharing or running infected programs)	Spreads from user-to-user or adversary-to-user, through physical insertion into circuits (e.g. infected ICs in supply-chain or untrusted entities)
Can be removed post-deployment (i.e. software updates)	Cannot be removed post-fabrication (i.e. no hardware updates)

Fig. 2.5 Hardware Trojans compared with software Trojans

execution, whereas hardware Trojans are hidden inside physical hardware and are activated after specific conditions occur during operation. Also, legitimate users of computer systems can unintentionally spread a software Trojan from user-to-user through routine activities, or adversaries can intentionally spread them to targeted or untargeted victims. This distribution can be accomplished in many ways such as through peer-to-peer file sharing or running infected programs from the Internet. Hardware Trojans, on the other hand, are typically spread from adversary-to-user since an IC is not easily replicated by the end-user. Another key difference is that hardware Trojans cannot be removed once the IC is fabricated, whereas the software Trojan can be removed post-deployment by way of local or remote updates to the program code [6].

Figure 2.5 provides a summary of comparisons between software and hardware Trojans.

2.3.3 Hardware Trojan Cause and Effect Misconceptions

The pressure on design companies to further reduce costs, decrease the time to market for product deliverables, and increase company profits has underpinned the semiconductor industry's shift toward horizontal and fabless business models. As a result, companies are relying more often on the acquisition and reuse of hardware third-party intellectual property (3PIP) as well as electronic design automation (EDA) and computer-aided design (CAD) software tools. This reliance is prevalent in the SoC industry where design specifications change many times during the design and manufacturing flow, and the hardware semiconductor companies must remain flexible to quickly respond to the market's demands. Unfortunately, these companies sometimes acquire 3PIP and design tools from untrusted entities (e.g.,

gray- or black-market vendors) without fully understanding the implications of their actions and the potential security risks to their SoC designs and to their customers.

Hardware Trojan attacks in hardware 3PIP are a serious security and trust risk that is difficult to mitigate. Companies and vendors that provide the trusted 3PIP (e.g., IP crypto cores) rarely make the golden model of their IP available to any outside entity. This results in a potential attack surface for adversaries: an untrusted entity (e.g., vendor, design house) can legally obtain a single version of the trusted 3PIP and insert a hardware Trojan into it, thus making it untrusted 3PIP. This infected 3PIP can be distributed to many naive SoC companies through various channels such as illegal file sharing services or other untrusted vendors. Since these SoC companies do not have the golden model that is required for conventional test and verification methods, they will have the virtually impossible challenge of verifying that the acquired 3PIP is secure and trustworthy [38, 47].

While it is true the SoC company can perform simulation of the 3PIP, this will only verify the functionality based on the design specifications; it will not guarantee the 3PIP is Trojan-free. The SoC company will not be able to compare their design, which is based on untrusted 3PIP, with the legitimate and trusted 3PIP. However, the work of [28] devised a technique to use the same hardware 3PIP acquired from several sources to reduce the risk and effects of a potential hardware Trojan. Also, the work of [47] developed a multiple-step methodology to identify and remove Trojans in 3PIP digital cores. Overall, the risk of using untrusted 3PIP still remains high as does its potential for delivering infected SoC products to customers and the supply chain.

Likewise, untrusted EDA tools present a similar risk of malicious modifications being inserted into the hardware design. Similarly to hardware 3PIP owners, EDA tool companies rarely make their golden model available to other companies. Untrusted EDA tools may be obtained through various methods such as illegal downloads, license key cracks, and untrusted vendors. It is a common practice for hardware design companies to use several tools from the same EDA tool suite, such as design automation, testing, and verification tools. Therefore, an adversary has several attack surfaces in which to maliciously modify the software tools to facilitate hardware Trojan insertion into a company's design.

Since hardware Trojans can be inserted into designs via the untrusted software EDA tools (e.g., through the hardware synthesis engine), detecting Trojans becomes more difficult as the design progresses. This results in the possibility of a hardware Trojan being inserted into a design early in the design flow via the untrusted EDA tool, which is later ignored or not detected by a different, albeit trusted, tool from the same vendor. The work of [26] uses the security paradigm of a completely specified design and low latency observability in order to design trustable hardware using untrusted software tools. However, again similar to untrusted 3PIP, using untrusted software EDA tools still runs the risk of delivering infected hardware products to customers and the supply chain [6].

Another popular misconception is that multiple untrusted entities result in improved security and trust. However, this allows for malicious collusion between multiple untrusted entities, also known as a multilevel attack, and results in a form

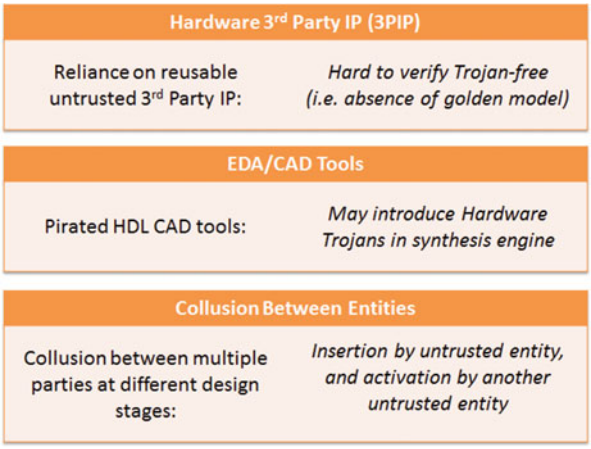


Fig. 2.6 Cause and effect misconceptions related to hardware Trojans

of risk that is difficult to defend against. Collusion can occur throughout different untrusted stages of the hardware design flow and life cycle [6]. For example, the work of [1] demonstrates a hardware Trojan inserted by an untrusted entity that is activated with a specific fault condition known only to the other colluding party. As another example, the work of [19] demonstrates a hardware Trojan designed to leak sensitive information through an analog side-channel. This Trojan is activated by another untrusted entity who also obtains and analyzes the leaked information. Collusion can also occur between multiple teams inside the same vendor. For example, the work of [27] illustrates this point and provides a codesign approach for preventing collusion between multiple rogue insiders at the design house. Although anti-collusion techniques do exist, the work of [1] demonstrates that multilevel collusion between untrusted entities results in a significantly stronger adversarial threat than what results from only a single adversary.

Figure 2.6 provides a summary of the cause and effect misconceptions discussed in this section.

2.4 Offensive Strategies

2.4.1 Taxonomy of Trojan Types

In order to properly model hardware Trojans for offensive and defensive postures, one must first understand the different types of Trojans. The categories of these types, referred to as taxonomies, represent the framework for classifying hardware Trojans based on their individual characteristics and also provide the foundation for building metrics to evaluate Trojan detection mechanisms. The first published hardware Trojan taxonomy was proposed in 2008 by [43] and consisted of six attributes including three principle categories based upon physical, activation, and

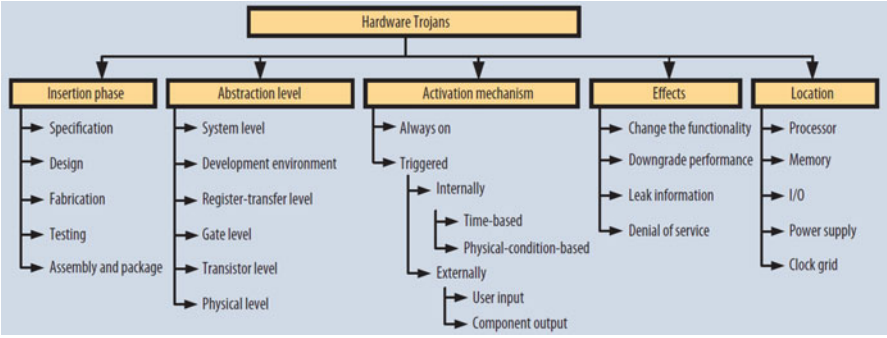


Fig. 2.7 Five comprehensive hardware Trojan taxonomy categories [17]

action characteristics. As hardware Trojans became more complex, this elemental taxonomy was improved to comprise of nine attributes for the same three principle categories [36]. The most comprehensive taxonomy to date is the work of [17] that comprises of five categories (insertion phase, abstraction level, activation mechanism, effects, and location), with each category containing multiple attributes. This taxonomy is predicated on two key criteria: (1) coverage (it should classify any-and-all Trojans) and (2) resolution (it should separate significantly different capabilities of Trojans). This comprehensive taxonomy is shown in Fig. 2.7.

The *insertion phase* describes the stages of the design and fabrication life cycle where the hardware is vulnerable to malicious modification. This phase ranges from defining the hardware characteristics (i.e., design specifications) to physical IC placement (i.e., assembly) on a printed circuit board (PCB).

The *abstraction level* describes the various development stages of the hardware IP prior to fabrication. This level spans from the physical dimensions and locations of the internal components in the circuit (i.e., physical level) to the final definitions of the interconnects and communication protocols used in the IC (i.e., system level).

The *activation mechanism* describes the means by which the Trojan is triggered. This includes always-on Trojans such as those continually leaking information through EM radiation, as well as Trojans requiring specific triggers for activation such as internal sequential counters or external triggers from input data streams.

The *effects* category describes the unwanted result from the Trojan's delivered payload. This ranges from introducing small errors that are difficult to detect (i.e., change the functionality) to full consumption or failure of hardware resources, thus preventing system availability (i.e., denial of service).

The *location* category describes where inside the hardware a Trojan can physically be inserted. This category spans from a single Trojan targeting a single component (e.g., system clock) to perform fault-injection attacks to multiple distributed Trojans targeting multiple complex components (e.g., processors) to alter the order of instruction execution.

This comprehensive Trojan taxonomy has been validated against a total of 56 hardware Trojans for the coverage and resolution criteria described in this section,

and it correctly captured all of the Trojans into the proper categories [17]. However, as with many aspects of security and trust, continuous advances are required to be made in order to stay ahead of the adversary. The website https://www.trust-hub.org maintains a collection of hardware Trojan benchmarks that have been developed and updated by researchers in the hardware security and trust community [32, 34, 41].

From 2007 through the present day, there has been much research focused on hardware Trojan modeling, circuit generation, and benchmarks [32, 34, 36, 41, 45]. However, within just the last few years, the number of Trojan design publications has started to trend downward, possibly indicating that Trojan designs have saturated. On the other hand, the number of research publications focused on countermeasures, such as detection and prevention, has greatly increased. These trend changes may be due to the fact that there exist a virtually unlimited number of different Trojan designs, and the critical research needs to be focused on the defensive postures with prevention mechanisms possibly outweighing detection mechanisms [45].

2.4.2 *Taxonomy of Trojan Triggers and Payloads*

The fundamental taxonomy of triggers and payloads, which are the two main components for hardware Trojans, is shown in Fig. 2.8. The trigger continually monitors specific signals in the circuit and activates when an expected event occurs, which is typically derived from the original circuit. The payload is activated by the trigger and delivers the malicious behavior to the circuit. Trojans may remain undetected and inactivated for many years while waiting for the specific circumstance to occur before they trigger and deliver their nefarious payload to the circuit.

Trojan triggers occur in two types: digital or analog. Digital is the most commonly researched Trojan as it consists of combinational and sequential circuits.

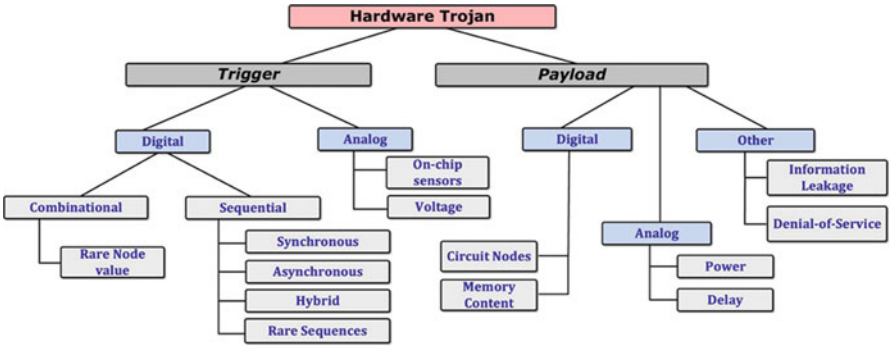


Fig. 2.8 Taxonomy of hardware Trojan triggers and payloads [6]

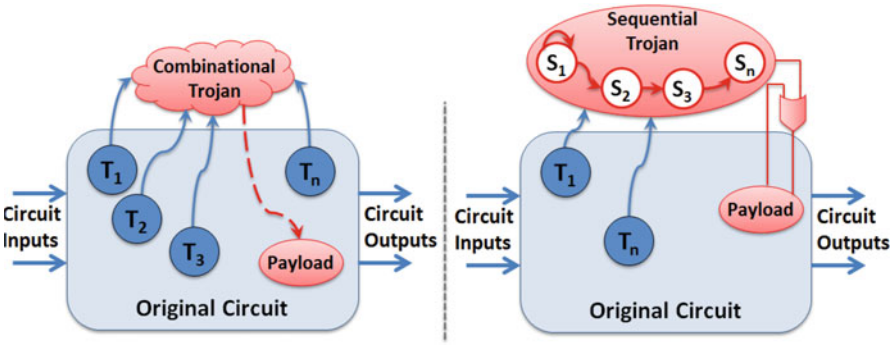


Fig. 2.9 Generic models of combinational and sequential Trojans

Combinational Trojans are stateless, meaning they contain no state elements (e.g., flip-flops, latches), and they rely on a specific condition occurring at a specific set of rare nodes within the circuit. Sequential Trojans are stateful, meaning they rely on a specific sequence of states to be traversed before activation (e.g., counters, finite state machines). Sequential Trojans are more difficult to detect since they require a number of arbitrary conditions to be met before activation, which can become computationally infeasible to detect with conventional testing methods. Analog triggers, on the other hand, rely on various natural phenomena (e.g., temperature, RF radiation, gate capacitance) for activation. A hybrid Trojan is a combination of a digital and an analog Trojan. Two generic Trojan models can be seen in Fig. 2.9.

Recall that Trojans should be virtually undetectable and rarely activated. Therefore, an adversary would choose nodes that are unlikely to activate during conventional testing methods. Upon activation, the Trojan's payload will be delivered to the circuit. The payload can be categorized as digital (e.g., affecting logic values, opening backdoors) or analog (e.g., affecting performance, EM emissions) or others (e.g., acceleration of IC aging, leaking information). The payload is the critical part of the Trojan as it is ultimately what modifies the original behavior of the circuit. Examples of fundamental hardware Trojans with triggers and payloads are shown in Sect. 2.4.3.

2.4.3 Fundamental Trojan Examples

Figure 2.10 shows a combinational Trojan with a NOR gate as a trigger and an XOR gate as the payload. This Trojan is activated only when the specific condition of $A = 0$ and $B = 0$ occurs at the NOR gate's trigger nodes, resulting in the payload delivering an inverted output *Cmodified*.

Figure 2.11 shows a synchronous sequential Trojan (a.k.a. Trojan "time bomb") with a simple counter for activation. The trigger consists of a $k - bit$ counter and an

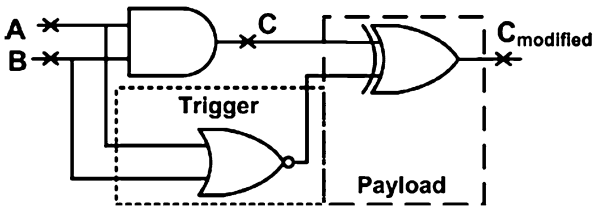


Fig. 2.10 Combinational Trojan circuit [9]

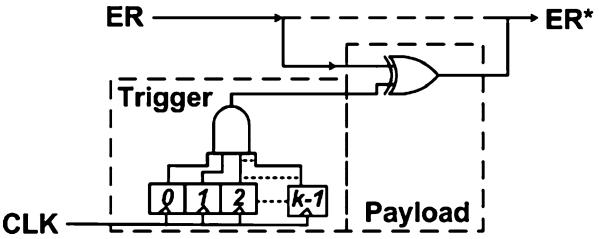


Fig. 2.11 Sequential Trojan circuit [9]

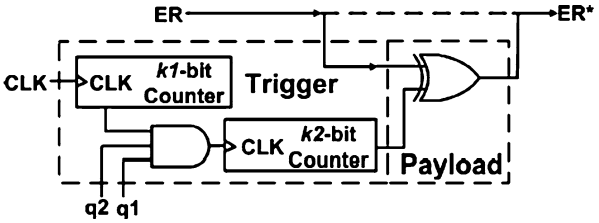


Fig. 2.12 Hybrid Trojan circuit [9]

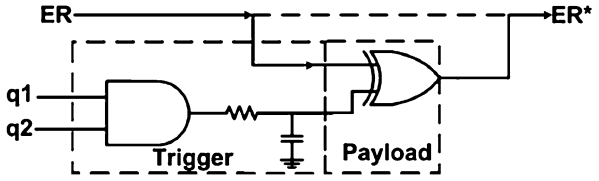


Fig. 2.13 Analog Trojan circuit [9]

AND gate, and the payload consists of an XOR gate. This Trojan is triggered after a predefined $2^k - 1$ counts, resulting in an inverted output *ER**. An asynchronous version of this Trojan can be created by substituting the clock (*CLK*) with another logic implementation.

Figure 2.12 shows a hybrid Trojan consisting of both synchronous ($k1$-bit) and asynchronous ($k2$-bit) counters. Both of these counters must reach their predetermined values for activation, resulting in an inverted output *ER**.

Figure 2.13 shows an analog Trojan in which a capacitor is charged if the AND gate output (driven by $q1$ and $q2$) has been set to 1 for a calculated period of time, resulting in an inverted output *ER**. If the AND gate output is not set high for the correct duration of time, the capacitor will discharge to ground and the Trojan will not activate.

The previous examples have shown hardware Trojans with digital or analog triggers delivering a digital payload. Recall that digital payloads are designed to

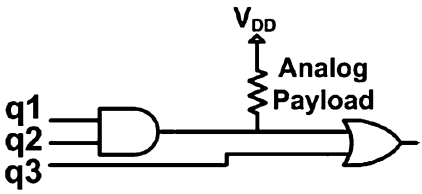


Fig. 2.14 Trojan with analog payload to Vdd [9]

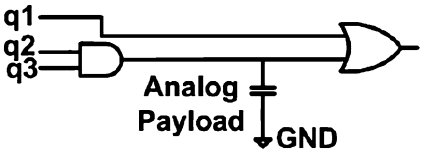


Fig. 2.15 Trojan with analog payload to GND [9]

affect targeted logic values at specific internal nodes, whereas analog payloads affect characteristics such as performance. Figures 2.14 and 2.15 offer examples of analog payloads.

Figure 2.14 shows an analog payload in which a fault is created, via the resistor to Vdd, when the output of the AND gate is logic zero.

Figure 2.15 shows an analog payload in which path delay is affected, via the capacitor to GND, when the output of the AND gate is logic one.

2.4.4 Innovative and New Trojan Attacks: Designs and Examples

Hardware Trojans have evolved to become more potent threats against secure and trustworthy hardware. Trojan attacks now include more advanced functions such as temperature-driven activation, radio, power, and optical side-channel leakage of information, accelerated aging of ICs, as well as denial-of-service (DoS) attacks against device availability [15, 18, 43]. This section explores several examples of innovative and recent designs of hardware Trojan attacks.

2.4.4.1 Side-Channel Trojan

Malicious off-chip leakage enabled by side-channels (MOLES) [19] demonstrates a hardware Trojan that is inserted by an untrusted foundry. MOLES leaks sensitive information outside of the IC through an analog side-channel, which is later acquired and analyzed by another untrusted entity. MOLES was designed to compromise hardware security modules (HSMs), which are the tamper-resistant cryptographic engines for embedded systems and general-purpose computers. It was implemented in an Advanced Encryption Standard (AES) IP core to leak

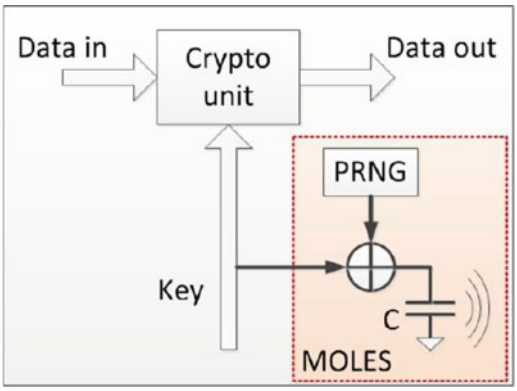


Fig. 2.16 MOLES circuit embedded inside a hardware security module [19]

multi-bit cryptographic keys. All of the bits were leaked with a signal-to-noise ratio (SNR) below the noise power level of the infected IC in order to remain hidden while the Trojan is activated. Also, MOLES was designed to be very small in size (i.e., fewer than 50 gates) in order to evade detection from automatic test pattern generation (ATPG) testing and layout inspection processes. Figure 2.16 shows the block diagram of MOLES.

2.4.4.2 Semiconductor Trojan

Stealthy dopant-level hardware Trojans are described in the work of [4] in which the dopant polarity was altered in the existing transistors of the infected IC. This type of Trojan is inserted by an untrusted foundry after placement and routing occurs during the layout level. Since there is no additional circuitry required for this Trojan, the appearance and functionality of the IC are not changed. Therefore, the malicious modification is virtually undetectable to optical inspection techniques and can defeat golden IC model verification methods. This Trojan was inserted into a case study model of a cryptographically secure processor in order to reduce the entropy of the random number generator (RNG) used to produce the cryptographic keys. The authors claim their Trojan will allow the compromised processor to still pass the built-in self-test (BIST) as well as the National Institute of Standards and Technology (NIST) test suite that is commonly used to rate the quality of random number generation. Figure 2.17a shows an unmodified inverter gate, and Fig. 2.17b shows a dopant Trojan inserted into the inverter gate resulting in a constant output of V_{DD}. A close inspection of this figure reveals the contact, metal, and polysilicon areas are indeed identical in both cases. The only change is to the polarities of the *N* and *P* dopants.

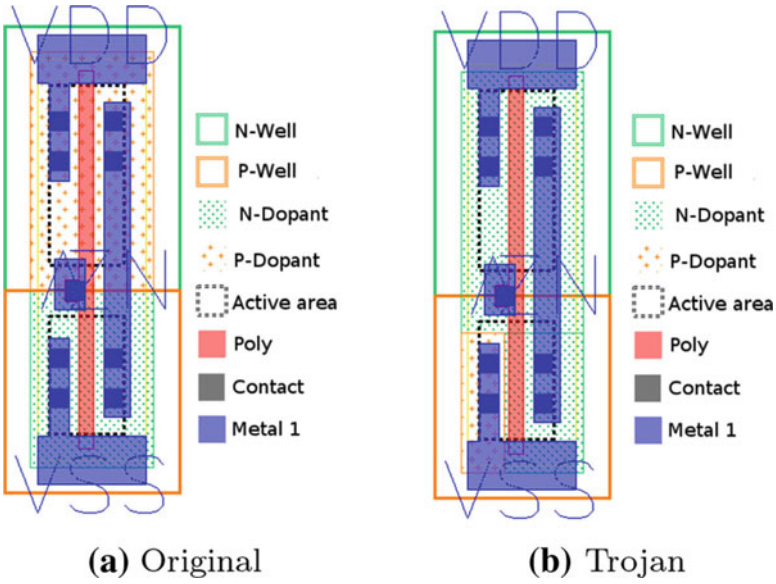


Fig. 2.17 Layout of (**a**) Trojan-free inverter gate and (**b**) inverter gate with Trojan to output constant V_{DD} [4]

2.4.4.3 Analog Trojan

Analog malicious hardware, called "A2", demonstrates how an untrusted foundry can insert an analog hardware Trojan into empty cells of a circuit [16]. A2 is a capacitor-based Trojan that slowly diverts charge from internal connections that rarely toggle their digital values. Once A2's capacitors reach a specified charge state, the Trojan triggers and delivers a payload that overrides a flip-flop's current state, thus forcing it to a predetermined value. Although A2 is a hardware Trojan, its objective is to enable a remotely controlled software privilege escalation attack by forcing a targeted bit in a security register to a specific value. A2 was implemented into an open-source CPU processor just prior to the CPU being fabricated, and the Trojan attacks were successful. Essentially, A2 implements an analog counter as a trigger, meaning that it does not require numerous additional gates as does a conventional digital counter-based trigger. A2 can be as small as one gate and is more stealthy than its digital counterpart; thus it is more elusive to functional verification, simulation, and side-channel Trojan detection methods. Figure 2.18 shows the behavior of the A2 Trojan, which takes multiple rising-edge triggers to charge the capacitor to the threshold voltage value required for activation.

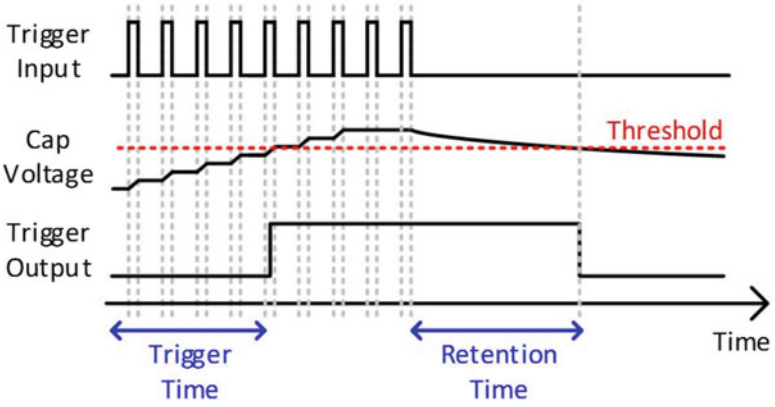


Fig. 2.18 "A2" analog Trojan circuit behavior. Notice capacitor requires multiple triggers prior to activation [16]

2.4.4.4 Digital Trojan

Hardware Trojans can also exist in digital finite state machines (FSMs) as demonstrated in [13]. Vulnerabilities exist in high-level incomplete design specifications, which can be exploited by adversaries during the design flow, as well as in post-fabrication through inadvertent trapdoors found in the defined FSM. When the number of protected states is not a power of 2 (as in n^2 states), there will be unused states in the FSM that will be treated as "*don't-care*" conditions. The adversary inserts the logic Trojan into the *don't-care* conditions inside a sequential FSM (i.e., where the next state or output is not specified), which allows an attacker access to protected states in the FSM once the Trojan is triggered. If a *don't-care* condition is not Trojanized, logic design tools may use it for optimization. However, at the circuit level, *don't-care* conditions will be assigned a deterministic next-state value by the EDA/CAD tools which may have been coerced by the Trojan. Since the Trojan is inserted early in the flow, it may pass undiscovered by Trojan detection mechanisms used later in the flow. Figure 2.19a shows a four-state transition graph, which was originally a three-state FSM with state "11" as a *don't-care*. The solid blue edge lines represent the defined state transitions, and the dashed orange lines represent the *don't-cares* in the FSM. The dashed orange edge lines will be implemented by the circuit (Fig. 2.19b) with their assignments allowing an attacker potential access to protected states in the FSM.

2.4.4.5 Other Notable Trojans

2.4.4.5a: Insertion of hardware Trojans into the silicon of a fabricated wireless cryptographic IC was accomplished in the work of [20]. These Trojans were inserted

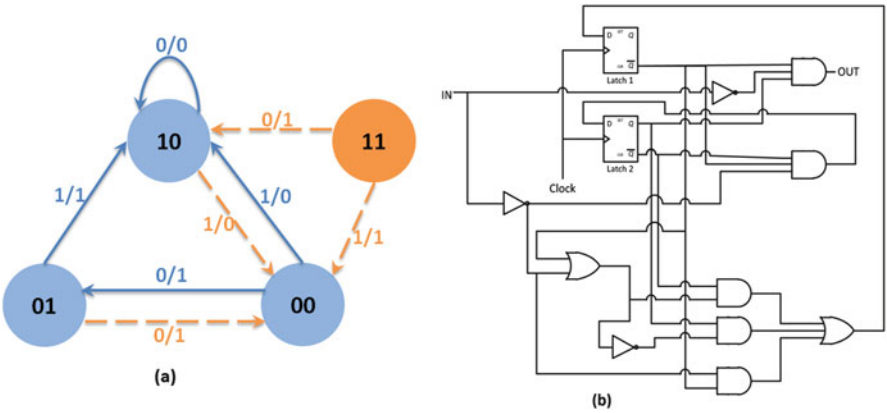


Fig. 2.19 Vulnerable four-state FSM (originally three states shown in blue) showing (**a**) dashed orange edge lines and state "11" representing *don't-care* conditions as a result of the digital logic implementation (**b**) where the *don't-cares* may allow an attacker access to protected FSM states [13]

into the AES core and the ultra-wideband (UWB) transmitter of the application-specific integrated circuit (ASIC) chipset. This attack resulted in encryption key leakage that was concealed within the amplitude and frequency design margins allowed due to fabrication process variations.

2.4.4.5b: A software-exploitable hardware Trojan was implemented inside an embedded processor as demonstrated by [44]. This hardware Trojan was modeled by a sequential FSM and was triggered with a specific combination of firmware instructions and data processing sequences. The attack resulted in leakage of the program IP, the encryption key, and also caused the system to malfunction.

2.4.4.5c: Reliability Trojans, which are malicious alterations of manufacturing conditions during semiconductor fabrication, are introduced in [35]. These Trojans exploit the wearing-out mechanisms for CMOS transistors, such as negative bias temperature instability (NBTI) and hot carrier injection (HCI). The attack objectives include reduced reliability, accelerated aging, and premature failure of ICs such as SRAM cache memory, all of which are triggered only with time and usage of the IC.

2.4.4.5d: Small, optimized, and performance-based Trojans have been designed to evade detection in the works of [7, 42]. These low-impact Trojans rely on modifications such as resizing of logic gates, interconnect tampering, and insertion of resistive bridging faults at single failure points in a circuit. They are designed to be inserted without impacting path delays, power consumption, or area overhead of the circuit. Their attack objectives include privilege escalation, erratic behavior, incorrect outputs, and hardware failure.

2.4.4.5e: Field programmable gate arrays (FPGAs) and systems on chip (SoC) are also vulnerable to hardware Trojan attacks. The work of [33] developed a com-

prehensive taxonomy of Trojan attacks in FPGAs. The work of [11] demonstrates an automated security analysis framework designed to detect hardware exploits including Trojan attacks in SoCs. Also, the work of [34] demonstrates an extensive database of hardware Trojan attacks which can be used to be insert Trojans into FPGAs and SoCs for research of defensive security and trust techniques.

2.4.5 *Trojan Attack Models*

The proper modeling of hardware Trojans is essential to accurately categorizing the adversarial threats and analyzing the effects of Trojan attacks. Prior to developing an attack or countermeasure method, the adversary or defender must consider the appropriate Trojan model. Recall that adversaries can insert Trojans into hardware during many phases of the design flow, which leads to the need for multiple attack models. Accurate Trojan attack models must be based on the entire semiconductor supply chain. For SoCs, this can be divided into three phases: IP core development, SoC development, and fabrication. These three phases result in three types of entities that can potentially attack the hardware design: 3PIP vendors, SoC developers, and fabrication foundries. This adversarial threat modeling concept can be extended to other ICs, and existing research has been performed to categorize different models of Trojans attacks [17, 29, 36]. The work of [45] provides us with seven comprehensive attack models for SoCs, shown in Fig. 2.20.

A summary of the seven comprehensive Trojan attack models follows:

Model A: Untrusted 3PIP Vendor—Most SoC designers have to acquire some forms of third-party IP (3PIP) cores to complete their designs. This is driven by demands for reduced costs, faster time to market, and decreases in physical IC size coupled with increased functional complexity. Adversaries at the untrusted vendor can insert hardware Trojans into the 3PIP without the SoC designer's knowledge.

Model	Description	3PIP Vendor	SoC Developer	Foundry
A	Untrusted 3PIP vendor	Untrusted	Trusted	Trusted
B	Untrusted foundry	Trusted	Trusted	Untrusted
C	Untrusted EDA tool, or rogue employee	Trusted	Untrusted	Trusted
D	Commercial off-the-shelf (COTS component	Untrusted	Untrusted	Untrusted
E	Untrusted design house	Untrusted	Untrusted	Trusted
F	Fabless SoC design house	Untrusted	Trusted	Untrusted
G	Untrusted SoC developer with trusted IPs	Trusted	Untrusted	Untrusted

Fig. 2.20 Seven comprehensive hardware Trojan attack models [45]

Model B: Untrusted Foundry—Most design houses are partially or fully fabless, meaning they outsource the fabrication of their ICs to offshore and untrusted entities. The outsourcing decisions are a tradeoff between the need for the latest fabrication technologies at the lowest costs and the security of their designs. Adversaries at these untrusted foundries have access to all layers of the design and are able to insert Trojans into any of the lithography masks, as well as reverse engineer the design for counterfeiting purposes.

Model C: Untrusted SoC Developer—Highly trained SoC designers and specialized design tools are required to produce complex hardware designs. Adversaries in this model are insider threats, and they may use untrusted (i.e., pirated) CAD and EDA software tools.

Model D: Untrusted COTS Components—Many commercial off-the-shelf (COTS) components are used in designs. COTS items are less expensive than a custom product, and they typically do not require custom development for integration into a system. These items are developed in a completely untrusted manner resulting in multiple vulnerable stages in the design flow.

Model E: Untrusted Design House—Designs in this model are fabricated in a trusted foundry, but the design house and 3PIP vendors are not trusted to produce Trojan-free designs. Outside of the trusted foundry, the entire supply chain is untrusted.

Model F: Untrusted Outsourcer—This is a combination of Model A and Model B, and it applies to almost all fabless IC design houses. These designers use 3PIP vendors and untrusted foundries resulting in the inability to guarantee a Trojan-free hardware design.

Model G: Untrusted Systems Integrator—This model includes untrusted system integrators catering to a variety of customers who wish to have a supplier capable of both design and fabrication. This developer can pull from a variety of resources to meet customer demands; however vulnerabilities may be introduced into the completed hardware design.

The work of [45] also included a comprehensive analysis of 161 published papers on countermeasures against these types of Trojan attack models. The results show a remarkable 89% of the countermeasure papers covered Model F (untrusted outsourcer), and almost 60% of the published papers covered Model B (untrusted foundry). Model A (untrusted 3PIP vendor) was covered in almost 30%, and Model C (untrusted SoC developer) was covered in 13% of these papers. It is worth noting that Models D, E, and G accounted for virtually 0% of the papers as these three models can be absorbed by other attack models.

It is widely known that Trojan attacks typically occur in untrusted entities. Consequently, countermeasures should be performed only by trusted entities. These seven attack models show that foundries, vendors, and designers all play the role of either untrusted or trusted entities, but they cannot be both roles at the same time.

2.5 Defensive Strategies

2.5.1 Taxonomy of Trojan Countermeasures

It is very difficult to detect hardware Trojans using conventional test and validation processes, as introduced in Sect. 2.3.1 and analyzed in Sect. 2.5.5. Whether performing pre-silicon or post-silicon verification or executing structural, functional, or random test patterns, these conventional approaches perform poorly for detecting hardware Trojans. These processes are designed to detect defects in manufacturing workmanship and typically only test for expected operating conditions within the circuit. Trojans, by nature, are hidden among rare internal nodes that are not normally activated during device testing. An adversary can choose from an extremely large selection of Trojans to insert into the circuit, whereas performing deterministic and exhaustive defensive testing for all conditions is computationally infeasible. Additionally, parametric parameters such as path delay, internal noise, and power consumption are different for each IC based upon manufacturing tolerances, thus making detection inherently more challenging [3, 6, 9, 36, 43, 45, 48]. Therefore, the need exists for both detection and prevention mechanisms to defend against hardware Trojans.

Hardware security and trust researchers have developed three broad categories of countermeasures for hardware Trojans and have proposed the taxonomy shown in Fig. 2.21. The three main categories of countermeasures are Trojan detection, design for trust, and split manufacturing for trust [45]. The single letters in the blocks of this taxonomy cross-reference the listed countermeasure in this taxonomy with its particular attack models that were discussed in Sect. 2.4.5.

2.5.1.1 Trojan Detection

The goal of Trojan detection is to verify hardware designs without requiring supplemental circuitry [45]. Additional circuitry would lead to an increase in manufacturing cost, circuit size and performance, and power overhead. Trojan detection is performed during pre-silicon and post-silicon design stages. *Pre-silicon verification* helps SoC designers to validate 3PIP in the final design stages prior to fabrication. This includes performing functional validation, structural and code analysis, and formal verification. Although these techniques are good for identifying unintentional design errors and manufacturing faults, they offer no guarantees against hardware Trojans. *Post-silicon verification* provides more guarantees against Trojans, but at a different cost, which can be further categorized into destructive and nondestructive methods [9, 45].

Destructive methods (e.g., depackaging of ICs, physical reverse engineering) offer the highest assurance against Trojans as the fabricated IC can be visually verified against a golden IC or golden model. It is a complicated process involving

layer-by-layer de-metallization using a chemical mechanical polishing (CMP) technique followed by image reconstruction and analysis using a scanning electron microscope (SEM). This method allows for the identification of individual gates, transistors, and routing elements contained within the IC. This process is performed on a one-by-one basis taking several weeks to conduct, and the IC is destroyed during this process. Additionally, an adversary can insert a Trojan into only a small number of ICs instead of the entire lot, thus making destructive detection methods impractical from a scalability perspective. However, destructively testing a limited number of ICs is still beneficial as the information gained from the samples may be used to form golden models for verification with other Trojan detection methods (e.g., side-channel analysis) [6, 9, 45].

Nondestructive methods include functional testing and side-channel analysis to identify possible hardware Trojans. *Functional testing*, also known as *logic testing*, is a method of inputting specific logic patterns into the IC with the goal of triggering any existing Trojans. This form of testing is not equivalent to conventional testing as those test vectors aim to identify bugs and defects. Trojans are typically hidden in low-controllable and low-observable nodes, thus making them difficult to reach with conventional testing methods. *Side-channel analysis* is a method of acquiring and analyzing characteristics that are unique to each IC. The goal is to identify additional circuitry containing Trojans through observing changes in the physical parameters of each IC. This method includes performing analysis on consumed and leaked power, gate timing, path delay, circuit temperature, and electromagnetic radiation. Side-channel analysis may use ring oscillators, shadow registers, and delay elements to detect fluctuations indicative of hardware Trojans. Nondestructive methods can be performed numerous times and on as many ICs as desired. Although functional testing and side-channel analysis typically require a golden IC or model, together they form a complementary approach toward nondestructive hardware Trojan detection [6, 9, 36, 43, 45].

Functional validation is a form of pre-silicon Trojan detection with the main idea being similar to functional testing (i.e., logic testing). Functional validation is performed using modeling and simulation, which requires no physical connections to the device under test (DUT). Functional testing, on the other hand, requires physical connection to the DUT and is usually performed on a specialized test stand. The test stand is required to apply all of the generated test input vectors (based on the design specifications) and to collect the output of the device. Functional testing methodologies can be applied to functional verification, but not vice versa [45, 47].

Formal verification is a pre-silicon or pre-synthesis design verification technique, which is typically performed to confirm that a circuit has been indeed designed in the precise manner as defined by the design requirements. In the context of security and trust, formal verification is a mathematical approach toward exhaustively validating the entire security and trust specification of an IC. It includes using a specified set of security policies and proof-checking methods. Formal verification for Trojan detection is based on three verification methods: property checking, equivalence checking, and model checking. Respectively, these methods allow for verification of requirements in hardware test bench properties, checking equivalence between

RTL, netlist, and GDSII files, as well as checking the models used for system-level languages (e.g., Verilog and VHDL) against the defined security specifications [45, 47].

Code and circuit coverage is another pre-silicon Trojan detection technique. Analysis of hardware description language (HDL) may be performed structurally or behaviorally to identify rare and redundant internal nodes within the HDL code and the circuit. The coverage typically includes RTL line execution, FSM reachability coverage, and coverage of gate-level netlists combined with functional assertions indicating success or failure. This analysis aims to locate and identify the nodes with the highest probability indicative of a location for Trojan insertion. Quantitative metrics and manual post-processing of the results may be used to identify unusually rare nodes or gates with low observability, low reachability, and low probability, all of which are ideal targets for adversaries to insert hardware Trojans [45].

2.5.1.2 Design for Trust (DfT)

An alternative to the Trojan detection methods described above is to design for trust (DfT). DfT is a method that integrates security and trust throughout the entire design and manufacturing flow. It includes facilitating detection, preventing Trojan insertion, and trustworthy computing on untrusted components as shown in Fig. 2.21.

The first approach to DfT is to *facilitate detection* by incorporating functional testing, side-channel analysis (both previously described), and run-time monitoring.

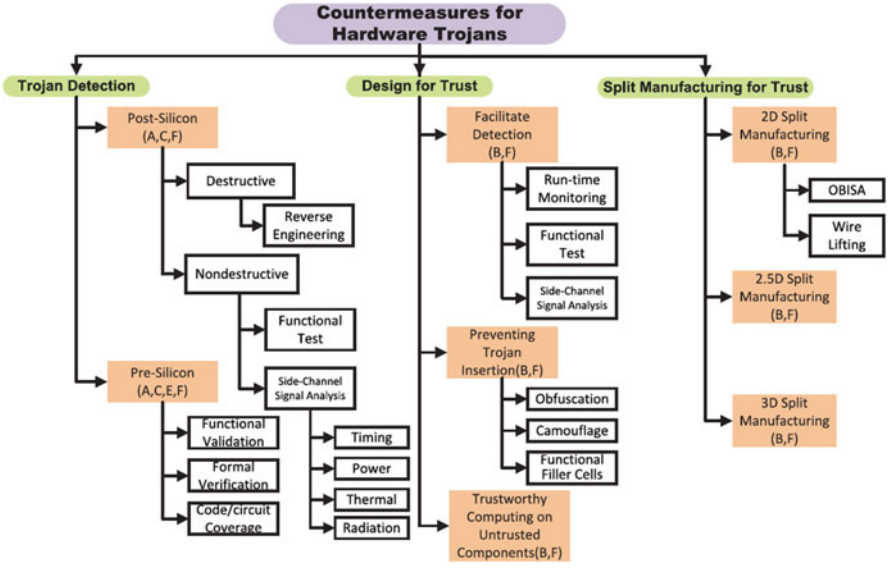


Fig. 2.21 Taxonomy of Trojan countermeasures [45]

Run-time monitoring can increase the trustworthiness of hardware by continually monitoring all critical computations for abnormalities, thus reducing the effect of Trojan attacks. Run-time monitoring can detect malicious behavior and automatically disable or bypass the malicious logic allowing the IC to restore reliable operation. It uses existing on-chip and supplemental off-chip (e.g., online) resources to continually monitor characteristics of the IC including behavior, operating conditions, transient power, and temperature [6, 9, 24, 36, 45].

Preventing Trojan insertion is another approach to DfT and includes obfuscation, camouflaging, and functional filler cells. *Obfuscation* is a method of hiding circuit functionality by inserting additional logic-locking circuitry into the design in order to conceal the correct functionality and the intended hardware design. The obfuscated circuit regains full functionality when the correct logic key is applied to the inputs. Obfuscation is effective for Trojan prevention in combinational, sequential, and reconfigurable logic. *Camouflaging* is a method of creating indistinguishable layouts of gates by using additional dummy contacts and fake interconnects between layers of the different gates in the circuit. Camouflaging prevents the attacker's ability to reverse engineer the circuit's netlist, thus preventing Trojan insertion. *Functional filler cells* are a method of inserting functioning gates into empty spaces in the hardware design. Typically, EDA/CAD tools fill empty spaces with non-functioning standard cells, thus allowing an attacker to replace these unused cells with a Trojan. Functional filler cells make use of all empty locations by inserting functional gates to form specified combinational logic, which can be tested during the design flow. A failure in the functionality of the filler cells can be indicative of an inserted Trojan [6, 45].

Another method for DfT is by performing *trustworthy computing on untrusted components*. This method is inherently resilient to Trojan attacks, which separates it from other prevention methods such as run-time monitoring and obfuscation. This method mitigates the effects of activated Trojans as the trusted computing software, which is running on the untrusted hardware, can be distributed over multiple independent mechanisms and processes. These risk reduction methods include distributed software scheduling over multiple multicore processors, using the identical untrusted 3PIP from different untrusted vendors, and comparing multiple 3PIP sources with similar untrusted designs [45].

An important point for hardware designers to consider is the balance between hardware Trojan detection and prevention methods with the actual need for increased security and trust of the hardware design. For Trojan detection, as the size of a circuit increases, so does the number of additional gates and internal nodes. These additional gates may unintentionally introduce low-controllable and low-observable nodes, which makes Trojan detection even more challenging. In addition, intentional modifications of gates for Trojan prevention may negatively impact the performance of the circuit. These impacts occur in many forms including path delays, power consumption, and area overhead of the circuit. As with design specifications, this inevitable tradeoff must be considered throughout the entire design flow and manufacturing process.

2.5.1.3 Split Manufacturing for Trust

Recently, split manufacturing has been offered to protect against hardware Trojans. This manufacturing process divides the hardware design into front-end-of-line (FEOL) and back-end-of-line (BEOL) sections that are fabricated by different foundries. Typically, an untrusted foundry will fabricate only the FEOL portion of the design and then ship their wafer sections to a trusted foundry who fabricates the BEOL section and integrates both sections. The process is conducted in this fashion as FEOL fabrication is of higher cost (i.e., monies, machinery, time) than BEOL fabrication. Split manufacturing prevents the untrusted foundry from having access to all layers of the IC, since having this information would allow an adversary to easily insert Trojans.

Common techniques of split manufacturing include *2D integration* (described above), 2.5D integration, and 3D integration as shown in Fig. 2.21. In *2.5D integration*, the design is divided into two sections (FEOL and BEOL), which are both fabricated by the untrusted foundry. A middle section, called a silicon interposer as it contains the interchip connections, is fabricated at a trusted foundry. The interposer, FEOL, and BEOL sections undergo final assembly in the trusted facility. In *3D integration*, both the FEOL and BEOL sections are fabricated by different foundries. The 3D assembly consists of vertically stacking the sections and inserting vertical interconnects called through-silicon vias (TSVs). Naturally with any Trojan countermeasure technique, there are tradeoffs with split manufacturing including higher manufacturing costs, increased area due to the interconnections, increased timing and power overheads, and higher temperatures in the middle tiers of the IC [6, 45, 46].

2.5.2 Detection of Trojans: Examples

2.5.2.1 Statistical Detection

Multiple excitation of rare occurrence (MERO) is a statistical form of Trojan detection [10]. MERO aims to maximize the probability of triggering Trojans during logic testing, while minimizing the number of test vectors required as compared with a weighted random pattern testing approach. It works by first detecting low-probability events at internal nodes and then creating a set of optimized test vectors that triggers each internal nodes to their rare logic value multiple times (e.g., $N > 1000$). It applies these vectors during testing in an effort to trigger any Trojans in the circuit. Recall that conventional testing does not scale to detect Trojans due to the exponential number of possible Trojan instances. Figure 2.22a shows the rare event ($abc = 011$) required to trigger a combinational Trojan, and Fig. 2.22b shows the rare occurrence ($ab = 10$) required to trigger a sequential Trojan. These two conditions will be identified by MERO and toggled many times in order to trigger and detect any Trojans in the circuits.

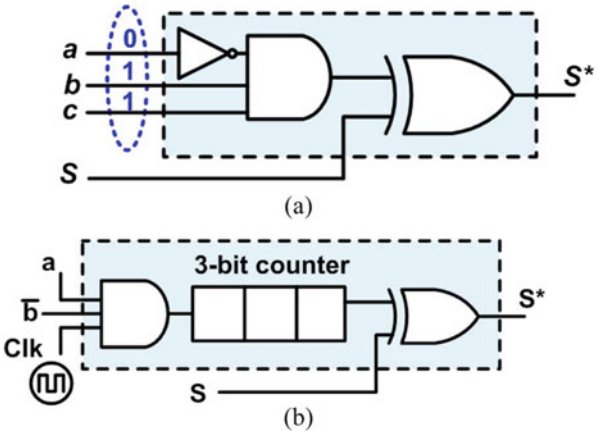


Fig. 2.22 Statistical logic testing of MERO for (**a**) combinational Trojan and (**b**) sequential Trojan [10]

2.5.2.2 Rare Event Removal

A methodology for identifying and removing rare nodes to increase Trojan detection is demonstrated in [31]. It is widely known that adversaries target low-controllable and low-observable nodes in which to insert Trojans, thus leading to challenges with Trojan detection methods. This technique analyzes the hardware design to identify nets with a transition probability less than a specific threshold value, which makes them ideal candidates for Trojans. The transition probability threshold value is modeled using a geometric distribution (GD) and estimates the number of clock cycles required for a node to transition. One or more dummy scan flip-flops (dSFF) are then inserted into the identified nodes to increase their probability of transitioning, all without the flip-flops affecting the timing or functionality of the design. The increased transitioning rate will result in a decrease of hard-to-activate nodes in the circuit (i.e., removal of the rare events). Removing these rare events is what allows for facilitating Trojan detection in the hardware, even if the gate-level netlist is not trusted. Small Trojans may be fully activated resulting in detection (via logic testing) through malfunctions and faulty circuit outputs. Large Trojans may be partially activated resulting in detection (via side-channel analysis) through measurable changes in signals such as transient power and path delay.

Figure 2.23a shows an original Trojan cone (i.e., logic gates connected to the input of a Trojan gate) consisting of all *AND* gates with the probability of generating a "1" ($1/256 = 0.0039$) at the Trojan gate ($T_g i$) being much less than generating a "0" ($255/256 = 0.9961$). Figure 2.23b shows a single dummy scan flip-flop *OR* gate inserted into the top net of the circuit. This single gate dramatically reduces the number of clock cycles required to transition the Trojan gate (T_{gi}), thereby increasing the transition probability of a "1" to be much greater than its original value ("1," $17/512 = 0.0332$; "0," $495/512 = 0.9668$). This increase in transition rate leads to an increased probability of Trojan detection.

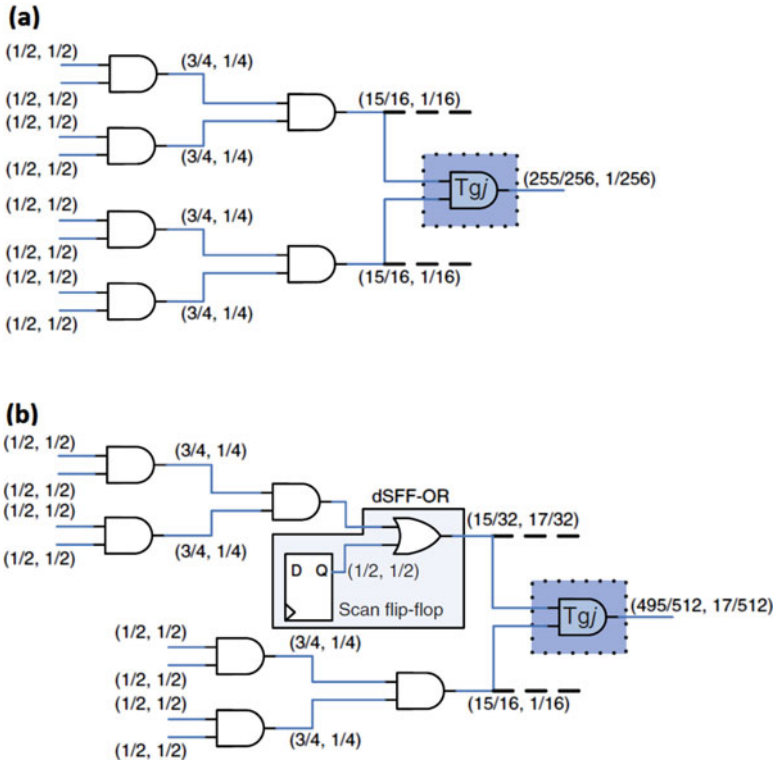


Fig. 2.23 Transition probability of the (**a**) original Trojan cone and (**b**) inclusion of a single dummy scan flip-flop [31]

2.5.3 Prevention of Trojans: Examples

2.5.3.1 Obfuscation

Key-based design obfuscation deters adversaries from inserting Trojans by transforming the circuit into another circuit that is functionally equivalent, but with the added benefits of security features to prevent hardware Trojans [8]. It allows a circuit to operate in two different modes (i.e., obfuscated mode and normal mode) and requires a sequential logic key to unlock correct circuit functionality. In the obfuscated mode (i.e., protected mode), the correct functionality and structural design of the circuit is obscured, resulting in incorrect behaviors being produced by the obfuscated circuit. In the normal mode, the correct behavior and functionality of the circuit is restored, although the design itself is still obscured from an adversary.

Transitioning from obfuscated mode to normal mode requires the correct logic key to be applied at the inputs during initial start-up of the circuit; otherwise the circuit will remain in its protected mode. This technique makes inserting

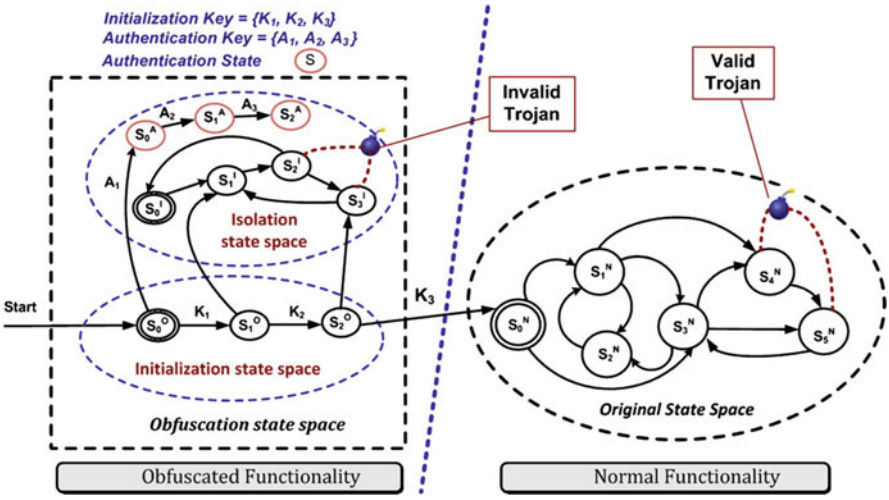


Fig. 2.24 Key-based obfuscation scheme for Trojan prevention and detection [8]

Trojans difficult for adversaries as the rare internal nodes are obfuscated (recall that adversaries insert Trojans into rare circuit nodes to prevent detection). If a Trojan is inserted, it is highly likely that it will be inserted into an isolated region, thus negating any effect of a triggered Trojan. This approach results in an increased prevention against hard-to-detect Trojans from being inserted into rare nodes, increased Trojan detection, as well as protection against IP piracy (i.e., IP theft via reverse engineering). It can prevent Trojans from being inserted by an adversary at an untrusted foundry or by untrusted EDA/CAD tools in IC design flows, albeit at a cost of higher overhead and the inability to prevent random Trojan insertions.

Figure 2.24 shows a state transition graph (STG) of a circuit. Upon power-up, the circuit starts in the obfuscated mode (i.e., state S_0^O). Only the correct sequential logic key (i.e., $K_1 \rightarrow K_2 \rightarrow K_3$) will transition the circuit into the normal mode (i.e., state S_0^N). Any incorrect key will result in the circuit transitioning into the isolation state space, where it will be trapped along with any invalid Trojans.

2.5.3.2 Hardware 3PIP

A design framework for a formal, yet computationally feasible, acquisition of trustworthy hardware 3PIP is demonstrated in the work of [21, 37]. This framework is called proof-carrying hardware intellectual property (PCHIP), and it is based on the concept of proof-carrying code (PCC). PCHIP focuses on the security and trust of the IP in the form of hardware description language (HDL) for FPGAs. Unlike many other approaches, it does not require a golden IC or model, nor does it require

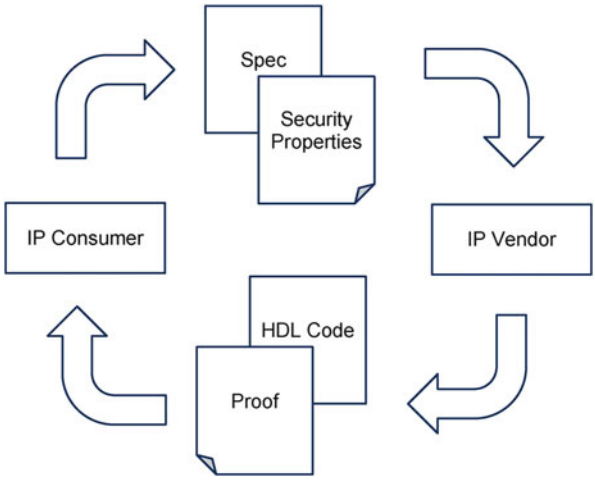


Fig. 2.25 Hardware intellectual property (IP) acquisition and delivery framework protocol [21]

a trusted 3PIP vendor. The IP consumer establishes a set of upfront security and trust properties with the IP vendor, which become integrated as a component in the entire design flow. These properties are temporal logic, meaning they are rule-based symbolic expressions instead of exact hardware functionality. The IP vendor creates a formal proof of these properties that is delivered with the 3PIP to the IP consumer. The IP consumer then validates the 3PIP against the agreed-upon security specifications. If the 3PIP fails verification, it means a security protocol has likely been violated, thus resulting in the prevention of malicious Trojans from being inserted into the IP consumer's FPGA design. Since an inadequate set of security specifications may have unknown vulnerabilities, this framework does not ensure complete coverage and should be used only in conjunction with other detection and prevention methods. Figure 2.25 shows the interaction cycle between the IP consumer and the IP Vendor.

2.5.4 Other Notable Trojan Detection and Prevention Methods

Similarly to the evolution of hardware Trojan designs attacks, Trojan detection and prevention mechanisms have also evolved in much the same manner. This section will introduce other notable and new Trojan detection and prevention techniques.

2.5.4.1 Voltage Inversion

A voltage inversion technique to ascertain malicious insertions (VITAMIN) in ICs was proposed by [2]. This technique utilizes an inverted voltage scheme to complement the voltage level of CMOS gates (e.g., changing an *AND* gate into a

NAND gate). It aims to detect the presence of hardware Trojans through enhancing the differences in the power profiles between a golden IC and a test IC, resulting in higher triggering frequencies for the rare nodes in the circuit. This approach was combined with a sustained vector technique to further strengthen its effectiveness.

2.5.4.2 Temperature Tracking

A run-time method for detecting hardware Trojans was introduced in [14]. The goal of this framework is to detect a deviation in the normal correlation between the thermal and power profiles within an IC. Abnormalities in this profile may indicate a Trojan activation. This approach consists of design-time, test-time, and run-time monitoring phases. It is low overhead as it exploits the existing internal thermal sensors on many FPGAs, SoCs, and other ICs. This approach enables online Trojan detection during the entire lifetime of the IC.

2.5.4.3 Split Fabrication

A secure split manufacturing methodology using vertical slit field effect transistors (VeSFET) inside of ICs is proposed by [46]. This approach uses the VeSFET's two-sided accessibility and 3D integration capability to hide transistors in a camouflaged 3D case. It allows two independent untrusted foundries to securely fabricate 2D and 3D ICs. If one foundry adds or moves transistors, the resulting crowbar current effect is detected by the other foundry. This design methodology prevents hardware Trojan insertion, reverse engineering, and IP piracy, as well as provides methods for Trojan detection capabilities.

2.5.4.4 FPGA Trust

Hardware Trojan attack prevention and detection methods for FPGAs are demonstrated in the works of [22]. This security and trust validation method, called adapted triple modular redundancy (ATMR), enables protection against Trojans that are inserted during device production at the foundry. It is adaptive since Trojans can be independent of the final design. ATMR is a combined approach consisting of logic testing and side-channel analysis, and it is designed to protect FPGAs from Trojans of various sizes, locations, and functionalities. This work also developed a Trojan attacks taxonomy and Trojan models that specifically target FPGAs. The taxonomy covers Trojans that alter the programmed state and I/O blocks of the FPGA, including Trojans inserted by the foundry that cause logical malfunctions and physical damage.

2.5.5 Comparisons of Various Trojan Defensive Methodologies

Detection and prevention of hardware Trojans attempt to solve the problem from two unique viewpoints. Many detection methods have been researched and the consensus is that detecting a small, quiet Trojan is very challenging, and the majority of the methods for detection are based on the use of a golden IC or golden model [6, 45]. Although the authors of [45] state that prevention may be better than detection, it is worth considering that a balance of the two methods is perhaps the best approach toward Trojan-free hardware.

Automatic test pattern generation (ATPG) is a method of circuit testing that is used to distinguish between the correct behavior of the design and the faulty behavior caused by defects. The goal of ATPG is 100% test coverage, and it is based upon the specification of the circuit. Structural testing may detect some Trojans since the number of nodes to test grows linearly with the number of inputs to the circuit. However, functional testing is much more complex as the number of nodes to test grows exponentially with the number of inputs to the circuit, thus making it almost impossible to detect all possible Trojans. Formal verification methods are algorithmic-based approaches that validate properties of the intended design. Several proposed methods for Trojan detection, including formal verification, identification and removal of suspicious signals, and equivalence checking, are detailed in [38, 43, 47].

Functional and structural testing, in the forms of logic testing, aim to activate unknown Trojans during the validation process and to propagate their effects to an observable output node [6]. Logic testing is a straightforward approach that works well for detecting ultrasmall Trojans. It is also robust in the presence of environmental noise and manufacturing variations. However, it is not scalable and quickly fails as the circuit or Trojan increases in size or complexity. The reason for the lack of scalability is difficulty in generating a complete set of test vectors due to the exponential increase in input combinations required to test all internal nodes and generate all outputs. Essentially, logic testing has difficulties exciting rare, low-controllable, and low-observable nodes. Also, logic testing cannot trigger externally activated Trojans, and it requires a golden IC or model for Trojan detection [6, 45].

Side-channel power analysis is a detection method predicated on the expected, albeit unknown magnitude, increase or decrease in the power overhead due to the inclusion of a Trojan. Common methods include static current analysis and transient current analysis. Static current analysis is a method of detection that relies on the inactivated Trojan causing noticeable change in current draw from the power supply due to the addition or modification of gates. Transient current analysis, on the other hand, allows for the detection of switching activity inside an activated Trojan [5, 6, 9, 45].

Side-channel analysis scales very well to detect large Trojans in both small and large circuits. It is effective for Trojans that do not cause observable malfunctions (e.g., data leakage), and it is easy to generate test vectors for Trojan detection.

However, this form of testing performs poorly in the presence of environmental noise and manufacturing variations and does not work well for detecting ultrasmall Trojans. This limitation is due to the Trojan's effects being masked by the noise in the IC. Side-channel detection is a noninvasive and passive form of Trojan detection, and its effectiveness depends on the signal-to-noise ratio (SNR) and the Trojan-to-circuit ratio (TCR). The SNR is important as the effect of the Trojan can be masked by system or environmental noise. The TCR is important as it is the measurement of Trojan size to circuit size. Detecting small Trojans in large circuits is a growing challenge as IC feature sizes are continually becoming smaller and the number of transistors continues to increase in hardware designs. As with logic testing, side-channel analysis also typically requires a golden IC or model for Trojan detection [6, 45].

Although logic testing and side-channel analysis can be performed numerous times and on as many ICs as desired, they too suffer from drawbacks including Trojans in rare nodes remaining inactivated, activated Trojans being masked by the presence of noise, or variations in the IC tolerances during the manufacturing process. Therefore, performing a combination of logic testing and side-channel analysis can provide the best coverage for detecting hardware Trojans. Figure 2.26 shows the comparison between these two noninvasive hardware Trojan detection methods.

Logic Testing (Functional / Structural)	Side-Channel Analysis
✓ Performs well for detecting *small* Trojans	✓ Performs well for detecting *large* Trojans
✓ Robust to process noise	✓ Easy to generate test patterns
✓ Typically requires no additional hardware overhead	✓ Typically requires no additional hardware overhead
✗ Hard to detect *large* Trojans	✗ Hard to detect *small* Trojans
✗ Hard to generate test patterns	✗ Susceptible to process noise
✗ Typically requires golden IC or golden model	✗ Typically requires golden IC or golden model

Fig. 2.26 Comparison of noninvasive hardware Trojan detection methods

2.6 Conclusion

Hardware Trojans are a valid threat to hardware security and trust of integrated circuits. These malicious modifications pose a safety and security risk to any electronic system as the underlying *hardware is no longer considered the root of all trust*. Trojans may remain undetected and inactivated for many years while waiting for a specific circumstance to trigger so they may deliver their nefarious payload. We have seen a plethora of Trojan attack designs and recognize the need for a solid defense posture. In order to combat the threat of Trojans, accurate models for attacks and countermeasures must be ensured. Only with these models will detection and prevention techniques be realized. Current work in this domain relies on tradeoffs between security and performance, and no defensive technique provides a 100% guarantee. Future work in this domain will help answer the question if golden-free models are a viable approach for hardware Trojan defense mechanisms.

References

1. S. Ali, D. Mukhopadhyay, R.S. Chakraborty, S. Bhunia, Multi-level attack: an emerging threat model for cryptographic hardware, in *Proceeding of the Design, Automation & Test in Europe (DATE) Conference Exhibition* (2011), pp. 1–4
2. M. Banga, M. Hsiao, VITAMIN: voltage inversion technique to ascertain malicious insertions in ICs, in *Proceeding of the IEEE International Workshop on Hardware-Oriented Security and Trust* (2009), pp. 104–107
3. C. Bao, D. Forte, A. Srivastava, On reverse engineering-based hardware Trojan detection. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **35**(1), 49–57 (2016)
4. G.T. Becker, F. Regazzoni, C. Paar, W.P. Burleson, Stealthy dopant-level hardware Trojans: extended version. J. Cryptogr. Eng. **4**(1), 1–13 (2014)
5. S. Bhunia, M. Abramovici, D. Agrawal, P. Bradley, M.S. Hsiao, J. Plusquellic, M. Tehranipoor, Protection against hardware Trojan attacks: towards a comprehensive solution. IEEE Design Test **30**(3), 6–17 (2013)
6. S. Bhunia, M.S. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan attacks: threat analysis and countermeasures. Proc. IEEE **102**(8), 1229–1247 (2014)
7. B. Cha, S.K. Gupta, A resizing method to minimize effects of hardware Trojans, in *2014 IEEE 23rd Asian Test Symposium* (2014), pp. 192–199
8. R.S. Chakraborty, S. Bhunia, Security against hardware Trojan attacks using key-based design obfuscation. J. Electron. Test. (JETTA) Theory Appl. **27**(6), 767–785 (2011)
9. R.S. Chakraborty, S. Narasimhan, S. Bhunia, Hardware Trojan: threats and emerging solutions, in *IEEE International High Level Design Validation and Test Workshop* (2009), pp. 166–171
10. R.S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, S. Bhunia, MERO: a statistical approach for hardware Trojan detection, in *Proceeding of the Cryptographic Hardware and Embedded Systems (CHES)* (2009), pp. 396–410
11. G.K. Contreras, A. Nahiyan, S. Bhunia, D. Forte, M. Tehranipoor, Security vulnerability analysis of design-for-test exploits for asset protection in SoCs, in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)* (2017), pp. 617–622

12. D. Du, S. Narasimhan, R.S. Chakraborty, S. Bhunia, Self-referencing: a scalable side-channel approach for hardware Trojan detection, in *Proceeding of the Cryptographic Hardware and Embedded Systems (CHES)* (2010), pp. 173–187
13. C. Dunbar, G. Qu, Designing trusted embedded systems from finite state machines. ACM Trans. Embed. Comput. Syst. **13**(5s), Article 153 (2014)
14. D. Forte, C. Bao, A. Srivastava, Temperature tracking: an innovative run-time approach for hardware Trojan detection, in *Proceedings of the 2013 IEEE/ACM International Conference on Computer-Aided Design, ICCAD* (2013), pp. 532–539
15. Y. Jin, N. Kupp, CSAW 2008 team report (Yale University). CSAW embedded system challenge (2008). [Online], Available: http://isis.poly.edu/vikram/yale.pdf
16. Y. Kaiyuan, M. Hicks, Q. Dong, T. Austin, D. Sylvester, A2: analog malicious hardware, in *2016 IEEE Symposium on Security and Privacy (SP)* (2016)
17. R. Karri, J. Rajendran, K. Rosenfeld, M. Tehranipoor, Trustworthy hardware: identifying and classifying hardware Trojans. IEEE Comput. **43**(10), 39–46 (2010)
18. S.T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, Y. Zhou, Designing and implementing malicious hardware, in *Proceeding of the 1st USENIX Workshop Large-Scale Exploits Emergent Threats (LEET)* (2008)
19. L. Lin, W. Burleson, C. Paar, MOLES: malicious off-chip leakage enabled by side-channels, in *Proceedings International Conference on Computer-Aided Design (ICCAD)* (2009), pp. 117–122
20. Y. Liu, Y. Jin, A. Nosratinia, Y. Makris, Silicon demonstration of hardware Trojan design and detection in wireless cryptographic ICs. IEEE Trans. Very Large Scale Integr. VLSI Syst. **25**(4), 1506–1519 (2017)
21. E. Love, Y. Jin, Y. Makris, Proof-carrying hardware intellectual property: a pathway to trusted module acquisition. IEEE Trans. Inf. Forensics Secur. **7**(1), 25–40 (2012)
22. S. Mal-Sarkar, R. Karam, S. Narasimhan, A. Ghosh, A. Krishna, S. Bhunia, Design and validation for FPGA trust under hardware Trojan attacks. IEEE Trans. Multi-Scale Comput. Syst. **2**(3), 186–198 (2016)
23. S. Narasimhan, X. Wang, D. Du, R.S. Chakraborty, S. Bhunia, TeSR: a robust temporal self-referencing approach for hardware Trojan detection, in *Proceeding of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (2011), pp. 71–74
24. S. Narasimhan, W. Yueh, X. Wang, S. Mukhopadhyay, S. Bhunia, Improving IC security against Trojan attacks through integration of security monitors. IEEE Des. Test Comput. **29**(5), 37–46 (2012)
25. S. Narasimhan, D. Du, R.S. Chakraborty, S. Paul, F.G. Wolff, C.A. Papachristou, K. Roy, S. Bhunia, Hardware Trojan detection by multiple-parameter side-channel analysis. IEEE Trans. Comput. **62**(11), 2183–2195 (2013)
26. M. Potkonjak, Synthesis of trustable ICs using untrusted CAD tools, in *Proceeding of the Design Automation Conference* (2010), pp. 633–634
27. J. Rajendran, A.K. Kanuparthi, M. Zahran, S.K. Addepalli, G. Ormazabal, R. Karri, Securing processors against insider attacks: a circuit-microarchitecture co-design approach. IEEE Des. Test **30**(2), 35–44 (2013)
28. T. Reece, D.B. Limbrick, W.H. Robinson, Design comparison to identify malicious hardware in external intellectual property, in *Proceeding of the IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, Changsha (2011), pp. 639–646
29. M. Rostami, F. Koushanfar, J. Rajendran, R. Karri, Hardware security: threat models and metrics, in *Proceedings of the International Conference on Computer-Aided Design (ICCAD'13)* (IEEE Press, Piscataway, 2013), pp. 819–823
30. J. Roy, F. Koushanfar, I. Markov, EPIC: ending piracy of integrated circuits. IEEE Comput. **43**(10), 30–38 (2010)
31. H. Salmani, M. Tehranipoor, J. Plusquellic, A novel technique for improving hardware Trojan detection and reducing Trojan activation time. IEEE Trans. Very Large Scale Integr. VLSI Syst. **20**(1), 112–125 (2012)

32. H. Salmani, M. Tehranipoor, R. Karri, On design vulnerability analysis and trust benchmark development, in *IEEE International Conference on Computer Design (ICCD)* (2013)
33. M. Sanchita, A. Krishna, A. Ghosh, S. Bhunia, Hardware Trojan attacks in FPGA devices: threat analysis and effective counter measures, in *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI* (2014), pp. 287–292
34. B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, M. Tehranipoor, Benchmarking of hardware Trojans and maliciously affected circuits. J. Hardw. Syst. Secur. (HaSS) **1**(1), 85–102 (2017)
35. Y. Shiyanovskii, F. Wolff, A. Rajendran, C. Papachristou, D. Weyer, W. Clay, Process reliability based Trojans through NBTI and HCI effects, in *Proceeding of the NASA/ESA Conference on Adaptive Hardware and Systems* (2010), pp. 215–222
36. M. Tehranipoor, F. Koushanfar, A survey of hardware Trojan taxonomy and detections. IEEE Des. Test Comput. **27**(1), 10–25 (2010)
37. M. Tehranipoor, C. Wang, *Introduction to Hardware Security and Trust* (Springer, New York, 2012)
38. M. Tehranipoor, H. Salmani, X. Zhang, *Integrated Circuit Authentication* (Springer, Cham, 2014)
39. M. Tehranipoor, U. Guin, D. Forte, *Counterfeit Integrated Circuits: Detection and Avoidance* (Springer, Cham, 2015)
40. R. Torrance, D. James, The state-of-the-art in semiconductor reverse engineering, in *IEEE/ACM Design Automation Conference* (2011), pp. 333–338
41. TrustHub. https://www.trust-hub.org/index.php
42. N.G. Tsoutsos, M. Maniatakos, Fabrication attacks: zero-overhead malicious modifications enabling modern microprocessor privilege escalation. IEEE Trans. Emerg. Top. Comput. **2**(1), 81–93 (2014)
43. X. Wang, M. Tehranipoor, J. Plusquellic, Detecting malicious inclusions in secure hardware: challenges and solutions, in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)* (2008)
44. X. Wang, S. Narasimhan, A. Krishna, T. Mal-Sarkar, S. Bhunia, Software exploitable hardware Trojan attacks in embedded processor, in *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)* (2012), pp. 55–58
45. K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, M. Tehranipoor, Hardware Trojans: lessons learned after one decade of research. ACM Trans. Des. Autom. Electron. Syst. **22**(1), 6:1–6:23 (2016)
46. P.L. Yang, M. Marek-Sadowska, Making split-fabrication more secure, in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Austin (2016), pp. 1–8
47. X. Zhang, M. Tehranipoor, Case study: detecting hardware Trojans in third-party digital IP cores. IEEE Int. Symp. Hardw.-Oriented Secur. Trust **22**(1), 67–70 (2011)
48. Y. Zheng, S. Yang, S. Bhunia, SeMIA: self-similarity-based IC integrity analysis. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **35**(1), 37–48 (2016)

Part II
Hardware Trojan Attacks: Threat Analysis

Chapter 3
Hardware Trojan Attacks in SoC and NoC

Rajesh JS, Koushik Chakraborty, and Sanghamitra Roy

3.1 Introduction

Mobile and ubiquitous computing have become an integral part of our everyday life, with system on chip (SoC) at the heart of these devices. Three specific factors have contributed to the massive growth of SoC in low-power computing devices: (a) technology miniaturization, (b) modular system design in the form of reusable intellectual property (IP), and (c) high device density. However, due to unprecedented pressure of time to market, and to counter the rising costs of design and verification, modern SoC designs use third-party intellectual property (3PIP), procured from a diverse pool of trusted/untrusted vendors. The pervasive use of 3PIP components has introduced a zone of untrustworthy hardware within the SoC, creating a cataclysmic security loophole. Hardware Trojans can be strategically placed in critical 3PIPs to cause serious economic repercussions to both the IP clients and the end user.

While the SoC is dominating the mobile computing market, more sophisticated multiprocessor system on chip (MPSoC) is deemed to be the future of energy-efficient high-performance computing systems. Pivotal to this development, network on chip plays a critical role in improving the SoC scalability and energy efficiency. NoC interconnects a diverse set of on-chip IP blocks to function seamlessly within a single substrate. A key challenge however is to provide secure and reliable communication in the SoC, even in the presence of an untrusted NoC IP. Due to

R. JS (✉) • K. Chakraborty • S. Roy
Utah State University, Logan, UT, USA
e-mail: rajesh.js@aggiemail.usu.edu; koushik.chakraborty@usu.edu; sanghamitra.roy@usu.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_3

its singular and central role in the SoC on-chip communication, the NoC has direct access to all resources and information within the SoC. Hence, a compromised NoC 3PIP can wreak havoc in the SoC by inflicting a plethora of attacks such as information leakage, data corruption, and denial of service.

The exponential growth of our digital footprints, in the context of an untrusted SoC environment (in both mobile and high-performance computing), makes SoC security assurance an immediate necessity. This chapter focuses on security assurance of the SoC, with a special emphasis on hardware Trojans that can be embedded in the NoC. Section 3.2 discusses the challenges in SoC security. Section 3.3 presents an overview of the most conspicuous threat model, while Sect. 3.4 summarizes some promising system level solutions proposed for SoC security assurance. Section 3.5 delves into the less-explored territory of an untrusted malicious NoC 3PIP, with sample attack models and solutions.

3.2 Challenges of SoC Security

Over the last decade, hardware security has formed the foundation of modern computing design. However, the rise in the adoption of a horizontal integration strategy in the semiconductor industry has created a large number of untrusted zones in the supply chain. The challenges of SoC security with respect to the SoC design flow are enumerated below.

1. The SoC design flow is complicated and comprises numerous stages that incorporate both application-specific hardware design and software design processes. During this laborious process, the design passes through several hands across the world, creating a haven for malicious and unethical design practices.
2. The SoC design compels for several iterations of a few key stages. For example, before the final design tape-out, a few principal engineers are iteratively involved in the design and verification of the integrated circuits. This iterative process gives them several opportunities to hide malicious circuits and backdoors in the SoC design.
3. To reduce the design costs, as well as the time to market, SoC integrators acquire pre-verified hard or soft IP blocks from a large pool of untrustworthy vendors. These IPs largely serve as black boxes, since the third-party IP vendors do not reveal the design information to maintain competitive advantage in the niche market.
4. The innumerous third-party IP vendors often make use of untrustworthy and sometimes unverified automation tools for the design and verification of their IP blocks. Use of unreliable software tools can lead to design discrepancy and open the door for below par IP designs, with malicious backdoors.
5. Due to the rising verification costs, the SoC integrator does not reverify the procured IP blocks. Further, modern verification tools are often designed to handle “no less” functionality but not for “no more.” In other words, the

verification tools are capable of detecting missing functions compared to the accepted specifications but cannot detect dormant functions/circuits in large and complex digital designs.
6. SoC security comes at a formidable cost. For smaller applications, the SoC integrator cannot afford to implement premium security features as it may lead to severe design overheads in terms of chip area, performance, and cost.
7. Finally, the sheer complexity of modern SoC designs makes it nearly impossible to verify and assure trustworthy behavior of every subcomponent, as well as the interaction between multiple IP blocks.

Amidst this conundrum that the SoC design flow is, one specific issue stands out as a necessary evil: the use of third-party IPs. The following section presents a sample threat model of this quagmire.

3.3 SoC Threat Model

Proliferation of untrusted 3PIP components has been the biggest cause for security concern in modern SoC. Figure 3.1 illustrates a classic scenario where a SoC integrator procures various components from a diverse pool of 3PIP vendors. In this example, the 3PIP NoC supplied to SoC integrator 4 has a hardware Trojan embedded in it. One or more of the untrusted 3PIP blocks may contain a malicious circuit. Table 3.1 presents an overview of the threat life cycle, when a malicious circuit is inserted during the design of the IP block. Only the components designed in-house by the SoC integrator go through complete security and functionality test

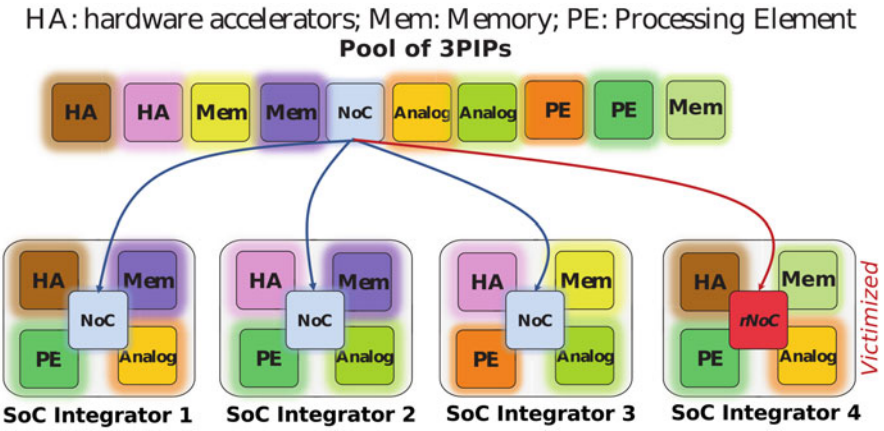


Fig. 3.1 Third-party NoC provider selectively supplies an NoC embedded with a hardware Trojan to one of its SoC integrators. The malicious NoC, with an accomplice software, can carry out a wide range of malicious activities at runtime damaging the unaware SoC integrator's reputation

Table 3.1 Threat life cycle in a SoC due to proliferation of untrustworthy third-party IPs

Trojan insertion
Malicious circuit can be inserted during the IP RTL design phase by one or few key IP design engineers. The act of few employees may result in an unsuspecting IP vendor supplying Trojan embedded IP to its clients
Trojan activation
To evade detection during IP verification and SoC integration tests, Trojans are designed to remain dormant. At runtime, they can be activated either by internal or external triggers using combinational or sequential rare event triggers
Trojan operation
At runtime, once active, the Trojan payload can inflict a plethora of attacks such as information leakage, denial of service, function manipulation, and data corruption. The type of attack carried out is only limited by the attackers' imagination

and validation. The lack of 3PIP design transparency and rising verification costs make it futile for the SoC integrator to assess the trust of procured 3PIPs, thereby creating a cataclysmic security loophole.

In this context, Sect. 3.4 presents a few promising measures for security assurance that have been explored in the recent past.

3.4 System-on-Chip Security Assurance

The goal of system-on-chip security is to ensure fair, secure, and reliable operation of the chip, often defined by the three central aspects of trustworthiness:

- *Confidentiality*: Protection of data privacy against leakage, theft, and illegal access.
- *Integrity*: Protection of data and chip functionality against undesired manipulations.
- *Availability*: Assurance of fair distribution and access to the physical resources based on correct functionality.

With research on SoC security still at its infancy, researchers have tried to adopt existing techniques employed in processors, memory, and hardware accelerators. Techniques such as physical side-channel analysis, built-in self-test (BIST), and proof-carrying hardware have been found insufficient to protect against the complex challenges imposed by hardware Trojan in SoC [2, 7, 8, 18, 19, 22]. For example, Balasch et al. measured the electromagnetic (EM) emanations from the SoC on actual hardware to detect hardware Trojan based on EM fingerprinting [2]. Although side-channel analysis techniques can detect hardware Trojans to some effect, they suffer from two important drawbacks. First, they require a golden/reference

circuit to compare against, which is unavailable in the 3PIP threat model. Second, many side-channel techniques suffer from inaccuracies due to process and test environment variations.

Dubrova et al. explored the use of logic BIST with a configurable key [8]. In logic BIST, the pseudorandom pattern generator (PRPG) is used to generate test inputs to the SoC circuit under test. The signature generated by the logic BIST is then compared to a precalculated signature to detect faults. Dubrova et al. proposed the use of a configurable key to set the initial state of PRPG, after the chip manufacturing process. The user calculated signature based on the key is stored as the "good" signature. Since the key is unknown at the manufacturing stage, addition of malicious circuit can change the signature exposing the presence of a Trojan. Despite its advantages, logic BIST suffers from high level of switching activity (power dissipation) and lacks sufficient coverage.

Love et al. proposed a novel IP acquisition and delivery protocol, where the vendor constructs a security compliance proof that is in agreement to the SoC integrator [18]. The formal proof then becomes a part of the IP deliverable, which can easily be checked by the SoC integrator. However, in this model, there is still room for the vendor to manipulate the proof, as well as test benches to not cover selected malicious modifications.

Dedicated infrastructure IPs and the use of fine-grained security policies have been gaining steam in SoC security assurance. Security policies establish quantifiable constraints based on abstract and complex high-level requirements such as access control, information flow, and time of resource usage. Wang et al. proposed a plug-and-play IP dedicated to security functions to protect against information leakage from scan-based attacks, counterfeiting, and hardware Trojans [23]. The centralized IP uses the core test wrapper architecture in the SoC and consolidates a few well-established security measures to counter against threats. For Trojan detection, a combinational path delay-based detection mechanism is employed. Basak et al. extend this work by enforcing system-level security policies as firmware code in the dedicated security IP block known as *extended infrastructure IP for Security* (E-IIPS) [3]. The SoC security policies can be configured by upgrading the firmware code stored in a secure memory. The security policy that can be implemented is limited only by the observable and controllable signals accessible from the security wrappers.

In the recent times, there has been a strong call for hardware software cooperation in Trojan detection. The increasing SoC design complexity, along with increasing untrusted boundary transactions between the hardware and software, requires a system- level approach to Trojan detection. In this regard, Jin et al. proposed a *hardware anchor* to allow cross-layer data and intelligence sharing between the operating system and the hardware [14]. The hardware module is also responsible for enforcing the security policies set by the OS. Guo et al. proposed a novel integrated formal verification framework that combines an automated model checker along with an interactive theorem prover [13]. The framework checks system-level security properties against the SoC design.

Security assurance of SoC is fast becoming a disciple by itself. Although hardware Trojan detection has garnered sufficient research interest, effort toward protection against malicious circuits is still limited. In this regard, Kim et al. proposed the use of an embedded or external reconfigurable logic to replace system functions that have been infected with hardware Trojans at runtime [16]. Such self-regenerative systems, although promising, are prohibitive in low-power computing applications.

3.5 NoC Security

Network on chip is pervasive in modern MPSoC as it serves as a modular and scalable interconnection fabric for all on-chip components. More than a decade's research in NoC has provided valuable insight to understanding the interconnection network and its security aspects. However, a lion's share of existing literature on NoC security studies attacks on the NoC emanating from the software or hardware Trojans in other IP blocks as shown in Table 3.2. In a majority of these works, the security checks are embedded in the NoC. In this regard, Evain et al. presented an interesting study on the various possible attacks in a NoC enumerated below [9].

1. Denial of service attacks aimed at degrading the system performance. The attack can be carried out by introducing packets of data in incorrect paths, intentional deadlock paths, and infinite livelock paths to consume valuable resources that can be used by other packets.
2. Information extraction by reading data from an unauthorized target.
3. Hijacking a secure area to modify the system behavior.

Table 3.2 Summary of threat in the NoC

	Trojan Loc[a]	Attack[b]	Prot[c]	TM[d]
DPU[10]	S/W	Mem (confidentiality)	NI	—
KeyCore[12]	S/W	Mem (confidentiality)	NI	—
surfNoC[24]	S/W	S/W (confidentiality)	NoC	—
AE[15]	S/W	Mem (confidentiality)	NI	—
IMP[17]	μP	μP/Mem (confidentiality)	—	S/W
NoC-MPU[20]	S/W	Mem (confidentiality)	NI	—
FortNoC[1]	NoC IP	NoC (confidentiality)	NI	S/W-H/W
RLAN[21]	NoC IP	NoC (availability)	NI	—
Mal-NI[11]	NI	NoC (integrity & availability)	NI	—
FI-DoS[5]	NoC Link	NoC (availability)	NoC	—

[a]The part of the system where a Trojan in inserted
[b]The part of the system that is attacked and the type of attack
[c]The part of the system where a protection mechanism is implemented to prevent the attack
[d]The triggering mechanism in case of a hardware Trojan

Owing to NoC's pivotal role in the SoC, it is essential to consider the presence of hardware Trojans in the 3PIP NoC circuit itself. This scenario gives rise to a unique challenge as NoC has direct access to all the data, as well as resources in the SoC. A compromised NoC can be far more potent in inflicting the attacks discussed by Evain et al. [9]. Further, the security measures to protect the SoC can no longer be placed in the NoC. The following sections showcase a handful of works that consider a hardware Trojan embedded in the NoC and innovative solutions to detect the malicious activity.

3.5.1 Information Leakage Attack

Dean et al. demonstrated that an information theft attack can be carried out by a hardware Trojan embedded by the NoC in the presence of an accomplice software, without relying on memory access [1]. To counter the threat, a three-layered security measure was proposed by augmenting the SoC firmware. The first security layer scrambles the data before transmitting it onto the network to deter Trojan activation based on sequential coded message. In the second layer, a packet authentication technique is used to prevent invalid packets from reaching undesired destinations. At the topmost layer, the source and destination nodes of communicating nodes are decoupled by periodically migrating running applications.

3.5.1.1 Attack Semantic

Figure 3.2 illustrates the attack semantics of the information leakage carried out by the compromised NoC. The study considers a cloud computing environment in an MPSoC, where multiple users are sharing hardware resources to execute their applications. First and foremost, the third-party NoC IP supplier embeds a hardware Trojan in the NoC during the IP design phase. The SoC integrator unaware of the

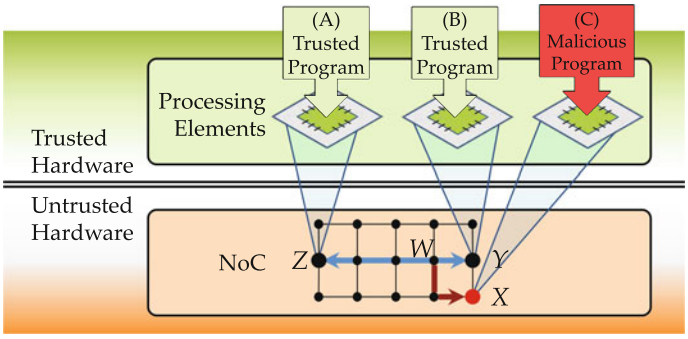


Fig. 3.2 Compromised NoC snooping data messages between programs A & B and leaking to the accomplice program C

malicious circuit tapes-out the design and supplies it to an end user. In the cloud setting, an accomplice thread in the form of a third-party application is scheduled on one of the on-chip processing elements. The malicious application establishes a covert communication channel with the hardware Trojan in the NoC to activate it. For example, in Fig. 3.2, an accomplice thread in the NoC node *X* could send a command to node *W* to activate the dormant Trojan in its NoC router.

Once active, the Trojan duplicates specific instances of ongoing communication in the network and leaks it to the accomplice thread. The hardware Trojan is designed to snoop on the communication from the network interface and duplicate the packets incoming to one of the input ports. The Trojan in node *W* (Fig. 3.2) leaks all the communication between nodes *Y* and *Z*. The malicious circuit present in NoC router of *W* will duplicate all packets traversing through it belonging to the requested communication and sends it to the accomplice thread in node *X*. Finally, the embedded hardware Trojan can also be instructed to suspend its malicious activity by sending a sequence of coded message.

The hardware Trojan is implemented using a state machine with three major states: *inactive* (Trojan dormant), *waiting* (Trojan active and awaiting snooping commands), and *Leaking* (ongoing Trojan attack). The state machine changes are only sanctioned by the accomplice thread using a specific coded sequence of flits. The probability of an accidental state change is extremely low. Considering that a code with five consecutive 32-bit sequences is required to activate the Trojan in the NoC router, the probability of accidental state change (Trojan activation) can be to the order of 10^{50}.

3.5.1.2 Security Measures: FortNoCs

FortNoCs employ a three-layered security measure that is aimed to provide both proactive and reactive protection against information theft. The security mechanism is implemented in the NoC firmware that interfaces the processing element with the network interface, by an in-house design team to forego any reliance on third-party vendors. The three layers target specific goals for assuring secure communication in the MPSoC.

1. **Layer 1**. The first line of defense against the hardware Trojan is to prevent its activation. The SoC firmware ensures that the data processed by the core is shuffled in the network interface, before injecting it into the network. Low-cost XOR cipher-based encryptors and decryptors are used to scramble the data. By distorting the data being injected into the network, the SoC firmware blocks the Trojan activation. The layer also achieves a secondary goal. As the data in the network is enciphered, the attacker must go through an additional step to understand the stolen data.
2. **Layer 2**. A packet authentication-based data integrity protection mechanism is employed, to protect against information theft in the event that the Trojan is activated. An encrypted tag specific to each source and destination is packetized

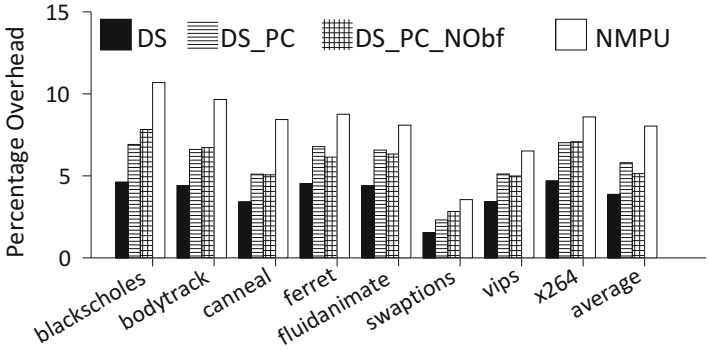


Fig. 3.3 Performance overhead incurred due to layers of FortNoCs security measures

along with the data required to be communicated. At each destination, the SoC firmware verified the data and ensures that the tags match the intended communication route. If a malicious application tries to reroute the packet to an unintended location, the defense mechanism drops the packet and triggers exceptions to alert the operating system of the anomalous behavior.

3. **Layer 3**. The final layer of FortNoCs aims to disrupt the formation of communication patterns by periodically migrating the applications to different nodes. By decoupling the source and destination of communicating nodes, the security mechanism introduces noise in the leaked data, making it harder for the attacker to extract useful information.

The layered approach employed in FortNoCs benefits by allowing the designers to configure the level of security required based on the design requirements. Figure 3.3 illustrates the incremental impact on performance overheads due to the addition of each security layer. FortNoCs security mechanisms have low overheads with the overall technique creating a performance deficit of 6%, as compared to another recent authentication-based security mechanism (NMPU, 8%). Individually Layer 1 impacts the performance by 3.8% on an average, whereas addition of Layer 2 and Layer 3 impacts the performance by 2% and 0.01%, respectively.

3.5.2 Packet Security Against Fault Injection Attacks

Boraten et al. have proposed a three-pronged packet validation technique to protect against fault injection-based side-channel attacks in the NoC [4]. They first enhance the NoC router fault tolerance using configurable error detection encoding schemes. At the second level, the packet header is encoded with the route information, and finally, packet allocation is prioritized to improve the NoC quality of service.

3.5.2.1 Attack Semantics

The aim of the attacker is to break the packet encoding by injecting faults onto the router links. The attacker controls the inputs to an algebraic manipulation detection (AMD) encoder and generates a set of error vectors on the link. Using the knowledge obtained from different input combinations to the AMD, and observing the effect of side-channel attacks on the encoder and decoders, the attacker can iteratively decipher the encryption key. Once the encryption key is known, the attacker can compromise the data computed and communicated in the MPSoC. Three specific attack scenarios are presented. In the first case, data encoded using cyclic redundancy check (CRC) encoding is attacked by inflicting side-channel attacks on the compromised NoC links. In the second scenario, an AMD-encoded packet is attacked on the compromised links. In the third attack, a compromised router duplicates a target packet and sends it to another core to leak information.

3.5.2.2 Security Measures: P-Sec

To protect the data integrity, the NoC microarchitecture fault tolerance is improved by adopting AMD encoding encryption for sensitive and critical data. For all other packets, a cyclic redundancy check (CRC) is employed to provide basic fault tolerance. To help decode the data correctly, the type of encoding is tagged along with the packet header.

Due to its encoding simplicity, a side-channel attack on the CRC-encoded packets goes undetected leading to silent data corruption. However, AMD-encoded traffic can detect malicious alterations to the packet as AMD codes by definition cannot be masked into other valid codes. The header information of all packets, critical and noncritical, are encoded using AMD code word, to ensure secure and reliable packet delivery. As an additional protection, to prevent a maliciously constructed packet from using a valid header, a second level of encryption is added. When the decoded packets are found to be maliciously rerouted to an undesired location, the packets are dropped within the network interface, and a NACK is sent to the source.

To enhance the quality of service of sensitive packets, the packets encoded using AMD have an additional flag to signal the arbitration units to grant higher priority to these packets. Two major drawbacks of AMD encoding are that it does not help hardware Trojan localization and the overheads of packet-level AMD encoding are very high.

3.5.3 Network Interface Malfunctions

Frey et al. demonstrated that a hardware Trojan in the state machines of the network interface (NI) can cause its malfunction and hence degrade the system performance

[11]. To counter the threat, they propose a key-based FSM with dummy states to detect malicious state transitions.

3.5.3.1 Attack Semantic

The finite-state machine (FSM) control in the NI has two sets of roles based on whether it is serving as the communication initiator or communication target. As the initiator, it is responsible to receive data from the IP module, encapsulate data into packets, and inject them into the network. Similarly, as a target, the FSM has a set of responsibilities denoted by finite states. A hardware Trojan in the FSM control of the NI can inconspicuously cause NI malfunctions by making undesired state changes.

3.5.3.2 Security Measures

To counter the state-changing hardware Trojan, the FSM control in NI is obfuscated by introducing an authentication key to the state change conditions and a few dummy states. Since the attacker is unaware of the authentication, any state change transactions made without the key or with a wrong key will result in a state change to one of the dummy states, indicating the presence of malicious activity. Each true state is connected to a dummy state, whereas in cases of hardware constraints, multiple true states can be connected to the same dummy state. The technique employed has a high Trojan detection rate.

3.5.4 Denial of Service Attack

Rajesh et al. proposed a focused bandwidth denial attack in the NoC, where the communication from only a subset of the NoC nodes are disrupted [21]. By selectively denying resources to certain packets, the hardware Trojan embedded in the NoC causes severe performance bottlenecks to select applications. However, the overall system performance remains largely unaffected thereby concealing the malicious activity. To counter the threat, they propose a noninvasive runtime solution based on packet latency monitoring.

3.5.4.1 Attack Semantic

Rajesh et al. consider a cloud computing-based environment, where multiple applications compete for the same underlying hardware resources. By disrupting the communication of specific applications, the NoC delivers unfair advantage to

other applications running on the same MPSoC. The malicious hardware Trojan is inserted into the NoC IP during the design phase and activated by a malicious application at runtime. Figure 3.4 shows the malicious NoC router design employed by them. The Trojan snoops on the input ports to collect statistics about the communication patterns and affects resource availability by covertly manipulating the NoC router control plane.

To implement a selective denial of service in a large network, the challenge lies in the selection of victim node so as to cause a noticeable drop in application performance. Figure 3.5 demonstrates an example of victim selection proposed by Rajesh et al.. The embedded malicious circuit in the NoC router dynamically identifies a node with high network usage rate. They propose a hierarchical approach to victim selection, so as to monitor only a handful of nodes at any given time and hence maintain a low Trojan design footprint. For example, in Fig. 3.5, a two-stage victim selection process is shown. In the first stage, a region with heavy traffic is selected using traffic aggregation techniques similar to those used in congestion-aware networks. Subsequently, in stage 2, the victim selection converges on that region to select one or few victim nodes with high ingress/egress rates. In this rudimentary victim selection process, they do not account for the application criticality or even the nature of packets that are attacked.

Once the victims are selected, the hardware Trojan covertly manipulates the traffic flow of flits pertaining to the victim nodes. Resource contention modules such as switch arbitration and virtual channel allocation stages within each router are manipulated as shown in Fig. 3.4. The attacker conceals the attack by manipulating the traffic flow in a distributed manner. Small delays are inflicted at each hop for

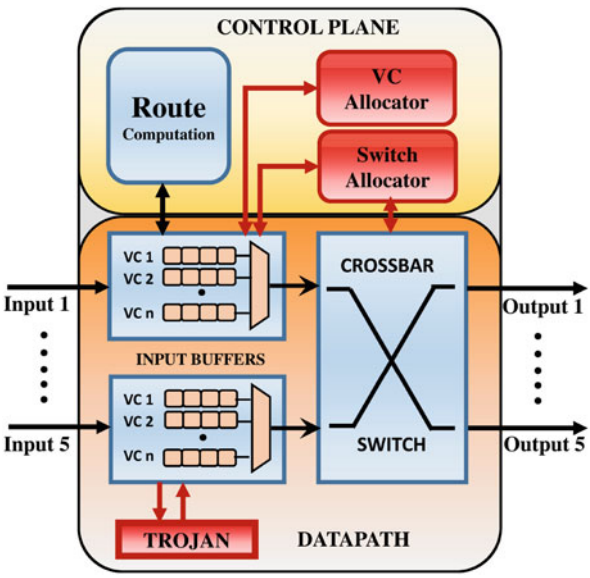


Fig. 3.4 Conceptual view of a hardware Trojan in the NoC router. The Trojan monitors the usage rate of specific nodes by snooping on the input ports and attacks by manipulating the resource contention stages

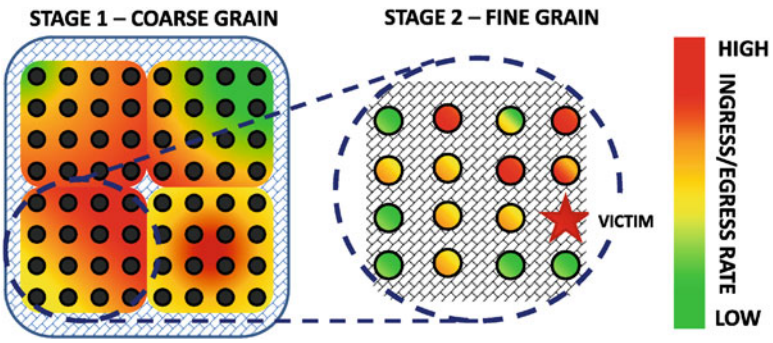


Fig. 3.5 Process of victim selection to inflict a focused denial of service attack. In stage 1, aggregate bandwidth information of the region is collected and in stage 2, a victim is identified based on high usage rates

the victim packets. The attack may be misinterpreted as traffic congestion-related delays. By carefully selecting the delay at each router, the attackers degrade the victim application packets by up to 72%.

3.5.4.2 Security Measures

To detect the traffic anomalies in the NoC, they employ a noninvasive runtime detection mechanism by monitoring the packet latency. Packets that fulfill the following three conditions are expected to have comparable packet latencies.

- First, the packets must have similar characteristics such as packet size and message class.
- Second, the packets must traverse the same path between two nodes and must have comparable hop count.
- Finally, the packets must traverse the network in nearly the same time.

The similarities between real-world automobile traffic and the communication packets in the network are used to formulate the hypothesis. Figure 3.6 shows the variance in time required to reach a destination if the spatial (different routes) and temporal (different time of travel) conditions are not met. Similarly, an analogous monitoring packet is inserted into the network for selected communication, such that the analogous packet traverses the same path at nearly the same time. The technique relies on comparing the difference in arrival time between the original packet and the analogous packet to detect malicious traffic activity.

Figure 3.7 provides an overview of the technique. An analogous packet to that of the victim is created by the SoC firmware after acquiring the characteristics of data passing from the processing element to the network interface. A proximal node is then selected considering that the route taken by the two packets must be the same and should have a comparable hop count. The packet traversal time stamps between

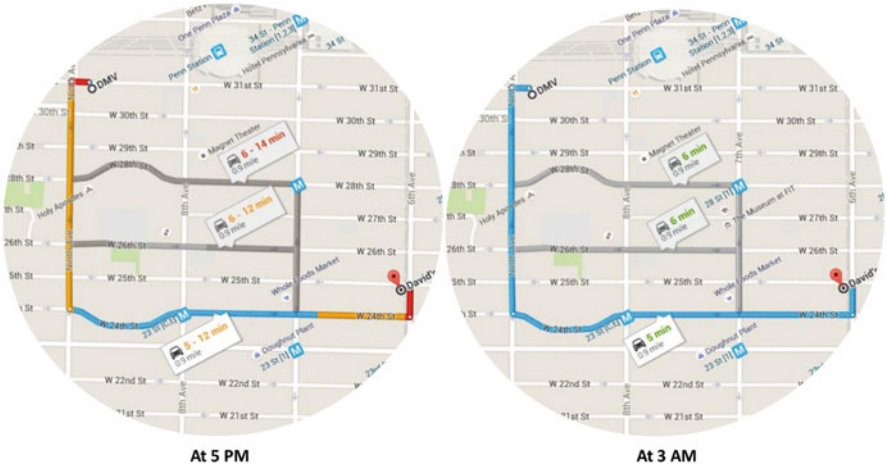


Fig. 3.6 Analogy used for the hypothesis formation of security mechanism

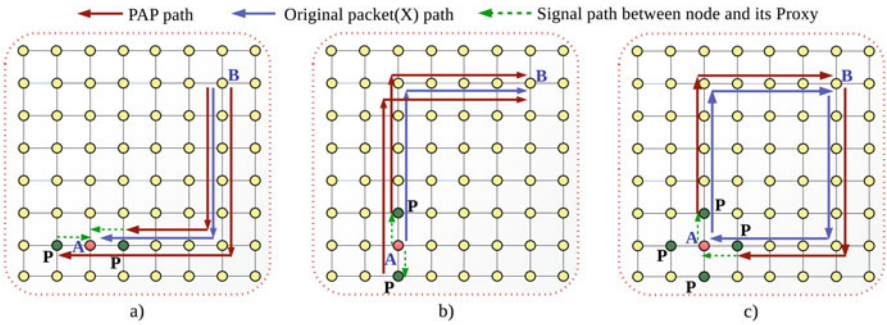


Fig. 3.7 Overview of latency auditor. The proximal nodes around the victim act as proxy source/destination to the duplicate packets. Both the original and the duplicate packet traverse the network in the same path at a similar timeframe. (**a**) Ingress packets of A under attack. (**b**) Egress packets of A under attack. (**c**) Ingress & Egress packets of A under attack

the analogous packet and original packet are compared. The figure shows three cases of attack and the process of proximal node selection.

- In case (*a*), only the packets incoming to node *A* are under attack. In this scenario, *P* is selected as a proxy node. In node *B*, when the packet is injected into the network, an analogous packet is created and injected at *B* such that its destination is *P*. Once the analogous packet arrives at *P*, the time stamp is noted, and the packet is absorbed and reinjected to traverse to *A*. At *A* the latency and packet arrival time are compared between the original and analogous.
- In case (*b*), the packets originating from *A* are attacked. In this scenario, *P* is selected as proxy, and an analogous packet is injected from *P*, with destination as *B*. At *B* the arrival times are compared.

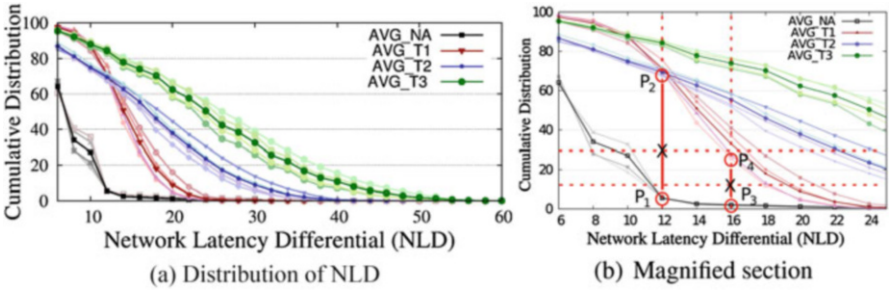


Fig. 3.8 Efficacy of RLAN for four cases: no-attack and three attack scenarios, T1,T2, and T3, with delay per hop as 1,2, and 3, respectively. All benchmarks are shown in lighter shade, with the average shown in the darker shade. (**a**) Distribution of NLD (**b**) shows the magnified section for selection of CDF-NLD threshold and analysis of false positives and false negatives

- In case (*c*), both, incoming and outgoing packets of *A*, are attacked. The proxy nodes are selected to serve as fake source, as well as fake destination.

If the difference in packet arrival time stamps between the original and the analogous packets are beyond a set threshold, abnormal traffic activity is flagged to be further inspected. Figure 3.8 presents the cumulative distribution (y-axis) of the latency difference between the original and the analogous packet (x-axis). There exists a clear contrast between the attack and no-attack scenario. In the no-attack scenario, only a minimal percentage of the overall packets have a latency differential greater than 12 cycles, whereas for all attack scenarios, a considerable portion of the packets have high latency differential between the original and analogous packets. A well-tuned latency auditor is able to detect selective denial of service attacks with marginal overheads of 5.5%, on an average and high accuracy. A similar contrast was also noticed between the no-attack and attack scenarios in a multi-application environment.

3.5.5 Fault Injection-Based Denial of Service

Boraten et al. presented a fault injection-based attack in NoC, where the injected fault triggers a response from the error correction code to cause repeated packet retransmission thereby denying resources to actual transmission [5]. To detect and counter the threat, they first classify the faults to discover compromised links and then implement a switch-to-switch error correction code along with data obfuscation to avoid Trojan triggering.

3.5.5.1 Attack Semantic

A lightweight target-activated sequential payload (TASP) Trojan is implemented to inspect the packets and identify specific packet information such as thread ID, memory address, source, and destination. Based on the deciphered information, the TASP selects a target packet to inject faults across multiple links. A finite-state machine-based counter is developed to carefully generate locations to inject the link faults and disguise them as transient link faults.

The TASP remains in dormant state until an externally driven kill switch is enabled. Once the Trojan is enabled, TASP actively snoops the packets to select a target/victim packet. The Trojan then injects faults across multiple links to inflict irrecoverable errors on the packet. To circumvent NoC fault tolerance techniques such as packet reroute in the presence of permanent link faults, the Trojan ensures that the same link does not suffer from multiple faults. A finite state machine is used to shift the injected faults to different links to mimic a transient fault in the network. The injected faults force retransmission of packets from the source. Since multiple links are rendered faulty by the hardware Trojan, the attacker expects retransmission of multiple packets thereby creating high network congestion and saturation. The attack potency can range from simple degradation in application performance to full-chip failure as a result of communication backpressure causing deadlock.

3.5.5.2 Security Measures: Switch-to-Switch Obfuscation

To counter the threat of triggering hardware Trojans, three methods of packet obfuscation are employed: *data shuffling, inverting, and scrambling*. Each router is equipped with an *L-Ob* module attached to the retransmission buffer that employs the packet obfuscation techniques for different granularity such as entire flit, only payload or only header. By carefully selecting the granularity, the Trojan trigger can be identified.

Lightweight switch-to-switch (*s2s*) error correction codes are employed within each router. Further, a threat detector locally records the packet header information when a link fault is detected in the network. If a packet with similar header information has been recorded earlier, a built-in self-test is triggered to test the link for permanent fault. Then, the mode of previous obfuscation is noted to alert the *L-Ob* module to try alternate obfuscation techniques. Iteratively, all modes of obfuscation are tried, and if no faults are detected at any given time, the method of obfuscation is noted to speed up future selection process. The upstream router is then notified of the successful transmission. Hardware Trojan by link classification requires marginal area and power overheads due to the addition of *L-Ob* module with switch-to-switch obfuscation.

3.5.5.3 Secure Model Checkers

Boraten et al. in their other work employ invariant model checkers to provide instantaneous fault detection [6]. To begin with, they first identify security points of interest (SPOI) based on the vulnerabilities in the NoC microarchitecture and fault-tolerant techniques. Manipulations to the NoC microarchitecture, such as illegal virtual channel allocations, unfair arbitration techniques, and inadvertent dynamic frequency voltage triggers, can result in covert performance degradation. In the current NoC microarchitecture, router components such as buffers, the router pipeline stages, and the status registers are all considered as points of interest. Similarly, techniques such as cache protocols, DVFS, routing reconfiguration, wear-out, and aging are all SPOI.

Model checkers compare the input of control logic to the output to determine if an event or action is illegal. When a rule violation occurs, an assertion flag is raised. Boraten et al. expand existing model checkers to include three new rules:

- Inefficient output: The NoC decisions that lead to flooding, starvation, and de-prioritization of network resources are considered inefficient outputs. Fault-tolerant techniques are adapted to detect the threat of inefficient outputs.
- Starvation and previous step validation: An invariant rule is introduced to correlate to previous output with the next output to ensure timely functionality.
- De-prioritization: To account for the violation, the previous requests and responses on resource contention stages are logged, and the arbitration fairness is evaluated.

Based on the invariant rules proposed, Boraten et al. explore a host of security enhancements. In the route computation stage, inefficient route selection and de-prioritization rules are added. In the virtual channel and switch allocation stage, rules are enforced to check allocation requests, mismatches in credit counts, inconsistent allocations, and de-prioritization of grants.

3.5.6 Trojan Detection Using Error Control Approach

Yu et al. explored the use of error control coding (ECC) to detect the presence of hardware Trojans in the NoC links [25]. The Trojans inserted in NoC links maliciously alter the packet header to potentially create deadlock, livelock, and packet loss. The Trojan model considered in this work attacks the NoC link wires to induce errors in one or more link wires.

The transmitter router in the NoC is equipped with an ECC encoder, and the receiver module is equipped with two ECC decoders. When a hardware Trojan manipulated the flit, it is stored in the erroneous flit buffer, and it is signaled to be retransmitted. During retransmission, the odd and even bits are switched to ensure that the same bits are not affected by the malicious circuit in the NoC link. When

the retransmitted flit is received, the reshuffled bits are combined with the flits saved in the erroneous flit buffer to create two versions of the corrected flits. The correct version is then passed to the next hop in the router. The proposed method can correct errors and detect hardware Trojan as long as one of the three conditions are met: (a) single link wire is affected, and (b) two even or two odd link wires are affected. Another limitation of the technique is that the protection mechanism works only if the hardware Trojan is active for a very short duration.

In cases where the hardware Trojan is active for longer duration, a link isolation algorithm is employed. The link wire isolation can solve two-bit errors induced by the hardware Trojan.

3.6 Open Challenges

The scope of hardware security requirements is largely defined by the application domain. Hardware used in the finance sector or for safety critical applications are subject to more stringent trust and security constraints, as compared to that used in the multimedia applications. In this context, there is a need for comprehensive analysis and classification of threats related to the NoC, with configurable and modular strategies to address these threats at low cost. Although research on NoC security has slowly progressed over the last decade, there exist several challenges that need further attention.

- First and foremost, traditional NoC security assurance techniques largely rely on the the NoC designer to perform circuit and architectural modifications. However, due to a boom in the use of untrustworthy 3PIP designs, existing threats should be reexamined with more emphasis on securing the entry and exit checkpoints of the communication network.
- NoC designs lack transparency and suffer from indeterministic packet delivery latency due to increasing demand for NoC state-aware routing and arbitration. Small malicious modifications to these adaptive techniques can often go unnoticed and have catastrophic effects on NoC functionality, energy efficiency, and performance. Runtime solutions to monitor and diagnose NoC fairness and allocation decisions are required to counter denial of service attacks. In addition, the recent introduction of learning techniques in NoC routing poses new challenges to fair and secure packet transmission.
- The advent of photonic, optical, and wireless NoC architectures reveal uncharted territory in NoC security. Introduction of crosstalk noise, altering signal-to-noise ratios to inflict DoS attacks, is the tip of a fast-growing security problem.
- Modern NoC accounts for a significant portion of the total chip power and also plays a critical role in chip temperature distribution. Power-based attacks in the NoC to disrupt local/global chip operations need to be carefully studied.

3.7 Summary

Modern electronic devices have complete access to our personal lives and handle a large share of our sensitive information. Owing to this critical role, security assurance of system on chips has quickly become a discipline by itself. With each passing generation, the increasing complexity of system on chips further piles on new and unique challenges to secure and reliable computing. Validation and assurance of the three basic principles of security, *confidentiality*, *integrity*, and *availability*, are quickly becoming unmanageable. In this context, there is an immediate need for low cost, low design, and verification overhead solutions for system-on-chip security assurance.

Security assurance requires a two-pronged approach. At one end, novel and innovative attack vectors must be designed to stay ahead of the attackers. Careful analysis of different hardware abstraction levels is required to identify and understand the various points of security vulnerability. On the other hand, efforts must be directed toward making designs more transparent and verifiable while still maintaining the design internals confidential to promote innovation in the niche market. In this regard, this chapter provides a brief insight into the two complementary worlds of security assurance.

References

1. D.M. Ancajas, K. Chakraborty, S. Roy, Fort-NoCs: mitigating the threat of a compromised NoC, in *Proceedings of the 51st Annual Design Automation Conference* (ACM, 2014)
2. J. Balasch, B. Gierlichs, I. Verbauwhede, Electromagnetic circuit fingerprints for hardware trojan detection, in *2015 IEEE International Symposium on Electromagnetic Compatibility (EMC)* (IEEE, 2015)
3. A. Basak, S. Bhunia, S. Ray, A flexible architecture for systematic implementation of SoC security policies, in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (IEEE Press, 2015)
4. T. Boraten, A.K. Kodi, Packet security with path sensitization for NoCs, in *Proceedings of the 2016 Conference on Design, Automation and Test in Europe* (EDA Consortium, 2016)
5. T. Boraten, A.K. Kodi, Mitigation of denial of service attack with hardware Trojans in NoC architectures, in *2016 IEEE International Parallel and Distributed Processing Symposium* (IEEE, 2016)
6. T. Boraten, D. DiTomaso, A.K. Kodi, Secure model checkers for Network-on-Chip (NoC) architectures, in *2016 International Great Lakes Symposium on VLSI* (IEEE, 2016)
7. D. Du et al., Self-referencing: a scalable side-channel approach for hardware Trojan detection, in *International Workshop on Cryptographic Hardware and Embedded Systems* (Springer, Berlin/Heidelberg, 2010)
8. E. Dubrova et al., Keyed logic BIST for Trojan detection in SoC, in *2014 International Symposium on System-on-Chip (SoC)* (IEEE, 2014)
9. S. Evain, J.-P. Diguet, From NoC security analysis to design solutions, in *IEEE Workshop on Signal Processing Systems Design and Implementation* (IEEE, 2005)
10. L. Fiorin et al., Secure memory accesses on networks-on-chip. IEEE Trans. Comput. **57**(9), 1216–1229 (2008)

11. J. Frey, Q. Yu, Exploiting state obfuscation to detect hardware trojans in NoC network interfaces, in *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)* (IEEE, 2015)
12. C.H. Gebotys, R.J. Gebotys, A framework for security on NoC technologies, in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI* (IEEE, 2003)
13. X. Guo et al., Scalable SoC trust verification using integrated theorem proving and model checking, in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (IEEE, 2016)
14. Y. Jin, D. Oliveira, Trustworthy SoC architecture with on-demand security policies and HW-SW cooperation, in *5th Workshop on SoCs, Heterogeneous Architectures and Workloads (SHAW-5)*, 2014
15. H.K. Kapoor et al., A security framework for NoC using authenticated encryption and session keys. Circuits Syst. Signal Process. **32**(6), 2605–2622 (2013)
16. L.-W. Kim, J.D. Villasenor, Dynamic function replacement for system-on-chip security in the presence of hardware-based attacks. IEEE Trans. Reliab. **63**(2), 661–675 (2014)
17. S.T. King et al., Designing and implementing malicious hardware. LEET **8**, 1–8 (2008)
18. E. Love, Y. Jin, Y. Makris, Proof-carrying hardware intellectual property: a pathway to trusted module acquisition. IEEE Trans. Inf. Forensics Secur. **7**(1), 25–40 (2012)
19. S. Narasimhan et al., Hardware Trojan detection by multiple-parameter side-channel analysis. IEEE Trans. Comput. **62**(11), 2183–2195 (2013)
20. J. Porquet, A. Greiner, C. Schwarz, NoC-MPU: a secure architecture for flexible co-hosting on shared memory MPSoCs, in *Design, Automation and Test in Europe Conference and Exhibition (DATE)* (IEEE, 2011)
21. J.S. Rajesh et al., Runtime detection of a bandwidth denial attack from a rogue network-on-chip, in *Proceedings of the 9th International Symposium on Networks-on-Chip* (ACM, 2015)
22. K. Rosenfeld, R. Karri, Security-aware SoC test access mechanisms, in *2011 IEEE 29th VLSI Test Symposium (VTS)* (IEEE, 2011)
23. X. Wang et al., IIPS: infrastructure IP for secure SoC design. IEEE Trans. Comput. **64**(8), 2226–2238 (2015)
24. H.M.G. Wassel et al., SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip. ACM SIGARCH Comput. Archit. News **41**(3), 583–594 (2013). ACM
25. Q. Yu, J. Frey, Exploiting error control approaches for hardware trojans on network-on-chip links, in *2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)* (IEEE, 2013)

Chapter 4
Hardware IP Trust

Mainak Banga and Michael S. Hsiao

4.1 Introduction

The current electronic market is dynamic and rapidly evolving with an ever-increasing demand of digital hardware. Such digital hardware can range from a discrete intellectual property (IP) to a system on chip (SoC) or even to a complete platform or system. As the price of electronic items continues to trend downward, hardware companies are primarily resorting on volume transaction for keeping up their profit margin. To this end, reducing the design cost as well as the manufacturing cost will help in bringing down the final cost of the product.

On the one hand, reducing the design cost often involves reusing or procuring parts of the design from a third-party entity without creating everything in-house. These parts might include processors like x86, ARM, NVIDIA general processing units (GPUs), input-output (IO) blocks like Peripheral Component Interconnect (PCI), Universal Serial Bus (USB), memory blocks like nonvolatile memories (NVMs), dual data rate memories (DDRs), and many more. A typical product involves multiple of these parts assembled together in a logical way to realize the product specification provided by the customer. The trustworthiness of individual parts is typically considered, though not rigorously challenged. Likewise, the integration of all the components is often done with the trust of individual components in mind. How to ensure trust of the final system remains a major question for safety-critical systems. On the other hand, reducing the cost of manufacturing often involves producing the part at a lower cost, typically achieved by contracting the

M. Banga (✉)
Intel Corporation, 1900, Prairie City Road, FM6-45, Folsom, CA, 95630, USA
e-mail: banga@vt.edu

M.S. Hsiao
Virginia Tech, Blacksburg, VA, USA

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_4

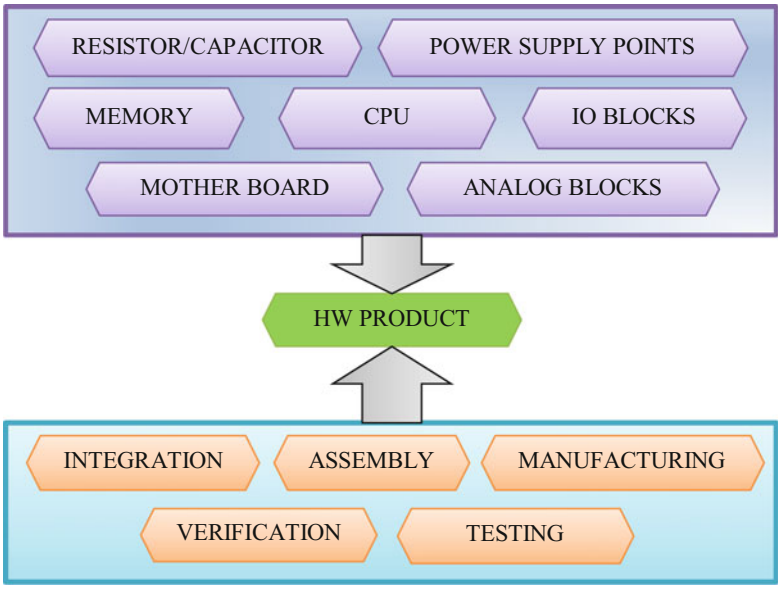


Fig. 4.1 Mix of IPs and processes that produce a HW product

job to a more economically viable offshore region, the process often termed as outsourcing. The trust of offshore manufacturer may be in question, as they may compromise the design integrity during the process.

In either of these cost reduction methods, since a part or the entirety of the design is handled by a third party at some point in the life cycle of the design, functional correctness, trust, and quality of the final product arise as a concern to the parent company. This forces them to seek for ways that can check these circuits for possible tampering/intrusions. At the same time, the cost associated with guaranteeing trust and correctness is not expected to be excessively high. All of these competing forces make this problem ever more challenging, as any compromise on either the quality or correctness will directly affect the reputation and trustworthiness of the supplier company. As evident from Fig. 4.1, today's hardware (HW) product is a combination of IPs and processes, both of which can be coming from untrusted sources. IPs are realized in HW either in the form of integrated circuits (ICs) or high-level description language (HDL) design files. In this chapter, the terms IPs and ICs have been used interchangeably.

Business relations are driven by political and socioeconomic factors. A third-party component with a malicious intent may introduce subtle alterations in the final design to make it work contrary to the normal in special or rare situations. Under the presence of such malicious modifications (which may be introduced either in the design phase or in the manufacturing phase), the design is just like a ticking time bomb waiting for the trigger to cause malicious outcomes. A traditionally faulty or defective part does not pose as a serious threat in the sense that most of

them can be detected during the production testing and/or the post-silicon validation phase. For those that escape testing, a random operational failure is unlikely to leak secret information of the chip or cause a disastrous consequence. This is because manufacturing defects lack intelligence in their intent. To the contrary, an intelligently tampered design is really severe to handle. They are inserted for a specific purpose. Once triggered off, it can cause catastrophic consequences.

Such malicious intelligent alterations in the design that can make it behave contrary to the expected under special situations are called as Hardware Trojans. Hardware Trojan insertion may include addition of extra logic, addition of extra wires to make undesired connections, removal of existing logic, or even varying the design parameters and/or geometries.

4.2 The Beginning

In an effort to mitigate the risk arising from structural and technical vulnerabilities of third-party IPs, Defense Advanced Research Projects Agency (DARPA) introduced the "Trusted Foundry Program" in 2007 [1]. The ICs used by Department of Defense (DOD) are very specialized ones. Domestic industry and national labs such as SANDIA, NSA and Honeywell, etc. may not be able to provide the performance, volume, cost, and variety of DOD needs. Consequently, they need to be designed/manufactured by parties/foundries which are not under direct surveillance from the national security agency. Thus, DARPA came up with these hard to answer questions:

- How does one trust chips when they are designed or manufactured in a non-trusted facility, such that they will faithfully perform only the function they are designed for?
- How does one ensure that the testing on the chips will guarantee that it will operate only as designed (no more – no less)?
- How does one ensure that the packaging of the chip does not introduce features into or misidentify the chip?
- How does one determine that the packaged chip has not been tampered with after installation, and how does one communicate the fact of tampering?

4.3 Trojan Characteristics

1. Trojans are minuscule when compared to the size of the design in which they are implanted. They can be very well accommodated within the non-used area of the IC thereby keeping the IC dimensions intact even with their presence. Consequently they do not incur any silicon area overhead on the die. Furthermore, since they are activated by internal signals and inject their effect

in some other internal signals, they do not require additional input or output for taking effect. Hence they do not increase the pin count either.
2. They are stealthy in nature. Activation of a Trojan generally depends on specific or very rare triggering scenarios that are difficult to produce during conventional testing. Even scan-based testing cannot expose the Trojan because the triggering condition is unknown, and it would be prohibitively expensive to exhaustively exercise the circuit. Furthermore, some Trojans may require a sequence of stimuli to activate and propagate to some output, which means they have internal state machines driven by flip-flops. In all likelihood, those flip-flops are created by the intrusion and hence are not a part of the scan chain, making them difficult to be controlled.
3. They are intentionally malicious. Triggering of an active Trojan affects one or more output(s) of the IC, changing the desired behavior of the device in a way such that the end result could be detrimental. A passive Trojan, on the other hand, does not affect the functionality of the device. Instead, it may transmit secret information to an adversary [2] who can use it to deter the performance of the device or may induce some physical changes like excessive heat generation to make the device fail.
4. The Trojans may be dormant for the most part of their life. Under normal operating conditions of the IC, active Trojans do not interfere with the logical behavior of the device unless the triggering condition arises. So there is no difference in the outputs of a tampered IC from that of a genuine one.

4.4 Inadequacies of Existing Testing and Security Features

IP theft has always been viewed as a problem, and so the techniques to mitigate the risk of such incidents have been explored. One of such technique is *Watermarking*, which entails sculpting the information directly and imperceptibly into the original data. Watermarking is employed for copyright protection, distribution tracing, authentication, and authorized access control [3]. More recently, processing technology variations have been exploited to characterize an IC. This is analogous to the fingerprint of the device. Since reproducing the exact physical conditions for creating a clone of an IC is virtually impossible, this technique is called as *Physically Unclonable Function* (PUF) protection technique [4–6]. When a physical stimulus is applied to the PUF structure, it reacts in an unpredictable and unique way due to the presence of randomness [7–9]. For example, the delay of two or more identical ring oscillators will be different across the chip. These can be used to construct PUF circuits as the races between the various ring oscillators offer a unique pattern of the chip. The applied stimulus is called the challenge, and the reaction of the PUF is called the response.

Conventional methods of testing ICs involving *scan chains* and *BIST* modules are not robust enough to detect intelligent intrusions or alterations in the design. This is because design-for-test techniques are architected keeping the functional logic

in mind. Anything that is beyond the expected functionality cannot be adequately checked. The universe of all possible insertions is theoretically infinite. Hence checking all input combination requires exhaustive pattern simulation. Even if the scan capability allows full controllability of all flip-flops, the prohibitively large number of inputs makes it infeasible to adopt such a test approach. In addition, owing to the fact that these malicious intrusions are rarely activated except for under very special conditions, the probability of accidental detection with a given test set is almost nil. As a result, new methods for testing, validating, and assessing these intrusions are needed.

4.5 Trojan Classification

Trojans can be classified in many different ways. Trojans can be distinguished based on their physical characteristics, activation characteristics, or action characteristics [10].

4.5.1 Trojan Classification Based on Physical Characteristics

Physical characteristics can be based on type, size, distribution, and structure (Fig. 4.2):

- *Type*: There are two subcategories in type section. *Functional* Trojans include alterations inflicted on the original structure of the design which can include addition or deletion of gates from the circuit. *Parametric* Trojans involves

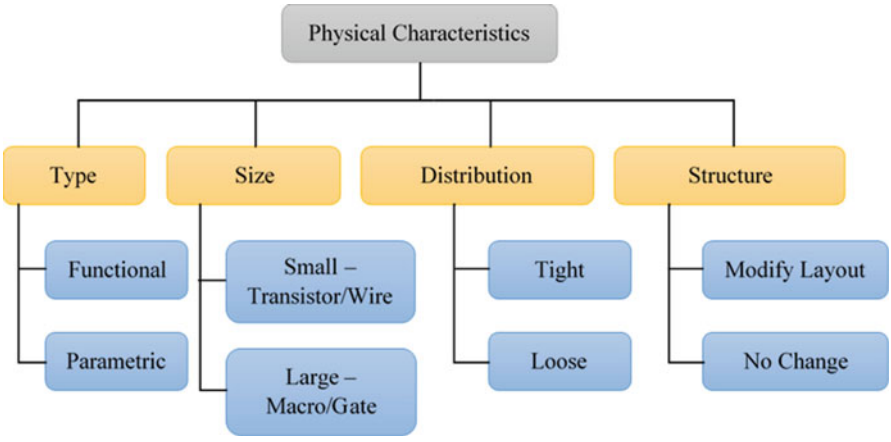


Fig. 4.2 Trojan classification based on physical characteristics

modifications of existing wires and logic normally intended to detrimentally affect the performance of the IC at runtime. This can include thinning of interconnect wires, the weakening of transistor strength by varying the length-width ratio to mention a few.

- *Size*: This refers to the actual silicon real estate consumed on the die by the Trojan. Size can be an important factor for Trojan activation and detection. *Large/macro*-Trojans are relatively easier to detect at lower frequencies because of higher leakage current consumption resulting in a power profile variation that is more easily observed [11]. On the other hand, *small transistor/wire* Trojans may be easier to activate because they require less constrained conditions to trigger them, but they add negligibility to any physical parameter difference in the design. So physical parameter profiling-based techniques may not prove effective for detecting them.
- *Distribution*: This refers to the location(s) of Trojan on the IC. While a *tight* Trojan consists of a few gates topologically coalesced together in a localized area, *loose* Trojan consists of gates distributed all over the circuit or portions of a circuit. Flexibility of Trojan fabrication depends on the available space in the original layout. Hence a malicious third-party manufacturer has to choose a proper distribution of the required components to design the Trojan.
- *Structure*: Changing the structure of the IC affect the power, delay characteristics of the device. Structural changes especially that leading to a *layout modification* is very difficult to achieve. Thus, to incorporate such changes which inevitably requires the layout to be reconfigured, the adversary is likely to use a Trojan with a very small physical footprint. Physical footprint is measured in terms of the area, power consumption, delay characteristics, and other such factors. Since for a functional Trojan size and distribution have significant impact on the original footprint of the Trojan, for larger Trojan sizes, distributing the components across the layout can assist in reducing the impact on the power and delay characteristics thereby making it stealthier. Trojans which require *no change* in the original layout uses the existing spare cells in the original design to fabricate the Trojan circuit in which case there will be no change in the original circuit structure.

4.5.2 Trojan Classification Based on Activation Characteristics

Activation characteristics refer to the conditions which trigger the Trojan toward its objective. Broadly speaking, there are two types of Trojans based on activation (Fig. 4.3):

- **Externally Activated Trojans**: These Trojans are those whose triggering conditions are controlled using the input pins of the IC. Thus, it is on the part of the operator as to when he/she wants to trigger the Trojan. This can be done by monitoring the device conditions using a side-channel signal that transmits the internal information of the device and using that information to appropriately trigger the device.

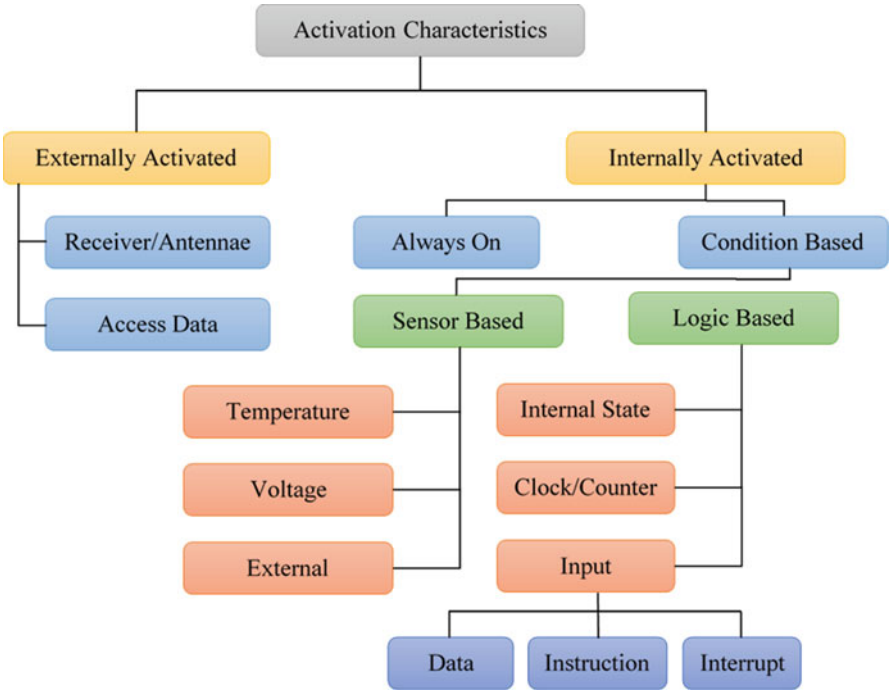


Fig. 4.3 Trojan classification based on activation characteristics

- **Internally Activated Trojans**: These Trojans monitor the internal configuration of the system for its operation, i.e., they derive their condition for activation from the existing internal environment. Internally activated Trojans are subdivided into two categories:
  - *Always-on* Trojans are perennially active and can get triggered any time inside the operating device. This includes Trojans like thinning of transistor wires, changing the drive strength of the transistors by tampering their length-width ratio, etc. A device with such change may start working exactly as a normal device, but it will fail whenever the internal conditions exceed the physical threshold supported by the fabricated device. Thus, their failure cannot be accurately predicted, but only a statistical probability of occurrence of such a failure can be estimated.
  - *Condition-based* Trojans are much more intelligent. They wait for the circuit to enter a specific configuration or the state bits to attain a specific value to get activated. Condition-based Trojans are further classified based upon the conditions that trigger them. Thus they can be:
  - *Sensor-based* Trojans whose activation depends on values/thresholds of some physical parameters like temperature, voltage, or any types of external environmental conditions like pressure, humidity, and electromagnetic interference that is monitored by the sensor.

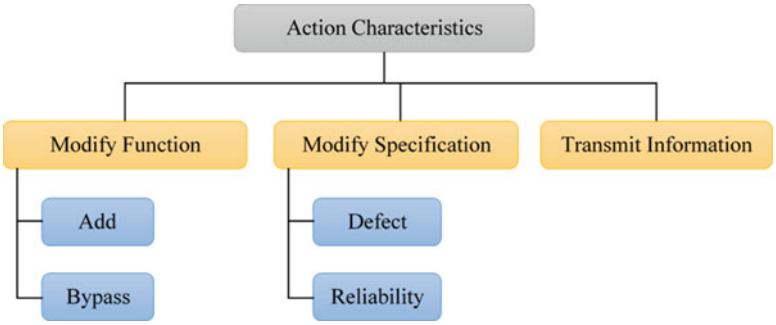


Fig. 4.4 Trojan classification based on action characteristics

– *Logic-based* Trojans are those where the logic inside the Trojan intelligently monitors the internal circuit environment to get triggered. Examples of logic-based Trojans are counter Trojans, sequence-detector Trojans, etc.

4.5.3 Trojan Classification Based on Action Characteristics

Action characteristics describe the effect of triggering of the Trojan on the underlying design. These are of three types (Fig. 4.4):

- *Modify Function*: These Trojans change the original functionality of the logic. This can imply *removal* of a portion of the logic to remove a property, *disablement of* some functionality to cause an operational failure, or *addition* of extraneous logic to realize something additional to what is intended.
- *Modify Specification*: This class of Trojans changes the properties of the chip such as delay to realize their intended objective. These are similar to parametric Trojans discussed earlier. Parametric Trojans tamper the strength like the fan-out supporting capability of an output of a gate, ability to supply a desired current through a wire, etc. Changing the strength of the wires or gates affects the delay of the combinational path and hence falls within this category.
- *Transmit Info*: This type of Trojans doesn't interfere with the operation of the device. It has been proved that side-channel signals can be decrypted to reveal important internal information embedded within the device. Trojans under this class emit signals containing such key information. This information can be misused by an adversary.

4.6 General Trojan Mitigation Techniques

Just like any hard to solve problems, Trojan detection is also a problem with no single silver bullet. A lot of research has already been conducted to solve the problem from different angles. Broadly speaking, these efforts can be divided into two categories:

4.6.1 Preventive Techniques

Preventive techniques focus on taking precautionary measures on the design itself such that its alteration due to unintended insertions becomes more difficult, or the alterations are relatively easier to detect. An example of preventive technique is modifying the design to include special testing capabilities that can make the Trojan effect either visible at the primary outputs or much more pronounced under an altered testing condition by changing the intended mode of operation in an unpredictable way [12, 13]. Another preventive technique is to put redundant logic inside the design to make it hard for the adversary to figure out the actual functional mode of the design [14]. The subsequent subsections will describe these methods in detail.

4.6.1.1 Design for Trust (DFT)-Based Techniques

One of the DFT-based techniques for identifying Trojan insertion in IPs is the voltage inversion technique [12]. Voltage inversion simply means that the power (V_{DD}) and ground (GND) sources for a set of gates are connected in the reverse way. Although sounding odd, a little understanding of device physics shows that it is possible the only reason it is not done is because the output voltage of a gate under such a connection does not swing rail to rail [15]. The Trojan gate would normally be inactive because the input condition to trigger the Trojan gate is difficult to achieve. So, the gate(s) affected by the Trojan output is not as much visible. When the voltage inversion is applied to any gate, its functionality complements. The complemented Trojan gate output becomes controlling value for the gate(s) in which the Trojan injects its effect. As a result, the effect of the Trojan gate is more pronounced on the rest of the circuit.

In [13], both output values of a flip-flop are utilized to introduce new states in the design. Conventionally, a D flip-flop has two outputs, Q and Q bar. Generally, only one output is used (Q) to drive the state machine embedded in the design. Since the original design is based on a specific logic, the finite-state machine realized by the circuit is defined. The constraints imposed by the state machine make it hard for the Trojan circuit to make itself impervious to an otherwise unexpected state. If the state space is increased during test mode, the chances of activation of the Trojan are

also increased. In the original functional mode, the activation of the Trojan is rare. With the addition of new states that are now possible, the excitation behavior of the Trojan changes.

4.6.1.2 Obfuscation-Based Techniques

In [14], the design is obfuscated by hiding the real functional mode with a starting FSM. Until the design receives a specific input sequence that can transition the internal state of the starting FSM through predefined stages, it would not enter the real operating mode. Such obfuscation incurs some area overhead, but it blurs the design in a way that the adversary cannot tell which portions of the design belong to the functional mode and which ones are the intentional obfuscation logic. This helps prevent the insertion of Trojans in two ways. First, it deters the adversary from discovering the logic behind the design. Second, it decreases the chance that any Trojan will adversely modify the core logic of the circuit.

4.6.2 Detection Techniques

Detection techniques focus on devising testing mechanisms to non-destructively detect the presence of Trojan after it has been implanted. These techniques base their premise on exaggerating and measuring one/more of the physical parameters of the design which may include delay profile [16, 17], functional power consumption [18, 19], or leakage power consumption [20].

4.6.2.1 Delay-Based Detection Techniques

In [16], negatively skewed clock has been used to characterize the delay of the functional paths. Any combinational path in the circuit is contained between a *source* register and a *destination* register. To characterize the delay signature of this path, a third register called as the *shadow* register is kept in parallel to the *destination* register. But unlike the *source* and *destination* registers which are clocked by the system clock, the *shadow* register is clocked by a different clock which is skewed negatively with respect to the system clock. The output of the *destination* and the *shadow* registers is compared, and the result bit is set to 1 if the outputs differ. For a design that is infected with Trojans, this failure (setting of the result bit to 1) starts happening at a much lesser skew that a genuine design because of the extra delay introduced by the Trojan logic. The experiment is repeated with sufficient number of genuine parts, statistical average of the skew gives a fairly accurate estimate of the expected skew difference, and a combinational path should support and hence can be used as a signature to distinguish it from tampered parts.

In [17], a genuine circuit netlist has been used to produce a high-coverage delay fault test set. The effect of each of these patterns is observed at each output. The experiment is repeated with a sample set of test ICs which are then destructively tested to assure they were indeed genuine.

Based on the data obtained from the experiments, the points are plotted on a three-dimensional chart to construct the convex hull. Once the hull is ready, if the delay profile points for an IC that mostly lie outside the convex hull, it indicates a suspicious behavior, and the IC is likely to be infected with Trojans.

4.6.2.2 Power-Based Detection Techniques

One of the earliest works attempting to detect a Trojan inserted into a circuit was done by DARPA [11]. The transient power drawn during the partial/full excitation of the Trojan was used as the parameter for comparison against the trusted reference model. A set of random vectors was used to simulate the design and capture its overall power consumption. Results provided in the work reveal that the portion of power consumption contributed by gates in the Trojan accounts for a very small difference in the overall circuit power consumption. Hence it can easily be submerged within process variation, concluding that detecting an intelligent malicious intrusion externally is a challenging task and more sophisticated methods are needed for this.

In [18], a "vector set" is sustained at the circuit input for a specific number of clock cycles before it is changed to different ones. This idea leverages the fact that circuit activity is a combination of changing inputs/states or both. Sustaining the inputs at a constant value for multiple clocks forces the circuit activity to be generated only from the changing internal state (because the inputs are held steady). At the same time, the synergic effect of both the inputs and state changing at the same time is also diminished. As a result, this approach nicely minimizes the total circuit activity which proves instrumental in exaggerating the activity difference between the original and the infected designs. Because the inserted Trojan is tiny, the power profile altered by the added Trojan may also be small. Thus, it is important to keep the total power consumption low in order to see a clear difference between a genuine and the compromised part. However, it must be ensured that the power is not reduced so low that the chip enters a sleep state.

In [19], the original circuit is partitioned into regions. A power consumption profile for each region is created by simulating the design with a test set that can selectively increase the switching activity in the intended region while attempting to keep the activity in other portions of the circuit as low as possible. The idea is to exaggerate the localized activity going on within the region as compared to the overall design. If a Trojan (or its part) belongs to the region under observation, the difference between the power profiles of the golden reference design and the tampered design would be enhanced beyond process variation, thereby giving an indication of the intrusion.

4.6.2.3 Charge Consumption-Based Detection Techniques

In [20] a current integration technique has been used to compute the total charge consumed by the circuit over a period of time. This can be the leakage current if the gates in the Trojan are not toggling or the dynamic current if a portion of the gates are toggling. First a golden charge consumption profile is obtained by testing a set of genuine ICs. Then the effect of process variation is included to reflect the change in the charge consumption curve. Finally, the IC under the test is subjected to the same testing mechanism, and if the charge consumption curve differs from the golden one by a threshold value, the IC is declared to be malicious. So if the chip is tested from the global power pads, chances are there that the process variation will overshadow the minute additional current consumed by the Trojan. To avert this problem, it has been suggested to test the chip from different power pin simultaneously, each of which is dedicated to a particular region inside the IC.

4.7 Trojan Mitigation at IP Level

Methodologies have been proposed for detecting Trojans at IP level that leverage one or more of the concepts explained in Sect. 4.6. A couple of such methods are discussed below.

4.7.1 Detection Technique: Suspect Signal-Guided Sequential Equivalence Checking

Malicious insertions in soft IPs in the pre-silicon stage are beyond simple, direct recognition. This is because IPs are normally developed from a user supplied or a publicly available specification. Final layout-/gate-level netlist obtained for such a specification need not be necessarily unique. Design stages convert this specification into behavioral code (Verilog), then into gate-level netlist (Register-Transfer Logic or RTL), and finally to physical layout using synthesis. The toolchain used, the optimization constraints specified for converting the gate-level netlist to physical layout, the physical gate library used to map the netlist in layout, has a profound impact on how the final output looks like. A highly constrained area optimization would result in usage of large fan-in complex gates that would consume more power. On the other hand, a power-optimized realization would use smaller gates and would result in a larger die area on silicon. In either case, the final netlist/layout would look much different than each other.

This makes the possibility of inserting a stealthy malicious alteration in soft IPs very likely, and unless otherwise tested intelligently and rigorously, chances are they would escape the conventional testing flow. Since these alterations are stealthy, from

a fault detection point of view, one of the faults (either stuck-at-0 or stuck-at-1) on such nodes is very hard to be detected. For a Trojan to be effective, it should be able to propagate its effect to one/more primary outputs. These two assumptions serve as the premises for the work described in [21]. This work attempts to address this problem of checking the suspicious netlist (referred to as *sus circuit*) against an original behavior (referred to as *spec circuit*). Since straightforward recognition of alterations is unlikely, the problem maps to performing a sequential equivalence checking (SEC) between the *spec circuit* and the *sus circuit*. However, in most of cases, such a full-blown SEC approach is infeasible. Hence, intelligent constraining at the inputs is required to narrow down the search space for the design.

In the approach proposed in [21], the easy-to-detect faults are first removed from the third-party IP (3PIP) using functional vectors followed by an optional N-detect full-scan automatic test pattern generation (ATPG) to identify those fault which are functionally hard to excite and/or propagate. These identified faults are then processed by a special SEC setup to compare behavioral consistency between the original design and the 3PIPs. Finally, a region isolation approach is applied on the filtered faults to map the possible location(s) of the insertion(s). The methodology consists of four steps as described below:

4.7.1.1 Step-I: Functional Vector Simulation

In the first step, functional vectors are used to detect and drop as many stuck-at-faults as possible from the *sus circuit*. These easy-to-excite/observe signals can be omitted for future consideration. Since Trojan(s) are generally hard to excite and propagate (even in the full-scan mode), it is highly unlikely that functional vectors would detect them, else the functional vectors themselves serve as a witness to differentiating the two circuits. Those signals corresponding to faults undetected by functional vectors are categorized as suspect candidates.

4.7.1.2 Step-II: N-Detect Full-Scan ATPG

In a conventional full-scan mode, the state bits which were otherwise controlled by the internal nets become fully controllable. This expands the state space. Hence it is possible to achieve multiple full-scan vectors to detect the same stuck-at-fault. Trojans have hard-to-achieve activation conditions because a Trojan is a very specific logical interconnection and one/few specific full-scan vectors can set its output to the triggering value and make its effect propagated to an observable point. Consequently, if multiple unique vectors can excite and propagate fault on a signal to a primary output, it is highly unlikely to correspond to a Trojan.

Nevertheless, an unconstrained initial state in the full-scan mode allows for functionally illegal states making Trojan observable at the output by some vectors which are not functionally reachable. Thus, an N-detect full-scan ATPG has been used instead of selecting faults which are uniquely excitable in full-scan mode.

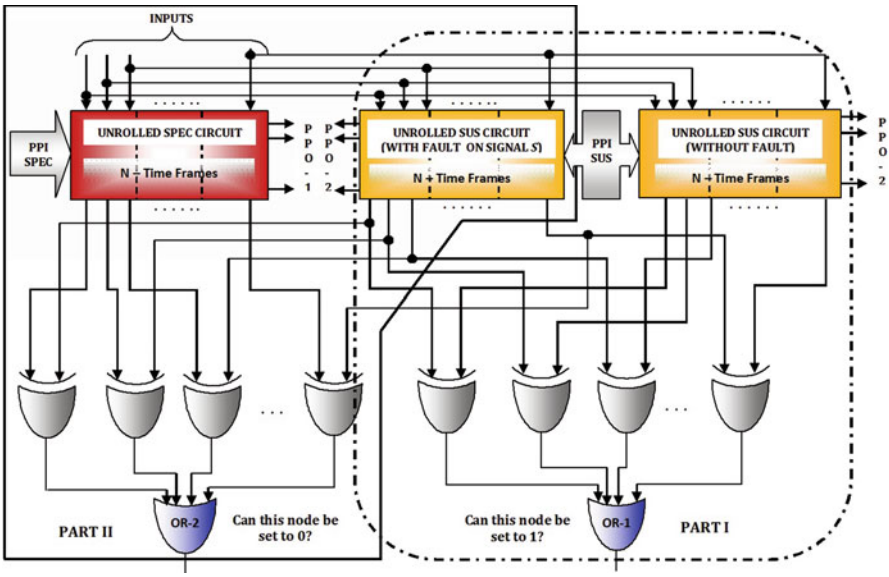


Fig. 4.5 Miter circuit setup for diluted SEC

Combinational untestable stuck-at-faults are removed from the suspect candidate list in this step. The threshold N is suitably chosen. Keeping N too low removes the Trojan-related signals (faults), whereas making it high results in a large set of suspect candidates to process in the next step.

4.7.1.3 Step-III: Diluted Sequential Equivalence Checking

This step consists of two conjoining parts as represented in Fig. 4.5.

The first part selects a stuck-at-fault from the list of suspected candidate and ensures that it is observed at primary output. This is realized with the miter shown in Fig. 4.5 Part-I. It contains two instances of the *sus circuit*, each of which has been unrolled a specific odd number of time frames. The fault on S is injected in the central time frame of one of the instances. The initial state (PPI SUS) as well as the INPUTS is left unconstrained. The effect produced by S is captured at the output of the miter (gate OR-1). The pseudo-primary outputs (PPO-2) are not considered for comparison to ensure that the selected signal specifically affects the output.

Three advantages of this setup are (1) untestable faults over multiple frames are dropped; (2) successive states appearing at the internal state boundaries converge toward reachable state space; and (3) faults which are excited but not propagated to the output within the given unroll depth are also discarded thereby further pruning the suspect candidate list. Although a completely unconstrained initial state will eventually allow a portion of the unreachable state space to penetrate the successive states, this does not discard any functionally possible state.

In the second part, the *spec circuit* is unrolled with the same number of time frames as that of the *sus circuit* (shown as N in Fig. 4.5 Part-II). The outputs of the *spec circuit* are constrained to have the same value as that in the *sus circuit* by forcing the output of the miter in Part-II to 0 (OR-2). The primary inputs (INPUTS) are thus constrained to a sequence that activated and observed signal S in *sus circuit* (obtained from the miter setup in Part-I) effectively checking if the *spec circuit* can produce the same behavior as the *sus circuit* when signal S has been activated and observed in the *sus circuit* within the given unroll bound.

This combined two-part setup is converted into a CNF instance and given to the SAT solver. If the solver returns a SAT result, nothing can be concluded about the signal S because we don't know the reachability of the state in the initial time frame in the spec circuit. However, if it is UNSAT, it can be due to two reasons. Either it is due to the effect of the Trojan and conditions produced by it in *sus circuit* that makes it trigger (which is absent in the spec circuit) or due to the effect of an illegal state at any time frame in *sus circuit* for which no corresponding state exists in the spec circuit. But with unrolling the state illegality factor diminishes, and so it is more likely that an UNSAT solution arises due to the effect of the Trojan. Results show that for the cases where the SAT solver could not solve the instance, majority of the signals belonged to the Trojan.

4.7.1.4 Step-IV: Infected Region Isolation

This is the final step in the approach. It aims to isolate the region(s) potentially infected by the Trojan. It starts with the suspect candidate list obtained from the N-detect ATPG. In general, the suspect candidate list may contain multiple undetected faults associated with same gate. For a gate G_i (where i is the gate ID), its weight W_i corresponds to its frequency of occurrence in the suspect candidate list. The *region* around each of these suspect candidates is expanded. Since the number of such gates is small, the number of regions is also small. For a region containing n gates centered on gate i within a radius of x (denoted by $R_i(x)$), the suspect count SC_i of the region is computed by:

$$SC_i(x) = \sum_{1}^{n} W_i \mid G_i \in R_i(x)$$

The suspect index *SI* of the corresponding region centered on gate i is defined as:

$$SI(x) = \frac{SC_i(x)}{\mid R_i(x) \mid}$$

where $|R_i(x)|$ denotes the total number of gates within region centered around gate i and x is the radius of the region around gate i. *Suspect index* represents a weighted value of each gate in a region to be associated with the Trojan. For the region not

containing the Trojan, with a growing radius, the number of gates in it increases without adding to the suspect count thereby reducing the suspect index.

SSG-EC tends to remove a large number of gates because of the unconstrained initial state; hence gates from the suspect candidate list obtained in N-Detect ATPG (step-II) are selected. Notably not all the stuck-at-faults in the Trojan region gives rise to conflicts within the given unroll depth and hence tends to get removed from the suspect candidate list in step-III. The suspect candidates identified in step-II are the ones which are definitely hard to detect. So it is more likely that a region craved out of the suspect candidate list in step-II is more likely to contain the cluster of gates in the Trojan. While a hard-to-test region need not necessarily represent a Trojan, step-III helps in filtering out those regions which contains gates that show inconsistent circuit behavior among the list of regions obtained. Thus, step-III and step-IV are mutually beneficial in narrowing down the search.

In Fig. 4.6 the shaded gates are the ones associated with the Trojan. After processing the fault list through steps I and II, let the suspected candidate list contain $\{G_1, G_2, G_3, G_5, G_6, G_9\}$. The weight distribution for these gates is as follows: $W_1 = 2$, $W_2 = 3$, $W_3 = 3$, $W_5 = 1$, $W_6 = 2$, and $W_9 = 1$. All other gates which are not in the suspect candidate list receive a weight of 0. Noticeably $W_4 = 0$ (belongs

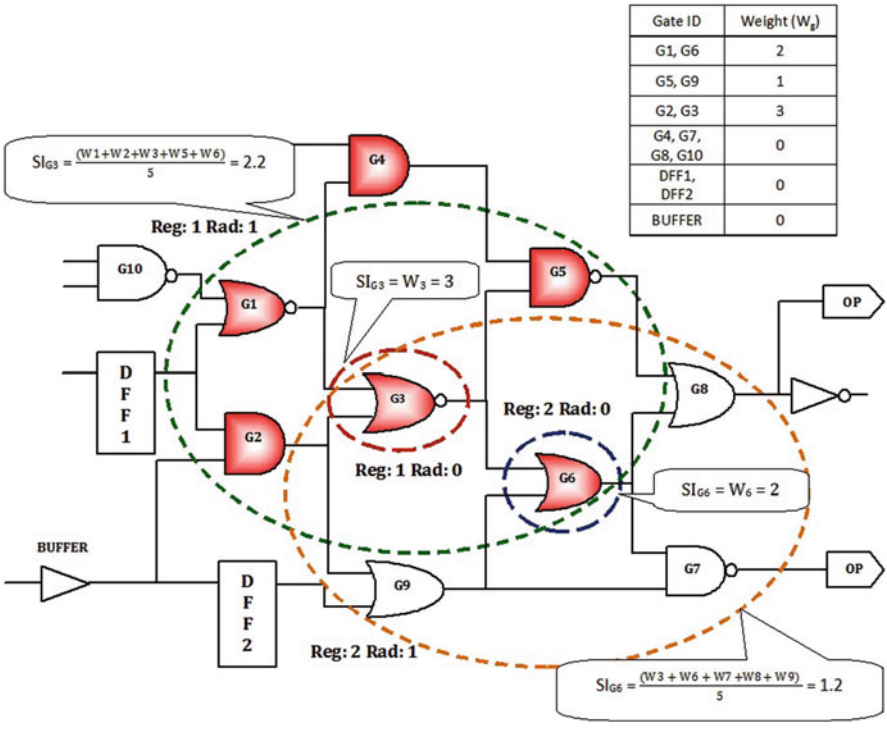


Fig. 4.6 Infected region isolation based on gate weights

to Trojan) and $W_9 = 1$ (doesn't belong to the Trojan). To explain the significance of suspect index, let's observe the effect of expanding the region around two of these candidates, viz., G_3 and G_6. A region around gate G_3 with radius of 1 unit gives a suspect index of $SI_{G3} = (W_1 + W_2 + W_3 + W_5 + W_6)/5 = 2.2$. The corresponding suspect index centered on gate G_6 with the same radius of 1 is given by $SI_{G6} = (W_3 + W_6 + W_7 + W_8 + W_9)/5 = 1.2$. Based on these values, the region around G_3 is more likely to contain the Trojan. Now, if we increase the radius around gate G_3 to 2, it will add gates G_7, G_8, and G_{10} along with DFF 1 and BUFFER without increasing the suspect count thereby reducing the suspect index to 1.1. Thus, the radius parameter needs to be balanced to get a maximum overlap with Trojan region. Experimental results show a remarkable match between the gates contained in the region with highest suspect index and those in the implanted Trojan.

4.7.1.5 Results

The results for this approach are summarized in Table 4.1. Column 1 represents the ISCAS'89 and ITC'99 benchmarks used for the experiments. The total number of collapsed stuck-at-faults in the *sus circuit* for each benchmark is given in column 2. Column 3 (step-I) represents the total number of faults detected using the functional vectors, with the remaining undetected faults in parentheses. Column 4 gives the fault count which are uniquely detected less than $N = 3$ times using the full-scan N-detect ATPG. The fifth column reports the percentage of faults associated with the Trojan portion of the circuit that are contained in the list represented by column 4. The result for Part-I of step-III is tabulated under column 6. A further reduction in the size of the suspect candidate list is achieved here because of the factors mentioned earlier: (1) non-propagation to output and (2) sequential untestability. Columns 7–10 report the result of the diluted SEC. The SAT columns contain the number of faults for which the SAT solver returned a solution; the UNSAT columns contain the faults for which the SAT solver could not solve the instance. These columns separately represent how many signals belong to the Trojan portion and how many belong to the genuine circuit. Column 11 shows the result of step-IV in our approach. The results have been tabulated for a radius of 2. For most of the ISCAS'89 circuits (except for s349) and more than half of the ITC'99 circuits, all the gates in the region that produced highest suspect index belonged to the Trojan, i.e., all gates in the region were signals associated with the Trojan.

To verify that the Trojan instances were indeed stealthy enough, two different analyses were performed. In the first experiment, functional simulation of both the *spec circuit* and the *sus circuit* starting from a known initial state was done to check that the Trojan effect is not visible at the output very easily. In the second experiment, a bounded model checking (BMC) on the *spec circuit* and the *sus circuit* at different unroll depths. The results are summarized in Table 4.2. Column 1 shows the circuit names. For both random functional simulation and BMC, time-out limit was capped at 3 hours. Column 2 shows the number of random vectors simulated on the instances in 3 hours. Columns 3 through 7 show the result of BMC on the same circuits with different unroll bounds. None of the Trojan instances were detectable

Table 4.1 Trojan detection results for ISCAS'89 and ITC'99 benchmarks

Ckt.		Step-I	Step-II		Step-III (I)	Step-III (II)				Step-IV
	# Flts Sus Ckt.	Func. vec (# Und. Flts.)	N-D ATPG (Det.<N)	*Tro* Net Cnt.(%)	Sel. for SSG-EC	SAT		UNSAT		RAd-2 % SI_{max}
						Tro	*Non-Tro*	*Tro*	*Non-Tro*	
s298	502	340(162)	65	73.8	25	18	7	0	0	**100**
s344	507	353(154)	43	88.4	17	13	2	0	2	**100**
s349	517	361(156)	7	28.6	6	2	2	0	2	25
s526	728	512(216)	91	41.8	40	12	19	0	9	**100**
s641	633	427(206)	60	93.3	34	29	2	1	2	**100**
s713	656	433(223)	41	92.7	16	13	1	0	2	**100**
s1196	1414	1259(155)	61	86.9	31	10	1	15	5	**100**
s1238	1557	1332(225)	47	76.6	9	2	0	1	6	**100**
s1423	1600	1329(271)	46	82.6	7	6	1	0	0	**100**
s3330	2878	2106(772)	89	42.7	50	1	31	0	18	**100**
s5378	4826	3496(1330)	208	18.3	132	13	110	0	9	**100**
b03	628	351(277)	51	70.6	1	0	1	0	0	**100**
b04	3308	2581(727)	117	8.5	21	10	11	0	0	0
b05	3762	1197(2565)	184	5.4	121	0	82	10	29	0
b08	576	371(205)	70	68.6	27	23	4	0	0	**100**
b09	846	2(844)	95	41.1	36	14	19	0	3	**100**
b10	672	467(205)	42	78.6	1	0	1	0	0	**100**
b11	2154	1169(985)	15	73.3	12	10	2	0	0	**61.1**
b12	4558	1287(3271)	1042	0.2	5	0	4	0	1	0
b13	622	380(242)	71	14.1	26	10	15	0	1	0

Table 4.2 Trojan detection result for ISCAS'89 and ITC'99 benchmarks using functional vector simulation and BMC

Circuit name	Functional vector sim. (in million)	BMC with different UNROLL bounds				
		10 time frames	20 time frames	30 time frames	40 time frames	50 time frames
s298	1222.9	SAT	SAT	SAT	SAT	SAT
s344	688.0	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT
s349	676.1	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT
s526	925.7	UNSAT	UNSAT	UNSAT	UNSAT	TO
s641	369.1	UNSAT	TO	TO	TO	TO
s713	354.9	UNSAT	TO	TO	TO	TO
s1196	202.9	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT
s1238	190.1	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT
s1423	241.0	UNSAT	TO	TO	TO	TO
s3330	95.0	UNSAT	TO	TO	TO	TO
s5378	76.1	UNSAT	TO	TO	TO	TO
b03	708.8	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT
b04	149.4	TO	TO	TO	TO	TO
b05	352.2	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT
b08	875.4	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT
b09	1179.7	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT
b10	543.2	UNSAT	UNSAT	TO	TO	TO
b11	255.0	UNSAT	UNSAT	UNSAT	TO	TO
b12	164.5	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT
b13	1142.2	UNSAT	UNSAT	UNSAT	UNSAT	UNSAT

by functional simulation for 3 h. In BMC almost all of the instances (except for s298) were shown to be unsatisfiable or time-out. For the rest, experimental evidences stand to the testimony of the fact that the Trojans used are not easily detectable.

4.7.2 Prevention Technique: Proof-Carrying Code

Proof-carrying code (PCC) tries to address the issue of trust on 3PIPs by making the consumer and the supplier agree on the using a proof-based language for representing critical safety properties of the design and optionally also the design. At the time of final delivery of the product, the supplier needs to provide the formal properties along with the soft-IP to the consumer. The formal properties relate to checking of critical circuit functionalities that cannot be compromised. These can include disruption of operation, manipulation of signals, and misuse of sensitive data to name a few. The IP vendor provides the proofs in the formal language using temporal logic, whose correctness can be checked by the consumer having the knowledge of the language syntax and semantics. While the implementation of a design may not be directly visible, the proofs are written in the agreed formal language and can be read and interpreted by someone having a background on the same. Thus, the consumer knows that the IP supplier is indeed proving the intended properties on the design. The formal language introduces a new semantic model for any hardware description language (HDL) in the theorem-proving platform, which facilitates tracking and proving security properties of the design.

For this to be feasible, the Verilog design model needs to be converted into the formal semantics so that the proofs can be run on it. The IP supplier has to either deliver a formal IP model or the soft-IP can be converted into the formal representation using a conversion tool [25] which is provided by a trusted third party, (understanding the fact that the IP supplier may not be apt in using the formal language). In [22], this formal language is called as Coq. The Verilog-to-Coq conversion is from a trusted third party that all users can trust. So if the Verilog source is tampered, the Coq generated will not model the temporal properties (also inspected and verified by the consumer) correctly. The IP supplier needs to make sure that the internal implementation of the soft-IP does not violate any of these properties. The process of creating the design and providing proofs for its data secrecy properties is shown in Fig. 4.7.

This technique offloads the burden of ensuring the trustworthiness to the supplier. As long as the consumer can make sure that the properties supplied by the vendor are correctly modeled in temporal logic and are as intended in the original specification, it is easy to check the correctness of the design by executing these properties on it. These properties (also referred to as proofs) are written in Coq. The final design with an inbuilt ability to prove its correctness on the stated properties has been called as *proof-carrying hardware* (PCH). This approach converts each Verilog construct into an equivalent formulation followed by creation of *proofs* in Coq.

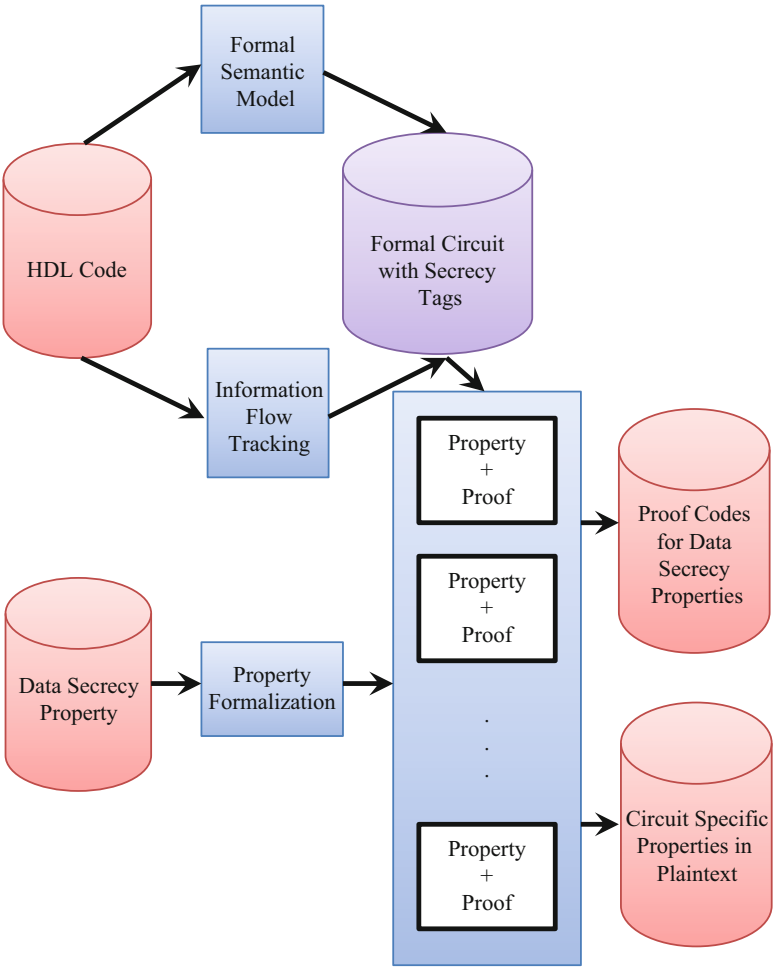


Fig. 4.7 Property generation and proof declaration flow

4.7.2.1 Formulating Verilog for Safe Hardware

In [22, 23] Verilog was selected as the HDL to convert into the corresponding declaration in Coq. The syntax below shows the rules for such a translation. It includes signals, expressions, operators, logic, and module declaration and instantiation.

Signal Definition

Signal values are defined in an inductive set with two values, viz., `lo` and `hi`. Every single- or multibit signal is treated as a `bus_value`. The `bus` is defined as a list of values representing the `bus_value` at any specific time represented by clock cycles and given as a natural number.

```
Inductive value := lo | hi.
Definition bus_value := list value.
Definition bus := nat -> bus_value.
```

Signal Operations

The semantic model of `bus`-handling methods include logical operations, such as `and`, `or`, `xor`, etc., as well as comparisons such as checking for bus equality, `bus_eq`, less than comparison, `bus_lt`, etc. Special functions to compare the bus value with 0, `bus_eq_0` also exists.

```
Fixpoint bv_bit_and (a b : bus_value)
{struct a} : bus_value := match a with
| nil => nil
| la :: a' => match b with
| nil => nil
| lb :: b' => (and la lb) :: (bv_bit_and a' b')
end
end.
Definition bus_bit_and (a b : bus) :
 bus := fun t:nat => bv_bit_and (a t) (b t).
Fixpoint bv_eq_0 (a : bus_value)
{struct a} : value := match a with
| hi :: lt => lo
| lo :: lt => bv_eq_0 lt
| nil => hi
end.
Definition bus_eq_0 (a : bus) (t : nat) : value :=
bv_eq_0 (a t).
```

Signal Operations

An expression is defined as an inductive set with operators to construct new expressions or combine expressions together. The `econv` and `econb` constructors directly convert a constant value list and a `bus`, respectively. The logical AND, OR, and XOR operations combining two expressions are performed by `eand`, `eor`, and `exor` constructors, respectively.

```
Inductive expr :=
| econv : bus_value -> expr
| econb : bus -> expr
| eand : expr -> expr -> expr
| eor : expr -> expr -> expr
| exor : expr -> expr -> expr
| enot : expr -> expr
| cond : expr -> expr -> expr -> expr
...
```

Expressions are recursively evaluated to compute its value at any instant of time and returned as `bus_value`.

```
Fixpoint eval (e : expr) (t : nat)
{struct e} : bus_value := match e with
| econv v => v
| econb b => b t
| eand ex1 ex2 => bv_bit_and (eval ex1 t) (eval ex2 t)
| eor ex1 ex2 => bv_bit_or (eval ex1 t) (eval ex2 t)
| enot ex => bv_bit_not (eval ex t)
| cond cex ex1 ex2 => match (bv_eq_0 (eval cex t)) with
| hi => eval ex1 t
| lo => eval ex2 t end
...
```

Coq Semantic Model

The Coq semantic model comprises of semantic model of signals and expressions. The output signals of the module is represented by `outb` and input signals by `inb`. `wireb` denotes internal wire signals, while `regb` stands for internal registers. The `assign_*` is used with combinational logic, while `nonblock_assign_*` is used for non-blocking assignment in sequential logic. The selection of the';' mark is consistent with the syntax of other HDLs.

```
Inductive code :=
| outb : bus -> code
| inb : bus -> code
| wireb : bus -> code
| regb : bus -> code
| assign_ex : bus -> expr -> code
| assign_b : bus -> bus -> code
| assign_case3 : bus -> expr -> code
| nonblock_assign_ex : bus -> expr -> code
| nonblock_assign_b : bus -> bus -> code
| codepile : code -> code -> code.
Notation " c1 ; c2 " := (codepile c1 c2)
(at level 50, left associativity).
```

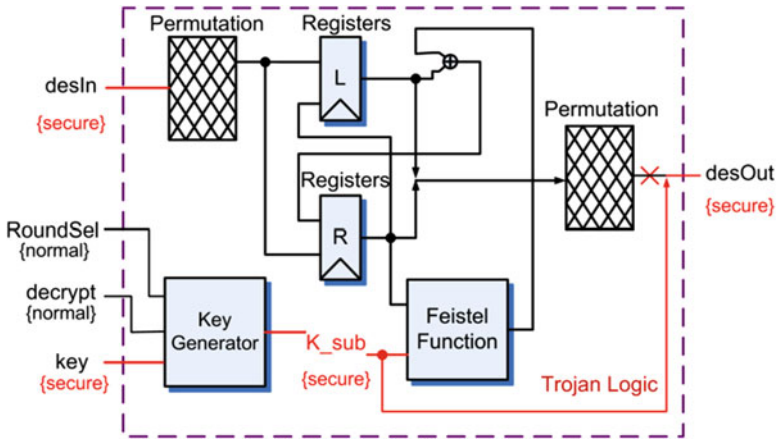


Fig. 4.8 Trojan-affected DES circuit

4.7.2.2 Constructing Proof in Coq

Due to the close resemblance of Coq semantic model with Verilog, the only conversion rule that needs to be obeyed during Verilog-to-Coq conversion is to keep the resultant code structurally the same as the source. Details about Coq proof assistant platform and the syntax of Coq language can be found in [24]. A combinational assign logic is mapped to an `assign_ex` statement, and module instantiation is mapped to `module_inst` statement as shown below.

```
 Verilog code: assign Lout = (roundSel == 0) ?
IP[33:64] : R;
 Converted Coq formal logic: assign_ex Lout (cond (eq
(econb roundSel)
(econv (lo::lo::lo::lo::nil))) (econb (IP @ [33, 64]))
(econb R));
 Verilog code: crp u0 (.P(out), .R(Lout),
.K_sub(K_sub));
 Converted Coq formal logic: module_inst2in out Lout
K_sub;
```

4.7.2.3 Trojan Detection Using PCC

The Trojan in the DES circuit shown in Fig. 4.8 bypasses the internal round key directly to the output. In the scheme discussed, Trojan detection is independent of the triggering condition, since it happens through a formal theorem-proving approach, rather than actual application of stimuli to the circuit. When the Trojan-

infested HDL code is converted into the coq formal model, and the proofs are checked against it, the `no_leaking_des` theorem could not be proven (`desOut` comes out to be of secure tag), thereby indication the presence of the Trojan.

4.8 Conclusion

This chapter provided a high-level overview of "third-party IP trust issues" and its possible consequences. As outsourcing continues to be the mainstay for remaining competitive and profitable in the IC design industry, preventing/detecting Trojans in HW components is emerging as a critical and exciting area of research. With a modern HW platform comprising of multiple IPs from different vendors, it becomes a matter of paramount importance to ensure that the integrity of the parts or the product as a whole has not been compromised in any way at any stage in the design flow. Such compromise may include unwanted tampering leading to targeted operational failures, leakage of secret information leading to IP theft, compromising on the expected operational life of the product by altering physical design parameters, etc. Toward the latter half of the chapter, we discussed some of the research that has been done in devising nondestructive ways of preventing as well as distinguishing such tampered parts from the genuine ones. A couple of methodologies that have been proposed for detecting Trojan implantation at IP level were discussed in detail. It is a requirement for the industry today where that the final product supplier can adopt such test methodologies to distinguish a tampered third-party IC from a genuine one in order to mitigate the aforementioned risks.

References

1. D. R. Collins, TRUST, A proposed plan for trusted integrated circuits, Government Microcircuit Applications and Critical Technology Conference, 2006, pp. 276–277
2. S. Adee, The hunt for the kill switch. IEEE Spectr. **45**(5), 34–39 (2008)
3. S. Voloshynovskiy, S. Pereira, T. Pun, J.J. Eggers, J.K. Su, Attacks on digital watermarks: Classification, estimation based attacks, and benchmarks. IEEE Commun. Mag. **39**(8), 118–126 (2001)
4. B. Gassend, D. Clarke, M. van Dijk, S. Devadas, Controlled physical random functions, IEEE Computer Security Applications Conference, 2002, pp. 149–160
5. J. Guajardo, S.S. Kumar, G.J. Schrijen, P. Tuyls, Physical unclonable functions and public-key crypto for FPGA IP protection, IEEE International Conference on Field Programmable Logic and Applications, 2007, pp. 189–195
6. V. Vivekraja, L. Nazhandali, Circuit-Level Techniques for Reliable Physically Uncloneable Functions, IEEE International Workshop on Hardware-Oriented Security and Trust, 2009, pp. 30–35
7. A. Maiti, R. Nagesh, A. Reddy, P. Schaumont, Physical unclonable function and true random number generator: a compact and scalable implementation, IEEE/ACM Great Lakes Symposium on VLSI, 2009, pp. 425–428

8. A. Maiti, P. Schaumont, Improving the quality of a Physical Unclonable Function using configurable Ring Oscillators, IEEE Int Conf Field Program Logic Appl, 2009, pp. 703–707
9. S. Morozov, A. Maiti, P. Schaumont, An Analysis of Delay Based PUF Implementations on FPGA, International Symposium on Applied Reconfigurable Computing, Lecture Notes in Computer Science, 2010, pp. 382–387
10. X. Wang, M. Tehranipoor, J. Plusquellic, Detecting Malicious Inclusions in Secure Hardware: Challenges and Solutions, IEEE International Workshop on Hardware Oriented Security and Trust, 2008, pp. 15–22
11. D. Agarwal, S. Baktir, D. Karakoyunlu, P. Rohatgi, B. Sunar, Trojan detection using IC fingerprinting, IEEE Symposium on Security and Privacy, 2007, pp. 296–310
12. M. Banga, M. Hsiao; VITAMIN: Voltage inversion technique to ascertain malicious insertions in ICs, IEEE International Workshop on Hardware Oriented Security and Trust, 2009, pp. 104–107
13. M. Banga, M. Hsiao, Odette: A Non-scan design-for-test methodology for Trojan detection in ICs, IEEE International Workshop on Hardware Oriented Security and Trust, 2011, pp. 18–23
14. R.S. Chakraborty, S. Bhunia, Hardware protection and authentication through netlist level obfuscation, IEEE/ACM International Conference on Computer-Aided Design, 2008, pp. 674–677
15. N.H.E. Weste, D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd edn. (Addison-Wesley, 2005)
16. J. Li, J. Lach, At-speed delay characterization for IC authentication and Trojan Horse detection, IEEE International Workshop on Hardware Oriented Security and Trust, 2008, pp. 8–14
17. Y. Jin, Y. Markis, Hardware Trojan detection using path delay fingerprint, IEEE International Workshop on Hardware Oriented Security and Trust, 2008, pp. 54–60
18. M. Banga, M. Hsiao, A Novel Sustained Vector Technique for the Detection of Hardware Trojans, IEEE International Conference on VLSI Design, 2009, pp. 327–332
19. M. Banga, M. Hsiao, A Region Based Approach for the detection of hardware Trojans, IEEE International Workshop on Hardware Oriented Security and Trust, 2008, pp. 43–50
20. X. Wang, H. Salmani, M. Tehranipoor, J. Plusquellic, Hardware Trojan detection and isolation using current integration and localized current analysis, IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, 2008, pp. 87–95
21. M. Banga, M. Hsiao; Trusted RTL: Trojan Detection Methodology in Pre-Silicon Designs, IEEE International Workshop on Hardware Oriented Security and Trust, 2010, pp. 56–59
22. E. Love, Y. Jin, Y. Makris, Proof-carrying hardware intellectual property: A pathway to Trusterd module acquisition. IEEE Trans Inf Forensics Secur **7**(1), 25–40 (2012)
23. Y. Jin, Y. Makris; Proof Carrying-Based Information Flow Tracking for Data Secrecy Protection and Hardware Trust, IEEE VLSI Test Symposium, 2012, pp. 252–257
24. INRIA, The Coq proof assistant Sept 2010 [Online]. Available: http://coq.inria.fr/
25. M. Bidmeshki, Y. Makris; VeriCoq: A Verilog-to-Coq Converter for Proof-Carrying Hardware Automation, IEEE International Symposium on Circuits and Systems (ISCAS), 2015, pp. 29–32

Chapter 5
Hardware Trojans in Analog, Mixed-Signal, and RF ICs

Angelos Antonopoulos, Christiana Kapatsori, and Yiorgos Makris

5.1 Introduction

Due to a number of factors entailing time to market pressure, intellectual property (IP) reusability, and outsourced manufacturing and testing, compromising the IC supply chain for sensitive commercial and defense applications has become possible through hardware Trojan attacks targeting critical industrial sectors, such as military, infrastructure, automotive, and telecommunication applications. For example, a Syrian radar failure to warn the military of the incoming Israeli attack due to a backdoor in off-the-self microprocessors, which were used in the radar system, was reported in 2008 [5]. Another hidden backdoor in a computer chip that could allow an attacker to control critical applications, such as navigation and flight control in a Boeing 787, was dispatched in 2012 [3]. Counterfeit Cisco products, which were bought by military agencies and contractors, and electric power companies in the United States with a potential threat of gaining access to highly secure systems have also been recorded [1]. In 2011, US military bought 59,000 counterfeit chips from China destined for installation in critical defense systems [2]. Finally, Dell warned of a hardware Trojan in some of its server motherboards in 2010 [35]. While extensive research efforts have been expended over the last decade in understanding the threat of hardware Trojans, as well as in developing prevention and detection solutions in digital circuits [10, 24, 42, 45, 53], the topic remains largely unexplored for their analog/mixed-signal (AMS) and radio-frequency (RF) counterparts. A recent effort in [38] focuses in threats and countermeasures in digital ICs and discusses their relevance with the AMS domain. Given the widespread use of analog functionality (i.e., physical interfaces, sensors, actuators, wireless communications,

A. Antonopoulos (✉) • C. Kapatsori • Y. Makris
ECE Department, University of Texas at Dallas, Richardson, TX, 75080, USA
e-mail: aanton@utdallas.edu; cxk161430@utdallas.edu; yiorgos.makris@utdallas.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_5

etc.) in most contemporary systems, in this chapter we summarize and present the existing, albeit limited work on known vulnerabilities and proposed remedies for AMS/RF ICs, emphasizing on hardware Trojans.

5.2 Hardware Trojans in RF ICs

Wireless networks have become an inseparable part of everyday life and are now prevalent in most electronic systems, due to the rapid growth of telecommunications and the Internet of Things. At the same time, they are particularly vulnerable and constitute an appealing target for malicious attacks; indeed, since they exchange information over public channels, an attacker does not need to obtain physical access to the nodes, making such attacks far more plausible. As a result, wireless networks have been the target of covert channel attacks, most of which are staged via software or firmware modifications. Beyond such attacks, which exploit legitimate hardware and protocol capabilities, the first hardware Trojan targeting the baseband part of an 802.11a/g transmitter to leak sensitive information over the air was recently shown in [44]. To date, however, only a few groups have investigated and reported existent vulnerabilities in the analog/RF circuitry of transceivers used in wireless networks. In the following sections, we summarize the existing exploitation examples of such vulnerabilities by hardware Trojans along with respective defensive mechanisms.

5.2.1 Hardware Trojans in Wireless Cryptographic ICs

5.2.1.1 Attacks

A hardware Trojan attack in a wireless cryptographic IC, specifically in an ultra-wideband (UWB) transmitter, was demonstrated through silicon measurements in [30, 31]. The attack targets both the digital and analog/RF parts of the IC, and its general principle is shown in Fig. 5.1. The attack is quite simple to implement and can be staged in various phases of the IC supply chain, e.g., in the design or the fabrication level. On the digital side, the added hardware taps into the register that stores the 128-bit advanced encryption standard (AES) key, in order to steal one bit at a time. The value of the stolen key bit is forwarded to the UWB transmitter, through which it is leaked by modulating the parameters of the wireless communication during transmission of one ciphertext bit. Overall, along with every 128-bit block transmitted by the UWB transmitter, the 128-bit key is also leaked [31]. Specifically, two hardware Trojans have been implemented on-chip, modulating the amplitude and frequency characteristics of the transmission. The first hardware Trojan, i.e., Trojan-I, which is modulating the amplitude, is implemented in the power amplifier (PA), and its overhead is very small, since it only demands one extra transistor to leak information. The key bit, which is forwarded through the digital part, is provided to the gate of the extra PMOS

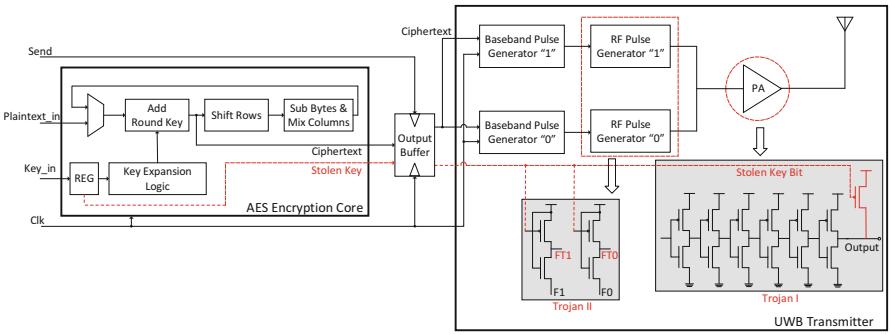


Fig. 5.1 Hardware Trojan modifications in digital and analog circuitry of a wireless cryptographic IC (Adapted from [31])

transistor. When the leaked key is "1", the transistor is off and, thus, the transmission power remains unaffected. However, when the leaked key bit is "0", the transistor turns on and adds a small current at the output node, thereby slightly increasing the output power which is, in turn, passed to the antenna. On the other hand, Trojan-II, which is modulating frequency, adds two extra transistors at the inputs of each of the two RF pulse generators. Again, when the stolen key bit is "0", the PMOS transistor of Trojan-II in Fig. 5.1 is turned on, thereby resulting in a higher frequency.

In order to investigate the Trojan impact on the legitimate and rogue transmission, the authors implemented 15 distinct Trojan levels [31]. Even for the maximum Trojan level, the Trojan impact on the legitimate transmission is carefully hidden in the transmission specification margins allowed for process variations. This is depicted in Fig. 5.2, where the measured transmission power for transmitting a ciphertext bit of "0" and "1" for 40 Trojan-free, 40 Trojan-I-infested, and 40 Trojan-II-infested ICs is plotted versus time. For the Trojan-infested transmissions, the maximum level of Trojan impact is employed. Each of the three distributions is enclosed in the $\mu \pm 3\sigma$ envelop of the Trojan-free ICs [30, 31]. Interestingly, none of the Trojan-infested ICs falls out of the envelop boundaries.

Despite being hidden in the process variation margins, the impact of the hardware Trojan on the transmission power waveform suffices for the informed adversary to obtain the secret key and, by extension, the plaintext by deciphering the ciphertext. All the attacker has to do is listen to the public wireless transmission channel, focusing on the parameter manipulated by the hardware Trojan (i.e., amplitude or frequency), in order to observe the different levels, which correspond to a key bit of "1" and "0", respectively, when a ciphertext bit of value "0" and a ciphertext bit of value "1" are transmitted. This is shown in Fig. 5.3 for a Trojan-I-infested chip, where the receiver waveform of a 4-bit ciphertext block is illustrated. The minute amplitude increase when a key bit of "0" is transmitted – regardless of the text value – provides the attacker the information needed to correctly obtain the key. Similarly with Trojan-I, the attacker can also obtain the leaked information for the Trojan-II-infested transmission [30, 31]. In both cases the receiver consists of an oscilloscope and an antenna connected on it.

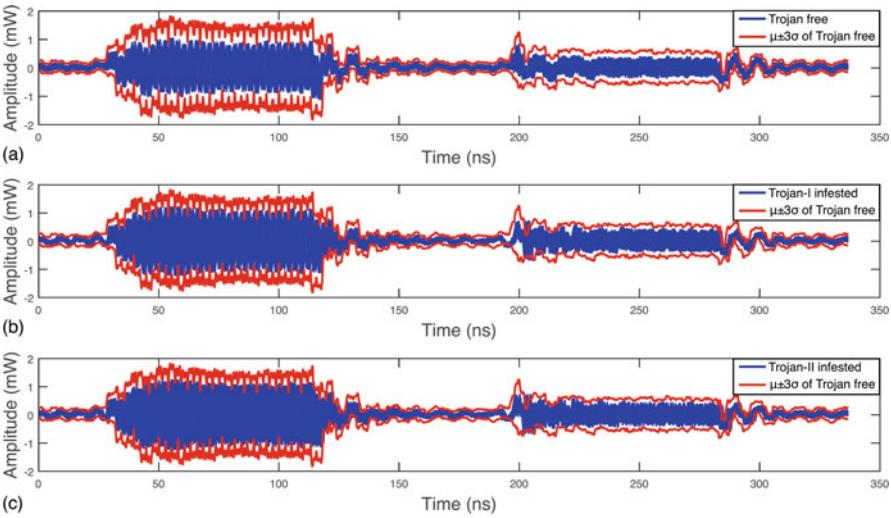


Fig. 5.2 Transmission power of 40 (**a**) Trojan-free ICs, (**b**) Trojan-I-infested ICs, and (**c**) Trojan-II-infested ICs enclosed in the $\mu \pm 3\sigma$ transmission [31]

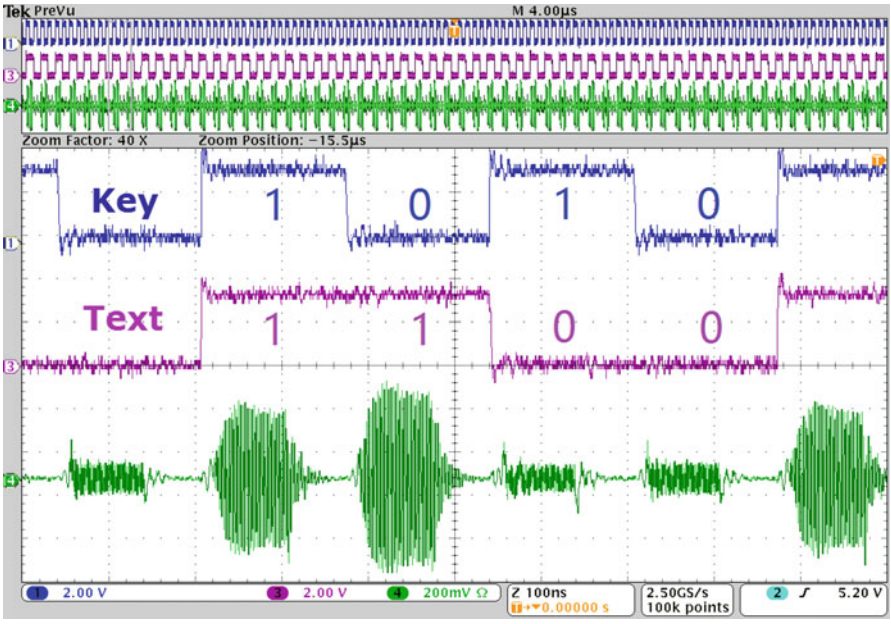


Fig. 5.3 Received waveform of a 4-bit ciphertext block transmitted by Trojan-I-infested chip [31]

5.2.1.2 Defenses

Unlike traditional test methods which are ineffective in detecting hardware Trojans (i) with small overhead (in terms of area and power), (ii) which do not violate any protocol specifications, and (iii) which remain within the margins allowed for process variations, several defensive methods have been lately reported, capable of raising a red flag under the presence of hardware Trojans, which manipulate transmission characteristics, similarly with those previously described. These defensive mechanisms range from statistical side-channel fingerprinting to concurrent and formal methods and are discussed below.

Statistical Methods: Constructing IC fingerprints based on side-channel parameters and using these fingerprints to statistically assess whether an IC is contaminated by a hardware Trojan or not were first presented in [6, 23] through a global power consumption-based and a delay-based method. The idea of side-channel fingerprinting is the basis for detecting the two hardware Trojans, which were previously presented. The general principle, which was originally described in [24, 29, 30], relies on the systematic impact that hardware Trojans impose on transmission characteristics. This systematic impact is essential for the attacker to be able to discern the hidden information, as is shown in Fig. 5.3. In practice, hardware Trojans add a statistical structure to the transmission characteristics, either in power or in frequency, which is precisely what statistical side-channel fingerprinting exploits in order to uncover malicious operation.

To demonstrate the effectiveness of statistical side-channel fingerprinting, the authors in [24, 29, 30] performed the following experiment: Initially, the same six blocks of ciphertext were transmitted by the UWB RF front end for each of the 40 Trojan-free, 40 Trojan-I-infested, and 40 Trojan-II-infested ICs. Visualization of the accumulated power in a 3-D space does not reveal any suspicious operation since all populations fall upon each other as depicted in Fig. 5.4a. However, when a simple statistical processing, such as principal component analysis (PCA) along with a one-class classifier, e.g., minimum volume enclosed ellipsoid (MVEE), is employed, the three populations become clearly distinguishable, as shown in Fig. 5.4b. Consequently, hardware Trojan existence can be detected.

Concurrent Hardware Trojan Detection Method: Statistical side-channel fingerprinting methods, such as the one discussed above, operate either before an IC is deployed or, periodically, during idle times, after an IC is deployed. Therefore, they can be easily evaded by a hardware Trojan which remains dormant at all times except during normal operation. To counteract this issue, a concurrent hardware Trojan detection (CHTD) method which operates along with the normal functionality of the IC was presented in [32] and is shown in Fig. 5.5.

The method checks an invariant property of the circuit and uses an on-chip one-class classifier to assert a CHTD output when the invariance is violated. The classifier is trained using trusted side-channel fingerprints obtained at test time when the Trojan is dormant. The trained classifier can, then, be used to examine

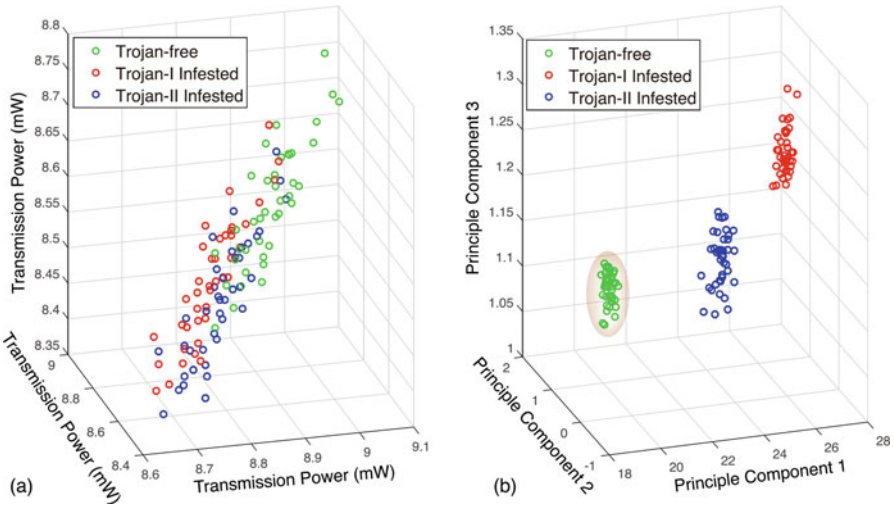


Fig. 5.4 Projection of hardware Trojan-free and hardware Trojan-infested circuits on a 3-D space where each dimension corresponds to (**a**) total transmission power for transmitting one ciphertext block, demonstrating that the populations are indistinguishable, and (**b**) one of the three top principal components yielded by performing PCA on the total transmission power for transmitting each of the six blocks for all the chips. The MVEE enclosing the hardware Trojan-free population, which can be used to classify a chip as hardware Trojan-free or hardware Trojan-infested, is also shown [31]

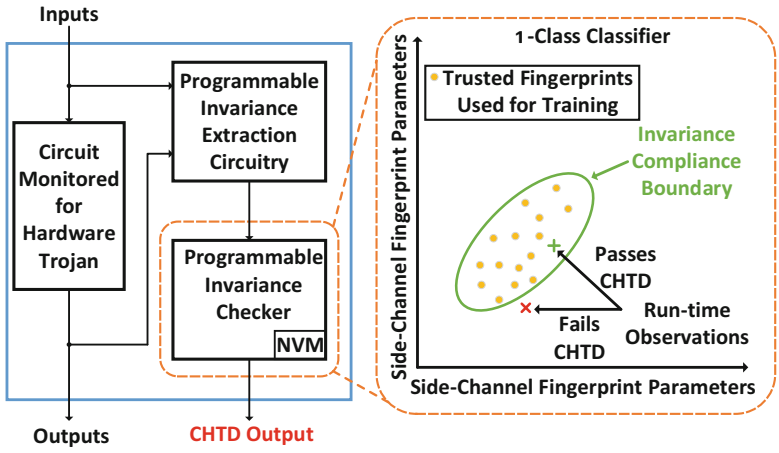


Fig. 5.5 CHTD experimentation platform (Adapted from [32])

compliance of runtime observations of the invariant property, by comparing their footprint in the side-channel fingerprinting space to the learned boundary. To assess whether the invariant property is violated or not, two observations are collected from a single ciphertext bitstream transmission. Each of the observations consists of k

number of bits, m of which are "1s." If the integrated voltage for observations A and B is V_A and V_B, respectively, then, the following invariance should always hold true:

$$|V_A(k, m) - V_B(k, m)| = \delta_{\text{noise}}, \tag{5.1}$$

where δ_{noise} represents measurement noise and nonidealities. If different k and m values are used for the two observations, the invariance becomes:

$$|V_A \cdot (k_A, m_A) - V_B \cdot (k_B, m_B)| = \delta_{\text{noise}} + |(m_A - m_B) \cdot V_{C1} + [(k_A - m_A) - (k_B - m_B) \cdot V_{C0}]|, \tag{5.2}$$

where V_{C1} and V_{C0} are the integrated voltage for a transmission of a bit equal to "1" and "0", respectively. Apart from its nonintrusive performance, CHTD is a self-referencing approach, which gives the flexibility of choosing different values for k_A, m_A, k_B, and m_B, thus making the design of a hardware Trojan which can evade the invariant property checking very difficult for an attacker. Effectiveness of the CHTD method was verified on the Trojan-infested ICs of Sect. 5.2.1 and was demonstrated in [32].

Formal Methods: The first information flow tracking (IFT) approach for ensuring data confidentiality in analog/RF designs was presented in [12]. The IFT was integrated into an automated proof-carrying hardware intellectual property (PCHIP)-based framework which was initially used to enforce information flow policies in digital designs. The proposed approach, which operates at the transistor level, converts the netlist of the analog/RF circuitry to a Verilog representation and accordingly uses an automated framework called *VeriCoq-IFT* to convert the Verilog design to the Coq representation, generate security property theorems for preventing sensitive information leakage, and construct their proofs [12].

As demonstrated in [12], the method is able to detect sensitive data leakage from the digital domain to the analog domain and vice versa, without requiring any modification of the analog/mixed-signal/RF circuit design flow. The formal method was applied on the AES UWB transmitter design of Sect. 5.2.1. Its Verilog netlist was extracted, and the *Plaintext* and *Key* signals were annotated with appropriate sensitivity levels, as shown in Fig. 5.6. The corresponding sensitivity-reducing operations in the AES core were also marked. Then, using *VeriCoq-IFT*, the design was converted to the formal representation, and *Coq* was used to evaluate the

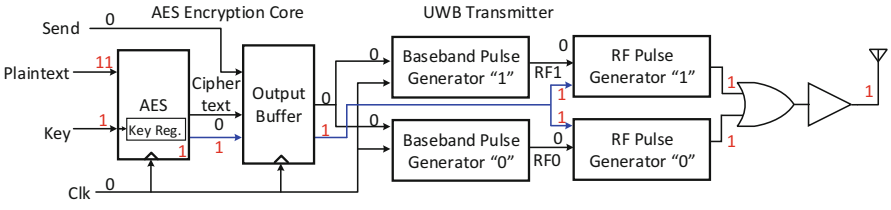


Fig. 5.6 Information leakage path in Trojan-II (Adapted from [12])

automatically generated proof for the security theorem asserting the sensitivity of the design output. In the Trojan-free case, the proof of the security property theorem for the output passes in *Coq*, attesting that this output never leaks sensitive information, under the provision that the initial sensitivity values and sensitivity reducing operations were annotated correctly in the design. The effectiveness of the method was also verified under the presence of Trojan-I and Trojan-II, wherein the proof did not pass in *Coq*. This implies that a possible path exists through which sensitive information may leak to the output. Figure 5.6 shows the information leakage path and the propagated signal sensitivity levels in the AES UWB design for Trojan-II.

Observation of Parasitic Loads: Detection of hardware Trojans, which add some structure to the compromised IC, was recently discussed in [16] for RF circuits. The method is based on the signature left due to rogue load capacitances. Specifically, stimulus optimization experiments were performed in a typical cascode low-noise amplifier (LNA) and allowed detection of capacitive loads (due to the presence of hardware Trojans in the legitimate circuit) in several of the circuit's internal nodes.

5.2.2 RF Transmission Below Noise Floor

5.2.2.1 Attacks

In line with [30, 31], the ability of hardware Trojans to hide unauthorized transmission signals within the ambient noise floor through the use of spread-spectrum techniques was presented in [15]. The original concept of communicating with attackers below the noise power level of a compromised crypto-processor was initially demonstrated in [28], where multi-bit information was leaked through a power side channel. Specifically, spread spectrum was used to distribute the power of side-channel leakage to multiple clock cycles, so that the signal-to-noise ratio (SNR) of each clock cycle is low enough to evade detection. The attacker can then exploit the side-channel information by averaging over a large number of clock cycles.

Similarly, the Trojan system in [15] spreads the rogue data and attenuates the Trojan signal so that it is pushed below the ambient noise floor. The principle of a spread-spectrum transmitter/receiver chain is shown in Fig. 5.7. The low-rate baseband data is multiplied by a higher-rate spread-spectrum code to generate a higher-rate sequence. The legitimate and Trojan spread signals are then added in the analog domain, constituting the signal to be transmitted, as shown in Fig. 5.7. The transmitted signal, containing both the legitimate and rogue coefficients, has an identical spectrum with the legitimate one, and thus the Trojan presence cannot be easily detected. This higher bitrate digital sequence is then transmitted over the noisy channel, which may undergo multipath fading and multiple interferers. At the receiver, both the intended signal and interferers are mixed with the same spread-spectrum code, de-spreading the original information and spreading the interferers instead. Extremely low power levels are required to retain effective communication.

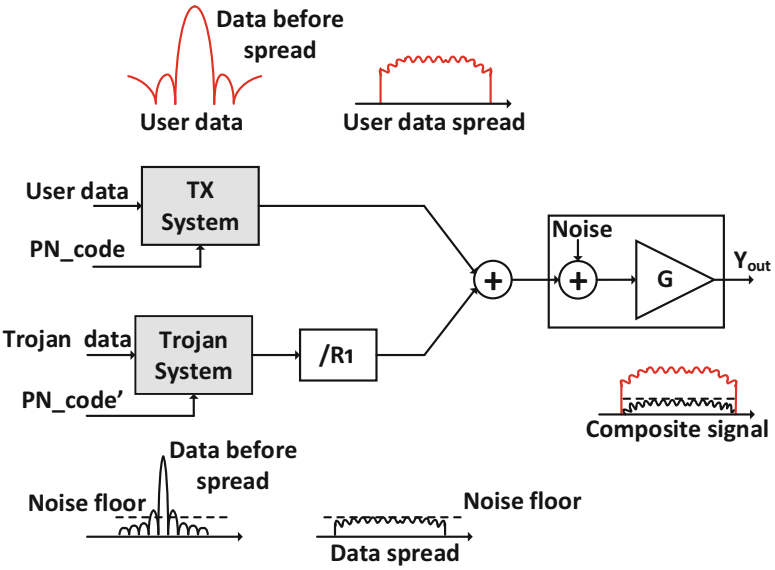


Fig. 5.7 Spread-spectrum technique used for evading detection of hardware Trojans in wireless networks (Adapted from [15])

However, this comes at the cost of reduced throughput for the attacker. The spread-spectrum attack does not affect the legitimate transmission since it remains hidden well below the noise floor, thereby evading any performance-based testing or monitoring [15].

5.2.2.2 Defenses

A method for detecting such a hardware Trojan, as well as hardware Trojans in mobile platforms, was presented in [25]. This method does not require any golden reference; rather, it is based on self-referencing. In [25], the output of a mobile platform running on a commercial MpSoC board is driven into a periodic steady state (PSS) to decouple the response of the board from that of noise and Trojan. The changes in the circuit behavior between periods indicate the existence of unauthorized activity. After exciting the circuit and obtaining its current consumption signal, this signal passes through a low pass filter to obtain the self-referencing signal. This is then subtracted from the original signal to obtain a difference signal, which consists of noise and malicious activity, if any. Analysis of the difference signal in the time domain is not capable of distinguishing the Trojan signal from channel noise. However, by using the fast Fourier transform of the difference signal, calculating the average noise level, and setting a threshold of $3\sigma_n$ for noise referencing, where σ_n is the noise variance, the Trojan signal can be detected. Any unexpected bin, i.e., frequency bins that do not correspond to the fundamental and its harmonics, above threshold is considered a spectral violation.

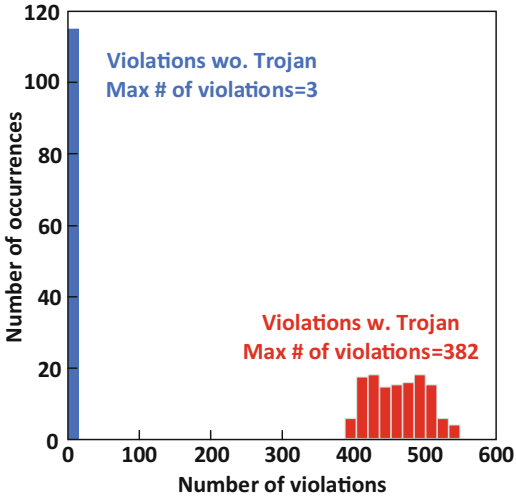


Fig. 5.8 Histogram of spectral violations for spectra with and without Trojan activity (Adapted from [25])

For demonstration purposes, the authors in [25] collected 240 data spectra, half of which were Trojan infested. For spectra without any Trojan activity, the maximum number of extra bins is 3, whereas the corresponding additional bins for the Trojan-infested spectra is much greater, i.e., greater than 380 for all spectra. This is shown in Fig. 5.8. Despite the Trojan operation at or below the noise level, its activity is clearly observable through the number of bin violations.

5.3 Hardware Trojans in AMS ICs

5.3.1 Attacks

Unlike RF circuits, where a hardware Trojan adds extra circuitry to the legitimate structure to exploit its vulnerabilities, hardware Trojans in AMS ICs may not need to add extra overhead to the target IC, neither must they leave a signature during normal operation; rather, they may exploit Trojan states that might be present in AMS components with positive feedback loops, which are commonly used to desensitize the circuit outputs from supply voltage variations. It all began back in 1980, when it was shown that transistor networks with positive feedback loops can have more than one solution to their DC equations, for some choice of network parameter values. In [37] it was reported that any circuit with positive feedback loops allows more than one DC operating points and, conversely, any circuit that has more than one solution in the DC equations must have positive feedback loops. These multiple DC operating points were demonstrated for verification purposes in a CMOS log-domain filter employing a positive feedback loop [17], but were never studied in the context of hardware security until recently. The problem of

multiple operating states in analog circuits is commonly referred to as the start-up problem, meaning that a start-up circuit should typically be added to remove the undesired state. However, if no start-up circuit is used, which is quite common in analog design, or if the start-up circuit is not effective, a redundant state harboring a Trojan may still exist [14].

Several research results have shown that an AMS IC can exhibit a Trojan state, which can be defined as an operating state that forces the circuit to behave in an unexpected and undesired way, producing inconsistent results at its output and, thus, directly affecting preceding blocks in a chain of IC components. These Trojan states have been shown to affect the output characteristics of operational amplifiers (OP-AMPs), current mirrors, bandgap references, oscillators, and filters [13, 14, 47–49, 51]. This has been verified via simulations performed in MATLAB and Cadence Spectre, and results are summarized in Table 5.1.

In the inverse Widlar mirror shown in Fig. 5.9a, the output voltage may reach values other than the expected for a specific temperature, due to multiple equilibrium points [17, 47]. Indeed, as plotted in Fig. 5.9b, when temperature is swept, the same output voltage, V_{out2}, is obtained for temperatures T1 and T2. Similarly, a Trojan state was shown to exist in a fully differential operational amplifier when performance enhancement feedback, i.e., a slew-rate enhancement circuit, producing a positive feedback loop, is being used [13]. Trojan states were also

Table 5.1 Trojan states in analog ICs

Reference	Circuit topology	Simulation level
[14, 47, 49]	Inverse Widlar	Cadence Spectre
[17, 48]	Filter	HSPICE
[33, 51]	Bandgap	Cadence Spectre
[13]	OP-AMP	Cadence Spectre
[48]	Wien bridge oscillator	N/A

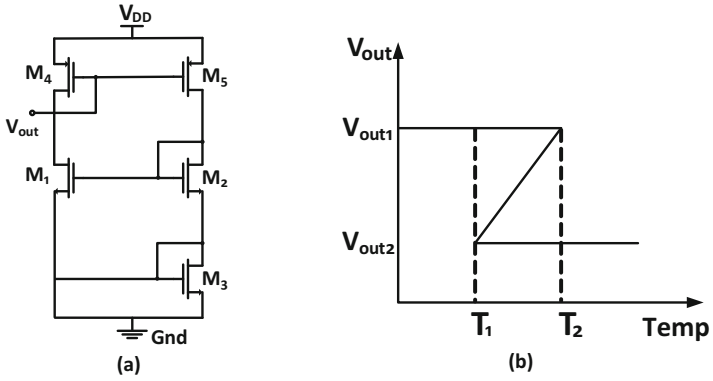


Fig. 5.9 (**a**) Schematic of the inverse Widlar current mirror and (**b**) multiple operating points in DC temperature sweep of the inverse Widlar current mirror (Adapted from [47])

demonstrated via simulation results for a Wien bridge oscillator [48]. These states occur when high nonlinearities in the input-output characteristic are present. Specifically, the circuit may have either a static (undesired) or a dynamic mode of operation, and further, even when in its dynamic mode, oscillation states of different amplitudes or frequencies may still occur, depending on the initial conditions of the capacitors [48]. Therefore, hardware Trojans in an oscillator can correspond either to a static mode, incapacitating the IC, or to unexpected oscillation characteristics, e.g., modified amplitude and frequency. Given the widespread use of oscillators in communication systems, a Trojan state could have devastating consequences in information exchange, e.g., it could result in a shift of the local oscillator frequency to a different band, which an attacker could exploit to leak sensitive information.

This class of hardware Trojans does not demand any increase in power, area, or architecture and, thus, leaves no signature. Therefore, even if the complete circuit schematic is available, the presence of multiple operating points during design and verification can remain undetected.

5.3.2 Defenses

Defense mechanisms that have recently been applied for detecting multiple operating points in analog circuits with positive feedback loops are based on homotopy theory, which has been long used for verification purposes [43]. Given an analog circuit, the first step toward identifying Trojan states relies on determining the circuit's positive feedback loops. This is achieved by constructing a directed dependency graph based on its circuit topology. For example, in the bootstrapped V_t reference circuit of Fig. 5.10a, two feedback loops can be identified. These loops are:

$$I_1 \rightarrow B \rightarrow I_2 \rightarrow A \rightarrow I_1 \tag{5.3}$$

$$I_1 \rightarrow B \rightarrow I_2 \rightarrow C \rightarrow I_1 \tag{5.4}$$

Specifically, changes in current I_1 flowing through transistors M_4 and M_1 affect voltage at node B, which in turn pushes I_2 to change. Thereby, node voltage A is also affected, which, accordingly, impacts I_1. Similarly, the second graph is described by Eq. (5.4). Since only the positive feedback loops need to be identified, voltage/current dependencies are annotated with a sign. For example, an increase in the node voltage A results in a decrease in I_1, since A is applied to the gate of PMOS transistor M_4. Therefore, the edge $A \rightarrow I_1$ in the top loop of Fig. 5.10c receives a $-$ sign. After all signs have been annotated, a positive feedback loop is defined as a feedback loop which contains an even number of negative dependencies, whereas a negative feedback loop contains an odd number of negative dependencies. Therefore, the feedback loop described by Eq. (5.3) is a positive feedback loop [33, 34, 50, 51].

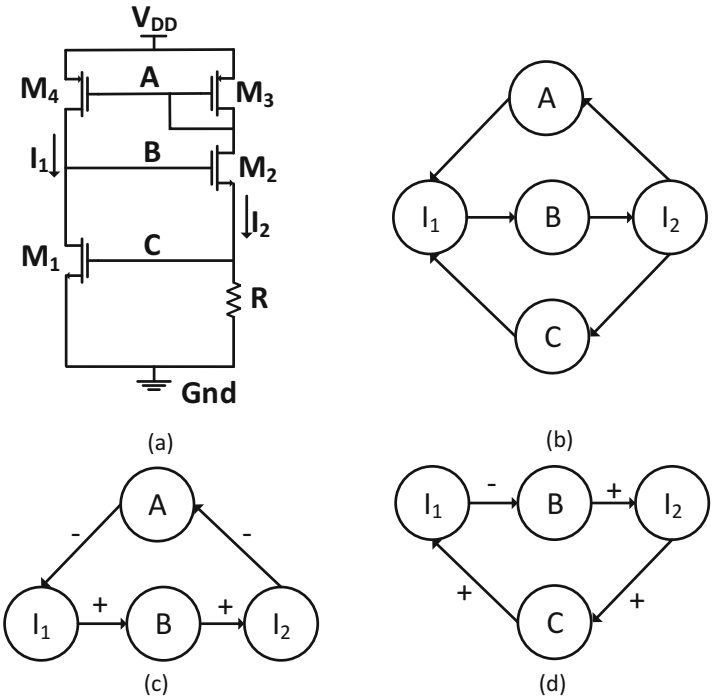


Fig. 5.10 Bootstrapped V_t reference circuit: (**a**) schematic, (**b**) directed dependency graphs, (**c**) dependence sign for the top loop, and (**d**) dependence sign for the bottom loop (Adapted from [33])

After the positive feedback loops have been identified, the continuation method can be applied to identify the presence of multiple operating states. This method involves the introduction of a voltage or current source that can be swept to trace operating points of a circuit other than the desired and can be viewed as a homotopy approach applied to each positive feedback loop [52]. The most common approach in the continuation method involves insertion of sources that do not break any loops in the circuit, thereby circumventing concerns about loop loading when a feedback loop is broken. Methods in which feedback loops are broken can also be used provided the operating points are not disturbed by breaking the loops [50, 52]. The continuation method has been demonstrated in several AMS ICs employing positive feedback loops [33, 49–51, 51, 52]. An example of the method is shown in Fig. 5.11, where a voltage source, V_{AUG}, is inserted in a constant g_m reference circuit. When this voltage source is swept, the resulting current plot indicates three operating points labeled A, B, and C. The right-most zero-crossing at point C is the desired one, since at this point $V_{AUG} = -0.2V$, which means that all transistors are biased in saturation [21].

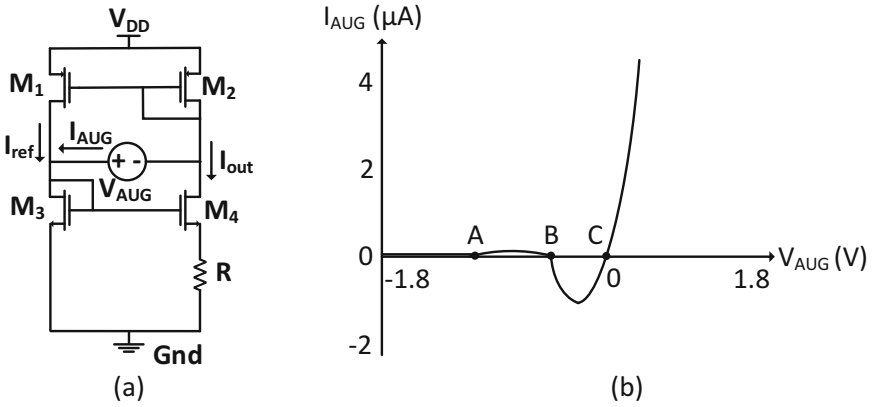


Fig. 5.11 (**a**) Constant g_m reference circuit and (**b**) I-V curve from the continuation method (Adapted from [21])

A homotopy-type circuit simulation employing temperature sweeping can also raise a red flag for the circuit designer, as has been shown in [47].

Ad hoc methods for preventing undesired states in analog ICs also exist in literature. Specifically, in [14] simulation results indicated that by decreasing the width of the diode-connected transistor M_3 in the inverse Widlar current mirror of Fig. 5.9a, the region in which output voltages overlap, thus resulting in potential Trojan states, can be eliminated. Another route toward detection of Trojan states in AMS circuits could be to use formal verification for AMS designs after they have been approximated as purely Boolean models, as in [26, 27]. However, this has not been investigated so far. The same holds for [56], wherein a formal-based solution to the verification of AMS designs was applied to a delta-sigma modulator, validating its operation with respect to a given set of properties. Recently, information flow tracking in AMS designs through proof-carrying hardware was shown in [12]. This method could also be applied for verifying security properties, e.g., detection of multiple equilibrium points, of AMS designs.

5.3.3 Analog Triggers

A key limitation of the AMS Trojan states which were previously presented stems from the lack of trigger mechanisms capable of driving a circuit into an undesired state. So far, a few analog triggers have been presented in literature and are discussed herein.

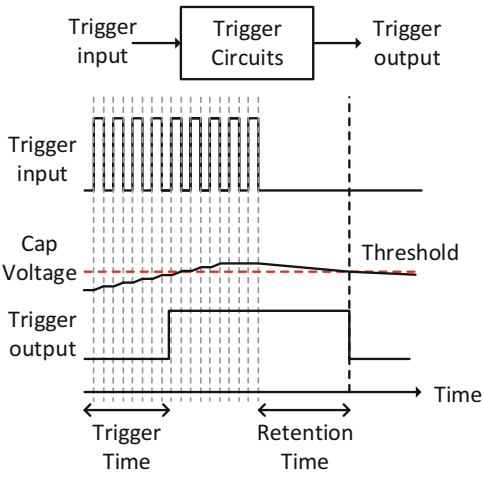


Fig. 5.12 Behavioral model of the capacitor-based analog trigger circuit (Adapted from [55])

5.3.3.1 Capacitors

An analog trigger targeting a digital microprocessor was demonstrated in [55]. Similarly with [28], where a capacitor of adjustable value is used to leak information conveyed by a power side channel, the circuit in [55] employs capacitors to siphon charge from nearby wires as they transition between digital values. When the capacitors are fully charged, an attack to a victim flip-flop is staged [55]. The capacitor performs analog integration of charge from a victim wire while at the same time being able to reset itself through charge leakage. Every time the victim wire toggles, the capacitor's charge increases and its voltage exceeds a predetermined threshold voltage. When the trigger input is inactive, leakage current gradually reduces the capacitor's voltage and eventually stops the trigger output. Operation of the analog trigger is described in Fig. 5.12.

5.3.3.2 Voltage Glitches

Voltage glitches of the power supply are another trigger mechanism whose impact was shown on a mixed-signal IC consisting of digital logic along with a phased-locked loop [7]. Voltage glitches can have devastating effects in frequency synthesis and can induce large variations in the output voltage of bandgap references. These voltage glitches can be produced using body biasing attacks [9]. The body biasing injection (BBI) method applies high-voltage pulses on the circuit substrate, thereby modifying the capacitive and/or resistive coupling between the substrate and power supply or ground, as shown in Fig. 5.13a. This, in turn, locally affects the power supply and/or ground voltages, as depicted in Fig. 5.13b, and can practically result

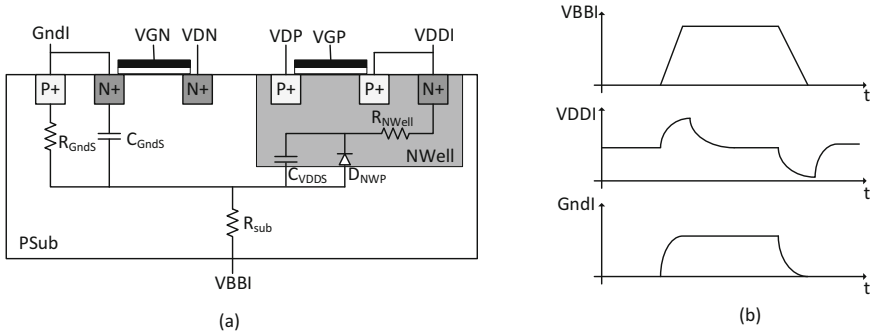


Fig. 5.13 (**a**) BBI effects on CMOS logic (cross-sectional view) and (**b**) forward BBI effects on V_{DD} and ground nodes (Adapted from [9])

in large deviations of power supply voltage values, as has been experimentally demonstrated in [9]. However, this requires that the packaged IC needs to be opened, in order to apply a very high and short substrate bias pulse.

5.4 Other Threats in AMS/RF ICs

As the complexity of electronic systems has significantly increased over the past years, the market of reusable IPs has tremendously grown. As a result, the vast majority of silicon dies and systems on chips currently comprise AMS IP blocks which are produced by third-party companies. Unlike their digital counterparts whose security concerns have been addressed during the last two decades, solutions for protecting AMS/RF IPs against reverse engineering and theft have only recently been proposed. The same holds true for AMS/RF IP counterfeiting which is considered a problem growing in magnitude [18, 20].

5.4.1 IC/IP Piracy and Counterfeiting

IP piracy has become a significant concern due to the large volume of reusable IPs in silicon dies. IP piracy scenarios can be staged either at the foundry level, e.g., through reverse engineering and illegal copying of an IP, or after the IP has been produced, e.g., claiming ownership and reselling it as a black box [46]. Reverse engineering can be performed at the chip, printed circuit board, and system level [39]. Many practical examples of counterfeit ICs, with an increased level of sophistication, typically deployed after the IC has been produced, have also been reported, raising health and safety concerns [18, 20]. To prevent reverse engineering, obfuscation-based techniques applied either at the netlist or at the layout level

(camouflaging) have been used to transform a design into one that is functionally equivalent to the original but much more difficult to reverse engineer [39, 40]. Along with obfuscation, logic encryption was recently introduced to encrypt the functionality of a design and protect it from malicious insertions by untrusted foundries [41, 42]. IC ownership and authenticity can be preserved via watermarking [46], which uniquely encodes the signature of the author both in the netlist and the physical implementation stages. Finally, several counterfeit detection methods have been proposed, which can be classified into physical and electrical inspections and aging-based fingerprints [20, 54]. More details on the aforementioned detection and prevention methods against reverse engineering and counterfeiting are provided in [18, 20, 39–42, 46]. Despite the extensive work effort in the digital domain, in the AMS/RF domain, only a small number of defenses against piracy and counterfeiting have been reported. Accordingly, existing remedies for AMS/RF piracy and counterfeiting are listed and described below:

5.4.2 Vulnerability Analysis

In [8] the authors propose an analysis in order to expose all vulnerabilities existent in the design of an analog IP. The methodology aims to identify potential security breaches in analog IP blocks and has to be performed after the IP specification stage rather than after manufacturing, as shown in Fig. 5.14. The IP block used for demonstrating the method is an analog block generating a clock signal, which consists of a bandgap reference, a voltage doubler, and a voltage-controlled oscillator. The subfunctions of each of the sub-circuits are analyzed, and the faults, as well as their signatures, are identified. Finally, the potential attacks for each of the faults are evaluated along with the identifying potential, which expresses the time and effort required for an adversary to identify the attack. Once all attacks have been identified, appropriate countermeasures can be taken by the designer.

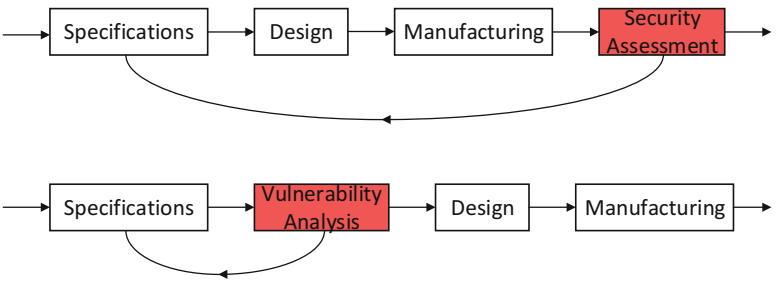


Fig. 5.14 AMS IP design flow before and after vulnerability analysis (Adapted from [8])

5.4.3 Split Manufacturing

Split manufacturing was recently proposed to protect analog/RF IPs from reverse engineering at the foundry level [11]. The key idea of split manufacturing is to protect designs by dividing manufacturing chips into front end of line (FEOL) and back end of line (BEOL). Accordingly, FEOL and BEOL layers are fabricated in untrusted and trusted foundries, respectively. The general concept of this method was presented in a PA design for RF applications. The top two metal layers of the technology were removed from the FEOL. In such a case, the inductors and capacitors of the PA become invisible to the attacker. The authors show that even if the inductors' and capacitors' positions and sizes can be estimated through the blank areas that are created, it is difficult to reverse engineer the chip given the wide range of component values, bias voltages, and operating frequencies. In RF designs, inductors are placed in metal rings, and lower metal layers inside the rings will be removed for performance optimization. Therefore, the rings themselves may indicate the exact position and size of the inductors. To counteract this issue, the authors obfuscate the original design, by inserting nonfunctional rings and creating empty zones. The empty blocks in the layout might increase performance overhead, but this can be alleviated if the designers consider security in the early design stages.

5.4.4 AMS IP Watermarking

To protect AMS IP ownership, a layout watermarking method was proposed in Algorithm 1 [22, 36]. This method uses an algorithm, which is described below, to parse the layout netlist and sort transistors one level at a time, based on their type (NMOS or PMOS), width, shortest distance to input, and shortest distance to output. The outcome of the search algorithm is a uniquely ordered list of transistors.

Once this list has been created, the owner generates the watermark he/she wants as a seed for a pseudorandom number generator. The bits which are generated from the random function form a long bitstream that can be aligned with the stream of uniquely sorted transistors. The bitstream is embedded by fingering the transistors depending on the bit that aligns with each transistor. A bit value of "1" or "0" corresponds to an even or odd number of fingers, respectively. Using this method a design entity A, having a netlist A, can prove ownership of an IP against a design entity B, having a netlist B. The IP owner can look into the nodes of the ordered array of netlist B and generate bitstream B, corresponding to the odd or even number of transistor fingers. The owner which has two bitstreams, A and B, can measure the degree of correlation between them. Unless design entity B generates the correct seed for bitstream B, design entity A can claim that design entity B has stolen the layout. This technique has been effectively applied to a two-stage Miller operational amplifier. The watermarked layout suffered only 0.25% increase in the chip area.

Algorithm 1 Pseudocode for ordering transistors [22]

```
Make a dummy transistor
Connect it to all of the inputs of the circuit
Add the dummy transistor to the FIFO queue
while queue is not empty do
    De-queue the next transistor
    if it is not marked visited then
        for the set of successor transistors do
            if any are matched then
                group them together
                rank by channel type, width and distance to output
            end if
            if any are indistinguishable then
                treat as matched
                add the ordered transistors to the FIFO queue
                append them to the final sequential array
            end if
        end for
        Mark the current transistor visited
    end if
end while
```

5.4.5 AMS Counterfeit Protection

Despite the fact that analog ICs are among the five most counterfeited parts [4], only a few protection mechanisms have been reported. Specifically in [20], protection against recycled analog ICs was achieved using statistical methods, such as one-class classifiers and degradation curve sensitivity analysis. Typical test results from production early failure rate analysis, such as minimum supply voltage ($V_{\min}$), quiescent current (I_{ddq}), and maximum oscillation frequency ($F_{\max}$), were used as parametric measurements for evaluating both methods. Results were demonstrated in a fully differential folded cascode operational amplifier designed in a 45 nm technology node, showing that both methods were able to achieve 100% correct classification between brand new and recycled devices. Recently, low-cost, on-chip ring oscillators were used for protecting ICs against recycling [19]. It is likely that such an approach can also be applicable in AMS and RF ICs.

5.5 Discussion

After over a decade of intense research efforts by numerous groups around the world, the objective of ensuring trustworthiness of digital ICs and IPs is a fairly well-understood and quite mature topic. Indeed, a large number of alternative threat scenarios, as well as detection and/or prevention methods, have been demonstrated and experimentally evaluated, often using actual silicon measurements. On the

other hand, the operational complexity and the continuous-domain characteristics of AMS/RF ICs have served as challenges which have limited the community's collective understanding, modeling, and mitigating of security risks in the analog/RF domain. Among the most notable contributions in this domain, we pinpoint the effectiveness of hardware Trojans in modifying the RF front end of cryptographic ICs in order to covertly steal sensitive information, which has been experimentally demonstrated in silicon by a few groups. In the AMS world, the key contribution to date consists in the demonstration of innate Trojan states, which may potentially result in undesired operating conditions. A few concepts of analog triggers, which have mostly been used to compromise digital circuits, and a few protection mechanisms against analog IP theft and reverse engineering complete the picture of the rather limited literature on this subject matter. Moving forward, a number of limitations in existing studies need to be addressed in order to raise our understanding of the problem to the next level and lead to breakthroughs in this area. For example:

- In AMS circuits, security implications have only been shown in a few basic analog blocks; moreover, all of the relevant work is based on simulations. While simulations are informative, demonstration and evaluation through actual silicon implementation is needed for drawing definitive conclusions.
- How an AMS circuit can be triggered to enter an undesired state and what the payload of such a Trojan state might be, other than circuit malfunction or denial of service, should be further investigated and better understood. Most of the current incarnations are either too simplistic or too unrealistic to be considered a real threat.
- Systematic, generalizable HT detection/prevention methods need to be developed for AMS/RF ICs, rather than the current ad hoc solutions. While this is inherently difficult in the analog domain, it is nevertheless important in order to facilitate automation and development of pertinent metrics.
- Formal, provably secure methods for protecting AMS/RF IPs are still at their infancy and are urgently required. While analog formal verification has made great strides recently, its findings have yet to be applied in the security and trust domain.

5.6 Conclusions

Despite the operational complexity and the continuous-domain characteristics of AMS/RF ICs, the research community is currently considering and investigating security threats in the AMS/RF circuitry of ICs. Therefore, initial steps toward understanding, modeling, and mitigating the security risks that AMS/RF ICs bring along have been taken. Accordingly, disseminations of existing works were summarized in this chapter. Specifically, we discussed the effectiveness of hardware Trojans modifying the RF front end of ICs to covertly steal sensitive information

and elaborated on Trojan states which may be present in several AMS circuits with positive feedback loops. We also explored detection methods targeting hardware Trojans in AMS/RF ICs, which are based on statistical analysis, formal approaches, and concurrent operation. Lastly, we summarized existing protection mechanisms against analog counterfeiting and reverse engineering, and we offered our thoughts regarding the next research directions that the AMS/RF community should focus on.

References

1. F.B.I. says the military had bogus computer gear (2008), https://goo.gl/QT90Nx
2. Fishy chips: spies want to hack-proof circuits (2011), https://goo.gl/wmJ2yL
3. Could a vulnerable computer chip allow hackers to down a Boeing 787 'Back Door' could allow cyber-criminals a way in (2012), https://goo.gl/i7aqm5
4. Top 5 most counterfeited parts represent a $169 billion potential challenge for global semiconductor market (2012), https://goo.gl/Ku4u6B
5. S. Adee, The hunt for the kill switch. IEEE Spectr. **45**(5), 34–39 (2008)
6. D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, B. Sunar, Trojan detection using IC fingerprinting, in *IEEE Symposium on Security and Privacy (SP)*, 2007, pp. 296–310
7. N. Beringuier-Boher, K. Gomina, D. Hely, J.B. Rigaud, V. Beroulle, A. Tria, J. Damiens, P. Gendrier, P. Candelier, Voltage glitch attacks on mixed-signal systems, in *Euromicro Conference on Digital System Design*, 2014, pp. 379–386
8. N. Beringuier-Boher, D. Hely, V. Beroulle, J. Damiens, P. Candelier, Increasing the security level of analog IPs by using a dedicated vulnerability analysis methodology, in *International Symposium on Quality Electronic Design (ISQED)*, 2013, pp. 531–537
9. N. Beringuier-Boher, M. Lacruche, D. El-Baze, J.M. Dutertre, J.B. Rigaud, P. Maurine, Body biasing injection attacks in practice, in *Workshop on Cryptography and Security in Computing Systems*, 2016, pp. 49–54
10. S. Bhunia, M.S. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan attacks: threat analysis and countermeasures. Proc. IEEE **102**(8), 1229–1247 (2014)
11. Y. Bi, J.S. Yuan, Y. Jin, Beyond the interconnections: split manufacturing in RF designs. Electronics **4**(3), 541–564 (2015)
12. M. Bidmeshki, A. Antonopoulos, Y. Makris, Information flow tracking in analog/mixed-signal designs through proof-carrying hardware IP, in *IEEE Design Automation and Test in Europe Conference (DATE)*, 2017
13. C. Cai, D. Chen, Performance enhancement induced Trojan states in op-amps, their detection and removal, in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 3020–3023
14. X. Cao, Q. Wang, R.L. Geiger, D.J. Chen, A hardware Trojan embedded in the Inverse Widlar reference generator, in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2015, pp. 1–4
15. D. Chang, B. Bakkaloglu, S. Ozev, Enabling unauthorized RF transmission below noise floor with no detectable impact on primary communication performance, in *IEEE VLSI Test Symposium (VTS)*, 2015, pp. 1–4
16. S. Deyati, B.J. Muldrey, A. Chatterjee, Targeting hardware Trojans in mixed-signal circuits for security, in *IEEE International Mixed-Signal Testing Workshop (IMSTW)*, 2016, pp. 1–4
17. R.M. Fox, M. Nagarajan, Multiple operating points in a CMOS log-domain filter, in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 1999, pp. 689–692
18. U. Guin, D. DiMase, M. Tehranipoor, Counterfeit integrated circuits: detection, avoidance, and the challenges ahead. J. Electron. Test. **30**(1), 9–23 (2014)

19. U. Guin, D. Forte, M. Tehranipoor, Design of accurate low-cost on-chip structures for protecting integrated circuits against recycling. IEEE Trans. Very Large Scale Integr. Syste. **24**(4), 1233–1246 (2016)
20. U. Guin, K. Huang, D. DiMase, J.M. Carulli, M. Tehranipoor, Y. Makris, Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain. Proc. IEEE **102**(8), 1207–1228 (2014)
21. W. Hou, Use of a continuation method for analyzing start-up circuits. Ph.D. thesis, University Of California, Irvine, 2011
22. D.L. Irby, R.D. Newbould, J.D. Carothers, J.J. Rodriguez, W.T. Holman, Low level watermarking of VLSI designs for intellectual property protection, in *IEEE International ASIC/SOC Conference*, 2000, pp. 136–140
23. Y. Jin, Y. Makris, Hardware Trojan detection using path delay fingerprint, in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 51–57
24. Y. Jin, D. Maliuk, Y. Makris, Hardware Trojan detection in Analog/RF integrated circuits, in *Secure System Design and Trustable Computing*, ed. by C.H. Chang, M. Potkonjak (Springer, Cham, 2016), pp. 241–268
25. F. Karabacak, U.Y. Ogras, S. Ozev, Detection of malicious hardware components in mobile platforms, in *International Symposium on Quality Electronic Design (ISQED)*, 2016, pp. 179–184
26. A.V. Karthik, S. Ray, P. Nuzzo, A. Mishchenko, R. Brayton, J. Roychowdhury, ABCD-NL: approximating continuous non-linear dynamical systems using purely Boolean models for analog/mixed-signal verification, in *IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 250–255
27. A.V. Karthik, J. Roychowdhury, ABCD-L: approximating continuous linear systems using Boolean models, in *IEEE Design Automation Conference (DAC)*, 2013, pp. 1–9
28. L. Lin, W. Burleson, C. Paar, MOLES: malicious off-chip leakage enabled by side-channels, in *IEEE International Conference on Computer-Aided Design (ICCAD)*, 2009, pp. 117–122
29. Y. Liu, K. Huang, Y. Makris, Hardware Trojan detection through golden chip-free statistical side-channel fingerprinting, in *IEEE Design Automation Conference (DAC)*, 2014, pp. 155:1–155:6
30. Y. Liu, Y. Jin, Y. Makris, Hardware Trojans in wireless cryptographic ICs: silicon demonstration & detection method evaluation, in *International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 399–404
31. Y. Liu, Y. Jin, A. Nosratinia, Y. Makris, Silicon demonstration of hardware Trojan design and detection in wireless cryptographic ICs. IEEE Trans. Very Large Scale Integr. Syst. **PP**(99), 1–14 (2016)
32. Y. Liu, G. Volanis, K. Huang, Y. Makris, Concurrent hardware Trojan detection in wireless cryptographic ICs, in *IEEE International Test Conference (ITC)*, 2015, pp. 1–8
33. Z. Liu, Y. Li, Y. Duan, R.L. Geiger, D. Chen, Identification and break of positive feedback loops in Trojan States Vulnerable Circuits, in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 289–292
34. Z. Liu, Y. Li, R.L. Geiger, D. Chen, Auto-identification of positive feedback loops in multi-state vulnerable circuits, in *IEEE VLSI Test Symposium (VTS)*, 2014, pp. 1–5
35. J. Markoff, Dell warns of hardware Trojan (2010), https://goo.gl/MQ8jYr
36. R.D. Newbould, D.L. Irby, J.D. Carothers, J.J. Rodriguez, W.T. Holman, Mixed signal design watermarking for IP protection, in *Southwest Symposium on Mixed-Signal Design*, 2001, pp. 110–115
37. R.O. Nielsen, A.N. Willson, A fundamental result concerning the topology of transistor circuits with multiple equilibria. Proc. IEEE **68**(2), 196–208 (1980)
38. I. Polian, Security aspects of analog and mixed-signal circuits, in *IEEE International Mixed-Signal Testing Workshop (IMSTW)*, 2016, pp. 1–6
39. S.E. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, M. Tehranipoor, A survey on chip to system reverse engineering. J. Emerg. Technol. Comput. Syst. **13**(1), 6:1–6:34 (2016)

40. J. Rajendran, M. Sam, O. Sinanoglu, R. Karri, Security analysis of integrated circuit camouflaging, in *ACM SIGSAC Conference on Computer & Communications Security*, 2013, pp. 709–720
41. J. Rajendran, H. Zhang, C. Zhang, G.S. Rose, Y. Pino, O. Sinanoglu, R. Karri, Fault analysis-based logic encryption. IEEE Trans. comput. **64**(2), 410–424 (2015)
42. M. Rostami, F. Koushanfar, R. Karri, A primer on hardware security: models, methods, and metrics. Proc. IEEE **102**(8), 1283–1295 (2014)
43. J. Roychowdhury, R. Melville, Delivering global DC convergence for large mixed-signal circuits via homotopy/continuation methods. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **25**(1), 66–78 (2006)
44. K.S. Subrmani, A. Antonopoulos, A.A. Abotabl, A. Nosratinia, Y. Makris, INFECT: INconsicuous FEC-based Trojan: a hardware attack on an 802.11a/g wireless network, in *IEEE Hardware Oriented Security and Trust Conference (HOST)*, 2017
45. M. Tehranipoor, F. Koushanfar, A survey of hardware Trojan taxonomy and detection. IEEE Des. Test Comput. **27**(1), 10–25 (2010)
46. M.M. Tehranipoor, U. Guin, D. Forte, Hardware IP watermarking, in *Counterfeit Integrated Circuits: Detection and Avoidance* (Springer International Publishing, Cham, 2015), pp. 203–222. doi:10.1007/978-3-319-11824-6_10, ISBN:978-3-319-11824-6, https://doi.org/10.1007/978-3-319-11824-6_10
47. Q. Wang, R.L. Geiger, Temperature signatures for performance assessment of circuits with undesired equilibrium states. Electron. Lett. **51**(22), 1756–1758 (2015)
48. Q. Wang, R.L. Geiger, D. Chen, Hardware Trojans embedded in the dynamic operation of analog and mixed-signal circuits, in *National Aerospace and Electronics Conference (NAECON)*, 2015, pp. 155–158
49. Q. Wang, R.L. Geiger, D.J. Chen, Challenges and opportunities for determining presence of multiple equilibrium points with circuit simulators, in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2014, pp. 406–409
50. Y.T. Wang, D. Chen, R.L. Geiger, Practical methods for verifying removal of Trojan stable operating points, in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 2658–2661
51. Y.T. Wang, D.J. Chen, R.L. Geiger, Effectiveness of circuit-level continuation methods for Trojan State Elimination verification, in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2013, pp. 1043–1046
52. Y.T. Wang, Q. Wang, D. Chen, R.L. Geiger, Hardware Trojan state detection for analog circuits and systems, in *IEEE National Aerospace and Electronics Conference*, 2014, pp. 364–367
53. K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, M. Tehranipoor, Hardware Trojans: lessons learned after one decade of research. ACM Trans. Des. Autom. Electron. Syst. **22**(1), 6:1–6:23 (2016)
54. K. Xiao, D. Forte, M. Tehranipoor, Circuit timing signature (CTS) for detection of counterfeit integrated circuits, in *Secure System Design and Trustable Computing*, ed. by C.H. Chang, M. Potkonjak (Springer International Publishing, Cham, 2016), pp. 211–239
55. K. Yang, M. Hicks, Q. Dong, T. Austin, D. Sylvester, A2: analog malicious hardware, in *IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 18–37
56. M.H. Zaki, O. Hasan, S. Tahar, G. Al-Sammane, Framework for formally verifying analog and mixed-signal designs, in *Computational Intelligence in Analog and Mixed-Signal (AMS) and Radio-Frequency (RF) Circuit Design*, ed. by M. Fakhfakh, E. Tlelo-Cuautle, P. Siarry (Springer International Publishing, Cham, 2015), pp. 115–145

Chapter 6
Hardware Trojans and Piracy of PCBs

Anirudh Iyengar and Swaroop Ghosh

6.1 Introduction

Authenticity and security concerns due to hardware Trojans and counterfeiting at the integrated circuit (IC) level have been studied extensively in recent times [1]. However, vulnerability with respect to hardware Trojans and cloning at higher levels of system abstraction, e.g., at printed circuit board (PCB) level, has not been well reported. PCBs are extensively used in every electronic system to enable electrical interconnections between components and for mechanical support. The emerging business model of PCB design and fabrication that favors extensive outsourcing and integration of untrusted components/entities in the PCB life cycle to lower manufacturing cost makes both malicious modification and cloning of PCBs highly feasible [2]. Closer inspection of several common electronic products (e.g., mobile devices and wearables) and their PCB manufacturers reveals that different parts of PCB designs (e.g., schematic and layout) are often carried out in different physical locations by different parties. Therefore the PCBs are greatly exposed to reverse engineering, counterfeiting, overproduction, Trojan insertion, and recycling. Unlike integrated circuits (ICs), where researchers have analyzed the various flavors of hardware security primitives [3, 4], PCB security primitives are still evolving. Previous studies have covered security of PCBs against piracy and various post-fabrication tampering attacks. JTAG (Joint Test Access Group) and other field programmability features, e.g., probe pins, unused sockets, and USB, have been extensively exploited by hackers to gain access to internal features of the designs

A. Iyengar (✉) • S. Ghosh
Department of Electrical engineering and computer science, Pennsylvania State University, Pennsylvania, PA 16802, USA
e-mail: asi7@psu.edu; szg212@engr.psu.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_6

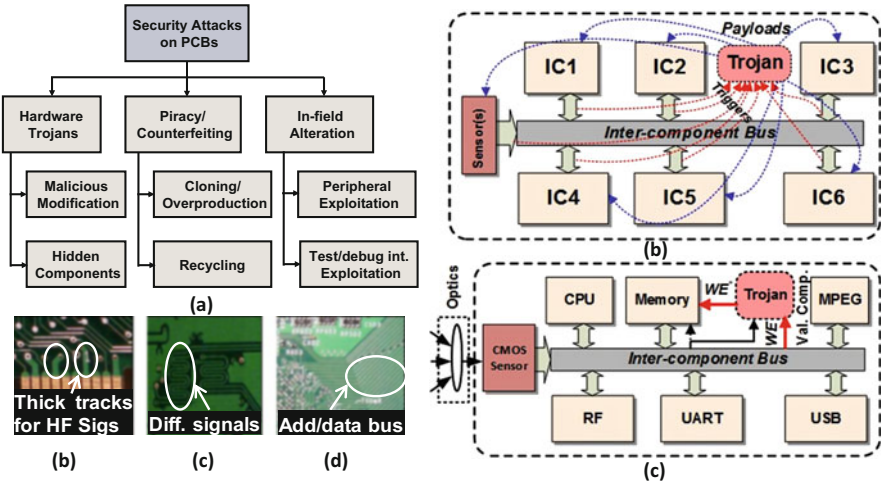


Fig. 6.1 (**a**) Taxonomy of various attacks on PCB; (**b**) general model of PCB-level hardware Trojan; (**c**) a specific example of Trojan affecting the external memory access in PCB. Vulnerabilities in PCB design with respect to Trojan attack: (**d**) thicker traces for high-frequency signals, (**e**) pair of signals for differential signaling, and (**f**) group of traces indicating bus [12]

[5] as well as snooping of secret key, collection of test responses, and manipulating JTAG test pins [6]. One instance demonstrated that Xbox can be hacked by disabling the digital rights management (DRM) policy using JTAG [5–7]. We note that modern PCBs are becoming increasingly vulnerable to malicious modification of PCBs during design or fabrication in untrusted design or fabrication facilities unlike the widely reported infield tampering attacks [3–9], such a vulnerability creates a new class of threat for PCBs.

This reliance to third-party manufacturing facilities makes the PCB fabrication process untrustworthy and, hence, vulnerable to malicious modifications, i.e., Trojan insertion and piracy, i.e., counterfeiting. Furthermore, an adversary can be present inside the design house, and the PCB design can be tampered with Trojans. Figure 6.1a shows broad classes of attacks on PCB including possible Trojan attacks.

In this chapter, we provide an overview of the various PCB security challenges [10]. We describe several instances of PCB attacks and review some of the more recent mitigation techniques/countermeasures. The chapter is organized as follows: The security challenges are introduced in Sect. 6.2.1. The attack instances of hard to detect Trojans are described in Sect. 6.2.2. Potential countermeasures are detailed in Sect. 6.2.3. Section 6.3.1 describes the authentication challenges, while Sect. 6.3.2 illustrates PUF structures. Finally, Sect. 6.3.3 provides a quantitative and qualitative analysis of the PUFs.

6.2 PCB Security Challenges, Attacks, and Countermeasures

6.2.1 Security Challenges

6.2.1.1 Security Analysis

An attacker can exploit the vulnerabilities, design/test features, and test hooks that are available on the board. Some common PCB features that can be exploited by an adversary for understanding a design intent and efficiently mounting a Trojan attack with minimal design modification are as follows:

(a) *JTAG interface* – JTAG is an industry standard to enable board-level test and debug. It can be exploited by a hacker to get a clue about the hidden test features or hidden control to access the data and address bus within the chip [5–7]. For example, hackers can deduce information about the length and properties of the instruction register through JTAG by trial and error. Next, a particular instruction can be executed to gain permission to tamper/feed internal data bus. JTAG can also be used to reverse engineer (RE) the board design by deducing the connectivity between components and executing external connectivity instructions. Aside from this, JTAG can be exploited too.

(b) *Test pins or probe pads* – Typical ICs contain several probe pads and test pins to observe/control important signals for test/debug purposes. A hacker can tap these pins and monitor the interesting signals to gain critical information about the functionality of the design or feed malicious data into the design. Test pins can also be used for RE where a test input can trigger certain data and address and control signals that can help identify the board functionality.

(c) *Infield alteration* via *ModChips* – This involves alterations caused by mounting integrated circuits, soldering wires, rerouting paths to avoid or substitute existing blocks, adding or replacing components, and exploiting traces, ports, or test interfaces in ingenuous manners [7, 11]. ModChips are one such class of devices that alter the functionality or disable restrictions within a system, such as computer or game console [12]. For instance, Xbox ModChips can modify or disable the built-in restrictions integrated in Xbox consoles, allowing to play pirated games in the tampered consoles.

(d) *Vulnerabilities in PCB design* – Figure 6.1d–f illustrates several additional vulnerabilities as described below:

  (i) *Distinct properties of special signals*: The thickness of clock and data bus provides clues to the hackers about the functionalities of these pins. Similarly, pins tied with identical pull-up/pull-down resistors indicate that they belong to a bus.

  (ii) *Remnant signatures from test/debug*: When pins (that are used for test/debug) are accessed through ports, the remnant of soldering provides clue to a hacker about the functionality of these pins. Similarly, an empty socket can be used for hacking purposes.

a

Security Vulnerability in PCB	Possible Exploitation for Trojan Attacks
JTAG	Hacking of scan chain / data bus for illegal access to memory / logic
RS232, USB, Firewire, Ethernet	Hacking of internal memory of an IC, leaking of secret data
Test Pins	Access to internal scan chain to leak data from ICs
Unused Pins, Multiple Layers, Hidden Vias	Alteration of connections using internal layers / hidden vias

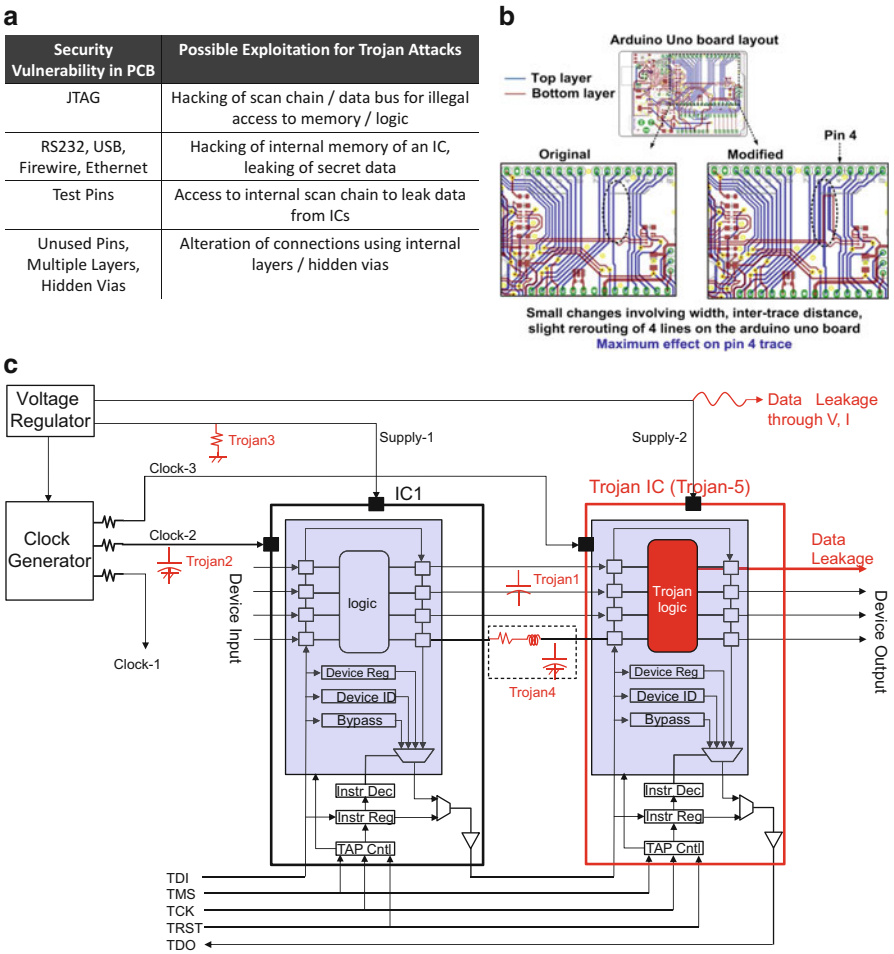


Fig. 6.2 (**a**) Vulnerability at PCB level with respect to Trojan attacks; (**b**) minor modifications in Arduino Uno board layout to insert Trojans without addition of any new component; and (**c**) examples of hardware Trojan insertion in a PCB component [10]

(iii) *Miscellaneous hints*: Figure 6.2a lists some additional vulnerabilities. Apart from the above component-level hooks, a PCB design itself provides lots of information to an adversary in fabrication house that can facilitate powerful Trojan attacks.

6.2.1.2 Attack Models

Trojan attacks in PCB can be divided into two broad classes, as described below:

1. *Case I*: The board design is trusted. In this model, the attack is mounted in the PCB fabrication house. It is expected that the attacker would change the design in a way that evades post-manufacturing test but causes functional deviations under certain rare conditions, which are unlikely to be triggered during test.
2. *Case II*: The board design is not trusted (e.g., outsourced). In this model, the attacker is assumed to be present in the board design or fabrication house. Only the functional and parametric specifications of the board are trusted. The attacker has higher flexibility of maliciously altering the design and/or choosing fake or untrustworthy (and potentially malicious) components. Again, an attacker would try to hide the modifications to avoid detection during functional and parametric testing process.

Note that in both cases, there are two possible objectives of the attacker: (1) malfunction and/or (2) information leakage. Next, the possible Trojan attacks of different forms in a PCB are described.

(a) *Signal trace modifications in the inner PCB layers*: For boards with fewer levels where hiding an extra component is difficult, an attacker can change the resistance, capacitance, or inductance of the signal traces (self, mutual). For example, signal trace in an internal layer can be made thinner to increase the resistance such that it fails during long hour of operation due to heating. Similarly, the metal coupling capacitance between two traces including traces in the supply planes can be increased (by changing trace dimensions, inter-trace distance via slight rerouting, and selective dielectric property modification) to trigger coupling-induced voltage and delay failures in one of the lines. Leakage resistance paths can also be incorporated in an internal layer trace for intentional voltage degradation. Impedance mismatch between interconnected traces can be introduced to cause malfunction in certain scenarios.

Modification in an example trace-4 scenario, causing malicious effects on circuit parameters (coupled voltage, delay failures, and voltage degradation), is illustrated in a commercial Arduino Uno board layout in Fig. 6.2b. It involves changing the thickness and inter-trace distances by 2X and rerouting of single trace. Even in a small two-layer board design like the Uno, these changes are minimal and difficult to detect during the test but can cause undesired functional behavior under certain scenarios. In complex PCB designs with more than four layers, these changes would be confined within a small area of an internal layer. Hence, the chances of detection with optical or X-ray-based imaging are extremely low. Functional testing is typically not exhaustive, and hence these can easily bypass detection. However, in certain rare conditions in field, their effect on circuit parameters can be significant causing system failures.

(b) *Hidden components*: For boards with more than two layers, an adversary can insert extra components in an internal layer to either leak information or conditionally cause malfunctions. Replacing a legitimate IC with Trojan-infected IC is another possibility (Fig. 6.2c). The tampered IC would be functionally equivalent to the legal IC; however, it may be equipped with capabilities to leak secret information. The leakage can be direct (through unused pins) or indirect (modulating supply voltage or current). Figure 6.1a summarizes different categories of PCB attacks.

6.2.2 Attack Instances

6.2.2.1 Design House Is Trusted

This scenario arises when the PCB is designed by a trusted designer and outsourced for fabrication. During the manufacturing process, malicious modifications can be intelligently inserted by an adversary, such that the final design structurally matches the original one (no additional components, logic, traces) but produces undesired functionality under certain conditions. The small alterations can be confined in the internal layers of a multilayer PCB. Hence, chances of detection with visual inspection, optical imaging, and X-ray-based imaging techniques are low. Besides, exhaustive functional testing is typically impossible with large number of test nodes. Therefore, the malicious functions are very likely not triggered during the in-circuit and boundary scan-based functional testing [13]. Usually modifications can be made in the existing traces to increase the mutual coupling capacitance, characteristic impedance, or loop inductance by changes in internal layer routing and small leakage path insertion. Additional components with ultra-low area and power requirements can also be inserted in the internal layers. Two Trojan examples are presented in this class. The first case considers a multilayer PCB (10 cm length) with possible application in a high-speed communication and video streaming systems. It has a common scenario of two high-frequency (HF) PCB traces in an internal layer running parallel to each other. Typically, HF traces are routed in the internal layer shielded by power and ground planes to avoid interference (Fig. 6.3a). However, it significantly complicates the internal layer testing and debug procedure and provides an opportunity to the attacker. The dimensions of the traces are designed optimally to carry normal HF signals, i.e., 1 oz. copper trace with width and thickness of 6 mils and 1.4 mils, respectively [14]. The dielectric is FR-4 with a relative permittivity of 4.5. The inter-trace distance is chosen to be 30–40 mils to avoid the negative effects of mutual inductive and capacitive coupling. These HF traces are modeled by lumped parametric form [14]. Functional simulation shows a maximum coupled near- and far-end voltage of $\sim$300 mV p-p on one of the traces, with the other trace swept with pulse voltages of 3 V p-p at 10–500 MHz with a 50% duty cycle. The maximum propagation delay for the pulse across the active trace is $\sim$0.4 ns.

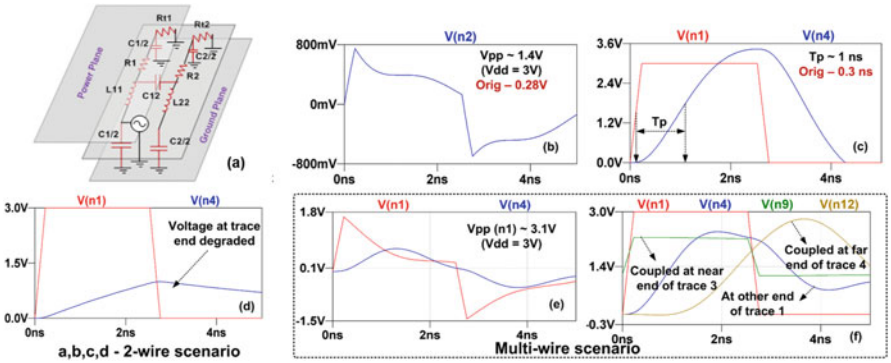


Fig. 6.3 Impact of Trojans in a PCB inserted by selective trace/s property modification, without addition of any new component (design house trusted scenario): (**a**) lumped trace-2 PCB model with associated capacitance, inductance, and resistance; (**b**) trace-2 near- and far-end voltages and (**c**) propagation delay in trace-1 (n1 and n4 nodes) at 220 MHz and 3 V p-p input; (**d**) effect on trace-1 far-end voltage, on insertion of a leakage resistance path from trace-1 to ground; (**e**) and (**f**) show the effect of change in trace properties in a four-wire scenario. Coupled voltages in near and far end of victim trace, when all aggressors are switching in phase at 220 MHz and 3 V peak-to-peak (p-p), are illustrated in (**e**), and voltage profile at near end (n9) of trace-3 and far end (n12) of trace-4 is shown in (**f**) [10]

With the simulation setup described above, we observe the effects of various trace level modifications during fabrication. The inter-trace distance in the internal layers is reduced by 2X, widths of both wires are increased by 2X, and the thickness is increased by 1.5X. These are minimal changes in a small target region of an internal layer, rendering them mostly undetectable during structural testing. The dielectric permittivity of the insulator between the traces is increased to 5.5 in order to model moisture retention in certain insulating areas, impurity addition to epoxy base [15], and, hence, the aging effect. Accelerated aging tests have a low probability of detecting this as the permittivity is selectively altered by an adversary in a small area. However, the effect of these changes on associated circuit parameters can be significant.

At 220 MHz, the near-end peak-to-peak voltage in trace-2 is ∼1.4 V for an input pulse voltage of 3 V p-p in trace-1 (Fig. 6.3b). This is an extraneous interference and may cause unexpected behavior in terms of erroneous circuit activation or feedback. The propagation delay increases by 2X, beyond 1 ns (Fig. 6.3c) which can induce functional failures for higher switching frequencies and greater trace lengths. From an attacker's perspective, inserting a small leakage path (2X the trace resistances) to ground can drain away the signal, resulting in a degraded, distorted waveform at the far end of trace-1 (Fig. 6.3d) and hence malfunction in the connecting circuits. This can easily evade detection by conventional PCB testing which is not exhaustive due to prohibitive cost and time to market.

The effect of coupling is more prominent when multiple HF traces, over different planes, are intentionally routed intelligently to increase mutual coupling. This can be

achieved by (a) bringing the in-plane neighboring traces closer and (b) increasing widths and thicknesses of the lines. These selective minute alterations are highly likely to pass structural and functional testing. However, the effect of these changes on the circuit performance could be significant as shown in Fig. 6.3e–f. The coupled voltages at the near and far end of victim trace-1 are 3.1 V p-p and 1.3 V p-p, respectively (Fig. 6.3e), with in-phase rising/falling transition on the adjacent three traces (one in-plane, one above, and one below). This is 34X greater than the case when active traces are switching in opposite directions. This interference would certainly result in failures in terms of erroneous activation, feedback, and performance. The voltage profile at the far end of trace-1 with the others inactive shows some distortion and an average propagation delay of 1 ns (Fig. 6.3f). Higher number of neighboring traces and greater trace lengths significantly affect the propagation delay and cause delay failures at high switching speeds. Extraneous coupled voltages in trace-3 and trace-4 are illustrated in Fig. 6.3f as well. It can be noted from the above results that detecting these Trojans is extremely hard since it is sensitized under very rare specific conditions. In the multi-wire scenario, the degraded performance is prominent only in two out of eight possible combinations of transition polarity (i.e., all rising/falling pulses) in the three neighboring traces. The frequency of operation and the input vector patterns serve as two example conditional triggers for these Trojans, inserted by selective trace property and routing alterations during PCB fabrication.

6.2.2.2 Design House Is Untrusted

In this attack model, both PCB design and fabrication are done in untrusted facilities, thus increasing the possibility of Trojan attacks. The board specifications generated by the system designer are trusted. Post-manufacturing testing is done by the system designer to ensure the board performance and functionality. Hence, along with the possibilities of trace level alterations, an attacker can also leverage the opportunity to modify the design structurally and/or insert additional components that would be activated upon certain trigger conditions. The design alterations would be hidden intelligently (e.g., physically or by rare input conditions) to evade detection during post-fabrication structural and functional testing. A simple example of this attack model is illustrated in Fig. 6.4, where a fan controller adjusts the speed of a 12 V DC brushless fan based on the input received from a temperature sensor. The sensor provides 0–5 V (depending on the current temperature) which is digitized by an ADC and sent to a microcontroller that controls the speed of the fan through linear regulation of the fan input voltage.

With minor structural modification, an attacker can maliciously tamper with the functionality of the circuit (Fig. 6.4a). In this case, the Trojan prevents the microcontroller from obtaining an accurate temperature. It includes a resistance, a capacitor, and a PMOS transistor. The capacitor is charged using the output from a voltage regulator (LM317) connected to the fan. The time needed to activate the Trojan (i.e., the trigger condition) can be adjusted by manipulating the resistance and

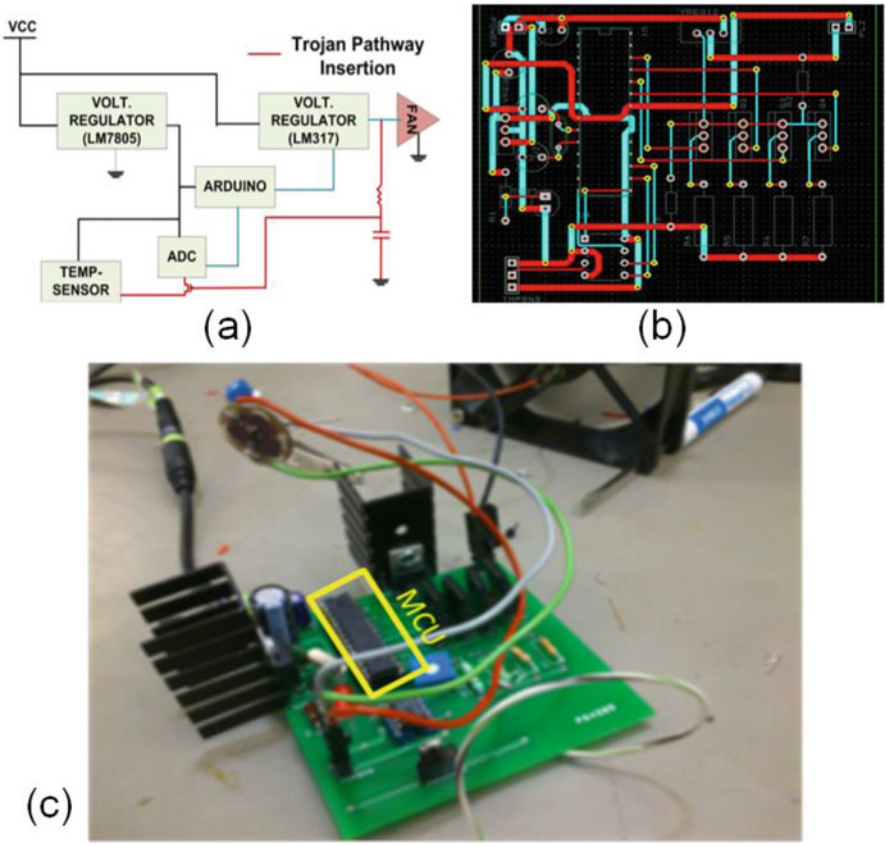


Fig. 6.4 (**a**) Fan controller circuit with a Trojan insertion; (**b**) a two-layer PCB layout of the original circuit; and (**c**) a fabricated PCB board which demonstrates triggering and payload of the Trojan [10]

capacitance values. The trigger deactivates the PMOS transistor inserted between the temperature sensor and the ADC, effectively disabling the connection. Hence, the microcontroller receives a null input that is interpreted as a very low temperature, and the fan speed is reduced significantly. With a large value of time constant, this design alteration has a high probability of evading functional testing. A two-layer PCB schematic of the fan controller (prefabrication) is shown in Fig. 6.4b. A fabricated PCB board to demonstrate the triggering and payload of the Trojan is shown in Fig. 6.4c.

From the above discussion, it is obvious that achieving these minor structural modifications in such a scenario is easy due to outsourcing of both design and fabrication. Since system integrator only possesses information about the major PCB components (constituent ICs) and functional specifications, such small alternations can go undetected. From the attacker's perspective, a multilayered PCB is more

attractive because it provides increased opportunities to hide the design changes. In addition to the structural changes, an adversary in the foundry can perform layout modifications to deliberately insert trace level Trojans, which are difficult to identify by functional tests.

6.2.3 Possible Countermeasures

6.2.3.1 Hardware Trojan Detection in PCB

In [10] a noninvasive RE and multiparameter side-channel analysis to detect the embedded Trojans is proposed. First, the Trojans are categorized by their nature, e.g., Trojans that induce (a) parametric failures, such as unacceptable delay and leakage (Trpar), (b) large static power (Trpwr), and (c) functional failures (Trfn). Trojans in each category are treated differently for detection. Next, a criticality analysis is performed to isolate the vulnerable nodes. This step is conducted to identify the critical signals from design specification, e.g., clock, control signals, data, and address bus. The PCB layout information is captured in the analysis to identify the potential Trojans (i.e., the longest trace that runs in parallel with the victim signal). For analysis we assume that (a) an intended PCB design is available to the validation engineer, (b) a single Trojan is inserted at a time, (c) a Trojan injection is limited to neighboring traces, and (d) the tampering does not involve change in the signal routing.

Test pattern generation for Trojan detection can leverage on the principles of conventional testing. The PCB layout and criticality analysis generate a list of possible triggers/payloads and their locations. Test patterns are generated to sensitize the trigger conditions of each Trojan and observe its effect. The process continues till all the Trojans in the list are exhausted. Side-channel analysis such as delay, frequency, static leakage, and dynamic current can also be administered to sensitize Trojans of other types. Side-channel analysis, however, would require a set of golden PCBs. The Trojan coverage and the test patterns are output of the proposed methodology (Fig. 6.5a). The test patterns obtained above are applied to both extracted parasitic model and actual PCB, and their responses are compared to detect Trojans. Figure 6.5b illustrates this methodology for capacitive Trojans through an example where net n3 is identified as the target signal from node criticality analysis. Therefore, the Trojan list would contain {c1, c5, c6}. For this example, test pattern will target to toggle nets n1, n3, and n5 in association with n3 to sensitize the Trojans for detection. Note that c2 is excluded from the list due to shorter track segment of n2 in parallel with n3.

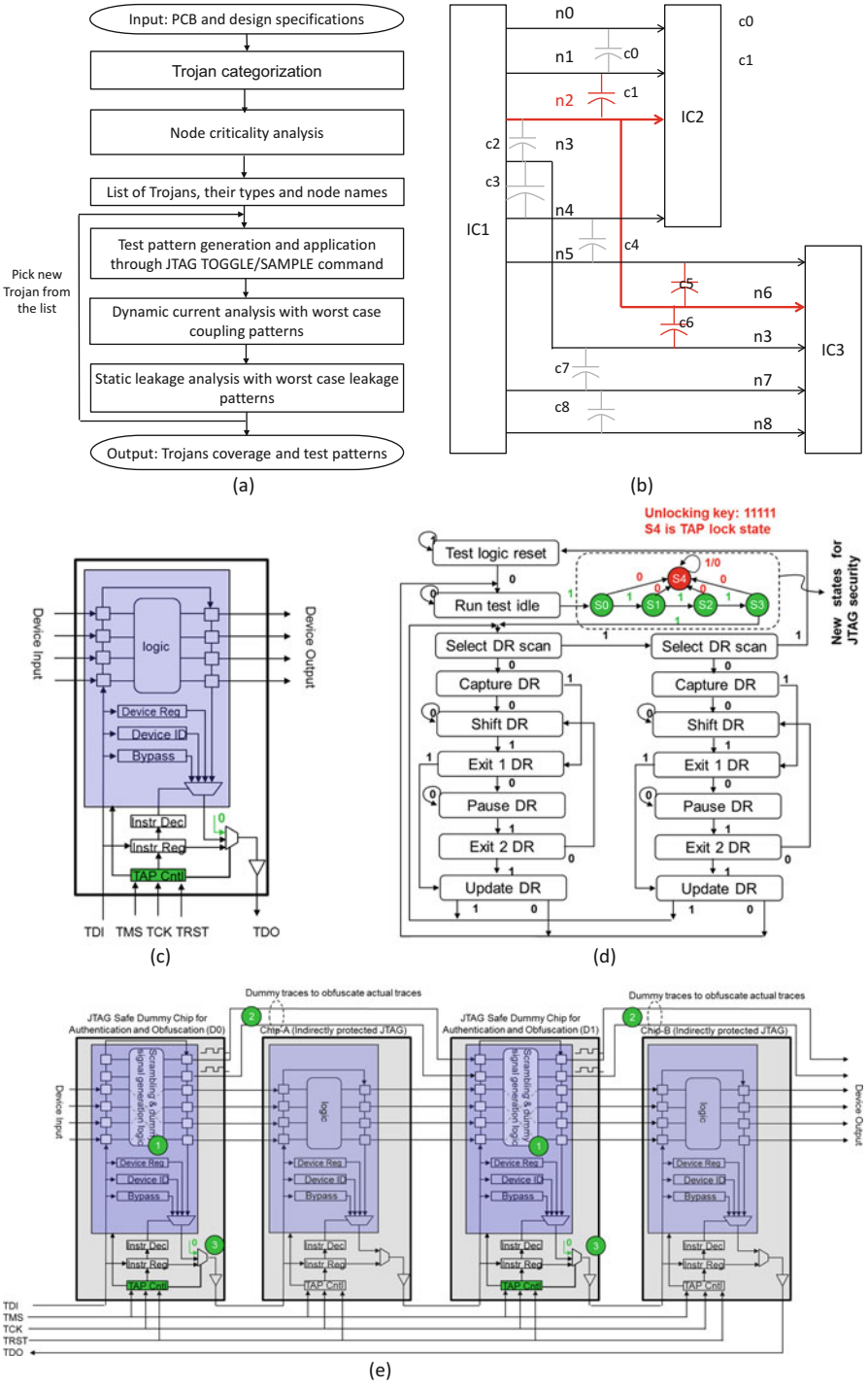


Fig. 6.5 (**a**) Flowchart of the Trojan detection methodology [10]; (**b**) example illustrating the capacitive Trojan detection; (**c**) modified tamperproof JTAG interface incorporating the TAP controller; (**d**) TAP controller state diagram; and (**e**) incorporation of dummy ICs to obfuscate the PCB interconnects

6.2.3.2 Preventive Countermeasures Through Hardening

The following proactive techniques are proposed in [10] to protect against Trojan attacks in PCBs:

1. *Secure interfaces*: Conventional JTAG cannot prevent unauthorized access, and therefore it can be exposed for tampering. The security features inside JTAG are updated so that the access to instruction and data registers is restricted until a code/password is fed to unlock the TAP controller. Figure 6.5c shows the JTAG structure containing TAP controller that provides access to the instruction and data register (device registers, ID register, and bypass register) based on TMS and TCK. The modified TAP controller (Fig. 6.5d) includes newly added states (S0–S3) that look for a certain security key. In order to allow legitimate access (for installing upgrades and patches), the secure JTAG can be unlocked by feeding the correct keyword (0011111 in this case). However, in the event of wrong keyword, the controller will go to a lock state (S4) where it will disconnect the TDI and TDO by feeding a fixed value to TDO. Note that once the TAP goes to the lock state, it cannot be reset back to factory state preventing the possibility of a trial-and-error method by the hacker.
2. *Secure PCB*: Conventionally designed PCB traces can be reverse engineered to discover the board design. An approach is proposed in [10] to address this challenge through interconnect obfuscation. Figure 6.5e shows one possible approach to obfuscate the interconnects by introducing dummy ICs that serves three purposes:
   (a) It scrambles the traces based on a scrambling function implemented in the dummy logic. The dummy device is RE resistant due to the presence of secure JTAG that prevents access to internal design without proper authentication.
   (b) It provides dummy outputs that are mixed with real traces to confuse the hacker. The dummy traces are driven by random logic and counters to obfuscate the data, address, and clock signals. Furthermore, the dummy traces are drawn in the same way as the real data and clock signals to obfuscate the real signals.
   (c) It implements secure JTAG; therefore the access to real chips is also secured even if they implement unsecured JTAG. In order to RE the real chips, the hacker needs to unlock the JTAG of two dummy chips (D0 and D1) each of them may have different security keywords. RE through JTAG could be deterred further by incorporating the security states in the TAP controller of the real chips as well.

6.2.3.3 Tamper Detection and Prevention

In this approach, a real-time system monitoring is enforced in order to prevent tampering [11]. A critical trace is identified and is compared with an expected value.

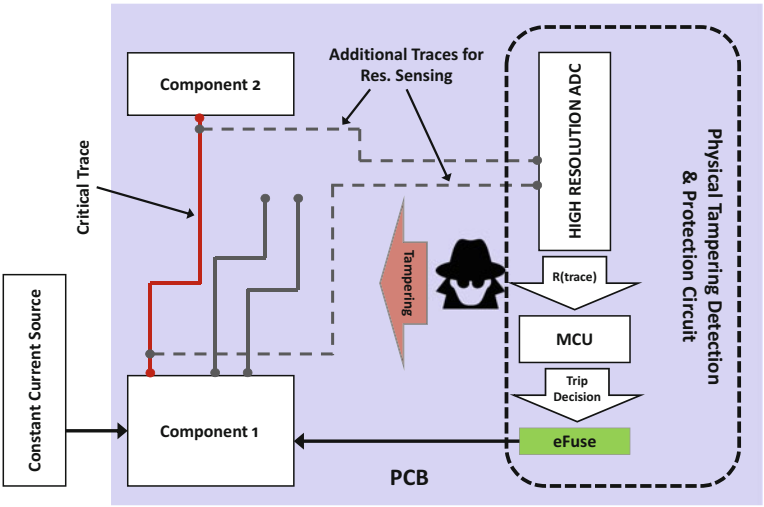


Fig. 6.6 A functional block diagram of the prototype tamper detection and prevention circuit on a PCB [11]

Any deviation from this value can be identified by which the attack can be prevented. Figure 6.6 describes the general methodology, where the components of interest are identified along with their critical trace. During an attack, the resistance of the critical trace will vary, which is correspondingly identified by the tamper detection circuit. This is then relayed to an eFuse to shut down the device.

6.3 PCB Authentication Challenges and Prospective Solutions

Aside from the Trojan attacks, PCBs are highly susceptible to cloning. This is further exacerbated by the high distributive manufacture process. To solve similar problem, ICs typically employ a physically unclonable function (PUF) [16], which exploits device specific parameters toward generating a unique and authentic signature. Such features however are still evolving in PCBs. One such work is the *BoardPUF* [17], which employs the capacitance difference of an embedded comb-like structure to generate response for a given challenge during authentication. Figure 6.6 illustrates the architecture of the BoardPUF, with the variation source to be the capacitor which is composed of a set of well-designed comb-like pattern which is placed between layers of PCB, to reflect the variations observed during PCB manufacture. A counter is used to record the signals by the sensing circuits for a given time interval. This is then compared with counts from different compare units specified by the compare pairs through which the final ID is generated.

Although attractive, most of the authentication process is done via the on-board chip, which can be corrupted and can lead to denial of service.

Another PCB-based PUF authentication methodology, proposed in [18], exploits the impedance of PCB traces toward generating a unique signature at no area overhead. The trace impedance being unique to each PCB acts as a good source of authentication. However, one of the major drawbacks of this technique is the sheer lack of eligible traces that can be directly probed. As most of the PCB interconnections are in the internal layers, gaining access to a significant set of interconnects at the top/bottom layer is difficult. This limits the number of responses that can be obtained from the PCBs. Aside from this, work done by Andrew et al. [7], utilizes the Boundry Scan Chain Architecture (BSA) design on the JTAG, to create a unique signature from each PCB for authentication. The technique exploits the inherent delay of the BSA for authentication. Due to manufacture-induced process variations, the delay value of the BSA also varies correspondingly. Therefore, the resultant delay is unique to each device. By correctly determining the delay, one can distinguish one device's delay with another. Thus, allowing us to deterministically authenticate the device at hand. In an effort to realize a "strong-PUF," Anirudh et al. [19] proposed a unique PCB-based PUF design that captures a high degree of process variation and also facilitates a large number of challenge-based responses. Before dwelling into the design, we first need to understand the various authentication challenges the PCBs face.

6.3.1 PCB Variations and Authentication Challenges

6.3.1.1 Sources of Variations

PCB manufacturing process can introduce various physical, electrical, and chemical variations. These variations are broadly categorized as follows [19]:

(a) *Variations in trace physical dimensions:* This variation alters the electrical properties of the trace such as resistance, inductance, and capacitance causing a deviation from the expected behavior.
(b) *Variations in* via *dimensions:* This leads to an overall variation of via impedance. Larger the number of vias for a single signal, greater is the impedance variation.
(c) *Density variations:* Variations in deposition density also leads to impedance variations. Cavities (mouse bites) are observed in long traces, causing a shift in the electrical characteristic of the trace [20].
(d) *Alignment between layers:* Misalignment of masks during manufacturing leads to variation in trace deposition.
(e) *Layer lamination:* Unevenly thick layers can result in some vias being taller than the others and can also impact embedded capacitance, which too impacts its intrinsic impedance.

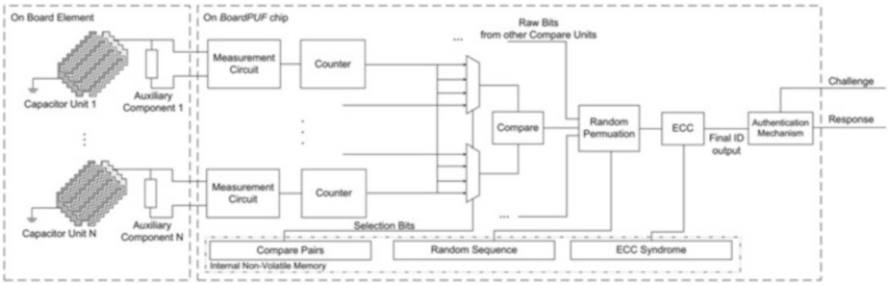


Fig. 6.7 BoardPUF architecture [17]

The above variations sources affect each PCB differently, making them perfect candidates for PCB authentication (Fig. 6.7).

6.3.1.2 Qualitative Analysis of Authentication Issues and Metrics

One of the primary issues with PCB authentication using PUF is limited number of challenges. Each PUF structure such as embedded comb [17] and PCB trace [18] acts as a single challenge and provides single response. This is unlike IC PUF where linear and exponential number of Challenges Response Pairs (CRPs) can be generated. Generating exponential CRPs at low area overhead is a challenge. Designing passive structures to convert the PCB variability (as discussed in Section II.A) into a response is another key issue.

A key metric used to determine the quality of the PUF is the Hamming Distance (HD). HD is the amount of variation one PUF signature is with another. For a good (strong) PUF, the HD should be ideally 50%. The simple HD can be divided into inter-die HD and intra-die HD. Where the inter-die HD is the HD between two PUF responses of different devices, while the intra-die HD is the variation in the PUF signature of one device w.r.t. environmental changes such as temperature, humidity etc.

The concept of inter-die Hamming Distance (HD) metric can be extended to obtain inter-PCB HD. The objective of this metric is to ensure sufficient difference between responses of two PCBs for the same challenge (i.e., PUF structure in this case). The concept of intra-HD used in IC authentication may not be readily applicable to PCB though. This is due to the fact that temperature and voltage during authentication depends on usage scenario. When the external bare PCB is authenticated by the vendor, they can follow the exact voltage and temperature conditions specified by the PCB manufacturer during authentication. Therefore, intra-PCB HD will be 0%. However, when the PCB is assembled and deployed in product it could be interrogated by the secure facility such as bank. Under this circumstance the ambient temperature could be different. The voltage could still be the same; however, the environmental moisture could be different. Therefore, the

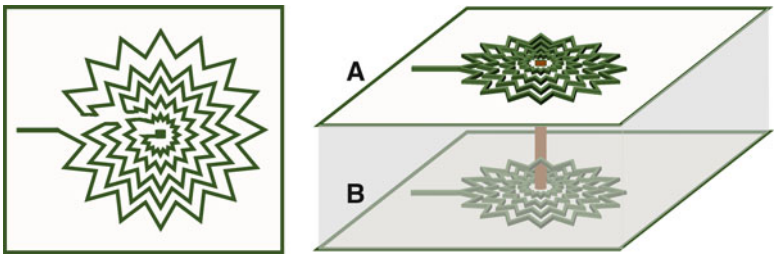


Fig. 6.8 PUF features: coil (**a**) *top view* and (**b**) structure comprising of symmetric zigzag coils separated by a dielectric. The notches capture edge rounding variations in resistance and inductance whereas two coils in two layers capture the capacitance variation due to misalignment [20]

intra-PCB HD should consider temperature and moisture variation to determine the stability.

6.3.2 Prospective PUF Structures

A passive structure has been proposed [19] that can capture various sources of variations to provide a unique device signature [20]. Figure 6.8 illustrates a coil-type structure which is designed using Allegro [21] PCB editor. Compared to a typical straight coil, this structure exhibits additional resistance (wire resistance), capacitance (enhanced due to the comb-shaped multilayer design) and inductance (enhanced due to the coil shape) variation due to the high number of notches. Therefore, this structure is suitable for capturing the many sources of manufacturing process variations including edge rounding, density, and alignment variation.

6.3.2.1 Star-Coil PUF

This PUF works by exploiting the resonance frequency (RF) of the star-coil for authentication. RF being unique to each coil (due to unique set of process variation) acts as a good source of authentication. The star-coil is excited by a voltage source, whose frequency is swept from a given minima to maxima. The frequency at which the current through the coil is the highest, i.e., impedance is the least, is its resonance frequency. The resonant frequency for an RLC circuit is given by:

$$f_{\text{res}} = {}^{1}\Big/_{2.\pi.\sqrt{\text{L.C}}} \tag{6.1}$$

where "f_{res}" is the resonance frequency in Hz, "L" is the inductance in Henry, and "C" is the capacitance in Farads. Due to the inverse relationship between impedance (ignoring resistance) and resonance frequency, a small change in the impedance

results in a large change in the resonance frequency. In order to understand the impact of process variation on the resistance (R), inductance (L), and capacitance (C) of the coil, a 10% introduced variation (assuming it to be 3σ variation) on the trace length, width and dielectric thickness. We know from [19] that a linear relationship of R, L, and C with respect to the trace length, width, and thickness persists. This observation is used in determining the mean and standard deviation of R, L, and C due to each variation.

$$\Delta 1 = \text{guassian distribution of trace length}$$

$$\Delta 2 = \text{guassian distribution of trace width}$$

$$\Delta 3 = \text{guassian distribution of dielectric width}$$

$$\begin{aligned}\text{Total Inductance (L)} = {} & \text{Inductance}\,(\Delta_1) + \text{Inductance}\,(\Delta_2) \\ & + \text{Inductance}\ (\Delta_3) + \mu_{\mathrm{L}}. \end{aligned} \tag{6.2}$$

$$\begin{aligned}\text{Total Capacitance (C)} = {} & \text{Capacitance}\,(\Delta_1) + \text{Capacitance}\,(\Delta_2) \\ & + \text{Capacitance}\,(\Delta_3) + \mu_{\mathrm{C}}. \end{aligned} \tag{6.3}$$

$$\begin{aligned}\text{Total Resistance (R)} = {} & \text{Resistance}\,(\Delta_1) + \text{Resistance}\,(\Delta_2) \\ & + \text{Resistance}\,(\Delta_3) + \mu_{\mathrm{R}}. \end{aligned} \tag{6.4}$$

where μ_{L}, μ_{C}, and μ_{R} are mean values of inductance, capacitance, and resistance, respectively.

By combining the effects of trace length, width, and dielectric thickness on the overall R, L, and C (using 2–4), a cumulative mean and standard deviation of the star-coil PUF is obtained. A 10000-point Monte Carlo analysis performed on the trace length, width, and dielectric thickness reveals the corresponding resonant frequency distribution. The plot is as shown in Fig. 6.9. A spread of 10 MHz in the resonant frequency is observed.

This design can be extended to incorporate more variations by using multiple such coils connected together in a serial or parallel fashion (illustrated in Fig. 6.10a–d). This allows us to exploit the use of multiple layers (thickness variations, lead to impedance variations), which further adds to the variations that the circuit experiences. Measuring the resonance frequency of this stack will provide a signature truly unique to the PCB.

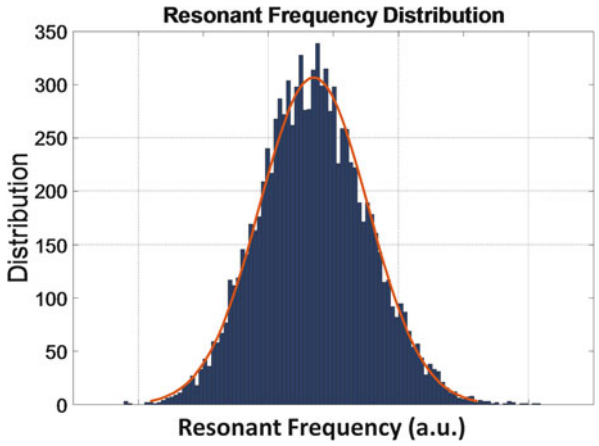


Fig. 6.9 Resonance frequency distribution [19]

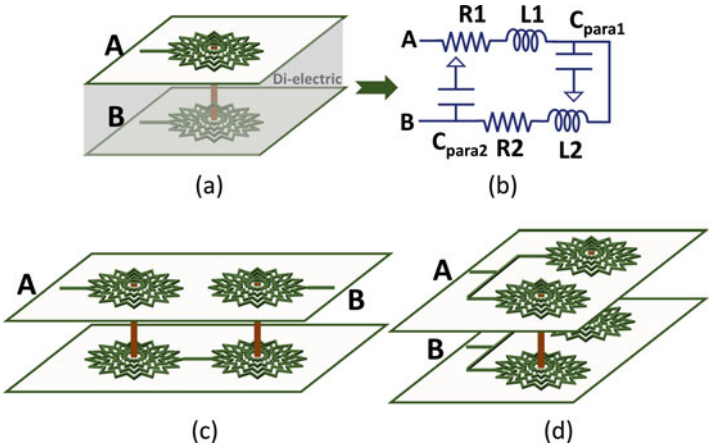


Fig. 6.10 Star-coil structure: (**a**) base configuration, (**b**) equivalent RLC circuit, (**c**) series connected coils, and (**d**) parallel coil combination [19]

6.3.2.2 Arbiter-Coil PUF

One major limitation of the coil PUF is that, it has only one possible outcome (one signature) for authentication. This does not adequately provide us linear or exponential number of responses for reliable authentication. In order to improve the CRPs a passive arbiter-PUF is proposed [17] where several stages of star-coils are connected in various possible combinations to form a path through external jumpers (manually connected during authentication) (as seen in Fig. 6.11). By varying the jumper connection in accordance to a certain "challenge", a corresponding response unique to the path is obtained. The process can be repeated for different challenges (paths) to get additional resonance frequency values. By comparing all these values to the values stored in the database, a PCB can be authenticated with a high degree

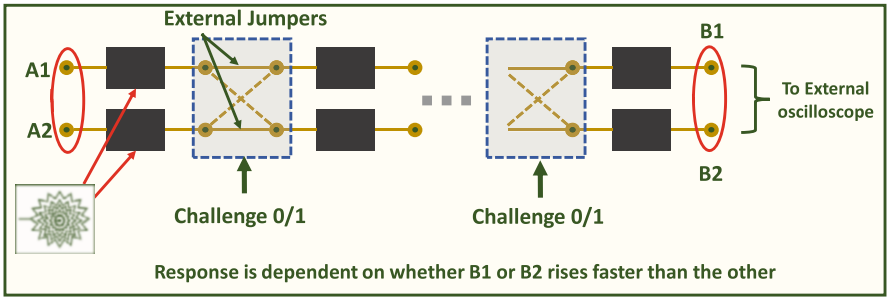


Fig. 6.11 Arbiter design-based PCB PUF [19]

of reliability. This PUF is considered as a strong-PUF due to its exponential increase in CRPs with the increase in the number of rows and/or stages of coil PUFs. Note that other types of passive structures (such as straight traces, comb traces) can also be used in the individual PUF stage instead of the star-coil.

Aside from the abovementioned resonant frequency determination approach, the arbiter-coil PUF can be used to measure the delay difference between the two paths (with cascaded stages). As the rise/fall of the signal is directly proportional to the capacitance and inductance of each individual coil, the cumulative effect will provide an alternative means of authentication. To show proof of concept, Anirudh et al. [19] show a 1 stage, 128 copy arbiter-coil-based delay PUF. A Gaussian distribution on the R, L, and C obtained from allegro is incorporated into HSPICE for the delay based analysis. The PUF signature is realized by exploiting the delay difference between the two paths. By having 128 such copies, we are able to obtain a 128-bit signature. It must be noted that the response in this example is static, and does not vary in w.r.t. the challenge-response methodology. Additionally, care must be taken to ensure correct termination to allow a reasonably accurate testing.

6.3.3 Qualitative and Quantitative Analysis

In order to perform inter- and intra-PCB HD, it is necessary to first determine the min and max frequency between which the authentication signal frequency will be swept. This is based on the calculated resonance frequency. The frequency range between max and min is digitized with 2 N where N is the number of bits in response. The resonance frequency of each PUF is associated with its digital equivalent, i.e., min frequency is all 0 s and max frequency is all 1 s. This digitization will require either an embedded hardware or an external digitizer. By comparing the response generated between the various PCBs, the inter-PCB hamming distance is observed. Next, the arbiter-coil-based delay PUF is subjected to an inter-die HD analysis, additionally, by subjecting the PCB to temperature and moisture variations, the intra-PCB HD can be obtained.

6.4 Conclusions

PCB is a potential target of broad range of new attack vectors such as cloning, Trojan insertion, Modchips, and reverse engineering. PCB authentication brings new challenges that were absent in conventional IC authentication. We have shown that clever localized modifications in PCB during design or fabrication can evade conventional structural and functional testing. They can lead to malicious and often catastrophic outcomes during field operation. We have also presented two possible countermeasures through judicious trust validation and low-cost design approaches. With growing complexity of multilayer PCBs including hidden vias and increasing reliance on third-party resources, more complex Trojan attacks would become feasible. Due to the widespread use of PCBs, their vulnerability to Trojan attacks poses major concerns in trust and security of electronic products. Untrusted PCBs can enable highly sophisticated and powerful attacks such as unauthorized access to a system or wirelessly transmitting secret data. Aside from Trojan insertion, piracy via counterfeiting of PCBs pose a significant threat as well. In this chapter we presented a systematic methodology to authenticate the PCBs using PCB PUF.

Acknowledgments This work supported by Semiconductor Research Corporation (#2727.001), National Science Foundation (CNS-1441757), and Defense Advanced Research Projects Agency under award #D15AP00089.

References

1. R.S. Chakraborty, S. Narasimhan, S. Bhunia, Hardware Trojan: Threats and Emerging Solutions. in *Proceedings of the IEEE International HLDVT Workshop*, pp. 166–171, 2009
2. PCB manufacturers look forward to Ultrabook and Wintel, http://www.chinapcbs.com/
3. Y. Alkabani, F. Koushanfar, Consistency-based characterization for IC Trojan detection, International Conference on Computer-Aided Design, 2009
4. H. Salmani, M. Tehranipoor, J. Plusquellic, A layout-aware approach for improving localized switching to detect hardware Trojans in Integrated Circuits, IEEE International Workshop on Information Forensics and Security (WIFS), 2010
5. F. Domke, Blackbox JTAG Reverse Engineering, http://events.ccc.de/congress/2009/Fahrplan/events/3670.en.html, 2009
6. K. Rosenfeld, R. Karri, Attacks and defenses for JTAG. IEEE Des Test Comput **27**(1), 36–47 (2010)
7. A. Hennessy, Y. Zheng, S. Bhunia, JTAG-based robust PCB authentication for protection against counterfeiting attacks. In Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific (pp. 56–61). IEEE, January 2016
8. K. Rosenfeld, R. Karri, *Security and Testing, Introduction to Hardware Security and Trust* (Springer, 2012)
9. N. Asadizanjani, A new methodology to protect PCBs from non-destructive reverse engineering, in *42nd International Symposium for Testing and Failure Analysis* (6–10 November 2016). Asm
10. S. Ghosh, A. Basak, S. Bhunia, How secure are printed circuit boards against trojan attacks? IEEE Des Test **32**(2), 7–16 (2015)

11. S. Paley, T. Hoque, S. Bhunia, Active protection against PCB physical tampering, in *Quality Electronic Design (ISQED), 2016 17th International Symposium on,* pp. 356–361. IEEE, March 2016
12. Modchips: Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Modchips
13. Printed Circuit Board Test Methodologies, http://download.intel.com/design/chipsets/applnots/29817901.pdf
14. J. Carlsson, *Crosstalk on Printed Circuit Boards*, 2nd edn., 1994
15. B. Sood, M. Pecht, Controlling Moisture in Printed Circuit Boards, IPC Apex EXPO Proceedings, 2010
16. G.E. Suh, S. Devadas, Physical unclonable functions for device authentication and secret key generation, in *Proceedings of the 44th annual Design Automation Conference* (ACM, 2007) pp. 9–14
17. L. Wei, C. Song, Y. Liu, J. Zhang, F. Yuan, X. Qiang, BoardPUF: Physical Unclonable Functions for printed circuit board authentication, in *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pp. 152–158. IEEE, 2015
18. F. Zhang, A. Hennessy, S. Bhunia, Robust counterfeit PCB detection exploiting intrinsic trace impedance variations, in *2015 IEEE 33rd VLSI Test Symposium (VTS)*, pp. 1–6. IEEE, 2015
19. A.S. Iyengar, Authentication of Printed Circuit Boards. In 42nd International Symposium for Testing and Failure Analysis (November 6–10, 2016). Asm
20. H. Rau, C.-H. Wu, Automatic optical inspection for detecting defects on printed circuit board inner layers. Int. J. Adv. Manuf. Technol. **25**(9–10), 940–946 (2005)
21. https://www.cadence.com/content/cadence-www/global/en_US/home/tools/allegro-downloads-start.html

Part III
Detection: Logic Testing

Chapter 7
Logic Testing for Hardware Trojan Detection

Vidya Govindan and Rajat Subhra Chakraborty

7.1 Introduction

The issue of ensuring *Trust* has become one of the biggest challenges in the semiconductor integrated circuit (IC) industry [1, 2, 9]. This issue has become prominent recently due to widespread outsourcing of the IC manufacturing processes to untrusted foundries in order to reduce cost. It is estimated that a considerable fraction of the ICs currently deployed and in circulation in the world are actually counterfeit [15]. A greater threat is the threat of an adversary potentially tampering a design in these fabrication facilities by the insertion of malicious circuitry termed *hardware Trojans*. An intelligent adversary will try to hide such tampering by ensuring that the IC's functional behavior is modified in a way that makes it extremely difficult to detect such tampering with conventional post-manufacturing test [9]. Intuitively, it means that the adversary would ensure that such a tampering is manifested or *triggered* under very rare conditions at the internal nodes, which are unlikely to arise during test but can occur during long hours of field operation [30].

Figure 7.1 shows general models and examples of Hardware Trojans. The *combinational Trojans* as shown in Fig. 7.1a do not contain any sequential elements and depend only on the simultaneous occurrence of a set of rare node conditions (e.g., on nodes T_1 through node T_n) to trigger a malfunction. The *sequential Trojans* shown in Fig. 7.1b, on the other hand, undergo a sequence of state transitions (S_1 through S_n) before triggering a malfunction. We refer to the condition(s) of inserted hardware Trojan activation as the *triggering condition* and the node affected by the hardware Trojan as its *payload*.

V. Govindan (✉) • R.S. Chakraborty
Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, 721302 Kharagpur, West Bengal, India
e-mail: vidya.govindan@iitkgp.ac.in; rschakraborty@cse.iitkgp.ernet.in

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_7

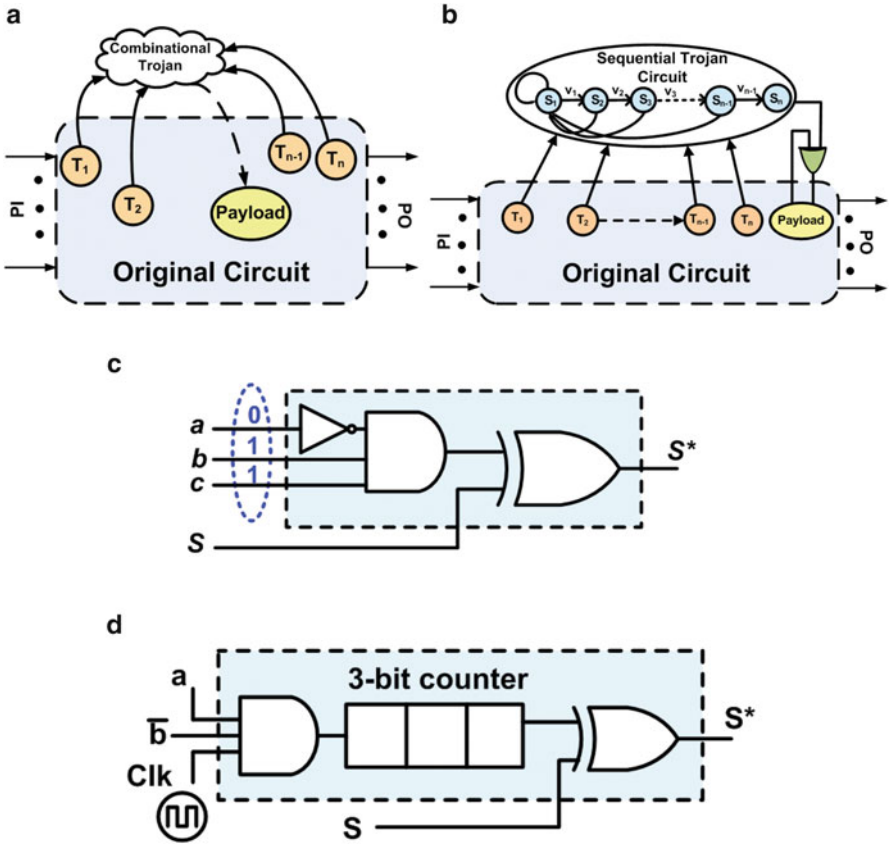


Fig. 7.1 Generic model for combinational and sequential Trojan circuits and corresponding examples. (**a**) Generic comb. Trojan. (**b**) Generic sequential Trojan. (**c**) Comb. Trojan example. (**d**) Sequential Trojan example

In order to detect the existence of a Trojan using logic testing, it is important not only to trigger a rare event at a set of internal nodes but also to propagate the effect of such an event at the payload to an output node and observe it. Hence, it is very challenging to solve the problem of Trojan detection using conventional test generation and application, which are designed to detect manufacturing defects. Most existing research on hardware Trojans mainly focuses on the modeling and detection of hardware Trojans [7, 20, 26, 27, 29, 32]. A vast majority of the detection mechanisms proposed till date utilize the anomaly in the side-channel signatures (e.g., delay, transient, and leakage power) in the presence of hardware Trojan in the circuit [5, 17, 29]. However, side-channel approaches are susceptible to experimental and process variation noise. Thus, detection of small Trojans (less than ten NAND-2 gate equivalents), especially combinatorial ones, becomes challenging through these approaches. Another approach is to employ design modification

techniques to either prevent Trojan insertion or to make inserted Trojans more easily detectable [6, 27, 31].

Usually, it is assumed that the attacker will generate the trigger signal from a combination of internal nets of the circuit whose transition probability is very low. She may try to activate them simultaneously to their rare values thus achieving an extremely low triggering probability. Based on this assumption, several Trojan detection techniques have been proposed till date [8, 27] which try to activate the Trojans either fully or partially by triggering the rare nodes, thus creating anomaly at the output logic values or in some side-channel signals, viz., transient power. In [27, 32], the authors proposed a design-for-testability (DFT) technique which inserts dummy scan flip-flops to make the transition probability of low transition nets higher in a special "authentication mode." However, it was found that careful attackers can evade this scheme easily [28]. Another powerful DFT technique is obfuscating or encrypting the design by inserting some extra gates in it [6, 10], so that the actual functionality of the circuit is hidden, consequently making it difficult for an adversary to estimate the actual transition probabilities at the internal nodes. However, such "logic encryption" schemes have also been broken [23].

In the sections that follow, we first propose a methodology, referred to as *MERO* (**m**ultiple **e**xcitation of **r**are **o**ccurence) for statistical test generation and coverage determination of hardware Trojans. The basic concept is to detect low probability conditions at the internal nodes, select candidate Trojans triggerable by a subset of these rare conditions, and then derive an optimal set of vectors that can trigger each of the selected low probability nodes *individually to their rare logic values multiple times* (e.g., at least N times, where N is a given parameter). The proposed methodology is conceptually similar to *N-detect test* [3, 21] used in stuck-at ATPG (automatic test pattern generation), where test set is generated to detect each single stuck-at fault in a circuit by at least N different patterns, in the process improving test quality and defect coverage [21]. We focus on digital Trojans [30], which can be inserted into a design either in a design house (e.g., by untrusted CAD tool or IP) or in a foundry. We do not consider the Trojans where the triggering mechanism or effect is analog (e.g., thermal).

Since the proposed detection is based on functional validation using logic values, it is robust with respect to parameter variations and can reliably detect very small Trojans, e.g., the ones with few logic gates. Thus, the technique can be used as *complementary to the side-channel Trojan detection approaches* [2, 4, 22] which are more effective in detecting large Trojans (e.g., ones with area $>0.1\%$ of the total circuit area).

Although *MERO* proposes a relatively simple heuristic for test generation, it is found to have some shortcomings as discussed in [25]. To overcome these shortcomings, later in this chapter, we propose an improved ATPG scheme to detect small combinational and sequential hardware Trojans, which are otherwise often difficult to detect by side-channel analysis or can bypass design modification-based detection schemes. We note that *for higher effectiveness, a test generation algorithm for Trojan detection must simultaneously consider trigger coverage and Trojan coverage*. Firstly, we introduce a combined *genetic algorithm* (GA)- and Boolean

satisfiability (SAT)-based approach for test pattern generation. GA has been used in the past for fault simulation-based test generation [24]. GA is attractive for getting reasonably good test coverage over the fault list very quickly, because of the inherent parallelism of GA which enables relatively rapid exploration of a search space. However, it does not guarantee the detection of all possible faults, specially for those which are hard to detect. On the other hand, SAT-based test generation has been found to be remarkably useful for hard-to-detect faults. However, it targets the faults one by one and thus incurs higher execution time for easy-to-detect faults which typically represent the majority of faults [11]. It has another interesting feature that it can declare whether a fault is untestable or not.

In case of hardware Trojan, the number of candidate trigger combinations has an exponential dependence on the number of rare nodes considered. Even if we limit the number of Trojan inputs to four (because of VLSI design and side-channel information leakage considerations), the count is quite large. Thus, we have a large candidate trigger list, and it is not possible to handle each fault in that list sequentially. However, many of these trigger conditions are not actually satisfiable and thus cannot constitute a feasible trigger. Hence, we combine the "best of both worlds" for GA- and SAT-based test generation. The rationale is that most of the easy-to-excite trigger conditions, as well as a significant number of hard-to-excite trigger conditions, will be detected by the GA within reasonable execution time. The remaining unresolved trigger patterns are input to the SAT tool; if any of these trigger conditions is feasible, then SAT returns the corresponding test vector. Otherwise, the pattern will be declared unsolvable by the SAT tool itself. As we show later, this combined strategy is found to perform significantly better than *MERO*. In the second phase of the scheme, we refine the test set generated by GA and SAT, by judging its effectiveness from the perspective of potency of the triggered Trojans. For each feasible trigger combination found in the previous step, we find most of the possible payloads using a fault simulator. For this, *we model the effect of each Trojan instance (defined by a combination of a feasible trigger condition and the payload node) as a stuck-at fault and test whether the fault can be propagated to the output by the same test vector which triggered the Trojan. This step helps to find out a compact test set which remarkably improves the Trojan coverage.*

7.2 MERO Approach for Hardware Trojan Detection

As described in Sect. 7.1, the main concept of our test generation approach is based on generating test vectors that can excite candidate trigger nodes individually to their rare logic values multiple (at least N) times. In effect, the probability of activation of a Trojan by the simultaneous occurrence of the rare conditions at its trigger nodes increases. As an example, consider the Trojan shown in Fig. 7.1c. Assume that the conditions $a = 0$, $b = 1$, and $c = 1$ are very rare. Hence, if we can generate a set of test vectors that induce these rare conditions at these nodes individually N times

where N is sufficiently large, then a Trojan with triggering condition composed jointly of these nodes is highly likely to be activated by the application of this test set. The concept can be extended to sequential Trojans, as shown in Fig. 7.1d, where the inserted 3-bit counter is clocked on the simultaneous occurrence of the condition $ab' = 1$. If the test vectors can sensitize these nodes such that the condition $ab' = 1$ is satisfied at least eight times (the maximum number of states of a 3-bit counter), then the Trojan would be activated. Next, we present a mathematical analysis to justify the concept.

7.2.1 Mathematical Analysis

Without loss of generality, assume that a Trojan is triggered by the rare logic values at two nodes A and B, with corresponding probability of occurrence p_1 and p_2. Assume T to be the total number of vectors applied to the circuit under test, such that both A and B have been individually excited to their rare values *at least* N times. Then, the expected number of occurrences of the rare logic values at nodes A and B is given by $E_A = T{\cdot}p_1 \geq N$ and $E_B = T{\cdot}p_2 \geq N$, which lead to:

$$T \geq \frac{N}{p_1} \qquad \text{and} \qquad T \geq \frac{N}{p_2} \tag{7.1}$$

Now, let p_j be the probability of simultaneous occurrence of the rare logic values at nodes A and B, an event that acts as the trigger condition for the Trojan. Then, the expected number of occurrences of this event when T vectors are applied is:

$$E_{AB} = p_j{\cdot}T \tag{7.2}$$

In the context of this problem, we can assume $p_j > 0$, because an adversary is unlikely to insert a Trojan which would never be triggered. Then, to ensure that the Trojan is triggered at least once when T test vectors are applied, the following condition must be satisfied:

$$p_j{\cdot}T \geq 1 \tag{7.3}$$

From inequality (7.1), let us assume $T = c{\cdot}\frac{N}{p_1}$. where $c \geq 1$ is a constant depending on the actual test set applied. Inequality (7.3) can then be generalized as:

$$S = c{\cdot}\frac{p_j}{p_1}{\cdot}N \tag{7.4}$$

where S denotes the number of times the trigger condition is satisfied during the test procedure. From this equation, the following observations can be made about the interdependence of S and N:

1. For given parameters c, p_1, and p_j, S is proportional to N, i.e., the expected number of times the Trojan trigger condition is satisfied increases with the number of times the trigger nodes have been individually excited to their rare values. This observation forms the main motivation behind the *MERO* test generation approach for Trojan detection.
2. If there are q trigger nodes and if they are assumed to be mutually independent, then $p_j = p_1 \cdot p_2 \cdot p_3 \cdots p_q$, which leads to:

$$S = c \cdot N \cdot \prod_{i=2}^{q} p_i \tag{7.5}$$

   As $p_i < 1 \quad \forall i = 1, 2, \cdots q$, hence, with the increase in q, S decreases for a given c and N. In other words, with the increase in the number of trigger nodes, it becomes more difficult to satisfy the trigger condition of the inserted Trojan for a given N. Even if the nodes are not mutually independent, a similar dependence of S on q is expected.
3. The trigger nodes can be chosen such that $p_i \leq \theta \quad \forall i = 1, 2, \cdots q$, so that θ is defined as a *trigger threshold* probability. Then as θ increases, the corresponding selected rare node probabilities are also likely to increase. This will result in an increase in S for a given T and N, i.e., the probability of Trojan activation would increase if the individual nodes are more likely to get triggered to their rare values.

All of the above predicted trends were observed in our simulations, as shown in Sect. 7.2.7.

7.2.2 Test Generation

Algorithm 7.1 shows the major steps in the proposed reduced test set generation process for Trojan detection. We start with the golden circuit netlist (without any Trojan), a random pattern set (V), list of rare nodes (L), and number of times to activate each node to its rare value (N). First, the circuit netlist is read and mapped to a *hypergraph*. For each node in L, we initialize the number of times a node encounters a rare value (A_R) to 0. Next, for each random pattern v_i in V, we count the number of nodes (C_R) in L whose rare value is satisfied. We sort the random patterns in decreasing order of C_R. In the next step, we consider each vector in the sorted list and modify it by perturbing one bit at a time. If a modified test pattern increases the number of nodes satisfying their rare values, we accept the pattern in the reduced pattern list. In this step we consider only those rare nodes with $A_R < N$. The process repeats until each node in L satisfies its rare value at least N times. The output of the test generation process is a minimal test set that improves the coverage for both combinational and sequential Trojans compared to random patterns.

Algorithm 7.1 Procedure *MERO*

Generate reduced test pattern set for Trojan detection

Inputs: Circuit netlist, list of rare nodes (L) with associated rare values, list of random patterns (V), number of times a rare condition should be satisfied (N)
Outputs: Reduced pattern set (R_V)

1: Read circuit and generate *hypergraph*
2: **for all** nodes in L **do**
3: **set** number of times node satisfies rare value (A_R) to 0
4: **end for**
5: **set** $R_V = \Phi$
6: **for all** random pattern in V **do**
7: Propagate values
8: Count the # of nodes (C_R) in L with their rare value satisfied
9: **end for**
10: Sort vectors in V in decreasing order of C_R
11: **for all** vector v_i in decreasing order of C_R **do**
12: **for all** bit in v_i **do**
13: Perturb the bit and recompute # of *satisfied rare values* ($C_R^{'}$)
14: **if** ($C_R^{'} > C_R$) **then**
15: Accept the perturbation and form $v_i^{'}$ from v_i
16: **end if**
17: **end for**
18: Update A_R for all nodes in L due to vector v_i
19: **if** $v_i^{'}$ increases A_R for at least one rare node **then**
20: Add the modified vector $v_i^{'}$ to R_V
21: **end if**
22: **if** ($A_R \geq N$) for all nodes in L **then**
23: break
24: **end if**
25: **end for**

7.2.3 Coverage Estimation

Once the reduced test vector set has been obtained, computation of Trigger and Trojan coverage can be performed for a given *trigger threshold* (θ) (as defined in Sect. 7.2.1) and a given number of trigger nodes (q) using a random sampling approach. From the Trojan population, we randomly select a number of q-trigger Trojans, where each trigger node has signal probability less than equal θ. We assume that Trojans comprising of trigger nodes with higher signal probability than θ will be detected by conventional test. From the set of sampled Trojans, Trojans with false trigger conditions which cannot be justified with any input pattern are eliminated. Then, the circuit is simulated for each vector in the given vector set and checked whether the trigger condition is satisfied. For an activated Trojan, if its effect can be observed at the primary output or scan flip-flop input, the Trojan is considered "covered," i.e., detected. The percentages of Trojans activated and detected constitute the *trigger coverage* and *Trojan coverage*, respectively.

7.2.4 Choice of Trojan Sample Size

In any random sampling process, an important decision is to select the sample size in a manner that represents the population reasonably well. In the context of Trojan detection, it means further increase in sampled Trojans renders negligible change in the estimated coverage. Figure 7.2 shows a plot of percentage deviation of trigger and Trojan coverage ($q = 4$) from the asymptotic value for two benchmark circuits with varying Trojan sample size. From the plots, we observe that the coverage saturates with nearly 100,000 samples, as the percentage deviation tends to zero. To compromise between accuracy of estimated coverage and simulation time, we have selected a sample size of 100,000 in our simulations.

7.2.5 Choice of N

Figure 7.3 shows the trigger and Trojan coverage for two ISCAS-85 benchmark circuits with increasing values of N, along with the lengths of the corresponding test set. From these plots, it is clear that similar to *N-detect* tests for stuck-at fault where defect coverage typically improves with increasing N, the trigger and Trojan coverage obtained with the *MERO* approach also improves steadily with N, but then both saturate around $N = 200$ and remain nearly constant for larger values of N. As expected, the test size also increases with increasing N. We chose a value of $N = 1000$ for most of our experiments to reach a balance between coverage and test vector set size.

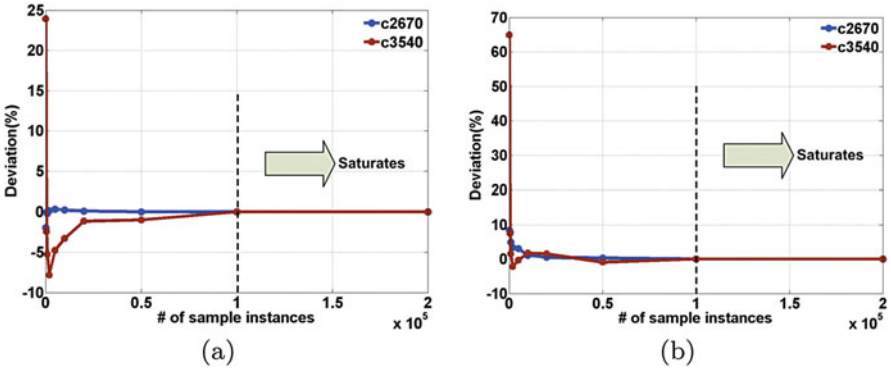


Fig. 7.2 Impact of sample size on trigger and Trojan coverage for benchmarks c2670 and c3540, $N = 1000$ and $q = 4$: (**a**) deviation of trigger coverage, and (**b**) deviation of Trojan coverage

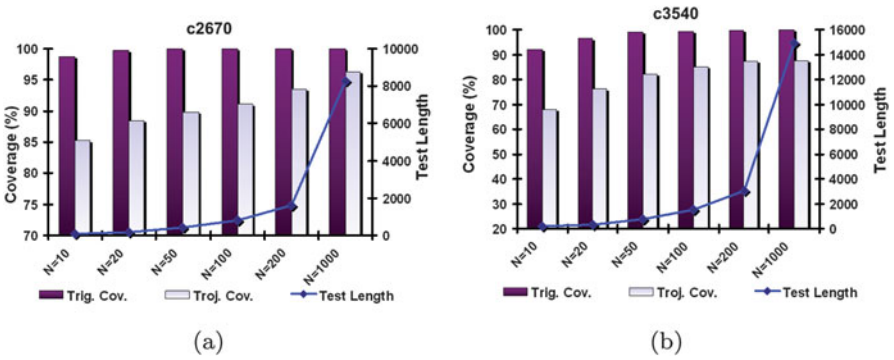


Fig. 7.3 Impact of N (number of times a rare point satisfies its rare value) on the trigger/Trojan coverage and test length for benchmarks (**a**) c2670 and (**b**) c3540

7.2.6 Improving Trojan Detection Coverage

As noted in previous sections, Trojan detection using logic testing involves simultaneous triggering of the Trojan and the propagation of its effect to output nodes. Although the proposed test generation algorithm increases the probability of Trojan activation, it does not explicitly target increasing the probability of a malicious effect at payload being observable. *MERO* test patterns, however, achieve significant improvement in Trojan coverage compared to random patterns, as shown in Sect. 7.2.7. This is because the Trojan coverage has strong correlation with trigger coverage. To increase the Trojan coverage further, one can use the following low-overhead approaches.

1. *Improvement of test quality*: We can consider number of nodes observed along with number of nodes triggered for each vector during test generation. This means, at steps 13–14 of algorithm 7.1, a perturbation is accepted if the sum of triggered and observed nodes improves over previous value. This comes at extra computational cost to determine the number of observable nodes for each vector. We note that for a small ISCAS benchmark *c432* (an interrupt controller), we can improve the Trojan coverage by 6.5% with negligible reduction in trigger coverage using this approach.
2. *Observable test point insertion*: We note that insertion of very few observable test points can achieve significant improvement in Trojan coverage at the cost of small design overhead. Existing algorithm for selecting observable test points

for stuck-at fault test [13] can be used here. Our simulation with *c432* resulted in about 4% improvement in Trojan coverage with five judiciously inserted observable points.

3. *Increasing N and/or increasing the controllability of the internal nodes*: Internal node controllability can be increased by judiciously inserting few controllable test points or increasing *N*. It is well-known in the context of stuck-at ATPG, that scan insertion improves both controllability and observability of internal nodes. Hence, the proposed approach can take advantage of low-overhead design modifications to increase the effectiveness of Trojan detection.

7.2.7 Results

7.2.7.1 Simulation setup

We have implemented the test generation and the Trojan coverage determination in three separate C programs. All the three programs can read a Verilog netlist and create a *hypergraph* from the netlist description. The first program, named as *RO-Finder* (**R**are **O**ccurence **Finder**), is capable of functionally simulating a netlist for a given set of input patterns, computing the signal probability at each node and identifying nodes with low signal probability as rare nodes. The second program *MERO* implements algorithm-7.1 described in Sect. 7.2.2 to generate the reduced pattern set for Trojan detection. The third program, *TrojanSim* (**Trojan Sim**ulator), is capable of determining both trigger and Trojan coverage for a given test set using random sample of Trojan instances. A *q*-trigger random Trojan instance is created by randomly selecting the trigger nodes from the list of rare nodes. We consider one randomly selected payload node for each Trojan. Figure 7.4 shows the flowchart for the *MERO* methodology. Synopsys *TetraMAX* was used to justify the trigger condition for each Trojan and eliminate the false Trojans. All simulations and test generation were carried out on a Hewlett-Packard Linux workstation with a 2 GHz dual-core Intel processor and 2 GB RAM.

7.2.7.2 Comparison with Random and ATPG Patterns

Table 7.1 lists the trigger and Trojan coverage results for a set of combinational (ISCAS-85) and sequential (ISCAS-89) benchmarks using stuck-at ATPG patterns (generated using the algorithm in [19]), weighted random patterns, and *MERO* test patterns. It also lists the number of total nodes in the circuit and the number of rare nodes identified by *RO-Finder* tool based on signal probability. The signal probabilities were estimated through simulations with a set of 100,000 random vectors. For the sequential circuits, we assume full-scan implementation. We consider 100,000

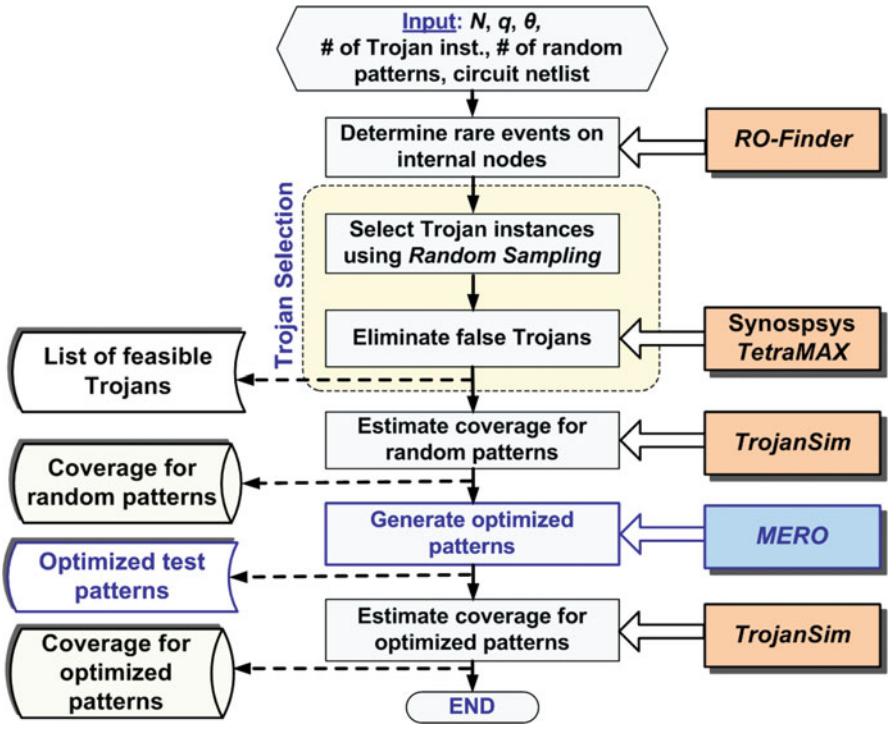


Fig. 7.4 Integrated framework for rare occurrence determination, test generation using *MERO* approach, and Trojan simulation

random instances of Trojans following the sampling policy described in Sect. 7.2.4, with one randomly selected payload node for each Trojan. Coverage results are provided in each case for two different trigger point counts, $q = 2$ and $q = 4$, at $N = 1000$ and $\theta = 0.2$.

Table 7.2 compares reduction in the length of the testset generated by the *MERO* test generation method with 100,000 random patterns, along with the corresponding run-times for the test generation algorithm. This run-time includes the execution time for *Tetramax* to validate 100,000 random Trojan instances, as well as time to determine the coverage by logic simulation. We can make the following important observations from these two tables:

1. The stuck-at ATPG patterns provide poor trigger and Trojan coverage compared to *MERO* patterns. The increase in coverage between the ATPG and *MERO* patterns is more significant in case of higher number of trigger points.
2. From Table 7.2, it is evident that the reduced pattern with $N = 1000$ and $\theta = 0.2$ provides comparable trigger coverage with significant reduction in test length. The average improvement in test length for the circuits considered is about 85%.

Table 7.1 Comparison of Trigger and Trojan coverage among ATPG patterns [19], Random (100 K, input weights: 0.5), and *MERO* patterns for $q = 2$ and $q = 4$, $N = 1000$, $\theta = 0.2$

		ATPG patterns				Random (100 K patterns)				*MERO* patterns			
		$q = 2$		$q = 4$		$q = 2$		$q = 4$		$q = 2$		$q = 4$	
Ckt.	Nodes (rare/tot.)	Trig. Cov. (%)	Troj. Cov. (%)	Trig. Cov. (%)	Troj. Cov. (%)	Trig. Cov. (%)	Troj. Cov. (%)	Trig. Cov. (%)	Troj. Cov. (%)	Trig. Cov. (%)	Troj. Cov. (%)	Trig. Cov. (%)	Troj. Cov. (%)
c2670	297/1010	93.97	58.38	30.7	10.48	98.66	53.81	92.56	30.32	100.00	96.33	99.90	90.17
c3540	580/1184	77.87	52.09	16.07	8.78	99.61	86.5	90.46	69.48	99.81	86.14	87.34	64.88
c5315	817/2485	92.06	63.42	19.82	8.75	99.97	93.58	98.08	79.24	99.99	93.83	99.06	78.83
c6288	199/2448	55.16	50.32	3.28	2.92	100.00	98.95	99.91	97.81	100.00	98.94	92.50	89.88
c7552	1101/3720	82.92	66.59	20.14	11.72	98.25	94.69	91.83	83.45	99.38	96.01	95.01	84.47
s13207[a]	865/2504	82.41	73.84	27.78	27.78	100	95.37	88.89	83.33	100.00	94.68	94.44	88.89
s15850[a]	959/3004	25.06	20.46	3.80	2.53	94.20	88.75	48.10	37.98	95.91	92.41	79.75	68.35
s35932[a]	970/6500	87.06	79.99	35.9	33.97	100.00	93.56	100.00	96.80	100.00	93.56	100.00	96.80
Avg.	**724/2857**	**74.56**	**58.14**	**19.69**	**13.37**	**98.84**	**88.15**	**88.73**	**72.30**	**99.39**	**93.99**	**93.50**	**82.78**

[a]These sequential benchmarks were run with 10,000 random Trojan instances to reduce run time of *Tetramax*

Table 7.2 Reduction in test length with MERO approach compared to 100 K random patterns along with runtime, $q = 2, N = 1000, \theta = 0.2$

Ckt.	*MERO* test length	% Reduction	Run-time (s)
c2670	8254	**91.75**	30,051.53
c3540	14,947	**85.05**	9403.11
c5315	10,276	**89.72**	80,241.52
c6288	5014	**94.99**	15,716.42
c7552	12,603	**87.40**	160,783.37
s13207[a]	26,926	**73.07**	23,432.04
s15850[a]	32,775	**67.23**	39,689.63
s35932[a]	5480	**94.52**	29,810.49
Avg.	**14,534**	**85.47**	**48,641.01**

[a]These sequential benchmarks were run with 10,000 random Trojan instances to reduce run time of *Tetramax*

3. Trojan coverage is consistently smaller compared to trigger coverage. This is because in order to detect a Trojan by applying an input pattern, besides satisfying the trigger condition, one needs to propagate the logic error at the payload node to one or more primary outputs. In many cases although the trigger condition is satisfied, the malicious effect does not propagate to outputs. Hence, the Trojan remains triggered but undetected.

7.2.7.3 Effect of Number of Trigger Points (q)

The impact of q on coverage is evident from the Fig. 7.5, which shows the decreasing trigger and Trojan coverage with the increasing number of trigger points for two combinational benchmark circuits. This trend is expected from the analysis of Sect. 7.2.1. Our use of *TetraMAX* for justification and elimination of the false triggers helped to improve the Trojan coverage.

7.2.7.4 Effect of Trigger Threshold (θ)

Figure 7.6 plots the trigger and Trojan coverage with increasing θ for two ISCAS-85 benchmarks, at $N = 1000$ and $q = 4$. As we can observe, the coverage values improve steadily with increasing θ while saturating at a value above 0.20 in both the cases. The improvement in coverage with θ is again consistent with the conclusions from the analysis of Sect. 7.2.1.

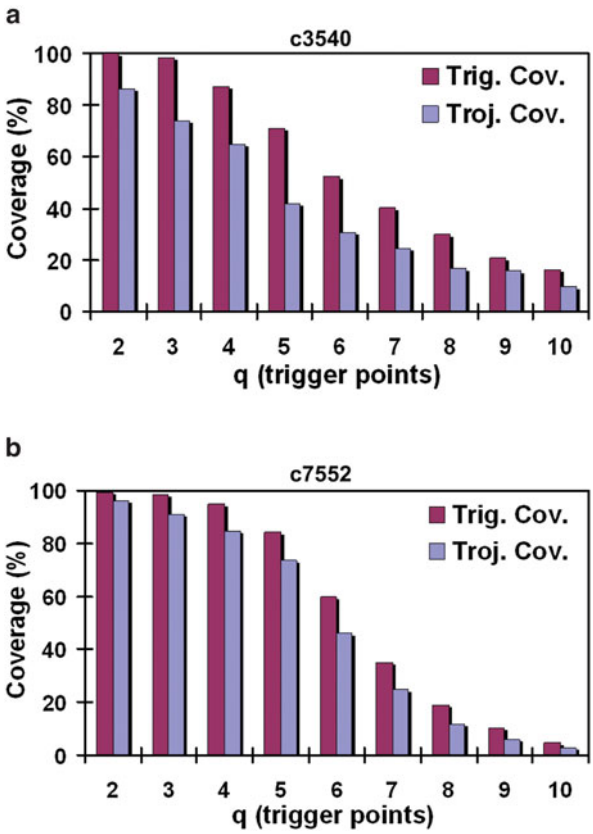


Fig. 7.5 Trigger and Trojan coverage with varying number of trigger points (q) for benchmarks (**a**) c3540 and (**b**) c7552, at $N = 1000$, $\theta = 0.2$

7.2.7.5 Sequential Trojan Detection

To investigate the effectiveness of the *MERO* test generation methodology in detecting sequential Trojans, we designed and inserted sequential Trojans modeled following Fig. 7.1d, with 0, 2, 4, 8, 16, and 32 states, respectively (the case with zero states refers to a combinational Trojan following the model of Fig. 7.1c). A cycle-accurate simulation was performed by our simulator *TrojanSim*, and the Trojan was considered detectable only when the output of the golden circuit and the infected circuit did not match. Table 7.3 presents the trigger and Trojan coverage, respectively, obtained by 100,000 randomly generated test vectors and the *MERO* approach for three large ISCAS-89 benchmark circuits. The superiority of the *MERO* approach over the random test vector generation approach in detecting sequential Trojans is evident from this table.

Although these results have been presented for a specific type of sequential Trojans (counters which increase their count conditionally), they are representative of other sequential Trojans whose state transition graph (STG) has no "loop." The STG for such a FSM has been shown in Fig. 7.7. This is an eight-state FSM which

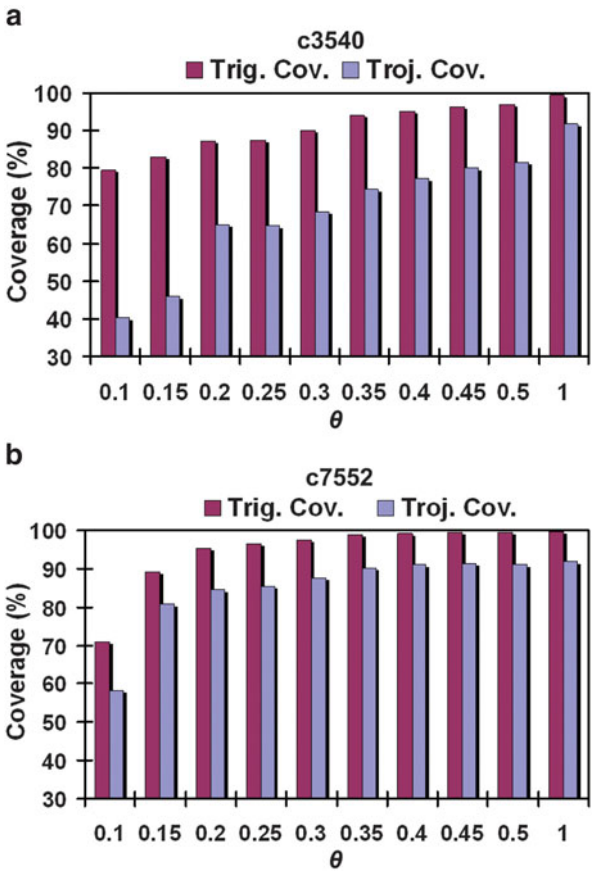


Fig. 7.6 Trigger and Trojan coverage with *trigger threshold* (θ) for benchmarks (**a**) c3540 and (**b**) c7552, for $N = 1000$, $q = 4$

changes its state only when a particular internal node condition C_i is satisfied at state S_i, and the Trojan is triggered when the FSM reaches state S_8. The example Trojan shown in Fig. 7.1d is a special case of this model, where the conditions C_1 through C_8 are identical. If each of the conditions C_i is as rare as the condition $a = 1, b = 0$ required by the Trojan shown in Fig. 7.1d, then there is no difference between these two Trojans as far as their rareness of getting triggered is concerned. Hence, we can expect similar coverage and test length results for other sequential Trojans of this type. However, the coverage may change if the FSM structure is changed (as shown with dotted line). In this case, the coverage can be controlled by changing N.

7.2.7.6 Application to Side-Channel Analysis

As observed from the results presented in this section, the *MERO* approach can achieve high trigger coverage for both combinational and sequential Trojans. This essentially means that the *MERO* patterns will induce activity in the Trojan

Table 7.3 Comparison of sequential Trojan coverage between random (100 K) and MERO patterns, $N = 1000$, $\theta = 0.2$, $q = 2$

	Trigger Cov. for 100 K random vectors (%)						Trigger Cov. for *MERO* vectors (%)					
	Trojan state count						Trojan state count					
Ckt.	0	2	4	8	16	32	0	2	4	8	16	32
s13207	100.00	100.00	99.77	99.31	99.07	98.38	100.00	100.00	99.54	99.54	98.84	97.92
s15850	94.20	91.99	86.79	76.64	61.13	48.59	95.91	95.31	94.03	91.90	87.72	79.80
s35932	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Avg.	**98.07**	**97.33**	**95.52**	**91.98**	**86.73**	**82.32**	**98.64**	**98.44**	**97.86**	**97.15**	**95.52**	**92.57**

	Trojan Cov. for 100 K random vectors (%)							Trojan Cov. for *MERO* vectors (%)					
	Trojan state count							Trojan state count					
	0	2	4	8	16	32	Ckt.	0	2	4	8	16	32
s13207	95.37	95.37	95.14	94.91	94.68	93.98		94.68	94.68	94.21	94.21	93.52	92.82
s15850	88.75	86.53	81.67	72.89	58.4	46.97		92.41	91.99	90.62	88.75	84.23	76.73
s35932	93.56	93.56	93.56	93.56	93.56	93.56		93.56	93.56	93.56	93.56	93.56	93.56
Avg.	**92.56**	**91.82**	**90.12**	**87.12**	**82.21**	**78.17**		**93.55**	**93.41**	**92.80**	**92.17**	**90.44**	**87.70**

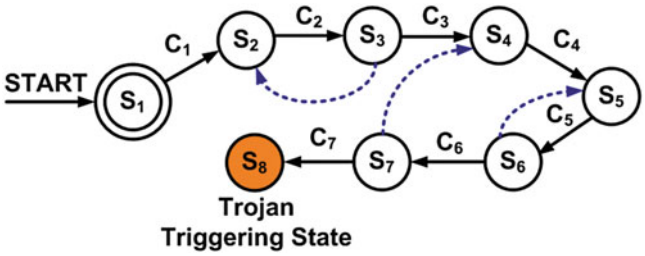


Fig. 7.7 FSM model with no loop in state transition graph

triggering circuitry with high probability. A minimal set of patterns that is highly likely to cause activity in a Trojan is attractive in power- or current signature-based side-channel approach to detect hardware Trojan [2, 4, 22]. The detection sensitivity in these approaches depends on the induced activity in the Trojan circuit by applied test vector. It is particularly important to enhance sensitivity for the Trojans where the leakage contribution to power by the Trojan circuit can be easily masked by process or measurement noise. Hence, *MERO* approach can be extended to generate test vectors for side-channel analysis, which requires amplifying the Trojan impact on side-channel parameter such as power or current. More detailed work on test vector generation using side-channel analysis can be found in [16] which focuses on statistical test generation for increasing the sensitivity of side-channel-based hardware Trojan detection.

7.2.8 Shortcomings of MERO

Although *MERO* proposes a relatively simple heuristic for test generation, it is found to have the following shortcomings:

1. When tested on a set of "hard-to-trigger" Trojans (with *triggering probability* in the range of 10^{-6} or less), the test vector set generated by *MERO* was found to have poor coverage both over triggering combinations and Trojans. Figure 7.8b presents the variation of trigger and Trojan coverage with the *rareness threshold* (θ) value for the ISCAS–85 circuit c7552, where the Trojan trigger probability is the *effective* Trojan triggering probability considering all the nodes together, unlike in [8], which considered trigger probability at individual nodes (see Fig. 7.8a). It was found that best coverage was achieved for θ in the range 0.08–0.12, and this trend was consistent for all the benchmark circuits considered. However, the best achievable coverage was still below 50%, for even circuits of moderate size like c7552.
2. Although individual activation of each individual rare nodes at least N-times increases the activation probability of rare node combinations on average, there is always a finite probability that combinations with extremely low activation

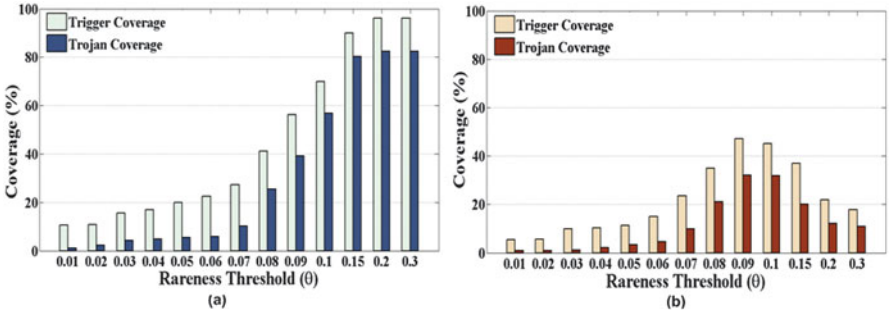


Fig. 7.8 Motivational example: variation of trigger and Trojan coverage with rareness threshold (θ) by the *MERO* [8] technique for c7552 (**a**) on sets of Trojans considered as in [8]; (**b**) on a set of rarely triggered Trojans (effective triggering probability below 10^{-6})

probability will not be triggered for a given value of N. As a result, even for small ISCAS–85 circuits like c432, the *MERO* test generation method misses some rare node patterns even after several independent runs. This fact can be utilized by an intelligent adversary.

3. *MERO* explores a relatively small numbers of test vectors, as the heuristic perturbs only a single bit at a time of an obtained test vector to generate new test vectors.
4. Another problem with the *MERO* algorithm is that, **while generating the test vectors, it only considers the activation of the triggering conditions and ignores whether the triggered Trojan actually caused any logic malfunction at the primary output of the circuit under test**.

In order to rectify the abovementioned shortcomings, we developed an improved test pattern generation for hardware Trojan detection using genetic algorithm and Boolean satisfiability which is discussed in the following section.

7.3 GA- and SAT-Based Approach for Hardware Trojan Detection

7.3.1 Hardware Trojan Models

We consider simple combinational and sequential hardware Trojans, where a hardware Trojan instance is triggered by the simultaneous occurrence of rare logic values at one or more internal nodes of the circuit. We find the rare nodes ($\mathcal{R}$) of the circuit with a probabilistic analysis. Details of this analysis can be found in [27, 32]. Once activated, the Trojan flips the logic value at an internal *payload* node. Figure 7.9 shows the type of Trojans considered by us.

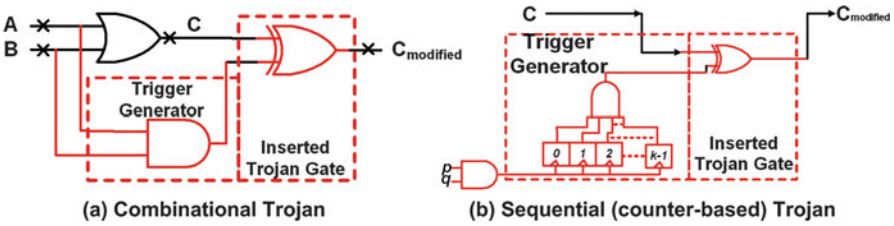


Fig. 7.9 Example of (**a**) combinational and (**b**) sequential (counter-like) Trojan. The combinational Trojan is triggered by the simultaneous occurrence of logic–1 at two internal nodes. The sequential (counter-like) Trojan is triggered by 2^k positive ($0 \rightarrow 1$) transitions at the input of the flip-flops

Notice that it is usually infeasible to enumerate all hardware Trojans in a given circuit. Hence, we are forced to restrict ourselves to analysis results obtained from a randomly selected subset of Trojans. The cardinality of the set of Trojans selected depends on the size of the circuit being analyzed. Since we are interested only in small Trojans, a random sample $\mathcal{S}$ of up to four rare node combinations is considered. Let us denote the set of rare nodes as $\mathcal{R}$, with $|\mathcal{R}| = r$ for a specific rareness threshold (θ). The set of all possible rare node combinations is then the *power set* of $\mathcal{R}$, denoted by $2^{\mathcal{R}}$). Thus, the population of Trojans under consideration is the set $\mathcal{K}$, where $\mathcal{K} \subseteq 2^{\mathcal{R}}$ and $|\mathcal{K}|$ is $\binom{r}{1} + \binom{r}{2} + \binom{r}{3} + \binom{r}{4}$. Thus, $\mathcal{S} \subseteq \mathcal{K}$.

We intentionally chose $\theta = 0.1$ for our experiments, which is lower than the value considered in [8] for MERO($\theta = 0.2$). The choice is based on the observed coverage trends of our experiments on "hard-to-trigger" Trojans in Sect. 7.1, where it was observed that the coverage is maximized for θ values in the range 0.08–0.12 for most ISCAS benchmark circuits.

7.3.2 *Genetic Algorithm (GA) for ATPG*

Genetic algorithm (GA) is a well-known bio-inspired, stochastic, evolutionary search algorithm based upon the principles of natural selection [14]. GA has been widely used in diverse fields to tackle difficult non-convex optimization problems, both in discrete and continuous domains. In GA, the quality of a feasible solution is improved iteratively, based on computations that mimic basic genetic operations in the biological world. The quality of the solution is estimated by evaluating the numerical value of an objective function, usually termed the "fitness function" in GA. In the domain of VLSI testing, it has been successfully used for difficult test generation and diagnosis problems [24]. In the proposed scheme, GA has been used as a tool to automatically generate quality test patterns for Trojan triggering. During test generation using GA, two points were emphasized:

– an effort to generate test vectors that would activate the most number of sampled trigger combinations
– an effort to generate test vectors for hard-to-trigger combinations

However, as mentioned in the previous section, the major effort for GA was dedicated to meet the first objective.

To meet both of the goals, we used a special data structure as well as a proper fitness function. The data structure is a hash table which contains the triggering combinations and their corresponding activating test vectors. Let $\mathcal{S}$ denote the sampled set of trigger conditions being considered. Each entry in the hash table ($\mathcal{D}$) is a tuple (s, $\{t_i\}$), where $s \in \mathcal{S}$ is a trigger combination from the sampled set ($\mathcal{S}$) and $\{t_i\}$ is the set of distinct test vectors activating the trigger combination s. Note that, a single test vector t_i may trigger multiple trigger combinations and thus can be present multiple times in the data structure for different trigger combinations (s). The data structure is keyed with trigger combination s. Initially, $\mathcal{D}$ is empty; during the GA run, $\mathcal{D}$ is updated dynamically, whenever new triggering combinations from $\mathcal{S}$ are found to be satisfied. The fitness function is expressed as:

$$f(t) = R_{\text{count}}(t) + w * I(t) \tag{7.6}$$

where $f(t)$ is the fitness value of a test vector t, $R_{\text{count}}(t)$ counts the number of rare nodes triggered by the test vector t, w (> 1) is a constant scaling factor, and $I(t)$ is a function which returns the *relative improvement* of the database $\mathcal{D}$ due to the test vector t. The term "*relative improvement of the database*" ($I(t)$) can be explained as follows. Let us interpret the data structure $\mathcal{D}$ as a histogram where the bins are defined by unique trigger combinations $s \in \mathcal{S}$, and each bin contains its corresponding activating test vectors $\{t_i\}$. Before each update of the database, we calculate the number of test patterns in each bin which is to be updated. The *relative improvement* is defined as:

$$I(t) = \frac{n_2(s) - n_1(s)}{n_2(s)} \tag{7.7}$$

where $n_1(s)$ is the number of test patterns in bin s before update and $n_2(s)$ is the number of test patterns in bin s after update.

Note that for each test pattern t that enters the database D, the numerator will be either 0 or 1 for an arbitrary trigger combination s. However, the denominator will have larger values with the minimum value being 1. Thus, for any bin s, when it gets its 1st test vector, the abovementioned ratio achieves the maximum value 1, whereas when a bin gets its nth test vector, the fraction is $\frac{1}{n}$. The value gradually decreases as the number of test vectors in s increases. This implies that **as a newer test vector is generated, its contribution is considered more important if it has been able to trigger a yet unactivated trigger condition s than if it activates a trigger condition that has already been activated by other test vector(s)**. Note that, for bins s having zero test vectors before and after update (i.e., trigger

conditions which could not be activated at the first try), we assign a very small value 10^{-7} for numerical consistency. The scaling factor w is proportional to the relative importance of the relative improvement term; in our implementation, w was set to have the value 10.

The rationale behind the two terms in the fitness function is as follows. The first term in the fitness function prefers test patterns that simultaneously activate as many trigger nodes as possible, thus recording test vectors each of which can potentially cover many trigger combinations. The inclusion of the second term has two effects. Firstly, the selection pressure of GA is set toward hard-to-activate patterns by giving higher fitness value to those test patterns that are capable of hard-to-trigger conditions. Secondly, it also helps the GA to explore the sampled trigger combination space evenly. To illustrate this, let us consider the following example.

Suppose, we have five rare nodes r_1, r_2, r_3, r_4, r_5. We represent the activation of these five nodes by a binary vector **r** of length five, where $r_i = 1$ denotes that the i^{th} rare node has been activated to its rare value and $r_i = 0$ otherwise. Thus, a pattern 11110 implies the scenario where the first four rare nodes have been simultaneously activated. Now, the test vector t, which generates this rare node-triggering pattern, also triggers the patterns 10000, 01000, 00100, 00010, 11000, ..., 11100, i.e., any subset of the triggered rare nodes. Mathematically, **if there are in total r rare nodes and r' rare nodes are simultaneously triggered in a pattern ($r' < r$) by a test vector t, then the $2^{r'}$ subsets of the triggered rare nodes are also triggered by the same test vector. Hence, maximizing r' increases the coverage over the trigger combination sample set**.

The test generation problem is modeled as a maximization problem and solved using a variation of GA termed *binary genetic algorithm* (BGA) [14]. Each individual in the population is a bit pattern called a "chromosome," which represents an individual test vector. Two operations generate new individuals by operating on these chromosomes: *crossover* and *mutation*. *Crossover* refers to the exchange of parts of two chromosomes to generate new chromosomes, while *mutation* refers to the random (probabilistic) flipping of bits of the chromosomes to give rise to new behavior. Figure 7.10 shows examples of a two-point crossover and mutation

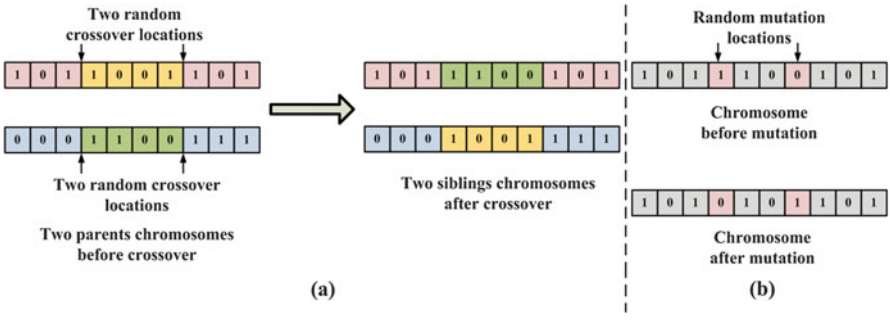


Fig. 7.10 Example of two–point crossover and mutation in Binary Genetic Algorithm

in BGA. We used a two-point crossover and binary mutation, with a crossover probability of 0.9 and mutation probability of 0.05, respectively. The collection of individuals at every iteration is termed a *population*. A population size of 200 was used for combinatorial circuits and 500 for sequential circuits. Two terminating conditions were used: (i) when the total number of distinct test vectors in the database crosses a certain threshold value $\#T$ or (ii) if 1000 generations had been reached. The initialization of the population is done by test vectors satisfying some rare node combinations from the sample set. These rare node combinations are randomly selected, and the test vectors were found using SAT tools (details are given in the following subsection). Algorithm-7.2 shows the complete test generation scheme using GA. Notice that the initial test vector population is generated by solving a small number of triggering conditions using SAT.

Among the sampled trigger instances, many might not be satisfiable, as we do not have any prior information about them. Moreover, although GA traverses the given trigger combination sample space reasonably rapidly, it cannot guarantee to be able to generate test vectors that would activate all the hard-to-trigger patterns. Thus, even after the GA test generation step, we were left with some trigger combinations among which some are not satisfiable and others are extremely hard to detect. However, as the number of such remaining combinations is quite less (typically 5–10% of the selected samples), we can apply SAT tools to solve them. We next describe the application of SAT in our ATPG scheme.

7.3.3 SAT for Hard-to-Activate Trigger Conditions

Boolean satisfiability (SAT) tools are being used to solve ATPG problems since the last decade [11]. They are found to be robust, often succeeding to find test patterns in large and pathological ATPG problems, where traditional ATPG algorithms have been found wanting. Unlike classical ATPG algorithms, SAT solver-based schemes do not work on the circuit representation (e.g., netlist of logic gates) directly. Instead, they formulate the test pattern generation problem as one or more SAT problems. A n-variable Boolean formula $f(x_1, x_2, \ldots, x_n)$ in *conjunctive normal form* (CNF) is said to be *satisfiable* if there exists a value assignment for the n variables, such that $f = 1$. If no such assignment exists, f is said to be *unsatisfiable*. Boolean satisfiability is an NP-complete problem. Sophisticated heuristics are used to solve SAT problems, and powerful SAT solver software tools have become available in recent times (many of them are free). The ATPG problem instance is first converted into a CNF and then input to a SAT solver. If the solver returns a satisfiable assignment within a specified time, the problem instance is considered to be satisfiable and unsatisfiable, otherwise.

As mentioned previously, we apply the SAT tool only for those trigger combinations for which GA fails to generate any test vector. Let us denote the set of such trigger combinations as $\mathcal{S}' \subseteq \mathcal{S}$. We consider each trigger combination $s \in \mathcal{S}'$ and input it to the SAT tool. This SAT problem formulation is illustrated by an example

Algorithm 7.2 ***TESTGEN_GA***

/* Generate Triggering Test Vectors Using Genetic Algorithm */

Input: Circuit Netlist, Set of rare nodes ($\mathcal{R}$), Set of sampled trigger combinations ($\mathcal{S}$), G_{max}, T_{max}, crossover probability, mutation probability, (empty) trigger database ($\mathcal{D}$)

Output: Data structure ($\mathcal{D}$) filled with triggering test vectors, set of unsatisfied trigger combinations ($\mathcal{S}' \subset \mathcal{S}$)

1: Fill $\mathcal{D}$ with tuples (s, ϕ), $\forall s \in \mathcal{S}$
2: Select a random subset $\mathcal{S}_{init} \subset \mathcal{S}$, such that $|\mathcal{S}_{init}| = k * |\mathcal{S}|$
3: /* k is 0.025 for combinatorial and 0.055 for sequential circuits */
4: Solve all trigger combinations $s \in \mathcal{S}_{init}$ using SAT tool and generate corresponding set of test vectors (T_{init})
5: Update $\mathcal{D}$ with tuples (s, t). where $s \in \mathcal{S}_{init}$ and $t \in T_{init}$
6: **set** $vectcount \leftarrow |T_{init}|$
7: **set** $gencount \leftarrow 0$
8: **set** $\mathcal{S}' \leftarrow \phi$
9: Initialize the population of GA (P) with T_{init}.
10: **repeat**
11: **for all** $t \in P$ **do**
12: Simulate the circuit with test vector t and find the corresponding rare node activation pattern (**r**).
13: Search $\mathcal{D}$ for all triggering patterns covered by **r**.
14: Compute the fitness using Eq. 7.6
15: Update $\mathcal{D}$ with all tuples (s, t), where s is a triggering pattern covered by **r**.
16: **set** $vectcount \leftarrow vectcount + 1$
17: **end for**
18: Perform Crossover on P
19: Perform Mutation on P
20: Update P with the best individuals
21: **set** $gencount \leftarrow gencount + 1$
22: **until** $(gencount \leq G_{max} \,||\, vectcount \leq T_{max})$
23: **for all** $(s, \{t_i\}) \in \mathcal{D}$ **do**
24: **if** $(\{t_i\} = \phi)$ **then**
25: Include s in $\mathcal{S}'$
26: **end if**
27: **end for**

in Fig. 7.11. Let us consider the three rare nodes shown in Fig. 7.11a with their rare values. To create a satisfiability formula which simultaneously activates these three nodes to their rare value, we construct the circuit shown in Fig. 7.11b. The SAT instance is thus formed which tries to achieve a value 1 at wire (node) d.

After completion of this step, most trigger combinations in the set $\mathcal{S}'$ will be found to be satisfiable by the SAT tool, which will also return the corresponding test vectors. However, some of the trigger combinations will still remain unsolved, which would be lebelled as unsatisfiable. Thus the set $\mathcal{S}'$ is partitioned into two disjoint subsets $\mathcal{S}_{sat}$ and $\mathcal{S}_{unsat}$. The first subset is accepted, and the data structure $\mathcal{D}$ is updated with the patterns in this subset, whereas the second subset is discarded. This part of the flow has been summarized in algorithm-7.3.

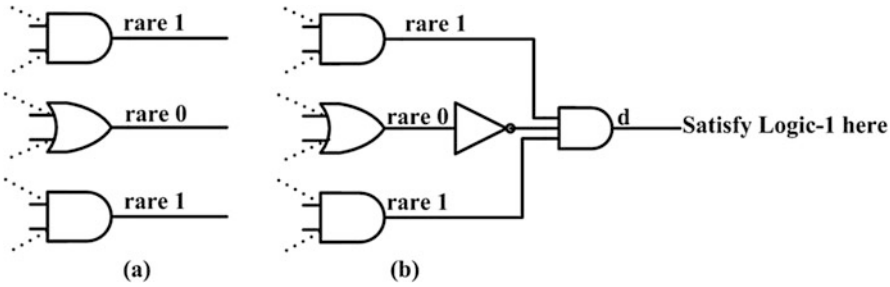


Fig. 7.11 Illustration: formulation of a SAT instance which activates three rare nodes simultaneously

Algorithm 7.3 ***TESTGEN_SAT***

/* Solve the triggering patterns which remain unsolved by GA ($\mathcal{S}'$) using SAT tool */

Input: Set of triggering patterns unsolved by GA ($\mathcal{S}'$), Data structure $\mathcal{D}$
Output: Updated $\mathcal{D}$ with triggering patterns generated by SAT tool

```
for all s ∈ S' do
    Input the triggering combination to SAT tool.
    if (SAT(s) = SOLVED) then
        Retrieve corresponding test vector t.
        Update D with tuple (s, t)
        S_sat ← {s}
    else
        S_unsat ← {s}
    end if
end for
```

As the basic ATPG mechanism now in place, we next describe the refinement of the scheme to take the impact of the payload into consideration and also achieve test compaction in the process.

7.3.4 Payload Aware Test Set Selection and Test Compaction

7.3.4.1 Payload Aware Test Vector Selection

Finding out proper trigger-payload pairs to enumerate feasible hardware Trojan instances a is nontrivial computational problem. In combinational circuits, one necessary condition for a node to be a payload is that its topological rank must be higher than the topologically highest node of the trigger combination, otherwise there is a possibility of forming a "combinational loop"; however, this is not a sufficient condition. In general, a successful Trojan triggering event provides no guarantee regarding its propagation to the primary output to cause functional failure of the circuit. As an example, let us consider the circuit of Fig. 7.12a. The Trojan

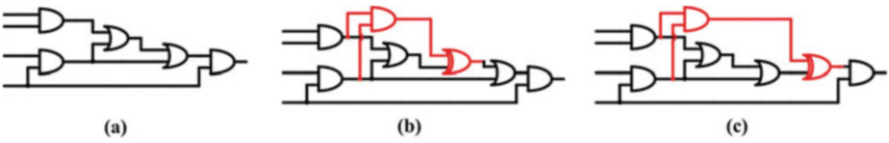


Fig. 7.12 Impact of Trojan payload selection: (**a**) golden circuit; (**b**) payload-1 which has no effect on output; (**c**) payload-2 which has effect on the output

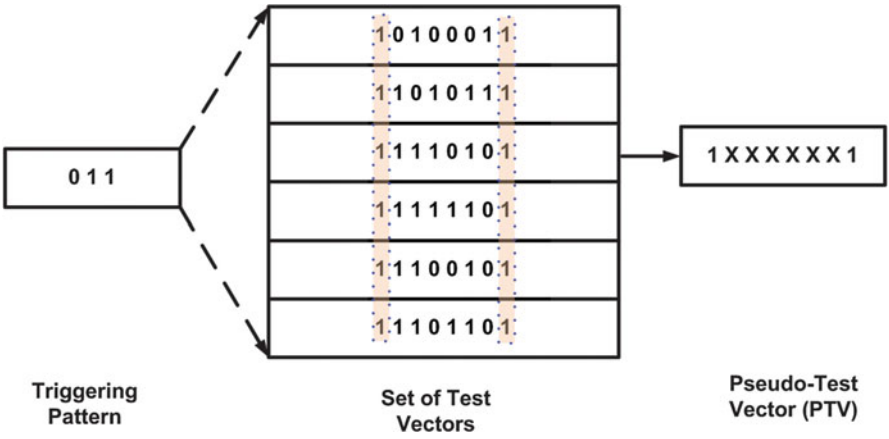


Fig. 7.13 PTV generation example: Triggering pattern (*left*); Corresponding set of test vectors (*center*); Generated PTV (*right*)

is triggered by an input vector 1111. Figure 7.12b, c show two potential payload positions. It can be easily seen that independent of the applied test vector at in circuit input, for position-1 the Trojan effect gets masked and cannot be detected. On the other hand, the Trojan effect at position 2 can be detected.

It is important to identify, for each trigger combination, the constrained primary input values. For this, we consider each trigger combination and their corresponding set of trigger test vectors at a time. To be precise, we consider the entries (s, $\{t_i\}$) from the database $\mathcal{D}$, one at a time. Let us denote the set of test vectors corresponding to a specific s as $\{t_i^s\}$. Next, for each test vector $t \in \{t_i^s\}$, we find out which of the primary inputs, if any, remains static at a specific value (either logic 0 or logic 1). These input positions are the positions needed to be constrained to trigger the triggering combination. We fill the rest of the input positions with don't-care (X) values, thus creating a single test vector containing 0, 1, and X values. We call such three-valued vectors *pseudo test vector* (PTV). Figure 7.13 illustrates the process of PTV generation with a simple example, where the leftmost and the rightmost positions of the vectors are at logic-1.

At the next step, we perform a three-valued logic simulation of the circuit with the PTV and note down values obtained at all the internal wires (nodes) which are at

Algorithm 7.4 ***SELECT_TEST_VECT***

/* Select Payload Aware test vectors */

Input: Data structure $\mathcal{D}$, circuit netlist

Output: Final test set (T_{final})

set $T_{final} \leftarrow \phi$
for all $(s, \{t_i\}) \in \mathcal{D}$ **do**
  Retrieve the test vector set $\{t_i\}$
  Compute the corresponding PTV
  do 3-value logic simulation and create the initial fault list $\mathcal{F}_s$
  if $|\{t_i^s\}| > 5$ **then**
    set $Testset \leftarrow \{t_i^s\}$
  else
    Generate extra test vectors $\{t_{ext}^s\}$ by randomly filling the X positions of the PTV
    Simulate the circuit with $\{t_{ext}^s\}$ and keep vectors satisfying s
    set $Testset \leftarrow \{t_i^s\} \cup \{t_{ext}^s\}$
  end if
  do fault simulation using HOPE with inputs $\mathcal{F}_s$ and $Testset$
  Retrieve $\mathcal{F}_{detected}^s \subseteq \mathcal{F}_s$ and $Testset_{detected} \subseteq Testset$
  Keep the subset $Testset_{comp}$ of $Testset_{detected}$, which completely covers $\mathcal{F}_{detected}^s$.
  set $T_{final} \leftarrow Testset_{comp}$
end for
return T_{final}

topologically higher positions from the nodes in the trigger combination. Then for each of these nodes, we consider a stuck-at fault according to the following rule:

1. If the value at that node is 1, we consider a stuck-at-zero fault there.
2. If the value at that node is 0, we consider a stuck-at-one fault there.
3. If the value at that node is X, we consider a both stuck-at-one and stuck-at-zero fault at that location.

At the next step, this fault list ($\mathcal{F}_s$) and the set of test vectors considered ($\{t_i^s\}$) are inputs to a fault simulator. We used the HOPE [18] fault simulator in diagnostic mode for this purpose. The output will be the set of faults that are detected ($\mathcal{F}_{detected}^s \subseteq \mathcal{F}_s$) as well as the corresponding test vectors which detect them. The detected faults constitute the list of potential payload positions for the trigger combination. Thus, after detecting the feasible payloads, we greedily select a subset of the test vector set $\mathcal{T}_s \subseteq \{t_i^s\}$ which achieves complete coverage for the entire fault list. The test vectors belonging to the rest of $\{t_i^s\}$, i.e., $\{t_i^s\} - \mathcal{T}_s$, can be discarded to be redundant. Although a greedy selection, we found that this step reduces the overall test set size significantly. Further test compaction can be achieved, at the cost of additional computational overhead, using specialized test compaction schemes (Fig. 7.14).

One important point worth noting is that it is not guaranteed by the proposed test generation scheme that all possible test vectors which trigger a particular trigger combination will get generated. As the fault list ($\mathcal{F}_s$) is calculated only based on the test vectors in $\{t_i^s\}$, it might not cover all possible payloads for a trigger

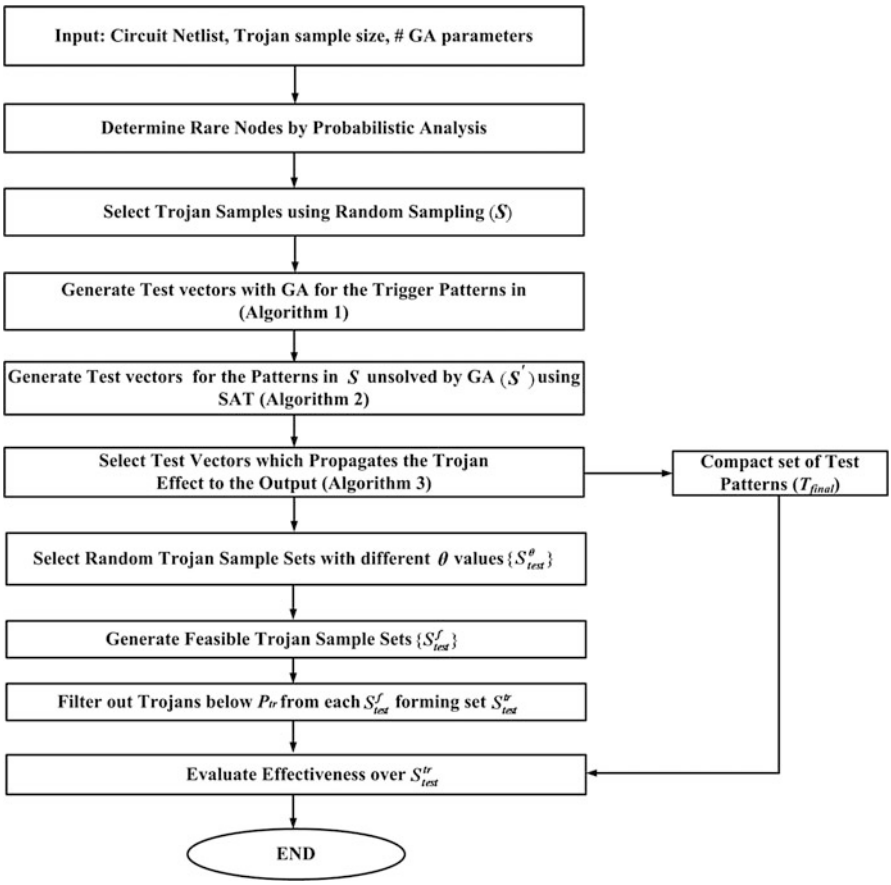


Fig. 7.14 The complete test generation and evaluation flow

combination s. However, for each test vector $t \in \{t_i^s\}$, it is guaranteed that all feasible payloads will be enumerated. Further, it can be deterministically decided if a test vector $t \in \{t_i^s\}$ will have any payload or not. In fact, for most of the trigger combinations, we got test vectors which are either redundant or doesn't have any payload. It is also observed that the number of test vectors for some hard-to-activate trigger combinations is really low (typically 1–5 vectors). For these cases, the fault coverage may be poor and many payloads for the trigger combination remain unexplored. To resolve this issue, we add some extra test vectors derived by filling the don't-care bits (if any) of the PTV. This is only done for those trigger combinations for which the number of triggering vectors is less than five. These newly generated vectors are needed to be checked by simulation so that they successfully trigger the corresponding triggering combination, before their inclusion to the test set $\{t_i^s\}$. This step is found to improve the test coverage. The compacted

test vector selection scheme is described in algorithm-7.4. At the end of this step, we obtain a compact set of test vectors with high trigger and Trojan coverage.

7.3.4.2 Evaluation of Effectiveness

It is reasonable to assume that an attacker will be only interested in Trojans with low *effective triggering probability*, irrespective of the individual *rareness values* of the constituent rare nodes. Thus, a natural approach for an attacker would be to rank the Trojans according to their *triggering probability* and select Trojans which are below some specific *triggering threshold* (P_{tr}). Intuitively, an attacker may choose the Trojan which is rarest among all, but it may lead to easy detection as very rare Trojans are often found to be significantly small in number and are expected to be well tracked by the tester. Thus, a judicious attacker would select a Trojan that will remain well-hidden in the pool of Trojans but achieves extremely low triggering probability at the same time. To simulate the abovementioned behavior of the attacker, we first select new samples of candidate Trojans from the Trojan space with varying range of θ values. We denote each of such samples as $\mathcal{S}^{\theta}_{test}$ and ensure that $|\mathcal{S}^{\theta}_{test}| = |\mathcal{S}|$. Subsequently, each of these sets is refined by the SAT tool by selecting only feasible Trojans. By feasible Trojans we mean the Trojans which are triggerable and whose impacts are visible at the outputs. We denote the set of such feasible Trojan sets obtained for different θ values as $\{\mathcal{S}^{f}_{test}\}$. Next, we filter out Trojans from these sets which are below a specified triggering threshold P_{tr}. Finally, all these subsets are combined to form a set of Trojans $\mathcal{S}^{tr}_{test}$. This set contains Trojans whose triggering probabilities are below P_{tr} and is used for the evaluation of the effectiveness of the proposed methodology. The value of P_{tr} is set to 10^{-6} based on the observation that for most of the benchmark circuits considered, there are roughly 30% Trojans which have triggering probabilities below 10^{-6}. Also, below the range ($10^{-7} - 10^{-8}$), the number of Trojans is extremely low, which may leave the attacker only with a few options.

7.3.5 Results and Discussion

7.3.5.1 Experimental Setup

The test generation schemes, including the GA and the evaluation framework, were implemented using C++. We used the *zchaff* SAT solver [12] and *HOPE* fault simulator [18]. We restricted ourselves to a random sample of 100,000 trigger combinations [8], each having up to four rare nodes as trigger nodes. We also implemented the *MERO* methodology side by side for comparison. We evaluated the effectiveness of the proposed scheme on a subset of ISCAS-85 and ISCAS-89 benchmark circuits, with all ISCAS-89 sequential circuits converted to full-scan

Table 7.4 Comparison of the proposed scheme with *MERO* with respect to testset length

Ckt.	Gates	Testset (before Algo.-3)	Testset (after Algo.-3)	Testset (MERO)	Runtime (sec.)
c880	451	6674	5340	6284	9798.84
c2670	776	10,420	8895	9340	11,299.74
c3540	1134	17,284	16,278	15,900	15,720.19
c5315	1743	17,022	14,536	15,850	15,877.53
c7552	2126	17,400	15,989	16,358	16,203.02
s15850	9772	37,384	37,052	36,992	17,822.67
s35932	16,065	7849	7078	7343	14,273.09
s38417	22,179	53,700	50,235	52,735	19, 635.22

Table 7.5 Trigger and Trojan coverage at various stages of the proposed scheme. at $\theta = 0.1$ for random sample of Trojans upto 4 rare node triggers (Sample size is 100,000 for combinational circuits and 10,000 for sequential circuits)

	GA only		GA + SAT		GA + SAT + Algo. 3	
Ckt.	Trig. Cov.	Troj. Cov.	Trig. Cov.	Troj. Cov.	Trig. Cov.	Troj. Cov.
c880	92.12	83.59	96.19	85.70	96.19	85.70
c2670	81.63	69.27	87.31	75.17	87.15	75.82
c3540	80.58	57.21	82.79	59.07	81.55	60.00
c5315	83.79	64.45	85.11	65.04	85.91	71.13
c7552	73.73	64.05	78.16	68.95	77.94	69.88
s15850	64.91	51.95	70.36	57.30	68.18	57.30
s35932	81.15	71.77	81.90	73.52	81.79	73.52
s38417	55.03	29.33	61.76	36.50	56.95	38.10

mode. The implementation was performed and executed on a Linux workstation with a 3 GHz processor and 8 GB of main memory.

7.3.5.2 Test Set Evaluation Results

Table 7.4 presents a comparison of the test set lengths generated by the proposed scheme, with that generated by *MERO*. It also demonstrates the impact of Algo-3, by comparing the test vector count before Algo-3 (TC_{GASAT}) and after Algo-3 (TC_f). As would be evident, for similar number of test patterns, the proposed scheme achieves significantly better trigger as well as Trojan coverage than *MERO*. The gate count of the circuits and the time required to generate the corresponding test sets are also presented to exhibit the scalability of the ATPG heuristic.

Table 7.5 presents the improvement in trigger and Trojan coverage at the end of each individual step of the proposed scheme, to establish the importance of each individual step. From the table, it is evident that the first two steps consistently increase the trigger and Trojan coverage. However, after the application of payload aware test set selection (Algo-3), the trigger coverage slightly decreases for some

Table 7.6 Comparison of trigger and Trojan Coverage among *MERO* patterns and patterns generated with the proposed scheme with $\theta = 0.1$; $N = 1000$ (for *MERO*) and for trigger combinations containing up to four rare nodes

	MERO		Proposed scheme	
Ckt.	Trigger coverage	Trojan coverage	Trigger coverage	Trojan coverage
c880	75.92	69.96	96.19	85.70
c2670	62.66	49.51	87.15	75.82
c3540	55.02	23.95	81.55	60.00
c5315	43.50	39.01	85.91	71.13
c7552	45.07	31.90	77.94	69.88
s15850	36.00	18.91	68.18	57.30
s35932	62.49	34.65	81.79	73.52
s38417	21.07	14.41	56.95	38.10

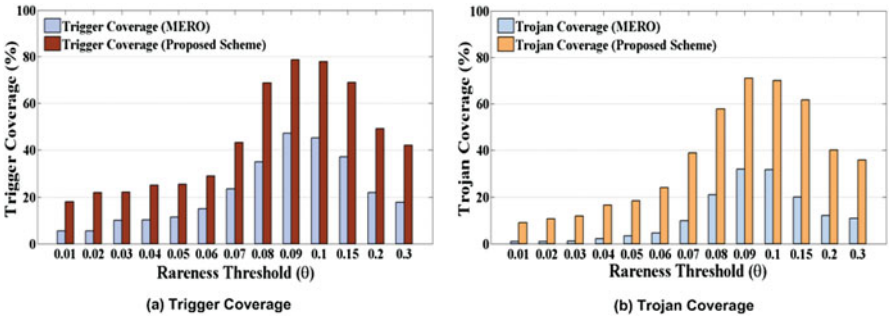


Fig. 7.15 Comparison of trigger and Trojan coverage of the proposed scheme with *MERO*, with varying triggering threshold (θ)

circuits, whereas the Trojan coverage slightly increases. The decrement in trigger coverage is explained by the fact that some of the trigger combinations do not have any corresponding payload – as a result of which, they are removed. In contrary, the addition of some "extra test vectors" by Algo-3 helps to improve the Trojan coverage.

Table 7.6 presents the trigger and Trojan coverage for eight benchmark circuits, compared *MERO* test patterns with $N = 1000$ and $\theta = 0.1$. In order to make a fair comparison, we first count the number of distinct test vectors generated by *MERO* (TC_{MERO}) with the abovementioned setup and then set GA to run until the number of distinct test vectors in the database becomes higher than TC_{MERO}. We denote the number of distinct test vectors after GA run as TC_{GA}. Note that the SAT step is performed after the GA run, and thus the total number of test vectors after the SAT step (TC_{GASAT}) is slightly higher than TC_{MERO}. The test vector count further reduces after the Algo-3 is run. We denote the final test vector count as TC_f.

To further illustrate the effectiveness of the proposed scheme, in Fig. 7.15 we compare the trigger and Trojan coverage obtained from *MERO* with that of the proposed scheme, by varying the rareness threshold (θ) for the c7552 benchmark

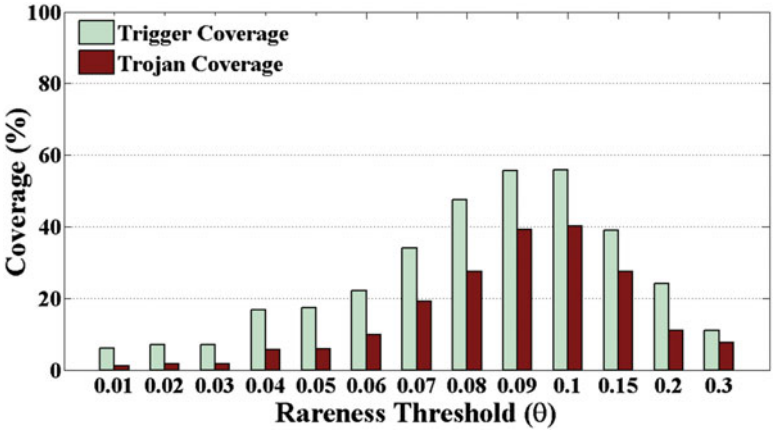


Fig. 7.16 Trigger and Trojan coverage of the proposed scheme on a set of special Trojans, which combine some easily triggerable nodes with some extremely rare nodes

circuit. It is observed that the proposed scheme outperforms *MERO* to a significant extent. Further, it is interesting to note that both *MERO* and the proposed scheme achieve the best coverage at $\theta = 0.09$. The coverage gradually decreases toward both higher and lower values of θ. The reason is that for higher θ values (e.g., 0.2, 0.3), the initial candidate Trojan sample set ($\mathcal{S}$), over which the heuristic implementation is tuned, contains a large number of "easy-to-trigger" combinations. Hence, the generated test set remains biased toward "easy-to-trigger" Trojans and fails to achieve good coverage over the "hard-to-trigger" evaluation set used. On the other hand, at low θ values, the cardinality of the test vector set created becomes very small as the number of potent Trojans at this range of θ is few, and they are sparsely dispersed in the candidate Trojan set $\mathcal{S}$. As a result, this small test set hardly achieves significant coverage over the Trojan space. The coverage for Trojans constructed by combining some easily triggerable nodes with some extremely rare nodes also follows a similar trend (shown in Fig. 7.16). It can be thus remarked that the tester should choose a θ value, so that the initial set $\mathcal{S}$ contains a good proportion of Trojans with low triggering probability, while also covering most of the moderately rare nodes.

Finally, we test our scheme with sequential Trojans. The counter-based Trojan model as described in [8] was considered. We consider Trojans up to four states, as larger Trojans have been reported to be easily detectable by side-channel analysis techniques [27]. It can be observed from Table 7.7 that as for the combinational circuits, the proposed scheme outperforms *MERO*.

Table 7.7 Coverage comparison between *MERO* and the proposed Scheme for sequential Trojans. The sequential Trojan model considered is same as [8]

	Trig. Cov. for proposed scheme		Trig. Cov. for *MERO*	
	Trojan state count		Trojan state count	
Ckt.	2	4	2	4
s15850	64.91	45.55	31.70	26.00
s35932	78.97	70.38	58.84	49.59
s38417	48.00	42.17	16.11	8.01
	Troj. Cov. for proposed scheme		Troj. Cov. for *MERO*	
	Trojan state count		Trojan state count	
Ckt.	2	4	2	4
s15850	46.01	32.59	13.59	8.95
s35932	65.22	59.29	25.07	15.11
s38417	30.52	19.92	9.06	2.58

7.3.5.3 Application to Trojan Diagnosis

Diagnosis of a Trojan once it gets detected is important for system-level reliability enhancement. In [29], the authors proposed a gate-level characterization (GLC)-based Trojan diagnosis method. The scheme proposed in this chapter can be leveraged for a test diagnosis methodology. The data structure $\mathcal{D}$ can be extended to a complete Trojan database, which will contain four tuples (s, V, P, O), where s is a trigger combination, V is the set of corresponding triggering test vectors, P is the set of possible payloads, and O is the set of faulty outputs corresponding to the test patterns in V, due to the activation of some Trojan instance. Based on this information, one can design diagnosis schemes using simple *cause-effect analysis* or other more sophisticated techniques. The complete description of a diagnosis scheme is however out of the scope of this chapter.

7.4 Conclusions

Conventional logic test generation techniques cannot be readily extended to detect hardware Trojans because of the inordinately large number of possible Trojan instances. We have presented a statistical Trojan detection approach using logic testing where the concept of multiple excitation of rare logic values at internal nodes is used to generate test patterns. Simulation results show that the proposed test generation approach achieves about 85% reduction in test length over random patterns for comparable or better Trojan detection coverage. The proposed detection approach can be extremely effective for small combinational and sequential Trojans with small number of trigger points, for which side-channel analysis approaches cannot work reliably. Hence, the proposed detection approach can be used as complementary to side-channel analysis-based detection schemes.

This work is further extended for improving the test quality by developing an ATPG scheme for detection of hardware Trojans dependent on rare input triggering conditions, based on the dual strengths of genetic algorithm and Boolean satisfiability. The technique achieves good test coverage and compaction and also outperforms a previously proposed ATPG heuristic for detecting hardware Trojans for benchmark circuits. Future research would be directed toward developing comprehensive Trojan diagnosis methodologies based on the database created by the current technique.

References

1. S. Adee, The hunt for the kill switch. IEEE Spectr. **45**(5), 34–39 (2008)
2. D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, B. Sunar, Trojan detection using IC fingerprinting, in *SP'07: Proceedings of the IEEE Symposium on Security and Privacy* (2007), pp. 296–310
3. M.E. Amyeen, S. Venkataraman, A. Ojha, S. Lee, Evaluation of the quality of N-detect scan ATPG patterns on a processor, in *ITC'04: Proceedings of the International Test Conference* (2004), pp. 669–678
4. M. Banga, M. Hsiao, A region based approach for the identification of hardware Trojans, in *Proceedings of International Symposium on HOST* (2008), pp. 40–47
5. M. Banga, M. Chandrasekar, L. Fang, M.S. Hsiao, Guided test generation for isolation and detection of embedded Trojans in ICs, in *Proceedings of the 18th ACM Great Lakes symposium on VLSI* (2008), pp. 363–366
6. R.S. Chakraborty, S. Bhunia, Security against hardware Trojan through a novel application of design obfuscation, in *Proceedings of the 2009 International Conference on Computer-Aided Design* (2009), pp. 113–116
7. R.S. Chakraborty, S. Narasimhan, S. Bhunia, Hardware Trojan: threats and emerging solutions, in *Proceedings of IEEE International Workshop on HLDVT* (2009), pp. 166–171
8. R.S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, S. Bhunia, MERO: a statistical approach for hardware Trojan detection, in *Cryptographic Hardware and Embedded Systems-CHES 2009* (2009), pp. 396–410
9. DARPA, TRUST in Integrated Circuits (TIC) – Proposer Information Pamphlet (2007). http://www.darpa.mil/MTO/solicitations/baa07-24/index.html
10. S. Dupuis, P.S. Ba, G. Di Natale, M.L. Flottes, B. Rouzeyre, A novel hardware logic encryption technique for thwarting illegal overproduction and Hardware Trojans, in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)* (2014), pp. 49–54
11. S. Eggersglüß, R. Drechsler, *High Quality Test Pattern Generation and Boolean Satisfiability* (Springer, New York, 2012)
12. Z. Fu, Y. Marhajan, S. Malik, Zchaff sat solver (2004). Available: http://www.princeton.edu/chaff
13. M.J. Geuzebroek, J.T. van der Linden, A.J. van de Goor, Test point insertion that facilitates atpg in reducing test time and data volume, in *Proceedings International Test Conference* (2002), pp. 138–147
14. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison Wesley, Boston, 2006)

15. U. Guin, K. Huang, D. DiMase, J.M. Carulli, M. Tehranipoor, Y. Makris, Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain. Proc. IEEE **102**(8), 1207–1228 (2014)
16. Y. Huang, S. Bhunia, P. Mishra, Mers: statistical test generation for side-channel analysis based trojan detection, in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), pp. 130–141
17. Y. Jin, Y. Makris, Hardware Trojan detection using path delay fingerprint, in *IEEE International Workshop on Hardware-Oriented Security and Trust, HOST 2008* (2008), pp. 51–57
18. H.K. Lee, D.S. Ha, HOPE: an efficient parallel fault simulator for synchronous sequential circuits. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **15**(9), 1048–1058 (1996)
19. B. Mathew, D.G. Saab, Combining multiple DFT schemes with test generation. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **18**, 685–696 (2006)
20. X. Mingfu, H. Aiqun, L. Guyue, Detecting hardware Trojan through heuristic partition and activity driven test pattern generation, in *Communications Security Conference (CSC)*, (2014), pp. 1–6
21. I. Pomeranz, S.M. Reddy, A measure of quality for n-detection test sets. IEEE Trans. Comput. **53**(11), 1497–1503 (2004)
22. R.M. Rad, X. Wang, M. Tehranipoor, J. Plusquellic, Power supply signal calibration techniques for improving detection resolution to hardware trojans, in *2008 IEEE/ACM International Conference on Computer-Aided Design* (2008), pp. 632–639
23. J. Rajendran, Y. Pino, O. Sinanoglu, R. Karri, Security analysis of logic obfuscation, in *Proceedings of the 49th Annual Design Automation Conference* (2012), pp. 83–89
24. E.M. Rudnick, J.H. Patel, G.S. Greenstein, T.M. Niermann, A genetic algorithm framework for test generation. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **16**(9), 1034–1044 (1997)
25. S. Saha, R.S. Chakraborty, S.S. Nuthakki, Anshul, D. Mukhopadhyay, Improved test pattern generation for hardware Trojan detection using genetic algorithm and boolean satisfiability, in *Proceedings of the 17th International Workshop on Cryptographic Hardware and Embedded Systems – CHES 2015*, Saint-Malo, 13–16 Sept 2015, (2015), pp. 577–596
26. H. Salmani, M. Tehranipoor, J. Plusquellic, A layout-aware approach for improving localized switching to detect hardware Trojans in integrated circuits, in *2010 IEEE International Workshop on Information Forensics and Security (WIFS)* (2010), pp. 1–6
27. H. Salmani, M. Tehranipoor, J. Plusquellic, A novel technique for improving hardware Trojan detection and reducing Trojan activation time. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **20**(1), 112–125 (2012)
28. S.M.H. Shekarian, M.S. Zamani, S. Alami, Neutralizing a design-for-hardware-trust technique, in *2013 17th CSI International Symposium on Computer Architecture and Digital Systems (CADS)* (2013), pp. 73–78
29. S. Wei, M. Potkonjak, Scalable hardware Trojan diagnosis. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **20**(6), 1049–1057 (2012)
30. F. Wolff, C. Papachristou, S. Bhunia, R.S. Chakraborty, Towards Trojan-free trusted ICs: problem analysis and detection scheme, in *DATE'08: Proceedings of the Conference on Design, Automation and Test in Europe* (2008), pp. 1362–1365
31. X. Zhang, M. Tehranipoor, RON: an on-chip ring oscillator network for hardware Trojan detection, in *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2011), pp. 1–6
32. B. Zhou, W. Zhang, S. Thambipillai, J. Teo, A low cost acceleration method for hardware Trojan detection based on fan-out cone analysis, in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis* (2014), p. 28

Chapter 8
Formal Approaches to Hardware Trust Verification

Farimah Farahmandi, Yuanwen Huang, and Prabhat Mishra

8.1 Introduction

With the globalization of the IC industry, the outsourcing and integration of third-party hardware IPs (intellectual property) have become a common practice for system-on-chip (SoC) design [4, 16]. However, it raises major security concerns as the attacker can insert malicious components in third-party IPs and tamper our system. There are various security threats throughout the IC design and manufacturing process. In the pre-silicon stage, we have (1) designer mistakes, rogue employees, and untrusted third-party IPs during the design integration phase, (2) untrusted EDA tools in the synthesize phase, and (3) untrusted EDA tools and untrusted vendors when design for test (DFT), design for debug (DFD), and dynamic power management (DPM) functions are added. In the post-silicon stage, we have (1) untrusted foundry during manufacturing and (2) physical attacks or side-channel attacks after the chip is shipped. It is of paramount importance to verify the trustworthiness of hardware IPs.

The task to differentiate trustworthy third-party IPs from untrustworthy ones is to detect any malicious insertions (widely known as hardware Trojans) which are not in the specification. Hardware Trojans consist of two main parts: trigger and payload. The trigger is a condition which activates the Trojan circuit, and the payload is the part of the circuit (functionality) which can be affected by an activated Trojan. A major challenge for Trojan identification is that Trojans are usually stealthy [4]. Therefore, conventional validation techniques [39, 40] fail to detect them. It is difficult to activate a Trojan and even more difficult to detect or locate it. Besides hardware Trojan, there are other trust and reliability issues concerning hardware

F. Farahmandi (✉) • Y. Huang • P. Mishra
Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA
e-mail: ffarahmandi@ufl.edu; yuanwenhuang@ufl.edu; prabhat@ufl.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_8

IPs, such as the vulnerability of test and debug infrastructure like scan chain [45] and trace buffer [25, 26], the vulnerability of cache [23] due to soft errors, and malicious parametric variations [24] like power or temperature virus.

There has been plenty of research on Trojan detection in the post-silicon stage, which focuses on first two directions in Fig. 8.1: simulation-based approaches and side-channel analysis. Simulation-based approaches aim on generating tests to activate a Trojan and propagate the payload of the Trojan to primary outputs to check with the golden circuit. The difficulty of logic testing is to generate efficient tests to activate and propagate the effect of Trojans, which are stealthy enough to hide through the traditional manufacturing testing. The MERO (multiple excitation of rare occurrences) approach [9], proposed by Chakraborty et al., can generate high-quality tests to achieve very high activation and coverage rate for Trojans. A later work by Saha et al. [41] extended this approach by using genetic algorithm to further improve the quality of tests. Side-channel analysis-based approaches focus on analyzing the side-channel signatures (such as current [27], leakage power [1], path delay [29], electromagnetic waves [34], etc.). If the side-channel signature of a chip is different from the golden chip over a certain threshold, a Trojan is detected. The problem with side-channel analysis is the presence of process variation and measurement noise. It is also difficult to detect ultrasmall Trojans which have little effect on the side-channel signal. The MERS (multiple excitation of rare switching) approach [27] can greatly increase the side-channel sensitivity of switching by generating high-quality tests. A major problem of Trojan detection in the post-silicon stage is that there is usually not much we can do with the Trojan (if detected) and need to discard the chip. Therefore, it is crucial to have design-time methods that can detect, localize, and eliminate the Trojan and produce Trojan-free circuits. Formal methods are suitable for detecting and localizing Trojans during the design phase (pre-silicon stage).

In this chapter, we focus on different formal methods to verify the design and detect undesired functions (possibly Trojans). As shown in Fig. 8.1, major formal trust validation methods include SAT-based equivalence checking, property checking, theorem proving, and symbolic algebra-based equivalence checking.

SAT-Based Equivalence Checking: Equivalence checking is used to verify that the implemented design is equivalent to the specification [19]. Both the specification and the implemented design need to be formalized into an abstract model, which is the Boolean formula in the case for satisfiability (SAT)-based equivalence checking. More details on this topic will be described in Sect. 8.2.

Property Checking: Model checkers operate on a finite transition representation of the design and check if a property holds on this model [38]. If the corresponding malicious property related to a Trojan is formally checked, the Trojan is identified. More details on this topic will be discussed in Sect. 8.3.

Theorem Proving: Given a set of axioms and hypotheses, theorem proving can work on a conjecture (logic statement) to prove or disprove it. Proof-carrying code

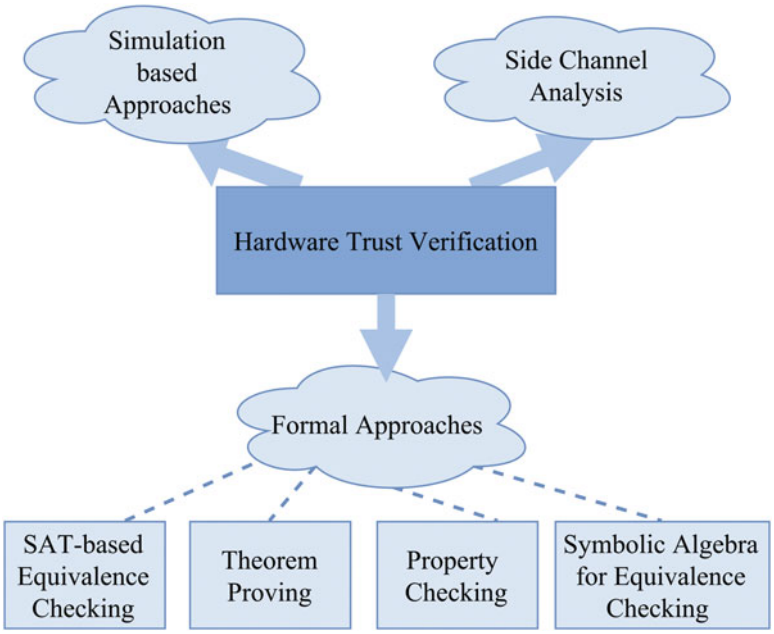


Fig. 8.1 Hardware trust verification can be categorized in three major directions: (1) simulation-based approaches, (2) side-channel analysis, and (3) formal approaches

[30] can be used as a trust metric which provides formal proof to ensure the integrity of code gathered from third-party suppliers. More details on this topic will be discussed in Sect. 8.4.

Symbolic Algebra for Equivalence Checking: This method maps the equivalence checking problem to ideal membership testing using Gröbner basis theory [16]. It starts with converting the design specification and implementation to polynomials $\{f_{\text{spec}}\}$ and $\{f_{\text{impl}}\}$ and uses symbolic algebra to check the equivalence between the two sets. More details on this topic will be discussed in Sect. 8.5.

8.2 Trust Validation Using Satisfiability Problem

Given a Boolean formula, the satisfiability problem relies on finding Boolean values to the formula's variables such that the formula is evaluated to true. If such an assignment does not exist, the formula is called unsatisfiable. It means that any possible assignments to formula's variables force the formula to be evaluated to false. The Boolean formula is constructed from AND, OR, and NOT operators between various variables which can be either assigned to true or false. Many of the validation and debugging problems can be mapped to satisfiability problems.

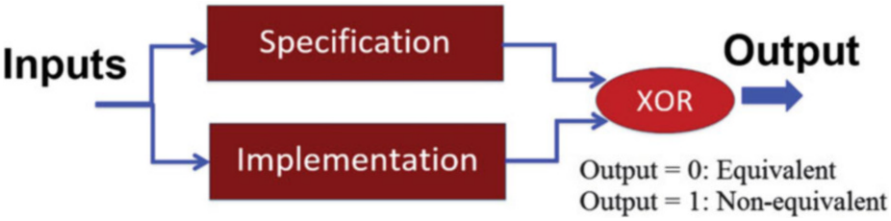


Fig. 8.2 Equivalence checking using SAT solvers

One of the applications is to check the equivalence between the specification of the circuit and its implementation using SAT solvers. Figure 8.2 shows the equivalence checking using SAT solvers. If the specification and implementation have the same functionality, the output of the XOR gate should always be false. If the output of XOR gate becomes true for any input pattern, it implies that the implementation and the specification do not have the same functionality for the same input pattern. In other words, if the circuit shown in Fig. 8.2 is converted to conjunction normal form (CNF), a SAT solver can be used to check the equivalence between the specification and implementation. If the SAT solver reports unsatisfiability, we can conclude that specification and implementation are equivalent. Otherwise, they are not equivalence, and the root of mismatch should be found. SAT solvers are excessively used for design validation [2, 3, 32].

Equivalence checking can be done using SAT solvers to identify hardware Trojans [20]. If hardware Trojans exist in the implementation, the SAT solver finds assignments to the internal variables to reveal the hidden Trojan. However, this method requires a golden model and suffers from scalability issues. The SAT solver may encounter state explosion when the design is large, and the specification and the implementation significantly differ from each other.

Several works explore the existence of Trojans in unspecified functionality [17, 18]. Therefore, the Trojan does not alter the specification of the design, and existing statistical or simulation-based methods cannot identify the Trojan-inserted design [19]. Fern et al. propose a SAT-based technique to detect Trojans which exploit the design signals in their unspecified functionality to cause malfunction. Figure 8.3 shows a hardware Trojan in an unspecified functionality of a FIFO. The designer did not specify the functionality of the FIFO when the "*read_enable*" of the FIFO is not asserted. The attacker takes the advantage of the incomplete specification and inserts a malicious circuit to leak the secret information when the *read_enable* signal is not asserted. This kind of Trojans can be inserted in RTL code or at any high-level description of the design.

Fern et al. try to address unspecified Trojan detection where the Trojan targets information leakage [19]. Suppose that the function "*func*" is unspecified when internal signal "*s*" is under condition "*C*." Suppose that signal s can have two possible values: v_0 and v_1. Under condition C, Eq. 8.1 should be unsatisfiable if the design is Trojan-free. Therefore, any assignment which make Eq. 8.1 satisfiable,

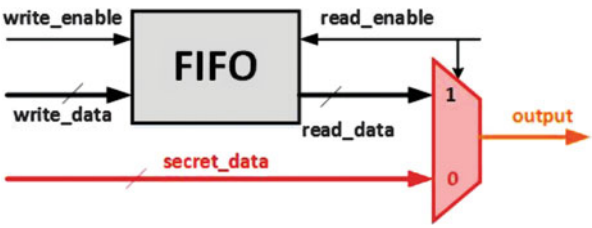


Fig. 8.3 Trojan in unspecified functionality of a FIFO [19]

it is a trace to detect the covert Trojan. For example, in the FIFO shown in Fig. 8.3, *output* signal should remain the same when the *read_enable* $= 0$ ($C =$ *read_enable* $= 0$ and $s =$ *output*).

$$C \wedge (func(s = v_0) \oplus func(s = v_1)) \tag{8.1}$$

To detect Trojans in an unspecified functionality of the design, pairs C and s should be identified. For any function in the design, several s and C pairs can be found, and the process of marking the potential pairs is not automatic yet. For every pair (s, C), one CNF formula is constructed, and an SAT solver (for Boolean values) or satisfiability module theory solvers (SMT solvers) can be used to find the potential threats. The Trojan can be detected when the CNF formula is satisfiable. The success of this approach is dependent on the SAT solvers and identifying (s, C) pairs. Moreover, the approach requires manual intervention.

8.3 Security Validation Using Property Checking

Model checking is a famous technique in design verification which checks a design for a set of given properties. To solve the model checking problem, the design and the given properties are converted to a mathematical model/language, and all of the design's states are checked to see whether the given properties are satisfied. A class of model checkers is designed based on temporal logic formula [11]. The properties are described using linear temporal logic (LTL) formulas to describe expected behaviors of the design. The properties are checked using the model checkers. A model checker either proves the correctness of a given property over all of the possible behaviors of the design or find a counterexample when the property fails.

A model checker tries all of the possible states of a design to prove a given property using a binary decision diagram (BDD). However, the number of design states can be huge since every bit introduces two states in the design. For example, a 32-bit register can add 2^{32} states to the design state space. Although some techniques such as slicing, abstracting, etc. have been proposed [10, 44], state space explosion still is the largest limitation of using model checking in property verification. Bounded model checking (BMC) is introduced to overcome the amount of memory that a model checker requires for constructing and storing different states of a design

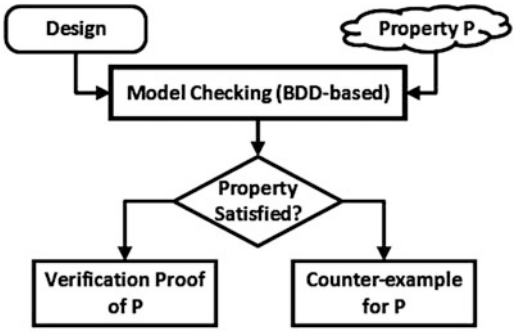


Fig. 8.4 Verification using model checking

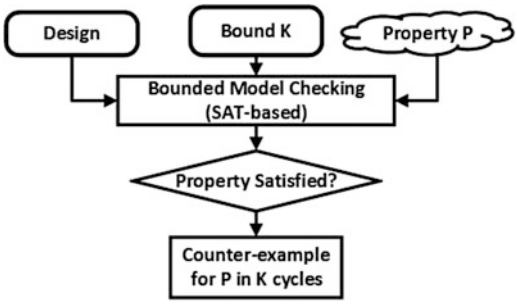


Fig. 8.5 Verification using bounded model checking

[5]. BMC tries to find a counterexample in the first K cycles during execution. If a counterexample is found within K cycles, the property does not hold. Otherwise, K can be increased in the hope of finding a counterexample in upper bounds. BMC is not able to prove a property since it unrolls the circuit for a specified number of clock cycles. However, it can provide a statistical metric for a given property when the model checker fails (e.g., no counterexample can be found in K clock cycles). The BMC problem can be mapped to satisfiability problem, and SAT solvers can be utilized to solve the problem. Therefore, the BMC addresses some of the state space explosion problems associated with BDDs in model checking. Figures 8.4 and 8.5 show the model checking and bounded model checking approaches, respectively. Clearly, bounded model checking cannot provide proof for property P. However, it can reveal when property P is violated within K clock cycles.

Security properties describe the expected behaviors which a trustworthy design is required to follow. Model checkers can be used to ensure safety properties. A SoC designer and a third-party vendor can agree on certain security properties that should be held on the design. When the design is sent to the SoC integrator, the SoC integrator converts the design to a formal description to check the security properties using a model checker. If all of the security properties are verified, the expected security behaviors are met. Rajendran et al. have proposed a Trojan detection technique which is based on using bounded model checking [36]. They have considered the threat model as an attempt to corrupt the critical data such as secret keys of a cryptographic design, random numbers which are required by most

of the cryptography algorithms, or stack pointer of a processor. The assumption is that these critical data should be stored in some specific registers, and accesses to these registers should be protected. In other words, the registers which contain critical data should be accessed through valid ways, and any undefined access to these registers is considered as a threat. The safe access conditions to these registers are formulated as properties (assertions), and a bounded model checker is utilized to find a counterexample when the security properties are violated.

Example 1 Suppose that the program counter (PC) register is considered as a critical data. The only valid ways to change the PC register is either using a reset signal (V_1), by CALL instruction which increments the PC register V_2, or using RET instruction which decrements the value of the PC register V_3. Otherwise, the PC register should keep its value. The safety property of PC register can be formulated as:

$$Safe\_PC\_change : assert \quad always \quad PC_{\text{access}} \notin \mathbb{V} = \{V_1, V_2, V_3\} \rightarrow PC_t = PC_{t-1})$$

When this property is fed into a bounded model checker alongside with the processor design, a counterexample is expected to be found whenever PC register or a part of it is changed using an unauthorized access. ■

Using model checking to find unauthorized access to secret and critical data of design is beneficial since the method does not require any golden model of the design. However, the success of this method is dependent on the SAT engine (it may fail for large and complex designs) and precise definition of security properties which needs prior knowledge of all safe ways to access a critical register. The performance of the presented method can be further improved using an automatic test pattern generation (ATPG) tool to ensure the trustworthiness of the assets for a large number of clock cycles. The property is synthesized as a circuit monitor, and it is appended to the original design. The ATPG tool is used to generate a test for a stuck-at-1 fault at the output of the monitor circuit. The counterexample can be found if the test can be generated. The success of this approach is dependent on the ATPG tools and complete definition of circuit monitors.

A similar model checking-based approach is used to detect information leakage [37]. The security properties check whether there is an input assignment (or a sequence of input assignments) I which triggers the leakage of secret data S to output ports or observable points (O) of the design.

$$\exists i \in I \rightarrow (S == O)$$

The property and formal description of the design are fed into a bounded model checker to find the possible leakage. However, the abovementioned property has several challenges. If the secret information S contains n bits, the model checker needs to check 2^n different values. Checking all possible values may not be feasible when n is in the order of hundred (which is normal for encryption algorithms). The authors have proposed some refinements to limit the Trojan search to make

information leakage detection feasible. However, the assumptions and refinement rules restrict the applicability of the solution to find different information leakage threats. Moreover, BMC may fail since the complexity of problem increases for each cycle of unrolling. Therefore, BMC works only for a certain number of clock cycles depending on the design size. When an adversary inserts a Trojan, which is triggered after a large number of clock cycles, this method cannot detect the Trojan.

Security property checking can be done in two general ways: (i) checking forbidden behaviors and (ii) checking expected security properties. The malicious behavior of design is formalized and checked using model checkers in [38]. The method can be applied only for known Trojan types. Hasan et al. have proposed a hardware Trojan detection technique using LTL and computation tree logic (CTL) security properties to generate hardware Trojan monitors in order to improve the resiliency of hardware designs against malicious functionality [22]. The attacker is considered as an untrustworthy third-party designer that can insert Trojans in the IP, and the defender is SoC integrator. The SoC integrator needs to formulate dangerous behaviors as security properties to perform vulnerability verification using model checkers. The generated counterexample, as well as the involved signals, is provided to the in-house designers to produce a guideline for efficient run-time security monitors.

Potential threats introduced from electronic design automation (EDA) tools of the third party are considered in [35]. It is possible that an adversary modifies a design using nontransparent EDA tools such as synthesis tools. A synthesis tool may optimize some registers and unsafely modify the finite-state machine (FSM). The authors have proposed a hardware Trojan detection technique which is based on property coverage analysis to ensure that a gate-level netlist is free from hardware Trojans inserted by synthesis tools. The proposed Trojan detection method is based on both security property checking and state coverage to mark suspicious unused circuit states. Figure 8.6 shows the different ways to insert Trojans in an FSM.

Example 2 Consider the FSM shown in Fig. 8.6a. Whenever the current state is A, the next state should be either A or B. The property can be formulated using a LTL formula as shown below:

$$assert \quad always \quad (cur\_state == A) \rightarrow X(next\_state = A || \quad next\_state = B)$$

Note that X symbol shows the next cycle and $\rightarrow$ shows implication. ■

The success of using model checking-based approaches to detect hardware Trojans is highly dependent on the size of the design, SAT engine, and the quality of the provided properties. The model checker cannot guarantee inexistence of hardware Trojans. However, it can provide a trust-level metric.

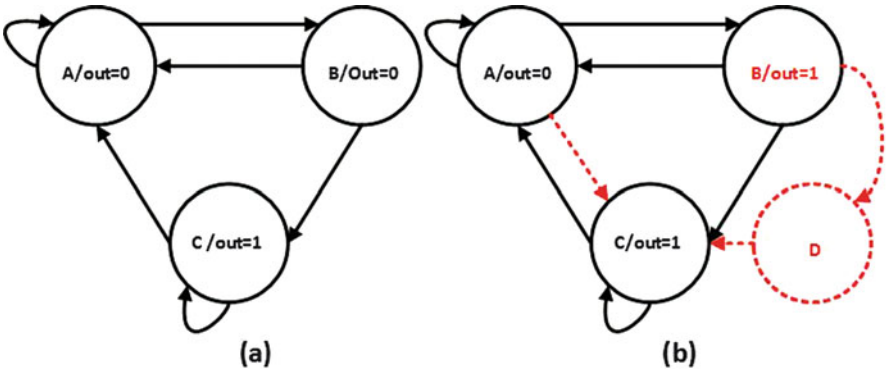


Fig. 8.6 Trojans in an FSM. (**a**) A Trojan-free FSM; (**b**) Trojan can be inserted to an FSM using different ways: (i) changing the state output (e.g., state *B*), (ii) modification to state transitions (e.g., extra transition from state *A* to *C*), and (iii) adding extra states (e.g., state *D*) and transitions (such as state transitions $B \rightarrow D$ and $D \rightarrow C$) to FSM

8.4 Theorem Provers for Trojan Detection

The attempt of proving a conjecture (logical statement) from a set of axioms and hypotheses is called theorem proving. Problems from different domains such as mathematics, hardware, and software verification can be mapped to theorem proving. Automated theorem provers (ATPs) are computer programs which try to prove given problems. They need appropriate and precise formulation of conjectures, axioms, and hypotheses in logical languages such as first-order logic or higher-order logic. The language provides a formal description of the problem's statements; therefore, mathematical manipulation of statements is possible using ATPs. In other words, ATPs can show how the conjecture can be logically proved by following a set of related fact and statements. Figure 8.7 shows an overview of a theorem proving flow.

8.4.1 Secret Data Protection Using Proof-Carrying Codes

Theorem provers are widely used in trust validation domain. Proof-carrying code (PCC)[33] has been proposed to provide a trust metric for a code often gathered from untrusted suppliers. The idea is that the code consumer should be able to confirm a set of predefined properties when the code producer delivers it. Using the PCC, the supplier is required to provide the formal proof of safety properties, and the consumer performs the proof validation to ensure the integrity of the code. Proofs are generated using theorem provers. Jin and Makris [30] have proposed an IP information tracking methodology which is based on proof-carrying concept. The method is proposed to establish the trust between the untrusted IP vendor and the

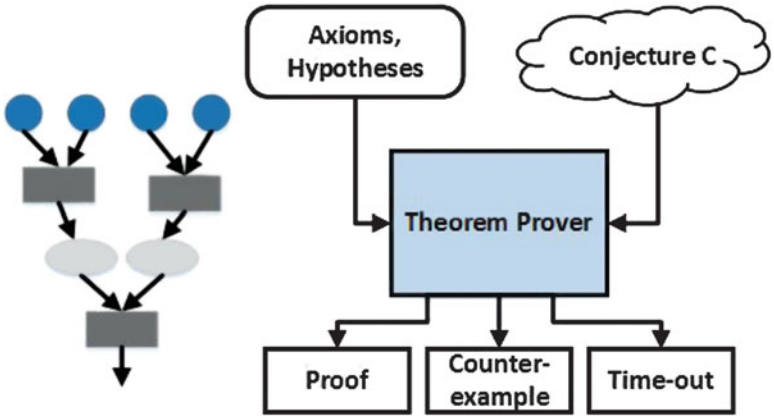


Fig. 8.7 Validation using theorem prover

SoC Integrator. The IP vendor creates proof of security properties that was agreed upon with the SoC integrator. The design is instrumented with secrecy tags, and it is converted to a formal description. The design, secret tags, as well as their formal descriptions are passed to the consumer as a package. The consumer formalizes the agreed security properties and regenerates the formal model of the HDL code to enable a formal property checker to validate proofs delivered by the producer. The assumption is that if an adversary inserts a hardware Trojan in the design to violate security properties, the proof validation will fail. Figure 8.8 shows the overall approach. The "Pass" result demonstrates the security preserving behavior of the delivered IP. However, the "Fail" result reveals the existence of malicious code to potentially leak secret information. Jin and Makris [30] have developed a set of rules to convert an HDL code to Coq formal language to automate formal model generation procedure.

Example 3 Suppose that the IP vendor is asked to deliver the implementation of DES algorithm [6] which is a data encryption algorithm. Our goal is to ensure that none of the secret data can be leaked through primary outputs. This requirement can be formalized as theorem "*Safe_DES*." In the implementation of DES algorithm, the secret input "*KEY*," plaintext "*DesIn*," and internal key rounds "*KEY_Rounds*" are considered as secret information, and they should be protected. Therefore, they will be marked with security tags. Three axioms showing the secret character of *KEY*, *Des_In*, and *KEY_Rounds* in all clock cycles are added to the formal logic which is used by the theorem prover. In the next step, three axioms, the *DES* implementation, as well as the *safe_DES* theorem, are converted to the Coq formal model. The theorem prover tries to prove the theorem by considering the formal behavior of the DES algorithm and axioms. If the proof can be generated, the *DES* algorithm is safe. Therefore, the design and the proofs can be delivered to the consumer. The user validates the proof using a property checker. ■

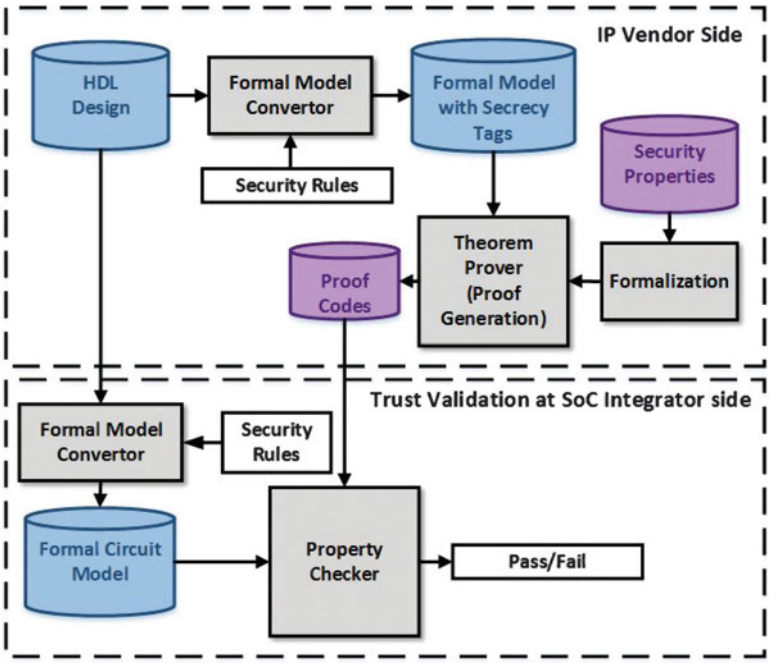


Fig. 8.8 Information flow tracking based on proof-carrying codes

A dynamic information tracking approach has been proposed in [31]. Similar to the statistical approach, the design, as well as security properties, is formalized and validated to detect unauthorized leakage of sensitive data. Unlike statistical scheme, all variables are assigned to a list of values showing their level of sensitivity during different clock cycles. Some updating rules are designed to change the sensitivity values of different variables over time. Two sensitivity lists are considered for data protection, (i) initial list and (ii) the stable list, where values indicate stable sensitivity levels of circuit signals. The SoC integrator checks the content of both lists first to ensure the safe distribution of the secret data through the circuit. In the next step, proofs are validated.

A similar approach has been applied for trust validation of EDA tools [28]. The idea is that the trustworthy RTL design cannot guarantee the security of the IP cores. The gate-level netlist may be contaminated either by malicious implants or untrusted EDA tools. Jin [20] has proposed a framework to check the trust level of the produced gate-level netlists of synthesis tools. The framework consists of three major steps: (i) generates proof-carrying code based on the security properties of the trusted RTL code, (ii) validates proofs on the corresponding synthesized gate-level netlist, and (iii) measures the trust level of the EDA tools based on the result of the second step. The aforementioned approaches require flattening the design to check the integrity of the whole system. However, design flattening increases

the complexity of formalization steps and proof generation phase and limits the applicability of these approaches due to scalability problems. Similar to property checking approaches discussed in Sect. 8.3, the Trojan detection is dependent on the quality of security properties.

8.4.2 Integration of Theorem Provers and Model Checkers

The primary challenge in using proof-carrying code approach is to measure the trust of RTL, or gate-level code is the scalability. SoC designs are usually large and complex, and proofs cannot be easily constructed as the size of the design is increased. Proof validation is also very time-consuming for large designs. To address these issues, an integrated approach has been proposed [21]. The main idea is to combine interactive theorem provers with model checkers to check security properties. The hierarchical structure of the SoC design is considered as a choice of design partitioning to reduce the verification efforts and address the scalability issues. The security properties are formalized as theorems, and theorems are decomposed into several lemmas each related to one partition of the design. The lemmas are converted to assertions, and a model checker is used to validate the assertions. Each assertion is called a sub-specification. Sub-specifications are selected in a way that they are dependent on each other. If all of the partitions satisfy security lemmas, we need to use the theorem prover to validate the security of whole design using checked lemmas. If the security theorem can be proven, the whole system is considered safe.

This method distributes the proof construction to overcome the scalability problem. However, the success of this approach is dependent on the model checking engine. In some cases, it may not be possible to validate the lemmas because of the model checker limitations.

8.5 Trojan Detection Using Symbolic Algebra

Equivalence checking is used to formally prove two representations of a design exhibit the same behavior. Equivalence checking can be done between different layers of design abstractions. Equivalence checking has been done using SAT solvers and industrial tools such as Formality [43] traditionally. However, these methods are promising when the specification and implementation structures are similar such as between RTL (pre-synthesis) and gate-level (post-synthesis) models. However, existing methods can lead to state space explosion when complex SoCs are involved with significantly different (lack of structural or FSM-level similarity) specification and implementation.

A promising direction to address the state space explosion problem of equivalence checking of hardware design is to use methods based on symbolic computer

algebra. Symbolic algebraic computation refers to the application of mathematical expressions and algorithmic manipulations methods to solve different problems. Symbolic algebra has received attention because of its applicability in equivalence checking of hardware designs. There are equivalence checking methods based on symbolic algebra that are successful to detect deviations from the specification for combinational circuits especially arithmetic circuits [13, 15, 20, 42]. This method maps the equivalence checking problem to ideal membership testing and solves the problem using Gröbner basis theory.

8.5.1 *Equivalence Checking Using Gröbner Basis Theory: Background*

A set of polynomials $\mathbb{F} = \{F_1, F_2, \ldots, F_n\}$ which are defined over a field generates an ideal I where $I =< F_1, F_2, ..., F_n >$. Set $\mathbb{F}$ is called basis or generator of ideal I. Generally, ideal I can have several bases. One of the bases is called Gröbner basis G which can be derived from Buchberger's algorithm [8]. The main characteristic of Gröbner basis is the ability to solve the ideal membership problem [7]. In other words, if we want to check whether polynomial f resides in ideal I, f can be reduced over setting G (reduction can be done using polynomial sequential division regarding a specific order). If the result of reduction is equal to zero polynomial, f belongs to ideal I. Otherwise f does not reside in the ideal I [12].

The equivalence checking problem can be efficiently mapped to ideal membership testing for arithmetic circuits. In order to apply Gröbner basis theory, specification is modeled as polynomial f_{spec}, and implementation is converted to a set of polynomials $\mathbb{I} = \{f_1, f_2, \ldots, f_s\}$, and ideal I is constructed as $I =< \mathbb{I} >=< f_1, f_2, \ldots, f_s >$. Set $\mathbb{I}$ is derived in a way that every pair (f_i, f_j) has relatively prime leading monomials. Thus, set $\mathbb{I}$ is also Gröbner basis (G) of ideal I $(\mathbb{I} = G)$. To check equivalence between specification and implementation, f_{spec} is reduced over set G. If the remainder is zero, it shows that the implementation has correctly implemented the specification. In other words, the specification and implementation are equivalent.

The arithmetic circuit equivalence checking problem formulation starts with converting the design specification to a polynomial f_{spec} which represents the word-level abstraction of arithmetic circuit functionality using primary inputs and primary outputs as variables. For example, the specification of a n-bit adder with primary inputs $A = \{a_0, a_1, \ldots, a_{n-1}\}$ and $B = \{b_0, b_1, \ldots, b_{n-1}\}$ and primary output $Z = \{z_0, z_1, \ldots z_n\}$ can be formulated as $Z = A + B$ or can be written as $(2^n.z_n + \ldots + 2.z_1 + z_0) - ((2^{n-1}.a_{n-1} + \ldots + 2.a_1 + a_0) + (2^{n-1}.b_{n-1} + \ldots + 2.b_1 + b_0)) = 0$ where $\{a_i, b_i, z_i\} \subset \{0, 1\}$.

The functionality of logic gates (such as AND, OR, XOR, NOT, and buffer) can be represented by polynomials such that the input and output signals of gates act as variables of the corresponding polynomial. Each variable x_i which appears

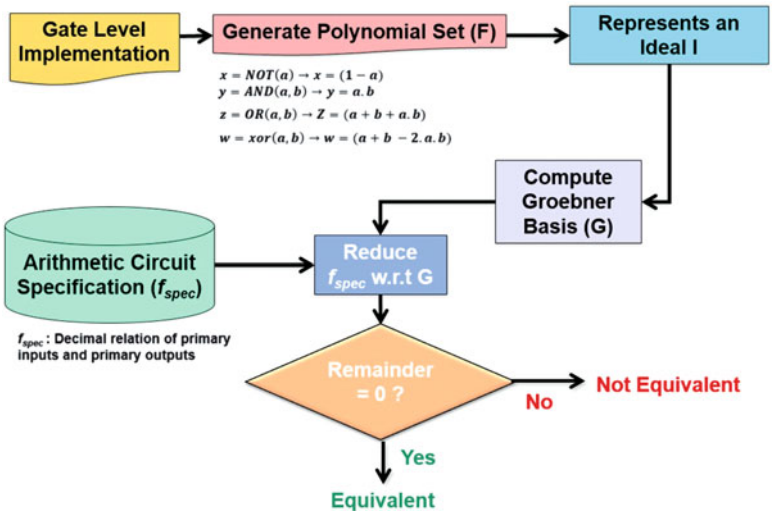


Fig. 8.9 Overview of arithmetic circuit verification using Gröbner basis theory

in a circuit polynomial belongs to $\mathbb{Z}_2$ where $({x_i}^2 = x_i)$. The basic logic gates are described using polynomials shown in Fig. 8.9. In other words, a gate-level netlist of a circuit can be modeled as a set of polynomials $\mathbb{F}$ by modeling each gate as a polynomial. Suppose that we want to make sure an arithmetic circuit implements correctly its specification. In other words, we want to verify that there are no functional errors in the arithmetic circuit. The equivalence checking starts with consecutively reducing the f_{spec} over implementation polynomials ($\mathbb{F}_{\text{imp}}$) until either zero remainder or a remainder that contains only primary input variables is reached. If the remainder is zero, it shows that the arithmetic circuit performs the exact specification. However, the non-zero remainder shows that the implementation is not trustworthy and there are some malfunctions. Figure 8.9 shows the overview of the equivalence checking approach.

Example 4 Suppose that we want to verify the functional correctness of a full-adder implementation shown in Fig. 8.10. The specification can be formulated as $(2.C_{\text{out}} + S - (A + B + C_{\text{in}}))$, and each gate in the implementation can be modeled as a polynomial based on Fig. 8.9. The topological order of the circuit (since the circuit is acyclic) is chosen for reduction as $C_{\text{out}} > \{S, n_3\} > \{n_2, n_1\} > \{A, B, C_{\text{in}}\}$. The reduction starts from the most significant primary output and ends at primary inputs. Variables in the curvy brackets have the same order, and they can be reduced in one iteration. Equation 8.2 shows the reduction process. It can be seen that the final result (remainder) is a non-zero polynomial and implementation is untrustworthy. ■

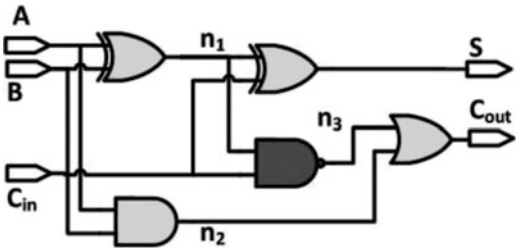


Fig. 8.10 Faulty gate-level netlist of a full adder

$$
\begin{aligned}
&step_0 : 2.C_{\text{out}} + S - A - B - C_{\text{in}} \\
&step_1 : S - 2.n_3.n_2 + 2.n_3 + 2.n_2 - A - B - C_{\text{in}} \\
&step_2 : 2.n_2.n_1.C_{\text{in}} - 4.n_1.C_{\text{in}} + n_1 - A - B + 2 \\
&step_3(remiander) : 8.A.B.C_{\text{in}} - 4.A.C_{\text{in}} - 4.B.C_{\text{in}} - 2.A.B + 2
\end{aligned}
\tag{8.2}
$$

8.5.2 *Trojan Activation and Detection in Arithmetic Circuits Using Symbolic Algebra*

From the security point of view, it is extremely important to make sure that the design performs exactly the intended specification, nothing more, nothing less. Every extra, incorrect, or missing component can threaten the security of the design. The remainder (as shown in Example 4) defines the results of the equivalence checking. It has been shown that the remainder can be beneficial in root causing the threat [14]. The threat model is considered any deviation from the expected functionality. The remainder can be utilized to generate tests to activate the Trojan. If there are more than one malicious functionality in the implementation, the remainder will be affected by all of them. Therefore, each assignment that makes the remainder non-zero activates at least one of the existing faulty scenarios. The generated tests and the remainder's patterns can be used to localize malicious scenarios.

8.5.3 *Trojan Localization in Third-Party IPs*

The approach presented in Sect. 8.5.2 can be extended to find whether a hardware Trojan, which changes the functionality, has been inserted in a combinational arithmetic circuit. However, applying the same approach on general IPs is limited due to several reasons. First, it is possible that the specification of a general circuit cannot be described as one simple polynomial. Second, the circuit may not be

acyclic, and loops may exist due to their sequential nature. Third, unrolling may increase the complexity of the problem, so the reduction of f_{spec} over implementation polynomials will face polynomial terms explosion. Finally, the Trojan activation may require an extremely large number of unrolling steps which may be practically infeasible, and also there is no specific information on after how many cycles Trojan will be activated.

To address these challenges, Farahmandi et al. have proposed a method to localize and detect Trojans in third-party IPs [16] based on Gröbner basis. The method localizes and activates Trojans between two versions of design. Suppose that there is a golden model of design (specification) and a modified version (implementation) of it (after performing some nonfunctional changes such as doing synthesis, adding clock trees, scan-chain insertion, etc.). Once the third-party IP comes back (after synthesis or other functionality-preserving transformations), it is crucial to ensure the trustworthiness of these IPs. The method is based on extracting two different sets of polynomials from the specification and the implementation netlist. The equivalency between two sets is checked using Gröbner basis reduction. Figure 8.11 shows the overall view of the approach.

Unlike arithmetic circuits which can be defined with one general specification polynomial, a general IP can be represented by a set of polynomials $\mathbb{S}$ extracting from the golden IP. The golden netlist (specification) is partitioned into several regions, and each region is converted to a polynomial. The output of each region is either input of a flip-flop (clock, enable, reset, etc.) or one of the primary outputs. The inputs of a region are either from primary inputs or inputs/outputs of flip-flops. Then, corresponding equations (based on Fig. 8.9) of gates inside a region

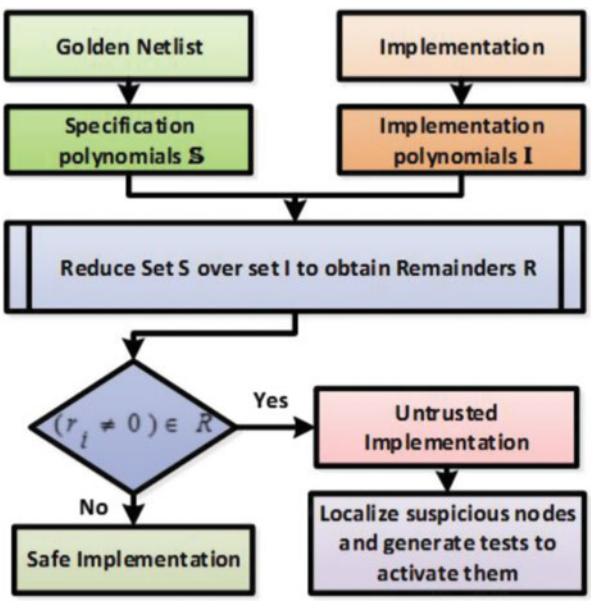


Fig. 8.11 Overall view of Trojan localization in third-party IPs using Gröbner basis reduction

are combined to construct one polynomial representing the functionality of the region. Similarly, the implementation polynomials $\mathbb{I}$ are driven by modeling every gate except flip-flops from the untrusted design as a polynomial.

To detect a Trojan, each polynomial f_{spec_i} from set $\mathbb{S}$ is reduced over a subset of polynomials from set $\mathbb{I}$ to check membership of every polynomial f_{spec_i} in ideal I constructed from polynomials from set $\mathbb{I}$ ($I =< \mathbb{I} >$). The process continues until f_{spec_i} is reduced either to zero polynomial or a remainder polynomial which contains primary inputs as well as flip-flop's inputs/outputs. The non-zero remainder indicates that implementation does not correctly implement the functionality of f_{spec_i} and that part of the implementation is suspicious. Note that, based on Gröbner basis theory, when the remainder is zero for a specific region, the region is safe. In other words, it is not possible for a smart attacker to insert malicious gates in a way that the remainder becomes zero. By using this approach, a set of malicious regions is identified. Suppose the adversary inserts some extra flip-flops as part of Trojans. These buggy flip-flops do not have any correspondence in the specification. In other words, there is no f_{spec_i} which describes their inputs' functionality. Therefore, the corresponding region in the implementation is also considered as a suspicious region. However, scan-chain flip-flops can easily be detected and removed from suspicious candidates because of their structures. To identify the gates that most likely are responsible for the malicious activity, gates which are contributing to the construction of safe regions (regions which have zero remainders) are removed from suspicious regions. This technique reduces the number of suspicious gates. Figure 8.12 shows the pruning procedure of suspicious gates.

In this chapter, we described various formal approaches for verification of hardware security and trust. Considering the increasing importance and complexity designing trustworthy SoCs, it is expected that there will be many more approaches for trust analysis and validation in the near future.

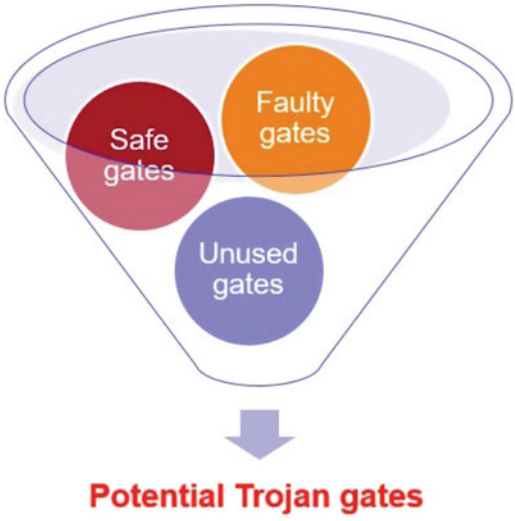


Fig. 8.12 Potential Trojan gates are equal to: $Gates_{\text{suspicious}} = (Gates_{\text{faulty}} - Gates_{\text{safe}}) \cup Gates_{\text{unused}}$

8.6 Conclusion

Several security issues such as intentional or unintentional design mistakes can be introduced during different stages of a SoC design. Therefore, trust establishment becomes critical. Formal methods are promising in hardware validation as they evaluate the functionality and structure of the mathematical model to verify that the design contains the functions described in the specification. Moreover, they can provide proofs for security properties. Formal approaches for security verification can be broadly classified into four groups: (i) approaches based on satisfiability problem, (ii) property checking approaches, (iii) theorem proving strategies, and (iv) equivalence checking methods. In this chapter, we discussed security validation methods based on each category of formal methods to improve security validation efforts at different stages of the design life cycle.

References

1. J. Aarestad, D. Acharyya, R. Rad, J. Plusquellic, Detecting trojans through leakage current analysis using multiple supply pads. IEEE Trans. Inf. Forensics Secur. **5**(4),893–904 (2010). doi:10.1109/TIFS.2010.2061228, ISSN:1556-6013
2. P. Behnam, B. Alizadeh, In-circuit mutation-based automatic correction of certain design errors using sat mechanisms, in *2015 IEEE 24th Asian Test Symposium (ATS)* (IEEE, 2015), pp. 199–204
3. P. Behnam, B. Alizadeh, Z. Navabi, Automatic correction of certain design errors using mutation technique, in *2014 19th IEEE European Test Symposium (ETS)* (IEEE, 2014), pp. 1–2
4. S. Bhunia, M.S. Hsiao, M. Banga, S. Narasimhan, Hardware trojan attacks: threat analysis and countermeasures. Proc. IEEE **102**(8), 1229–1247 (2014)
5. A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, Y. Zhu, Bounded model checking. Adv. comput. **58**, 117–148 (2003)
6. E. Biham, A. Shamir, Differential cryptanalysis of des-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991)
7. B. Buchberger, Some properties of gröbner-bases for polynomial ideals. ACM SIGSAM Bull. **10**(4), 19–24 (1976)
8. B. Buchberger, A criterion for detecting unnecessary reductions in the construction of a groebner bases, in *EUROSAM*, 1979
9. R.S. Chakraborty, F. Wolf, C. Papachristou, S. Bhunia, Mero: a statistical approach for hardware Trojan detection, in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'09)*, 2009, pp. 369–410
10. A. Cimatti, E. Clarke, F. Giunchiglia, M. Roveri, Nusmv: a new symbolic model checker. Int. J. Softw. Tools Technol. Transfer **2**(4), 410–425 (2000)
11. E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. (TOPLAS) **8**(2), 244–263 (1986)
12. D. Cox, J. Little, D. O'Shea, *Ideals, Varieties, and Algorithms* (Springer, New York, 1997)
13. F. Farahmandi, B. Alizadeh, Grobner basis based formal verification of large arithmetic circuits using gaussian elimination and cone-based polynomial extraction, in *Microprocessor and Microsystems – Embedded Hardware Design*, 2015, pp. 83–96

14. F. Farahmandi, P. Mishra, Automated test generation for debugging arithmetic circuits, in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, 2016), pp. 1351–1356
15. F. Farahmandi, B. Alizadeh, Z. Navabi, Effective combination of algebraic techniques and decision diagrams to formally verify large arithmetic circuits, in *2014 IEEE Computer Society Annual Symposium on VLSI* (IEEE, 2014), pp. 338–343
16. F. Farahmandi, Y. Huang, P. Mishra, Trojan localization using symbolic algebra, in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE, 2017), pp. 591–597
17. N. Fern, S. Kulkarni, K.-T.T. Cheng, Hardware Trojans hidden in RTL don't cares – automated insertion and prevention methodologies, in *2015 IEEE International Test Conference (ITC)* (IEEE, 2015), pp. 1–8
18. N. Fern, I. San, C.K. Koç, K.-T.T. Cheng, Hardware Trojans in incompletely specified on-chip bus systems, in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe* (EDA Consortium, 2016), pp. 527–530
19. N. Fern, I. San, K.-T.T. Cheng, Detecting hardware trojans in unspecified functionality through solving satisfiability problems, in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE, 2017), pp. 598–504
20. X. Guo, R.G. Dutta, Y. Jin, F. Farahmandi, P. Mishra, Pre-silicon security verification and validation: a formal perspective, in *ACM/IEEE Design Automation Conference (DAC)*, 2015
21. X. Guo, R.G. Dutta, P. Mishra, Y. Jin, Scalable SoC trust verification using integrated theorem proving and model checking, in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (IEEE, 2016), pp. 124–129
22. S.R. Hasan, C.A. Kamhoua, K.A. Kwiat, L. Njilla, Translating circuit behavior manifestations of hardware Trojans using model checkers into run-time trojan detection monitors, in *IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)* (IEEE, 2016), pp. 1–6
23. Y. Huang, P. Mishra, Reliability and energy-aware cache reconfiguration for embedded systems, in *2016 17th International Symposium on Quality Electronic Design (ISQED)* (IEEE, 2016) pp. 313–318
24. Y. Huang, P. Mishra, Test generation for detection of malicious parametric variations, in *Hardware IP Security and Trust* (Springer International Publishing, Cham, 2017), pp. 325–340
25. Y. Huang, P. Mishra, Trace buffer attack on the AES cipher. J. Hardw. Syst. Secur. (HaSS) **1**(1), 68–84 (2017). Springer
26. Y. Huang, A. Chattopadhyay, P. Mishra, Trace buffer attack: security versus observability study in post-silicon debug, in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2015, pp. 355–360
27. Y. Huang, S. Bhunia, P. Mishra, MERS: statistical test generation for side-channel analysis based Trojan detection, in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)* (ACM, New York, 2016), pp. 130–141
28. Y. Jin, EDA tools trust evaluation through security property proofs, in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2014, pp. 1–4
29. Y. Jin, Y. Makris, Hardware Trojan detection using path delay fingerprint, in *IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 51–57
30. Y. Jin, Y. Makris, Proof carrying-based information flow tracking for data secrecy protection and hardware trust, in *VLSI Test Symposium (VTS)*, 2012, pp. 252–257
31. Y. Jin, B. Yang, Y. Makris, Cycle-accurate information assurance by proof-carrying based signal sensitivity tracing, in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 99–106
32. H.-M. Koo, P. Mishra, Test generation using sat-based bounded model checking for validation of pipelined processors, in *Proceedings of the 16th ACM Great Lakes Symposium on VLSI* (ACM, 2006), pp. 362–365
33. G.C. Necula, Proof-carrying code, in *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (ACM, 1997), pp. 106–119

34. X.T. Ngo, I. Exurville, S. Bhasin, J.L. Danger, S. Guilley, Z. Najm, J.B. Rigaud, B. Robisson, Hardware trojan detection by delay and electromagnetic measurements, in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 782–787
35. Y. Qiu, H. Li, T. Wang, B. Liu, Y. Gao, X. Li, Property coverage analysis based trustworthiness verification for potential threats from EDA tools, in *2016 IEEE 25th Asian Test Symposium (ATS)* (IEEE, 2016), pp. 43–48
36. J. Rajendran, V. Vedula, R. Karri, Detecting malicious modifications of data in third-party intellectual property cores, in *ACM/IEEE Design Automation Conference (DAC)*, 2015, pp. 112–118
37. J. Rajendran, A.M. Dhandayuthapany, V. Vedula, R. Karri, Formal security verification of third party intellectual property cores for information leakage, in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)* (IEEE, 2016), pp. 547–552
38. M. Rathmair, F. Schupfer, Hardware Trojan detection by specifying malicious circuit properties, in *2013 IEEE 4th International Conference on Electronics Information and Emergency Communication (ICEIEC)* (IEEE, 2013), pp. 317–320
39. E. Sadredini, M. Najafi, M. Fathy, Z. Navabi, BILBO-friendly hybrid BIST architecture with asymmetric polynomial reseeding, in *2012 16th CSI International Symposium on Computer Architecture and Digital Systems (CADS)* (IEEE, 2012), pp. 145–149
40. E. Sadredini, R. Rahimi, P. Foroutan, M. Fathy, Z. Navabi, An improved scheme for precomputed patterns in core-based SoC architecture, in *2016 IEEE East-West Design & Test Symposium (EWDTS)* (IEEE, 2016), pp. 1–6
41. S. Saha, R. Chakraborty, S. Nuthakki, Anshul, D. Mukhopadhyay, Improved test pattern generation for hardware Trojan detection using genetic algorithm and boolean satisfiability, in *Cryptographic Hardware and Embedded Systems (CHES)*, 2015, pp. 577–596
42. A. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, R. Drechsler, Formal verification of integer multipliers by combining gröbner basis with logic reduction, in *Design Automation and Test in Europe Conference(DATE)*, 2016, pp. 1–6
43. *Synopsys, Formality*, 2015 http://www.synopsys.com/Tools/Verification/FormalEquivalence/Pages/Formality.aspx
44. S. Vasudevan, E.A. Emerson, J.A. Abraham, Efficient model checking of hardware using conditioned slicing. *Electron. Notes Theor. Comput. Sci.* **128**(6), 279–294 (2005)
45. B. Yang, K. Wu, R. Karri, Scan based side channel attack on dedicated hardware implementations of data encryption standard, in *ITC*, 2004, pp. 339–344

Chapter 9
Golden-Free Trojan Detection

Azadeh Davoodi

9.1 Introduction

A vast number of existing Trojan detection techniques rely on a golden IC (GIC) during an IC authentication phase. A GIC is a fabricated instance of the design which is guaranteed to be free of any Trojans. During the authentication phase, the GIC is used to provide reference signals reflecting the correct chip behavior under a set of applied input patterns. These reference signals could be, for example, in the form of transient timing or power patterns. They will then be used to evaluate an IC under authentication. This is done by applying the same input patterns and comparing the signals generated by the IC under authentication with the ones obtained from GIC.

However, a fundamental limitation with the above detection approach is that a GIC may not even exist. For example, consider the following scenarios when a GIC may not exist: (a) the design team has a malicious member who inserts the Trojan during the design phase and the Trojan is not detected during the verification process, and (b) a design without Trojan is sent for fabrication at an untrusted foundry and an attacker inserts the Trojan during the fabrication phase by altering one of the masks. In both scenarios, all fabricated instances of the chip will be infected by Trojans, and thus a GIC won't exist.

Even when there are no reasons to suspect that a GIC may not exist, one needs to first identify a GIC before it can be used to compare against the IC under authentication. For example, this identification may be done by making a potential GIC candidate to go through an aggressive testing procedure – possibly far more aggressive than the one used for IC under authentication, for example, by applying

A. Davoodi (✉)
Department of Electrical and Computer Engineering, University of Wisconsin,
53706 Madison, WI, USA
e-mail: adavoodi@wisc.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_9

a significantly higher number of input patterns and verifying correct functionality under various environmental variations. However an aggressive testing procedure would still not be able to cover all possible workloads and environmental conditions; a Trojan may specifically be designed to only be triggered by a rare event, and even though that such a GIC candidate would be a more reliable chip, the aggressive test may not have included the rare event.

The above issues fundamentally limit the vast majority of Trojan detection techniques because the detection framework relies on having access to a GIC.

9.2 Golden-Free Trojan Detection and Its Challenges

The objectives of golden-free Trojan detection are similar to any other Trojan detection scheme: it needs to determine if an IC under authentication contains a Trojan. It can also point to the location of the Trojan on the layout and conditions (workload, process, and environmental factors) that triggers the Trojan.

Detection of a Trojan is a challenging task because of the combination of the following factors: (1) lack of observability and controllability in a fabricated chip; (2) complexity due to existence of billions of nanoscale components and due to the large number of soft, firm, and hard IP cores that may have been integrated in the system; (3) high cost and challenges associated with physical inspection of nanometer feature sizes for reverse engineering which may yet be intrusive and only applicable to a small portion of the chip population that are infected by a Trojan; (4) difficulty to activate a Trojan since it may only be triggered by rare events; and (5) increasing fabrication and environmental variations in nanometer technologies which cause deviation in the expected characteristics of an IC.

Golden-free detection is also subject to all the above challenges. In addition, lack of a GIC means that the detection scheme cannot assume existence of a set of reference patterns which can guarantee the existence of a Trojan. This assumption significantly limits the types of approaches that can be taken for golden-free detection. Next we give examples of golden-free detection techniques which have aimed to overcome these barriers.

9.3 Some Possible Solutions

In this section, we give an overview of different schemes that have been proposed for golden-free Trojan detection. Similar to golden-dependent techniques, some of these techniques also utilize on-chip sensors or side-channel information. However they do so without the need for a golden IC.

Detection with machine learning and imaging techniques: The work [1] proposes techniques using machine learning to reverse engineer a chip using layout images in order to identify a GIC. While this work aims to identify a GIC, the proposed

techniques are naturally suitable for golden-free Trojan detection. This work uses a combination of machine learning techniques in order to identify between "suspected" and "expected" structures in an IC. This requires detailed images from various layers of the layout in order to develop and train the models to identify these structures.

Detection with self-referencing analysis: Another approach is to use multiple measurements from the chip at different times in order to detect a Trojan. This self-referencing can be done using side-channel information to record temporal current signatures [7]. It is specific to identifying sequential Trojans; these Trojan may manifest over multiple time windows. A broader variation of this scheme also uses current measurements but at block level, within a chip, among blocks which are similar and ideally adjacent to each other [12]. While this approach requires knowledge of similarity within the chip, it is likely able to eliminate the impact of process-induced variations on the current measurements because similar blocks will be subject to similar variations, especially if they are physically close or adjacent to each other. Besides current measurements, self-referencing has also been implemented based on delay measurements on design paths with similar structures which also can help to eliminate the impact of process-induced delay variations [11].

Authentication with golden PCM using side-channel information: The work [6] proposes chip authentication using golden process control monitors (PCMs) rather than golden ICs. The PCMs are simple structures such as ring oscillators which are typically placed on the wafer. The authentication process uses measurements from PCMs to get an idea about process variations for the dies belonging to the same wafer which provides a more accurate idea about their expected post-fabrication characteristics. The PCMs typically won't be able to differentiate between dies within the same wafer. Nevertheless, they are standard structures which allow measuring the process characteristics at the wafer-to-wafer level fairly accurately. The authentication scheme utilizes statistical techniques applied to the collected side-channel information and is able to eliminate the impact of variations caused by the fabrication process which are obtained from the PCM measurements. Identification and existence of golden PCMs are less challenging tasks than golden IC. This is because it is less challenging for a process engineer to detect if a PCM is not behaving correctly, in case it may be Trojan-infected.

Sensor-assisted self-authentication with on-chip sensors: This detection scheme relies on on-chip detection sensors for self-authentication of each IC, thus alleviating the GIC requirement. Both delay and power sensors have been proposed in prior work [3, 4]. In [4], the detection sensors are design-dependent, as opposed to generic on-chip structures such as ring oscillators. As a result, they allow coverage of characteristics of the design, such as features of the netlist and dependencies due to the physical implementation on the layout. Specifically the detection sensors are designed to provide custom information for as many paths in the design's netlist as possible. Moreover, they are designed to capture individual process variations at a die-to-die level and are placed at various regions on the chip to also capture variations a within-die level. The sensor measurements should be made for individual chips but they allow making chip-specific delay-based calibration to more

effectively detect the impact of an inserted Trojan on the delay of the design. Next we will discuss the sensor-assisted self-authentication scheme in more detail.

9.4 Case Study: Sensor-Assisted Self-Authentication

9.4.1 Overview

Figures 9.1 and 9.2 give an overview of a sensor-assisted framework for self-authentication of a suspect IC. There are two stages in the framework: (1) design of so-called detection sensors and their layout integration which occur at the design stage (and during the implementation of the design using the CAD tools) and (2) self-authentication of a suspect IC using sensor-assisted analysis at the post-fabrication stage.

Self-authentication uses on-chip measurements from the detection sensors and from arbitrary design paths and involves a correlation analysis between the two measurements to determine if a Trojan exists in the suspect IC.

First, at the design stage, a set of detection sensors are identified and integrated with the layout. Each sensor is realized as a new logic structure and added to the layout. If the sensor reading is in the form of delay, then special delay measurement circuitry should also be added to each sensor to allow post-fabrication reading of the sensor delay.[1]

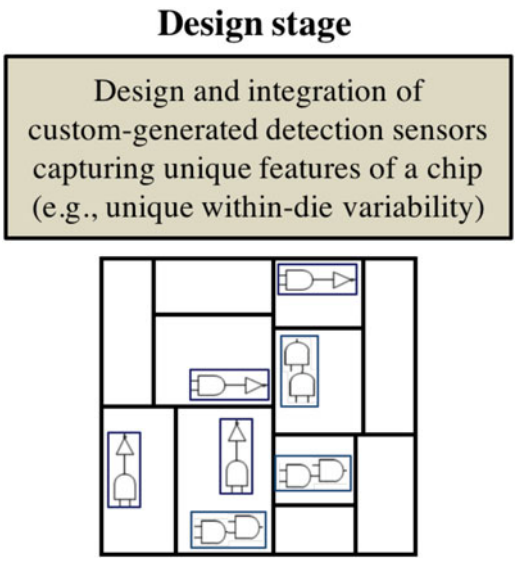


Fig. 9.1 In sensor-assisted self-authentication, detection sensors are created and integrated with the layout during the design stage. The sensors are custom-built to capture most commonly used logic structures in the design and capture within-die process variations for each region in the layout and help create a unique fingerprint of an IC

[1] Prior work such as [8] have already proposed solutions for designing on-chip delay measurement circuitry.

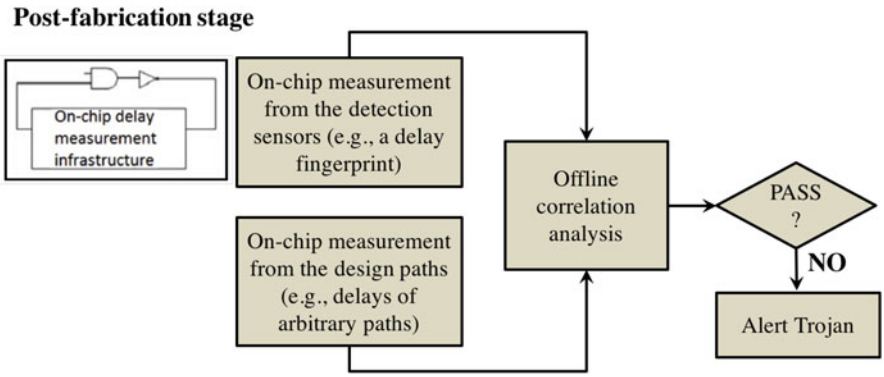


Fig. 9.2 At the post-fabrication stage, the measurements of the detection sensors are combined with measurements made to some design paths within a correlation analysis framework which alerts existence of a Trojan

There are various ways to design a delay sensor. A desirable goal during self-authentication is to capture *unique* characteristics of that design and that chip, so the reading from the delay sensor can serve as a source of delay "fingerprint" representing that chip. In [4], the delay sensor is designed by finding the most commonly used logic structure for different regions of the chip. For example, as shown in Fig. 9.1, the chip is divided into different regions. Each region has its unique within-die process variations. Then for each region of the layout, the design is analyzed to identify a logic structure (i.e., subset of connected gates) which is repeatedly found in that region.

This structure could be a sequence of logic gates in the most simple scenario. The sensor is then realized by inserting the commonly used logic structure in that particular region. This approach allows predicting the delays of those portions of the design which are included in that region while accounting for specific die-to-die and chip-to-chip variations as well as within-chip variations seen in different regions. The integration of the sensor in the existing layout should be done in the available white spaces as much as possible, so the original layout is not changed much. This will be a specific consideration in the sensor generation process.

Second, at the post-fabrication stage, the Trojan identification framework takes advantage of the delay fingerprints obtained from the detection sensors. For an IC under self-authentication, these delay fingerprints help estimate an expected delay range for various paths in the design because the design paths include the same common gate structures captured by the sensors and are subject to the same die-to-die and within-die process variations as the sensors.

This chip-specific path delay estimation using the sensors relies on having a model of logic and layout which has been verified to be accurate and Trojan-free, for example, one that is obtained from a trusted design environment but may be subject to an untrusted fabrication process. Besides obtaining estimates for some path delays, at the post-fabrication stage, the delays of these paths are also directly

measured.[2] If one of these paths contains a Trojan, a mismatch will be detected between the measured path delay and the estimated delay using the detection sensors. This mismatch alerts existence of the Trojan.

Given the above overview of a sensor-assisted self-authentication process, in the remainder of this chapter, we will discuss in more details how the delay sensors can be designed. Moreover, the above example discusses one scenario that a Trojan may be detected; we will discuss all possible scenarios to understand the detection scheme in more details. We will also discuss how a Trojan can be localized using the above framework. Finally, we will discuss some limitations and areas for improvement.

9.4.2 Sensors for Capturing Design-Dependent Delay Features

The work [5] gives a mathematical formulation which is able to generate custom sensors. These sensors depend both on logic synthesis of the design as well as its layout realization.[3] (The layout realization impacts the delay of a fabricated design because each chip has its own unique die-to-die and within-die process variations.) Given an arbitrary design, the objective of the optimization is to form groups of similar structures in order to maximize the coverage of the design paths. This is subject to area constraint for the sensor. Maximizing the design coverage helps with increasing the likelihood of Trojan detection while not making any assumptions about the potential locations in the netlist that the Trojan may be inserted.

More specifically, consider a model of a design described as a placed and routed netlist of connected gates and interconnects. First, a timing graph is extracted for this netlist which is a directed acyclic graph (DAG) denoted by $G = (\mathscr{V}, \mathscr{E})$. Each node corresponds to the output pin of a gate. An edge between two nodes represents the delay of the path between two gates, so it includes factors such as loading capacitance and interconnect delay on this path. Let a sequence be a directed path between two arbitrary nodes in this DAG. Therefore a sequence corresponds to a set of consecutively connected gates and interconnects.

Denote $\mathbf{X}$ to be a column vector representing the parameters which are subject to process variations. Each entry in $\mathbf{X}$ lies in the category of die-to-die, within-die, and random variations in physical parameters. All the entries in $\mathbf{X}$ are assumed to be independent from each other. This model of process variations is identical to

[2] One way to measure arbitrary path delays is using an automatic test equipment (ATE), by generating test patterns which excite the desired path(s), assuming they are testable, and then sweeping the clock period in a similar way until failure occurs.

[3] In [5], these sensors were proposed in the context of custom test structures to isolate the paths that fail their delay constraints during post-silicon validation. But they can also be used in the described self-authentication process.

the many works in statistical static timing analysis such as [2, 9, 10]. Define the variation-aware delay of edge e_i using the following linear model:

$$D_{e_i} = \mu_{e_i} + \mathbf{a}_{e_i}^T \mathbf{X} \tag{9.1}$$

where μ_{e_i} is the nominal delay of e_i and $\mathbf{a}_{e_i}$ is a sensitivity vector corresponding to $\mathbf{X}$. These sensitivities depend on the fabrication facility and technology and can be pre-characterized and communicated beforehand to the designer. The above model also includes interconnect delay variations.

Let a sequence s_i be a set of consecutive edges in G. The variation-aware delay expression of s_i is found by adding the corresponding delay expressions of the edges that are included in it. It is given by $D_{s_i} = \sum_{\forall j | e_j \in s_i} D_{e_j}$ which can be further reexpressed in a form similar to the above equation as follows: $D_{s_j} = \mu_{s_j} + \mathbf{a}_{s_j}^T \mathbf{X}$.

Using the above variation-aware delay model, the work [5] defines a mathematical optimization procedure which automatically *forms* a set of nonoverlapping sequences such that "similar" sequences are grouped together so they can be represented by one sensor. Two sequences s_i and s_j are similar if their sensitivities to the parameter variations vector $\mathbf{X}$ are less than a small error tolerance denoted by ϵ. This condition is given by the following inequality:

$$\left\| \mathbf{a}_{s_i}^T - \mathbf{a}_{s_j}^T \right\|_1 \leq \epsilon \tag{9.2}$$

The sensor is then implemented to be the smallest-area sequence that it represents in a neighborhood close to it in order to satisfy the above similarity constraint. If whitespace is available, then the addition of the sensor does not result in any additional area overhead. Otherwise, the sequences are formed such that the sensor area overheads do not exceed a specified threshold.

The objective of the optimization is to maximize the coverage of edges $e_i \in \mathscr{E}$. The mathematical optimization framework is described as an integer linear programming which can optimally solve the above problem. The reader is referred to [5] for the technical details.

9.4.3 Scenarios in Post-fabrication Self-Authentication

After the sensors are added to the die, Trojan detection happens during a self-authentication process at the post-fabrication stage. For each chip, an arbitrary set of paths are picked and their delays are measured together with the sensors to detect if a Trojan exists. This detection process is also able to account for inaccuracy in on-chip delay measurement (for both the paths and on-chip sensors). The duration of self-authentication depends on the number of selected paths; if a higher number of paths are picked, the authentication obviously takes longer since the analysis is made on a path-by-path basis. However, in this case, it will be more likely to detect

the Trojan. Specifically if none of the selected paths and the sensors are affected by the Trojan, the self-authentication process fails to generate an outcome in either positive or negative way.

We divide the discussion in three different cases, when the Trojan is inserted in the design paths, in the sensors, and in both design paths and the sensors. We discuss these details next.

9.4.3.1 Trojan Inserted in the Design Paths

Consider the case when the Trojan is inserted somewhere in the logic representing the design. (Note that this case excludes the situation when the Trojan is added in the layout in the existing white space because the logic of the design remains unchanged.) As a result, one or more design paths will have their delays affected due to the Trojan insertion. If the path that is affected by the Trojan is among the design paths that has been selected, then the sensors may be able to detect the Trojan.

This case is shown in Fig. 9.3 using a very simple example to illustrate the idea. The affected path is shown with a dashed arrow, and the Trojan is shown as an extra chain of inverters. The sensor is shown on the right corner and its matched sequences in the netlist are circled. Note that the sequences are similar according to Eq. (9.2) which means they may not necessarily be identical to each other.

To detect the Trojan, for each path, an actual and a predicted delay *range* is computed. First, the *actual delay range* of path p is computed by measuring its delay using an on-chip measurement infrastructure. Having a range and not an exact value is due to a measurement error associated with the used infrastructure.

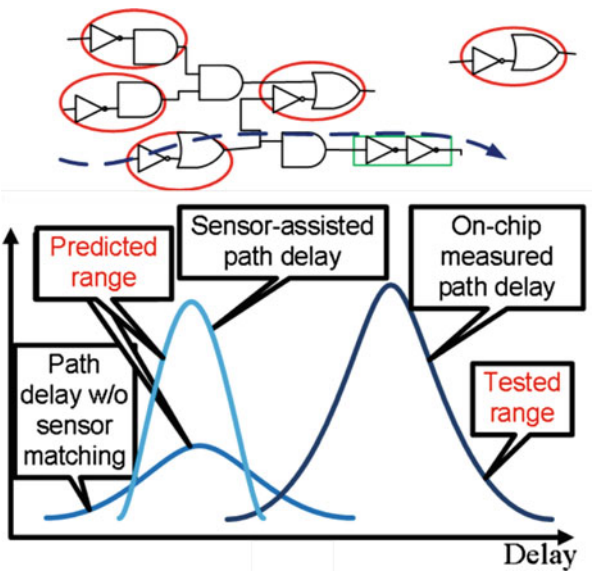


Fig. 9.3 Trojan detection when it is inserted in one or more design paths but the sensor is unaffected

Next, a sensor-assisted *predicted delay range* is found for path p using the sensors. Specifically, the on-chip delays are computed for the sensors which have partial similarity on the path. Recall, each sensor is formed to have the same change in its on-chip delay as the sequences that it represents. So any post-fabrication change in the delay of the sensor can be computed by comparing against its prefabricated delay model in order to predict the on-chip delay of the sequence that it matches on the path. Denote the aggregate delay of the matched sequences on one path by $\mu_{\text{match}} \pm \gamma$, where γ is the error associated with the on-chip measurement infrastructure.

These sensor delays will be used to predict the rest of the path delay. For path p, consider its remaining portion which is not matched by any sensor. It is denoted by p_{rmn} and identified by a set of edges on p which are not matched by any sensors during the sensor generation process.

Denote the delay corresponding to p_{rmn} by D_{rmn} which has a variation-aware delay expression obtained by adding the expressions of the edges that are included in it, similar to Eq. (9.1). Therefore the delay of the path is given by

$$D_{\text{rmn}} = \sum_{\forall i | e_i \in P_{\text{rmn}}} (\mu_{e_i} + a_{e_i}^T \mathbf{X}) = \mu_{\text{rmn}} + \mathbf{a}_{\text{rmn}}^T \mathbf{X}. \tag{9.3}$$

Next, worst-case and best-case values of D_{rmn} are computed which are denoted by $D_{\text{rmn}}^{(WC)}$ and $D_{\text{rmn}}^{(BC)}$. This is by assuming the varying parameters $x \in \mathbf{X}$ are simultaneously in their extreme values in the D_{rmn} expression. The predicted delay range of path p is then given by the following expression:

$$\mu_{\text{match}} + \mu_{\text{rmn}} \pm \left(\gamma + D_{\text{rmn}}^{(WC)} - D_{\text{rmn}}^{(BC)}\right). \tag{9.4}$$

Figure 9.3 shows the Trojan detection analysis in this first scenario when the Trojan is inserted in one or more design paths. On the left-hand side, two predicted delay ranges are drawn, corresponding to with and without sensor matching. The actual delay range always falls on the right-hand side of the predicted delay ranges. This is due to the insertion of the Trojan on the path which shifts its measured delay range to the higher values. The following analysis can be made:

- If a predicted delay range has no overlap with the actual delay range, then a Trojan is detected. Furthermore, the observation that the actual delay range is higher also prompts that the Trojan is inserted on the path which helps with localization of the Trojan.
- Recall the sensor-assisted prediction results in a smaller delay interval compared to the one without sensor matching. This reduction in interval increases the likelihood that the predicted and the actual delay ranges do not overlap if a Trojan is present.
- The decrease in the Trojan delay results in the predicted and actual delay ranges to be closer to each other and makes the detection more challenging.

Please note that, in the situation when the path does not have similarity with any of the sensors, the path delay range can only be determined based on worst case and best case of process variations and prefabrication delay models. In other words, we have $D_p = D_{\text{rmn}}$. In this case, the predicted path delay range is expressed as $\mu_p \pm \left(D_p^{(WC)} - D_p^{(BC)}\right)$. This delay range is larger than the sensor-assisted one since the partial matching of a path with a sensor helps reduce the path delay range to a smaller one.

For such paths, the same analysis can still be made. However, without the aid of the sensors, it is more likely that the (now) wider path delay range overlaps with its actual delay range, so the Trojan may not be detected. However, note that the objective of sensor generation process is to maximize the coverage to as many design paths as possible which should reduce the number of such cases.

Alternatively the set of paths which can get selected may be restricted based on information provided by sensor generation process. If most of the design paths have a match with the generated sensors, the information can be communicated by providing the set of paths which does not match a sensor, so they are excluded from sensor-assisted analysis.

9.4.3.2 Trojan Inserted in the Detection Sensors

Figure 9.4 shows this case. The insertion of the Trojan increases the measured on-chip sensor delay. Since the Trojan does not impact the path, the actual delay range of the path and the predicted one *without* sensors will both be accurate and computed

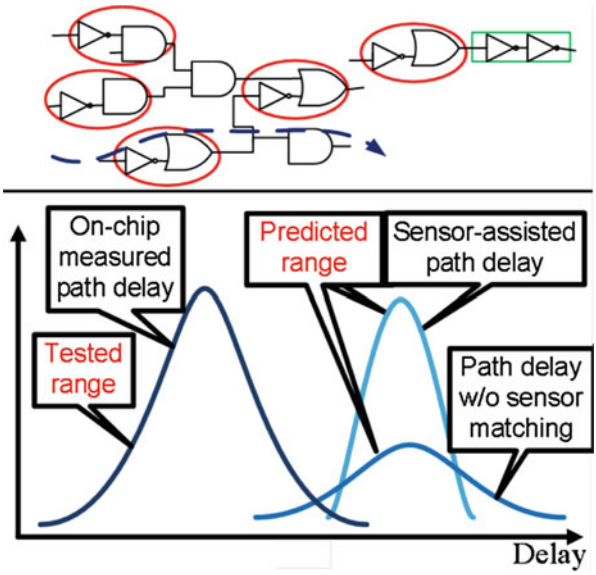


Fig. 9.4 Trojan detection when it is inserted in the sensor but not any of the design paths

similar to the previous case. These two ranges will overlap with each other. So by solely comparing those two ranges, the existence of the Trojan cannot be identified.

However, the sensor-assisted prediction of the path delay range will be inaccurate and shifted to the right-hand side of the actual delay range. This interval will likely not overlap the actual interval (obtained from on-chip measurement) which allows detection of the Trojan. If the predicted interval falls completely on the right-hand side of the actual, we conclude that the Trojan is inserted in the sensor.

For localization, the Trojan-affected sensor(s) will be one or more of the sensors which have matched with a sequence on the path.

9.4.3.3 Trojan Inserted in the Paths and Sensors

As shown in Fig. 9.5, the sensor-assisted predicted range and the actual one are more likely to overlap with each other in this case, because the insertion of the Trojan shifts both of the intervals to the right-hand side. The predicted delay without sensors may still overlap due to its larger interval. Recall that Trojan can only be detected when no overlap exists. Therefore in this scenario, it will be less likely for the Trojan to be detected.

We also note that in all the described three scenarios, if a path does not have any match with the sensor, then sensor-assisted authentication cannot be helpful to it. When this happens, the only way to detect the Trojan is if the predicted path delay range using prefabrication model is nonoverlapping with the actual path delay range using post-fabrication measurement. This may still be possible if the Trojan significantly impacts the path delay; the delay of Trojan-inserted path is

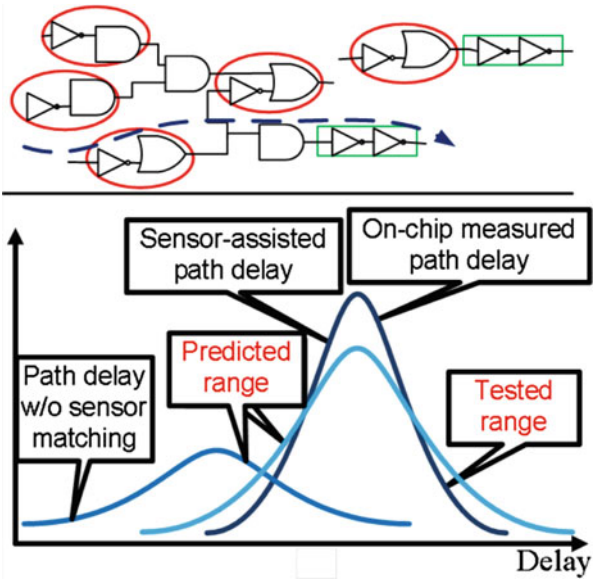


Fig. 9.5 Trojan detection when it is inserted in both the sensor and the design paths

higher than the worst-case prediction of path delay under process variations and using prefabrication models. Even though such a case self-authentication can be used without assistance of sensor, occurrence of this case (of having a Trojan with very significant impact on path delay) is even less likely.

9.5 Summary

In this chapter, we discussed the challenges associated with a golden-dependent Trojan detection which inspired switching to a golden-free IC authentication process in order to eliminate the golden IC from the authentication loop. We gave an overview of existing techniques for golden-free Trojan detection and then focused on one technique in detail, as a case study for self-authentication of a chip based on assistance from custom design-dependent on-chip sensors.

For the sensor-assisted self-authentication framework, we discussed three scenarios that can happen during a sensor-assisted self-authentication process when the sensor is inserted in the design paths, inserted in the sensors, and inserted in both of them. As far as the accuracy of the framework, if it had alerted existence of a Trojan, then its detection was accurate. Otherwise the framework would not generate an outcome to indicate that it was not able to determine if a Trojan existed. The framework also relied on testing an arbitrary set of design paths. These paths can be randomly selected, or they can be selected based on prior information about the design or if there are clues about areas on the chip that would be more important candidates for testing for Trojans. In general, the higher number of paths increased the likelihood of Trojan detection (if it existed) but required a longer testing time. The main idea behind sensor-assisted self-authentication was to use on-chip delay information from the sensors to make a more accurate prediction about the expected path delays.

References

1. C. Bao, D. Forte, A. Srivastava, On reverse engineering-based hardware Trojan detection. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **35**(1), 49–57 (2016)
2. D. Blaauw, K. Chopra, A. Srivastava, L. Scheffer, Statistical timing analysis: from basic principles to state of the art. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **27**(4), 589–607 (2008)
3. S. Kelly, X. Zhang, M. Tehranipoor, A. Ferraiuolo, Detecting hardware trojans using on-chip sensors in an ASIC design. J. Electron. Test. **31**(1), 11–26 (2015)
4. M. Li, A. Davoodi, M. Tehranipoor, A sensor-assisted self-authentication framework for hardware Trojan detection, in *Design, Automation & Test in Europe Conference* (2012), pp. 1331–1336
5. M. Li, A. Davoodi, L. Xie, Custom on-chip sensors for post-silicon failing path isolation in the presence of process variations, in *Design, Automation & Test in Europe Conference* (2012), pp. 1591–1596

6. Y. Liu, K. Huang, Y. Makris, Hardware Trojan detection through golden chip-free statistical side-channel fingerprinting, in *Design Automation Conference* (2014), pp. 155:1–155:6
7. S. Narasimhan, X. Wang, D. Du, R.S. Chakraborty, S. Bhunia, TeSR: a robust temporal self-referencing approach for hardware Trojan detection, in *IEEE International Symposium on Hardware-Oriented Security and Trust* (2011), pp. 71–74
8. X. Wang, M. Tehranipoor, R. Datta, Path-RO: a novel on-chip critical path delay measurement under process variations, in *International Conference on Computer-Aided Design* (2008), pp. 640–646
9. L. Xie, A. Davoodi, Bound-based statistically-critical path extraction under process variations. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **30**(1), 59–71 (2011)
10. L. Xie, A. Davoodi, J. Zhang, T.-H. Wu, Adjustment-based modeling for timing analysis under variability. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **28**(7), 1085–1095 (2009)
11. N. Yoshimizu, Hardware trojan detection by symmetry breaking in path delays, in *IEEE International Symposium on Hardware-Oriented Security and Trust* (2014), pp. 107–111
12. Y. Zheng, S. Yang, S. Bhunia, SeMIA: self-similarity-based IC integrity analysis. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **35**(1), 37–48 (2016)

Part IV
Detection: Side-Channel Analysis

Chapter 10
Detecting Hardware Trojans Using Delay Analysis

Jim Plusquellic and Fareena Saqib

10.1 Introduction

Hardware Trojans (HT) are a deliberate and malicious change to an IC that adds or removes functionality or reduces reliability of an integrated circuit (IC) or system [1–8]. The changes can be designed to leak secret information, e.g., encryption keys or other types of private internal information, or they may be designed to cause the system to fail at some specific or predetermined time while the IC is in mission mode. Adversaries design the HT to be difficult to discover, either accidentally via manufacturing test or purposely using tests specifically designed to activate the HT. Sophisticated HT insertion strategies also consider resilience to advanced HT detection methods that utilize high-resolution measurements of *side-channel* signals, such as electromagnetic (EM) emanations, power consumption (steady-state I_{DDQ} or transient I_{DDT}), delay testing, and temperature profiling.

In addition to these testing challenges, HT detection methods are further tasked to deal with several other fundamental HT properties. First, the task of identifying an HT is akin to finding a needle in a haystack, i.e., the adversary has a huge advantage because he/she can choose to insert the HT anywhere while the trusted authority is tasked with determining if the IC has in fact been modified and if so, finding the unknown malicious function in a "sea" of gates. Second, HT and "bugs," either hardware or software, share the same characteristics, and it is widely accepted that finding all the bugs in a complex program is generally infeasible [9]. In fact, cleverly inserted HT can be designed to appear as bugs, making it difficult to decide if the

J. Plusquellic (✉)
University of New Mexico, Albuquerque, NM, USA
e-mail: jimp@ece.unm.edu

F. Saqib (✉)
University of North Carolina Charlotte, Charlotte, NC, USA
e-mail: fsaqib@uncc.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_10

malicious function, if discovered, was accidental or purposeful. Third, any attempt by the trusted authority to increase the "ease" of HT detection may be visible to the adversary, i.e., the adversary can reverse engineer the IC and avoid countermeasures added by the trusted authority. Fourth, the adversary can choose to "selectively" insert the HT into only a subset of the manufactured ICs, making it necessary to verify all manufactured ICs. Last, HT designed to leak information may not cause a change in the functional behavior of the IC, and, therefore, the trusted authority may need to apply nonstandard tests, e.g., tests for anomalous EM radiations. Moreover, the appropriate detection strategy will vary greatly depending on the assumptions made regarding the "insertion point," i.e., design-inserted HT requires very different detection techniques than those inserted into a layout description of the design.

The only advantage afforded to the trusted authority is that his/her detection strategy can be "parallelized" because the HT needs to be detected only once and is, in most cases because of mask cost issues, inserted in the same fashion in every copy of the targeted IC population. Therefore, tests applied post-manufacturing can be partitioned among multiple independent IC testers (referred to as automatic test equipment or ATE) and applied in parallel. Unfortunately, even high levels of parallelism "run out of gas" when the full extent of the search space, both combinational and sequential, is considered.

This chapter is focused on surveying methods that utilize very precise analog-based testing to discover HT. The underlying basis of these methods can be characterized by the Heisenberg principle or *observer effect*, i.e., any attempt to measure or monitor a system changes its behavior. The testing methods described herein attempt to determine if an adversary has inserted an HT that is "observing" the evolving state of the IC, which is used by the adversary as the mechanism to activate the HT. In particular, we survey path delay-based testing methods which are designed to detect subtle changes in delay introduced by the HT connections and gate insertions, referred to as the trigger and payload of the HT, respectively. The authors of [10] propose a generic characterization of these concepts as shown in Fig. 10.1.

The rest of this chapter is organized as follows. Section 2 presents a high-level view of HT insertion strategies and discusses the constraints on the detection methods. Section 3 covers HT detection strategies designed to detect layout or GDSII Trojans (other HT insertion points are detailed in other chapters of this book) with subsections that survey detection methods that analyze "side-channel" signals, e.g., power and delay. Section 4 describes important fundamental concepts related to implementing path delay-based HT methods. Section 5 provides a survey of delay-based HT detection techniques, while Section 6 describes the first proposed multiple-parameter side-channel method. Conclusions are provided in Sect. 7.

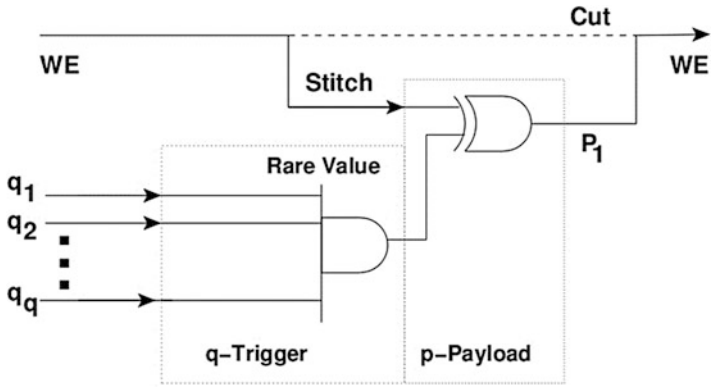


Fig. 10.1 Generic characterization of a hardware Trojan trigger and payload from [10]. Trigger signals q_1 through q_q typically connect to nodes in the existing design and therefore add capacitive load to these signals, creating an *observer effect*. Both the trigger signals and payload add delay to paths in the existing design

10.2 Hardware Trojan Insertion Points

The horizontal dissemination of the IC design, fabrication, and test processes to many distinct companies around the world has dramatically increased the potential for malicious activities. Intellectual property (IP) block reuse has compounded this threat by partitioning the design space itself among multiple third-party vendors. Standardization activities have enabled multiple independently designed IP blocks to seamlessly integrate into CAD tool flows. However, the electronic design automation (EDA) community developed this multiparty collaborative design system using a model in which all parties are largely trusted. Unfortunately, the same types of malicious activities endured by the software community are now presenting themselves in the hardware design community.

All of the primary processes associated with design, manufacture, and test are vulnerable to malicious activities where adversaries can add to, remove from, or change the functionality of the IC. We refer to these opportunities as *insertion points*. Figure 10.2 provides a graphical illustration of the major insertion points, which are further distinguished by the following list:

- Designing third-party IP blocks
- Developing CAD tool scripts
- Integration activities where IP blocks and glue logic are assembled into system-on-chip (SoC) ICs
- Behavior synthesis and place and route (PnR) carried out by CAD tools
- Layout mask data generation and mask preparation
- Process parameter control mechanism used in the multistep fabrication process
- Supply chain transactions associated with transferring wafers from one facility to another

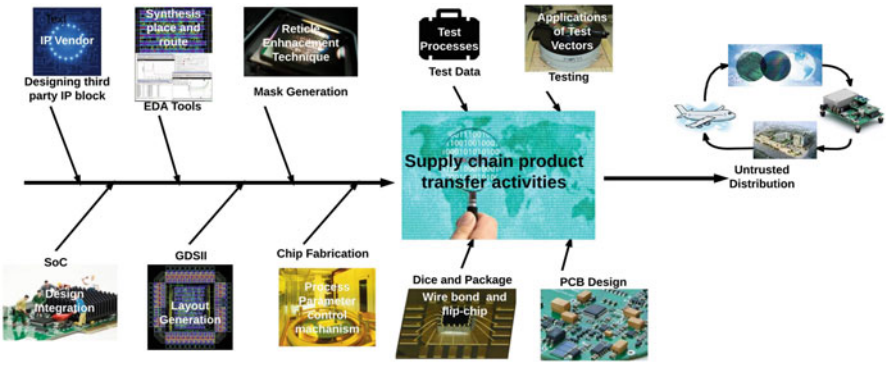


Fig. 10.2 Hardware Trojan insertion points

- Generating test vectors using automatic test pattern generation (ATPG)
- Wafer-probing activities associated with measuring test structures and detecting defects
- Supply chain transactions associated with creating and transferring dice
- Processes responsible for packaging ICs
- Applying ATPG vectors to packaged ICs using ATE
- Supply chain transactions associated with transferring packaged parts
- Printed circuit board (PCB) design and fabrication
- Processes responsible for installing PCB components (populating PCBs)
- Supply chain transactions associated with transferring boards
- System integration and deployment activities

The wide range and widely distributed nature of these activities presents an overwhelming opportunity for subversion. Moreover, the wide diversity among the tasks will require a very sophisticated and complex system to manage the entire set of trust vulnerabilities from start to finish. The research community is tackling the trust challenges one at a time and is focused on those that are the most attractive insertion points for adversaries. For example, subversion of IP blocks is a serious concern given the ease in which malicious functionalities can be covertly inserted and the absence of alternate representations and models to which the IPs can be compared [11]. Layout modifications and IC fabrication insertion points represent another important focus area, especially given the huge complexity associated with analyzing fabricated ICs at this lowest layer of design abstraction, and the wide range of opportunities available to the adversary in designing HT with sophisticated, sometime analog, triggering and payload mechanisms.

Note that significant differences exist in the HT countermeasures and detection strategies that are applicable even when only considering these two insertion points. For example, golden models are not available at the IP block insertion point, but architectural changes that obfuscate the design are available as countermeasures. On the other hand, the layout insertion point allows layout design data to be used

to validate the functional and analog behaviors of the IC, but obfuscation is limited to "dummy via" insertion and other nano-level manipulations of the design. Also, side-channel information is not available or is not accurate enough to be useful for IP blocks but can be leveraged as a very powerful HT detection method for layout-level validation. The focus of this chapter is on HT detection methods, and countermeasures where appropriate, that are applicable at the layout level. Other chapters of this text survey techniques which target other insertion points.

10.3 Approaches to Detecting Layout-Inserted Hardware Trojans

A layout is a physical representation of the design, i.e., it is a set of geometric shapes that represent a physical model of the IC. The shapes define transistors, wires, vias, and contacts. A layout is the lowest layer of abstraction in the design process and contains all the logic gates that define the function as well as all of the electrical connections between the logic gates and the power supply rails. The complexity of layouts increases as technology feature sizes shrink into the nanometer regime, and additional wiring layers are added. Figure 10.3 shows several standard (std.) cell layouts on the left and a tool-synthesized layout of a relatively small functional unit called the Advanced Encryption Standard (AES). The layout of the AES IP block contains approx. 12,000 std. cells and 50,000 wires and typically would represent one IP block of several 100 on a modern SoC. The technology used in this example is an IBM 90 nm process which provides nine vertically stacked layers for metal wires. The image is a screen capture of the designer's view of the layout using Cadence Virtuoso. Most layout design tools provide this type of top-down view, with upper metal layers obscuring the transistors in the bottom-most layer, i.e., nearly all of what is shown in the AES layout are metal wires.

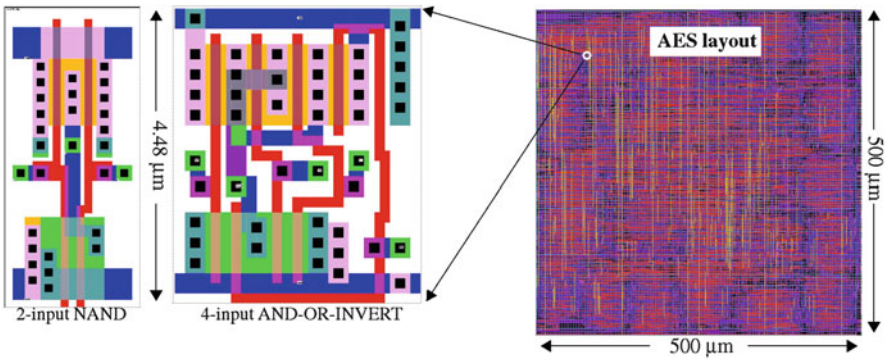


Fig. 10.3 Layout of std. cells (*left*) and AES layout (*right*)

Once the physical model of the layout is completed as shown by the AES layout in Fig. 10.3, a set of masks are generated. The masks decompose the layout into a set of (x,y) planes, which can be vertically aligned to define the transistors and wiring layers. Layout-inserted HT are characterized as changes in one or more of the masks used in photolithography process to create physical instances of the IC. The multitude of overlapping wires and the tightly packed form of the transistors define a complex structure which represents the haystack in the "needle-in-a-haystack" paradigm. Adversaries are free to add or change very small regions in the masks, which can affect connectivity relationships between a small set of existing std. cells, or new std. cells can be added. The latter is possible using "white space," i.e., areas in the lowest layers of the layout that contain nonfunctional filler cells or cells implementing decoupling capacitors.

10.3.1 Layout-Oriented HT Detection Methods

HT detection methods which are designed to detect malicious modifications to the IC layout fall into three fundamental categories [1]:

- Nondestructive logic-based testing methods
- Nondestructive side-channel-based testing methods
- Destructive physical inspection techniques

10.3.1.1 Nondestructive Logic-Based HT Detection Methods

Logic-based methods derive test vectors that attempt to activate the HT [12–18]. Unlike manufacturing tests which activate and propagate faults on each node individually within the fabricated IC, HT activation is akin to multiple fault activation, which is rarely practiced in manufacturing test because of the high time complexity for ATPG and high cost of applying very large numbers of vectors. Also, unlike manufacturing defects which tend to distribute randomly across circuit nodes, the adversary chooses a stealthy location for the HT, i.e., he/she inserts the HT on circuit nodes that are difficult to control or observe. Unfortunately, the task of generating test vectors that provide coverage of all possible states for these nodes is orders of magnitude more difficult than it is for manufacturing defects, and, therefore, achieving high levels of HT coverage is difficult or impossible given limited resources and existing manufacturing test cost constraints. The authors of [12–18] present alternative test generation strategies that are optimized to deal with these challenges, either alone or in combination with design modifications and side-channel-based testing approaches, as detailed in other chapters of this text.

10.3.1.2 Side-Channel Analysis Approaches

Side channels refer to access and measurement techniques that bypass the designer-intended input-output mechanisms, e.g., the digital I/O pins of an IC. Side channels, as the name implies, refer to auxiliary electrical and/or electromagnetic (EM) access mechanisms, such as the V_{DD} and GND (power supply) pins or the top-layer metal connections in the physical layout of the IC. Side-channel attacks utilize these auxiliary electrical paths to introduce signals, usually in an attempt to create a fault while the IC is operational [19], or to measure signals, in an attempt to extract private internal information [20].

Side channels can also be leveraged by the trusted authority to obtain information regarding the integrity of the IC. For example, leakage current (I_{DDQ}) and transient current (I_{DDT}) measurements have been widely used to detect manufacturing defects [21–23]. Moreover, the trusted authority can also introduce on-chip design for testability (DFT) [24] and other types of specialized instruments [25, 26] which allow access to additional side channels that are not directly accessible using auxiliary channels to the IC. DFT components are designed to improve visibility of the internal and localized behavior of the IC and include mechanisms to measure path delays, localized transients, and temperature profiles. DFT added by the trusted authority can also be leveraged by adversaries as "backdoor" access mechanisms to internal secrets, e.g., encryption keys, so security features such as fuses must be included to disable DFT after the IC is fabricated.

Side-channel signals are typically *analog* in nature and can provide detailed, high-resolution information about the internal timing and regional signal behavior of the IC. For example, I_{DDT} measurements reflect performance characteristics of individual gates as logic signals propagate along one or more paths in the circuit. This type of temporal information can be reverse engineered and compared with simulation-generated data to validate the structural characteristics of the fabricated layout, i.e., to ensure the chip is consistent with the *golden model* representation described by the design data.

Path delay measurements, if measured at high resolutions, can also serve this role. Unlike I_{DDx} measurements which provide a large-area regional observation, path delays are influenced by only those components that interact with the wires and gates along the *sensitized* path (defined as a path that propagates a logic signal transition). Therefore, path delay measurements can potentially be used to define a high-resolution HT detection methodology. Unfortunately, path delays are also affected by variations which occur in fabrication processing conditions, commonly referred to as *process variations*. Path delay variations caused by process variation effects are unavoidable and must be distinguished from delay variations introduced by an HT. Failure to do so is costly both in terms of the time and effort involved in verifying false alarms and, worse, in HT escapes that leave fielded systems vulnerable to attack. Subsequent sections of this chapter investigate both the benefits and challenges of using path delays as a HT detection method.

10.3.1.3 Destructive Physical Inspection-Oriented Methods

A third tactic to determining whether a chip is free of malicious inclusions is to apply destructive delayering and imaging techniques. Companies such as TechInsights [27] and Analytical Solutions [28] provide services that reverse engineer the physical characteristics of a chip to design data such as a schematic, which can then be inspected to identify IP infringements or HT circuitry. Failure analysis techniques, including scanning optical microscopy (SOM), scanning electron microscopy (SEM), picosecond imaging circuit analysis (PICA), voltage contrast imaging (VCI), light-induced voltage alteration (LIVA), charge-induced voltage alteration (CIVA), are used as needed in the reverse-engineering process [1, 29]. The primary disadvantage of these methods is their high cost and long processing times. Moreover, many destroy the chip and, therefore, cannot be used to validate chips for field use.

10.4 Fundamentals of Delay-Based HT Detection Methods

This section introduces the three fundamental technical domains that need to be considered by path delay-based methodologies: (1) the test vector generation strategy, (2) the technique employed for measuring path delays, and (3) the statistical detection method for distinguishing between process variation effects and HT anomalies. A commercially viable HT detection method must address each of these in a cost-effective manner. We investigate the challenges associated with each of these domains and describe proposed solutions in this section. Many of the methods surveyed in Sect. 5 address only a subset of these technical domains and therefore must be combined with other techniques to be fully operational in practice.

10.4.1 Path Delay Measurement Schemes and Other Concepts

When technology scaling entered the deep submicron era circa 2000, higher frequency operation, within-die variations, coupling, modeling challenges, and power supply noise drove the IC design and test community to more sophisticated statistical modeling approaches for IC development and test [30, 31]. This era also renewed interest in delay fault models [32], namely, transition fault, gate delay fault, and path delay fault models, which were introduced earlier in previous works [33–36]. Although it became apparent that delay fault testing was needed to keep defect levels low, work-arounds were developed to allow the two-vector sequences which define a delay fault test (described below) to be applied. The work-arounds became known as *launch-on-shift* (LOS) and *launch-on-capture* (LOC). LOS and LOC allow two-vector delay tests to be applied while minimizing the amount of additional on-chip logic needed to support this type of manufacturing test.

Unfortunately, LOS and LOC delay test mechanisms also create constraints on the form of the two-vector sequences, i.e., they do not allow the two vectors that define a sequence to be independently specified. These constraints reduce the level of fault coverage that can be attained for delay defects. More elaborate design-for-testability (DFT) structures that do allow both vectors of the sequence to be independently specified have been proposed [24], but are difficult to justify because of their negative impact on area and performance, and the fact that they would only be used during manufacturing test. These constraints continue to hold for modern-day SoCs. However, increasing awareness of hardware trust concerns may provide the impetus for a paradigm shift which would justify additional on chip support, particularly given the significant security and trust benefits associated with path delay testing, as we discuss in the following.

10.4.1.1 Path Delay Testing Defined

Path delay tests are defined as a two-vector sequence $<V_1,V_2>$, with the initialization vector V_1 applied to the inputs of a circuit at time t_0. The circuit is allowed to stabilize under V_1. At time t_1, vector V_2 is applied, and the outputs are *sampled* at time t_2. The *Clk* signal is used to drive both the launch flip flops (FFs), which apply V_1 and V_2 to the combinational block inputs, and the capture FFs which sample the new functional values produced by V_2. The time interval $(t_2 - t_1)$ is referred to as the *launch-capture interval* (LCI) and is typically set to the operational clock period for the chip. Figure 10.4 shows the standard form of a path delay test. Note that the standard form places no constraints on the values used for V_1 and V_2.

Unfortunately, external, off-chip access to the launch and capture FFs which connect to the combinational blocks within an IC is not possible. Figure 10.5 shows a typical configuration with several cascaded combinational blocks B_1 and

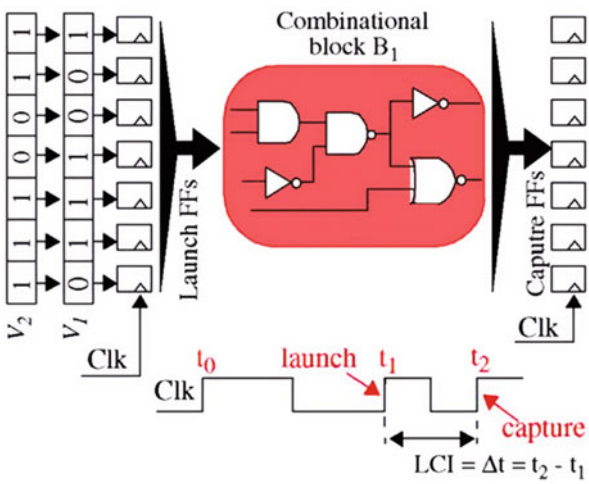


Fig. 10.4 Standard form of path delay test

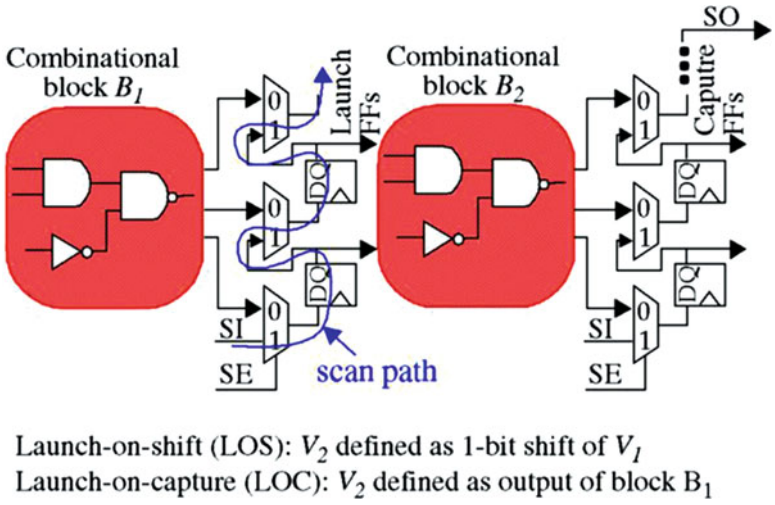


Fig. 10.5 Actual form uses scan flip-flops

B_2, with interleaved FFs. The manufacturing test community introduced a design-for-testability (DFT) feature called **scan** to address the difficulty of applying manufacturing tests to embedded combinational blocks [24]. Scan provides a second, serial path through all (or most) of the FFs in the IC. The second path is commonly implemented by adding a 2-to-1 MUX before the input of every FF (as shown in the figure). A *scan-enable* (SE) control signal is added as an I/O pin on the chip to allow test engineers to enable the serial path and to shift in a sequence of 0s and 1s using a second I/O pin referred to as *scan-in* (SI). The scan path allows the internal FFs to be configured with test data that is designed to maximize fault coverage. Once a test vector is scanned into the chip, *Clk* is used to apply a launch-capture test, which captures the functional outputs of the blocks B_i in the capture FFs. A second scan operation allows those values to be read out using a third I/O pin called *scan-out* (SO).

The scan architecture shown in Fig. 10.5 allows only a single vector V_1 to be applied. Manufacturing tests that target defects which prevent circuit nodes from switching (called stuck-at faults) can be applied directly using scan because the process involves applying a fixed set of values to the combinational block inputs (represented by V_1) and determining if the outputs possess the correct functional values. Stuck-at fault testing is referred to as a DC test because no timing requirements exist, i.e., delays are irrelevant. As discussed at the beginning of this section, the deep submicron era brought with it more occurrences of timing-related failures and the need to apply delay tests. The two-vector requirement for delay testing can be solved in two ways as discussed earlier. Launch-on-shift (LOS) derives V_2 by shifting the scanned in vector V_1 by 1 bit position using the scan chain. Launch-on-capture (LOC) derives V_2 from the outputs of the previous combinational block, shown as B_1 in Fig. 10.5 for testing paths in B_2. In both cases,

it is not possible to choose V_2 arbitrarily as shown for the standard form in Fig. 10.4. It is important to recognize that these constraints exist (they are often ignored) and that the effectiveness of deriving delay tests for detecting HT will be negatively impacted by them.

Another issue that is often ignored deals with obtaining accurate timing information for paths. The timing diagram shown in Fig. 10.4 shows that it should be possible to set the **launch-capture interval (LCI)**, i.e., the interval of time between the application of V_2 at t_1 and the capture event at t_2, to any arbitrary value. Unfortunately, this is not the case. The external tester (ATE) driving the clock pin on the chip is limited in how close consecutive edges of *Clk* can be placed. Moreover, most applications of delay tests for manufacturing defects only need to determine if the chip runs at the operational clock frequency. As a consequence, the LCI is typically fixed for all tests, and only upper bounds on the delays of paths within the chip can be obtained. Therefore, HT detection methods that require picosecond resolutions for individual path delays will require alternative clocking strategies and/or additional DFT components to be incorporated on the IC, which are described below.

A last important issue regarding path delay testing is related to circuit hazards. Combinational logic blocks often possess instances of *reconvergent fanout*. A simple example is shown in Fig. 10.6a for a NAND gate implementation of the XOR function. The integers inside the NAND gates represent one possible assignment of gate delays. The test sequence $AB = \{01,11\}$ is designed to test the highlighted path but in fact propagates logic transitions along both branches of the *fanout* point *C*. The timing diagram shown on the right side of Fig. 10.6b identifies a "glitch" on the output *F* that is created by differences in the relative delays of these two paths.

Although this test is classified by the manufacturing test community as *robust*, the glitch introduces uncertainty for the security community in cases where the precise delay of the highlighted path is needed. The three transitions that occur on *F* each represent the delay of a subpath in the circuit, with the first, leftmost edge in this case corresponding to the highlighted path. Although subpath information might prove useful in providing additional HT coverage, process variations render this information challenging to leverage because of the difficulty associated with deciding which edge corresponds to which subpath. In other words, the same test

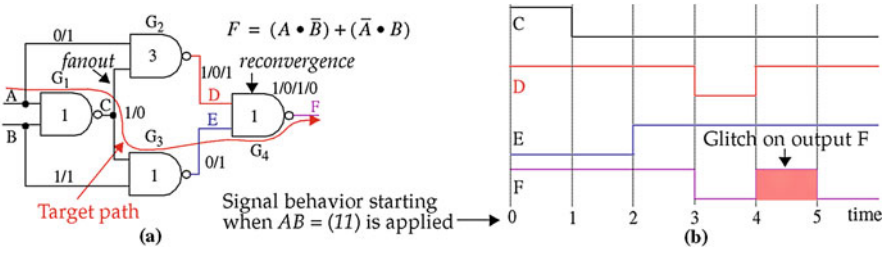


Fig. 10.6 (**a**) Reconvergent fanout and (**b**) circuit hazards

applied to a different chip with different assignments of delays to the NAND gates may reorder the edges or may in fact result in only single transition, i.e., the glitch disappears altogether. All major synthesis tools are oblivious to hazards, making them very common in synthesized implementations of functional units. Special logic synthesis algorithms are needed to construct circuits that are hazard-free, but hazard-free implementations usually have large area overheads and therefore are rarely used. Unfortunately, hazards are largely ignored in many proposed HT test generation strategies even though they can invalidate tests and raise false alarms.

10.4.1.2 Similarities and Distinctions of Delay Test for Manufacturing Defects and HT

Unlike logic-based testing, the goals of testing for defects and testing for HT using path delay tests are very similar. Path delay tests for defects are designed to determine if an imperfection introduced during the fabrication process causes a signal propagating along a path to emerge later than designed. Similarly, path delay tests for HT are designed to determine if an adversary has added fanout to logic gate inputs and outputs, i.e., additional wires that monitor the state of the IC (**trigger**) or inserted additional gates in series with the original design as a means of modifying its function (**payload**). Both of these scenarios also cause the delay of paths to increase.

The main distinguishing characteristic between defects and HT relates to *false positives*. False positives are situations in which a test for an HT indicates it is present when in fact it is not. This issue is less relevant for defects and can be minimized using modern ATPG tool flows. False positives can occur for HT when the detection method does not adequately account for normal delay variations introduced by process variations. The cost associated with false-positive detection decisions is very different for defects and HT. A false positive in manufacturing test results in a defect-free chip is being falsely discarded, while a false-positive HT detection can initiate a very expensive and time-consuming reverse-engineering process of the IC.

False negatives, on the other hand, need to be handled by both manufacturing defect and HT testing communities. False negatives are situations in which a defect or HT exists, and it is not detected by the applied tests. False negatives can occur in either application either because the measurement technique does not provide sufficient resolution or because the applied tests do not provide adequate coverage. The cost associated with false negatives can be high in either case, resulting in system failure once the IC is installed in a customer application.

10.4.1.3 High-Resolution Path Delay Measurement Techniques

Delay-locked loop (DLL), phase-locked loops (PLLs), and digital clock managers (DCM) are on-chip IP blocks responsible for maintaining phase alignment with

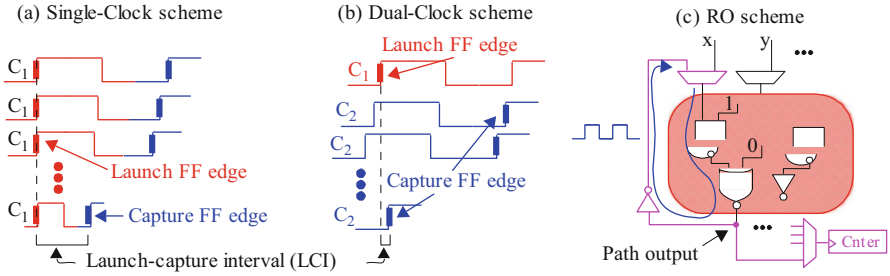


Fig. 10.7 Path delay measurement techniques

external oscillators and for creating multiple internal clocks at different frequencies and with specific phase shifts.[1] They can be used to create the *Clk* signal shown in Fig. 10.4 for path delay testing. Although automatic test equipment (ATE) can be used to carry out path delay testing, on-chip clock and phase shift mechanisms generally provide higher accuracy and resolution because off-chip parasitic resistor-inductor-capacitor (RLC) components are eliminated and noise sources are reduced. Many of the HT detection techniques described in subsequent sections depend on high-resolution timing measurements, making on-chip techniques better suited.

Figure 10.7 shows three examples of measurement techniques that can be used to provide fine-grained timing resolution. The first, called *single-clock scheme* (or *clock sweeping*), requires repeated application of a two-vector sequence (Fig. 10.7a). On each iteration, the *frequency* of C_1 is increased, which moves the launch and capture edges, i.e., the LCI, of the *Clk* signal closer together. The process is halted as soon as a condition is met or violated, which is usually related to whether the capture FF successfully captures the functional value produced by vector V_2 (see Fig. 10.4). An estimate of the path delay is computed as $1/frequency_{final}$ where $frequency_{final}$ is the stop point frequency. Although this scheme requires the fewest resources, i.e., only one clock tree is included on the chip, it lower bounds the length of the path that can be measured. For example, short paths would require a very high-frequency clock, which creates undesirable secondary effects, e.g., power supply noise, that make it difficult to obtain accurate timing measurements. Single-clock schemes which use an externally generated (ATE) clock constrain the minimum path length even further.

The second, called *dual-clock* scheme (or *clock strobing*), also requires repeated application of the two-vector sequence [37, 38]. On each iteration, the *phase* of the capture clock C_2 is decremented by a small Δt relative to C_1 as shown in Fig. 10.7b. The additional overhead introduced by the second clock tree is offset by the benefit of being able to time a path of any length. This is possible because the two clock networks are independent and modern clock manager IP designs are capable of allowing the time base of C_2 to be very precisely shifted. Moreover, the

[1] FPGA vendors commonly refer to IP blocks for clock control as DCMs.

power supply noise issue mentioned above is also mitigated because only two clock edges are required to carry out a launch-capture delay test instead of three.

The third timing mechanism, referred to as the *RO* scheme, is shown in Fig. 10.7c [44].[2] It adds the components shown in magenta to the design. Paths in the circuit are timed by creating a ring oscillator (RO) configuration where the output of a path is connected back to the input of the path using a MUX (and optionally a NOT gate as shown). A timing measurement is performed by enabling the MUX connection and then allowing the path to "ring" for a specific time interval. A counter (*Cnter*) is used to record the number of oscillations. This is accomplished by tieing the output signal from the path to the clock input of the counter. The actual path delay is obtained by dividing the time interval by the counter value (note, the NOT gate and MUX add two gate delays to the delay of the actual path). No launch-capture event is required in this scheme. Therefore, the clock noise associated with high-frequency clocks in the *single-clock* scheme is eliminated. The main drawback is related to the limited number of paths that can be timed in this fashion. For example, paths that have hazards as discussed in reference to Fig. 10.6 produce artifacts in the count values. As discussed, hazards are very common in combinational logic circuits, and therefore, they will negatively impact HT coverage.

A fourth alternative, called a time-to-digital converter (TDC), is shown in Fig. 10.8 [25, 26, 59]. Similar to the *RO* scheme, it eliminates clock strobing and, therefore, is able to obtain path delay measurements that better represent *mission mode* path delays. The TDC is an example of a "flash" converter, which is a class of converters that digitize path delays very quickly. The *path select unit* shown on the left is responsible for selecting a pair of paths, one of which can be the clock signal. The *delay chain unit* shown on the right is responsible for creating a digital representation of the relative difference between the delays of the two input paths, P_{Ax} and P_{Bx}. The arrival of a rising or falling transition on one path creates the first edge in the delay chain (labeled *first* in the figure), while a transition on the second

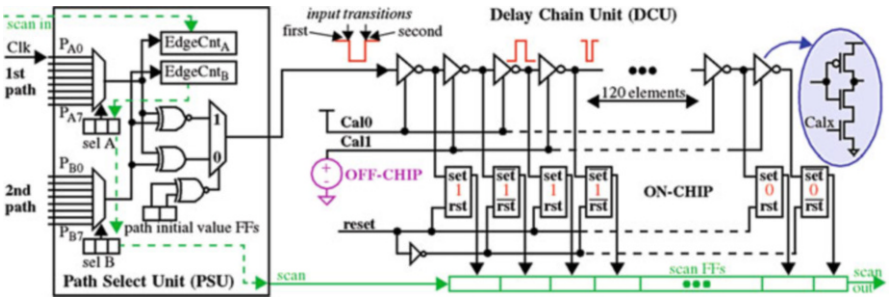


Fig. 10.8 Time-to-digital converter (TDC)

[2] A similar scheme called Path RO is proposed earlier by the authors of [39] but for application to design for manufacturability.

path generates the trailing edge (labeled *second*). The width of the initial (leftmost) pulse shown in red represents the delay difference between the two signals being timed. The pulse propagates along the delay chain as shown by the annotations along the top of the figure.

The inverters in the delay chain include an additional series-inserted *NFET* transistor as shown by the callout on the far right. An analog control signal labeled *Calx* is used to control the pull-down strength of the inverter, with higher gate voltages allowing faster operation. The inverter chain is configured with two such control signals, *Cal0* and *Cal1*. The combination of the two allows independent control over the propagation speed of the *first* and *second* edges. A calibration process is carried out in advance of making delay measurements to determine the best values of *Cal0* and *Cal1*. These analog control signals are set to allow the worst-case (widest) pulse to propagate through most of the inverters before "disappearing." The pulse disappears when the second edge catches up to the first edge. The calibration process is described later in Sect. 5.11.

The output of the inverters in the delay chain also each connects to a "set-reset" latch. The presence of a negative pulse (for odd inverters) or positive pulse (for even inverters) changes the latch value from 0 to 1. A digital *thermometer code* (TC), i.e., a sequence of 1s followed by zero or more 0s, is produced in the sequence of latches after a test completes. The TC can be converted into a discretized delay value (if needed) using pulse width information applied during the calibration process. In addition to being very fast (less than 100 ns per measurement), the TDC is also resilient to some types of circuit hazards. For example, a series of pulses can be introduced by circuit hazards but only the widest one determines the TC value (shorter pulses die out earlier in the delay chain). The *EdgeCnt* components in the *path select unit* can be used to decide when hazards are present.

A fifth scheme, called REBEL in [26], also uses a delay chain to obtain timing information. REBEL is a light-weight *embedded test structure* that combines the delay chain component of the TDC (without the pulse shrinking characteristic) with the *clock strobing* technique referred to in Fig. 10.7. A significant benefit of REBEL over the TDC is complete resilience to circuit hazards. In fact, REBEL is able to provide timing information regarding each of the edges associated with hazards in a single launch-capture test. As indicated earlier, the edges produced by circuit hazards each represent the delay of some internal segment in the functional unit. Although process variations add uncertainty and diminish their usefulness, as discussed above, the ability to instantly have knowledge of their presence adds robustness to the delay measurement process and can help reduce the likelihood of false-negative HT detection decisions.

10.4.2 Dealing with Process Variations

A significant benefit of techniques designed to detect HT in fabricated chips is the availability of a *golden model* of the IC. The assumption made by most of the

techniques described in Sect. 5 is that the HT is introduced by changing the layout, via mask manipulation or through other fabrication process-related steps. Therefore, all design data prior to the mask and chip fabrication steps, e.g., HDL, schematic, and even the geometric layout data itself, are considered trusted. A golden model, and simulation data derived from it, provides a trusted reference to which hardware data can be compared. Path delay methods attempt to identify anomalies in the hardware data that cannot be explained by the golden model.

The most significant challenge associated with detecting HT with path delay testing is distinguishing between changes in delay introduced by an HT and those introduced by process variation effects. Failing to distinguish between these two types of delay variations leads to false-positive and false-negative HT detection decisions. The former declares an HT is present when it is not, while the latter fails to detect the presence of an actual HT. Minimizing false-positive and false-negative rates is a critical design parameter of an HT detection technique.

There are three basic approaches for dealing with process variation effects in HT methods. The first method, called **GoldenSim-based** and **GoldenChip-based**, creates simulation models or uses HT-free chips, respectively, to characterize the HT-free space. The second, called **PCM-based**, uses data from *process control monitors* (PCM) to "tune" the boundaries of the HT-free space derived from golden models using chip-measured test structure data. The third, called **Chip-Centric**, creates a *nominal* simulation model and *calibrates and averages* path delays to the nominal model (or data from HT-free chips). All approaches create a *bounded HT-free space* that represents normal variations in path delays introduced by process variations and/or measurement noise. Data collected from the test chips is compared with this bounded HT-free space. Data points that fall outside the boundaries are called **outliers**, e.g., are path delays that exceed the limits defined by the HT-free space. Chips that produce outlier data points are considered HT candidates.

The three approaches are graphically portrayed in Fig. 10.9 and are described in more detail throughout Sect. 5 as needed. The 2-D shapes labeled "Simulations with process variations modeled" and "Delay variations across chip population" can in fact be multidimensional, with each dimension representing one path delay or one of multiple features extracted from the set of path delays using statistical techniques such as principle component analysis (PCA).

GoldenChip-based and **GoldenSim-based** techniques train a classifier using HT-free data from chips or simulations, respectively. GoldenChip-based methods measure delays from HT-free chips, which are then destructively validated to be HT-free using techniques from Sect. 10.3.1.3. GoldenSim-based methods typically use data from Spice-level simulations of a resistor-capacitor-transistor (RC-transistor) model of the *golden* design. Both techniques can be expensive in terms of reverse-engineering effort, model development, and simulation time. Delayering technologies utilized for GoldenChip-based methods can take weeks or months. For GoldenSim-based methods, CAD tools such as Mentor Graphics Calibre must first be used to create the RC-transistor models of the layout using complex process models obtained from the foundry in which the chips are fabricated. The modeling files can be very large, e.g., 100s of MB, even for relatively small designs on order

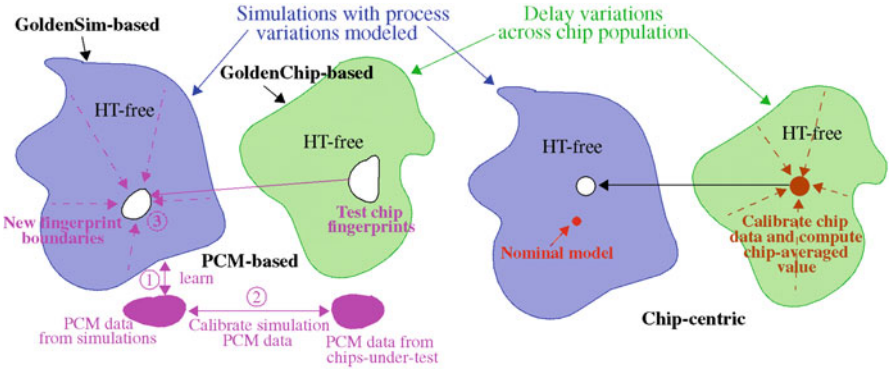


Fig. 10.9 Mechanisms to account for process variations using a *golden* model approach for HT detection

of 20,000 gates, and simulation times can easily extend to weeks and months when performing transient simulations with only a couple 100 input vectors. The effort required to construct and/or confirm the HT-free boundaries using either technique is very large and is often under-reported.

Of even greater concern for GoldenSim-based techniques is the level of mismatch that can exist between the simulation results and the hardware. Foundry models in advanced technologies have become very complex, providing the user with a variety of statistical evaluation methodologies including *fixed corners* and *Monte Carlo* options. Fixed corner models are provided to enable the user to predict worst-case and best-case performance of the chip by modeling the range of global process shifts that can occur over time. Unfortunately, this typically expands the HT-free space beyond what is required to represent the behavior of the chips under test. The expansion leads to a decrease in the sensitivity of HT methods and increases the level of mismatch between simulation and hardware data. Moreover, foundry models typically provide limited capabilities for modeling within-die variation effects, making it difficult to predict the uncertainties related to specific hardware path delays. These modeling and simulation challenges are compounded by measurement noise that occurs during chip testing and by nonzero jitter and drift tolerances introduced by the tester during the generation and delivery of high-frequency clocks. Taken together, these issues work to increase in the possibility of false-positive and false-negative HT detection decisions.

10.4.3 Test Vector Generation Strategies

The last issue deals with an important distinction that exists between fault models used in manufacturing test and those required for detecting HT. The manufacturing test community developed several fault models, including **transition delay faults**

and **path delay faults**, for dealing with timing problems resulting from a wide variety of defect mechanisms. For example, the transition delay fault (**TDF**) model assumes that defects occur on individual *nodes* in the circuit and that they manifest as slow-to-rise and slow-to-fall signal behaviors at those nodes. The path delay fault (**PDF**) model, on the other hand, makes no such assumptions, i.e., it accounts for defects which may in fact be distributed across one or more logic gates and wires that define the paths, and as a result, the PDF model provides more complete information about the integrity of the tested chip.

Unfortunately, obtaining 100% PDF coverage requires all (or a large fractions) of the paths in a chip to be tested. For even moderately sized circuits, the costs associated with the generation and application of a complete PDF test set is prohibitive. This is true because the number of paths can be exponentially related to the number of inputs to the chip (or functional unit). Therefore, most chip companies generate and apply TDF vectors instead because the number of such tests is linear to the number of circuit nodes in the design. Fortunately, for the security and trust community, the TDF model is a better match to the types of malicious modifications an adversary is likely to make to the layout. The node-oriented TDF model used for defects is leveraged by a large fraction of the proposed HT detection techniques described in Sect. 5.

There are two important points to consider with regard to test generation for HT detection. The first relates to the options that are available when the TDF model is used. Although far fewer tests are required under the TDF model to obtain high levels of HT coverage (when compared to the PDF model), there are typically many choices for the *path* that is sensitized through each of the nodes. A variety of techniques are proposed by authors of published work including random vectors, an *incremental coverage* strategy driven by the sequence of vectors generated so far, traditional TDF vectors, or, in some cases, the test generation strategy which is left unspecified. Others leverage the TDF model and direct ATPG to target the **shortest paths** through the node because the additional delay added by the HT (via fanout load or gate insertion) has a larger fractional impact on the path delay. The traditional TDF model for defects, on the other hand, typically target the **longest paths** as a mechanism to ensure that at least this subset of tested paths meet the timing constraints.

The length of the path relates to the second important point regarding test generation. Automatic test equipment is outfitted for manufacturing test, which is focused on testing the longest paths. For test cost reasons, it is common that only one clock frequency is used to apply TDF tests to the chip because the primary goal of manufacturing test is to ensure that the delays of all tested paths are less than the upper bound defined by the clock period. The most sensitive tests for defects therefore are those that test the longest paths. This is true because the longest paths minimize the **slack**, i.e., the difference between the clock period and the delay of the tested path.

Many believe that these manufacturing test constraints for defects are not sufficient for providing high levels of HT coverage. This is reflected in the proposed use of *clock sweeping*, *clock strobing*, and other on-chip embedded test structures for

obtaining precise measurements of path delays. In other words, the slacks inherent in tests for defects provide too many opportunities for adversaries to "hide" the additional delay of the HT in the slack, and, therefore, a paradigm shift is required regarding the manner in which delay testing is carried out on the test floor. Clock sweeping and clock strobing are expensive in terms of test time, and HT detection techniques which use these clocking strategies need to account for the higher levels of clock noise associated with high-frequency clocks and invalidations introduced by circuit hazards. It remains to be seen how the economic trade-offs of delay-based HT detection schemes will play out.

10.5 HT Detection Methods Based on Path Delay Analysis

This section is dedicated to describing a selected subset of the proposed HT detection strategies that have been proposed over the last decade. Our goal is to describe methods that offer some unique perspective, and therefore, this exposition does not provide an exhaustive survey of every published paper on the topic. The choice to include a description of a published work was based on whether it promoted the state of the art in at least one of the three technical domains described earlier, including the path measurement technique, the statistical method used to distinguish between HT anomalies and process variation effects, and the test vector generation strategy. The techniques are presented chronologically instead of by technical domain. The latter organization presented challenges because many techniques propose solutions to more than one domain.

10.5.1 Early HT Detection Techniques and On-Chip Measurement Methods

The first works on using path delays for HT detection are described in [40] and [41]. The primary focus of each paper is on only one of the technical domains, in particular [40] on a statistical method for distinguishing between process variations effects and HT and [41] on a high-resolution on-chip measurement technique.

In [40] the HT detection method assumes that high-resolution path delay measurements are available, i.e., no measurement strategy is proposed. Although not explicitly stated, the test vector generation strategy appears to be based on the **standard transition delay fault** model. They base their detection method on the **GoldenChip-based** model described in Sect. 4.2. A multivariate statistical technique is used to extract distinguishing features from the full set of path delays. HT-free chips are used to construct the HT-free boundaries, which they refer to as a *fingerprint*. The fingerprints define the boundaries of the shape labeled "Delay variations across chip population" shown on the left side of Fig. 10.9. HT are

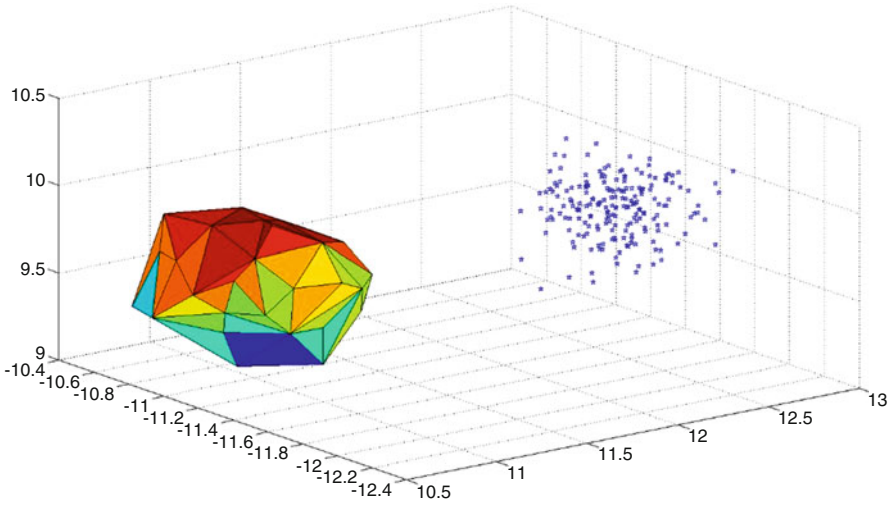


Fig. 10.10 Convex hull characterization showing detection of an HT [40]

detected by comparing the delay fingerprints measured from the untrusted test chips with the boundaries defined by the HT-free fingerprints.

They demonstrate their technique using simulations in which ATPG-derived two-vector sequences are applied to a DES functional unit. Principle component analysis (PCA) is used to extract distinguishing features from a set of 10,432 simulated path delays as a means of reducing the HT-free space to a 3-D structure. A statistical technique based on a *convex hull* characterization of the HT-space is used to define the boundaries for each of the 64 outputs of DES. Four HT are inserted into another set of models, with three representing *explicit payload HT* and one representing an *implicit payload HT*. The explicit payload HT inserts one or more additional gates in series with paths in the HT-free design, while the implicit payload HT is represented as a simple counter with no ability to change the functional characteristics of DES. They show that the explicit payload HT are easily detected (see Fig. 10.10), while the implicit payload HT is only detected approx. 36% of the time.

A high-resolution on-chip path delay measurement technique is proposed in [41], which is extended in [42] to include a **GoldenSim-based** HT detection strategy. The measurement technique is based on the **dual-clock** scheme described in reference to Fig. 10.7. A set of *shadow registers* are added to each of the outputs from the combinational components of the design, next to the capture FFs or *destination register*s as shown in Fig. 10.11a. The second clock of the dual-clock scheme, *CLK2*, is used to drive the clock inputs of the shadow registers. *CLK2* is generated as a "fine-phase-shift" adjusted version of *CLK1* using a DCM on the FPGA.

The process of measuring the path delay of the combination path from Fig. 10.11 begins by setting the phase shift of *CLK2* to a small negative value, on order of 10–100 ps (see Fig. 10.7). A two-vector sequence is applied to the source registers

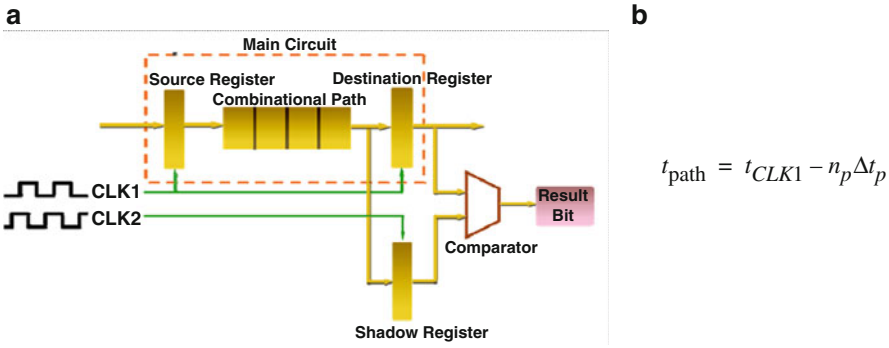


Fig. 10.11 (**a**) On-chip path timing architecture proposed in [41] and (**b**) equation giving actual path delay

using a launch-capture test. The *comparator*, also added to the design, is used to determine if the captured values in the destination and shadow register are the same or different. If they are the same, which is the case when the clock strobe operation begins, the negative phase shift difference between *CLK1* and *CLK2* is increased, and the same two-vector sequence is applied. This process is repeated until the comparator indicates the values are different. The actual delay of the path is computed by multiplying the number of phase shifts, n_p, by the Δt_p provided by each phase shift increment. Subtracting this value from the *CLK1* period yields an estimate of the path delay, t_{path}, as given by the equation in Fig. 10.11b.

The extended work in [42] investigates the detection capabilities of a **GoldenSim-based** technique on an 8-bit Braun multiplier functional unit. HT are modeled as series-inserted two-inverter chains. The proposed method derives a path delay distribution using simulation data but is constrained by the measurement resolution provided by the timing technique. Process variations are modeled by varying transistor threshold voltage, V_{th}, and transistor channel length, L_{eff}, in simulations with and without HT. Data from these simulations is used to define the boundaries of the shape labeled "Simulations with process variations modeled" shown on the left side of Fig. 10.9. The amount of skew in the mean of the distributions is used as the detection criteria. The results using four inserted HT show that three can be detected and the last one is detectable on some outputs but not others.

10.5.2 Ring Oscillator-Based HT Detection Approaches

A distributed set of ring oscillators (RO) is proposed in [43] as a means of detecting HT. The array of ROs is distributed uniformly across the (x,y) space of the functional unit as shown in Fig. 10.12. The detection criteria are based on HT

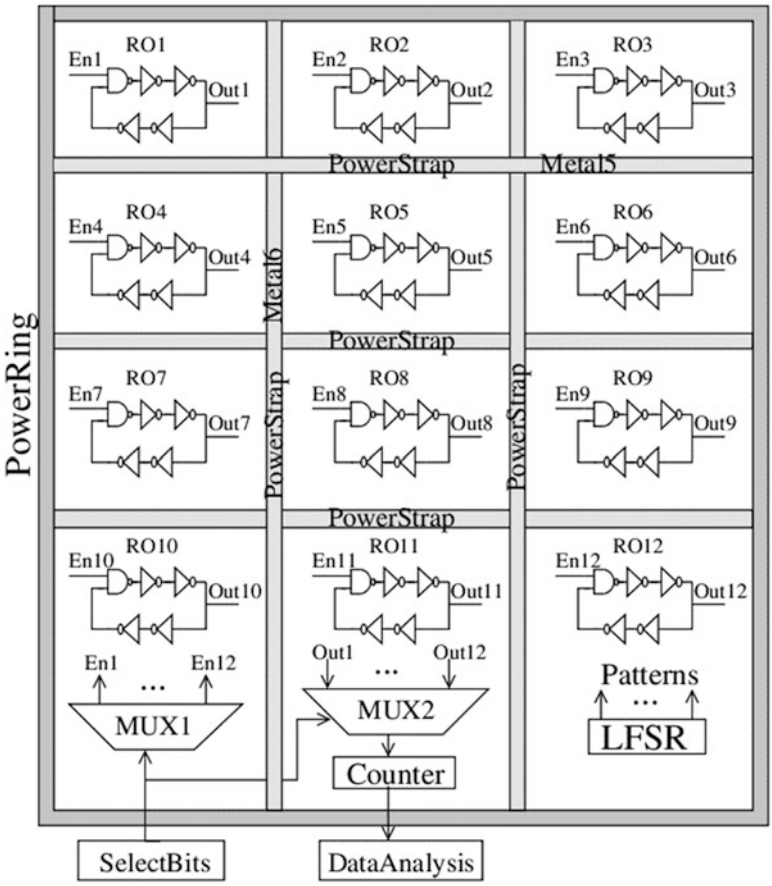


Fig. 10.12 RO distribution architecture proposed in [43] for detecting HT switching activity

power consumption. If an HT is present, the test stimulus applied to the functional unit may cause at least some gates within the HT to switch (referred to as *partial activation* in [1]), and the HT will necessarily consume power. The additional HT power consumption creates localized voltage drops on the supply rail (V_{DD}) that can be detected by comparing the delay of a nearby RO with that of an HT-free chip or simulation. The delay variations introduced by the HT-switching-induced voltage drops in the RO are integrated by the RO over time and are reflected in a counter value. A counter is connected to the RO using MUX2 as shown along the bottom of Fig. 10.12. MUX1 is used to select and enable one RO at a time as a random, LFSR-based test vector sequence is applied to the inputs of the functional unit. This process is repeated for the n ROs (12 in figure) with the set of counts representing the *signature* for the chip.

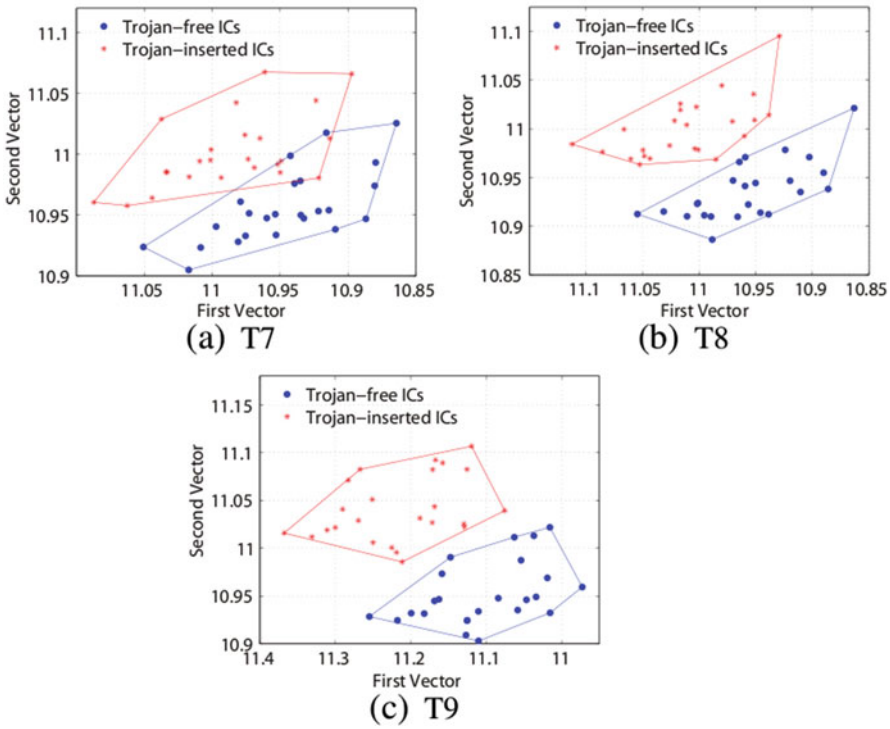


Fig. 10.13 Subset of results from [43] using proposed *advanced outlier analysis* method on FPGAs

Process parameter variations are accounted for by collecting data from a large number of HT-free chips (**GoldenChip-based** model), and a statistical analysis is applied to the signature using principle component analysis (PCA) and correlation analysis. One of the techniques, called *advanced outlier analysis*, analyzes data obtained from pairings of ROs as a means of detecting **regional** power droop anomalies. The results shown in Fig. 10.13 plot the RO pairings that show the best detection capability for each of the six HT inserted into the design. The points within each graph represent experimental results with process variations derived from FPGA experiments. The separation of the red HT-inserted and blue HT-free points illustrates that nearly all are detected in every FPGA.

The authors of [44] propose a *design-for-trust* (DFTr) technique designed to detect HT by creating ROs from the functional paths in a design.[3] They propose an algorithm that selects paths with the maximum number of "unsecured gates," i.e., gates that have not already been included in other ROs, i.e., a method characterized

[3] A related design-for-manufacturability scheme was proposed earlier in [39] for measuring critical path delays.

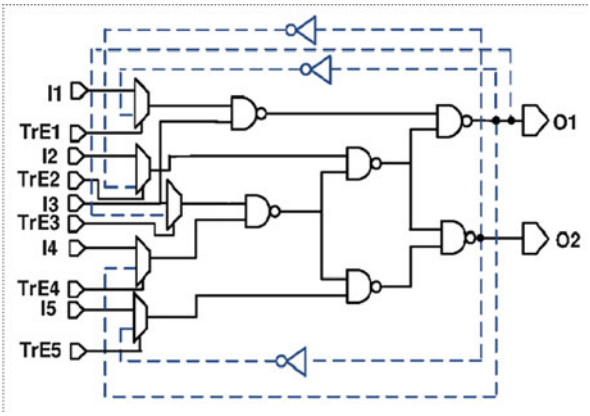


Fig. 10.14 ISCAS-85 benchmark C17 configured with a set of ROs using the DFTr method proposed in [44]

as an *incremental coverage*-driven strategy. For each selected path, a MUX, a control signal, and optionally an inverter are added to the design to complete the ring. Automatic test pattern generation (ATPG) is then used to generate input patterns that place *nondominant* values on the *off-path* inputs of gates sensitized by the ring. Nondominant refers to gate input values that do not determine the gate's output value by themselves, e.g., a "1" is the nondominant value for an AND gate. Off-path inputs refer to side inputs of gates in the RO that are not on the sensitized path of the RO. These conditions ensure that the RO will "ring" when enabled by the control signal. Figure 10.14 shows the ISCAS-85 benchmark circuit *C17* configured with a set of ROs.

Simulation experiments with process variation effects modeled are carried out to determine the golden frequencies, F_{golden} (**GoldenSim-based** model). HT detection is carried out by comparing $F_{measured}$ obtained from each of the untrusted test chips with F_{golden}. The authors implemented their technique on a Xilinx Spartan 3 FPGA using six ISCAS-85 benchmark circuits. The number of ROs required to attain a specific level of HT coverage is given in Fig. 10.15a. As is typical of test generation for manufacturing defects, coverage per RO drops dramatically for coverage targets above 90%. Test times are given in Fig. 10.15b which shows similar trends. Although the proposed technique is very promising, the authors do not address hazards that occur within circuits with reconvergent fanout.

10.5.3 Lightweight On-Chip Path Timing Techniques for HT Detection

Two on-chip delay measurement techniques, called TDC and REBEL, are proposed in [26]. An HT detection method which leverages REBEL [45] is described in the following. A second HT detection method which uses the TDC [59] is covered in Sect. 5.11.

REBEL is designed as an embedded test structure (ETS). ETS is an instrument that is integrated directly into functional units as a mechanism to obtain regional,

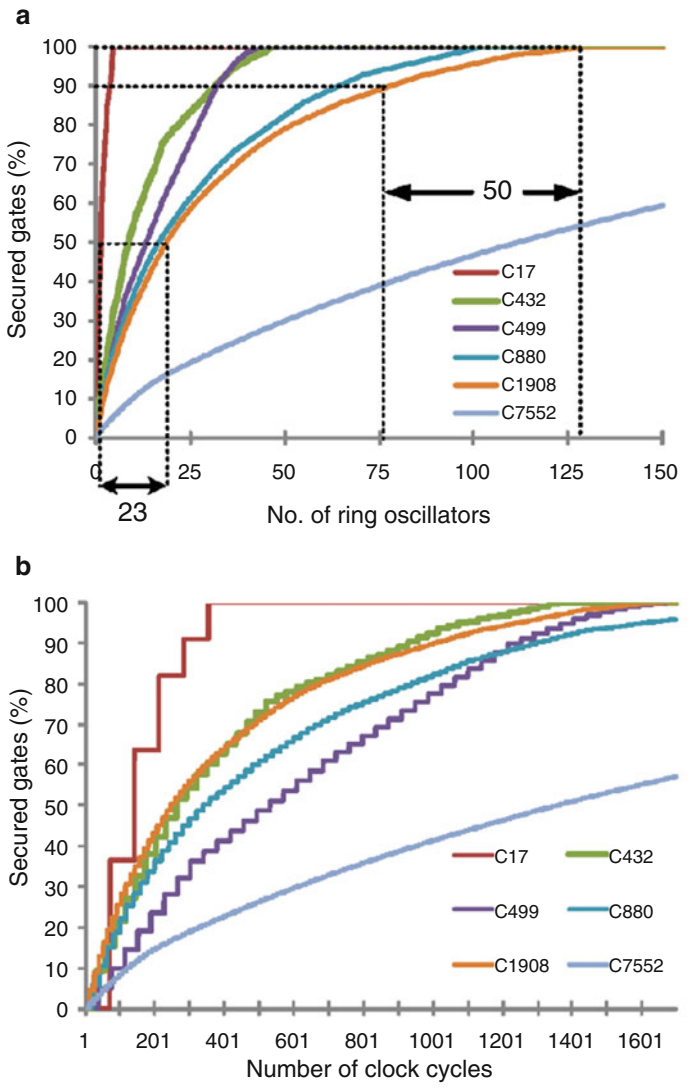


Fig. 10.15 (**a**) Number of ROs required as a function of coverage for six ISCAS-85 benchmark circuits, and (**b**) test time required to obtain $F_{measured}$ for each chip [44]

high-precision information about its operational characteristics. ETS must be designed to be minimally invasive and low in overhead to avoid violating power, area, and performance constraints associated with the functional unit. REBEL satisfies these ETS attributes by leveraging components in the existing scan chain architecture.

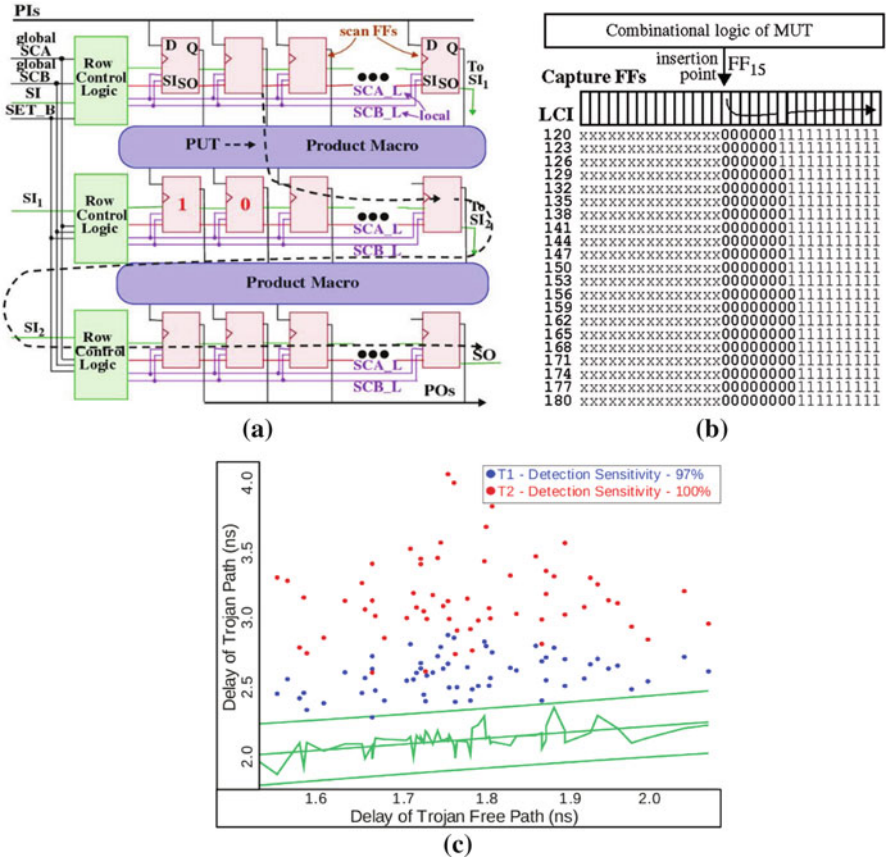


Fig. 10.16 (**a**) Integration of an embedded test structure called REBEL in a pipelined functional unit as described in [45], (**b**) sequence of digital "snapshots" stored in the REBEL delay chain with successive rows showing an increasing launch-capture interval (LCI) and (**c**) illustration of regression analysis applied to path delays measured from 62 chips for an HT-free path (*x*-axis) and a second path (*y*-axis) with and without the inclusion of HT

REBEL is designed to provide regional, high-resolution measurements of path delays. It also addresses the clock noise issue discussed in Sect. 4.1 related to using single-clock schemes to obtain timing information for short paths. The architecture of REBEL allows paths within the functional unit to be extended along a delay chain, effectively eliminating the need for high-frequency clocks. The delay chain is created using the existing capture FFs attached to the outputs of the functional unit. Figure 10.16a shows an example configuration with REBEL integrated into a pipelined functional unit. A path under test (PUT) within the functional unit is highlighted as well as the delay chain that is created through the capture FFs. Row control logic is added to the design to enable one of the path outputs to be selected as the target of the timing measurement process.

A path is timed by applying a two-vector sequence to the inputs of the functional unit. The transition along the PUT emerges at the output and propagates along the delay chain. The capture edge of the clock creates a digital snapshot of the transition by storing in the capture FFs a sequence of digital values which represent its behavior over time. Each of the snapshots immediately reveal whether the propagating signal has more than one transition, i.e., whether a hazard is present or not.

A sequence of digital snapshots is shown in Fig. 10.16b as rows labeled 120 to 180, which represent a *clock sweeping* sequence of LCIs as described earlier in reference to the single-clock scheme of Fig. 10.7. For each successive LCI, the propagating falling transition driving the input of the capture FF_{15} is given more time to propagate along the delay chain. The path tested in this example does not generate any type of hazard (is hazard-free); otherwise, one or more of the snapshots would show interleaved "1s" in the sequence of "0s." In practice, the LCI test sequence is actually applied in reverse, starting with 180, because larger LCI increases the amount of temporal information stored in the capture FFs regarding the propagating transition. The larger time window provides a better opportunity to detect hazards which can invalidate the HT test.

As proof of concept, a 90 nm chip is designed and tested which allows paths in the functional unit (in this case, an eight-function floating point unit or FPU) to be reconfigured with and without an HT. Although the experimental results presented use hardware data to define the HT-free space (**GoldenChip-based** model), the authors also acknowledge that a **GoldenSim-based** approach is possible.

The statistical method proposed in [45] is based on linear regression analysis. Figure 10.16 shows a subset of the results in which delays from one HT-free path (x-axis) and a second path that can be configured with and without HT (y-axis) are compared. The HT-free space is highlighted in green and is delineated by three sigma regression bands. It is derived using data from 62 copies of the test chips configured without the HT included in the second path. The red and blue points each represent the results with the second path configured with one of two possible HT. All but one of the data points fall outside the regression limits and are therefore classified as detected. Additional results for a larger set of HT are reported in the paper.

10.5.4 Self-Authentication: A Golden Model-Free HT Detection Method

A golden model-free HT detection method is described in [46] which inserts a framework of *HT detection sensors* into the layout representation of a design.[4]

[4] A self-referencing technique is also proposed earlier in [47] but is based on the correlation of transient power supply currents produced by replicas of the functional unit.

The sensors are designed as replicas of common sequences of logic gates (path sequences) that already exist in the design. Custom CAD tools are used to decompose the timing graph of a design to identify a set of commonly occurring *delay features*. Delay features correspond to layout-specific patterns of gates and interconnect that share common geometries and sensitivities to process parameter variations. Similarity among features is determined by evaluating the *changes* in delay that occur when the path sequences are subjected to similar process conditions. Two sequences are considered *similar* if the changes in their delay track within a small error tolerance.

Once a set of target path sequences are identified in the design, a set of matching sensors are integrated into the layout in close proximity to the targeted path sequences. After fabrication, the delay fingerprints of the sensors and corresponding full-length paths (that contain the path sequence(s)) are measured. Data from each of the sensors is used to construct an HT-free delay range, which captures the measurement noise profile for the sensor. A similar process is carried out for each of the paths. The delay associated with other components of the full-length paths, in which the path sequences are contained, is accounted for using variation-aware expressions. A nominal simulation or static timing analysis estimate is used to determine the nominal delay of the sensor, which, in combination with the measured delays, allows the delay of the full-length paths to be predicted. Correlation analysis is used to compare the predicted and measured delay ranges for the sensors and paths. Outliers are considered anomalies introduced by HT in either the sensor, the path, or both. Figure 10.17 provides a flow diagram of the self-authentication process and shows examples of the HT insertion and detection scenarios considered [46].

The authors assume on- or off-chip delay measurement schemes such as those described in reference to Fig. 10.7 are available. The sensors act as *silicon-anchor* points for calibration, and therefore the proposed HT detection technique shares similarities to the *process control monitor* approach described in Sect. 4.2 and referred to as **PCM-based**.

The authors apply the technique to the ISCAS-89 benchmark circuits, synthesized to layouts using a 90 nm TSMC technology. Process variations are modeled in the simulations by varying major device parameters within 10% of nominal using a Monte Carlo selection process. A multilevel hierarchical model of the layout is processed as a means of partitioning the design into regions where it is assumed that process variations are more highly correlated. Sensors are identified and designed but constrained to use no more the 15% additional area in the layout. A set of paths are randomly selected to serve as the HT insertion points for evaluation of the method. A set of 30 HT are inserted into each path with varying amounts of delay to determine sensitivity, and 10 K process models are created and simulated (300 K per path). HT detection rates are shown to improve from between 2 and 16% when compared to a similar method that does not leverage sensors as a sensitivity-enhancing technique.

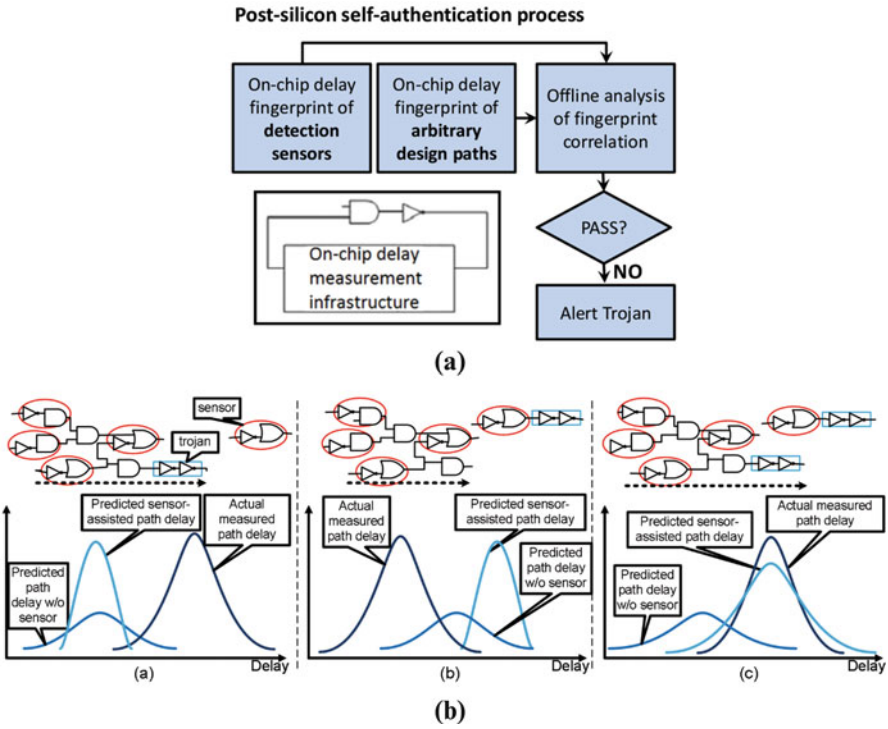


Fig. 10.17 (**a**) Self-authentication chip testing process proposed in [46] and (**b**) sensor (both simulated and measured) and measured full-length path delay range illustrations under three HT attack models. Mismatches in the overlap among the sensor distributions or low levels of correlation between sensor and path distributions is flagged as an HT detection

10.5.5 Linear Programming Methods and Test Point Insertion for HT Detection

A linear programming method is proposed to derive leakage, power, and delay characteristics of individual gates based on solving a system of equations, referred to as *gate-level characterization* (GLC) [48–50]. Chip measurements of power and delay are used in the system of equations, along with estimates of measurement errors, to derive scaling factors for the parameters associated with the logic gates in the design.

GLC is combined with a test point insertion technique in [49] as a means of improving coverage of HT. The authors propose to add FFs (test points) to components of the design that exhibit *reconvergent fanout*. An example of reconvergent fanout is shown in Fig. 10.18 from [49] where both the fanout and reconvergence points are identified. The task of generating path delay tests for circuits which contain complex reconvergent fanout networks is an NP-complete

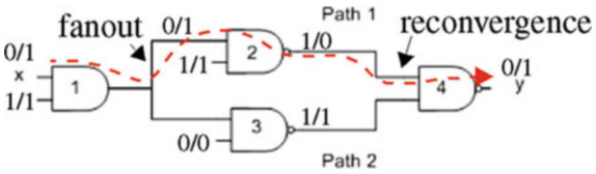


Fig. 10.18 Circuit showing reconvergent fanout [49]

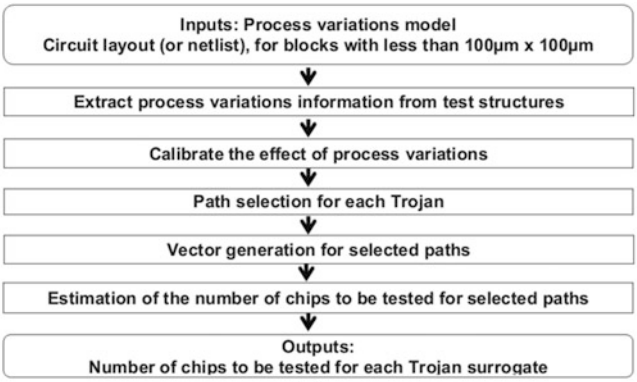
problem. Automatic test pattern generation (ATPG) algorithms can fail to determine two-vector sequences that are able to test the individual paths within reconvergent fanout blocks, such as those labeled "path 1" and "path 2," even when such tests exist. Although in the example it is trivial to derive test patterns that test these two paths individually (node assignments are shown that allow path 1 to be tested by itself), there are other more complex configurations which require an exhaustive search, proportional to 2^n, to find suitable two-vector sequences.

The authors propose an algorithm that first identifies all paths that can be easily tested and then a set of paths in reconvergent fanout logic structures that are the best candidates for test point insertion. A SAT-based process is proposed to select input vectors that maximize the number of independent linear equations for application of GLC. A second circuit partitioning scheme based on *maximum fanout free cones* is proposed in [50] to increase the number of delay access points within the circuit as a means of improving coverage further for large designs.

10.5.6 Process Calibration and Test Vector Selection for Enhancing HT Detection

A delay calibration technique is described in [51] that leverages information obtained from test structures as a means of detecting HT delay anomalies that are very small, i.e., within the margin of those introduced by process parameter variation effects in advanced technologies. The test structure measurements are used to estimate the global mean shift in delay introduced by variations in the process parameters for each chip. Based on the estimate, the mean value for the paths in the region of the embedded test structures is calibrated to eliminate the mean shift.

The process flow proposed in [51] is shown in Fig. 10.19. The first step is to extract information from the embedded test structures, such as ring oscillators, as a means of obtaining process parameter information for each chip. Test structures are added to the layout in regions close to the functional unit to be tested. This ensures that both global process variations and systematic within-die variations are accurately captured in the measurements. Path selection and vector generation are carried out such that test cost is minimized. ATPG is constrained to generate robust tests (when possible) for critical (longest) paths passing through each possible HT site, and therefore test generation is based on the **traditional TDF** model. Path delays are measured using the **dual-clock** scheme shown in Fig. 10.7 as a



(a)

TABLE V. DELAYS MEASURED ON VARIOUS INVERTER CHAINS (L: INVERTER CHAIN LENGTH, σ: STD. DEV.. OF DELAY, Δ: EXTRA DELAY INDUCED BY A TROJAN, μ: MEAN DELAY, AND N: THE NUMBER OF CHIPS TO BE TESTED)

Case I				Case II				Case III			
L	σ/μ	Δ/μ	N	L	σ/μ	Δ/μ	N	L	σ/μ	Δ/μ	N
2	0.269	0.207	14	2	0.246	0.207	13	2	0.244	0.206	11
4	0.185	0.163	14	4	0.164	0.162	13	4	0.162	0.161	11
6	0.143	0.126	14	6	0.125	0.125	13	6	0.122	0.125	11
8	0.124	0.101	17	8	0.105	0.100	13	8	0.103	0.099	12
10	0.113	0.088	20	10	0.092	0.087	13	10	0.088	0.086	12
12	0.107	0.076	20	12	0.084	0.075	14	12	0.081	0.074	12

(b)

Fig. 10.19 (**a**) Process flow model proposed in [51], (**b**) simulation-derived delay statistics obtained by applying the proposed method under different types (global and within-die) process variations

means of minimizing clock noise and obtaining high-resolution measurements. The integration of silicon anchor points for calibration of process variations classifies the proposed technique as **PCM-based**.An estimate of the mean shift in each region of the chip is computed using test structure data and a *minimum mean square (MMSE) estimator*. The MMSE finds the mean that minimizes the sum of the squared differences between the test structure data and the computed mean as given by Eq. 10.1. This estimator can then be used to calibrate all the path delays for a given chip by simply subtracting the mean from each measurement. A novel nonparametric hypothesis testing method based on a *likelihood-ratio test* is proposed which leverages integer linear programming (ILP) for determining the number of chips that need to be tested to achieve a specific confidence level against false-positive and false-negative HT detection decisions.

$$\sum_{i=1}^{M} (d_i - \widehat{\mu})^2 \tag{10.1}$$

The technique is evaluated on a set of inverter chains of length 2 through 12 with and without HT insertions. HT are modeled using a minimum size inverter connected to the output of the first inverter in the chain. Monte Carlo simulations were performed using circuit models with different types of global and within-die process variations modeled, referred to as case I (global only), case II (across-chip random and systematic within-die only), and case III (local random and systematic within-die only). Figure 10.19b shows that the best results are obtained for case III which uses simulation models *with only local process variations included*. The proposed calibration method in this case makes this possible by eliminating case I and case II via the test structure measurements, which minimizes the σ/μ statistical variation parameters as well as the number of required chips.

This technique is extended in [52] to address the best paths to target for HT detection. In contrast to their earlier work, the authors argue that the shortest paths through each HT site maximize detection sensitivity (see **shortest path TDF** discussion in Sect. 4.3). The column labeled Δ/μ in Fig. 10.19b expresses the impact of the HT on path delay and is the focus of the current work. Given that the adversary's goal is to minimize the impact of the HT on path delay, shorter paths are better suited to reveal these small delay variations because the (constant) delay added by the HT becomes a larger fraction of the total delay for short paths. A similar argument regarding the effect of process variations also holds. In particular, the σ of variations is approx. Proportional to the nominal delay of the path, i.e., shorter paths have smaller σ. This characteristic is illustrated in Fig. 10.20 which shows the path distributions for a long path (top) and short path (bottom) with and without HT. The HT, represented as an "additional fanout," creates a more distinguishable shift in the short path distribution when normalized as a fraction of the total width of the distribution. In both cases, the HT adds only 8 ps to the path delay but the smaller σ corresponding to the shorter path provides a higher level of confidence in detecting the anomaly.

The authors also argue that shorter paths are more likely to be the targets of an HT insertion because longer paths, particularly critical paths, increase the chance of accidental discovery. Moreover, generating vectors for shorter paths is generally "easier" for ATPG tools to accomplish because fewer side inputs need to be "justified" (forced to specific values) in order to sensitize the path from PI to PO. The main benefit of short paths, however, according to the authors, is the *reduction in the number of chips that need to be tested* (see column labeled *N* in Fig. 10.19b). On the downside, shorter paths are harder to time, especially when using the single-clock scheme from Fig. 10.7, because the chip needs to be tested at much faster than at speed to obtain precise delay measurements. The dual-clock scheme provides a solution, but it also requires the addition of a second clock tree as described in [41].

An algorithm is presented that both selects the shortest path through each circuit node (each HT site) and enforces constraints on the *robustness* of the test to ensure the target path is in fact the path tested by the two-vector sequence. The authors

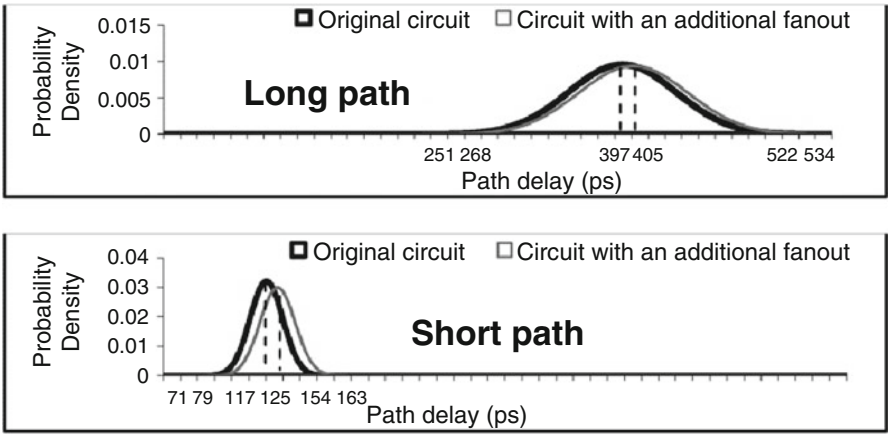


Fig. 10.20 Impact on path delay distribution for a long and short path, with short path showing larger fractional change [52]

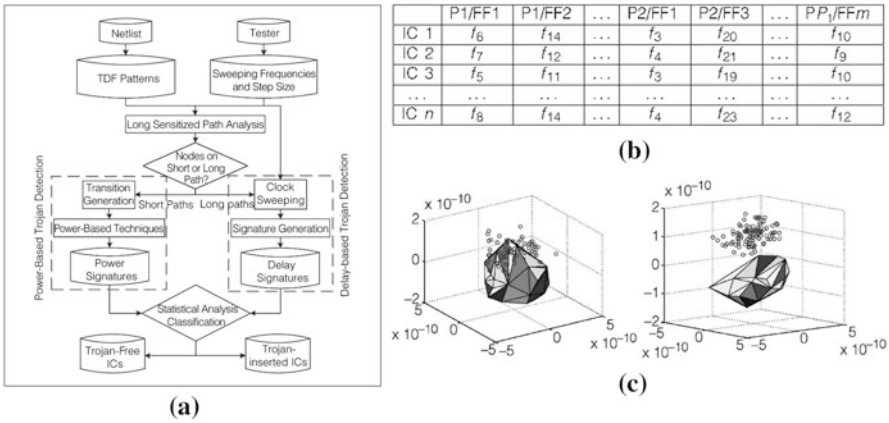
present simulation results using the ISCAS-89 benchmark circuits that show a 2.1X improvement in test cost over a traditional TDF strategy. They further show that the improvement increases to 4.51X when combined with the calibration technique proposed in [51].

10.5.7 Clock Sweeping for HT Detection

The authors of [53] propose a *clock sweeping* method to address sensitivity issues associated with using a traditional TDF model and the path delay fault (PDF) model for detecting HT. Clock sweeping refers to the **single-clock** scheme referenced earlier in Fig. 10.7 in which the clock frequency is incrementally increased (by a fixed *step size*) and a two-vector sequence is applied repeatedly until a *delay fault* is detected in the capture FFs for one or more of the tested paths.

The authors propose to generate tests using the TDF model described earlier and acknowledge that short paths whose delay is smaller than the maximum frequency are not testable because of the limits of ATE and clock noise. The algorithm that they propose is shown in Fig. 10.21a. It partitions TDF tests into two groups, those sensitizing long paths to be tested in the proposed HT delay technique and those sensitizing short paths to be tested using a power-based HT method. They argue that long paths experience less switching activity because more conditions need to be met in order to sensitize them. Therefore, power-based methods are less effective for detecting HT on these paths.

The failing frequencies for long paths are recorded in a table as shown in Fig. 10.21b. Chips are listed on rows, while the columns identify the pattern, Px, and Capture FF, FF_x, of the tested paths. A multidimensional scaling (MDS) statistical



	P1/FF1	P1/FF2	...	P2/FF1	P2/FF3	...	PP_1/FFm
IC 1	f_6	f_{14}	...	f_3	f_{20}	...	f_{10}
IC 2	f_7	f_{12}	...	f_4	f_{21}	...	f_9
IC 3	f_5	f_{11}	...	f_3	f_{19}	...	f_{10}
...	...	...	...	...	...	...	...
IC n	f_8	f_{14}	...	f_4	f_{23}	...	f_{12}

Fig. 10.21 (**a**) Algorithm proposed for HT detection in [53], (**b**) chip signatures recording the first failing frequency for pattern (Px) and capture FF (FFx), and (**c**) MDS/convex hull results for 2 HT

method is proposed for distinguishing between delay variations introduced by process variation effects and those introduced by HT. MDS leverages PCA to map from a higher-dimensional space to a smaller space. Unlike the technique proposed in [40], however, they configure MDS to preserve signature components that represent dissimilarities introduced by HT delay anomalies in the lower-dimensional space. A 3-D convex hull is constructed using signatures from HT-free chips and outlier data points from the untrusted chips are classified as HT candidates. Their detection technique therefore is based on the **GoldenSim-based** or **GoldenChip-based** model described in reference to Fig. 10.9.

An ISCAS-89 benchmark circuit s38417 is used in their simulation experiments to validate the technique. Simulation models representing process variations are constructed by varying threshold voltage, oxide thickness, and channel length over 5% of nominal both globally and locally to model within-die variations. A total of six HT are introduced in a layout representation of the benchmark circuit with varying trigger and payload configurations. The clock frequency range and step size used for clock sweeping is set to 700 MHz to 1.5 GHz and 10 ps, respectively. The results of applying MDS and constructing a convex hull are shown in Fig. 10.21c. The detection rate for HT #1 is 64%, while the rate for HT #2 (and the remaining four HT not shown) is 100%. A similar set of results are obtained in hardware experiments using a set of 44 FPGAs.

10.5.8 A Golden Chip-Free Method for HT Detection

The authors of [54] propose the use of *process control monitors* (PCMs) that are designed to eliminate the need for a set of HT-free *golden chips*. PCMs are in-line

test structures traditionally inserted by process engineers for tracking wafer-level variations in transistor parameters such as threshold voltage. The authors use the delay of a special path as a surrogate for a PCM as a silicon calibration method. Path delays from this PCM are measured from the test chips and used to improve the accuracy of the classification boundary first obtained from simulation data. This detection strategy is therefore **PCM-based** as discussed in Sect. 4.2.

The authors employ *nonlinear regression* and *kernel mean matching* techniques to learn the relationship between PCM data and side-channel fingerprints, in this case, output power measurements from a set of 40 wireless cryptographic chips which instantiate AES with and without HT. A series of "learned" boundaries are incrementally tuned as each of five statistical transformations are applied using simulation data obtained from the PCM and AES process models and from the PCM and power measurements from the test chips; Fig. 10.22a shows the first set of transformations which are derived from simulation data, illustrating transformations that take place in the shape and boundaries of the HT-free space. The remaining transformations are derived using PCM and path delay data measured from a set of HT-free chips and are illustrated in their paper. Figure 10.22b shows experimental results in which all 80 HT are correctly classified as HT infested, while only three of the HT-free chips are classified incorrectly, i.e., are false positives.

10.5.9 HT Detection by Comparing Paths with Structural Symmetry

An HT detection method based on validating delay consistency among instances of distinct transistor-level paths with the same topology is proposed in [55]. Symmetry is defined by considering both the structural characteristics of the logic gate(s) and state assignments on its inputs under each of the vectors of an applied two-vector sequence. For example, a NAND gate exhibits symmetry in delay by having two identical pull-up paths through its two PMOS transistors and when input transitions are crafted to exercise each of these paths, at a time, during a delay test. An HT detection algorithm is proposed that first identifies transistor-level symmetry in the netlist or layout and then adds constraints to ATPG algorithms to test pairs of pairs that exhibit this symmetry. A *self-referencing* detection algorithm is proposed that compares the delays of symmetrical paths and classifies a chip as having an HT when the two path delays are not identical within a threshold.

The authors present an example of transistor-level symmetry using the ISCAS-85 c17 benchmark circuit, which is reproduced with enhancements in Fig. 10.23. The gate-level netlist of c17 is shown on the far left, while transistor-level netlists representing subsets of the netlist are shown in (a) through (d). The transistor-level diagrams are annotated with numbers to enable the NAND gates to be cross-referenced to the c17 schematic. The transistor level schematics along the top and bottom rows represent the two paths that exhibit symmetry. The first two-vector

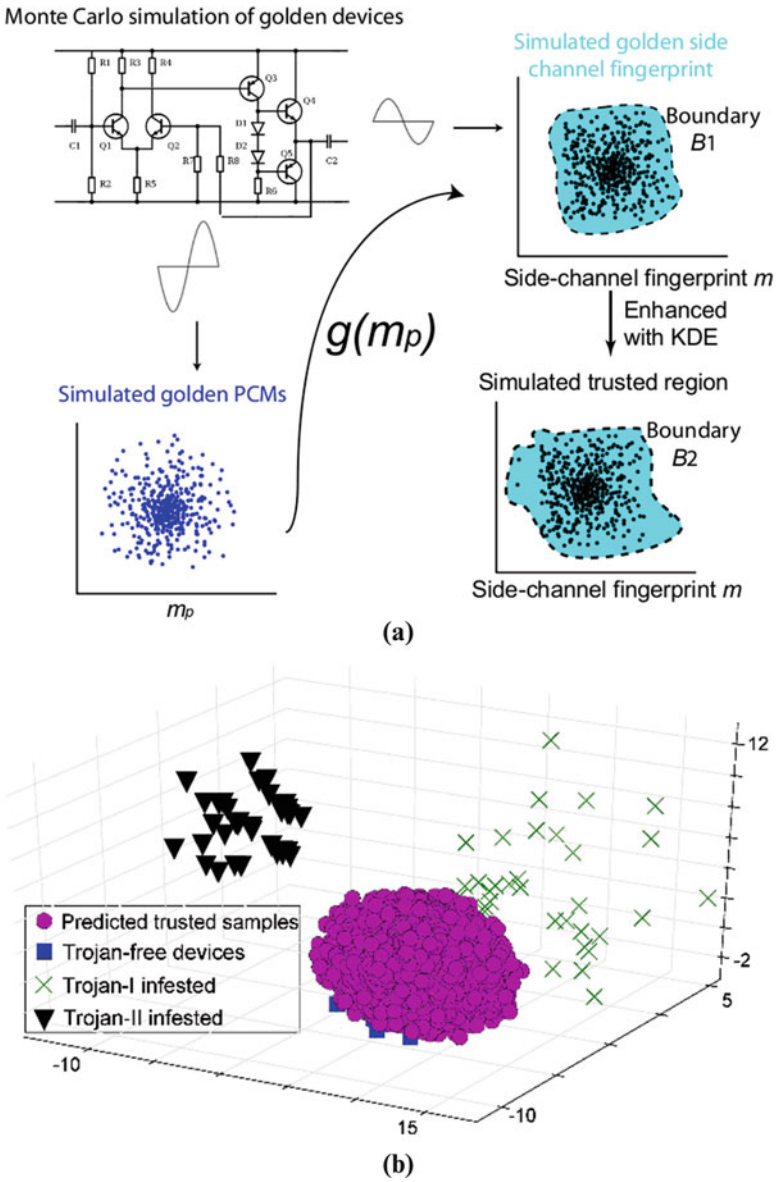


Fig. 10.22 (**a**) Initial simulation-based statistical transformations designed to iteratively learn the best boundaries associated with the HT-free fingerprint space for a wireless cryptographic IC from [54], (**b**) top three principle components from PCA analysis after application of proposed statistical learning process. All 80 HT are detected, and all but three of the HT-free chips (of 40) are classified correctly

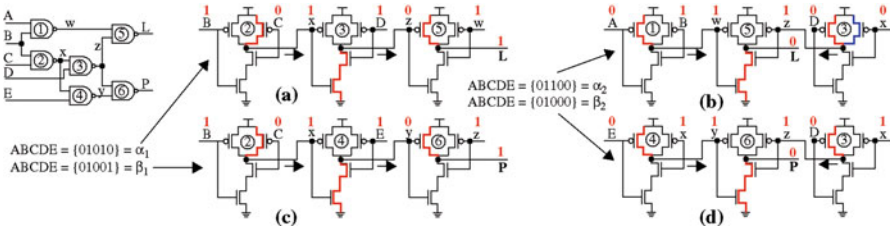


Fig. 10.23 Transistor-level symmetry illustration adapted from [55]

sequence of the symmetry pair is labeled α_1 and α_2, while the second two-vector sequence is labeled β_1 and β_2. The two-vector sequences used as the tests are given as (α_1,α_2) and (β_1,β_2).

The red highlighted components show the pull-up and pull-down paths that connect the output of each NAND gate to one of the supply rails, which is determined by the logic state imposed by each of the four vectors. For example, the outputs of the three NAND gates in (a) are connected to V_{DD}, GND, and V_{DD} for gates labeled 2, 3, and 5. The key observation is the consistency of the highlighting between (a)–(c) and (b)–(d) and the fact that the actual gates in these pairs are different except for one of the gates. In other words, the application of the two two-vector sequences tests the same pull-up and pull-down paths in the NAND gates but do so along different paths in c17. Given that the NAND gates have identical layout structures, the path delays are expected to be nearly identical. Therefore, if an adversary inserts one or more payload gates in series with either of these paths, the delays will be different and can be flagged as a malicious modification. Simulation and FPGA results are shown to demonstrate this concept.

The delay changes introduced by global shifts in process variations (and within-die variations to some degree) are eliminated because the comparisons are made between paths on the same chip and preferably in close proximity. Therefore, none of the *golden model* techniques referenced in Sect. 4.2 for dealing with process variations are required. However, margins are needed to account for measurement noise and routing differences in the two paths; otherwise, the false-positive rates will be high. The authors indicate that finding structural symmetries in the layout and then deriving qualifying test patterns can be challenging given the large number of constraints that must be satisfied to ensure consistency in the behaviors of the pull-up and pull-down components of the tested paths. This *feature* can be argued as a benefit because it makes the task difficult for the adversary to carry out and then defeat the technique by inserting the HT such that the delays of symmetrical pairs of paths remain consistent.

10.5.10 HT Detection Using Pulse Propagation

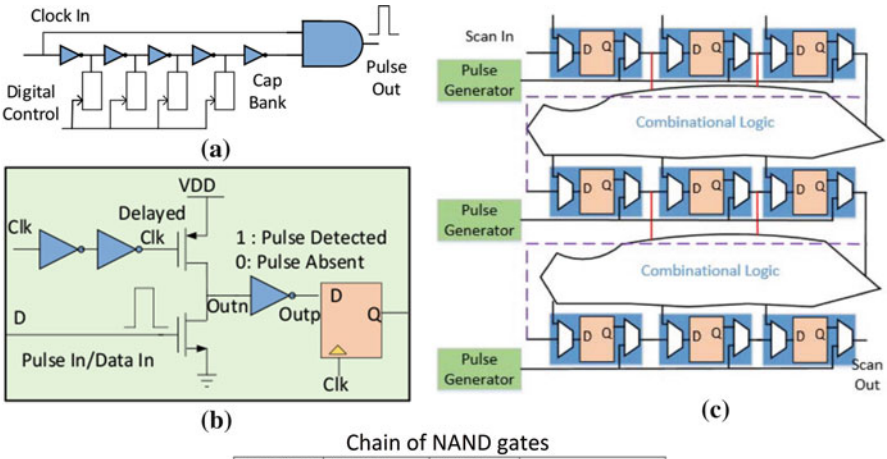
A high-resolution HT detection method is proposed in [56] that is based on propagating pulses along digital logic paths. HT detection is accomplished by detecting whether pulses survive, i.e., do not die out, before reaching the capture FF where they are detected. Minimum pulse widths that allow the gates along the path to sustain the pulse are constrained by only one of the gates along the path, in particular, the gate that has the largest rise + fall time. The authors argue that this characteristic greatly enhances the HT detection sensitivity of their method to capacitive loading effects over other delay testing methods, particularly for long paths and when considering process variation effects.

Delay variations introduced by process variation effects are cumulative, and, therefore, the HT-free boundaries or *margins* associated with standard delay methods must be increased for longer paths, which reduce their sensitivity to small, fixed-sized variations in delay introduced by HT. On the other hand, pulses will shrink when they encounter the capacitive load of an HT and will die out at the gate that was used to determine the minimum pulse width for the path (note: this assumes the HT insertion occurs before the gate that was used to define the minimum-sized pulse). Therefore, the authors argue that HT detection sensitivity remains constant and is independent of the length of the path. The embedded components needed for pulse generation and pulse detection can be designed as shown in Fig. 10.24a, b, respectively, and these components can be shared among multiple FFs, as shown for the pipelined architecture in Fig. 10.24c.

An algorithm is presented that uses n random test patterns that are each evaluated through simulation using k pulses of different widths. Test for paths that are able to propagate the pulse from a launch FF and to a capture FF is deemed valid. The authors refer to these paths as *single-path sensitizable*[5]). Simulations are again used to determine the minimum pulse width for each path using worst-case process models. HT are emulated on each node of every path using different capacitive loads to determine the minimum capacitance that succeeds in "killing" the propagating pulse.

The proposed method is validated using simulation experiments on a chain of NAND gates, a ripple carry adder and 4x4 multiplier. Process variations are modeled by changing threshold voltage (V_t) by +/− 10% globally and +/− 10% locally. The detection results for the chain of NAND gates are shown in Fig. 10.24d, with "pulse test" results corresponding to the proposed technique and "delay test" identifying results using a standard delay test strategy. The columns labeled "Min Cap Detected" represent the smallest HT capacitive load that was detectable for the paths of different lengths specified by the rows. The last column expresses the improvement in sensitivity of the proposed method over the standard delay test method and supports the claim that the pulse method remains sensitive to small

[5]Single-path sensitizable refers to paths that are hazard-free robust testable, indicating all side-inputs along the path must remain constant under both applied vectors.



Chain of NAND gates

Logic Depth	Pulse Test	Delay Test	
	Min Cap Detected	Min Cap Detected	Improvement of Pulse Test over Delay Test
5	880aF	1fF	1.13X
10	880aF	2fF	2.27X
16	880aF	3fF	3.40X
22	880aF	4.5fF	5.11X
38	880aF	9fF	10.22X
70	880aF	15f	17.04X

(d)

Fig. 10.24 (**a**) Proposed pulse generator, (**b**) pulse detector, (**c**) an example illustrating sharing within a pipelined architecture, and (**d**) simulation results comparing proposed technique (labeled pulse test) with standard delay test (labeled delay test) [56]

HT even when the path length becomes very large. Similar results are obtained for the other functional units as reported in [56].

10.5.11 Chip-Centric Calibration Techniques for HT Detection

The authors of [57, 59] propose HT detection methods which use actual path delay measurements, in contrast to PCMs and other types of on-chip test structures, as a mechanism to calibrate for global shifts and within-die process variation effects.[6] These methods represent variants of the chip-centric technique described in Sect. 4.2. Chip-centric techniques can potentially provide higher levels of sensitivity to HT because the path delays used in the detection method also serve as the basis for calibration. Moreover, by using chip-measured path delays to shrink the HT-free space, as depicted on the right side of Fig. 10.9, such methods can also simplify the development of a simulation-based *golden model*, as demonstrated in [59].

[6]Note the path delay technique described in [59] is based on the same concept presented earlier in [58] which uses leakage currents.

The authors of [57] **average** path delays measured from a set of chips to reduce the adverse impact of both inter-chip and intra-die process variation effects on HT detection sensitivity. The proposed *golden model* is based on hardware measurements of delays from HT-free chips, i.e., design and simulation data are not used to develop the HT-free space. The data collected from the chips is multidimensional. The authors use # to symbolize the chip number, P to represent the pattern (two-vector sequence) number, N_P to represent the number of patterns, α to identify functional unit outputs (Capture FFs), and N_α to represent the number of outputs. Calibration of inter-chip (global) process variation effects on path delays uses a *centering* operation in which the delays D under all patterns P to an output α for chip # are averaged and subtracted from each of the raw delays as given by Eq. 10.2. Therefore, the method uses the distribution of delays to each output α for calibration of the global *mean* shift in path delays that occurs within chip #. A second *centering* operation is then performed to further reduce intra-die variations which averages the globally calibrated delays to each output α across all chip outputs as given by Eq. 10.3. This chip-wide average is then subtracted from the raw path delays for a chip to provide a set of locally calibrated delays.

$$D_P(\alpha, \#) = D(P, \alpha, \#) - \frac{\sum_P D(P, \alpha, \#)}{N_P} \tag{10.2}$$

$$D_{P,\alpha}(\#) = D_P(\alpha, \#) - \frac{\sum_\alpha D_P(\alpha, \#)}{N_\alpha} \tag{10.3}$$

$$RP_{P,\alpha,\beta}(\#) = \frac{D_{P,\alpha}(\#)}{D_{P,\beta}(\#)} \tag{10.4}$$

$$Dg^{\#_{\text{test}}}_{P,\alpha,\beta} = \frac{RP_{P,\alpha,\beta}(\#_{\text{test}}) - \overline{RP}_{P,\alpha,\beta}(\#_{GM})}{\sigma_{P,\alpha,\beta}} \tag{10.5}$$

The ratios of two locally calibrated delays for a pattern P are used in the formulation of a *golden model*, each referred to as a *relative performance* metric, $RP_{P,\alpha,\beta}$ as given by Eq. 10.4. A matrix of relative performances is constructed for each chip #, and the mean value $RP_{P,\alpha,\beta}$ computed across all HT-free chips, N_{GM}, is used as the references for comparison of the $RP_{P,\alpha,\beta}$ values computed from the untrusted chips. A margin referred to as the *coefficient of irrelevance* is proposed for dealing with false-positive HT detections. It is defined as the standard deviation, $\sigma_{P,\alpha,\beta}$, of the $RP_{P,\alpha,\beta}$ computed using N_{GM} HT-free chips. The threshold that bounds the HT-free space is given by Eq. 10.5 and is referred to as a *distinguisher*.

The technique is validated using a set of four Xilinx Spartan FPGAs programmed with an AES-128 functional unit and modified in a second design to include one combinational and one sequential HT. The golden model is built using delays measured from the AES-128 without the HT. The **single-clock** scheme (or *clock sweeping* from Sect. 4.1.3) is used with a step size of 35 ps and a frequency range from 100 MHz to 121.2 MHz. Test vector selection is performed randomly, i.e.,

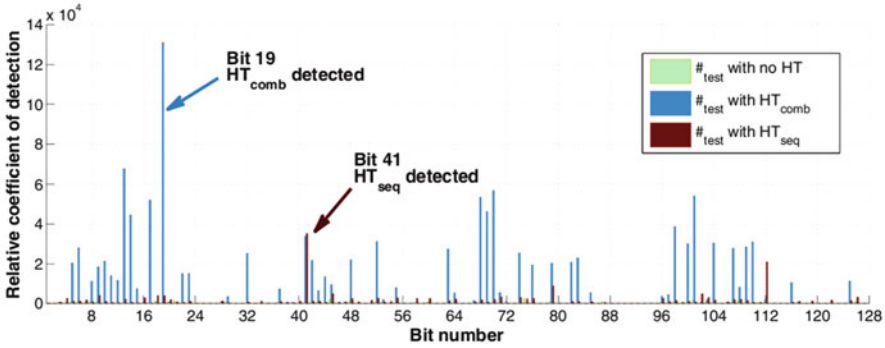


Fig. 10.25 HT detection results as presented in [57]

no test vector generation strategy is proposed. A set of 50 patterns (plaintexts) are used as the test vector set, and paths from all 128 bits of the AES are monitored. Path delays shorter than 8.25 ns (1/121.2 MHz) are ignored. The authors report on a subset of the *distinguishers*, in particular, the distinguishers which produced the maximum value for each of the 128 outputs when computed using the HT-free data and data from the two HT experiments. The results are shown in Fig. 10.25, with highlights indicating the outputs that provide the highest levels of confidence in detecting the two HT.

Although the technique proposed in [57] is demonstrated to work well, the averaging techniques that the authors employ do not deal directly with intra-chip process variations. The technique presented in [59] (discussed below), on the other hand, averages delays across chips for each path and two-vector sequence, instead of across vectors and outputs. Within-die variations have been shown to have a significant random component in each chip instance [60], and, therefore, a path-by-path averaging strategy is likely to be more effective in reducing unwanted intra-chip variations. Moreover, the strategy proposed in [57] only calibrates for the global shift in the *mean* values of path delays introduced by inter-chip process variation effects. The technique described in the following also considers *scaling* effects.

A **chip-averaging** HT detection method that calibrates for both intra-chip and inter-chip process variations and measures path delays using an on-chip time-to-digital converter (TDC) is proposed in [59]. The TDC was described earlier in reference to Fig. 10.8 in Sect. 4.1.3. The TDC provides approx. 25 ps of timing resolution, is very fast, e.g., no clock strobing or clock sweeping operation is required, and can be multiplexed and shared across a large number of the functional unit outputs. The method is also classified as **chip-centric** but unlike [57] does not depend on a set of golden chips. Rather, a *golden simulation model* is used to characterize the HT-free space. The development of the golden model requires only a **single nominal simulation** to be run for each of the applied two-vector sequences, and therefore the approach significantly reduces the level of effort and time required over previously proposed simulation-based golden model approaches.

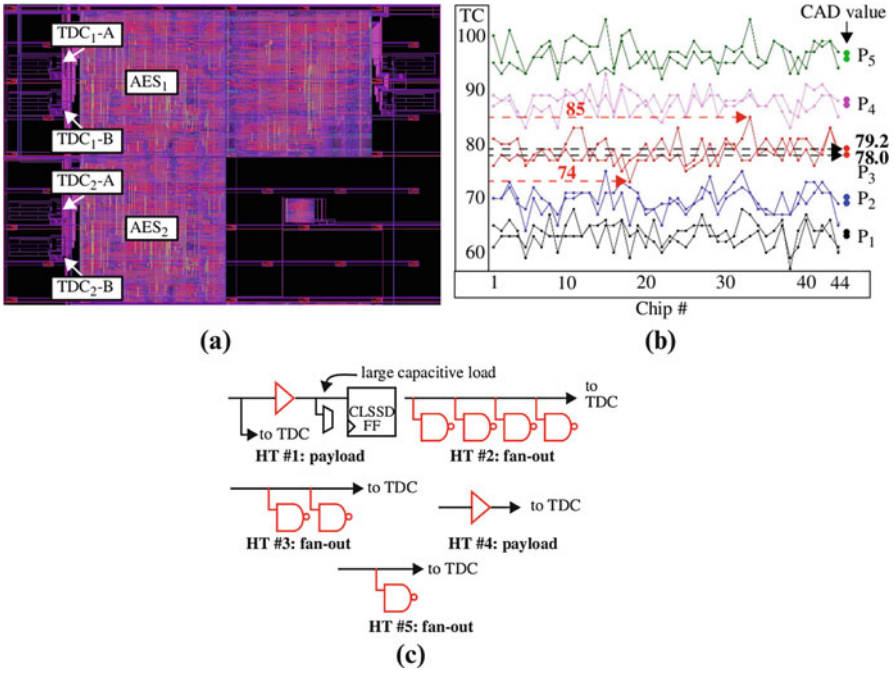


Fig. 10.26 (**a**) Chip layout showing 2 copies of the AES [26], (**b**) TCs from 44 chips with CAD values shown on far right, and (**c**) configurations of HT added to AES_2

This is possible because the calibration processes are geared toward deriving a nominal chip-averaged-delay (**CAD**) value for each path from hardware data, and therefore, process variation effects do not need to be accounted for in the golden model.

Calibration and chip averaging are designed to reduce performance differences and the adverse effects of process variations on delay while preserving any type of systematic variation that shows up in all (or a large subset) of the tested chips. Chip averaging leverages a key difference between random process variations and HT anomalies; random variations average to 0, while HT anomalies introduce systematic differences that survive the averaging process.

The authors validate the method using data collected from 44 copies of an ASIC fabricated in a 90 nm technology which has two exact copies of the layout of an AES functional unit, one representing the original design and one with five embedded HT. A layout of the chip showing the two copies of the AES and four instances of the TDC is shown in Fig. 10.26a. The two 8-to-1 multiplexers shown in the block diagram of the TDC from Fig. 10.8 connect to 15 of the 128 outputs of the AES (the 16th input is connected to the *Clk*). The two copies of the TDC in each AES instance allow signals propagating to 30 of the outputs of AES to be timed against the *Clk*.

The calibration process used to reduce inter-chip process variations is carried out in advance of the HT detection procedure on each chip separately. Similar to the HT detection process, calibration involves measuring delays from paths of various lengths within the functional unit on each chip. Unlike HT detection, the goal of calibration is to tune the control signals, *Cal0* and *Cal1*, of the TDC as a means of *shifting* (and *scaling*) the delay distribution obtained for each chip to a fixed mean value. From Sect. 4.1.3, the output of the TDC is a thermometer code (TC), i.e., an integer value between 0 and 120, that represents the relative delay difference between the *Clk* and the path being tested. The fixed mean value is set to the halfway point (60). By using the same fixed mean value for all chips, this process effectively standardizes the TCs, thereby eliminating most of the delay variations introduced by chip-to-chip process variation effects.

The chip-averaging technique is designed to remove the remaining intra-chip variation that exists in the path delays. Once the TDC is calibrated, a set of TDF-based vectors designed to test each possible HT site in a hazard-free fashion is applied to the chips. The HT detection method is applied once data from all or a large sample, e.g., 50 or more, chips is collected. A **chip-averaged-delay** (CAD) value is computed for each tested path by averaging the TC delays obtained from all chips. The CAD averages and ideally eliminates random within-die variations, making it possible to observe very small systematic differences which occur in the chip values but are not present in a Spice-level simulation of the *nominal* model.

As an illustration, Fig. 10.26b plots the raw TC values for 5 HT-free paths of different lengths. The x-axis lists the chip, 1–44, for each TC value on the y-axis. The two curves represent the data collected from each of the two nearly identical AES instantiations shown in Fig. 10.26a. The variations in the data points across chips and between the AES instantiations are what remain after calibration and are attributed to intra-die variations and measurement noise. The CAD values for each of the five paths are shown as the last point of the waveforms on the far right. The chip-averaging effect is reflected in the "closeness" of the two points computed from the 44 chips of each AES instantiation. The layout of the two AES instantiations is identical, and, therefore, ideally, the CAD values should be superimposed. Although this is not the case, the CAD values are closer than most of raw TC values for any given chip. This reduces the boundaries associated with the HT-free space, which in turn, improves the HT detection sensitivity of the proposed method.

The authors validate the detection sensitivity of the method by measuring the delay anomalies introduced by five layout-inserted HT. Figure 10.26c gives schematic-level diagrams illustrating the structure and insertion points of the HT, which are highlighted in red. Four fanout HT and one series-inserted HT are added to the layout of AES_2 by replacing filler cells and connecting the inputs and outputs of the HT as shown by the schematic. A *nominal* simulation model of the AES layout and TDC is created using Mentor Graphics Calibre XRC extractor and the foundry-provided models for the 90 nm technology in which the chips were fabricated. Transient simulations using Cadence Spectre are carried out to obtain the TC values associated with the nominal model.

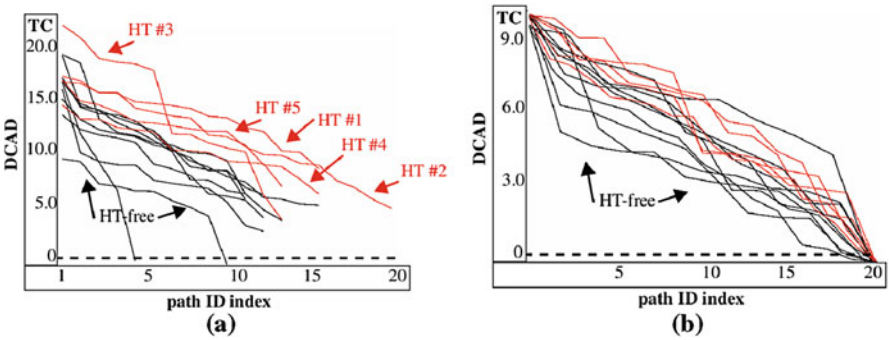


Fig. 10.27 (**a**) DCAD values of golden simulation model against HT-infested AES_2 and (**b**) DCAD value of golden simulation model against HT-free AES_1 from [59]

The graphs shown in Fig. 10.27 plot two sets of results, (a) plots the simulation nominal model data against the HT-infested AES_2 data, while (b) plots the simulation data against the HT-free AES_1 data for the same 20 paths. The y-axis plots a **DCAD** value, which is simply the difference between the simulation TC value and hardware-derived CAD values. HT that introduce larger anomalies therefore generate larger DCAD values. The paths are sorted left to right according to the magnitude of the HT delay anomaly, with the largest DCAD values on the left. The red curves represent data collected from paths that include one of the HT shown in Fig. 10.26c, while the black curves represent data from HT-free paths. The displacement of the red curves upward with respect to the black curves in (a) portrays the presence of the delay anomaly introduced by the HT. The curves in (b), on the other hand, show that the DCAD values for these same paths from AES_1 (which does not include the HT) are interleaved with the HT-free (black) curves.

10.6 Multiparameter Detection Methods

The authors of [61] leverage correlations between maximum operating frequency, F_{max}, and transient current, I_{DDT}, as a mechanism to enhance the HT detection sensitivity of I_{DDT}. Multiparameter side-channel analysis refers to the joint analysis of two or more circuit parameters, such as power and delay, as a means of accounting for process variation effects or to provide higher levels of confidence that an HT exists through corroborative evidence, or a lack thereof, from multiple signal sources. This concept is portrayed in Fig. 10.28a which plots I_{DDT} against F_{max}. Here, simulation experiments are used to show an embedded HT effect I_{DDT} because of additional HT switching activity but does not impact F_{max}. The mismatch in the correlation of I_{DDT} and F_{max} allows the HT to be identified in the "tampered" I_{DDT} curve that would otherwise not be possible. F_{max} is effectively used to track process variation effects. The distinction is blurred to some degree with the addition

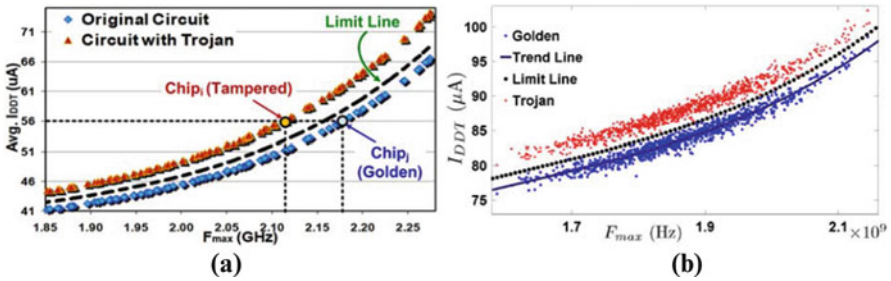


Fig. 10.28 (**a**) Correlations between I_{DDT} and F_{max} distinguish HT-free and HT-infested chips in the presence of process noise, (**b**) similar analysis with random intra-die variations added [61]

of random within-die variations as shown in Fig. 10.28b, but the correlation and benefit provided by F_{max} remain apparent in the displacement and separation of the red (HT) and blue (HT-free) data points. The authors note that any path or set of paths can be used for the correlation analysis to make it nearly impossible for the adversary to defeat the technique.

The authors propose a test vector generation strategy that first partitions the multi-module design into nonoverlapping functional blocks as a mechanism to amplify the HT I_{DDT} contribution (signal) over normal background I_{DDT} (noise). Vectors optimized to target HT nodes are selected and directed at testing one of the blocks while simultaneously minimizing activity in other functional blocks. The test vectors for I_{DDT} and a separate set for F_{max} are used in the proposed test flow to optimize correlations as shown in Fig. 10.28. Simulation and FPGA results are presented which validate their approach.

10.7 Conclusion

Hardware Trojans (HT) represent a serious threat and a significant challenge. Side-channel techniques, such as power and delay analysis, can be argued as the most sensitive and cost-effective strategies for detecting HT. This chapter surveyed a wide variety of delay-based approaches that have been proposed over the last decade. Important technical aspects and distinctions that characterize the proposed HT detection methods can be summarized as follows:

- Path delay measurement strategy for obtaining precise measurements of path delays:
  - Clock sweeping implemented by adjusting the frequency of applied clock
  - Two clock approaches which tune the phase between launch and capture clocks (clock strobing)
  - On-chip, embedded test structures which create (tunable) delay chains

- Test stimulus strategies for HT detection:
  - Random vectors
  - Vectors generated using the traditional transition fault delay (TDF) model
  - Vectors generated from a pseudo-TDF model targeting shortest sensitizable paths
  - Pulse stimulus-based techniques
- Approaches to account for process variation effects, both chip-to-chip and within-die:
  - HT-free space created from process simulation models
  - HT-free space created from data collected from golden (HT-free) chips (which are validated using destructive delayering techniques)
  - Simulation-derived HT-free space calibrated with hardware data from process control monitors (PCMs), ring oscillators (ROs), critical paths, etc.
  - Techniques which average path delays measured from (untrusted) chips and compared against (nominal) simulation models or golden HT-free chips
  - Techniques which correlate multiple side-channel signals
- Design-for-trust additions, modifications, and analyses to support HT detection methods:
  - Techniques which create ROs from functional unit paths
  - Techniques which add a distributed set of ROs designed to detect HT switching activity
  - Methods designed to find structural symmetry in path delays for comparison
  - Techniques which add symmetrical components to enable calibration using chip data
- Statistical HT detection methods:
  - Simple thresholding and linear regression-based methods
  - Advanced statistical analysis techniques which employ nonlinear regression, kernel mean matching, principle component analysis, multidimensional scaling, and convex hull construction
  - Ad hoc statistical techniques which leverage path delay differences, ratios, and other mathematical transformations

Taken collectivity, three critical features emerge as requirements for a fully specified and effective HT detection method:

- First, traditional manufacturing test methods are not capable of providing precise measurements of path delays, which is a requirement of nearly all proposed HT detection methods. Therefore, a paradigm shift is required in the way path delay testing is carried out by automatic test equipment and/or in the capabilities of design-for-testability support structures included on the chip. Several low-cost embedded test structures were described that support high-resolution on-chip measurements of path delays.

- Second, both within-die and chip-to-chip process variations pose significant limits on HT detection sensitivities and must be dealt with in a cost-effective manner. Golden model-based methods must be based on realistic assumptions regarding the availability of golden chips and the amount of simulation time and effort required to define the boundaries of a multidimensional HT-free space. Golden model-free methods must have validation techniques to guard against subversion by the adversary.
- Third, a low-cost test vector generation strategy must be developed that is effective at detecting subtle HT loading effects and which also provides high levels of HT coverage while minimizing test cost.

Achieving all of these goals is very challenging, but the commercial acceptance of path delay testing as a mainstream HT detection strategy critically depends on low-cost solutions to all three of these technical domains.

References

1. X. Wang, M. Tehranipoor, J. Plusquellic, Detecting malicious inclusions in secure hardware: Challenges and solutions, in *International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 15–19
2. R.S. Chakraborty, S. Narasimhan, S. Bhunia, Hardware Trojan: Threats and emerging solutions, in *International High Level Design Validation and Test Workshop*, 2009, pp. 166–171
3. M. Tehranipoor, F. Koushanfar, A survey of hardware Trojan taxonomy and detection. Des. Test Comput. **27**(1), 10–25 (2010)
4. R. Karri, J. Rajendran, K. Rosenfeld, M. Tehranipoor, Trustworthy hardware: Identifying and classifying hardware Trojans. Computer **43**(10), 39–46 (2010)
5. M. Beaumont, B. Hopkins, T. Newby, Hardware Trojans – prevention, detection, countermeasures, in *Department of Defense*, Australian Government, 2011
6. S. Bhunia, M. Abramovici, D. Agrawal, P. Bradley, M.S. Hsiao, J. Plusquellic, M. Tehranipoor, Protection against hardware Trojan attacks: Towards a comprehensive solution. Des. Test **30**(3), 6–17 (2013)
7. N. Jacob, D. Merli, J. Heyszl, G. Sigl, Hardware Trojans: Current challenges and approaches. IET Comput. Digit. Tech. **8**(6), 264–273 (2014)
8. S. Bhunia, M. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan attacks: Threat analysis and countermeasures. Proc. IEEE **102**(8), 1229–1247 (2014)
9. https://users.ece.cmu.edu/~koopman/des_s99/sw_testing/#reference
10. F. Wolff, C. Papachristou, S. Bhunia, R.S. Chakraborty, Towards Trojan-free trusted ICs: Problem analysis and detection scheme, in *Design, Automation and Test in Europe*, 2008
11. E. Love, Y. Jin, Y. Makris, Proof-carrying hardware intellectual property: A pathway to trusted module acquisition. Trans. Inf. Forensics Secur. **7**(1), 25–40 (2012)
12. M. Banga, M. Chandrasekar, L. Fang, M. Hsiao, Guided test generation for isolation and detection of embedded Trojans in ICs, in *Great Lakes Symposium on VLSI*, 2008, pp. 363–366
13. M. Banga, M. Hsiao, A region based approach for the detection of hardware Trojans, in *Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 40–47
14. R.S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, S. Bhunia, MERO: A statistical approach for hardware Trojan detection, in *Workshop on Crytographic Hardware and Embedded Systems*, 2009, pp. 396–410

15. M. Banga, M. Hsiao, A novel sustained vector technique for the detection of hardware Trojans, in *International Conference on VLSI Design*, 2009, pp. 327–332
16. R.S. Chakraborty, S. Narasimhan, S. Bhunia, Hardware Trojan: Threats and emerging solutions, in *International High Level Design Validation Test Workshop*, 2009, pp. 166–171
17. H. Salmani, M. Tehranipoor, J. Plusquellic, A layout-aware approach for improving localized switching to detect hardware Trojans in integrated circuits, in *International Workshop on Information Forensics and Security*, 2010
18. H. Salmani, M. Tehranipoor, J. Plusquellic, A novel technique for improving hardware Trojan detection and reducing Trojan activation time. Trans. VLSI Syst. **20**(1), 112–125 (2012)
19. D. Karaklajic, J.-M. Schmidt, I. Verbauwhede, Hardware designer's guide to fault attacks. Trans. VLSI Syst. **21**(12), 2295–2306 (2013)
20. P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in *Advances in Cryptology*, 1999
21. D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, B. Sunar, Trojan detection using IC fingerprinting, in *Symposium on Security and Privacy*, 2007, pp. 296–310
22. R. Rad, J. Plusquellic, M. Tehranipoor, Sensitivity analysis to hardware Trojans using power supply transient signals, in *Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 3–7
23. J. Aarestad, D. Acharyya, R. Rad, J. Plusquellic, Detecting Trojans though leakage current analysis using multiple supply pad I_{DDQ}s. Trans. Inf. Forensics Secur. **5**(4), 893–904 (2010)
24. M. Bushnell, V.D. Agrawal, Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits. Vol. 17 (Springer, 2000)
25. J. Kalisz, Review of methods for time interval measurements with picosecond resolution. Metrologia **41**(1) (2003)
26. C. Lamech, J. Aarestad, J. Plusquellic, R.M. Rad, K. Agarwal, REBEL and TDC: Embedded test structures for regional delay measurements, in *International Conference on Computer-Aided Design*, 2011, pp. 170–177
27. http://techinsights.com/
28. https://sstp.org/companies/analytical-solutions-inc
29. J. Soden, R. Anderson, C. Henderson, Failure analysis tools and techniques – magic, mystery, and science, in *International Test Conference, Lecture Series II "Practical Aspects of IC Diagnosis and Failure Analysis: A Walk through the Process"*, 1996, pp. 1–11
30. S.R. Nassif, Design for variability in DSM technologies, in *International Symposium on Quality Electronic Design*, 2000
31. J.-J. Liou, K.-T. Cheng, D.A. Mukherjee, Path Selection for delay testing of deep sub-micron devices using statistical performance sensitivity analysis, in *VLSI Test Symposium*, 2000
32. A.K. Majhi, V.D. Agrawal, Delay fault models and coverage, in *International Conference on VLSI Design*, 1998
33. Y.K. Malaiya, R. Narayanaswamy, Modeling and testing for timing faults in synchronous sequential circuits. Des. Test Comput. **1**(4), 62–74 (1984)
34. J.L. Carter, V.S. Iyengar, B.K. Rosen, Efficient test coverage determination for delay faults. in *International Test Conference*, 1987, pp. 418–427
35. G.L. Smith, Model for delay faults based upon paths, in *International Test Conference*, 1985, pp. 342–349
36. C.J. Lin, S.M. Reddy, On delay fault testing in logic circuits. Trans. Comput-Aid Des. **CAD-6**(5), 694–703 (1987)
37. D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N.S. Kim, K. Flautneret, Razor: Circuit-level correction of timing errors for low-power operation. Micro **24**(6), 10–20 (2004)
38. J. Li, J. Lach, Negative-Skewed shadow registers for at-speed delay variation characterization, in *International Conference on Computer Design*, 2007, pp. 354–359
39. X. Wang, M. Tehranipoor, R. Datta, Path-RO: A novel on-chip critical path delay measurement under process variations, in *International Conference on Computer-Aided Design*, 2008
40. Y. Jin, Y. Makris, Hardware Trojan detection using path delay fingerprint, in *Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 51–57
41. J. Li, J. Lach, At-speed delay characterization for ic authentication and Trojan horse detection, in *Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 8–14

42. D. Rai, J. Lach, Performance of delay-based Trojan detection techniques under parameter variations, in *International Workshop Hardware-Oriented Security and Trust*, 2009, pp. 58–65
43. X. Zhang, M. Tehranipoor, RON: An on-chip ring oscillator network for hardware Trojan detection, in *Design and Test in Europe*, 2011
44. J. Rajendran, V. Jyothi, O. Sinanoglu, R. Karri, Design and analysis of ring oscillator based design-for-trust technique, in *VLSI Test Symposium*, 2011, pp. 105–110
45. C. Lamech, J. Plusquellic, Trojan detection based on delay variations measured using a high-precision, low-overhead embedded test structure, in *Hardware-Oriented Security and Trust*, 2012, pp. 75–82
46. M. Li, A. Davoodi, M. Tehranipoor, A sensor-assisted self-authentication framework for hardware Trojan detection, in *Design, Automation & Test in Europe Conference*, 2012
47. D. Du, S. Narasimhan, R.S. Chakroborty, S. Bhunia, Self-referencing: a scalable side-channel approach for hardware Trojan detection, in *Cryptographic Hardware and Embedded Systems*, 2010, pp. 173–187
48. M. Potkonjak, A. Nahapetian, M. Nelson, T. Massey, Hardware Trojan horse detection using gate-level characterization, in *Design Automation Conference,* 2009, pp. 688–693
49. S. Wei, K. Li, F. Koushanfar, M. Potkonjak, Provably complete hardware trojan detection using test point insertion. in *International Conference on Computer-Aided Design*, 2012, pp. 569–576
50. S. Wei, M. Potkonjak, Malicious circuitry detection using fast timing characterization via test points, in *Symposium on Hardware-Oriented Security and Trust*, 2013
51. B. Cha, S.K. Gupta, Efficient Trojan detection via calibration of process variations, in *Asian Test Symposium*, 2012
52. B. Cha, S.K. Gupta, Trojan detection via delay measurements: A new approach to select paths and vectors to maximize effectiveness and minimize cost, in *Design, Automation & Test in Europe*, 2013
53. K. Xiao, X. Zhang, M. Tehranipoor, A clock sweeping technique for detecting hardware Trojans impacting circuits delay. Des. Test **30**(2), 26–34 (2013)
54. Y. Liu, K. Huang, Y. Makris, Hardware Trojan Detection through Golden Chip-Free Statistical Side-Channel Fingerprinting, in *Design Automation Conference*, 2014, pp. 1–6
55. N. Yoshimizu, Hardware Trojan detection by symmetry breaking in path delays, in *International Symposium on Hardware-Oriented Security and Trust*, 2014, pp. 107–111
56. S. Deyati, B. J. Muldrey, A. Singh, A. Chatterjee, High resolution pulse propagation driven trojan detection in digital logic: Optimization algorithms and infrastructure, in *Asian Test Symposium*, 2014, pp. 200–205
57. I. Exurville, L. Zussa, J.-B. Rigaud, B. Robisson, Resilient Hardware Trojans Detection based on Path Delay Measurements, in *International Symposium on Hardware-Oriented Security and Trust*, 2015, pp. 151–156
58. I. Wilcox, F. Saqib, J. Plusquellic, GDS-II Trojan detection using Multiple Supply Pad V_{DD} and GND I_{DDQ}s in ASIC Functional Units, in *International Symposium on Hardware-Oriented Security and Trust*, 2015
59. D. Ismari, C. Lamech, S. Bhunia, F. Saqib, J. Plusquellic, On detecting delay anomalies introduced by Hardware Trojans, in *International Conference on Computer-Aided Design*, 2016
60. W. Che, M. Martin, G. Pocklassery, V.K. Kajuluri, F. Saqib, J. Plusquellic, A privacy-preserving, mutual PUF-based authentication protocol. Cryptography **1**(1) (2016)
61. S. Narasimhan, D. Du, R.S. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, S. Bhunia, Multiple-parameter side-channel analysis: A non-invasive hardware trojan detection approach, in *International Symposium on Hardware-Oriented Security and Trust* (IEEE, Anaheim, 2010), pp. 13–18

Chapter 11
Reverse Engineering-Based Hardware Trojan Detection

Chongxi Bao, Yang Xie, Yuntao Liu, and Ankur Srivastava

11.1 Introduction

Nowadays, most integrated circuit (IC) designers do not keep an in-house foundry for economic reasons. Instead, they often outsource their designs to outside foundries (maybe untrusted) for fabrication. This leaves the possibility that an adversary in the untrusted foundry may maliciously modify the original circuit to do evil. Therefore, although design outsourcing helps to lower costs and reduce time-to-market pressure, it also leads to security problems described above. The malicious modifications made to the original circuit are called hardware Trojans (HTs). HTs can cause a number of damages to the original design, including a change in functionality, a reduction in IC reliability, leakage of valuable information from the IC, and denial of service [1]. Depending on the applications, the consequences of HTs range from loss of profit if used in consumer-electronic devices to life-threatening if used in military devices.

To reduce the devastating consequences hardware Trojans may bring, researchers have proposed different approaches to detect them. Among all approaches, test-time detection approaches [2–5] have drawn the greatest amount of attention from researchers because (1) they are noninvasive and (2) they usually require fewer resources compared to other techniques. In many functional testing-based approaches, functional and/or side-channel behavior of suspect ICs are compared to a "golden model" that represents the expected behavior of a Trojan-free IC. If the functionality of the suspect IC deviates sufficiently from that of the golden model, it is classified as Trojan infected. Although some of these approaches have been quite successful, it is still an open problem how to obtain such golden models/data.

C. Bao (✉) • Y. Xie • Y. Liu • A. Srivastava
University of Maryland, College Park, 8223 Paint Branch Drive, 20742 College Park, MD, USA
e-mail: chongxi.bao@gmail.com; yxie.ece@gmail.com; ytliu@umd.edu; ankurs@umd.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_11

Motivation. In prior works such as [6], it is suggested that reverse engineering (RE) should be used to verify that an IC is Trojan-free and data extracted from this IC can subsequently be used as golden models. However, the reverse-engineering process is actually very complex, time-consuming, and error-prone. RE process usually consists of five steps [7]: decapsulation, delayering, imaging, annotation, and schematic creation. After the first three steps, images of the physical layout of the IC under test are taken. Then during the last two steps, netlists for the circuit are extracted based on the images. One way (and also a naïve way) to use RE to detect hardware Trojans is to apply all these five steps and compare the resulting netlist with a golden netlist. However, this approach is flawed for the following reasons. First, some Trojans will be missed out by only comparing netlists. For example, parametric Trojans [8] differ from a golden IC only in some circuit parameters such as wire length. Comparing at the netlist level won't detect them. Second, the above naïve approach requires unnecessarily excessive human effort and is very time-consuming because generating the netlist via the last 2 RE steps requires extensive human interaction.

Organization. The fundamentals of RE of integrated circuits are explained in Sect. 11.2. The applications of RE in hardware Trojan detection are introduced in Sect. 11.3. In Sect. 11.4, we propose a novel RE-based hardware Trojan detection method which also involves support vector machines (SVMs). A design-for-security approach is proposed in Sect. 11.5. The chapter is concluded in Sect. 11.6.

11.2 Reverse Engineering of Integrated Circuits

11.2.1 Introduction to Reverse Engineering

Reverse engineering (RE) of integrated circuits is the process of examining and analyzing the internal structures of the chip to extract the schematic or reveal some information about the fabrication process. This is often done by removing the chip package and stripping off the die layer by layer. Torrance and James [7] introduce a state-of-the-art RE flow which contains the following steps:

1. *Decapsulation:* The package of the die is opened, and the die is removed from its package.
2. *Delayering:* The layers of the die are stripped off one at a time using chemical methods. The new surface is polished to keep planar.
3. *Imaging:* Extremely high-resolution images of each layer are taken using a scanning electron microscope when the layer is exposed which are then stitched together into a complete image of the layer. After the images of all layers are obtained, the images are aligned so that the interlayer structures, i.e., contacts and vias, are lined up.

4. *Annotation:* The transistors and other structures (interconnects, vias, etc.) in the chip are annotated manually or automatically.
5. *Schematic Generation:* The schematic of the circuitry in the chip is generated based on what was obtained in the last step as well as other information.

Even though the RE process is highly automated nowadays, all of the above steps are still error-prone. In the first two steps, the die is physically manipulated which will affect the structure of the exposed layer and introduce additional noise in the measurements. The chemical process may even affect the lower layers. Annotating the structures and generating the netlist are also challenging. Even advanced image recognition software sometimes make mistakes, and experienced analysts are often needed [7].

11.2.2 Applications of Reverse Engineering

Many parties within the supply chain of integrated circuits are interested in RE.

- *Circuit designer.* The circuit designer may wish to reverse-engineer the fabricated chip that has returned from the foundry and see whether the fabricated chip has incurred any malicious modifications.
- *Foundries.* Foundries may wish to reverse-engineer chips manufactured by their competitors to analyze the fabrication process that they use.
- *IP vendors.* The IP vendors may reverse-engineer some suspicious chips to see whether there is an unauthorized use of their IP.

11.3 Hardware Trojan Detection Using Reverse Engineering

11.3.1 General Information

Researchers in both academia and industry have proposed different approaches to detect hardware Trojans (HTs). These methods fall into the following four categories:

- *Design-time approaches* have two types: (1) formal verification and (2) design-for-security (DFS). Formal verification [9] requires that the design house proves certain security properties hold in the design. DFS approaches (see [10]) are aimed at increasing the controllability and observability of ICs in order to aid test-time detection approaches.
- *Test-time approaches* consist of tests in addition to normal IC post-manufacturing tests are the most widely studied approaches in the literature. There are two types: (1) functional testing and (2) side-channel fingerprinting [5]. Functional testing approaches [2, 4] aim to detect HTs that change the functionality (primary

outputs) of the IC from the intended one. Side-channel fingerprinting [11, 12] is an alternative approach that measures side-channel signals (timing, power, electromagnetic, etc.) and uses them to distinguish genuine ICs from Trojan-infected ones.

- *Run-time approaches* add circuitry that monitors the behavior/state of a chip after it has been deployed. If deviation from the expected golden behavior is detected, additional circuitry can disable the chip or bypass the malicious logic before the HT can do any damage. Most test-time and run-time approaches assume that a TF chip, also known as a golden chip (a chip with intended functionality and behavior), is available to compare with. They suggest RE be used to obtain such a golden chip [6].
- *RE-based approaches* apply the RE process (discussed above) to ICs in order to detect HTs. One naïve approach is to extract the design/netlist by applying all five RE steps and compare it with an intended (golden) netlist. However, such approach is not only time-consuming but also destructive. They have been pursued the least for HT detection. Their main use of RE has been to verify the TF chips used in the golden model development [6]. Another type of RE-based approaches [13–15] develops a more efficient and robust RE approach without extracting netlists altogether. They only utilize the first three steps of reverse engineering (decapsulation, delayering, and imaging) to obtain internal images of each layer (poly, metal1, etc.) for the ICs. By omitting reverse engineering (RE) steps 4–5, they save lots of unnecessary effort. The images recovered represent the physical structures and layout of the ICs. Then, ICs are classified using these images and support vector machines. In contrast to the naïve RE approach which requires manual effort, these SVM supported RE-based approaches are fully automated and more efficient in terms of computational and storage resources.

11.3.2 Advantages of Applying Reverse Engineering to Trojan Detection

RE-based approaches have several advantages:

1. It does not need a golden chip to compare with. Instead, it can verify that a chip is golden, serving as foundations for many other test-time approaches.
2. It is able to detect small Trojans and parametric Trojans. These Trojans are hard or even impossible to detect using other techniques.

11.3.3 Challenges of Applying Reverse Engineering to Trojan Detection

For naïve RE-based HT detection approach, the challenges are as follows. First, some Trojans (e.g., parametric Trojans) can actually be missed by only comparing netlists. Second, the effort required by this naïve approach is excessive. RE steps 4–5 are time-consuming and require manual input for annotation and schematic read-back. Furthermore, Plaza and Markov [16] stated that RE at advanced technology nodes such as 22 nm is not feasible without access to the gate-level netlist. This means that extracting the gate-level netlists from the layout images (RE steps 4–5) will be infeasible in technologies beyond 22 nm.

11.4 Reverse Engineering-Based Hardware Trojan Detection Using SVM

In Sect. 11.3, several reverse engineering-based hardware Trojan detection approaches are reviewed. In this section, we will go over one detection method in greater details, i.e., reverse engineering-based hardware Trojan detection approach using support vector machine (SVM) [13]. This approach is very efficient and robust. It involves a machine learning approach that classifies ICs as Trojan-free and Trojan-inserted based on features extracted from the IC images. The approach essentially eliminates the annotation and schematic recover steps in the reverse-engineering (RE) process, which can save lots of unnecessary effort.

11.4.1 Problem Statement

Assume that ICs fabricated by one or more untrusted foundries are given. The problem is to determine which ICs are Trojan-free (TF) and which have Trojan-inserted (TI). Three types of Trojans are considered:

- *Trojan Addition (TA)*: In this type of Trojans, malicious transistors, interconnects, and gates are added into the original circuit.
- *Trojan Deletion (TD)*: In this type of Trojans, components such as transistors, interconnects, and gates are maliciously removed, usually causing functional error or denial of service.
- *Trojan Parametric (TP)*: In this type of Trojans, circuit parameters such as wire width are modified by an adversary to change circuit property such as timing behavior.

Examples of TF, TA, TD, and TP are shown in Fig. 11.1. Note that the above problem can be viewed as an instance of the classification problem defined as

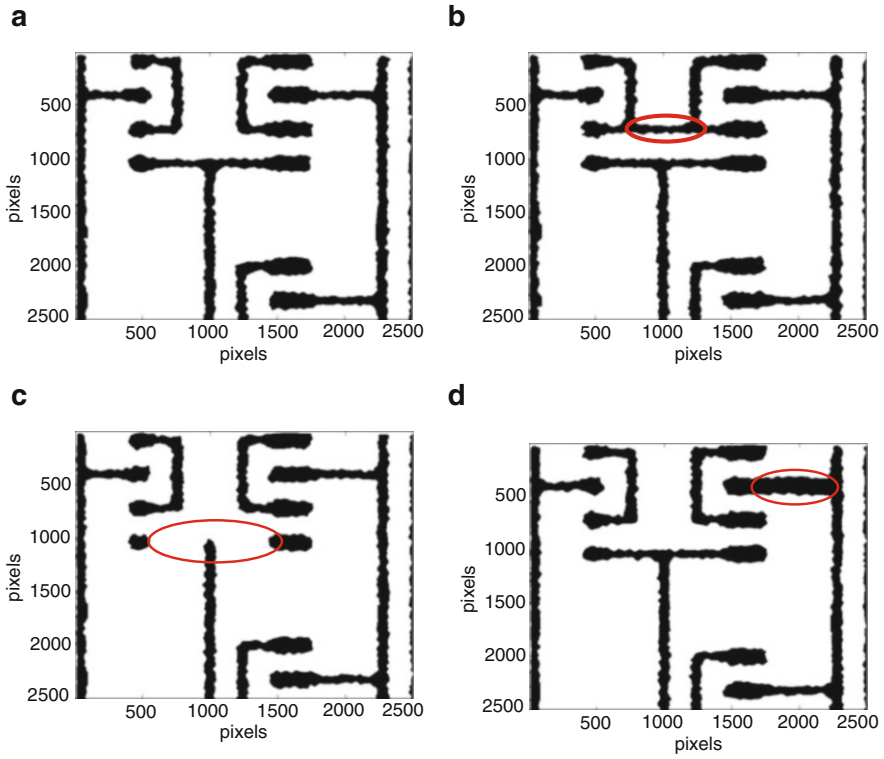


Fig. 11.1 Example of three kinds of trojans. SEM image of metal1 layer is shown. (**a**) Trojan free case (TF). (**b**) Trojan addition case (TA). (**c**) Trojan deletion case (TD). (**d**) Trojan parametric case (TP)

follows. There are two classes of objects, denoted by C_0 and C_1, respectively. Each object is represented by a feature vector $\mathbf{x} = (x_1, \ldots, x_n)$ where x_i denotes the ith feature, $x_i \in \mathbb{R}$ and n denotes the # of features. Given an unknown object A, the problem is to determine the correct class of A. In our case, objects are chips under test, and TF represents class C_0, while the TI cases (TA, TD, and TP) represent class C_1.

Prior works such as [6] suggest reverse engineering (RE)-based methods be used to identify Trojan-free ICs in order to develop the golden models for other HT detection approaches. However, there was not any work devoted toward performing this classification prior to [13]. One could only assume that a golden netlist exists and naïvely apply all steps in the reverse-engineering 180 process to extract netlists from all unknown chips. There are some issues with this naïve approach. First, some Trojans (e.g., parametric Trojans) can actually be missed by only comparing netlists. Second, the effort required by this naïve approach is excessive. Schematic recovery step in the RE process is time-consuming and requires manual input for annotation and schematic read-back. To address these issues, [13] introduced a more

efficient and robust RE approach for solving the above classification problem. In the work, extracting netlists is avoided altogether. Instead, a one-class SVM-based approach is developed. One-class SVM makes a classification decision based on features extracted from the IC images. The salient contribution of [13] is the ability to detect all the above Trojan cases automatically and efficiently. We will introduce implementation details next.

11.4.2 Proposed Approach

11.4.2.1 Overall Algorithm

The approach takes as input the golden layout, N chips to classify, and some learning parameters governing learning accuracy. It will output a label for the circuit, which is either Trojan-free or Trojan infected. The N chips undergo only decapsulation, delayering, and imaging steps in the reverse-engineering process, and as a result, images of each layer for all N chips are produced. Then those images belonging to the same layer in a chip are divided into nonoverlapping grids. Physical features are extracted from each grid in all chips. Then a one-class SVM classifier is trained for each layer using a subset of the chips (*possibly even just one chip*). After training, all the grids in the one layer are classified using the classifier corresponding to that layer. Each grid is either labeled as Trojan-free (TF) or Trojan-inserted (TI). Finally, based on the label for all the grids within a chip, a label for the whole chip is determined.

The culprit of the above approach is to solve the classification problem defined in Sect. 11.4.1. Many machine learning approaches can perform classification. Support vector machine (SVM) is chosen because of its high accuracy and easy to tune nature [17]. Generally, classification via SVM usually involves two phases:

- *Training:* During the training phase, SVM takes as input a training dataset $DS_T = \{(\mathbf{x}_j, y_j) \mid j = 1, \ldots, |DS_T|\}$ where $\mathbf{x}_j$ and y_j are the jth feature vector and its class label (C_0 or C_1). Then it solves an optimization problem that relies on the dot product of feature vectors to find a decision boundary$\boldsymbol{\omega}$ which separates the feature vectors of class C_0 from class C_1 so that as many objects from DS_T as possible are correctly classified.
- *Classification:* In the classfication phase, SVM takes as input a feature vector $\mathbf{x}^*$ of an unknown object and classifies it based on which side of the decision boundary $\boldsymbol{\omega}$ that $\mathbf{x}^*$ resides.

11.4.2.2 Feature Selection

As discussed above, we break the images (layouts) into smaller nonoverlapping grids as shown in Fig. 11.2. Features are extracted from each grid by comparing the corresponding grids of the IC under test (ICUT) with the golden layouts grids (which we know from design).

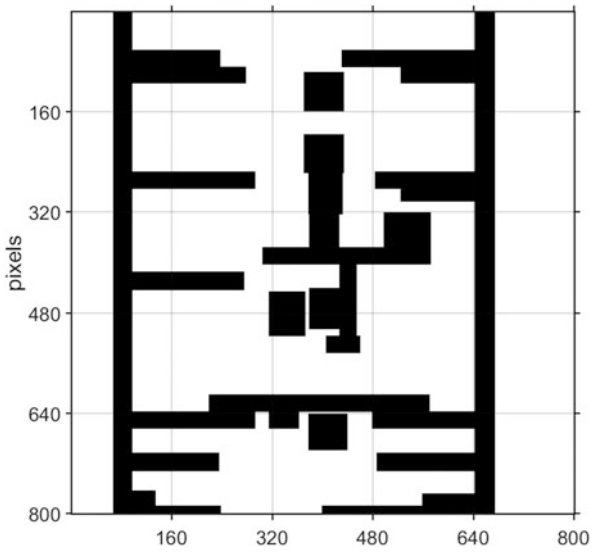


Fig. 11.2 Gridding techniques. Part of metal1 layer of s298 benchmark with gridsize 160×160 pixels is shown

We select several features which are determined based on area and centroid differences between the RE and golden layouts. To be specific, we calculate the percentage of the layout that is added to/removed from the golden layout. We also calculate the centroid difference (in X and Y direction) between actual layout and golden layout. These features should be able to capture most of the differences between actual layout and golden layout.

11.4.2.3 Final Classification

Once each grid has been classified, we can classify the whole chip accordingly. Note that if we classify a chip as Trojan-infected only because it has a few sparse Trojan-infected grids, the false-positive rate will be very high due to imperfections in the reverse-engineering process. Instead, we look at the number of grids classified as Trojan-inserted (TI) and their location and then classify the whole chip. More specifically, we assume that in order for the chip to be classified at TI, there must be at least n neighboring TI classified grids in the chip. Both horizontally adjacent grids and vertically adjacent grids are considered neighboring grids in this context. The intuition behind it is that malicious modifications tend to be continuous. In order to be functional, these modifications are connected either to one layer or to some other malicious modifications through neighboring layers. The maximum number of connected negative grids reflects the level of deviations from the golden layout existed in the chip under test.

Table 11.1 Benchmarks used in the experiments

Benchmarks	#gates	Source	#training chips	Training time(s)
b18	122,559	ITC99	1	13,054
s27	57	ISCAS89	20	121
s298	283	ISCAS89	5	175
s5378	3455	ISCAS89	1	351
s15850	10,984	ISCAS89	1	1032
s38417	30,347	ISCAS89	1	3055

Table 11.2 Chip classification accuracy rate averaged over 500 trials

Benchmark	TF (%)	TA (%)	TD (%)	TP (%)
s27	100	100	100	99.8
s298	100	100	100	99.6
s280	100	100	100	100
s15850	100	100	100	100
s38417	100	100	100	99.8
b18	99.4	100	100	100

11.4.3 Experiments and Results

Benchmarks. The performance of the proposed approach is tested on six publicly available benchmarks. These benchmarks are drawn from ISCAS89 and ITC99 are all synthesized using Cadence RTL Compiler with Synopsys 90 nm generic library. The gate count of each benchmark given by the Cadence tool is listed in Table 11.1.
Results. We list the detection accuracy averaged over 500 trials are listed in Table 11.2. Clearly, the proposed method can detect three kinds of malicious modifications, including Trojan addition, Trojan deletion, and parametric Trojans, with high accuracy. It is also noteworthy to point out that the method can recognize Trojan-free chips reliably (low false-positive rate).

11.5 Design-for-Security Approaches

We have reviewed several reverse engineering-based hardware Trojan detection approaches in Sects. 11.3 and 11.4. In this section, we explore security-aware design strategies. Note that although test-time and run-time hardware Trojan detection have been well-studied, design strategies remain a vital link in keeping hardware secure [18]. These strategies either help to prevent the insertion of hardware Trojans or to facilitate the detection of hardware Trojans in test-time and run-time. Combating hardware Trojan attacks is not an easy job, and the cooperation between design-time, test-time, and run-time approaches is always preferred.

Many reverse engineering-based hardware Trojan detection approaches including the one introduced in Sect. 11.4 extract some physical features from the layout and decide whether the Trojan is present on the chip depending on those features. Since there are many ways to lay out a chip, one design strategy can be choosing a layout that is more sensitive to hardware Trojan insertion (we will define sensitivity later in this section). As digital circuits are built (synthesized) using standard cells, the above strategy translates to finding the best set of standard cells, in a given technology library, that are more sensitive to Trojan insertion. Then intuitively, when the circuit is synthesized on that subset of standard cells, the resulting chip is more sensitive to hardware Trojan insertion. We will formally define this problem and give one solution in this section.

11.5.1 Problem Definition and Challenges

Motivation. We will use the hardware detection approach introduced in Sect. 11.4 as our post-manufacturing hardware Trojan detection method. We are interested in finding any design-time strategies that can make detection of Trojans easier using that method.

As a motivating example, we choose one benchmark, s298, from ISCAS89 [19]. We select two subsets of standard cells from Synopsys 90 nm generic library, namely, {DFFX1, INVX4, NAND2X1, NOR2X0} and {DFFX1, INVX2, NAND2X4, NOR2X2}. We synthesize the benchmark using these two subsets, respectively, and generate two synthesized designs. For each design, we randomly select one standard cell from it as the attack target and make several malicious modifications including structure deletion and addition to that target at the layout level. These modified designs will serve as the Trojan-infected designs. In total, for each design, random selection of the attack target is done 10 times, and each time 10 different modifications are applied to it, making the total number of Trojan designs 100 for each synthesized design. We then apply training and classification techniques introduced in [13] to both designs. The training is done on five Trojan-free chips. After classification of each grid in the IC is done, the Trojan miss rate (TMR) and the false-positive rate (FPR) defined below averaged over 100 Trojan designs are recorded. The area and leakage power are also reported in Table 11.3. Note that no timing constraints are specified during the synthesis.

$$\text{TMR} = \frac{\#\text{grids with malicious modifications and are } \textbf{NOT} \text{ detected}}{\#\text{grids that have malicious modifications}} \tag{11.1}$$

$$\text{FPR} = \frac{\#\text{grids without malicious modifications but are mislabeled}}{\#\text{grids that have no malicious modifications}} \tag{11.2}$$

This table shows that two designs have a significant difference in Trojan miss rate. In general, we can detect 41% more malicious modifications made to the

Table 11.3 Trojan miss rate, false-positive rate, area, and leakage power values of two benchmarks. Numbers in parenthesis are ratios of values in two designs

	TMR (%)	FPR (%)	Area (μm^2)	Power (nW)
Design 1	33.14 (1.41×)	0.35	1167	6622
Design 2	23.55	0.43 (1.21×)	1403 (1.20×)	9122 (1.38×)

second design than those made to the first design. However, this does not come for free. Design 2 also has a higher false-positive rate and incurs 20% overhead in area and leakage power. We summarize our findings from the above motivating examples:

- Malicious modifications made to some standard cells in a given library may be easier to be detected than those made to other standard cells. Put in another way, some standard cells are more sensitive to malicious modifications and are thus favored over others in terms of Trojan detection.
- Though design 2 has a higher false-positive rate (FPR) which is 0.43%, it is negligible. Moreover, since we intend to use the technique to identify golden chips, what is more important is to decrease the Trojan miss rate, which is also the false-negative rate. Because false-positive rate and false-negative rate cannot be decreased at the same time, we will neglect FPR and only focus on TMR.
- The increase in Trojan detection accuracy usually leads to extra area/power overhead.

This motivates us to find a "best" subset of standard cells such that if the circuit is synthesized on it, detection of Trojans using [13] will be the most accurate, while area/power/timing overheads are acceptable. However, we do not want to write our own EDA tool because it is tedious and error-prone. Instead, we are interested in modifying existing EDA tool flow to make it security-aware so that our approach can be readily used. We will define this problem formally next.

Problem Definition. Given a technology library l consisting of a set of n standard cells such that $l = \{C_1, C_2, \ldots, C_n\}$, a circuit in the form of RTL code or netlist denoted by d, an area upper bound A_{up}, leakage power upper bound P_{up} and some timing constraints $t_1, t_2, \ldots$, find a subset of l, namely, $l_s = \{C_{s_1}, C_{s_2}, \ldots, C_{s_i}\}$, such that when we synthesize d on l_s, malicious modifications made to it will be the easiest to be detected (among all possible subsets l_s) given that the following constraints are met:

$$\begin{aligned} &l_s \text{ is universal, } t_1, t_2, \ldots \text{ are satisfied} \\ &\text{area} \leq A_{\text{up}}, \text{ leakage power} \leq P_{\text{up}} \end{aligned} \tag{11.3}$$

Challenges. Note that the above problem is actually an optimization problem where the possible solution space is the power set (the set of all subsets) of l, the objective is to maximize Trojan detection accuracy, and the constraints are defined by (11.3).

We argue that the following challenges have to be addressed before solving the above problem:

- The objective function is vague. We have to define mathematically Trojan detection accuracy.
- In order to find the best subset of standard cells, a characterization of each standard cell has to be done.
- The possible solution space is exponential. For example, the saed90nm library has 340 standard cells. Thus, its power set contains $2^{340} = 2.23 \times 10^{102}$ elements. Moreover, for each possible solution, we need to run the entire synthesis flow to estimate the area, power, etc. thereby making the approach highly complicated. Obviously, an exhaustive search is computationally prohibited, and heuristics are needed.

11.5.2 Our Proposed Method

11.5.2.1 Characterization of Each Standard Cell

Before proposing our method of standard cell selection, we first investigate what causes the difference in sensitivity to malicious modifications between different standard cells. Reverse engineering-based Trojan detection approaches such as [13] usually extract several features like centroid difference, area of difference, etc. from each grid. Since different primitive structures (such as "T" shape, "L" shape, "F" shape, etc.) in a grid have different centroid patterns, they may have different sensitivities to malicious modifications. Therefore, in order to characterize the sensitivity of each standard cell, we have to characterize its building blocks, which are these primitive structures first.

We identify 16 possible structures (shown in Fig. 11.3) in a grid as primitives. Note that though these 16 primitives are not exhaustive, our experiments show that over 90% of the grids contain exactly one of them. Therefore, they are a good representative of primitive structures. We will synthesize several benchmarks, apply gridding, and use the method in [13] to train one classifier for each benchmark. We say a grid is of type $i (1 \leq i \leq 16)$ if it contains only ith primitive. Otherwise, we say it is of type 17. After training, we make some malicious deletions and additions to these benchmarks and use the classifier to classify each grid. We then calculate the Trojan miss rate (TMR) defined by Eq. (11.1) for each type of the grid, respectively. We let p_i $(1 \leq i \leq 17)$ denote the TMR of grid type i.

After the characterization of each primitive structure is done, given a technology library, we can apply the same gridding to each standard cell and label each grid as 1–17 using the same way as above. Say that there are n_i grids of type i $(1 \leq i \leq 17)$ in the standard cell. We define below CTMR (cell Trojan miss rate) which measures the average probability that a malicious modification is not detected if it happens at one random grid of the cell.

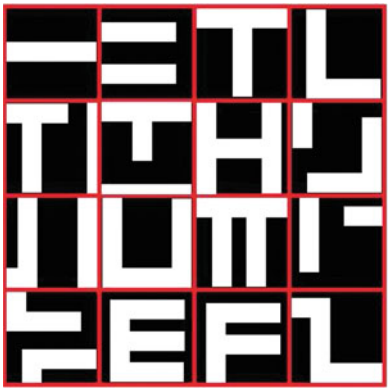


Fig. 11.3 16 primitive structures

Table 11.4 CTMR and actual Trojan miss rate of some standard cells

Standard cell	CTMR (%)	TMR (%)
NAND2X0	36.08	27.32
NAND2X1	43.08	39.33
NAND2X2	38.31	32.83
NAND2X4	33.18	25.38

$$\text{CTMR} = \frac{\sum_{i=1}^{17} n_i \times p_i}{\sum_{i=1}^{17} n_i} \tag{11.4}$$

By definition, the smaller CTMR a cell has, the easier it is to detect malicious modifications made to it. To see this, we measure the actual Trojan miss rate of all NAND2 standard cells by synthesizing a NAND2 gate using these standard cells respectively, making some malicious modifications to the synthesized design and calculating TMR defined by Eq. (11.1). We also calculate CTMR of these standard cells and list them in Table 11.4. The results indicate that CTMR is a great metric for cell's sensitivity to malicious modifications.

11.5.2.2 Mathematical Description of the Objective Function

Now that we have a good metric for cell's sensitivity to malicious modifications, we can define a design's sensitivity accordingly. Suppose a circuit is synthesized on a technology library and the synthesized design contains k_i $(1 \leq i \leq n)$ standard cell C_i. Let $l = \{C_1, C_2, \ldots, C_n\}$. Then we calculate this synthesized design's Trojan miss rate (DTMR) as below:

$$\text{DTMR}(l) = \frac{\sum_{i=1}^{n} k_i \times \text{CTMR}_i}{\sum_{i=1}^{n} k_i} \tag{11.5}$$

where CTMR_i is the Trojan miss rate for cell C_i (defined by Eq. 11.4). We use DTMR as the metric to measure the design's sensitivity to malicious modifications. The smaller DTMR a design has, the easier it is to detect malicious modifications made to this design. Note that when the circuit is fixed, DTMR is only determined by a subset of standard cells, denoted by l, on which the circuit is synthesized. Thus, we write DTMR as a function of l.

We can reformulate the problem in Sect. 11.5.1 as:

$$\min_{l_i \in 2^l} \ \text{DTMR}(l_i) \tag{11.6}$$

subject to constraints defined in Eq. (11.3). Here 2^l means the power set (the set of all subsets) of the given technology library l. The subset of standard cells denoted by l_{opt} that leads to the optimal value of the above optimization problem is what we want. We will call l_{opt} the *solution* to the above problem.

11.5.2.3 Steepest Descent Method

As stated in Sect. 11.5.1, an exhaustive search is computationally prohibited. Heuristics are needed to reduce the search space and find a near-optimal solution. We will, instead, adopt the idea of the steepest descent method (SDM) to solve the above problem. To find a local minimum of function f, the general SDM works as follows:

1. Find an initial solution x_0 and set $x_{\text{old}} = x_0$.
2. Calculate the gradient at the current solution and construct a new solution $x_{\text{new}} = x_{\text{old}} - \lambda \nabla f(x_{\text{old}})$ where λ is the step size that controls the convergence rate and should be kept relatively small.
3. If $f(x_{\text{new}}) < f(x_{\text{old}})$, then we set $x_{\text{new}} = x_{\text{old}}$ and go back to step 2. Otherwise, a local minimum point x_{old} has been found.

Note that the performance of SDM depends heavily on the initial solution. However, when the initial solution and the step size are chosen carefully, it can lead to very good solutions. Therefore, we will adapt its idea to solve our problem. But applying SDM to our problem has the following challenges, some of which are due to the nature of the method and some are unique to our problem.

- The performance of the SDM relies heavily on the choice of the initial solution. How to select the initial solution should be carefully investigated.
- The SDM is based on recursively updating the current solution to get a better solution. How to do this efficiently and effectively has yet to be examined.
- The SDM only works where the function to be optimized is differentiable. However, in our problem, it is a discrete function. We have to define the gradient accordingly.

We will address these challenges in the following text.

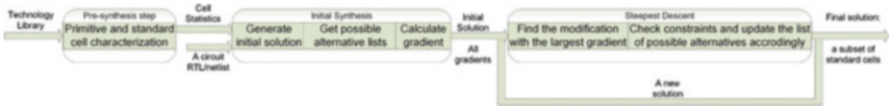


Fig. 11.4 Overall block diagram of our proposed method

11.5.2.4 Our Proposed Algorithm

The overall proposed flow is shown in Fig. 11.4. We will explain each step as well as the intuition behind the step.

Choice of initial solution. We can make up an arbitrary initial solution, which is to choose an arbitrary subset of standard cells. However, there is no guarantee that synthesizing the circuit d on this subset will meet area/timing/power constraints. In fact, the chance is very low. However, we can obtain an initial solution in a better way. We synthesize the design on the whole library applying all timing constraints. This will give us a list of standard cells used together with the area and the leakage power of the design. This is a baseline design, and its area and leakage power are the minimum values we can get given that all timing constraints are met. So it must be one feasible solution (if this design does not meet the area and power constraints, then no design will meet those constraints and no feasible solution exists) and is a very good point to start with. We will set the list of standard cells used in that synthesis as our initial solution.

How to modify the current solution. In the context of our problem, the modification is done by adding/removing several standard cells into/from the current solution. This is where the exponential complexity comes in because there are exponential numbers of ways of adding/removing standard cells to/from the old solution. To reduce the computation complexity, we make a key observation here.

Typically in a library, there are multiple standard cells with different drive strength that implement exactly the same logic function. For example, AND2X1, AND2X2, and AND2X4 all implement logic AND function of two operands: $Out = In1\&In2$ but with increasing drive strength. We group all standard cells that implement the same logic function but with different drive strength into one group. For each standard cell C_i, we say standard cell C_i' is a *possible alternative* of C_i if it satisfies the following three properties: (1) It is in the same group with C_i. (2) It has a larger drive strength than C_i. (3) It has a lower CTMR defined by Eq. (11.4) than C_i. We call all possible alternatives of C_i its *list of possible alternatives*, and we limit the way we modify the old solution such that a standard cell in the old solution can only be replaced by another standard cell in its *list of possible alternatives*. If such list is empty, that standard cell is fixed in the solution and can never be moved out from the old solution. Such limitations have the following advantages:

- By ensuring that a standard cell is replaced with another one that implements exactly the same function, we can perform incremental synthesis in EDA which can obtain power/area/timing information much faster than synthesizing a design from scratch. The correct functionality of the new design is also trivially guaranteed.
- Since all standard cells in the possible alternative list have a greater drive strength, enough drive strength originally required by the design is guaranteed.
- By limiting the replacement of one standard cell with only another standard cell in its *list of possible alternatives*, the search space is cut significantly. By the third property of the possible alternative, the search space is further reduced without affecting the optimality of the solution.
- It can be easily implemented, fully automated, and 100% compatible with the current flow of a commercial CAD tool.

Note that when we impose the above limitations, the modifications to the current solution can be done additively. This means that instead of adding/removing multiple standard cells from the design in one shot, we can always limit each modification to replacing only one standard cell say C_i with another standard cell C_i^j in its *list of possible alternatives*.

Gradient calculation. Since each modification to the current solution is done by replacing one standard cell with another one that is in its list of possible alternatives, we only need to define the gradient for each standard cell in all lists of possible alternatives of the initial solution. The gradient should assess the quality of the modification. Due to the discrete nature of our problem, gradient has to be defined in a different way. The detailed steps to obtain the gradient are defined below:

1. Let $l_0 = \{C_1, C_2, \ldots, C_k\}$ denote the initial solution. Let p_0, a_0, p_0 denote the leakage power, area, and timing slack obtained after initial synthesis.
2. For each standard cell in l_0, we identify its list of possible alternatives. We replace a standard cell C_i with one of its possible alternative C_i^j and keep all the other standard cells in l_0 unchanged. We call this new subset of standard cells l_i^j. We resynthesize the design on l_i^j and obtain new leakage power p_i^j, area a_i^j, and timing slack t_i^j. Let DTMR(l_0) and DTMR(l_i^j) denote the design's Trojan miss rate (defined by Eq. 11.5) when synthesizing the circuit on l_0 and l_i^j, respectively. We define the following quantity first:

$$\Delta\text{DTMR}(l_i^j) = \text{DTMR}(l_0) - \text{DTMR}(l_i^j)$$

$$\Delta t(l_i^j) = \frac{t_0 - t_i^j}{t_0},\ \Delta a(l_i^j) = \frac{a_i^j - a_0}{a_0},\ \Delta p(l_i^j) = \frac{p_i^j - p_0}{p_0}$$

We then define the gradient as follows:

$$\nabla\text{DTMR}(l_i^j) = \frac{\Delta\text{DTMR}(l_i^j)}{\Delta t(l_i^j) + \Delta a(l_i^j) + \Delta p(l_i^j)} \tag{11.7}$$

Intuitively, this quantity measures how much improvement in DTMR we can get with 1% overhead in power, area, and timing slack by replacing C_i with C_i^j in l_0 to form the new library l_i^j. The greater this gradient is, the more improvement in DTMR such replacement can yield given that the overhead stays the same.

Overall Algorithm. Putting all things together in the framework of the steepest descent method, we detail the steps of our algorithm below:

1. Identify and characterize primitive structures. Then characterize each standard cell in a given technology library. Note that for a given technology library, this step only needs to be done once.
2. Synthesize the design using the whole technology library and obtain the subset of standard cells used denoted by l_0. This is our initial solution. We set $l_{\text{old}} = l_0$.
3. Identify the list of possible alternatives for each of the standard cells in l_0. Obtain the gradient defined by Eq. (11.7) for every standard cell in every list of possible alternatives.
4. Find in all lists of possible alternatives the standard cell with the largest corresponding gradient, say C_i^j which is jth standard cell in the possible alternative list of C_i.
5. Replace C_i with C_i^j in l_{old} and let l_{new} denote the new subset of standard cells. Synthesize the design on l_{new}, and check if area/power/timing constraints are met. If so, set $l_{\text{old}} = l_{\text{new}}$, and set the possible alternative list of C_i to empty. If not, remove C_i^j from the possible alternative list of C_i, and keep l_{old} unchanged.
6. Repeat steps 4–5 until all lists of possible alternatives are empty. l_{old} is what we want.

11.5.3 Experiments and Result

11.5.3.1 Experiment Setup

Benchmarks. We test our approach on eight publicly available benchmarks (from ISCAS89 [19] and trustHub [20]). They are all synthesized using Cadence RTL Compiler with Synopsys 90 nm generic library. Reports from Cadence tool show that these benchmarks have gate counts ranging from 69 for s298 to 175,456 for AES (shown in Table 11.5).

Applying constraints. We set the load capacitance to 100 pF on all output ports for all benchmarks. We set the input/output delays to 200 ps. The clock frequency of each benchmark is listed in Table 11.5. For each benchmark, we apply the above timing constraints (no timing violations are tolerated in our experiments) and synthesize the design using the Synopsys 90 nm generic library. We can then obtain the area and leakage power after synthesis is done. We then set the allowed overhead for leakage power and area to 30%, which means the leakage power and area should not exceed their original values by more than 30%.

Table 11.5 Benchmarks used in the experiments

Benchmark	Source	#Gates	Clock frequency
s298	ISCAS89	69	25 MHz
s5378	ISCAS89	728	25 MHz
s15850	ISCAS89	1968	25 MHz
s38417	ISCAS89	5066	25 MHz
AES	TrustHub	175,456	500 MHz
b19	TrustHub	45,150	25 MHz
MC8051	TrustHub	6927	50 MHz
XGE_MAC	TrustHub	60,222	156.25 MHz

Baseline design and security-aware design generation. For each benchmark, we synthesize the circuit using the whole library with all timing constraints applied. We call this design the baseline design. It represents a design that is optimized in area and meets all timing constraints but lacks any security concerns. We then apply our algorithm and obtain a list of standard cells. We resynthesize the circuit on this list of standard cells and call it security-aware design. We will use Cadence Encounter tool to place and route both designs.
Trojan insertion. For each benchmark, for both baseline design and security-aware design, we randomly choose a standard cell as our attack target and make four malicious modifications to it (two additions of structures and two removals of structures). The random selection of attack target is done 15 times and in total 60 Trojan-inserted chips are generated.
Experiment conducted and results recorded. For each benchmark, for both baseline design and security-aware design, we train SVM over 5 Trojan-free chips for s298 and 1 Trojan-free chip for all other benchmarks and use the classifier to classify the 60 Trojan-inserted chips. The detailed training and classification steps as well as parameter selection follow the work in [13] with one small distinction. Instead of classifying the whole IC, we only generate the labels for each grid in the IC. We then calculate the Trojan miss rate defined in Eq. (11.1) for both designs averaged over 60 chips. The difference of Trojan miss rate between baseline design and security-aware design is then calculated, and the ratio of the difference to the Trojan miss rate of the baseline design is reported as TMR improvement. We also report the leakage power and area overhead of security-aware design compared with baseline design. The results are listed in Table 11.6. The algorithm run-time as well as the time consumed for the initial synthesis is also reported. Note that the algorithm run-time does not include the pre-synthesis step shown in Fig. 11.4.

11.5.3.2 Results

From Table 11.6, we can see that using our method, we can detect on average 16.87% more malicious modifications with only 7.87% area overhead and 17.72% leakage power overhead. Note that Cadence RTL Compiler always optimizes

Table 11.6 TMR improvement, leakage power overhead (LO), area overhead (AO), algorithm run-time (AT), and the original synthesis time (OT) of all benchmarks

Benchmark	TMR improvement (%)	AO (%)	LO (%)	OT(s)	AT(s)
s298	20.93	10.09	17.67	16.4	283.9
s5378	13.04	7.91	16.05	31.3	663.6
s15850	15.23	6.21	13.94	44.4	906.1
s38417	16.00	4.00	8.58	103.4	990.4
AES	23.45	16.68	28.21	2302	9941
b19	23.21	8.51	29.74	759.1	3841
MC8051	15.62	7.70	21.99	204.3	10,010
XGE_MAC	7.48	1.85	5.60	2031	9055

the area given that all timing constraints are met, making the area overhead very small compared with leakage overhead. We also note that our algorithm's performance depends heavily on the timing requirements of the design. If a design's timing requirements are easily met, then our algorithm can improve TMR greatly. Examples are AES and b19. However, if a design's timing requirements are very hard to meet, then the improvement in TMR is limited. The example is XGE_MAC benchmark which implements MAC functions for 10 Gbps operation. The reason is that when timing requirements are hard to meet, each standard cell will have only very few possible replacement alternatives because a replacement is very likely to cause timing violations. Therefore, the search space for our algorithm is small and the improvement in TMR is limited. Nevertheless, we still can detect 7.48% more Trojans with only 1.85% overhead in area and 5.60% in leakage power in the latter case. This proves the efficacy of our algorithm.

11.6 Conclusion

In this chapter, we introduce the fundamentals of IC reverse-engineering and the hardware Trojan detection methods using RE. Then, we introduce the RE-based hardware Trojan detection approach which leverages SVM to determine whether a chip is Trojan-infected. Experimental results show that this approach correctly detects Trojan-free and Trojan-infected chips in more than 99% of the cases. To make Trojan detection more accurate, a design-for-security approach is introduced where the standard cells that are more sensitive to the insertion of hardware Trojans are selected to synthesize the layout of the chip. Significant improvement in the fraction of the Trojans detected is achieved using this design approach with acceptable overhead.

References

1. R. Karri, J. Rajendran, K. Rosenfeld, Trojan taxonomy, in *Introduction to Hardware Security and Trust*, ed. by M. Tehranipoor, C. Wang (Springer, New York, 2012), pp. 325–338
2. X. Zhang, M. Tehranipoor, Case study: detecting hardware trojans in third-party digital IP cores, in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (IEEE, 2011), pp. 67–70
3. F. Wolff, C. Papachristou, S. Bhunia, R. Chakraborty, Towards trojan-free trusted ICs: problem analysis and detection scheme, in *Proceedings of the Conference on Design, Automation and test in Europe* (ACM, 2008), pp. 1362–1365
4. R. Chakraborty, F. Wolff, S. Paul, C. Papachristou, S. Bhunia, MERO: a statistical approach for hardware trojan detection, in *Cryptographic Hardware and Embedded Systems – CHES 2009*, ed. by C. Clavier, K. Gaj. Lecture Notes in Computer Science, vol 5747 (Springer, Berlin/Heidelberg, 2009), pp. 396–410
5. Y. Jin, Y. Makris, Hardware trojans in wireless cryptographic integrated circuits. Des. Test IEEE (99), 1–1 (2013)
6. S. Narasimhan, S. Bhunia, Hardware trojan detection, in *Introduction to Hardware Security and Trust* (Springer, New York, 2012), pp. 339–364
7. R. Torrance, D. James, The state-of-the-art in semiconductor reverse engineering, in *Proceedings of the 48th Design Automation Conference* (ACM, 2011), pp. 333–338
8. Y. Shiyanovskii, F. Wolff, A. Rajendran, C. Papachristou, D. Weyer, W. Clay, Process reliability based Trojans through NBTI and HCI effects, in *2010 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)* (IEEE, 2010), pp. 215–222
9. Y. Jin, Y. Makris, Proof carrying-based information flow tracking for data secrecy protection and hardware trust, in *2012 IEEE 30th VLSI Test Symposium (VTS)* (IEEE, 2012), pp. 252–257
10. H. Salmani, M. Tehranipoor, J. Plusquellic, A novel technique for improving hardware Trojan detection and reducing Trojan activation time. IEEE Trans. Very Large Scale Integr. VLSI Syst. **20**(1), 112–125 (2012)
11. S. Narasimhan, X. Wang, D. Du, R. Chakraborty, S. Bhunia, TeSR: a robust temporal self-referencing approach for hardware Trojan detection, in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (June 2011), pp. 71–74
12. S. Narasimhan, D. Du, R. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, S. Bhunia, Multiple-parameter side-channel analysis: a non-invasive hardware Trojan detection approach, in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, (IEEE, 2010), pp. 13–18
13. C. Bao, D. Forte, A. Srivastava, On application of one-class SVM to reverse engineering-based hardware Trojan detection, in *2014 15th International Symposium on Quality Electronic Design (ISQED)* (IEEE, 2014), pp. 47–54
14. C. Bao, D. Forte, A. Srivastava, On reverse engineering-based hardware Trojan detection. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **35**(1), 49–57 (2016)
15. C. Bao, Y. Xie, A. Srivastava, A security-aware design scheme for better hardware trojan detection sensitivity, in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (IEEE, 2015), pp. 52–55
16. S.M. Plaza, I.L. Markov, Solving the third-shift problem in IC piracy with test-aware logic locking. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**(6), 961–971 (2015)
17. V. Vapnik, *The Nature of Statistical Learning Theory* (Springer, New York, 2000)
18. M. Tehranipoor, F. Koushanfar, A survey of hardware trojan taxonomy and detection. IEEE Design Test Comput 27(1), 10–25 (2010)
19. F. Brglez, D. Bryan, K. Kozminski, Combinational profiles of sequential benchmark circuits, in *IEEE International Symposium on Circuits and Systems, 1989* (IEEE, 1989), pp. 1929–1934
20. trust HUB.org, http://trust-hub.org/resources/benchmarks

Part V
Design for Security

Chapter 12
Hardware Obfuscation Methods for Hardware Trojan Prevention and Detection

Qiaoyan Yu, Jaya Dofe, Zhiming Zhang, and Sean Kramer

12.1 Introduction

Integrated circuits (ICs) and systems are increasingly using the global resources to reduce the hardware cost. Unfortunately, the untrusted parties in the global supply chain may tamper the original design or bring in malicious components, which are known as hardware Trojans. Being aware of potential hardware Trojans in the design, researchers have developed functional testing, optical analysis, and side-channel analysis methods. Hardware Trojan detection through functional testing is challenging in terms of finding effective input vectors to activate the rarely triggered Trojans. The success of optical analysis on circuit layout pictures strongly relies on the availability of precise delayering, high-resolution imaging, and pattern recognition techniques. As the size of hardware Trojans becomes relatively tiny compared with the circuit under attack, the amount of changes appeared on the side-channel signals, delay, power, area, and temperature will not be sufficiently noticeable for side-channel analysis-based Trojan detection. Moreover, the lack of a golden model for reference makes Trojan detection difficult.

In this chapter, we introduce another set of countermeasures, *hardware obfuscation*, for hardware Trojan prevention and detection. Different with functional testing, scanning electronic microscope (SEM) image-based analysis, and side-channel analysis methods, hardware obfuscation methods are applicable to design time. Therefore, obfuscation-based countermeasures prevent the outsourced circuit design from being tampered or reverse engineered, rather than simply discarding

Q. Yu (✉) • J. Dofe • Z. Zhang • S. Kramer
Department of Electrical and Computer Engineering, University of New Hampshire, 03824 Durham, NH, USA
e-mail: qiaoyan.yu@unh.edu; jhs49@wildcats.unh.edu; zz1017@wildcats.unh.edu; sdq46@wildcats.unh.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_12

the compromised chips. Furthermore, the circuit modification made through obfuscation techniques will facilitate hardware Trojan detection after fabrication.

The remainder of this chapter is organized as follows. The principle of obfuscation is introduced in Sect. 12.2. After a short summary of hardware Trojans, the obfuscation methods for different abstraction levels are reviewed in Sect. 12.3. In Sect. 12.4, we present the key idea of the hardware obfuscation methods at device, circuit, gate, and register-transfer levels. Obfuscation in FPGAs and printable circuit boards (PCBs) are briefly discussed in Sects. 12.5 and 12.6, respectively. Since there are no any standard evaluation metrics available for hardware obfuscation methods, we convene the metrics that have been used in the existing literature so far in Sect. 12.7. This chapter is concluded in Sect. 12.8.

12.2 Obfuscation

12.2.1 Concept of Obfuscation

Obfuscation is a process to make a target of interest obscure or unclear. Obfuscation techniques are common in the software world. In particular, obfuscation in software is often referred to code obfuscation techniques. The purpose is to modify source codes, execution instructions, or metadata but do not change the program final output. The obfuscated source code is altered in a way that is difficult for a hacker to understand and reverse engineer. Consequently, obfuscation techniques provide a defense layer to prevent the source code from intellectual property (IP) theft, unlicensed reuse, and tampering. Figure 12.1 shows a simple example of using ASCII (American Standard Code for Information Interchange) code to replace the characters in a string used in software programming. The string *Save Secure Key* is replaced with a series of *Chr* or hexadecimal codes, which are not readable for humans.

The original definition of *circuit obfuscation* (or generally speaking, *hardware obfuscation*) was first proposed by Barak et al. [3]. The initial *circuit obfuscation* has two strong requests: (1) the obfuscated circuit executes the same function as its original circuit at the expense of "polynomial-time slowdown", and (2) the circuit after obfuscation will only leak as much information that is available on the input and output names. The second requirement means that an obfuscated circuit should be recognized as a black box. Because the second request is too strong to be reachable, the term "obfuscation" refers to what is known as *best-possible obfuscation*, which does not guarantee that the obfuscated circuit will completely hide the functionality of the original circuit.

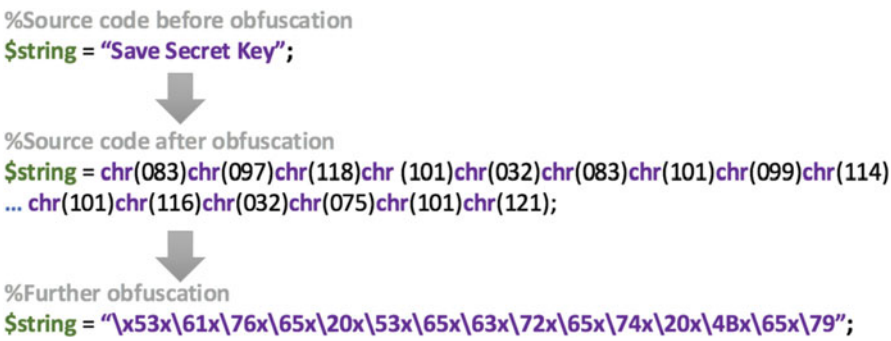


Fig. 12.1 Example of obfuscation in software

12.2.2 Differentiating Obfuscation from Encryption

Obfuscation is fundamentally different from *encryption*. An obfuscated source code is executable without de-obfuscation. Obfuscation increases the level of difficulty in understanding a design while maintaining the original design functionality. We hope that the benefit of obfuscation will deter attackers. A secret key is not always required in the process of obfuscation. Once the principle of adopted obfuscation process is leaked, the defense layer provided by obfuscation is nullified. In contrast, any encrypted program is not executable before being decrypted. Without the correct cipher key, the wrongly decrypted design will produce a completely different output than the original one due to the avalanche effect. As long as the cipher key is kept secret, the leak of encryption/decryption algorithms does not affect the security offered by encryption.

12.2.3 Obfuscation Techniques in Software World

In software, there are many ways to obfuscate source codes. Renaming is a basic technique, which makes variables meaningless or uses unprintable or invisible characters to replace the readable phrases. The *.NET*, *iOS*, *Java*, and *Android* system often use this technique to obfuscate source codes. Common types of obfuscation techniques used in software are summarized in Table 12.1. Due to the different nature of hardware implementation over software, the techniques in Table 12.1 are not applicable to hardware.

Table 12.1 Summary of typical obfuscation techniques in software

Obfuscation techniques	Main process
Renaming	Make variable name meaningless, unreadable
String encryption	Hide the string in the executable and restore the string when needed
Control flow obfuscation	Synthesize conditional, branching, and iterative program constructs
Instruction pattern transformation	Change the common instruction constructs into an irregular pattern
Dummy code insertion	Insert extra dummy code to the original executable
Unused code and metadata removal	Remove unused code and information that may serve as a hint for hackers
Binary linking/merging	Combine multiple executables/libraries into a flattened single one
Opaque predicate insertion	Insert dead code as an additional branch, which will never be reached in real-time execution
Anti-tamper	Add self-protection code to verify the presence of tampering codes
Anti-debug	Remove the debug interface

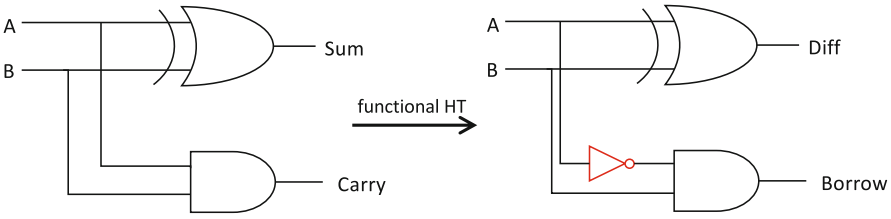


Fig. 12.2 Example of functional hardware Trojan

12.3 The Role of Obfuscation in Hardware Trojan Prevention and Detection

12.3.1 Hardware Trojan

Hardware Trojan refers to malicious hardware modification that is made by an adversary. Hardware Trojans can be introduced during the process of design or fabrication. The hardware Trojan taxonomy is extensively studied in [57]. Some hardware Trojans are manifested as an addition or a removal of logic gates, transistor, or interconnect, resulting in malfunctions. This type of hardware Trojans is called functional hardware Trojans. As Fig. 12.2 shows, if an inverter is added between input signal A and the AND gate of the half adder, then the circuit function is changed to a half subtracter. Another type of hardware Trojans changes the

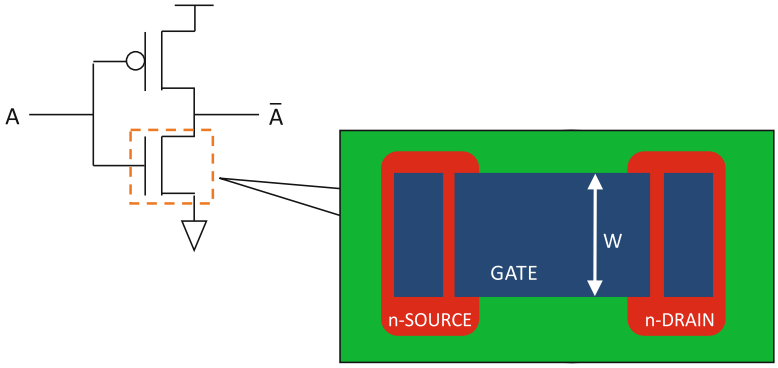


Fig. 12.3 Example of parametric hardware Trojan

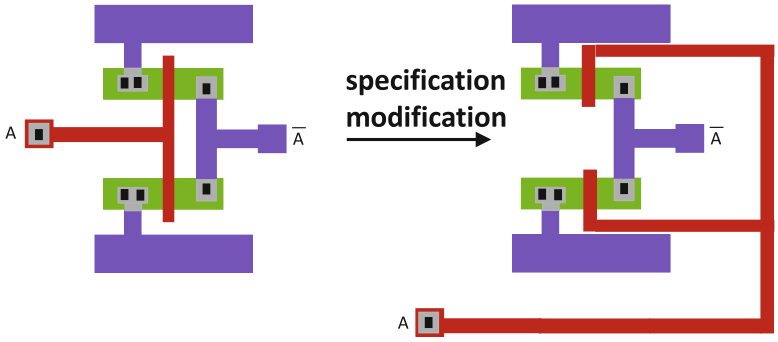


Fig. 12.4 Another example of parametric hardware Trojan causing delay variation

electrical parameters, named as parametric hardware Trojans. An example, provided in Fig. 12.3, shows that if the NMOS transistor of an inverter is weakened, it will become more difficult to pull down the output. Figure 12.4 provides another example of parametric hardware Trojans. The layout on the left is the standard one, while the layout on the right is modified by increasing the wire's length. The change on the layout leads to an increased delay. Some hardware Trojans are externally activated, meaning that they can be triggered by antennas or sensors connected to the outside world. Figure 12.5 illustrates an example of hardware Trojan being triggered remotely via an antenna. In contrast, hardware Trojans can also be internally activated, where one portion of hardware Trojan is always operating. Internal hardware Trojans could potentially modify the geometries of a circuit or be conditionally active. In this case, the hardware Trojan activates when the trigger condition is satisfied. Overall functional hardware Trojans triggered by internal states or logic signals are the main focus of this chapter.

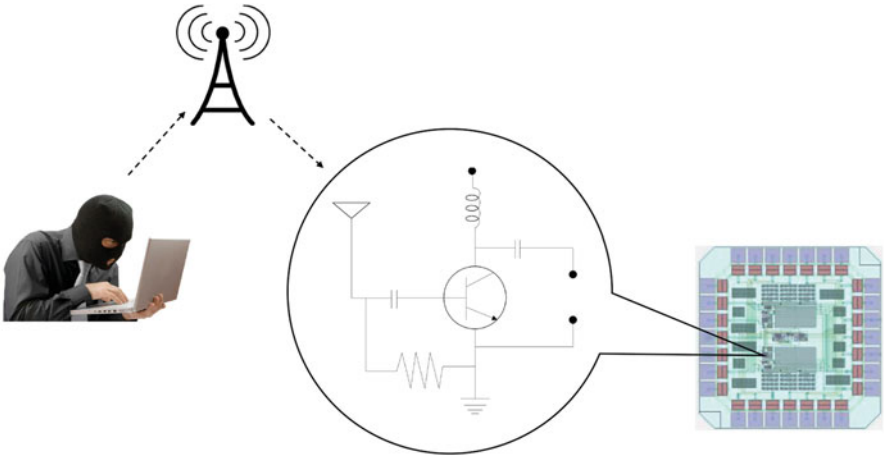


Fig. 12.5 Externally triggered hardware Trojan

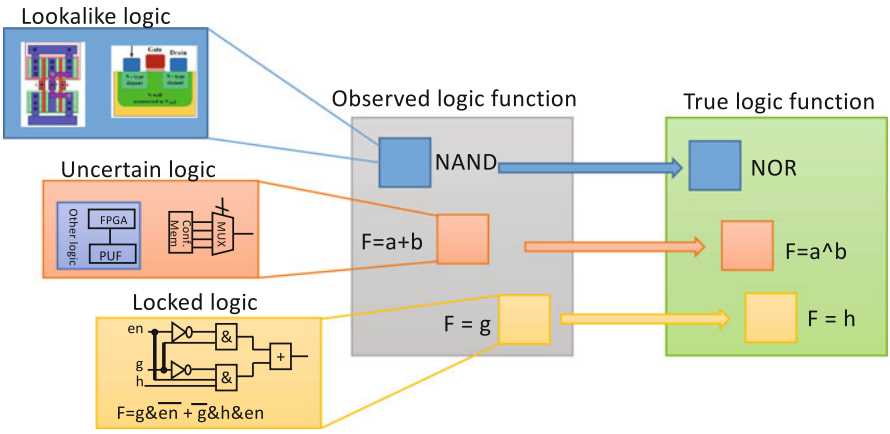


Fig. 12.6 Hardware obfuscation obscuring logic function to prevent hardware attacks

12.3.2 Overview of Hardware Obfuscation

The principle of hardware obfuscation is to alter the design in a way that has the equivalent functionality as the original one but appears to be significantly more difficult for an adversary to understand and exploit. As shown in Fig. 12.6, obfuscation methods facilitate to create look-alike logic, uncertain logic (until post-fabrication configuration), and key-controlled logic. An attacker who does not have sufficient knowledge of the applied specific obfuscation procedure could misunderstand the logic function of the module of interest. For instance, the NOR gate is recovered as a NAND gate; an XOR function is recognized as an addition. Consequently, hardware Trojan insertion following that misunderstanding would

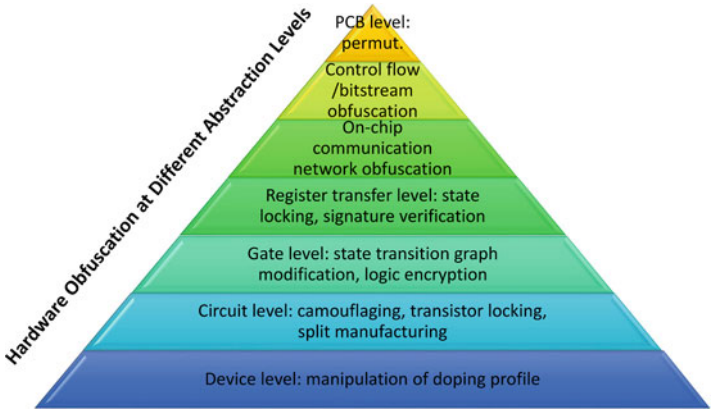


Fig. 12.7 Hardware obfuscation at different abstraction levels

lead to either a non-triggered hardware Trojan or a Trojan which cannot fulfill the desired attack purpose. The application of hardware obfuscation has been demonstrated at different abstraction levels in integrated circuits and systems. We summarize that in Fig. 12.7. The following subsections introduce the key idea of the methods at each level in details.

12.4 Chip-Level Obfuscation

Chip-level obfuscation includes the hardware modification that we make on the device, circuit, gate, and register-transfer levels to confuse attackers who intend to insert hardware Trojans.

12.4.1 Device-Level Obfuscation

Device-level obfuscation is to disguise the real function of a device by introducing *controllable* faults, such as stuck-at and delay faults. Without knowing the exact details of the device implementation procedure, a reverse engineer may misinterpret the device (and the associated circuits) function based on the superficial understanding from, for instance, the SEM images. We can realize the device-level obfuscation by altering the doping concentration for transistors. To disguise a PMOS transistor, we can obfuscate the transistor in a way shown in Fig. 12.8. The source and drain regions of the transistor are doped with N-type dopant, rather than P-type dopant in the original setting. Once we connect the source to Vdd, a short circuit is created by the doping-induced obfuscation. An obfuscated device can be further applied

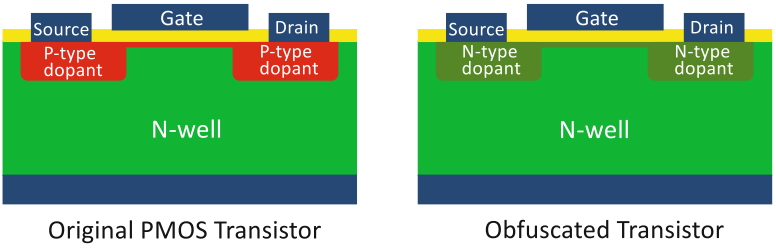


Fig. 12.8 Hardware obfuscation through changing doping concentration. Source and drain regions of a transistor use the same dopant. The PMOS transistor yields a constant Vdd output if the source terminal is connected to Vdd [63]

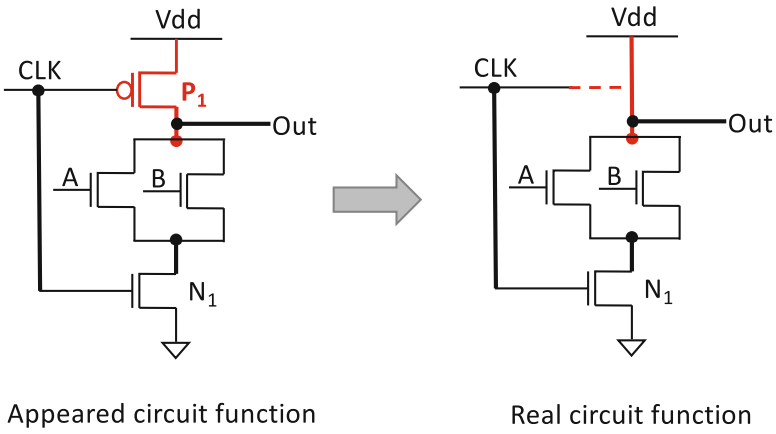


Fig. 12.9 The fault induced by the obfuscated PMOS transistor leading to a different logic function

to a circuit to mislead the reverse engineer. A simplified example is illustrated in Fig. 12.9. The transistor P_1 is an obfuscated one (using the doping method from Fig. 12.8). As the transistor P_1 is constantly connected to Vdd, the dynamic NAND gate no longer functions like a real NAND gate. The use of obfuscated logic gates will cause reverse engineers tremendous amounts of extra effort to extract the original functionality from the circuit using obfuscated devices. As a result, an attacker who intends to insert hardware Trojans in such circuits will not easily achieve his/her ultimate attack goal, as the logic gate contaminated by a hardware Trojan may not be a true functioning gate initially.

In addition to source/drain regions, transistor channel and transistor-to-transistor interconnect can also be a place for an obfuscation candidate [63]. Virtual connection leads to an open circuit. Manipulation of interlayer dielectric (ILD) results in stealthy capacitive signals (i.e., crosstalk). The techniques optical proximity correction (OPC) and sub-resolution assist features (SRAFs) are initially developed to improve the quality of fabricated chips, and those techniques can be exploited

to generate timing faults for the obfuscation purpose. However, the doping-based obfuscation technique does not change the transistor geometry. Passive voltage contrast (PVC) [54] and picosecond imaging circuit analysis (PICA) [59] can detect the modification on the source/drain doping and channel doping, respectively. There is superconducting quantum interference device (SQUID) [42] microscopy available for attackers to examine the open/short interconnect. Table 12.2 provides a summary of device-level obfuscation methods and the corresponding detection schemes. More details for device-level obfuscation are available in the survey paper [63].

12.4.2 Circuit-Level Obfuscation

The circuit-level obfuscation techniques are divided into two categories: camouflaging layout and transistor locking. Camouflaging at circuit layout is used to thwart the reverse engineers from successfully reading the layout and thus recovering the transistor-level schematic for that circuit. The camouflaging techniques have been applied to two-dimensional (2D) and three-dimensional (3D) ICs. Transistor locking is another type of obfuscation (in a general sense), which changes the logic gate functionality by muting some transistors in the schematic [32, 40]. A recent work [20] combines the camouflaging and transistor-level locking techniques for obfuscating circuits in 3D ICs. In the next subsections, representable circuit-level obfuscation methods are introduced.

12.4.2.1 Camouflaging in 2D ICs

Circuit camouflaging is a technique that makes subtle changes within the physical layout of a logic cell to hide its actual logic function [15, 16, 48]. The main goal of camouflaging is to disguise the circuit such that a reverse engineer who utilizes SEM pictures cannot recover the original chip design. The general principle of camouflaging is either to make the connected nodes appear to be isolated or to have the isolated nodes appear to be connected. In real applications, one can partially camouflage the circuit using a strong custom cell camouflage library [14, 16, 21] or a generic camouflaged cell layout [48]. The work [16] illustrates that the conventional and camouflaged 2-input AND gates have the same SEM image, as shown in Fig. 12.10. As a result, the attacker could easily extract a wrong netlist if he/she relies on the SEM image analysis only. Furthermore, because of the wrong netlist extraction, the attacker cannot precisely modify the netlist to insert hardware Trojans. In the camouflage library [16], every cell looks identical at every mask layer to prevent automated layout recognition [14, 21]. However, making every cell identical in terms of size and spacing with other cells will reduce the cell performance. To address this limitation, Cocchi et al. provide elementary block camouflage libraries that are derived from each logic gate layout [21] and could be programmed after the manufacturing process.

Table 12.2 Summary of fault mechanisms and corresponding obfuscation classes and detection schemes [63]

			Obfuscation classes			Detection
			Stuck-at faults	Timing faults	Stealthy signaling	
Physical design mechanisms	Doping	Source/drain	×			PVC
		Channel	×	×		PICA
	Metal-fill		×	×	×	SEM
	ILD manip	Thinning		×	×	–
		Thickening	×	×	×	–
	Interconnect mask manipulation		×	×	×	SEM, SQUID
	SRAFs			×		–

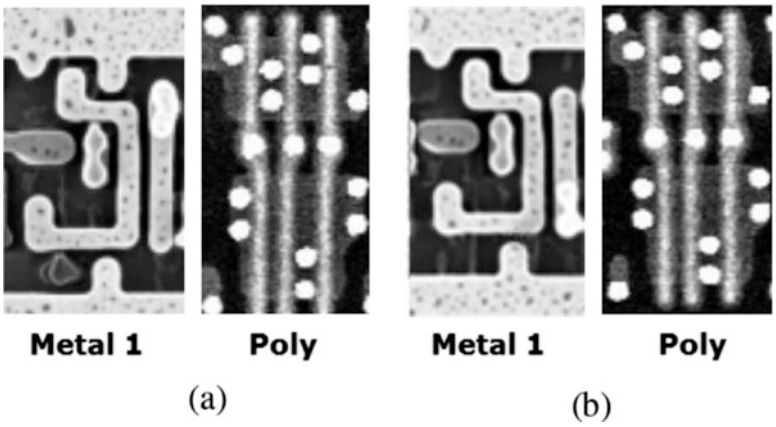


Fig. 12.10 SEM images of (**a**) conventional and (**b**) camouflaged AND gate [16]

12.4.2.2 Camouflaging and Multi-tier Locking in 3D ICs

A multi-tier logic locking mechanism for monolithic three-dimensional (M3D) ICs is proposed in [20] to thwart IP piracy and reverse engineering attacks [52, 58]. In M3D ICs, a functional block is fabricated in two tiers, PMOS pull-up network (PUN) on the bottom tier and NMOS pull-down network (PDN) on the top tier [64]. PUN and PDN on different tiers are independently locked by the camouflaged parallel or serial locking circuit. The number of locking units, key values, camouflaged contact information, and locking circuit locations for two tiers is different. This arrangement protects the 3D circuit from attacks that try to exploit the collaborative analysis on two tiers. Without the correct key, the locked functional block either leads to malfunctions (flip the output value or result in either a floating ground/power pin or a shorted ground/power line) or significant changes on the power profile. The locking keys are only available to authorized users. Even if the complete layout is available to an adversary, it would still be highly challenging for attackers to reverse engineer the entire locked 3D circuit. In total, the work [20] introduces four locking configurations: PMOS parallel locking (PPL), NMOS parallel locking (NPL), PMOS serial locking (PSL), and NMOS serial locking (NSL). Moreover, that work exploits the contact camouflaging to thwart image-analysis-based reverse engineering and hardware Trojan insertion.

Serial 3D Logic Locking: The concept of the serial locking circuit is depicted in Fig. 12.11. The PMOS transistor (*P1*) is controlled by a key bit (*Key1*). The power pin VDD and the *P1* source terminal are connected with PUN through camouflaged contacts. One of the camouflaged contacts is filled with dielectric, which results in only one real connection. As demonstrated in 2D ICs, contact camouflaging is a feasible and promising method to thwart image-analysis-based reverse engineering attacks [48]. The locking circuit can also be applied to the PDN tier, where NMOS

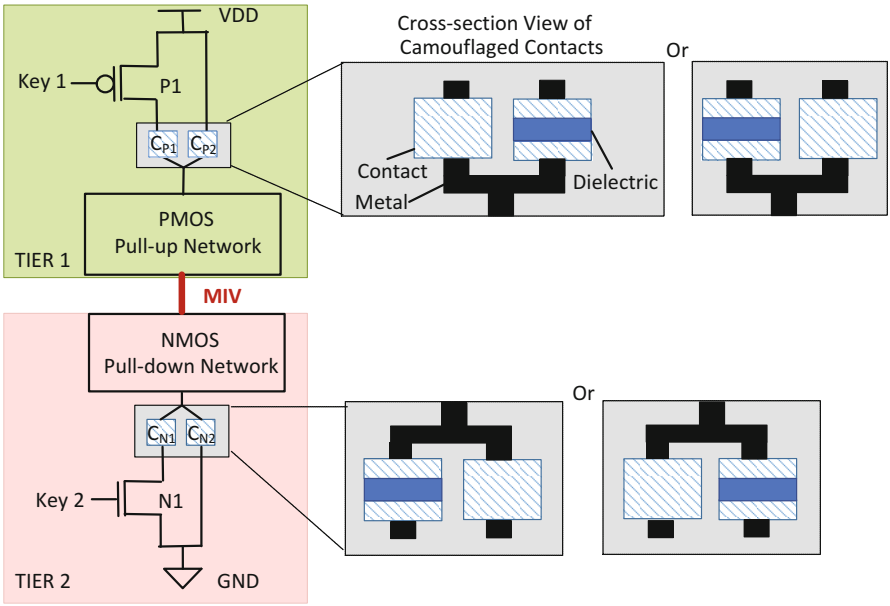


Fig. 12.11 3D logic cell with serial locking (PSL and NSL) and camouflaged monolithic inter-tier vias (MIVs) [20]

locking with a short circuit wire is inserted between the NMOS PDN and the real ground line. Different *Key1* and *Key2* will help to reduce the correlation between tier 1 and tier 2. For simplicity, the same value for *Key1* and *Key2* is used in the following example. Table 12.3 lists the connection configuration for the camouflaged contacts C_{N1}, C_{N2}, C_{P1}, and C_{P2} for different key value scenarios. In the first half of Table 12.3, the real design setting is as follows: the correct key bit is 0, the contacts C_{N1} and C_{P2} are disconnected with dielectric; only C_{N2} and C_{P1} are truly connected. The hypothesis key of 1 will turn off PMOS $P1$, thus causing a floating VDD. Figure 12.11 depicts this scenario. The second half of Table 12.3 shows another configuration if the correct key bit is 1. In this case, the camouflaged contacts C_{N2} and C_{P1} are not truly connected. The wrong hypothesis key of 0 will turn off NMOS $N1$ and cause PDN to have a floating ground. To implement this configuration, the camouflaged contacts in Fig. 12.11 need to be modified.

Parallel 3D Logic Locking: Alternatively, the proposed camouflaged logic locking can be performed in parallel with the original PDN and PUN, as shown in Fig. 12.12. Contrary to the serial locking circuit, no short circuit wire is needed in parallel locking. If the correct key is 0 (the first half of Table 12.4), the contact C_N is truly connected, but the contact C_P is disconnected in the camouflaged layout. Because of the camouflaged disconnection in C_P, the wrong key bit (i.e., 1) produces a pull-down network always shorted to ground. The second half of Table 12.4 indicates that the camouflaged disconnection in C_N will cause the pull-up network

Table 12.3 Contact and transistor status in serial locking

Correct Key1, Key2 = 0							
Key	C_{N1}	C_{N2}	C_{P1}	C_{P2}	$N1$	$P1$	Result
0	X	✓	✓	X	Off	On	Normal
1	X	✓	✓	X	On	Off	**Floating VDD**
Correct Key1, Key2 = 1							
Key	C_{N1}	C_{N2}	C_{P1}	C_{P2}	$N1$	$P1$	Result
0	✓	X	X	✓	Off	On	**Floating GND**
1	✓	X	X	✓	On	Off	Normal

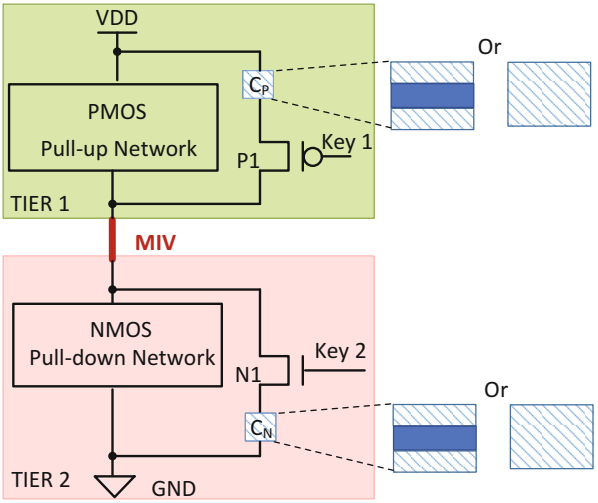


Fig. 12.12 3D logic cell with parallel locking (PPL and NPL) and camouflaged MIVs [20]

always shorted to VDD if the wrong key of 0 is applied to $P1$. Figures 12.11 and 12.12 show that a single transistor is used in locking units. In real designs, the locking circuits for PUN and PDN are not necessarily symmetric. Asymmetric locking circuits will provide stronger protection against reverse engineering attacks.

This method is particularly designed to prevent the attacker from correlating PUN and PDN after the separation of the PMOS and NMOS tiers. The ultimate goal of the proposed method is to circumvent attackers from understanding the entire 3D IC design and inserting hardware Trojans. Even if the attacker retrieves the design of one tier, it is still difficult to completely derive the design in another tier.

In the work [20], monolithic cells are used to implement an ISCAS'85 benchmark circuit, c17. The VDD of one NAND2X1 gate in c17 is locked by PSL shown in Fig. 12.13a. Camouflaged contacts are applied in the PDN of that same NAND2X1 gate. As the NMOS locking transistor is shorted to ground via C_{N2} contact, the locking circuit is omitted in Fig. 12.13a. When the key bit is low, the PMOS is turned on and thus c17 operates normally. Figure 12.13b shows the impact

Table 12.4 Contact and transistor status in parallel locking

Correct Key1, Key2 = 0					
Key	C_N	C_P	$N1$	$P1$	Result
0	✓	X	Off	Off	Normal
1	✓	X	On	Off	**Always pull-down**
Correct Key1, Key2 = 1					
Key	C_N	C_P	$N1$	$P1$	Result
0	X	✓	Off	Off	**Always pull-up**
1	X	✓	Off	Off	Normal

of key on the c17 primary outputs and power. The input patterns for valid and invalid key period are exactly the same. However, the primary outputs, N22 and N23, yield different values for invalid and valid key scenarios. The corresponding power profiles for valid and invalid key periods are also different. This example demonstrates that the circuit locked by key bits through PSL do alter the primary outputs and power profile, thus obscuring the 3D circuit if the attacker does not have the valid key. The experiment is repeated by replacing PSL with PPL. The corresponding output signals and power profile are shown in Fig. 12.13c. Compared to Fig. 12.13b, the consequence of always pull-down caused by an incorrect key leads to a significant change on power, which does not match to the power profile for any logic gate. Hence, the proposed locking circuit can also resist power-based side-channel attacks.

12.4.3 Gate-Level Obfuscation

There is a considerable amount of hardware Trojans that are based on rare events. Due to the rareness of Trojan trigger conditions, traditional testing methods that examine the functionality are not practical because tremendous amounts of different input patterns are required to trigger the Trojan. As the chip scale increases, the size of a hardware Trojans is minuscule compared with the chip size. As a result, side-channel analysis methods are not feasible to detect the changes on delay, area, and power caused by the presence of hardware Trojans. Gate-level netlist obfuscation complements the device-level obfuscation. Instead of obscuring the true device functionality, the netlist obfuscation methods modify the state transition function of a circuit such that the attacker is not able to reach the real power-up state. Thus, the inserted hardware Trojan is either in the states belonging to the authentication mode or in the normal operation mode. As the attacker does not know the correct obfuscation key, the Trojans in the latter mode will never be triggered. Consequently, the gate-level obfuscation methods hinder the hardware Trojans from affecting the module under protection.

Logic encryption is prevalent to protect combinational logic circuits. The locked combinational logic will yield, ideally, the opposite logic output to the original one. Literatures [22, 39, 44, 47, 50] have done extensive studies on how to choose the

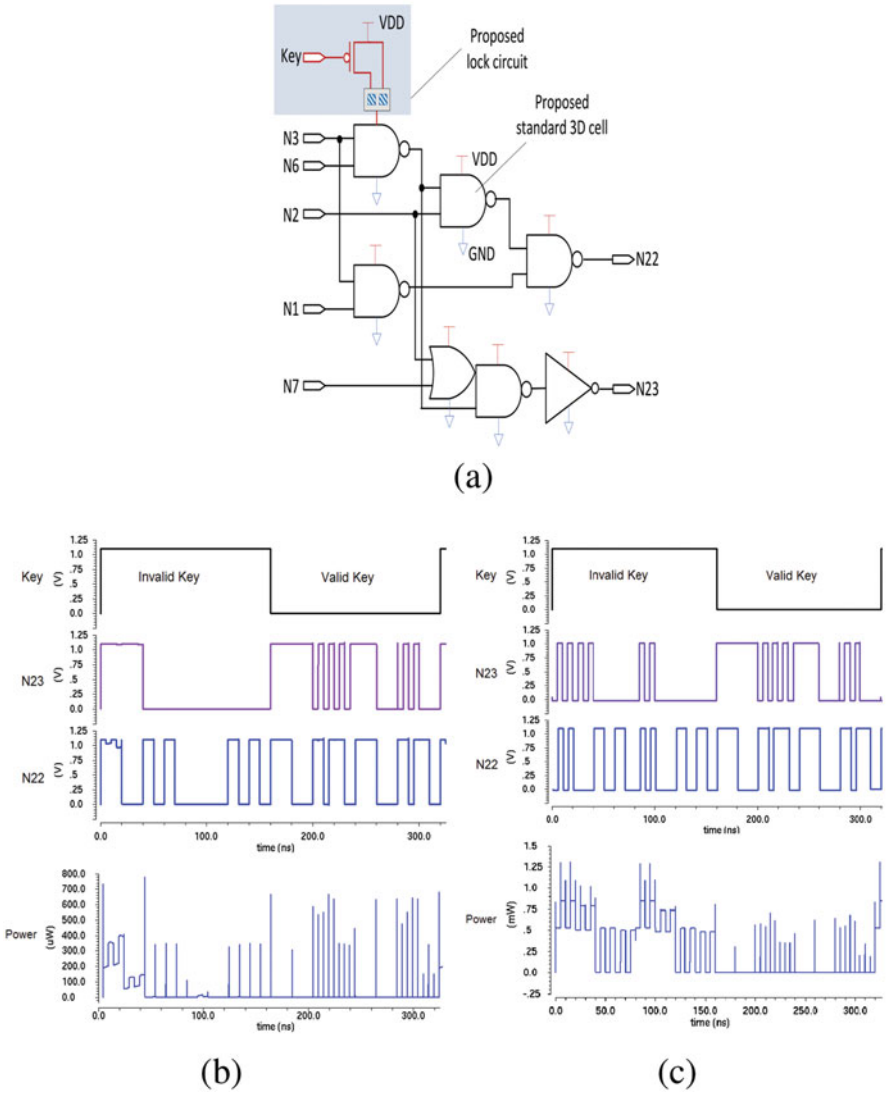


Fig. 12.13 (**a**) Schematic for locked c17 circuit and impact of invalid/valid key on output signals and power of the (**b**) serial and (**c**) parallel locked c17

location for encryption and how to strengthen the logic encryption methods to thwart possible IP piracy reverse engineering attacks. In the following subsection, we will focus on the obfuscation for sequential circuits.

12.4.3.1 HARPOON: Obfuscation on State Transition Graph and Internal Node Structure

Obfuscation on the state transition graph (STG) [6, 7] was first proposed to prevent IP piracy. The key idea of obfuscation on STG is depicted in Fig. 12.14. A small finite state machine (FSM) for the obfuscation mode is added before the normal operation mode. Only the correct key sequence will drive the obfuscation FSM to take the path of $P_0 \rightarrow P_1 \rightarrow P_2$ and then reach the state S_5. As soon as the obfuscation FSM completes the correct transition path before the real initial state, the FSM output configures the modification cells in the combinational logic to realize the original circuit functionality. A wrong obfuscation key will lead to the FSM never reach the real initial state S_5 and turn the modification cell into a different logic function. The locations of the modified cells are vital to disguise the original functionality. Subhra and Bhunia [7] suggested that the nodes with large fan-in and fan-out logic cones are preferred for strong obfuscation. A larger fan-in logic cone typically indicates that the associated node may have a higher logic depth; while a larger fan-out logic cone will affect more nodes. Hence, it is more cost-effective to choose those logic cones for FSM-controlledmodification. The same

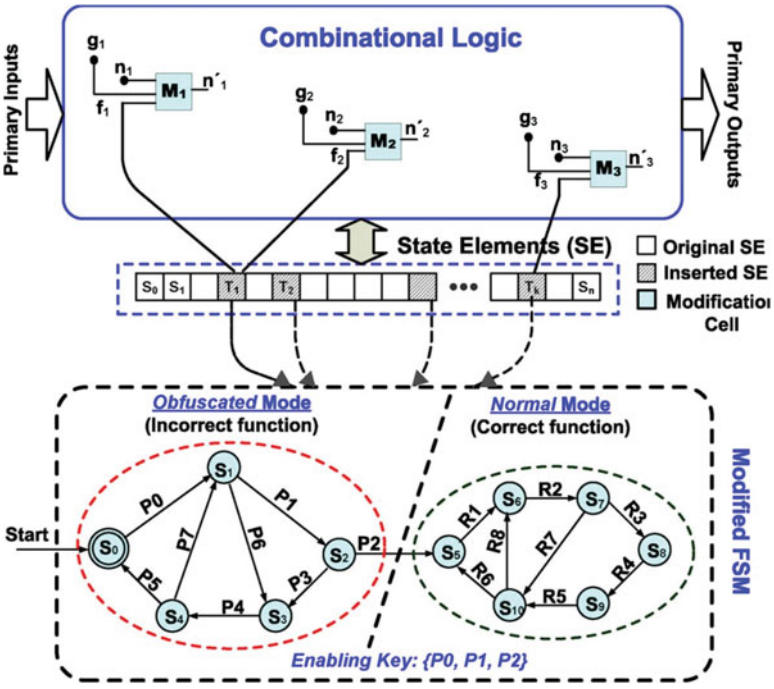


Fig. 12.14 HARPOON: functional and structural obfuscation by modification on STG and internal nodes [7]

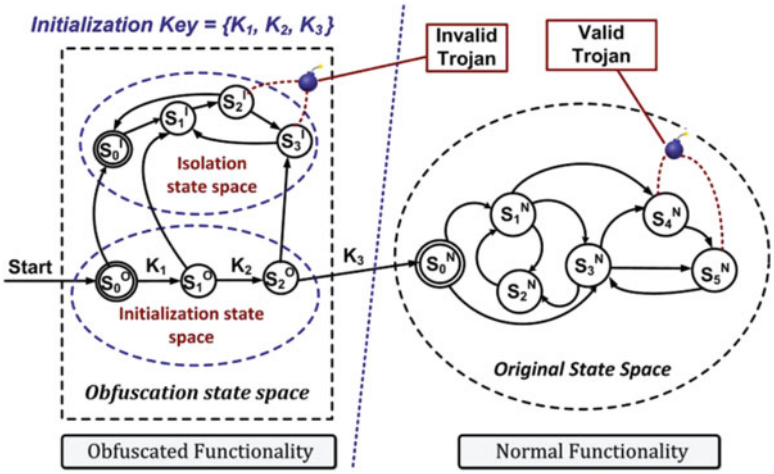


Fig. 12.15 The obfuscation scheme for protection against hardware Trojan [10]

authors further expanded the obfuscation FSM to include an authentication FSM in the same obfuscation mode to simultaneously thwart hardware tampering and piracy.

Later, the concept of that obfuscation method was proved to be effective for hardware Trojan prevention and detection as well [10, 11]. Different from the obfuscation for piracy prevention, the obfuscation for hardware Trojan prevention and detection, shown in Fig. 12.15, introduces an isolation state space to the obfuscation mode. The isolation states are either unused states from the original FSM, or new states are created by adding new bits to the original state vector. As the hardware Trojan has a nature of stealthiness, the rare events exploited by the Trojan designer have a high chance of falling into the isolated state space. Consequently, once the attacker does not have the correct obfuscation key, the hardware Trojans are confined in the obfuscation mode. Valid Trojans in the original state space may alter the original function if the Trojan trigger condition is satisfied, but an invalid obfuscation key will block the FSM from entering the normal operation mode, thus preventing the effect of Trojan from harming the system. The automatic obfuscation flow for gate-level netlist was developed in [10].

12.4.3.2 Dynamic State-Deflection Method

Obfuscation on the true power-up state of a finite state machine (FSM) has been demonstrated as a promising countermeasure to thwart IP piracy. The state machine cannot enter in the normal mode without the correct key. However, once the FSM operates in the normal operation mode, no more protection is available. This leaves a security vulnerability for attackers to exploit, especially in the context of soft or

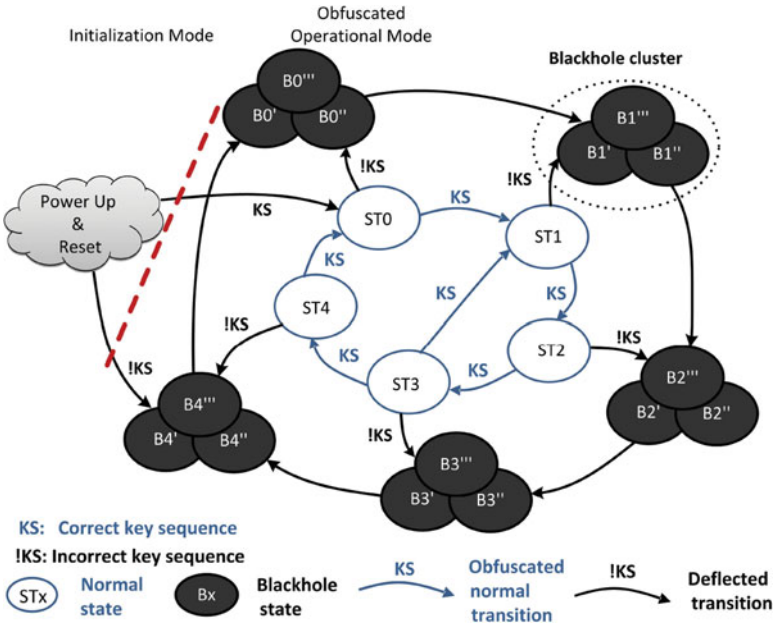


Fig. 12.16 Proposed dynamic state-deflection method for netlist obfuscation

firm IP piracy scenarios. To address this security vulnerability, a novel dynamic state-deflection method [19] is proposed to protect the states in the normal operation mode. As shown in Fig. 12.16, each normal state ST_x transition in the normal mode needs to pass the key authentication. If a wrong key KS is applied to the FSM, a normal state will be deflected to its black hole cluster B_x. The FSM never returns to the normal state once it enters the black hole cluster.

Each black hole cluster is composed of multiple states (only three shown), rather than a single one. Each black hole state can be mapped to a unique wrong key. Attackers can try a brute-force attack with the different wrong key attempts; hence the method in [19] utilizes a *Mapfunc* function to dynamically assign one black hole state to one wrong key case. The other significant feature of the method is that the state within the black hole cluster does not remain stable. Instead, the FSM constantly switches to other black hole states to obscure the attacker from reverse engineering.

Considering that a gate-level netlist does not provide a clear picture of used and unused states, a state flip vector *FlipReg* is used to selectively invert the state bits and a logic flip vector *FlipLogic* to modify the primary outputs. Thanks to the inherent feedback loops in the sequential circuit, the FSM state bits will naturally switch over time without using predetermined transition specifications.

Dynamic State-deflection in Black Hole Cluster: In the new FSM, each state (i.e., $\{S3, S2, S1, S0, B3, B2, B1, B0\}$) is composed of all the original state bits $\{S3, S2, S1, S0\}$ and several newly added flip-flops $\{B3, B2, B1, B0\}$ (for black hole

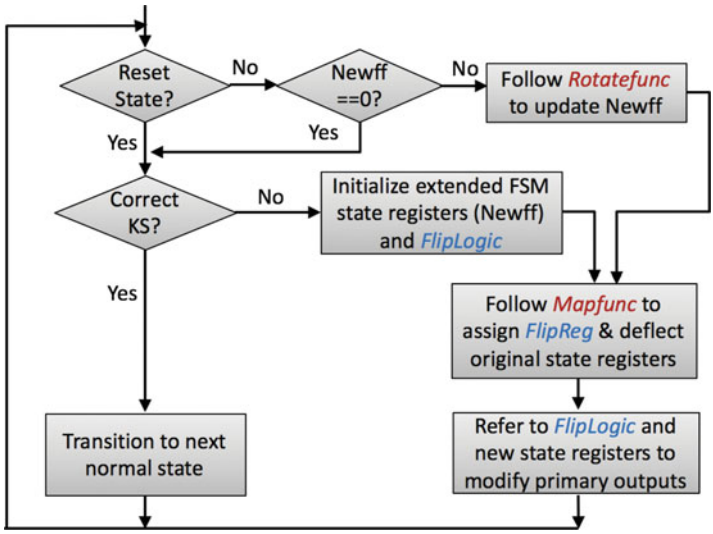


Fig. 12.17 Flow chart for the dynamic state-deflection method [19]

states). When the correct key sequence is applied, the newly added flip-flops $\{B3, B2, B1, B0\}$ are set to zero, and the new states remain consistent with the FSM state before the state extension. Hence, the newly added flip-flops do not change the original function if the key is correct. If the state of $\{B3, B2, B1, B0\}$ is not all zero, due to a wrong key, then the new state $\{S3, S2, S1, S0, B3, B2, B1, B0\}$ belongs to one of the black hole states in the cluster.

To raise the difficulty for reverse engineering attacks, the FSM is further hardened by creating dynamic transitions in black hole clusters. Different than the existing works [7, 8, 10], state transitions among the black hole states is not static and predefined. As highlighted in the obfuscation flow chart shown in Fig. 12.17, the dynamicity in the dynamic state-deflection method is achieved by utilizing the secret *Rotatefunc* and *Mapfunc* algorithms. A *Mapfunc* algorithm is used to deflect the original state bits from the correct state transition. The algorithm takes the newly added state bits and the wrong key sequence as an input to generate the *FlipReg* vector, which will selectively flip some bits in the FSM state. Each wrong key sequence will lead to a unique *FlipReg* vector.

A detailed example of dynamic state-deflection is shown in Fig. 12.18. If a wrong key sequence (!KS) is used, the new state bits are preset to a nonzero vector (e.g., 4'b0001), and the original state bits remain whatever they are defined in the original design. After cycle 0, the newly added state bits are modified by a *Rotatefunc* algorithm, which changes the vector $\{B3, B2, B1, B0\}$ to a new nonzero vector. The *Rotatefunc* is a circular left-shift function, but it could be any algorithm defined by the obfuscation provider. When $\{B3, B2, B1, B0\}$ is 4'b0001, the $S3$ original state bit is flipped by a *FlipReg* of 4'b1000. As the entire FSM state registers are deflected to a new value, the FSM will generate a new *FlipReg* vector of 4'b1101 after cycle 1. Thus, the $S2$ and $S0$ bits in the original state are then flipped to its complementary

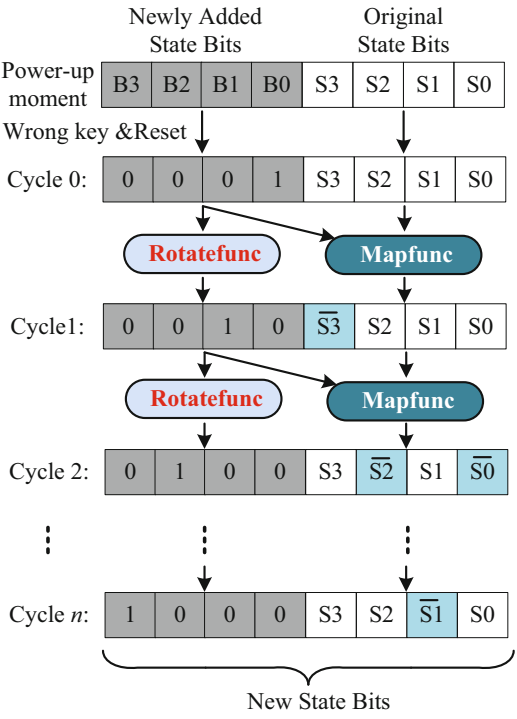


Fig. 12.18 An example of FSM state deflection induced by a wrong key sequence [19]

value, and $S3$ is inverted back at the end of cycle 2. Hence from the description, it is clear that the dynamic state-deflection method inverts FSM state bits pseudo-randomly and selectively, rather than setting the original state bits to a fixed vector. As a result, the black hole state is not determined at the design time and dynamically selected depending on the applied wrong key. The correct key can be hard wired in the netlist or saved in a trusted memory. The former one is less secure (but more hardware efficient) than the latter one to a certain degree.

12.4.3.3 Obfuscation Using Reconfigurable Logic

Alternatively to key-controlled obfuscation method discussed in Sects. 12.4.3.1 and 12.4.3.2, a logical function can also be obfuscated with post-fabrication reconfigurable logic. Liu and Wang utilized LUT-based reconfigurable implementation to achieve design obfuscation [40]. The reconfiguration-based logic turns the functional unit into a moving target, rather than a determined attack node. Wendt and Potkonjak proposed to implement a portion of the functional module with a combination of a physically unclonable function (PUF) unit and a FPGA [66]. As the output of a PUF is unpredictable and the FPGA is not configured at design time, the functionality of the module is not completely implemented before fabrication.

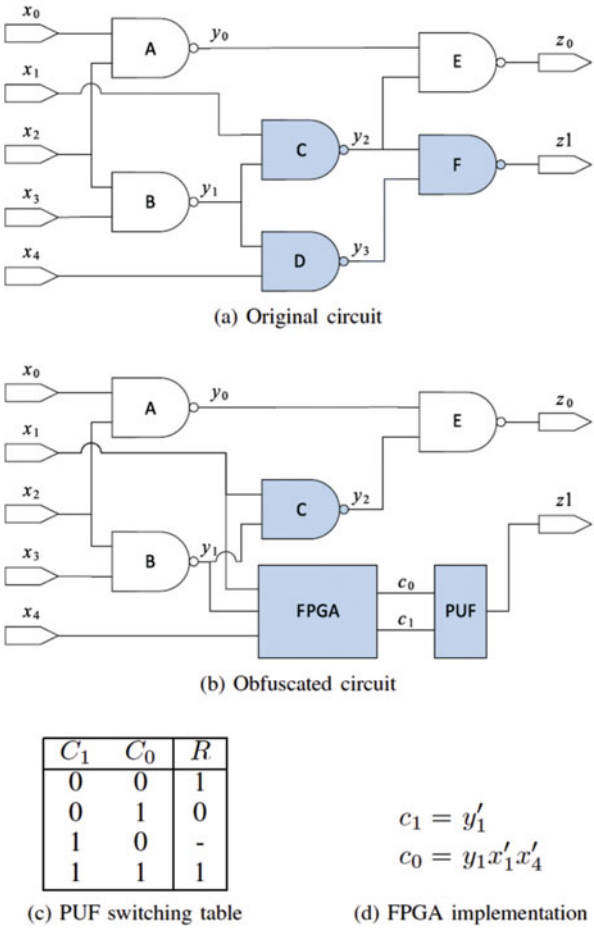


Fig. 12.19 Hardware obfuscation using PUF and FPGA based logic [66]

C_1	C_0	R
0	0	1
0	1	0
1	0	-
1	1	1

$$c_1 = y_1'$$
$$c_0 = y_1 x_1' x_4'$$

Hence, theoretically, it is not a trivial task for an attacker to identify extremely rare events for a hardware Trojan insertions. Moreover, the blindly inserted Trojan may be nullified after post-fabrication configuration. It is reasonable to assume that the incomplete function unit composed of PUF and FPGA will be reserved to realize the security-critical functions for the system. In other words, that incomplete function unit has the potential to be the Trojan target. Functional testing on the circuit obfuscated with PUF and FPGA can detect the hardware Trojans in the obfuscated circuit. Figure 12.19 shows one simplified example of PUF-based logic obfuscation. The NAND gates *D* and *F* are obfuscated by FPGA and PUF. In the work [66], the authors also analyze possible ways to obfuscate the interconnect network with the PUF-based logic.

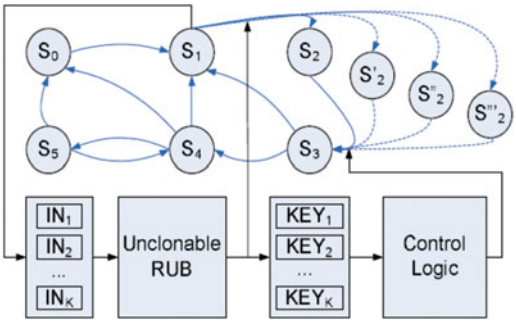


Fig. 12.20 Interlocking based obfuscation [1]

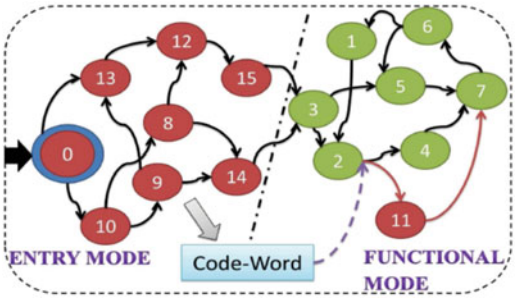


Fig. 12.21 Interlocking based obfuscation [18]

12.4.4 Register-Transfer Level

A remote activation scheme [1] shown in Fig. 12.20 was initially proposed to resist IP piracy. The continuousness of state transition is granted by a key uniquely generated by the random unique block on the chip. This method duplicates one of the existing states three times to stop the transition from the state under protection to the next valid state when a wrong key is applied to the FSM. In this method, an attacker could identify the stalled state, then force the state registers to change to other values. Thus, the obfuscation mechanism may fail.

The work [18] suggests using obfuscation to thwart hardware tampering. Figure 12.21 illustrates the main idea of that work. Different from using an external key sequence in [7], this work locks the design with an internally generated *code word* that is based on a special order of state transitions chosen as secret. Although this method may be helpful to address the issue of key leaking in [7, 8, 10], the degree of circuit obfuscation is significantly reduced. This is because a wrong key only causes the FSM to temporarily deviate from the correct transition path; the normal state transition will resume (after one or few state transitions). Another limitation of this method is, if the internal code word generation module is compromised, the obfuscation on the entry mode will be nullified.

The piracy-aware IC design flow in [51] proposes to enrich the RTL description with an on-chip true random number generator and public-key cryptography (RSA). The authors further extend their work in [37]. Besides the idea of an obfuscation

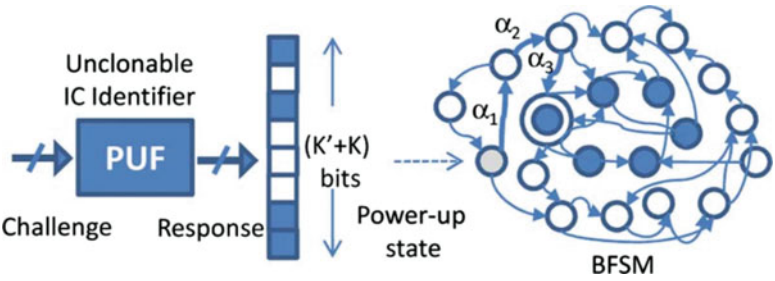


Fig. 12.22 Black hole state obfuscation [37]

mode similar to the work in [8], the authors in [37] also add a channel from the normal state transition to the black hole states, which prohibits the state recovery to the normal operation. The path from normal states to black hole ones provides protection for the normal operation mode, as shown in Fig. 12.22.

Subhra and Bhunia extended their gate-level obfuscation method to register-transfer level (RTL) [8, 9, 11]. They transform the RTL description to gate-level netlist using synthesis tools and then utilize their new STG and modification cells to obfuscate the synthesized netlist [8]. In [11], Subhra et al. propose an on-demand transparency method to facilitate logic testing based Trojan detection. First of all, Subhra et al. define a transparent mode, in which the circuit generates a specific signature based on a user-defined key and passes that signature to the primary output. If a hardware Trojan is inserted to the circuit, the specific signature will be different from the expected one. Thus, we can examine the signature to detect the presence of hardware tampering. Figure 12.23a summarizes the on-demand transparency scheme. For a multi-module design, this scheme can be applied in a cascade fashion, as shown in Fig. 12.23b, until the specific signature is transferred to the primary output port.

12.4.5 On-Chip Communication Level

As billions of transistors are integrated on a single die, Networks-on-Chip (NoCs) emerge as an efficient on-chip communication infrastructure [4, 17, 62]. The consequence of using a compromised NoC has been highlighted in [2, 5, 26, 31, 46]. The hardware Trojans inserted in NoCs will lead to information leakage, unauthorized memory access, and denial-of-service attacks, such as incorrect path routing, deadlock, and livelock. This section introduces a runtime hardware Trojan detection and mitigation method to harden the NoC against tampering. Compared to the existing approaches, this method fully considers the NoC features, including high modularity, large scalability, and parallel transmission channels, while strengthening the NoC capability to resist potential hardware Trojan attacks.

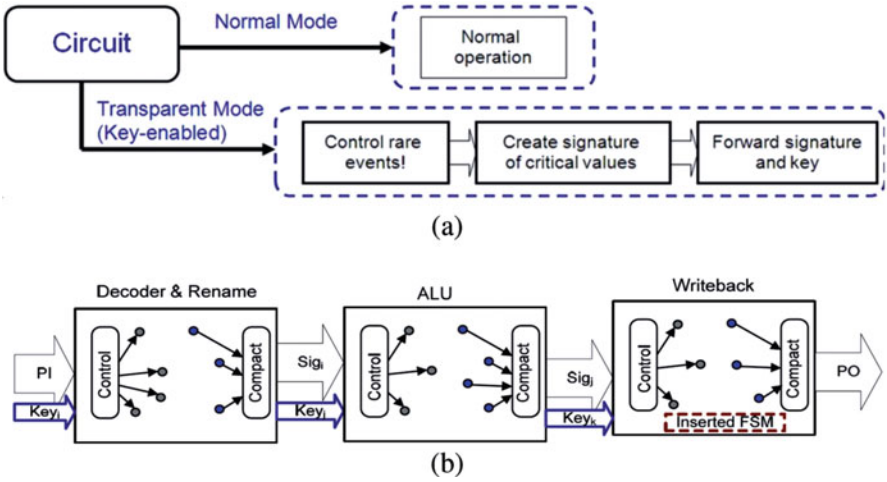


Fig. 12.23 On-demand transparent scheme for hardware Trojan detection. (**a**) Detection scheme and (**b**) signature propagation in a multi-module design [11]

In the previous work [24, 36], examination on the memory access range is used as a main method to ensure the security of NoC. However, those countermeasures have a common limitation – the memory range checking unit itself is lack of the protection (unless a cipher is used in the memory). Moreover, after the packet passes the address filter, no further protection is available in the network interface (NI) or router. Unfortunately, NIs and routers can be compromised after the route computation stage. For instance, one may modify the memory address of a packet that is saved in the output buffer. Many hardware Trojan countermeasures for NoCs [23, 24, 45] are mainly located in NIs, rather than routers. Routers have five interconnection channels with neighboring nodes (i.e., routers and NI); in contrast, NIs only have one connection to its IP core and one connection to its router. The hardware Trojan mitigation method in [25] aims to detect and mitigate the hardware Trojan attacks that (1) modify the flit type, (2) change the legal packet destination address to an unauthorized one, and (3) sabotage the integrity of a packet. The main consequence of the hardware Trojan targeted in [25] is the NoC bandwidth depletion.

As shown in the left side of Fig. 12.24, a generic mesh-NoC is composed of links, NIs, and routers with five ports (north, east, south, west, and local input/output ports). A pair of input and output ports (shadowed area in Fig. 12.24) is used as an example to present the novelty of the router design. The zoomed in picture on the right side of Fig. 12.24 depicts the high-level view of the hardware Trojan countermeasures (HTC) applied in the south input port and the east output port. In the input port, the HTC1 module dynamically permutes the incoming NoC flit bits before reaching the FIFO. As a result, the NoC flit saved in the input FIFO is scrambled. Due to the randomness and dynamicity of the permutation patterns, it is much harder for an attacker to change the flit content into something meaningful

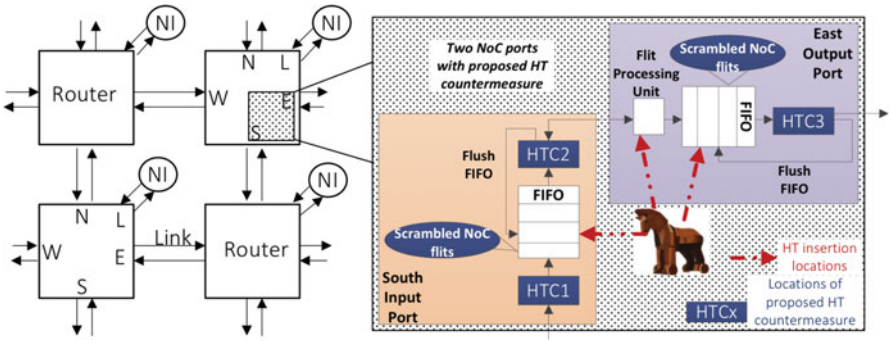


Fig. 12.24 A high-level overview of the countermeasure for hardware Trojan prevention and detection for on-chip communication network [25]

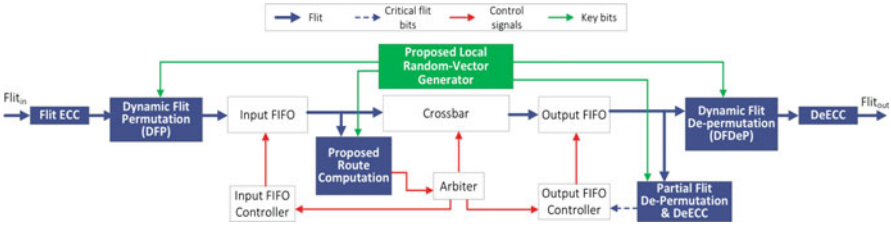


Fig. 12.25 Obfuscated router architecture proposed in [25]. Solid blocks highlight the proposed innovations over the generic router architecture

through hardware Trojans in the proactively defended router than in a router with static protection. The goal of the flit permutation in [25] is to reduce the probability of a hardware Trojan inserted in the FIFO successfully modifying the critical bits in the NoC packet. The HTC2 module is used to examine the integrity of flits, which could be sabotaged by the hardware Trojans placed in the input FIFO. The main function of the HTC2 is to prevent the malicious flits from entering the rest of the network by dropping the flit and flushing the input FIFO if necessary. Hardware Trojans could also be located in the flit processing unit and the FIFO in the east output port. Hence, a HTC3 module is placed after the FIFO to stop the malicious flit from leaving the current router and then recover the scrambled flit back to the original bit order. The three HTC modules proposed in Fig. 12.24 cooperatively provide a proactive defense to thwart the potential hardware Trojans that harm the NoC flit integrity and thus deplete the NoC bandwidth.

The HTC modules in Fig. 12.24 are detailed in Fig. 12.25. The highlighted solid blocks in Fig. 12.25 are the changes over the generic router. To guarantee a packet has reached its original destination, the critical fields (i.e., source/destination address and flit type bits) in a NoC packet cannot be tampered. To maintain the flit integrity despite the potential hardware Trojans in the router, the method in [25] first encode the critical flit fields with an error control code (ECC). Next, apply a dynamic flit

permutation (DFP) technique to the packet buffers so that critical information bits in a NoC flit are randomly reordered before that flit is sent to the input FIFO. If the attacker could not precisely know the dynamic permutation pattern at each specific moment, the hardware Trojans he/she inserted in the router will not be triggered as expected or execute the malicious mission as designed. Symmetrically, the dynamic flit de-permutation (DFDeP) is applied at the output port. After de-permutation, an ECC decoder (DeECC) facilitates the router to detect and correct all 1-bit errors in the critical flit content.

The permutation randomness is achieved by a pattern selection vector, which is dynamically generated by the Local Random-Vector Generator. Each router has its own random number generator, which provides the selection vector to choose one of the flit permutation patterns for the DFP and DFDeP units. The unique routing history saved in the arbiter is exploited to generate a random vector through a PUF structure implemented in each router. Due the uniqueness of each PUF and the routing history, the dynamic permutation pattern for DFP and DFDeP varies over time and is different from router to router. As the packet routing history in each router depends on the NoC application at deployment stage, the output of Local Random-Vector Generator is unpredictable at the NoC design stage. Thus, the adversary cannot easily place hardware Trojans to perform successful attacks at the design time. The proposed route computation unit is designed to interpret multiple versions of the permutated flit content into a request to use the correct output port for the next routing hop. To reduce the hardware cost, partial flit de-permutation and DeECC are proposed to correctly interpret important internal and external signals such as write request and FIFO buffer status signals.

Furthermore, the flit bit positions are permuted before storing it to the input buffer and de-permute the flit after the output buffer. The principle of proposed permutation is as expressed in (12.1). The pattern selection vector is the output of a PUF-based structure and varies with process variations and the dynamic challenge vector X, as expressed in (12.2):

$$\mathrm{flit}_{dp}(t,i,j) = \Phi[\mathrm{flit}(i,i,j), \mathrm{PPS}(t,i,j)] \tag{12.1}$$

$$\mathrm{PPS}(t,i,j) = f\big(\mathrm{ProcessVariation}, X(t,i,j)\big) \tag{12.2}$$

In which, the variables i and j in (12.1) and (12.2) mean the router ID and the input port ID, respectively. Variables flit(t) and flitdp(t) are the original flit and the flit after dynamic permutation at the time t, respectively. The function ϕ stands for the proposed dynamic permutation algorithm, which is a function of the incoming flit and a unique PPS vector. Equation (12.1) is not an explicit mathematic expression of flit permutation; instead, this equation is used to show the dependant factors for the proposed dynamic permutation. The challenge vector $X(t,i,j)$ depends on the routing history in each router's output port. As each router has a unique and unpredictable routing history, the random $\mathrm{PPS}(i,j,t)$ and permutation configurations are unique for each router's input port at different moments. The permutation circuit is shown in Fig. 12.26.

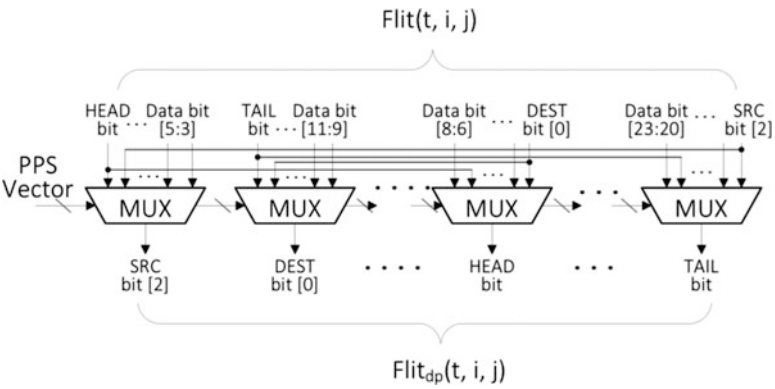


Fig. 12.26 Schematic for the PUF-based flit permutation function ϕ. The permutation algorithms are implemented with hardwiring before the multiplexers [25]

12.4.6 Other Methods

Split manufacturing methods [29, 43, 60, 67, 69] obfuscate the design by dividing a circuit into multiple tiers, which are sent to different foundries for fabrication. As each foundry does not have the complete design, attackers from the untrusted foundry have limited understanding on the entire design and may not be able to insert effective hardware Trojans to the design portion they have. In recent years, researchers have realized that split manufacturing may not be as necessarily secure as expected [41, 49, 61, 65].

12.5 FPGA Obfuscation

Hardware Trojan attacks can also happen to FPGAs. Trojans may be inserted in an unencrypted configuration bitstream [12] to cause redundant circuit switching, which results in increased power consumption and circuit operating temperature. Even if the configuration bitstream is protected with encryption, attackers can use algorithms [55] and Boolean matching [56] to reverse engineer a device netlist from its bitstream. The recent attack presented in [38] slightly alters the hardware description language (HDL) code in a way that the inserted Trojan can evade the detection from synthesis tools or other existing security measures.

A multitude of countermeasures do exist with the purpose of detecting and thwarting hardware Trojan in FPGAs. One particular method is built on the concept of code flattening [33]. That method flattens the program source code and obfuscates the control flow through hardware-assisted branching functions to prevent exploits or modifications to source code. In addition to bitstream encryption, the work in [34] proposes design obfuscation through the use of a key bit. The fact that the look-up

tables (LUTs) in FPGAs are seldom fully utilized indicates that extra memory bits are always available. We can store the key bits for obfuscation in the unused memory within LUTs. Attackers now have increased difficulty in identifying the functions inside the LUTs; hence, they have less chances to successfully perform a malicious modification attack in that FPGA. Another countermeasure utilizing the free space on FPGAs is inserting dummy logic [35]. All unused resources such as flip-flops, LUTs, and multiplexers are instantiated with low-level HDL files without changing the original design functionality. Attackers now have to deal with a substantially harder bitstream to reverse engineer and have less space to insert a Trojan in the unused clusters. Similarly, a constant value generator (CVG) circuit [53] can be placed in the design for a form of lexical obfuscation. Here, 0 and 1 pins are permanently substituted with appropriate logical values, creating fictive connections and increased dependencies between circuit nodes. When employed, the HDL code is rendered illegible to humans, and the circuit becomes more difficult to analyze.

12.6 Board-Level Obfuscation

Obfuscation at PCB level is even harder than that at chip level because attackers can directly probe the board and bypass the applied countermeasures for Trojan insertion. The work [28] proposes an intelligent permutation method, which permutes the unused general ports (as dummy ports) together with the other ports for dedicated purposes. The permutation is controlled by a secret key. Ghosh et al. [27] propose to realize design obfuscation by incorporating dummy chips at the PCB level. The dummy chips scramble the traces on the board and mix their fake output signals with the real signals generated from the original components. Moreover, the dummy chips can also provide security for joint test access group (JTAG), which has to be exploited if adversaries want to reverse engineer the PCB design.

12.7 Evaluation Metrics for Hardware Obfuscation

The common metrics used to define the obfuscation strength are defined in the Table 12.5. The metrics shown in rows from 1 to 7 are quantifiable, while ones in row 8 are not quantifiable (obfuscation degree is not proved through these metrics).

Hamming Distance: Hamming distance (HD) between two binary vectors of equal length measures the number of positions at which the vectors have different values. In security perspective, HD is measured by comparing the outputs of the design on applying the valid key and a wrong key. An attacker can utilize this metric to reverse engineer the design. HD approaching 50% is desirable, which indicates that the functionality of the reverse engineered logic diverges substantially from the original functionality. In combinational circuits the output HD is used to evaluate

Table 12.5 Evaluation metrics for hardware obfuscation

Sr. No	Evaluation	Obfuscation method	Security evaluation metric
1.	Q	Key-based obfuscation [19]	—— Register and output Hamming distance —— Code coverage —— Autocorrelation coefficient for FSM registers
2.	Q	Logic encryption [50]	—— Output hamming distance
3.	Q	Layout obfuscation [49]	—— Proximity attack correctness
4.	Q	Transistor-level logic locking [20]	—— Output hamming distance
5.	Q	Circuit similarity-based partition and trimming method for obfuscation [13]	—— Circuit similarity
6.	Q	Key-based control and data flow obfuscation [9]	—— Potency —— Resilience and stealth
7.	Q	Permutation of interconnections by a secret key [28]	—— Bruteforce attack attempts
8.	NQ	Split-fabrication obfuscation [29, 43]	—— Neighbor connectedness —— Cell-level obfuscation

Q quantifiable metric, *NQ* nonquantifiable metric

the obfuscation strength [50]. A similar idea is extended for sequential circuits in [19] where along with output HD, the register HD is also assessed for defining the resilience toward reverse engineering attacks. The mathematical expression for output HD is represented in (12.3):

$$\mathrm{OHD}(t) = \frac{\mathrm{Out}(t) \wedge \mathrm{Out}'(t)}{\text{Total Number of Outputs}} \tag{12.3}$$

Where the $\mathrm{Out}(t)$ and $\mathrm{Out}'(t)$ are outputs when the correct key and incorrect key are applied at time instance t.

Code Coverage: Code coverage is a metric that evaluates the effectiveness of a test bench in exercising the design. Code coverage analysis report (e.g., from Cadence Incisive Comprehensive Coverage (ICCR)) clearly highlights which nets of the design under test are toggled or never toggled in the given test bench. This metric can be used by an attacker to reverse engineer a design using advanced EDA tools like ICCR. The high coverage of obfuscated design with wrong key sequences indicates that the method will yield less net toggle complementary percentage for correct and wrong key scenarios. The high value of code coverage is desirable for successful obfuscated design. It can be obtained by blending the dummy (extra states added for obfuscation) states with the original states to make it difficult for attackers to identify the dummy states through code coverage analysis.

Autocorrelation Coefficient for FSM Registers: If the FSM register contents are repeated in a predictable way for the scenarios of wrong key sequences, attackers may get a clue to correlate the key sequence with the register pattern. In the ideal case, the FSM register content should randomly change without a clear pattern, even in the obfuscation mode. Unfortunately, due to the constraint on hardware cost, once the FSM enters the obfuscation mode, the FSM register pattern may be repeated after a while. The desired value of autocorrelation coefficient for FSM registers should be low.

Proximity Attack Correctness: This metric is related to proximity attacks [49, 68] that can be used to conduct the reverse engineering attacks in split manufacturing fabrication using layout geometry information under the assumption that two connected gates are placed nearby. The attacker can be an untrusted foundry that has access to the GDSII layout files of two sub-netlists, but he/she lacks the knowledge of the correct connection between them. Proximity attack correctness can be used to measure the security of split manufacturing and defined as the percentage of correct connections that are recovered by the proximity attack algorithm [49]. The desired proximity attack correctness should be around 0% for a secure layout design. The lowest value indicates the inability of an attacker to reveal the correct connections within a design.

Circuit Similarity: This metric demonstrates the actual similarity degree of two circuit structure, including both the topology and all containing logic gates. The bigger the circuit similarity is, the harder it is to differentiate similar sub-circuits and attack particular cells in one specific sub-circuit. The fact that circuits have their own internal structure similarity which can be used to increase the difficulty of malicious insertions in the design if the value of the metric is high.

Potency: This metric is the complexity in comprehending the obfuscated program compared to the unobfuscated one. In digital design, it is estimated by the structural and functional differences between the obfuscated and the original circuit. The success of the functional/structural modification is evaluated by the percentage of nodes failing verification when the original and the obfuscated designs are subjected to formal verification. This is because the modifications in the control and data flow of the circuit have a direct impact on the Boolean functionality of the internal nodes in the circuit. A higher percentage of verification failures indicates that the obfuscated design differs substantially from the original design in both functionality and structure.

Resilience and Stealth: This metric is defined in [9]. Instead of having separate mode-control FSM (which is used as an obfuscation FSM), the state element of mode-control FSM can be hosted in the original registers to increase the level of obfuscation. Once the FSM is hosted in set of host registers, several control and data flow graph nodes are modified with the help of control signals generated by FSM. The resilience and stealth of the obfuscation scheme is calculated by the level of difficulty faced by an attacker in inventing the hosted mode-control FSM and the modification signals. The mathematical representation of this metric is shown in (12.4):

$$M_{\text{obf}} = \frac{1}{\sum_{k=1}^{n} \left(\binom{N}{k} \cdot k! \right) \cdot \binom{M}{m}} \tag{12.4}$$

where n is mode-control registers hosted with N blocking/non-blocking assignment statements, M is the total no. of blocking/non-blocking assignment statements in design, and m is size of the modification signal pool.

A small value of M_{obf} is desirable as it indicates the greater obfuscation efficiency. The values of n and M in real circuits could be much greater making M_{obf} smaller and increasing the reverse engineering efforts of attackers.

Neighbor Connectedness: This metric is used in split-fabrication obfuscation. It measures the proximity of the interconnected cells to their neighbors. A higher value of this metric reveals more details of the circuit by exploring the location of cells and their neighbors. A lower value shows that the cells are spread out more in the circuit and neighboring cells and do not divulge any information of the design. Hence, the circuit is more difficult to reverse engineer. A method to determine the neighbor connectedness can be determined by calculating the number of neighbors, a specific cell has and how many connections it has to those for a given area around the cell; this relationship is represented in (12.5):

$$C(R) = \frac{\sum \text{Connections to the neighbors in } R}{\sum \text{neighbors in } R} \tag{12.5}$$

Cell-Level Obfuscation: This metric also designed for split manufacturing obfuscation, where a circuit is split into a front half, incorporating front-end-of-line (FEOL) processing, which is sent to a first foundry, and a back half, incorporating the remainder of the back-end-of-line (BEOL) wiring interconnect [30]. Finding the details of standard cell is one of the first steps performed during reverse engineering attacks. This metric measures the percentage of standard cells that have been obfuscated to hide their functionality in the FEOL layers. To evaluate this metric one has to calculate the percentage of cells that have been modified through either a finer or coarser cell implementation and are no longer identifiable visually.

Brute-force Attempts: Brute-force attack is a trial and error method used by attackers for gaining access to restricted designs through exhaustive effort rather than employing intellectual strategies. This metric is applicable for the most of the obfuscation method whose objective is to protect designs from reverse engineering. The probability of success for brute-force attacks depends heavily on the number of combinations (obfuscation strength) and the time required to validate the system for correct behavior after applying each key.

12.8 Summary

Complementary to testing, SEM image-based analysis, and side-channel signal analysis, hardware obfuscation methods facilitate chip designers to build a defense line in their product to thwart potential hardware tampering, reverse engineering, IP privacy, and cloning attacks. In particular, literatures reviewed in this chapter demonstrate that hardware obfuscation at different abstraction levels is a promising solution to prevent hardware Trojan insertion or detect the malicious modification made by untrusted entities in the IC supply chain. The obfuscated hardware design makes it more difficult for attackers to understand the original circuit functionality. Thus, it will be more challenging for attackers who lack the full understanding of the design to successfully place hardware Trojans in the obfuscated system for several reasons. First, the hardware Trojans may be triggered more often than the attacker expected because the rareness of the Trojan triggering condition may not be true in the original design. Second, because of hardware obfuscation, attackers who do not have the correct secret key may not be able to access the states for the normal operation mode. Consequently, the inserted hardware Trojan is likely to be isolated or never triggered. Lastly, the obfuscation process can also be incorporated with authentication to generate a special signature for Trojan detection.

Despite significant growth that has been made on hardware obfuscation, there is a great need of quantifiable evaluation metrics for obfuscation methods. ISCAS benchmarks are typically used to assess the hardware overhead and performance degradation induced by various hardware obfuscation methods. More meaningful benchmarks would help the community to perform fair and effective comparison.

References

1. Y. Alkabani, F. Koushanfar, M. Potkonjak, Remote activation of ICs for piracy prevention and digital right management, in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, Nov 2007, pp. 674–677
2. D.M. Ancajas, K. Chakraborty, S. Roy, Fort-NoCs: mitigating the threat of a compromised NoC, in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6
3. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, K. Yang, On the (im) possibility of obfuscating programs, in *Annual International Cryptology Conference* (Springer, 2001), pp. 1–18
4. L. Benini, G.D. Micheli, Networks on chips: a new SoC paradigm. Computer **35**, 70–78 (2002)
5. S. Bhunia, M.S. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan attacks: threat analysis and countermeasures. Proc. IEEE **102**, 1229–1247 (2014)
6. R.S. Chakraborty, S. Bhunia, Hardware protection and authentication through netlist level obfuscation, in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design* (IEEE Press, 2008), pp. 674–677
7. R.S. Chakraborty, S. Bhunia, Harpoon: an obfuscation-based SOC design methodology for hardware protection. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **28**(10), 1493–1502 (2009)

8. R.S. Chakraborty, S. Bhunia, Security through obscurity: an approach for protecting register transfer level hardware IP, in *Proceedings of the 2009 IEEE International Workshop on Hardware-Oriented Security and Trust, HST'09* (IEEE Computer Society, Washington, DC, 2009), pp. 96–99
9. R.S. Chakraborty, S. Bhunia, RTL hardware IP protection using key-based control and data flow obfuscation, in *VLSI Design* (2010)
10. R.S. Chakraborty, S. Bhunia, Security against hardware Trojan attacks using key-based design obfuscation. J. Electron. Test. **27**(6), 767–785 (2011)
11. R.S. Chakraborty, S. Paul, S. Bhunia, On-demand transparency for improving hardware Trojan detectability, in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust* (June 2008), pp. 48–50
12. R.S. Chakraborty, I. Saha, A. Palchaudhuri, G.K. Naik, Hardware Trojan insertion by direct modification of FPGA configuration bitstream. IEEE Des. Test **30**, 45–54 (2013)
13. Y. Cheng, Y. Wang, H. Li, X. Li, A similarity based circuit partitioning and trimming method to defend against hardware Trojans, in *2015 IEEE Computer Society Annual Symposium on VLSI* (July 2015), pp. 368–373
14. Circuit camouflage technology (2012), p. 697. http://www.smi.tv/SMI_SypherMedia_Library_Intro.pdf
15. R. Cocchi, L. Chow, J. Baukus, B. Wang, Method and apparatus for camouflaging a standard cell based integrated circuit with micro circuits and post processing, 13 Aug 2013. US Patent 8,510,700
16. R.P. Cocchi, J.P. Baukus, L.W. Chow, B.J. Wang, Circuit camouflage integration for hardware IP protection, in *Proceedings of Design Automation Conference (DAC)*, June 2014, pp. 1–5
17. W.J. Dally, B. Towles, Route packets, not wires: on-chip inteconnection networks, in *Proceedings of the 38th Annual Design Automation Conference, DAC'01* (ACM, New York, 2001), pp. 684–689
18. A.R. Desai, M.S. Hsiao, C. Wang, L. Nazhandali, S. Hall, Interlocking obfuscation for anti-tamper hardware, in *Proceedings of Cyber Security and Information Intelligence Research Workshop (CSIIRW)*
19. J. Dofe, Q. Yu, Novel dynamic state-deflection method for gate-level netlist obfuscation. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (99), 1–1 (2017)
20. J. Dofe, C. Yan, S. Kontak, E. Salman, Q. Yu, Transistor-level camouflaged logic locking method for monolithic 3D IC security, in *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, Dec 2016, pp. 1–6
21. G. D'Souza, D. Laird, M. Wing, C. Murphy, D. How, R. Yu, J. Patel, I. Dobbelaere, J. Golbus, S. Subramaniam et al., Heterogeneous configurable integrated circuit, 19 Mar 2009. US Patent App. 11/855,666
22. S. Dupuis, P.S. Ba, G.D. Natale, M.L. Flottes, B. Rouzeyre, A novel hardware logic encryption technique for thwarting illegal overproduction and hardware Trojans, in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, July 2014, pp. 49–54
23. L. Fiorin, G. Palermo, S. Lukovic, C. Silvano, A data protection unit for NoC-based architectures, in *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS)*, Sept 2007, pp. 167–172
24. L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, C. Silvano, Secure memory accesses on networks-on-chip. IEEE Trans. Comput. **57**, 1216–1229 (2008)
25. J. Frey, Q. Yu, A hardened network-on-chip design using runtime hardware Trojan mitigation methods. Integr. VLSI J. **56**, 15–31 (2017)
26. C.H. Gebotys, R.J. Gebotys, A framework for security on NOC technologies, in *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, Feb 2003, pp. 113–117
27. S. Ghosh, A. Basak, S. Bhunia, How secure are printed circuit boards against Trojan attacks? IEEE Des. Test **32**, 7–16 (2015)
28. Z. Guo, M. Tehranipoor, D. Forte, J. Di, Investigation of obfuscation-based anti-reverse engineering for printed circuit boards, in *Proceedings of the 52nd Annual Design Automation Conference* (ACM, 2015), p. 114

29. M. Jagasivamani, P. Gadfort, M. Sika, M. Bajura, M. Fritze, Split-fabrication obfuscation: metrics and techniques, in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, May 2014, pp. 7–12
30. R. Jarvis, M. McIntyre, Split manufacturing method for advanced semiconductor circuits, 27 May 2004. US Patent App. 10/305,670
31. R. JS, D.M. Ancajas, K. Chakraborty, S. Roy, Runtime detection of a bandwidth denial attack from a rogue network-on-chip, in *Proceedings of the 9th International Symposium on Networks-on-Chip, NOCS'15* (ACM, New York, 2015), pp. 8:1–8:8
32. K. Juretus, I. Savidis, Reduced overhead gate level logic encryption, in *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI, GLSVLSI'16* (ACM, New York, 2016), pp. 15–20
33. M. Kainth, L. Krishnan, C. Narayana, S.G. Virupaksha, R. Tessier, Hardware-assisted code obfuscation for FPGA soft microprocessors, in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar 2015, pp. 127–132
34. R. Karam, T. Hoque, S. Ray, M. Tehranipoor, S. Bhunia, Robust bitstream protection in FPGA-based systems through low-overhead obfuscation, in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Nov 2016, pp. 1–8
35. B. Khaleghi, A. Ahari, H. Asadi, S. Bayat-Sarmadi, FPGA-based protection scheme against hardware Trojan horse insertion using dummy logic. IEEE Embed. Syst. Lett. **7**, 46–50 (2015)
36. L.-W. Kim, J.D. Villasenor, c.K. Koç, A Trojan-resistant system-on-chip bus architecture, in *Proceedings of the 28th IEEE Conference on Military Communications, MILCOM'09* (IEEE Press, Piscataway, 2009), pp. 2452–2457
37. F. Koushanfar, Provably secure active IC metering techniques for piracy avoidance and digital rights management. IEEE Trans. Inf. Forensics Secur. **7**, 51–63 (2012)
38. C. Krieg, C. Wolf, A. Jantsch, Malicious lut: a stealthy FPGA Trojan injected and triggered by the design flow, in *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD'16* (ACM, New York, 2016), pp. 43:1–43:8
39. Y.W. Lee, N.A. Touba, Improving logic obfuscation via logic cone analysis, in *2015 16th Latin-American Test Symposium (LATS)*, Mar 2015, pp. 1–6
40. B. Liu, B. Wang, Reconfiguration-based VLSI design for security. IEEE J. Emerg. Sel. Top. Circuits Syst. **5**, 98–108 (2015)
41. J. Magana, D. Shi, A. Davoodi, Are proximity attacks a threat to the security of split manufacturing of integrated circuits?, in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–7
42. K. Nikawa, Laser-squid microscopy: novel nondestructive and non-electrical-contact tool for inspection, monitoring and analysis of LSI-chip-electrical-defects, in *Digest of Papers. Microprocesses and Nanotechnology 2001. 2001 International Microprocesses and Nanotechnology Conference (IEEE Cat. No.01EX468)*, Oct 2001, pp. 62–63
43. C.T.O. Otero, J. Tse, R. Karmazin, B. Hill, R. Manohar, Automatic obfuscated cell layout for trusted split-foundry design, in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, May 2015, pp. 56–61
44. S.M. Plaza, I.L. Markov, Solving the third-shift problem in IC piracy with test-aware logic locking. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**, 961–971 (2015)
45. J. Porquet, A. Greiner, C. Schwarz, NoC-MPU: a secure architecture for flexible co-hosting on shared memory mpsocs, in *2011 Design, Automation Test in Europe*, Mar 2011, pp. 1–4
46. A. Prodromou, A. Panteli, C. Nicopoulos, Y. Sazeides, Nocalert: an on-line and real-time fault detection mechanism for network-on-chip architectures, in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2012, pp. 60–71
47. J. Rajendran, Y. Pino, O. Sinanoglu, R. Karri, Security analysis of logic obfuscation, in *DAC Design Automation Conference*, June 2012, pp. 83–89
48. J. Rajendran, M. Sam, O. Sinanoglu, R. Karri, Security analysis of integrated circuit camouflaging, in *Proceedings of Computer Communications Security (CCS)* (2013), pp. 709–720
49. J.J. Rajendran, O. Sinanoglu, R. Karri, Is split manufacturing secure?, in *Proceedings of DATE'13* (2013), pp. 1259–1264

50. J. Rajendran, H. Zhang, C. Zhang, G.S. Rose, Y. Pino, O. Sinanoglu, R. Karri, Fault analysis-based logic encryption. IEEE Trans. Comput. **64**, 410–424 (2015)
51. J. Roy, F. Koushanfar, I. Markov, EPIC: ending piracy of integrated circuits, in *Proceedings of DATE'08*, Mar 2008, pp. 1069–1074
52. M. Schobert, *Softwaregestutztes reverse-engineering Von Logikgattern in integrierten schhaltkreisen*. PhD thesis, Humboldt-Universität zu Berlin (2011)
53. V.V. Sergeichik, A.A. Ivaniuk, C.H. Chang, Obfuscation and watermarking of FPGA designs based on constant value generators, in *2014 International Symposium on Integrated Circuits (ISIC)*, Dec 2014, pp. 608–611
54. T. Sugawara, D. Suzuki, R. Fujii, S. Tawa, R. Hori, M. Shiozaki, T. Fujino, *Reversing Stealthy Dopant-Level Circuits* (Springer, Berlin/Heidelberg, 2014), pp. 112–126
55. P. Swierczynski, M. Fyrbiak, P. Koppe, C. Paar, FPGA Trojans through detecting and weakening of cryptographic primitives. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**, 1236–1249 (2015)
56. P. Swierczynski, M. Fyrbiak, C. Paar, C. Huriaux, R. Tessier, Protecting against cryptographic Trojans in FPGAs, in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2015, pp. 151–154
57. M. Tehranipoor, F. Koushanfar, A survey of hardware trojan taxonomy and detection. IEEE Des. Test Comput. **27**, 10–25 (2010)
58. R. Torrance, D. James, The state-of-the-art in semiconductor reverse engineering, in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2011, pp. 333–338
59. J.C. Tsang, J.A. Kash, D.P. Vallett, Picosecond imaging circuit analysis. IBM J. Res. Dev. **44**, 583–603 (2000)
60. K. Vaidyanathan, B.P. Das, E. Sumbul, R. Liu, L. Pileggi, Building trusted ICS using split fabrication, in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, May 2014, pp. 1–6
61. K. Vaidyanathan, B.P. Das, L. Pileggi, Detecting reliability attacks during split fabrication using test-only BEOL stack, in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6
62. S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, S. Borkar, An 80-tile sub-100-w teraflops processor in 65-nm CMOS. IEEE J. Solid State Circuits **43**, 29–41 (2008)
63. A. Vijayakumar, V.C. Patil, D.E. Holcomb, C. Paar, S. Kundu, Physical design obfuscation of hardware: a comprehensive investigation of device and logic-level techniques. IEEE Trans. Inf. Forensics Secur. **12**, 64–77 (2017)
64. H. Wang, *Enhancing Signal and Power Integrity in Three-Dimensional Integrated Circuits*. PhD thesis, Stony Brook University (2016)
65. Y. Wang, P. Chen, J. Hu, J.J.V. Rajendran, The cat and mouse in split manufacturing, in *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6
66. J.B. Wendt, M. Potkonjak, Hardware obfuscation using PUF-based logic, in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2014, pp. 270–271
67. K. Xiao, D. Forte, M.M. Tehranipoor, Efficient and secure split manufacturing via obfuscated built-in self-authentication, in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, May 2015, pp. 14–19
68. Y. Xie, C. Bao, Y. Liu, A. Srivastava, 2.5D/3D integration technologies for circuit obfuscation, in *2016 17th International Workshop on Microprocessor and SOC Test and Verification (MTV)*, Dec 2016, pp. 39–44
69. P.L. Yang, M. Marek-Sadowska, Making split-fabrication more secure, in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–8

Chapter 13
Deterrent Approaches Against Hardware Trojan Insertion

Qihang Shi, Domenic Forte, and Mark M. Tehranipoor

13.1 Introduction

The hardware Trojan threat has become a major security concern for modern integrated circuits (ICs) [1]. A hardware Trojan is a modification of original circuitry with malicious intent during the design or manufacturing processes [2]. Hardware Trojans can be designed to increase malfunctions in the field, lower the reliability of the ICs, leak confidential information to adversaries, or destroy the system under a predefined condition [3]. This makes it possible for hardware Trojans to threaten both commercial and government procurements for both civilian and military applications. Unlike manufacturing defects, Trojans can be designed specifically to evade state-of-the-art IC testing and diagnosis processes and therefore are very difficult to detect [4]. The distributed nature of modern IC supply chain implies that most parties involved in the development of the end product have the opportunity of inserting hardware Trojan (see Fig. 13.1) and therefore might be untrustworthy. A more comprehensive summary of possible Trojan attack scenarios is discussed in [5]. This makes the hardware Trojan threat a concern for both producers and consumers of electronic products: infection by hardware Trojans can both hurt the consumer by making his device defective and out of spec and/or leaking sensitive information and hurt the producer by giving his product a reputation of being so. Each party in the IC supply chain must ensure ability to detect a Trojan that might exist in products it procures and prevent Trojan from infesting the product it sells to its clients.

Q. Shi (✉)
ECE Department, University of Connecticut, Mansfield, Connecticut, USA
e-mail: qihang.shi@engr.uconn.edu

D. Forte • M.M. Tehranipoor
ECE Department, University of Florida, Gainesville, FL, USA
e-mail: dforte@ece.ufl.edu; tehranipoor@ece.ufl.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_13

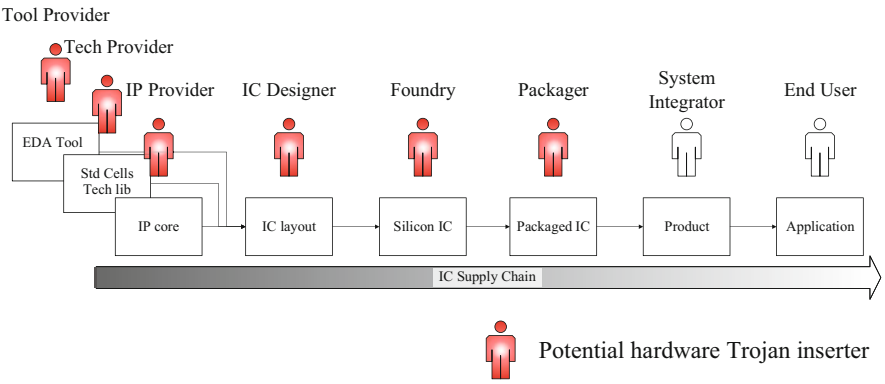


Fig. 13.1 Modern IC supply chain and potential Trojan inserters

To address the hardware Trojan threats and establish trust among parties in the IC supply chain, several countermeasures have already been developed [6]. These techniques generally approach the problem from one of the following angles: Trojan detection, obstruction of Trojan insertion, design-for-trust (DfTr) circuitry, or detection through measurements with test equipment [5, 6]. Anti-Trojan techniques generally provide one or both forms of security: to prove the existence of or lack of Trojan in a procured product or to make it unlikely for the user's product to be infected with Trojans. We term techniques designed to provide the second form of security a *deterrent* approach. The most important distinction between the two categories is that in an investigative technique, the ultimate goal is to determine whether any unspecified product is Trojan-free or not, while in a deterrent technique, the objective is to discourage hardware Trojan insertion into a specific product, which can be achieved either by making it very easy to detect Trojans in it or making it very difficult for certain untrusted parties to insert Trojans. This chapter is focused on *deterrent* approaches against hardware Trojan insertion, including representative technical solutions and their relative weaknesses and strengths, as well as the kinds of Trojan threats that each solution is most and least suitable to address.

Several approaches to implement Trojan-deterrent techniques [5, 6] exist (shown in Fig. 13.2). The most popular approach among them is the *monitoring* approach, which uses on-chip sensors to monitor side-channel information such as power and/or temperature as a result of circuit activities [7–10]. Data collected by the on-chip sensors are then sent through various classification algorithms to identify Trojan-infested and/or Trojan triggered samples. For such classification to succeed, switching activity due to the Trojan trigger and/or payload circuitry must cause it to depart from the normal range of circuit operations. Therefore, it would help if normal functional circuit activity can be kept low so that Trojan activity becomes more significant. This gives rise to the so-called scan cell reordering technique [11]. By optimizing the choice of scan flip-flops for each scan chain, this technique limits circuit activity to specific regions, so that side-channel measurements of

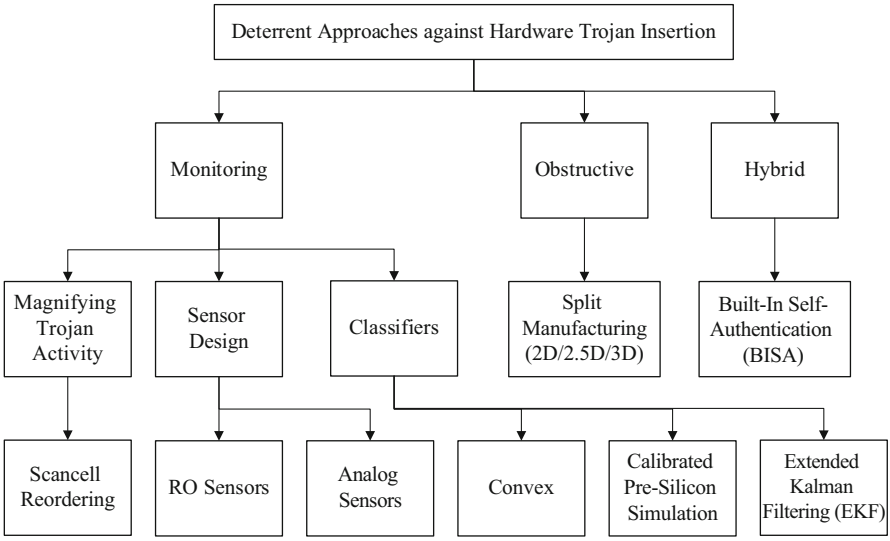


Fig. 13.2 Classification on deterrent approaches to hardware Trojan protection

Trojan-infested ICs can be more easily distinguished from Trojan-free ICs. In contrast to the monitoring approach, other techniques seek to make Trojans insertion physically impossible. We term this approach the *obstructive* technique, as it prevents Trojan insertion as opposed to monitoring Trojan activity. One most widely known obstructive technique is split manufacturing [12]. By preventing the untrusted party – the untrusted foundry – from obtaining complete layout information, split manufacturing also prevents any Trojan insertion targeted at specific circuit function. In addition to these two dichotomies, techniques that combine features from both approaches also exist, an example of which is the built-in self-authentication (BISA) [13–15]. The BISA technique prevents Trojan insertion by occupying all available spaces in the layout with its member gates, and it prevents its member gates from being removed by monitoring them.

Each of these approaches has its own unique merits and satisfies a particular need; however none of them is suitable for all needs or strong against all threats. For example, monitoring approach usually requires a sample known to be good (also known as the "golden IC") to compare suspected samples against, a requirement that is not always practical in real world. Obstructive approach does not rely on this requirement but may be vulnerable to Trojan insertion that does not require information it precludes. A hybrid approach could solve both problems but may be vulnerable to attacks specifically designed to overcome its defenses. As of yet, no approach has emerged substantially more secure than other techniques. Therefore, this chapter is dedicated to discuss relative applicability of each approach by introducing representative techniques in all of them.

The rest of this chapter is organized as follows: Sect. 13.2 presents a number of techniques archetypal of the monitoring approach and discusses their relative strengths. Section 13.3 presents two techniques that do not require any monitoring to deter Trojan insertion with a focus on how their protection may be attacked or overcome by the attacker. Section 13.4 introduces the BISA technique, specific attacks against it, and their possible remedies. Finally, Sect. 13.5 concludes the chapter.

13.2 Monitoring Approach

Detection of Trojan presence through monitoring is most commonly done through side-channel measurements impacted by Trojan activity. This is due to the unpredictability of a Trojan's payload. A functionality-based detection approach may miss a Trojan that is designed to leak information through unconventional methods such as radio antennas, while any circuit consumes power, produces heat, and creates noise. As hardware Trojans on an IC are overwhelmingly made with parasitic circuits, monitoring side-channel signatures is most likely to capture Trojan activity. Another advantage of detection through side-channel measurements is that they can function at any time; this allows implicit feature to double as runtime measurement, which can be useful in deterring very hard to trigger Trojans, such as Trojans that only come into existence when certain conditions are met. Such Trojans can be placed in an IP core in bitstream format intended for reconfigurable devices [16] and would have been very hard to detect without runtime monitoring. Although the requirement of having on-chip measurement sensors seems to add an overhead to the design, such sensors can also serve other performance objectives such as in situ timing slack monitoring [17] and are inserted frequently enough for their existence to be assumed in some cases.

The problem with this side-channel measurement approach lies primarily in difficulty to irrefutably identify a Trojan signature. To begin with, identification of Trojan signature through side-channel measurements requires side-channel signatures of Trojan-free samples (often referred to as "golden" chips). Without knowing what kind of signatures are Trojan-free ("normal"), it is nigh impossible to know what kind of signatures are Trojan infected ("abnormal"). However, the assumption of having access to golden chips is not always valid. For example, if an untrusted foundry chooses to insert a Trojan into chips it fabricates, it is natural for it to insert into all of them. Unless service of a trustworthy foundry can be procured – in which case it becomes doubtful why the untrusted foundry is used – it may not be possible to locate samples without Trojans. Simulations with typical values can help; however due to the presence of parametric variation due to fabrication process (commonly referred to as "process variation"), simulated values may not even match chips that are Trojan-free. Further, side-channel signatures are also affected by short-term conditions, such as temperature, power supply noise and power drop in power grids, switching activity and crosstalk, as well as input test

patterns. Finding a simple separation between Trojan-infected samples and Trojan-free samples is often nontrivial, and it is difficult to avoid overfitting or underfitting. Therefore, side-channel measurement methods are usually evaluated with correct detection rates and false alarm percentages.

In this section, we present two classes of techniques that address the above problem. The first class of methods uses various classification methods to achieve reasonable separation between Trojan-infected and Trojan-free samples, while another method tries to restrict functional circuit switching activity and therefore magnify Trojan switching activity.

13.2.1 Side-Channel Signature Measurement

Two types of sensors have been used for side-channel measurements for the purpose of identifying Trojan-infected chips. Most techniques use some kind of ring oscillator (RO)-based sensors [7, 9], while some other research makes use of current monitors [8] or process control monitors (PCM) [10]. As shown in Fig. 13.3a, RO-based sensors are often inserted at regular geometric positions on design layout. This design simply assumes that all side-channel impacts of Trojan activity, such as noise, power drop, and crosstalk, will be captured by RO sensors in the form of RO frequency changes; classification is then performed based on that measurement alone. In this sense, the use of PCM can be considered a similar choice as PCM does not have one single standardized implementation and is intended to measure parametric drift during fabrication – in other words, measurement of side-channel signatures is not even its intended use. Since most of them also make use of classifiers that does not rely on mathematical modeling of the data to be precise [7, 10], this is unlikely going to constitute a very serious problem to the approach. One advantage of RO and PCM sensors is that they are often already present in the design, negating area overhead due to their insertion.

On the other hand, the current monitor approach is much more precise: As shown in Fig. 13.3b, the design is divided into regions, and each region is measured by one current monitor. This allows a very precise measurement and can yield multidimensional data: for example, transient current can be measured, and based on it delays between current peaks can be gauged and matched to major circuit activities, an analysis less feasible with ROs of limited precision. Another advantage of this approach is that current monitor is an analog circuit; it is likely quieter and leaves less impact on circuit function. This would be very helpful when used in runtime monitoring.

However, all of these benefits are not without drawbacks. The first problem, ironically, is also its impact on functional circuit: its function relies on analog-to-digital conversion, a circuit likely much larger and sophisticated than ring oscillators. This will force its user to make a compromise between area overhead and measurement precision. Also, precision in its measurement means it needs multiple bits in the bus to shift the data out, which may be difficult in a large and congested

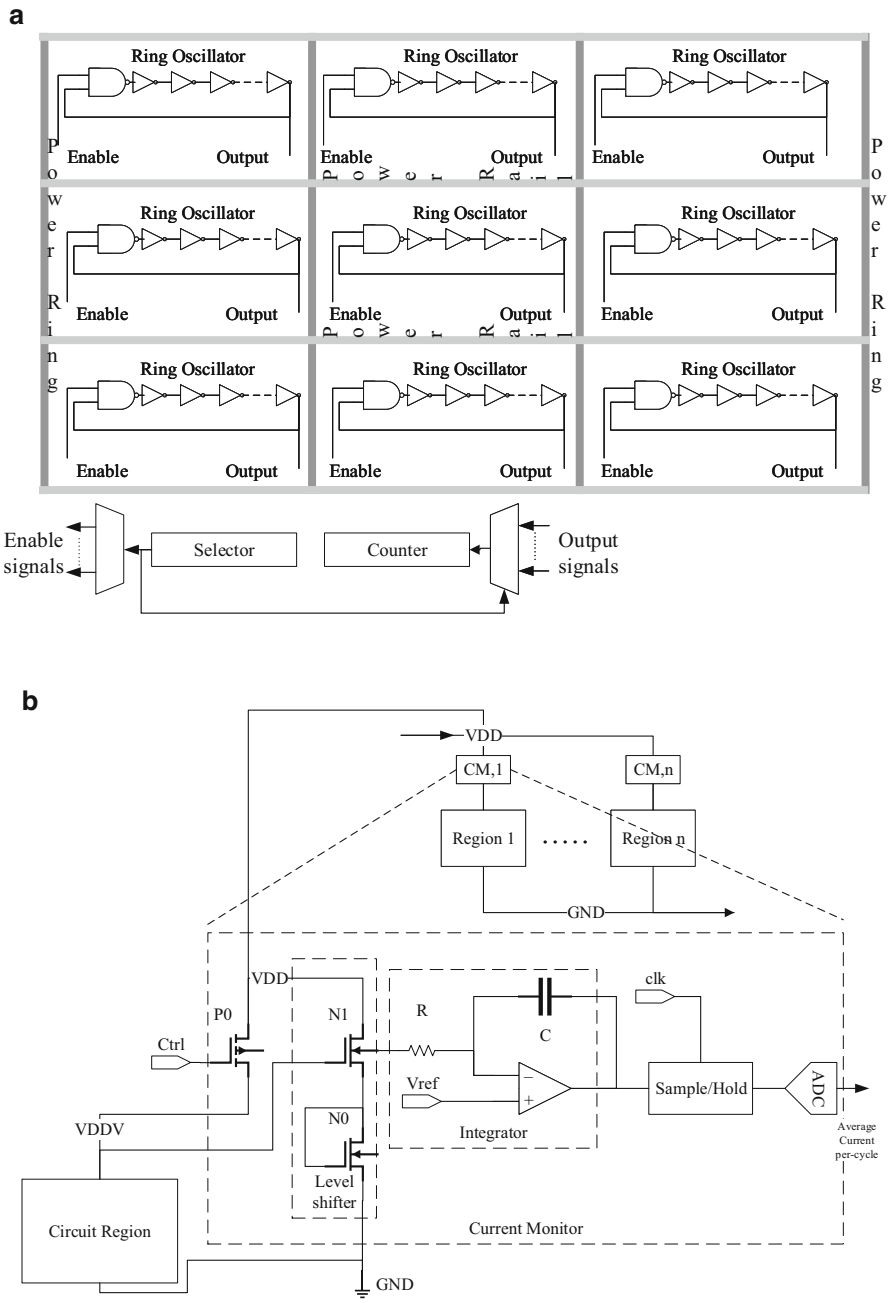


Fig. 13.3 Two typical choice of sensors: (**a**) ring oscillators and (**b**) current monitors

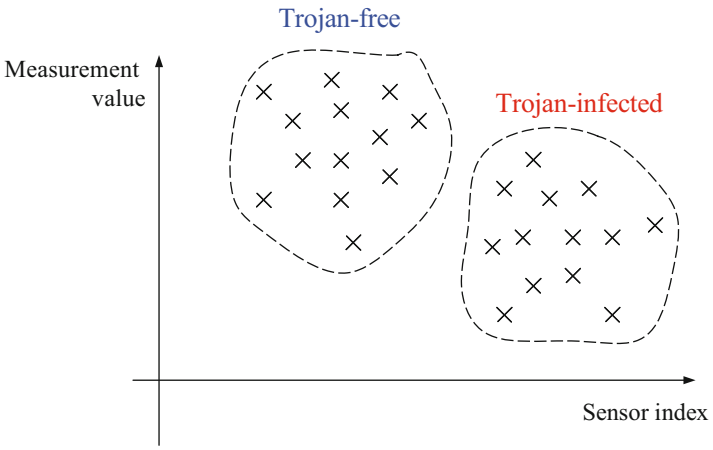


Fig. 13.4 A typical classifier based on training with golden IC samples

design. Being analog in nature also makes the measurement more susceptible to process variation. Lastly, the detailed data it measures will likely become a target for information theft itself – after all, matching current draw to circuit activities is also known as timing attack, a tool in side-channel cryptoanalysis.

13.2.2 Classifiers for Side-Channel Measurements

After side-channel data are measured, classification needs to be done to separate Trojan-infected samples from Trojan-free samples. As mentioned earlier, how this is done is heavily dependent on whether the technique assumes existence of golden ICs. One archetypal design where the technique does make this assumption is presented in [7]. In this technique, measurements from golden ICs are first collected to compute a power signature for each sensor, i.e., a convex hull that envelops all data points (see Fig. 13.4). Then this convex hull is used in authentication of other unknown samples: if a sample falls outside of this convex hull, it is rejected as Trojan infected.

Requirement of golden ICs leaves much to be desired, and techniques have been proposed to remove this requirement. One way to do this is to build a convex hull classifier with the help of pre-silicon simulations. One technique following this approach is presented in [10], where the said pre-silicon simulations are further calibrated with post-silicon measurements using PCMs. Such measurements will update the model with process variation information of each sample. The authors of this technique argues that this calibration step removes the need of golden ICs; in reality, this likely removed inaccuracy due to process variation, while functional circuit activity is still addressed by pre-silicon models, which is hard to calibrate

since modeling this highly nonlinear and pattern-dependent process is difficult. Moreover, calibrating the convex hull classifier is unlikely an efficient process, and having to do it for each sample will be time-consuming. This might result in forcing a trade-off between efficiency and accuracy onto the designer.

Another similarly minded technique [9] also performs calibration, but instead of calibrating a convex hull classifier, it calibrates a parameterized model. This is likely more efficient to calibrating a convex hull classifier, which is a learning process. In addition, instead of using a convex hull classifier, this technique uses sensor measurements to perform an estimation of chip temperature using extended Kalman filtering (EKF) and then computes the discrepancy between estimated temperature and temperature sensor measurements (referred to as "residual"). Trojan activity will cause autocorrelation of residual to diverge from zero and become detected. The advantage of this approach is that functional circuit activity is taken care of by EKF and therefore should not be a concern. However, calculating autocorrelation is a statistical process and requires a large number of samples; observation-correction cycle of EKF further extended this requirement. Further, the temperature model used in this technique can be considered as a low-pass filter of transient power and therefore may miss Trojans with short and/or small power signatures.

13.2.3 Scan Cell Reordering

Scan cell reordering is a technique otherwise used for scan power reduction, improvement of fault coverage, or reducing scan paths [18]. Its application in scan power reduction is apparently also useful in reducing functional circuit side-channel signatures to make Trojan impact more noticeable [11]. To this end, composition of scan chains is reorganized so that each will only consist of scan cells located in the same geometric neighborhood (see Fig. 13.5). Then, when performing the test to activate Trojans and take side-channel measurements, each scan chain is activated individually to restrict functional circuit activity to each region of the design. This is expected to lower activity in functional circuit and dissociate it from that of Trojans. The effectiveness of this approach is then gauged by Trojan-to-circuit switching activity (TCA) and Trojan-to-circuit power consumption (TCP).

One advantage of this technique is that it can be used in conjunction with other sensor designs and classifiers to improve their performance. A problem lies in that a Trojan designer who has access to the layout can choose to scatter his triggering inputs across the layout to reduce likelihood of activation and the difficulty to activate the correct regions increases exponentially as more regions are used.

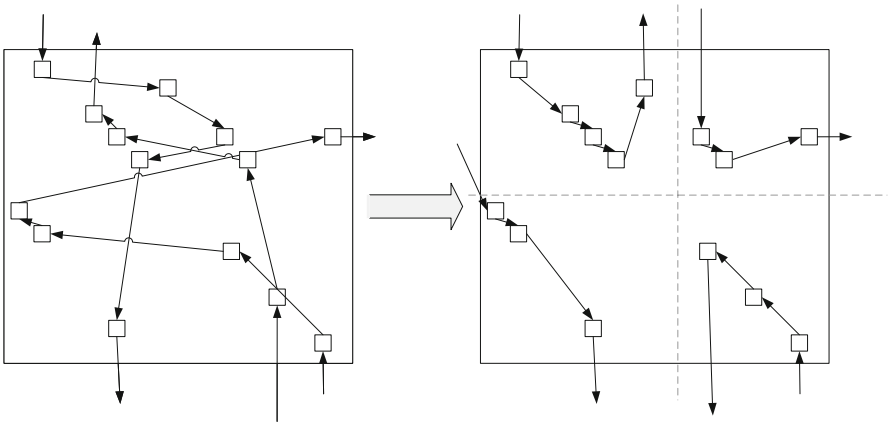


Fig. 13.5 Scan cell reordering: reorder scan chains post-route so that activity in functional circuit can be localized

13.3 Obstructive Approach

Instead of trying to activate the Trojans and capture their activity, the obstructive approach seeks to prevent their insertion altogether. The advantages of this approach are apparent: If Trojans cannot be inserted, then there is no need to activate and/or detect a Trojan. As a corollary, golden chips/models, side-channel profiling, and accurate classification are not needed. Techniques in this approach are often also intended against a wider spectrum of threats in addition to Trojans and malicious modifications, for example, intellectual property (IP) theft, cloning, and overproduction, since malicious intentions are difficult to quantify and model. To achieve these objectives, obstructive techniques often seek to deny leakage of critical information, such as the design or the layout to untrusted parties, and in doing so deny any attack that requires this information, including insertion of Trojans. Hence, techniques in this category rarely require one to insert any additional circuit to the design and therefore greatly reduce design overhead.

Despite the obvious advantage of being efficient and effective against a wider spectrum of threats, the obstructive techniques are not perfect. The first limitation of this approach is that it can only prevent modifications and leakage by a party that *receives* design of the user; it is not possible to obstruct Trojans inserted in a product the user acquires, for example, design blocks in the form of IP cores or electronic design automation (EDA) tools. It is also ineffective against Trojans that do not require the specific information the technique denies leakage of. And finally, for reconfigurable devices, if the Trojan is designed to only manifest during function, no existing technique of the obstructive approach can prevent it.

In this section we present one representative technique of the obstructive approach: split manufacturing, a technique intended to defeat attacks by an untrusted foundry.

13.3.1 Split Manufacturing

Split manufacturing is a technique intended to address the threat to IP security posed by untrusted foundries [12]. Untrusted foundries exist as a result of current economical reality resulting in a globalized IC supply chain. State-of-the-art IC fabrication plants become separated from IP owners and moved to overseas locations where it is hard for IP owners to inspect or influence. Hence, IP owners and sometimes end users (e.g., government and military organizations) are presented with the hard choice between losing access to affordable and state-of-the-art fabrication service and losing security to the IP and/or the end product. Split manufacturing addresses this problem by removing complete knowledge of the IP from the untrusted foundry. In split manufacturing, an IP owner contracts an untrusted foundry manufacture the front-end-of-line (FEOL) part of the IC and then ships it to a trusted foundry to deposit back-end-of-line (BEOL) part onto it (see Fig. 13.6). By denying the untrusted foundry complete layout information, split manufacturing prevents it from stealing IP information or committing attacks that require knowledge of the complete design. This technique deters Trojan insertion by preventing an untrusted foundry from locating important gates and nets of the design and therefore deters the attacker from inserting Trojans at these places.

Split manufacturing can be done with 2D, 2.5D, or 3D fabrication processes. In 2.5D and 3D fabrication, not only BEOL but also parts of the design including transistors are denied from the untrusted foundry, making it even more secure. One

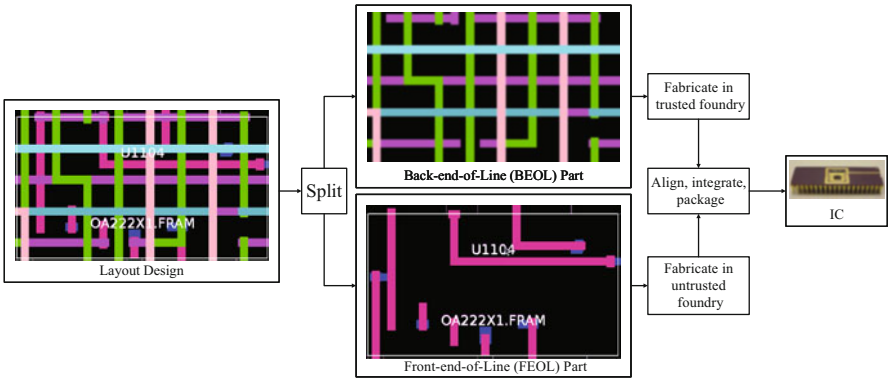


Fig. 13.6 Split manufacturing: separation between Front-End-Of-Line (FEOL) and Back-End-Of-Line (BEOL) part of the IC [19]

method to improve its security is through optimizing separation of the design so that minimum amount of information is left on the FEOL part which the untrusted foundry has access to. This process is called *wire lifting*, as in 2D case it only consists of lifting wires to BEOL part of the layout [20]. A quantified security metric would be required for such an optimization to use as objective, and in [20] the concept of *k*-security was presented to serve as optimization objective function. *k*-security is defined based on number of possible mappings between cells in the layout and cells in the netlist. This is because by definition, a specific attack against split manufacturing will be aiming at recovering the BEOL connections; any attack that does not have this ultimate goal is not an attack specifically against split manufacturing, and any attack that succeeds in this goal will have completely compromised split manufacturing. Short of BEOL connections, the strongest position an untrusted foundry can have is to have access to netlist of the design. In this scenario, the last line of defense would be difficulty for the attacker to map cells in the layout (ones he want to attack) to the cells in the netlist (one which he has knowledge of), a method that makes the most use of the netlist information. *k*-security therefore defines the difficulty of matching one gate to its correct counterpart in the known netlist (i.e., identifying its function) as the number of other gates (*k*) that are indistinguishable from it; and the security of the FEOL layout to be the lowest *k* among all FEOL gates. As an example, consider the full adder design shown in Fig. 13.7. For the purpose of demonstration, let's assume

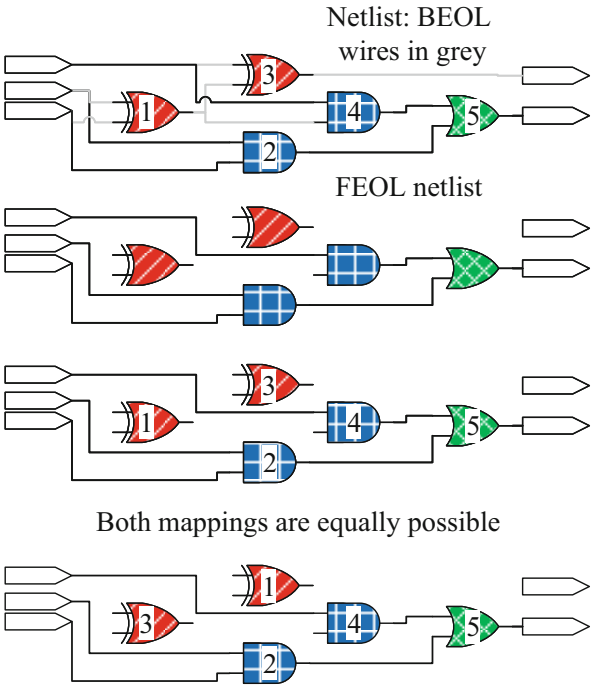


Fig. 13.7 *k*-security: defined as number of identical gates in FEOL

that the wires shaded with gray are routed on BEOL stack of the IC and therefore invisible to the untrusted foundry. Since the untrusted foundry cannot see any wire connected to either XOR gates (painted in red in Fig. 13.7), it is not possible for him to distinguish the two gates. Therefore, the *k*-security of either XOR gate is 2. Since no other gate has any identical FEOL doppelganger, the security of the full adder will be 1. In addition to wire lifting, other approaches to improve split manufacturing security also exist [21–26], mostly by proposing possible attacks and security metrics to split manufacturing. One of the most widely known work in this approach is the *proximity attack*, which simulates an attacker who makes educated guesses on BEOL connections of open input/output pins in FEOL. Unlike wire lifting, techniques of this category focus more on preventing specific exploits than proposing a universal approach to quantitatively improve split manufacturing security.

There are two major weaknesses associated with split manufacturing: its implementation overhead and the kind of attacks it cannot prevent. Although split manufacturing does not require insertion of additional circuits, its requirement to have different parts of the design fabricated by different foundries does indeed impact yield, and wire lifting usually requires non-polynomial processing time to generate a solution. Also, although split manufacturing prevents all attacks that require complete knowledge of the whole layout, not all attacks require that information. One example is the untargeted Trojan insertion [13], the details of which can be found in [27]. Untargeted hardware Trojans do not target specific functions of the original circuitry and therefore cannot commit attacks that require knowledge of such functions. Nevertheless, untargeted hardware Trojans are still capable of degrading the performance and/or reliability of manufactured ICs or trigger a denial of service (DoS) in critical control systems [28]. Further, since split manufacturing is designed to deter attacks committed by an untrusted foundry, it does not prevent attacks by other possible perpetrators, such as untrustworthy EDA tool provider, IP core designer, or SoC designer (e.g., sabotage by employee).

13.4 Hybrid Approach

The hybrid approach refers to techniques that consist of both monitoring and obstruction and therefore does not have the same strengths and weaknesses of either. The built-in self-authentication (BISA) technique is one such example.

13.4.1 Built-In Self-Authentication (BISA)

Built-in self-authentication (BISA) prevents hardware Trojan insertion by exhausting one resource essential to it: *white spaces*. Normally, during the placement step of the back-end design of the circuit, gates in the circuit are placed at optimized locations based on density and routability [29]. This leaves spaces in the layout that

are not filled with standard cells. White spaces have to exist, because meeting power dissipation, crosstalk, and routing objectives demand it. For design purposes other than security against hardware Trojan insertion, these white spaces can be filled with *filler cells* to serve as decoupling capacitors and/or extension of power tracks [30].

However, unsupervised filler cells are not monitored by any logic and therefore prone to malicious removal by Trojan inserters in order to make room for hardware Trojans. If white spaces or decoupling filler cells are replaced by Trojan gates, it would likely incur a mild level of performance loss, nothing more, and certainly not raising suspicion of Trojan presence.

13.4.1.1 BISA Architecture

BISA prevents hardware Trojan insertion by occupying white spaces with testable standard cells instead. Then, all inserted BISA cells are connected into tree-like structures to form a built-in self-test (BIST) circuitry, so that they could be tested to verify no BISA cell has been removed. Removal of its member cells will lead to a BIST failure, so that no attempt to make room for hardware Trojans will evade detection. As shown in Fig. 13.8a, BISA consists of three parts: the BISA circuit under test, the test pattern generator (TPG), and the output response analyzer (ORA). The BISA circuit under test is composed of all BISA cells that have been inserted into unused spaces during layout design. In order to increase its stuck-at fault test coverage, the BISA circuit is divided into a number of smaller combinational logic blocks, called BISA blocks shown in Fig. 13.8a. Each BISA block can be considered as an independent combinational logic block. The TPG generates test vectors that are shared by all BISA blocks. The ORA will process the outputs of all BISA blocks and generate a signature. TPG has been implemented with linear feedback shift register (LFSR), while ORA has been implemented with multiple-input signature register (MISR) in prior work [31]. LFSR and MISR are used in the generation of random vectors and compression of responses into a signature. Other types of TPG and ORA can also be applied [32].

Effective monitoring of the existence of all BISA cells requires very high testability of the BISA blocks, since untestable BISA cells are vulnerable to identification and removal without knowledge of the BISA testing circuits. Therefore, each BISA block needs optimized routing so that they are completely testable. The BISA technique addresses this requirement by organizing each BISA block into combinational logic trees, as shown in Fig. 13.8b. This process is performed by first sorting BISA cells chosen for a BISA block, then creating connections between BISA cells according to their sorted sequence. The method shown in Fig. 13.8b chooses BISA cells by finding all BISA cells in a rectangular bounding box, sorting them randomly, and creating connections between them by iteratively connecting outputs of each BISA cell to unconnected inputs of BISA cells before it in the sorted list.

The main advantage of BISA over other techniques with similar objectives is that it has no golden chip requirement. Since BISA relies on logic testing, process

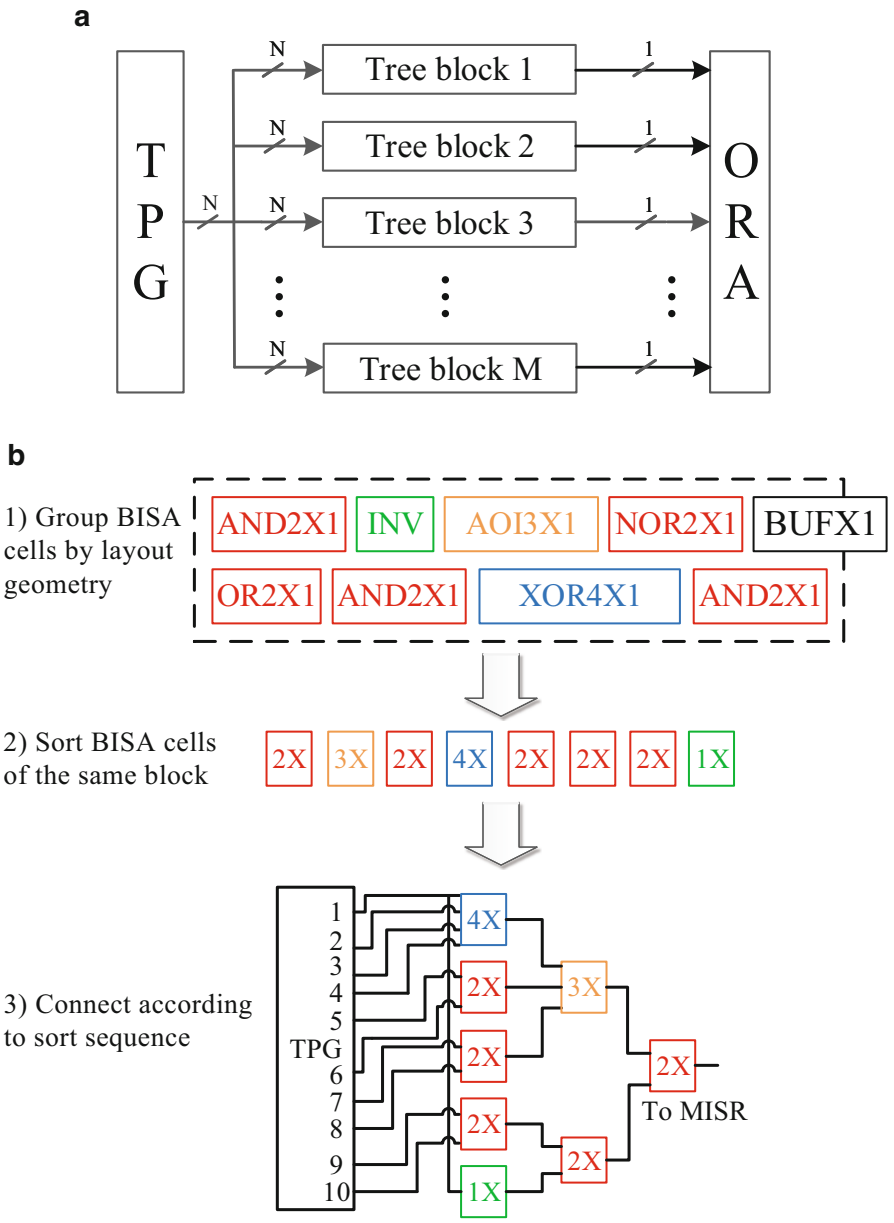


Fig. 13.8 (**a**) Structure of BISA, (**b**) process of constructing a BISA tree block

variation is not a factor either, as compared to Trojan detection techniques based on side-channel analysis. As an additional advantage, impact of BISA on the original design in terms of area and power is also negligible.

13.4.1.2 Specific Attacks and Limitations of BISA

Unfortunately, specific attacks exist for BISA. To attack BISA, an attacker would have to find a way to remove enough of cells to create room for his Trojan insertion, without triggering detection by BISA. Depending on targets and methods used in this removal, several possible alternatives exist to attack BISA:

1. Attack TPG or ORA of BISA;
2. Directly removing cells from BISA or original circuitry, this is known as a *removal attack*;
3. Replace BISA or original circuitry with a smaller functionally equivalent, known as a *redesign attack*; in particular, if large cells are replaced with functional equivalents of smaller area, this is known as a *resizing attack*.

Of the three possible attacks against BISA, attacking TPG or ORA is the least likely to succeed. BISA uses pseudorandom pattern to perform BIST; this makes it very easy to increase pattern count and consequently very difficult for the attacker to make sure all responses of modified TPG and/or ORA will stay the same for arbitrarily many patterns. Similarly, direct removal of BISA cells is unlikely to succeed as they are covered by BISA test coverage during BISA insertion. It is indeed possible to remove cells from original design as long as they do not serve crucial functions; however, design optimization and test coverage will minimize this opportunity for the attacker.

The attack most likely to succeed against BISA is the so-called redesign attack. This attack replaces original circuitry with smaller functionally equivalent circuitry to make room for Trojan insertion. Both BISA and original circuitry can be targeted in this attack. Redesigning the original circuit will result in significant changes of the electrical parameters, such as power and path delay. These can be detected much more easily by delay-based and power-based techniques [33–42]. Therefore, it is more likely for the attack to succeed against BISA cells. It is possible to further secure BISA cells by performing this optimization on BISA design to prevent this particular attack; however, the attacker can also choose to design a custom cell functionally equivalent to several BISA cell in order to make room for Trojan insertion. Prevention of such an attack would require anticipation of all possible custom cell designs that are functionally equivalent to any combination of BISA cells. That is not likely feasible except for very small BISA circuitry. Therefore this attack remains a possible threat to BISA security.

In addition to specific attacks, BISA is also under a few limitations on its performance. For example, due to the existence of resizing attack, all BISA cells have to be of the smallest variant in area among standard cells of the same function, which might make it easier for the attacker to identify them. Further, although it conducts logic test to verify the absence of tampering, it still prevents Trojan insertion by obstruction of its insertion and does not use the logic test to measure any classifiable signature. Therefore, it cannot be used for runtime monitoring and therefore is unlikely effective against Trojans in bitstreams to be downloaded into reconfigurable devices.

Some of the existing limitations and attacks may be eliminated; for example, if split manufacturing is used in conjunction with BISA, it may help to eliminate the problem with redesign attack and limitation of only being able to use minimum-sized cells, because it takes away an adversary's ability to identify any part of the circuitry and hence takes away his ability to reducing their area occupation [13]. For this to achieve best effect, wire lifting is a desirable candidate as it reduces recognizable circuitry size with a mathematical certainty [43].

The BISA technique obstructs Trojan insertion by occupying layout spaces with gates. However, simply occupying available spaces will not be effective, as they can be removed by the adversary. In this case, obstruction necessitates monitoring of inserted gates to ensure that none of them is removed by the adversary to make room for Trojan insertion. In a sense, the BISA technique achieves an obstructive objective by implementing monitoring sensors.

13.5 Conclusion

In this chapter, we provided an introduction and discussion on deterrent approaches against hardware Trojan insertion, defined as techniques designed to deter insertion of hardware Trojans into certain specified products. We show three main approaches that best represent these deterrent approaches: the monitoring approach which predominately works by measuring and classifying side-channel signatures, the obstructive approach which seeks to take away necessary information for the adversary to insert a Trojan, and the hybrid approach that combines features of both approaches. Then, we introduced a few archetypal techniques of each approach and discussed on their relative strengths and weaknesses. From the discussion, it is seen that each approach is advantageous against some category of threats, and a silver bullet technique that can address all threats has yet to be found.

References

1. Trust in integrated circuits (TIC) – proposer information pamphlet (2007). [Online]. Available: www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA503809
2. M. Tehranipoor, C. Wang, *Introduction to Hardware Security and Trust* (Springer, New York, 2012)
3. M. Tehranipoor, F. Koushanfar, A survey of hardware trojan taxonomy and detection. IEEE Des. Test Comput. **27**(1), 10–25 (2010)
4. What's new about hardware Trojans. [Online]. Available: https://www.trust-hub.org
5. K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, M. Tehranipoor, Hardware Trojans: lessons learned after one decade of research. ACM Trans. Des. Autom. Electron. Syst. (TODAES) **22**(1), 6 (2016)
6. S. Bhunia, M.S. Hsiao, M. Banga, S. Narasimhan, Hardware Trojan attacks: threat analysis and countermeasures. Proc. IEEE **102**(8), 1229–1247 (2014)

7. X. Zhang, M. Tehranipoor, Ron: an on-chip ring oscillator network for hardware trojan detection, in *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, 2011), pp. 1–6
8. S. Narasimhan, W. Yueh, X. Wang, S. Mukhopadhyay, S. Bhunia, Improving ic security against trojan attacks through integration of security monitors. IEEE Des. Test Comput. **29**(5), 37–46 (2012)
9. C. Bao, D. Forte, A. Srivastava, Temperature tracking: toward robust run-time detection of hardware Trojans. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**(10), 1577–1585 (2015)
10. Y. Liu, K. Huang, Y. Makris, Hardware Trojan detection through golden chip-free statistical side-channel fingerprinting, in *Proceedings of the 51st Annual Design Automation Conference, DAC'14* (ACM, New York, 2014), pp. 155:1–155:6. [Online]. Available: http://doi.acm.org/10.1145/2593069.2593147
11. H. Salmani, M. Tehranipoor, Layout-aware switching activity localization to enhance hardware trojan detection. IEEE Trans. Inf. Forensics Secur. **7**(1), 76–87 (2012)
12. IARPA Trusted Integrated Circuits (TIC) program announcement. http://www.fbo.gov
13. K. Xiao, D. Forte, M.M. Tehranipoor, Efficient and secure split manufacturing via obfuscated built-in self-authentication, in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (IEEE, 2015), pp. 14–19
14. K. Xiao, D. Forte, M. Tehranipoor, A novel built-in self-authentication technique to prevent inserting hardware trojans. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **33**(12), 1778–1791 (2014)
15. K. Xiao, M. Tehranipoor, Bisa: built-in self-authentication for preventing hardware trojan insertion, in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (IEEE, 2013), pp. 45–50
16. D.B. Roy, S. Bhasin, S. Guilley, J.-L. Danger, D. Mukhopadhyay, X.T. Ngo, Z. Najm, Reconfigurable lut: a double edged sword for security-critical applications, in *International Conference on Security, Privacy, and Applied Cryptography Engineering* (Springer, 2015), pp. 248–268
17. M. Sadi, L. Winemberg, M. Tehranipoor, A robust digital sensor ip and sensor insertion flow for in-situ path timing slack monitoring in socs, in *2015 IEEE 33rd VLSI Test Symposium (VTS)* (Apr 2015), pp. 1–6
18. Y. Bonhomme, P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, Efficient scan chain design for power minimization during scan testing under routing constraint, in *ITC* (Citeseer, 2003), pp. 488–493
19. J. Rajendran, O. Sinanoglu, R. Karri, Regaining trust in VLSI design: design-for-trust techniques. Proc. IEEE **102**(8), 1266–1282 (2014)
20. F. Imeson, A. Emtenan, S. Garg, M. Tripunitara, Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation, in *Presented as Part of the 22nd USENIX Security Symposium (USENIX Security 13)* (2013), pp. 495–510
21. C.T.O. Otero, J. Tse, R. Karmazin, B. Hill, R. Manohar, Automatic obfuscated cell layout for trusted split-foundry design, in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (IEEE, 2015), pp. 56–61
22. Y. Xie, C. Bao, A. Srivastava, Security-aware design flow for 2.5d IC technology, in *Proceedings of the 5th International Workshop on Trustworthy Embedded Devices, TrustED'15* (ACM, New York, 2015), pp. 31–38. [Online]. Available: http://doi.acm.org/10.1145/2808414.2808420
23. J.J. Rajendran, O. Sinanoglu, R. Karri, Is split manufacturing secure? in *Proceedings of the Conference on Design, Automation and Test in Europe* (EDA Consortium, 2013), pp. 1259–1264
24. J. Magaña, D. Shi, A. Davoodi, Are proximity attacks a threat to the security of split manufacturing of integrated circuits? in *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD'16* (ACM, New York, 2016), pp. 90:1–90:7. [Online]. Available: http://doi.acm.org/10.1145/2966986.2967006

25. Y. Wang, P. Chen, J. Hu, J.J. Rajendran, The cat and mouse in split manufacturing, in *Proceedings of the 53rd Annual Design Automation Conference* (ACM, 2016), p. 165
26. M. Jagasivamani, P. Gadfort, M. Sika, M. Bajura, M. Fritze, Split-fabrication obfuscation: metrics and techniques, in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (IEEE, 2014), pp. 7–12
27. Q. Shi, K. Xiao, D. Forte, M.M. Tehranipoor, Obfuscated built-in self-authentication, in *Hardware Protection Through Obfuscation* (Springer, Cham, 2017), ch. 11, pp. 263–289
28. R.J. Turk et al., *Cyber Incidents Involving Control Systems* (Idaho National Engineering and Environmental Laboratory, Idaho Falls, 2005)
29. X. Yang, B.-K. Choi, M. Sarrafzadeh, Routability-driven white space allocation for fixed-die standard-cell placement. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **22**(4), 410–419 (2003)
30. S. Charlebois, P. Dunn, G. Rohrbaugh, Method of optimizing customizable filler cells in an integrated circuit physical design process, 28 Oct 2008, US Patent 7,444,609. [Online]. Available: https://www.google.com/patents/US7444609
31. K. Xiao, D. Forte, M. Tehranipoor, A novel built-in self-authentication technique to prevent inserting hardware Trojans. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **33**(12), 1778–1791 (2014)
32. M. Bushnell, V.D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, vol. 17. (Springer, New York, 2000)
33. X. Wang, M. Tehranipoor, J. Plusquellic, Detecting malicious inclusions in secure hardware: challenges and solutions, in *IEEE International Workshop on Hardware-Oriented Security and Trust, HOST'08* (IEEE, 2008), pp. 15–19
34. D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, B. Sunar, Trojan detection using ic fingerprinting, in *2007 IEEE Symposium on Security and Privacy (SP'07)* (IEEE, 2007), pp. 296–310
35. S. Narasimhan, X. Wang, D. Du, R.S. Chakraborty, S. Bhunia, Tesr: a robust temporal self-referencing approach for hardware Trojan detection, in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (IEEE, 2011), pp. 71–74
36. J. Zhang, H. Yu, Q. Xu, Htoutlier: hardware Trojan detection with side-channel signature outlier identification, in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (IEEE, 2012), pp. 55–58
37. S. Wei, S. Meguerdichian, M. Potkonjak, Gate-level characterization: foundations and hardware security applications, in *Proceedings of the 47th Design Automation Conference* (ACM, 2010), pp. 222–227
38. J. Aarestad, D. Acharyya, R. Rad, J. Plusquellic, Detecting Trojans through leakage current analysis using multiple supply pad s. IEEE Trans. Inf. Forensics Secur. **5**(4), 893–904 (2010)
39. Y. Alkabani, F. Koushanfar, Consistency-based characterization for IC Trojan detection, in *Proceedings of the 2009 International Conference on Computer-Aided Design* (ACM, 2009), pp. 123–127
40. Y. Jin, Y. Makris, Hardware Trojan detection using path delay fingerprint, in *IEEE International Workshop on Hardware-Oriented Security and Trust, HOST'08* (IEEE, 2008), pp. 51–57
41. K. Xiao, X. Zhang, M. Tehranipoor, A clock sweeping technique for detecting hardware Trojans impacting circuits delay. IEEE Des. Test **30**(2), 26–34 (2013)
42. B. Cha, S.K. Gupta, Trojan detection via delay measurements: a new approach to select paths and vectors to maximize effectiveness and minimize cost, in *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, 2013), pp. 1265–1270
43. Q. Shi, K. Xiao, D. Forte, M. Tehranipoor, Securing split manufactured ICs with wire lifting obfuscated built-in self-authentication, in *2017 ACM 27th Great Lakes Symposium on VLSI (GLSVLSI)*, May 2017

Chapter 14
Hardware Trojan Attacks in FPGA and Protection Approaches

Vinayaka Jyothi and Jeyavijayan (JV) Rajendran

14.1 Introduction

Field-programmable gate arrays (FPGAs) are integrated circuits (ICs) containing programmable logic components that can be reconfigured by an end-user post manufacturing. In the recent years, the usage of FPGAs has drastically increased; the FPGA market share is estimated to reach $9.9 billion by 2020 [36]. FPGAs are used in a wide range of applications such as application-specific integrated circuit (ASIC) prototyping, communication devices, medical instruments, high-performance computing systems, aerospace, and defense systems. The growing demand for power-efficient and high-performance ICs has created a surge in usage of FPGAs in the recent years. FPGAs are also available as cloud services [3], where one can create and run custom hardware designs on a remote FPGA in a server farm. Hence, exploring security issues associated with FPGA designs is critical.

Simultaneously, globalization of IC design flow has reduced design complexity and fabrication cost, but it has introduced several security vulnerabilities [13]. A rogue element anywhere in the IC supply chain can perform the following attacks: reverse engineering (RE), hardware Trojans, counterfeiting (specifically, recycled ICs), and IP piracy [17, 35]. These attacks cost the semiconductor industry billions of dollars annually [30, 37], undermine national security [1, 14], and put critical infrastructure into danger [15]. Though part of the problem is that designers have no control over their design in this distributed supply chain, a more important issue is that current IC design tools do not consider security as a design metric.

V. Jyothi (✉)
New York University, New York, NY, USA
e-mail: vinayaka.jyothi@nyu.edu

J. (JV). Rajendran
The University of Texas at Dallas, Dallas, TX, USA
e-mail: jv.ee@utdallas.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_14

This chapter explores insertion of hardware Trojans into genuine designs targeted for FPGAs. Such compromised designs may result in subpar performance, leakage of confidential information, and unauthorized and pernicious operations by an attacker. The use of compromised designs in critical infrastructures such as smart grids, nuclear power plants, medical prosthetic devices, and military equipment can be catastrophic. To explain the different classes of Trojans, this chapter uses Xilinx FPGA design flow. However, the same methodology can be extended to any FPGA and CAD tool vendors.

This chapter is organized as follows: We first present the threat model and a taxonomy of FPGA Trojans in Sect. 14.2. Next, we focus on three broad categories of FPGA Trojans: Trojans in FPGA fabric (see Sect. 14.3), Trojans in FPGA tool chain (see Sect. 14.4), and Trojans in FPGA bitstreams (see Sect. 14.5). Section 14.6 discusses the countermeasures that specifically target Trojans in FPGA bitstreams. Finally, Sect. 14.7 concludes the chapter.

14.2 Threat Models and Taxonomy

14.2.1 FPGA Design Flow

Figure 14.1 shows the FPGA design flow. An FPGA designer designs the FPGA fabric. Fabless FPGA design houses send the layout of the FPGA fabric to a foundry

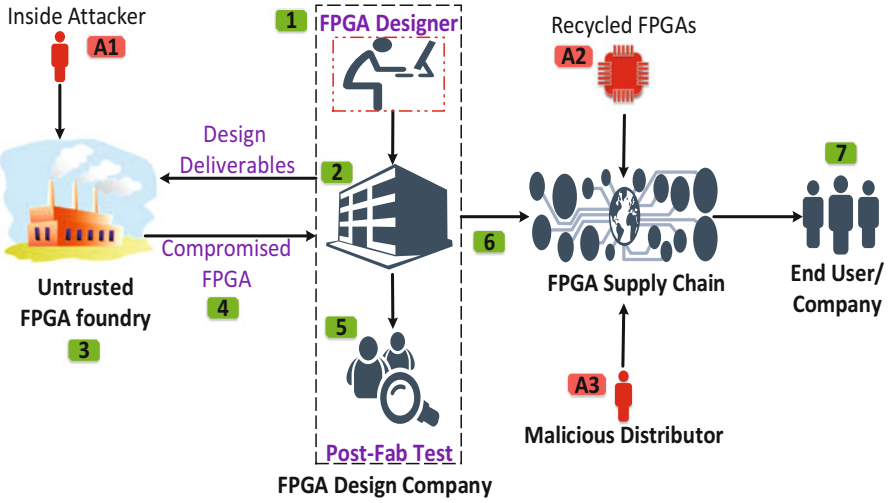


Fig. 14.1 FPGA threat model: The attacker can insert hardware Trojans at the untrusted foundry (A1). A malicious distributor can reduce the reliability of an FPGA in the supply chain (A3), and even recycled FPGAs can be inserted into the FPGA supply chain (A2). Design Trojans can also enter through FPGA CAD tool flow

for manufacturing. Many of these foundries are located typically offshore and are untrusted. Post-fabrication, the FPGAs are tested for defects and faults. FPGAs are sold on the market. The end-user implements the target design on the FPGA. Converting a design described in a modeling language (VHDL or Verilog) into an FPGA-specific programming file involves multiple steps, as explained below:

1. Synthesis involves the conversion of HDL into a logical netlist (similar to logic diagram or circuit).
2. Implementation consists of translate and map processes,[1] where the logical netlist gets converted and mapped to target device's physical primitives.
3. Place and route (PAR) takes a mapped native circuit description (NCD) file, places and routes the design, and produces an NCD file to be used by the programmable file generator.
4. In bitfile generation, the routed NCD is used to create a bitfile that can be programmed onto an FPGA.

14.2.2 Threat Model

In the quest to reduce the development cost of hardware/system, the silicon industry has inadvertently created a complex and extremely vulnerable supply chain shown in Fig. 14.1. An attacker can be present anywhere in the supply chain. The threat model, shown in Fig. 14.1, involves:

Overproduction: An untrusted foundry that has access to the FPGA layout mask fabricates more number of FPGAs than requested or authorized by the design company. It can insert these FPGAs into the supply chain without the knowledge of FPGA design company. These FPGAs may not be properly tested and can introduce reliability issues. This results in either loss of revenue or reputation for the design company.

Recycling and remarking: FPGAs can be extracted from electronic waste, used FPGAs can be removed, and their package can be repainted and/or remarked. The die can also be removed from the packaging, repackaged, and remarked. These FPGAs are then reinserted into the supply chain as genuine and new FPGAs. These FPGAs can be highly unreliable, are prone to defects, and typically lead to subpar performance.

Cloning and piracy: It is an unauthorized reproduction of an FPGA by reverse engineering without the legal intellectual property (IP) rights to manufacture the FPGA. These FPGAs can also have malicious modifications.

Apart from these threats, FPGAs are also susceptible to insertion of Trojans, which is described in detail below.

[1] Translate and map processes are the terms used by Xilinx, an FPGA vendor. These processes may use different names/terms.

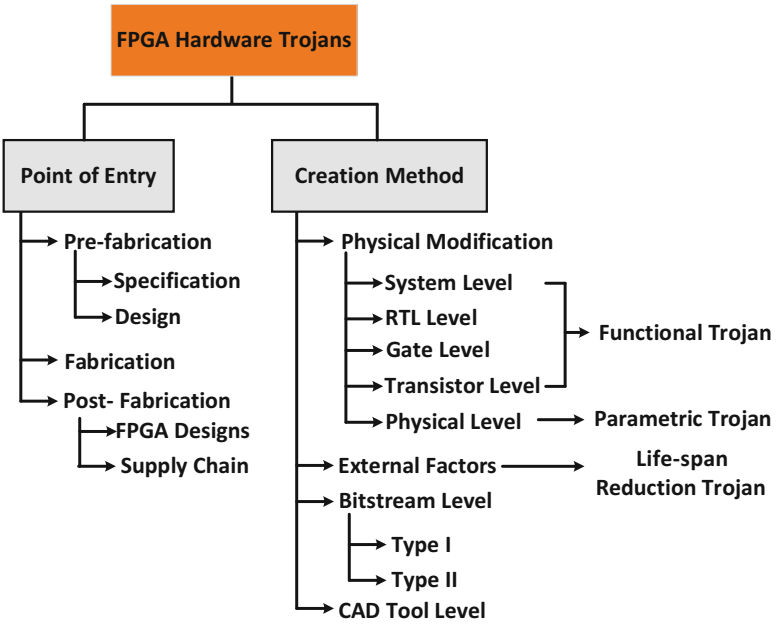


Fig. 14.2 FPGA hardware Trojan taxonomy based on two primary attributes

14.2.3 Taxonomy

Malicious changes can be made at any phase of the FPGA design such as design, fabrication, packaging, and in the supply chain as shown in Fig. 14.1. A taxonomy based on hardware Trojans' physical, activation, and functional characteristics have already been proposed [19, 34]. We classify Trojans based on the method of creation, activation, and point of entry into the FPGA fabric as shown in Fig. 14.2. The definitions of most of the FPGA Trojans are similar to the IC Trojan taxonomy in [19, 34]. In this chapter, we will focus on the FPGA-specific attributes of the taxonomy. Interested readers can refer to the comprehensive taxonomy presented in [19, 34].

14.2.4 Point of Entry

Based on the point of entry of Trojans in FPGA, they can be classified as:

- *Prefabrication:* It is the phase where the specification of systems such as functionality, size, power, delay, etc., is finalized. Trojan insertion in this step will result in alteration of design or constraints. For example, it could alter the timing of circuit or increase switching frequency of the circuit. A rogue employee

can insert a malicious circuit, e.g., a backdoor, to take control of the chip at a later point in time when the FPGAs are deployed in the field. These manifest as FPGA fabric Trojans.
- *Fabrication:* Here, a set of masks are designed to fabricate the digital circuit on a silicon wafer. Trojans can be added by a malicious attacker inside an untrusted foundry. These Trojans can be either functional or parametric. These are called as FPGA fabric Trojans.
- *Post-fabrication:* In this phase, RTL/HDL designs are used to program an FPGA to achieve desired functionality. Trojans can be either inserted in RTL/HDL designs by a rogue employee or can also enter RTL/HDL designs from IPs from third-party IP providers. These are FPGA design Trojans. Additionally, even the reliability of an FPGA can be reduced by such type of Trojans.

14.2.5 Creation Method

Based on the creation method, Trojans can be classified into:

- *Functional Trojan:* They are created by modifying the FPGA fabric. This includes addition/deletion of gates/transistors, modifying the RTL or layout without affecting the primary functionality of the FPGA fabric. It can enter during prefabrication phase by a rogue employee in the FPGA design company or during fabrication phase by a malicious insider at an untrusted foundry.
- *Parametric Trojan:* They are created by modifying physical device parameters, such as thinning of wires, gate channel length variation, dopant level variation [5], transistor size variation, etc. It is always on and primarily created to reduce the reliability and lifespan of an FPGA.
- *Life-span reduction Trojan (LRT):* It is the only class of Trojans that are not inserted in the hardware during or before fabrication. It is created by subjecting the FPGA with external factors, such as extreme temperatures, focused ion beams [9], etc. LRT accelerates aging of complete or part of FPGA fabric. It is typically created by a malicious distributor in the FPGA supply chain to reduce the reliability and, hence, reduce the lifespan of FPGAs.
- *Bitstream Trojan:* It is inserted by modifying the FPGA bitfile itself. Bitstreams can be reverse engineered to identify the areas of FPGA occupied by the programmed logic, and malicious circuits can be inserted into it. If the malicious circuit does not disturb the original circuit, it is called Type-I bitstream Trojan. Type-II Trojans typically modify the original circuitry with respect to CLBs or other FPGA resources to perform malicious operations.
- *CAD tool Trojan:* They are FPGA design Trojans that exploit the CAD tool flow to insert the Trojans at various intermediate netlist formats. These Trojans can be inserted in a synthesized netlist and even in mapped or placed and routed netlists. Due to the lack of resources to understand the intermediate and typically proprietary formats, these Trojans can easily evade detection.

14.3 Trojans in FPGA Fabric

FPGA fabric Trojans are inserted into the FPGA silicon fabric. They can be inserted either during fabrication by a untrusted foundry or during the design phase of FPGA by a rogue employee in the FPGA design company. Functional fabric Trojans are characterized by addition/deletion of gates by the attacker to carry out malicious activities, whereas parametric fabric Trojans are created by changing device parameters/specification such as thinning of wires and weakening of transistors or flip-flops to reduce the reliability of the FPGA [18, 25]. In this section, we describe three Trojans that can be inserted into the FPGA fabric: Trojans that increase delay, create voltage fluctuations, and reduce lifetime.

14.3.1 Trojans That Increase Delay

The delay-based fabric Trojan is created by modifying interconnect connecting lookup tables (LUTs) across two configurable logic blocks (CLBs). The delay-based fabric Trojans correspond to change or perturbation in the physical layout of FPGA due to the addition of malicious elements as shown in Fig. 14.3. The assumption is that the silicon fabric of the FGPA is dense and highly utilized. An attacker needs to alter the FPGA silicon in order to add a Trojan. For example, when a Trojan is inserted in a CLB or routing switch matrix (RSM), it will perturb the physical layout of original fabric, thereby increasing the delay.

Example Figure 14.4 is a screenshot of an FPGA editor tool. The routes colored in red are before and after modification, respectively. This emulates a Trojan inserted in an interconnect that affects the net delay which in turn impacts total delay of the path connecting the two I/O pins of CLBs. Consider this path is configured to be a path of a ring oscillator (RO) programmed using the LUTs in the corresponding CLBs. All the routes, LUTs, and pins remain the same, for both ROs without and

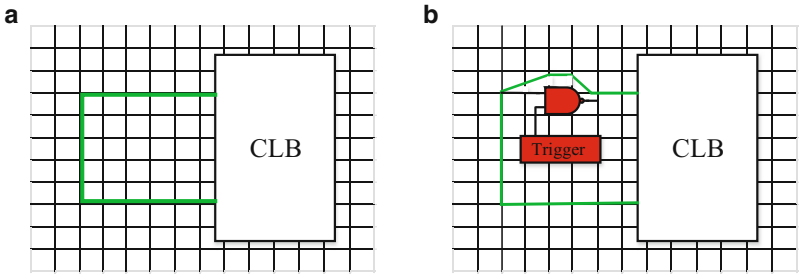


Fig. 14.3 FPGA fabric Trojan disturbing the original layout. (**a**) Trojan-free FPGA CLB with the RSM. (**b**) A Trojan inserted in the RSM

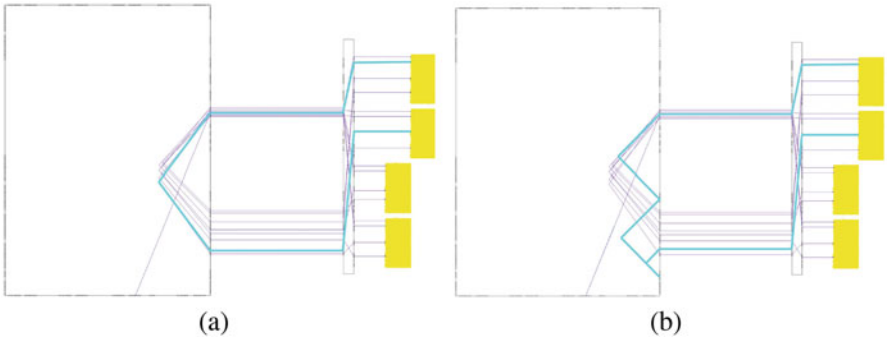


Fig. 14.4 Trojans that increase delay. (**a**) RSM without Trojan and (**b**) the route in RSM changes due to the presence of a Trojan

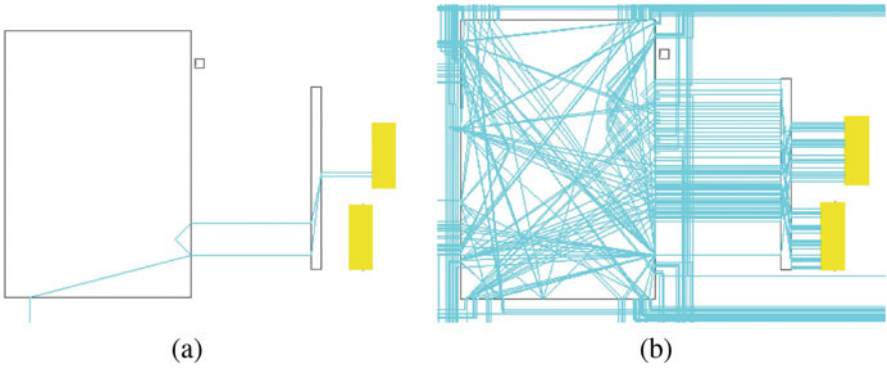


Fig. 14.5 Trojans that induce voltage fluctuations. (**a**) CLB without Trojan and (**b**) CLB with a Trojan. The Trojan causes the increased switching activity near the path/route of interest. This, in turn, manifests as significant voltage drop near the CLB

with Trojan, except the Trojan-perturbed route. The delays are 0.140 and 0.419 ns, before and after the interconnect modification, respectively. The delay difference caused by the Trojan is 0.279 ns.

14.3.2 Trojans That Induce Voltage Fluctuations

This set of Trojans is implemented by adding simultaneous switching signals that utilize dense interconnect resources around a CLB. This corresponds to the addition of malicious elements without disturbing the genuine layout of FPGA fabric. This switching signal is connected to unused wires and programmable interconnect points (PIPs) in the tile where the target CLB is configured as shown in Fig. 14.5.

This Trojan increases switching activity that will increase dynamic power and therefore impacts the oscillation frequency. In [39], it was observed that voltage drop due to the Trojan switching activity had impacted the RO frequency.

14.3.3 Life-Span Reduction Trojan (LRT)

Life-span reduction Trojan (LRT) can be induced into an FPGA by artificially creating conditions that accelerate aging of FPGA fabric. Key contributors for an FPGA aging (or any IC) among several physical factors are negative-bias temperature instability (NBTI) and hot carrier injection (HCI) [6]. Both the factors lead to a shift of threshold voltage of the affected transistors, which manifest as increase in switching and path delays. This will subsequently lead to timing violations and wears out an FPGA faster.

NBTI. Transistor parameters can be evolved over a large range of stress conditions. Bias temperature instabilities are observed when a transistor is stressed at relatively high temperatures. Electrical parameters of transistors such as the threshold voltage (V_{th}) can be shifted by the creation of interface traps and trapping of holes into the oxide [16]. This in turn reduces the mobility and degrades device parameters, and the transistor yields a lower level of performance. This inherently ages the devices faster. NBTI primarily affects PMOS devices. Supply voltage and temperature exponentially increase the NBTI effect. Thus, NBTI degradation can be induced by subjecting an FPGA to higher operating temperatures and supply voltage.

HCI primarily affects NMOS; where accelerated electrons in the channel can collide with gate oxide interface, electron-hole pairs are created. V_{th} increases due to free electrons trapped in the gate oxide layer. The degradation associated with HCI attributes to physical breakdown and characteristic distortion of the transistor. Operating conditions and switching frequency are the primary contributors to the rate of HCI degradation [31]. The V_{th} shift has a sublinear dependence on frequency, runtime, and activity factor (ratio of transitions and runtime) [7].

14.3.3.1 Accelerating FPGA Aging

Because of NBTI and HCI effects, the aging of transistors can be increased significantly by increasing temperature, supply voltage, and switching activity. A malicious distributor can subject an FPGA to extreme temperatures and supply voltage and even program the FPGA to run rogue design with a high switching activity under stressful conditions. To negate the aging effects, manufacturers specify safe operating margins. Running an FPGA outside the safe operating margins would inherently degrade the performance and, hence, reduce the reliability of the chip.

A simple heat gun and an external power supply can be used to operate an FPGA under such extreme conditions. An attacker with access to sophisticated equipment can use focused ion beam irradiation [9, 32] and even create selective-LRT, where only a few specific tiles or CLBs in an FPGA wear out. Using a focused heat gun can also be used to create selective-LRT. However, as the FPGA packaging is designed to spread the heat evenly throughout the FPGA, it is difficult to achieve such precision when compared to focused ion beams.

14.3.3.2 An Example LRT Trojan

An LRT Trojan is created by subjecting the FPGA to extreme temperature of 180°C, while the fabric is configured with a design that increases the switching activity inside the FPGA. The setup uses a soldering heat gun placed on the FPGA with a small air gap as shown in Fig. 14.6. FPGA sensors register an internal temperature of 170°C (50°C above the manufacturer-specified maximum temperature of 120°C).

16 ROs are used to measure the performance degradation on Day 3, Day 6, and Day 7. To accurately measure the impact of aging effects due to NBTI and HCI, the heat gun is switched off, and the FPGA is allowed to cool down for 24 h, as studies show that the transistors recover from the aging effects to a certain extent [4]. Figure 14.7 shows the frequencies of the 16 ROs from Day 0 to Day 7. Figure 14.8 shows that the FPGA fabric at the RO locations has degraded by 10% on Day 7.

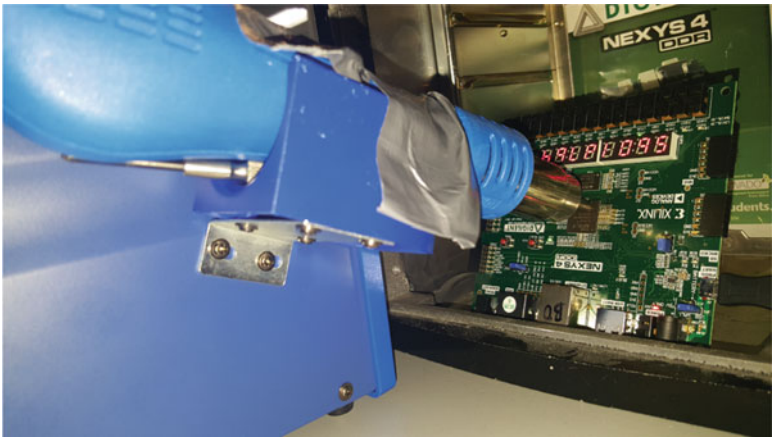


Fig. 14.6 Experimental setup to create the LRT Trojan. The FPGA board, Xilinx Nexys 4, is subjected to an external temperature of 200 °C for 7 days using a soldering heat gun. The internal supply voltage is around 0.905 V. This setup avoids damaging any interfaces and peripherals on FPGA board

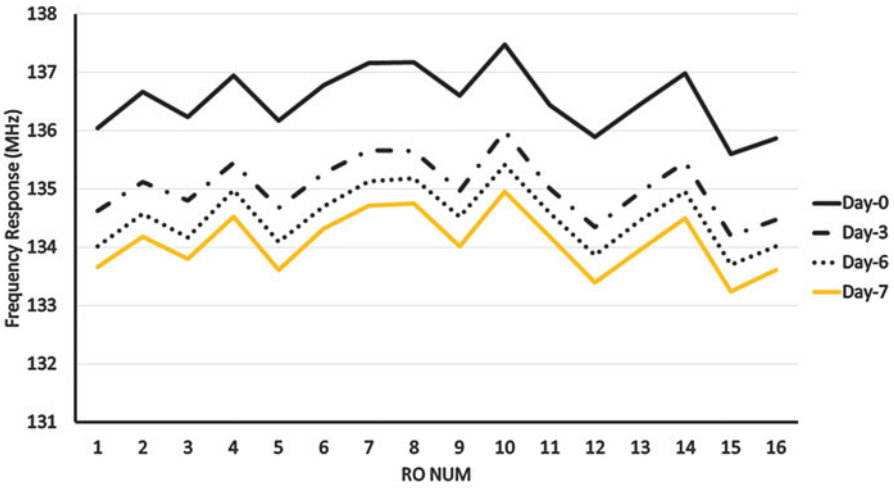


Fig. 14.7 Change in the frequency response of FPGA tiles due to the LRT Trojan. As the FPGA tiles age, the performance degrades due to the LRT Trojan

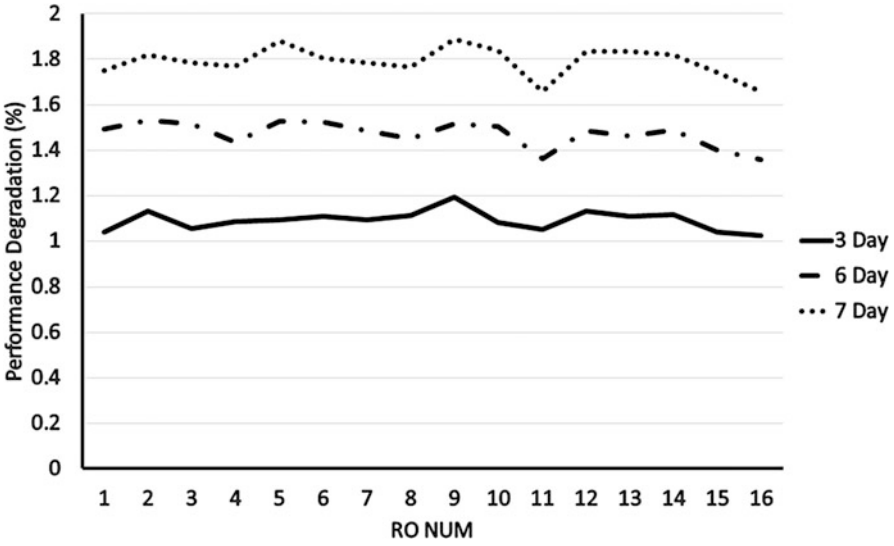


Fig. 14.8 The delay of the CLB increases by 1.9% in 7 days due to the LRT Trojan

14.4 Trojans in FPGA Design

14.4.1 Trojan Insertion in FPGA Designs

The goal of the attack tool is to decide where to place the Trojan for a given design. The placement of Trojan can be achieved with or without disturbing the original design mapping and routing. The latter requires considerable effort and access to multiple files from the FPGA design cycle.

To insert hardware Trojans in FPGA designs, an attacker may need to have knowledge of internal wires or logic and preferably where the design is physically placed on the FPGA. If the Trojan is conditionally activated based on the input or internal states, the attacker needs to tap into the required wires of the design and connect the Trojan activator circuit. The Trojan payload can be connected to the target elements by disconnecting the wires connecting to these elements and reconnecting with the output of the payload circuit. The original payload and Trojan payload can be connected using multiplexers, with the select line controlled by the activator circuit.

After the logic synthesis process, FPGA CAD tools typically rename and merge (after logic optimization) the internal wires connecting logic elements. An attacker needs to track the name changes in the design, to connect them with the Trojans. This can be achieved by converting the synthesized binary netlist (called NGC by Xilinx) to a readable Electronic Design Interchange Format (EDIF) file and Xilinx Design Language (XDL) file. Figure 14.9a shows the HDL code, and Fig. 14.9b shows the corresponding XDL file obtained from routed netlist (NCD) after PAR,

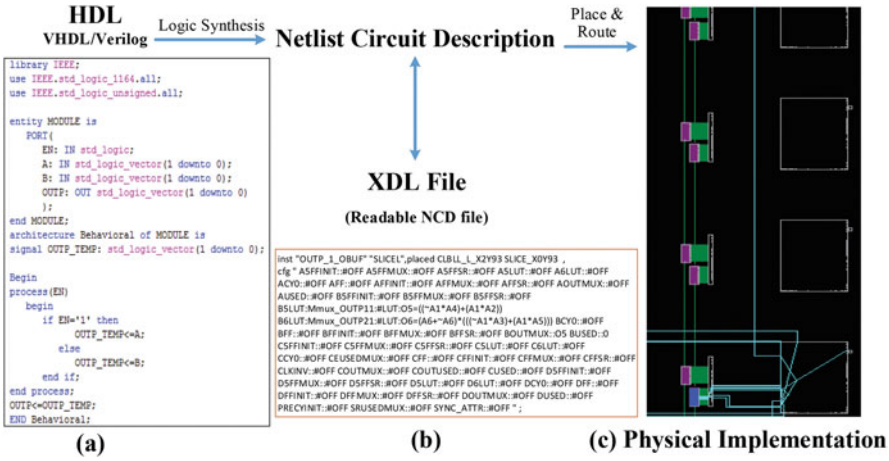


Fig. 14.9 HDL to FPGA physical implementation. (**a**) Description of the design in HDL. (**b**) The configuration of a routed CLB described in XDL format. Only LUT B is used in this CLB. (**c**) The physical implementation of the design

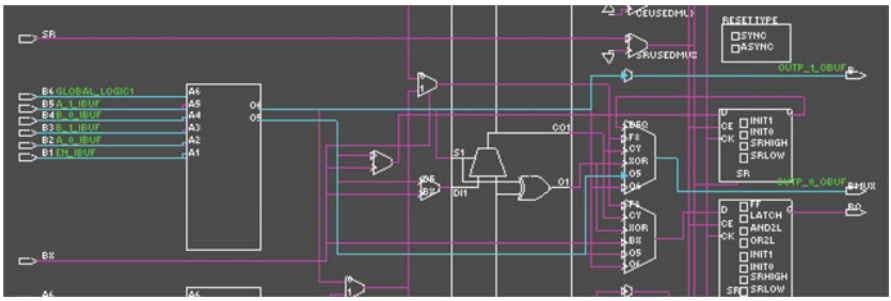


Fig. 14.10 Partial view of the detailed configuration of a slice in a CLB (slice marked in blue in Fig. 14.9). The highlighted path represents how the primitives in CLB are used

which describes how the HDL is mapped into LUTs. Additional information on the location of configurable logic block (CLB) and LUT is also present in the XDL file. Figure 14.9c shows how the CLB blocks are connected with each other to implement the functionality described in HDL. We can extract the locations and interconnections used by the original design from the NCD or XDL files. Additionally, even the post-PAR timing simulation model can also be used to get the locations used by the design (Fig. 14.10).

14.4.2 Trojans in HDL

An attacker can insert the Trojan in the HDL design. In the HDL, the attacker can easily track the logic elements or states to be used as an activator and deliver the Trojan payload to the target. Inserting Trojan at this level is significantly easier for an attacker, as the wires and logic elements can be found from the behavioral or structural code.

Example Consider the C17 circuit of ISCAS 85 benchmark [8]. The RTL representation is shown in Fig. 14.11. The number of internal logic nodes available in RTL for an attacker is four: N1, N11, N16, and N19 (except five primary inputs (PI) and two primary outputs (PO)). A functional Trojan inserted at the internal node N16 is shown in Fig. 14.12. The Trojan component is a MUX activated when N2 is logic "1" and the payload is switching result of node N16 with N11. This Trojan corrupts the functionality when activated. This example demonstrates the design space for an attacker with RTL or HDL circuits.

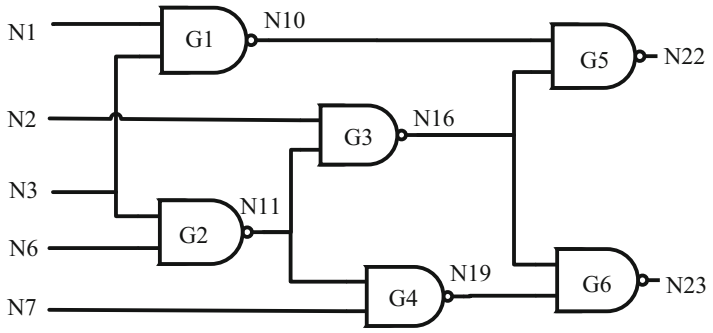


Fig. 14.11 Possible Trojan insertion points of ISCAS-85 C17 genuine HDL circuit. Each input and output of a gate are possible insertion points in RTL/HDL

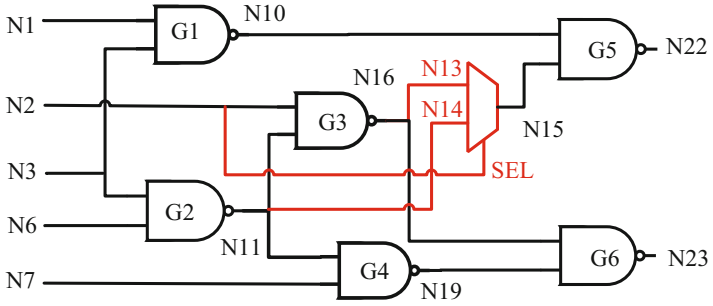


Fig. 14.12 A functional Trojan inserted in ISCAS-85 C17 HDL circuit. The Trojan is a MUX that corrupts the operation when N2 is logic 1

14.4.3 Trojans in Post-synthesis Netlist

An attacker can insert the Trojan in the post-synthesized netlist (called as the NCD file). To perform this, the post-synthesized NCD file can be converted to the XDL format. The XDL file contains the CLB configuration, the interconnection between CLB's, and information on routing inputs and outputs of the FPGA design. The mapped/routed netlist typically would have undergone logic optimization; wires and interconnections will be renamed and, hence, will not have the same wires and components as described in the behavioral/structural code. Logic merging by the CAD tools also makes it difficult to manually tap into internal logic. For instance, consider two equations, $A = B + CD$ and $E = F \oplus CM$. A CAD tool can implement these two equations by using just one 6-input LUT by selecting parameters appearing left-hand side as inputs to LUT and outputs E and F can be connected to O5 (output of 5-input LUT) and O6 (output of 6-input LUT), respectively.

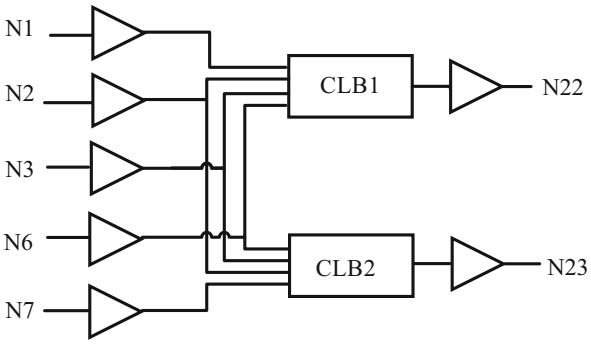


Fig. 14.13 Graphical view of C17 post-synthesis circuit

To insert Trojans at this level, the attacker needs to have information of renamed signals that can be obtained from synthesized netlist, by converting it to the EDIF. After obtaining the available wires and logic in the design, the attacker can adapt the Trojan activation circuit and payload circuit. The Trojan will be described in XDL language and inserted into the synthesized design by following a similar approach as explained in inserting Trojan at HDL level.

Example Consider the same C17 circuit targeted for FPGA implementation; the design space for an attacker changes significantly to insert Trojans in post-synthesis netlists. The equivalent C17 circuit after synthesis stage of implementation for Nexys4-DDR FPGA is shown in Fig. 14.13. All the internal nodes of RTL/HDL shown in Fig. 14.11 are unavailable. Xilinx CAD tool implements the C17 using just two LUTs due to optimization. The new post-optimized nodes are N221 and N231. N221 corresponds to merged logic from nodes N10 and N16. N231 corresponds to N11 and N16 in Fig. 14.11. The contents and the connections of the LUT are shown using LUT-perspective graphical representation as shown in Fig. 14.14. As node N16 is not available, the RTL Trojan described in Fig. 14.12 cannot be inserted in the same way after post-synthesis without disrupting the original mapped design. We present two methodologies of inserting this Trojan.

Additional hardware. Here we do not modify the mapped LUTs in the netlist. We create an additional LUT (say MUXLUT) that uses the inputs from IBUF N2 (Fig. 14.14) to get the same activation condition as the Trojan in Fig. 14.12. Another LUT, say Payload-LUT, to generate the payload using N1, N2, N3, and N6 is used. The output is connected to MUXLUT. The connection between N221 LUT and N221_OBUF in Fig. 14.14 is deleted, and N221 is connected to MUXLUT. When N2 is 1, MUXLUT will output the content of Payload-LUT to output else will output the content of N221 (normal output of C17). Without optimization, this requires two additional LUTs. However, only one LUT is required after optimization. The number of nodes involved including activator and payload is five: N1, N2, N3, N6, and N221. Hence, we can pack the logic into a single 6-input LUT.

Reuse existing hardware. In this case, we modify the content of original LUT itself to achieve the desired functionality. From the original C17 in Fig. 14.11 and

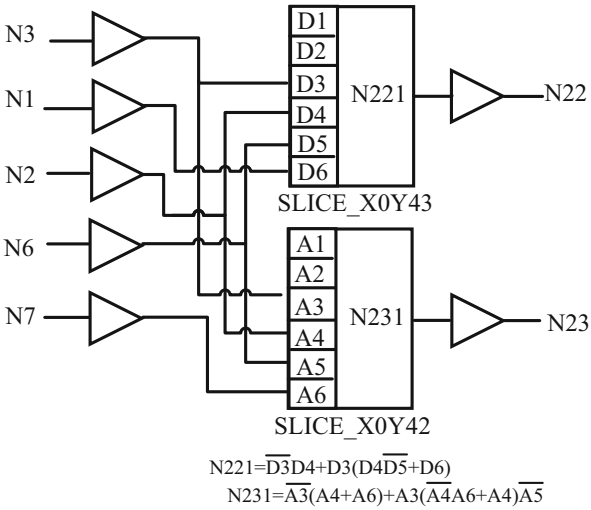


Fig. 14.14 Graphical view of C17 post-synthesis with malicious LUT contents

Trojan in Fig. 14.12, we can see that both depend on nodes – N1, N2, N3, and N6 – and the LUT in Fig. 14.14 after post-map consists of truth table expressed as a function of normal C17 circuit. We could modify the truth table to achieve the functionality of the Trojan and normal C17 using the same LUT. This way, we introduce the Trojan into the circuit without incurring any extra additional logic and without any changes to the hardware. In this case, the truth table of LUT 221 is given by the expression $(\overline{D3}D4) + (D3((D4\overline{D5}) + D6))$ for the Trojan-free circuit. We reprogram the LUT 221 with malicious truth table given by expression $((D6 + D4)(D6 + D5)D3)$, where D4 is the input of LUT driven by N2 (Trojan trigger). This way C17 will function correctly when the Trojan is not triggered (N2 is 0) and will deliver the Trojan payload when activated (N2 is 1). The required expression is found by converting the logical equivalence of Trojan with normal operation and minimizing it to the conjunctive normal form.

14.4.4 Case 3: Trojans in the Mapped/PAR Netlist

Inserting Trojans in mapped/PAR netlist follows the same procedure as inserting Trojans in the post-synthesized netlist. After the Trojan insertion, the XDL file is converted back to NCD (the CAD tool may rerun the "place and route" step), and the bitstream file is generated. After the Trojan is inserted at this level, the consequent step is generating the programmable bitstream file. Inserting Trojans in a fully mapped or routed design requires significant effort. The effort can be alleviated by designing scripts to insert Trojans and verify the routing of design. Detecting Trojans at this level is also considerably difficult, and the probability of detection of such Trojans is low.

14.5 Trojan in Bitstream

FPGAs are configured (or programmed) by loading a binary file, called as configuration bitstream or bitfile on the FPGA's configuration memory from a PC or over the network. The structure of a Xilinx configuration bitstream file is essential to insert the Trojans. The syntax and semantics of this bitfile are usually proprietary, with very less and basic information publicly available about the bitstream organization, e.g., in [38]. However, useful software tools have been designed in the academic community by utilizing this relatively scarce information, primarily to achieve greater control over the configuration process [23].

14.5.1 Xilinx Bitstream Structure

The description about the configuration bitstream is proposed in [10, 38] and [23] and focuses on Virtex-II FPGAs. It is described below.

Xilinx Virtex-II FPGA building blocks. In Virtex-II FPGAs, CLBs are the basic building blocks. They are typically organized in rows and columns and occupy the majority of the FPGA silicon area. A CLB consists of four slices, with each slice containing two flip-flops (FFs), two four-input LUTs used to implement logic functions, and multiplexers to connect and route the logic resources internally. Each CLB is connected to the RSM which provides paths to interconnect multiple CLBs in the FPGA. Block BRAM (BRAM) are used as memories. The input/output logic for each pin on the FPGA device is present in the input/output blocks (IOBs). These resources are represented using dedicated and separate columns in the bitstream.

Xilinx configuration bitstream file organization. The Xilinx Virtex-II configuration bitstream is stored in a configuration memory and is arranged in frames that are 1 bit wide and stretches from the top edge of the device to the bottom edge, as shown in Fig. 14.15. All operations must act over complete configuration frames as they are the smallest addressable segments of the configuration memory space. Single piece of hardware is not directly mapped by configuration memory frames, but, they configure a vertical slice of many physical resources [38]. The size of the frame depends on the device family such as Virtex-II and specific FPGA device (XC2V40) of interest for which the configuration file is being generated. The relation between the number of configuration bits and frame count is given by:

$$\text{No. of configuration bits} = \text{frame count} \times \text{frame size} \tag{14.1}$$

Apart from the configuration bits in the frames, the bitfile also contains other auxiliary information that includes the header of ASCII-encoded name of the top-level module of the design, cyclic redundancy check (CRC) words, device family, and timestamp. It also consists of a long trailing series of zeros at the end of the

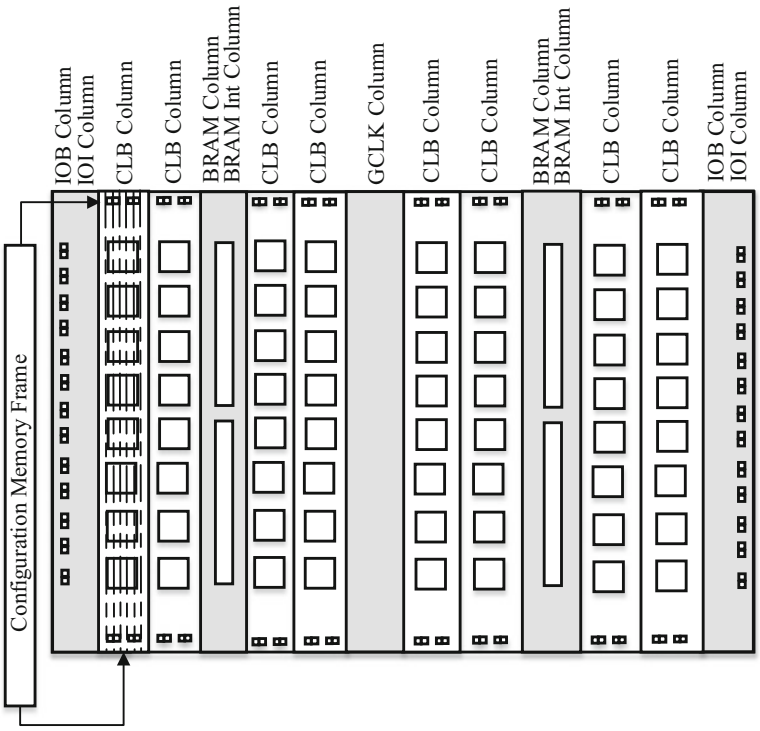


Fig. 14.15 Bitstream file organization of Xilinx Virtex-II [10]

files that signify no operation. Each configuration frame has a unique 32-bit address that consists of a block address, a major and minor address, and a byte number. The major address represents a specific column within a block, and the minor address represents a specific frame within a column.

14.5.2 Trojans That Modify the Bitstream

Modification of the configuration bitstream to add a hardware Trojan can be performed in two ways:

- *Type-I Trojans.* Without any resource or area overlap between inserted Trojan circuitry and the original circuitry on the FPGA. In other words, the part of the bitstream that configures the Trojan circuitry on the FPGA is simply inserted into the existing bitstream file at positions where no resources have been utilized originally, and the unoccupied resources of the FPGA are configured with the extra circuit. In this case, the functionality of the original circuit and the Trojan

are independent of each other as there is no interconnection between the original design and the hardware Trojan. This kind of Trojans are called Type-I Trojans.

- *Type-II Trojans.* There is an overlap between inserted Trojan circuitry and the original circuitry in terms of the resources occupied on the FPGA. The original circuit and the Trojan circuits are interconnected, and this kind of Trojans is called Type-II Trojans. Here, the Trojan can either extract information from the original circuitry or interfere with its functionality.

Type-I Trojans are easier to insert into the bitstream. However, Type-II Trojans are relatively harder to insert as it requires the detailed knowledge of the correlation between routing of the interconnections in the FPGA fabric with the bitstream. Even for a relatively low-end FPGA model like Virtex-II (device XC2V40), insertion of Type-II Trojan is an extremely complex task in the absence of supporting documentation to provide with some basic information. This has been tackled in [24]. However, the approach taken in [24] derives the design description in XDL, modifies this file, and transforms to bitfile. This approach does not directly operate on the bitfile.

14.5.3 Example Trojans in Literature

In [10], a software-based bitstream Trojan insertion technique using an unencrypted bitstream is demonstrated. The proposed Trojan attack directly modifies the bitfile to insert Trojans into it. The effectiveness of the attack lies in the fact that at present there is no verification mechanism for the correctness of FPGA bitfile, apart from an inbuilt CRC generating and matching mechanism, which can be disabled. This modification would be extremely difficult to detect before deployment. A major strength of this attack is that there will be no trace of Trojan insertion in the generated log files during FPGA CAD tool design processes, as the Trojan is inserted after these steps are completed.

Researchers have exploited the dynamic reconfiguration capability of modern FPGAs to add extra functionality [20]. Furthermore, many modern high-end FPGA families such as Xilinx Virtex family support bitstream encryption. However, there are many other FPGA families with numerous deployed FPGAs such Xilinx Spartan family FPGAs with no bitstream encryption support. Moreover, it was also recently reported that bitstream encryption, which relied on a Triple DES engine inside the FPGA, was broken using side-channel attacks [22].

Additionally, it has been shown in [33] that it is possible to reverse engineer crypto algorithms, identify cryptographic elements in bitstream, and perform modifications that can weaken the FPGA cryptographic implementations. It has also been demonstrated that by employing such modifications to bitstreams, one can easily extract the secret key from cryptographic algorithms [33].

14.6 Countermeasures Against FPGA Trojans

Hardware Trojan detection techniques in the literature can be classified into two categories: invasive and noninvasive. Invasive techniques require some modification to the original design to aid in fingerprinting the IC and to verify the authenticity after fabrication. A few examples include ring oscillator-based design-for-trust, IC camouflaging, and logic encryption [26, 27].

FPGAs consist of a massive amount of programmable components, and invasive techniques would require additional gates or hardware for each of these programmable components. Thus, invasive techniques are not feasible for FPGAs as this would lead to exponential silicon area overhead. Moreover, physical reverse engineering techniques can be used to test a small number of ICs, but do not guarantee that the remaining ICs are free from malicious modifications (Fig. 14.16).

Noninvasive techniques, on the other hand, do not modify the original design. Instead, a fingerprint of power, timing delay, and/or other side channels of a golden design, in combination with functional testing, are used. Most of the techniques in this category use statistical analysis to distinguish a malicious chip from a genuine one.

There are only a handful of works that detect hardware Trojans in the FPGA fabric [18, 21, 25] and are described below.

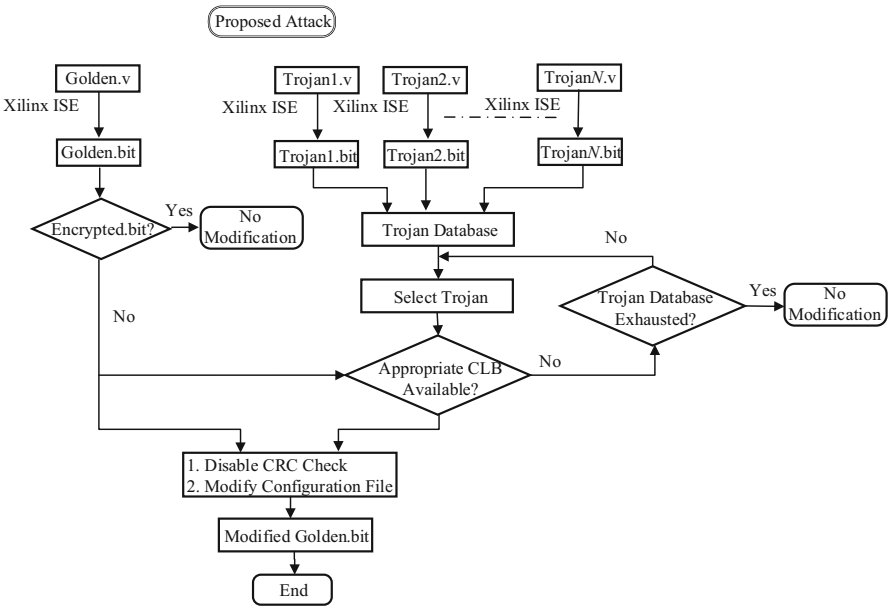


Fig. 14.16 Methodology of inserting Trojans in bitstream [10]

14.6.1 *Hardware Trojan Tolerance Using Modular Redundancy* [21]

A triple modular redundancy (TMR)-based technique is used to create a Trojan-tolerant design methodology in [21, 28, 29]. TMR is a renowned fault mitigation methodology used to mask circuit faults wherein three redundant copies of the original system perform a process, and the result is processed by a majority voting system to produce a single output. Any single fault in one of the redundant modules will not lead to an error at the output as the majority voter selects the result from the two faultless modules. TMR, however, leads to 3× area and power overhead. To reduce the overhead, adapted TMR (ATMR) is proposed where only two modules are used at a time, and the third module is employed only when there is a mismatch in the results of two active modules and shown in Fig. 14.17. An arbiter is used to identify the erroneous module. The experimental results show a 1.5× reduction in power by using ATMR with negligible performance and hardware overhead when compared to TMR.

The threat model assumes that the Trojans will not be activated simultaneously in the redundant modules. This assumption may not always be true. To overcome this limitation, the Trojan tolerance scheme is extended to improve the protection by using variants of the redundant modules. The key idea here is that instead of implementing all the redundant modules using the same logic circuit, the redundant modules implement variants of each other in terms of logic structures and storage blocks. This way the same nodes are not available across the redundant modules, and hence, if a Trojan activates across both the modules, then it is assumed to have a different effect on both the modules. The primary requirement is that the variants need to differ in both LUT and interconnect structures while keeping the functional behavior and parametric specifications intact.

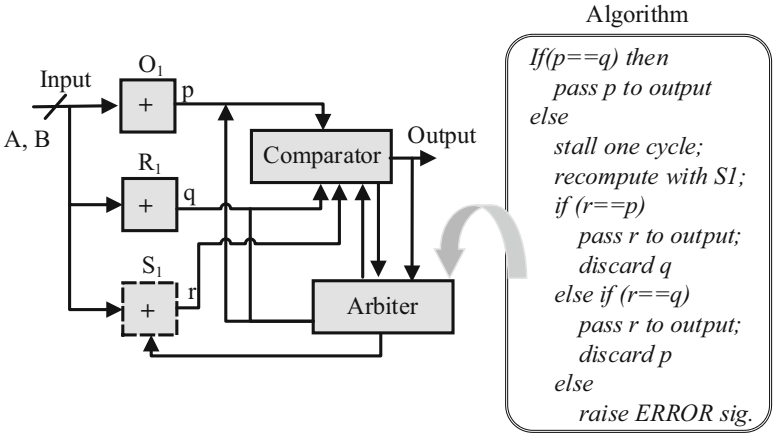


Fig. 14.17 Adapted TMR Trojan-tolerant technique [21]

To address the limitation of the comparator or arbiter being programmed on a compromised area, the use of majority voting system for comparator and arbiter is suggested. However, this approach has considerable hardware overhead.

Moreover, this technique can be used to identify functional Trojans that effect functionality of the original circuit. It might fail when key-leaking Trojans are present, as they typically do not interfere or change the functionality of original designs.

14.6.2 FPGA TrustFuzion [18, 25]

The FPGA TrustFuzion (FTZ) security mechanism is a nondestructive methodology to detect and isolate anomalies such as Trojans in the FPGA fabric. The authors use the term anomalies to indicate the presence of Trojans and reliability problems in FPGA fabric. Anomalies are detected based on the violation to the spatial correlation of intra-die PV in the FPGA fabric. These anomalies are then isolated such that any designs can run reliably even on a Trojan-infected FPGA chip. FTZ is based on the observation that physical characteristics of FPGA fabric's intra-die process variation (PV) display a huge amount of spatial correlation [2, 11, 12].

Based on the observation in [2, 12, 40], it is assumed that intra-die variations measured from the devices under test should exhibit a strong spatial correlation, with tiles that are closer exhibiting similar characteristics when configured with the same logic and interconnect resources. An anomaly is said to be detected if the spatial correlation is violated by even a single device or set of devices for FPGA under test. Spatial correlation violation is characterized by large deviation in spatial correlation caused by inconsistent variations among neighboring tiles. Ring oscillators are used to measure PV [18, 25, 39].

Anomalies are detected based on the spatial correlation violations. The device locations with anomalies are isolated and excluded from being used in the designs targeting the FPGA device with anomalies. The FPGA is then divided into different zones accounting for the exclusion of locations with anomalies. These FPGA areas are called TrustFuzion zones.

14.6.3 Bitstream Trojan Countermeasures

Countermeasures against bitstream Trojans involve:

Filling up unused resources of the FPGA with dummy logic. Type-I bitstream Trojans inserts malicious circuits in unused FPGA resources. This can be prevented by filling up all the unused resources with dummy logic. This way, the attack space for the adversary is reduced significantly [10]. However, this would increase the power consumption of the FPGA and adversely affect performance.
Grounding unused pins of an FPGA design. Type-II bitstream Trojans can leak secret key through covert channels using unused I/O pins of the FPGA. This can

be prevented by grounding the unused pins [10]. However, this does not prevent Trojans that leak key through the pins used in the FPGA design or any other side channels.

Built-in self-test. Type-II Trojans that weaken crypto-algorithms can be identified by using an integrated self-test that checks the functionality of the algorithm implemented in FPGA for a fixed key and plaintext [33]. However, integrity value has to be stored within the FPGA bitstream itself and can be identified and modified by an attacker.

Dedicated hardware to check CRC status. Type-I Trojan described in [10] can disable the CRC in the bitstream. A custom hardware can be placed between the configuration memory and the FPGA to check for disabling CRC and can be used to prevent the loading of bitstream when CRC is disabled [10].

Scrambling and descrambling the bitstream file. The bitstream can be scrambled by software, and a dedicated hardware can be used to descramble prior to loading into FPGA [10]. This would add an additional layer of complexity to an attacker and prevent direct modification of the bitstream. However, the scrambling mechanism has to be kept secret, and this acts a low-cost alternative to encrypting bitstreams in FPGA.

14.7 Conclusion

In the last decade, the use of FPGAs has increased significantly and even has been employed in various applications including mission critical systems. Hence, studying the threat landscape for FPGA security is critical. This chapter identifies the different FPGA Trojans that can be inserted at various phases of FPGA life cycle. Most research on FPGA security emulates Trojan on FPGA while trusting the FPGA fabric. Trojans can be inserted in even FPGA fabric, similar to any other type of ICs/ASICs. FPGA fabric Trojans that can be inserted by an untrusted foundry and a malicious actor in the supply chain in literature are identified, and the taxonomy is introduced. The bitstream files and the designs in HDL format can also be corrupted with Trojans.

This chapter also discussed the countermeasures proposed in the literature. Only a couple of technique exists to verify the physical fabric of FPGA for hardware Trojan. Trojans present in physical FPGA fabric could be detected by accounting for spatially correlated intra-die process variations. The intra-die process variation-based approach can identify anomalies contributing to delay or voltage change by locating inconsistent physical characteristics from the ones in close-by regions. Comprehensive work on malicious modification effects can be done by designing and simulating layout without and with anomalies. This would give a better insight into anomaly characteristics and would potentially aid in indicating the precise type of anomaly inserted into the FPGA fabric. However, many of the design details remain proprietary information and are not available to researchers, thus impeding the research.

References

1. S. Adee, The hunt for the kill switch (2008), http://spectrum.ieee.org/semiconductors/design/the-hunt-for-the-kill-switch. Last accessed 13 July 2016
2. A. Agarwal, D. Blaauw, V. Zolotov, Statistical timing analysis for intra-die process variations with spatial correlations, in *IEEE International Conference on Computer Design* (2003), pp. 900–907
3. Amazon, Amazon EC2 F1 instances – run custom FPGAs in the AWS cloud, https://aws.amazon.com/ec2/instance-types/f1/. Last accessed 12 May 2017
4. A. Amouri, M. Tahoori, High-level aging estimation for FPGA-mapped designs, in *IEEE International Conference on Field-Programmable Logic and Applications* (2012), pp. 284–291
5. G.T. Becker, F. Regazzoni, C. Paar, W.P. Burleson, Stealthy dopant-level hardware trojans, in *International Workshop on Cryptographic Hardware and Embedded Systems* (2013), pp. 197–214
6. K. Bernstein, D.J. Frank, A.E. Gattiker, W. Haensch, B.L. Ji, S.R. Nassif, E.J. Nowak, D.J. Pearson, N.J. Rohrer, High-performance CMOS variability in the 65-nm regime and beyond. IBM J. Res. Dev. **50**, 433–449 (2006)
7. A. Bravaix, C. Guerin, V. Huard, D. Roy, J. Roux, E. Vincent, Hot-carrier acceleration factors for low power management in DC-AC stressed 40 nm NMOS node at high temperature, in *IEEE International Reliability Physics Symposium* (2009), pp. 531–548
8. D. Bryan, The ISCAS85 benchmark circuits and netlist format. North Carolina State University, 25 (1985)
9. A.N. Campbell, K.A. Peterson, D.M. Fleetwood, J.M. Soden, Effects of focused ion beam irradiation on MOS transistors, in *IEEE International Reliability Physics Symposium* (1997), pp. 72–81
10. R.S. Chakraborty, I. Saha, A. Palchaudhuri, G.K. Naik, Hardware Trojan insertion by direct modification of FPGA configuration bitstream, in *IEEE Design & Test* (2013), pp. 45–54
11. H. Chang, S.S. Sapatnekar, Statistical timing analysis under spatial correlations, in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2005), pp. 1467–1482
12. B. Cline, K. Chopra, D. Blaauw, Y. Cao, Analysis and modeling of CD variation for statistical static timing, in *IEEE International Conference on Computer Design* (2006), pp. 60–66
13. DARPA, Defense Science Board (DSB) study on high performance microchip supply (2005). http://www.acq.osd.mil/dsb/reports/ADA435563.pdf. Last accessed 13 July 2016
14. Defense Tech, Proof that military chips from China are infected? (2012). http://www.defensetech.org/2012/05/30/smoking-gun-proof-that-military-chips-from-china-are-infected/. Last accessed 13 July 2016
15. EETimes, Report: Bogus U.S. military parts traced to China (2011). http://www.eetimes.com/document.asp?doc_id=1125076. Last accessed 13 July 2016
16. V. Huard, M. Denais, C. Parthasarathy, NBTI degradation: from physical mechanisms to modelling. Microelectron. Reliab. **46**, 1–23 (2006)
17. Intelligence Advanced Research Projects Activity, Trusted integrated circuits program. https://www.fbo.gov/utils/view?id=b8be3d2c5d5babbdffc6975c370247a6. Last accessed 13 July 2016
18. V. Jyothi, M. Thoonoli, R. Stern, R. Karri, FPGA trust zone: incorporating trust and reliability into FPGA designs, in *IEEE International Conference on Computer Design* (2016), pp. 600–605
19. R. Karri, J. Rajendran, K. Rosenfeld, M. Tehranipoor, Trustworthy hardware: identifying and classifying hardware trojans. Computer **43**, 39–46 (2010)
20. P. Lysaght, B. Blodget, J. Mason, J. Young, B. Bridgford, Invited paper: enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs, in *IEEE International Conference on Field Programmable Logic and Applications* (2006), pp. 1–6

21. S. Mal-Sarkar, A. Krishna, A. Ghosh, S. Bhunia, Hardware trojan attacks in FPGA devices: threat analysis and effective counter measures, in *ACM Great Lakes Symposium on VLSI Design* (2014), pp. 287–292
22. A. Moradi, A. Barenghi, T. Kasper, C. Paar, On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from Xilinx Virtex-II FPGAs, in *ACM conference on Computer and Communications Security* (2011), pp. 111–124
23. C.J. Morford, Bitmat-bitstream manipulation tool for Xilinx FPGAs. PhD dissertation, Virginia Tech (2005). https://theses.lib.vt.edu/theses/available/etd-12162005-144728/unrestricted/CMorford_Thesis.pdf. Last accessed 22 May 2017
24. J.-B. Note, É. Rannaud, From the bitstream to the netlist, in *International ACM/SIGDA Symposium on Field Programmable Gate Arrays* (2008), vol. 8, pp. 264–264
25. Y. Pino, V. Jyothi, M. French, Intra-die process variation aware anomaly detection in FPGAs, in *IEEE International Test Conference* (2014), pp. 1–6
26. J. Rajendran, V. Jyothi, O. Sinanoglu, R. Karri, Design and analysis of ring oscillator based design-for-trust technique, in *IEEE VLSI Test Symposium* (2011), pp. 105–110
27. J. Rajendran, Y. Pino, O. Sinanoglu, R. Karri, Logic encryption: a fault analysis perspective, in *Design, Automation Test in Europe Conference Exhibition* (2012), pp. 953–958
28. J. Rajendran, H. Zhang, O. Sinanoglu, R. Karri, High-level synthesis for security and trust, in *IEEE International On-Line Testing Symposium* (2013), pp. 232–233
29. J. Rajendran, O. Sinanoglu, R. Karri, Building trustworthy systems using untrusted components: a high-level synthesis approach. IEEE Trans. Very Large Scale Integr. Syst. **24**(9), 2946–2959 (2016)
30. SEMI, Innovation is at risk as semiconductor equipment and materials industry loses up to $4 billion annually due to IP infringement (2008). www.semi.org/en/Press/P043775. Last accessed 13 July 2015
31. Y. Shiyanovskii, F. Wolff, A. Rajendran, C. Papachristou, D. Weyer, W. Clay, Process reliability based Trojans through NBTI and HCI effects, in *NASA/ESA Conference on Adaptive Hardware and Systems* (2010), pp. 215–222
32. S.P. Skorobogatov, R.J. Anderson, Optical fault induction attacks, in *International Workshop on Cryptographic Hardware and Embedded Systems* (2002), pp. 2–12
33. P. Swierczynski, M. Fyrbiak, P. Koppe, C. Paar, FPGA Trojans through detecting and weakening of cryptographic primitives. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**, 1236–1249 (2015)
34. M. Tehranipoor, F. Koushanfar, A survey of hardware Trojan taxonomy and detection. IEEE Des. Test Comput. **27**, 10–25 (2010)
35. R. Torrance, D. James, The state-of-the-art in semiconductor reverse engineering, in *IEEE/ACM Design Automation Conference* (2011), pp. 333–338
36. Transparency Market Research, FPGA market – Global industry analysis, size, share, growth, trends and forecast, 2014–2020. http://www.transparencymarketresearch.com/field-programmable-gate-array.html. Last accessed 22 May 2017
37. USPTO, Piracy of intellectual property (2005). http://www.uspto.gov/about-us/news-updates/piracy-intellectual-property. Last accessed 13 July 2016
38. Xilinx, Virtex-II platform FPGA user guide (v 2.2). www.xilinx.com/support/documentation/user_guides/ug002.pdf. Last accessed 22 May 2017
39. X. Zhang, M. Tehranipoor, RON: an on-chip ring oscillator network for hardware Trojan detection, in *IEEE Design, Automation Test in Europe Conference Exhibition* (2011), pp. 1–6
40. W. Zhang, K. Balakrishnan, X. Li, D.S. Boning, S. Saxena, A. Strojwas, R. Rutenbar, Efficient spatial pattern analysis for variation decomposition via robust sparse regression. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**, 1072–1085 (2013)

Part VI
Emerging Trend, Industrial Practices, New Attacks

Chapter 15
Hardware Trust in Industrial SoC Designs: Practice and Challenges

Sandip Ray

15.1 Introduction

We are living in the regime of Internet of things (IoT)—a regime in which the number of connected, smart computing devices (or "things") exceeds the human population and is proliferating at an even faster pace [12]. Today, our environment includes billions of these computing devices sensing, identifying, tracking, and analyzing some of our most intimate personal activities including health, sleep, location, and network of friends. With such pervasive, ubiquitous, and invasive application of computing systems on all aspects of our life, it is critical to our well-being to ensure that we can trust these systems to behave reliably, securely, and according to specification.

Unfortunately, ensuring such trustworthiness in modern computing systems is a highly challenging enterprise. Vulnerabilities undermining trustworthiness can creep into the industrial system development flow at various stages, including architecture, design, system integration, and even validation. Furthermore, the effect of an "obscure" vulnerability can get magnified into a recipe for catastrophe given the reality that the same system design is replicated across multiple platforms and the same device is deployed in a myriad operating environments. On the other hand, system complexity is shooting up with progressive miniaturization of VLSI technology, advances in system architecture and design techniques, and exacting demands of modern applications. Furthermore, time-to-market requirements for these complex systems have become more aggressive than ever before, making it ever more challenging to perform comprehensive design validation. Consequently, it is getting imperative that we rethink our trust paradigms, understand the limitations

S. Ray (✉)
NXP Semiconductors Inc., 78735 Austin, TX, USA
e-mail: sandip.ray@nxp.com; sandip.r.ray@gmail.com

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_15

of the current paradigms, and invent technologies that ensure trustworthiness of computing systems being developed under these challenging constraints.

This chapter is about hardware trust validation techniques. Trustworthiness, as suggested above, would encompass the entire platform and include functionality, software compatibility, electrical and physical characteristics, marginality, and security. Clearly, it is impossible to provide a comprehensive treatment of this entire spectrum within the context of a single article. Rather, keeping with the motivations of the collection, we focus here on trust issues as they directly pertain to *system security*, i.e., techniques to prevent an adversary from gaining access to some unauthorized information or data in the system.

Even with this restriction, however, trust challenges in a modern hardware design is a vast topic. Security challenges arise from a myriad potential attack opportunities, the large attack surface of modern computer systems, variety of attacker motivations, etc. In Sect. 15.2 we provide a brief high-level overview of this diversity of factors. For the reader interested in details of these, we recommend a variety of other related research in the area [1, 11, 14]. The focus of this chapter is on the practicum of addressing these challenges, i.e., trust validation technologies being actually used in industry. The goal is to provide a consolidated reference for security trust validation practice and emphasize the big gaps between the state of the practice and the trustworthiness requirements in modern hardware designs.

Unlike most other chapters in this treatise, the trust validation practice itself is not confined solely to Trojan detection. While handling Trojans remain an important target of hardware trustworthiness requirements, current approaches to (security-based) trust assurance attempt to provide general flows and methodologies for ensuring trust independent of whether the source of vulnerability is a malicious Trojan or innocuous mistake. Indeed, in many cases, the difference is unclear from the detection standpoint.

The remainder of the chapter begins with an introduction of the spectrum of trust vulnerability sources as they pertain to security. We then look at detection practices currently being use. These vary depending on the adversarial and deployment models. Rather than going into detail on any specific aspect of the techniques, we focus here on providing a comprehensive overview of the slew of techniques involved, from software/design targets to those involving the PCB and the platform. The exposition in this chapter does not assume any familiarity on hardware trust challenges or previous experience with any of the trust validation techniques used in practice.

15.2 The Spectrum of Trust Challenges

Perhaps the most important source of system vulnerability in design that directly affects security arises from the globally distributed nature of the modern system design flow. One of the critical achievements of the Internet-based communication and computing technology over the last decades has been to bring humans across the

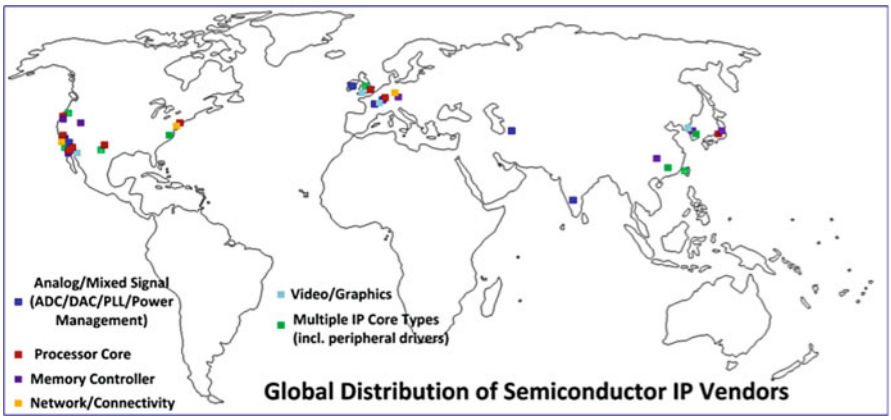


Fig. 15.1 An SoC would often contain hardware IP blocks obtained from entities distributed across the globe (Source [13])

globe closer together than any time in the history. We are living today in a highly connected world where physical proximity is getting increasingly irrelevant as an obstacle for collaboration and coordination. Indeed, it is common for a single company to have globally dispersed teams across multiple continents, with workforce highly diverse in terms of culture, religion, history, and heritage (cf. Fig. 15.1). One upshot of this globalization has been the emergence of increasingly global supply chain. Virtually, any complex product development today—independent of sector—involves multiple players typically dispersed physically all across the world. This of course implies that all players of this complex, distributed supply chain must work in concert to create the product. Unfortunately, like any distributed system, this is hard to get right. In particular, from the supply chain perspective, there are at least three unique challenges that influence system vulnerability.

Interface Consistency: A modern hardware design is typically developed as a System-on-Chip (SoC), whereby a collection of hardware blocks of coherent functionality (often referred to as "Intellectual Properties" or "IPs") are developed by various IP providers which are then integrated to provide the complete system functionality. In today's supply chain, individual IPs are developed by different organizations (whether multiple organizations inside the same company or by independent IP vendors), geographically and culturally dispersed. For this architecture to be trustworthy, it is important that the interfaces of through which IPs communicate with each other are clearly defined and respected by each all IPs participating in such communication. Geographical separation and distributed control can often make such communication difficult, leaving each party with a different expectation on the nature of the interface. Of course, a primary effect of this gap is a compromise in functional correctness; e.g., if an IP `A` can send a message across an interface that IP `B` never expected to receive, then the message may never get processed leading eventually to starvation or deadlock. However, the mismatch of expectation

can lead to leakage of secure or sensitive information as well; e.g., a receiving IP might mishandle a security-sensitive message as a result of ambiguities in interface specification.

Functional Expectation Consistency: Consistency in functional specification is a more general case of interface consistency, whereby ambiguities in functional specification may lead to security vulnerabilities. For example, suppose that an IP `A` is required to maintain the invariant that some data variable should not exceed a certain size bound. This invariant may be assumed (potentially by other IP providers) to define container data structures for data from the variable with hard size bounds. If the invariant is not enforced (perhaps inadvertently) by the developer of IP, `A` the result may be a buffer overflow resulting in a security compromise or backdoor. Other results from inconsistent functional expectations for various IP developers may include confusion between the use of British vs. metric units for quantities, real-time expectations, etc.

Design Modifications for Fun or Profit: Another obvious challenge in a globalized, distributed design flow is to have effective control over the quality of the design. An upshot of lack of control is the possibility of malicious modifications of the design—perhaps by a rogue employee or a malicious IP vendor—that enables the system to leak information under certain (difficult-to-trigger) circumstances. For example, in 2012, a study by a group of researchers in Cambridge revealed an undocumented silicon-level backdoor in a highly secure military-grade ProAsic3 FPGA [16], which was later described as a vulnerability induced unintentionally by on-chip debug infrastructure. In a recent report, researchers have demonstrated such an attack where a malicious upgrade of a firmware destroys the processor it is controlling by affecting the power management system [7]. This problem—also known as the Trojan problem—has been discussed extensively in various chapters of the book, and we do not go into details here. However, we point out that such malicious modifications may come not only from a malicious intent but perhaps even an inadvertent or innocuous optimization of the design. This is particularly true in the context of side-channel Trojans. For example, an obvious performance optimization in computation-intensive algorithms is to develop heuristics that speed up the common cases. If this is done without proper care, it can result in a side-channel Trojan that enables the external user to infer, through observing computation speed, whether the common case is being exercised or not. Trusted and untrusted CAD tools pose similar trust issues to the SoC designers. Such tools are designed to optimize a design for power, performance, and area. Security optimization is not an option in today's tools; hence, sometimes during the optimization, new vulnerabilities are introduced [9].

The Problem of Untrusted Foundries: Another crucial outcome of globalization (and potential lack of quality) is the potential for counterfeit systems and devices. During distribution of fabricated SoC designs through a typically long distribution supply chain, consisting of multiple layers of distributors, wholesalers, and retailers, the threat of counterfeits is a growing one. These counterfeits can be low-quality clones, overproduced chips in untrusted foundry, or recycled ones [19].

The above discussion suggests that we need to rethink security concerns of our devices as it passes through the entire distributed supply chain, identify potential vulnerabilities that can be introduced at each point, and determine mitigation actions. Of course, given the distributed nature of the production process, the mitigation action can only be defined from the point of view of one player, who treats the rest of the players in the supply chain as potentially untrusted. In the remainder of the chapter, we will take two specific points of view. From the perspective of design, we will treat the SoC integration house as the trusted party that treats individual IP vendors as untrusted; from the perspective of the full platform, we will treat the chip supplier as the trusted party—after all, they are selling the chip and have some stake in the chip being trusted—with all other players (e.g., foundry, production, validation, etc.) being untrusted.

15.3 Security Policies and Enforcement

A key component of security assurance in a modern SoC designs is enforcement of *security policies*. A security policy is a (system-level) requirement that specifies how access to security-sensitive assets must be handled by the system. Here, we give a brief high-level overview of security policies and typical industrial flow to enforce them. The following are two example policies:

- *Example 1:* During boot time, data transmitted by the crypto engine cannot be observed by any IP in the SoC other than its intended target.
- *Example 2:* A programmable fuse containing a secure key can be updated during manufacturing but not after production.

In general, security policies can specify a variety of requirements ranging from access control, information flow, liveness, and time-of-check time-of-use. A key component of trust assurance in a modern SoC design involves ensuring that the policies are indeed enforced as expected. In current industrial practice, this is done by a process referred to as *threat modeling*. Threat modeling roughly involves the following five steps, which are iterated until completion:

Asset Definition: Identify the system assets governing protection. This requires identification of IPs and the point of system execution where the assets originate. As discussed above, this includes statically defined assets as well as those generated during system execution.

Policy Specification: For each asset, identify the policies that involve it. Note that a policy may "involve" an asset without specifying direct access control for it. For example, a policy may specify how a secure key $\mathscr{K}$ can be accessed by a specific IP. This in turn may imply how the controller of the fuse where $\mathscr{K}$ is programmed can communicate with other IPs during boot process for key distribution.

Attack Surface Identification: For each asset, identify potential adversarial actions that can subvert policies governing the asset. This requires identification, analysis, and documentation of each potential "entry point," i.e., any interface that transfers data relevant to the asset to an untrusted region. The entry point depends on the category of the potential adversary considered in the attack; e.g., a covert-channel adversary can make use of nonfunctional design characteristics such as power consumption or temperature to infer the ongoing computation.

Risk Assessment: The potential for an adversary to subvert a security objective does not, in and of itself, warrant mitigation strategies. The risk assessment and analysis are defined in terms of the so-called DREAD paradigm, composed of the following five components: (a) damage potential; (b) reproducibility; (c) exploitability, i.e., the skill and resource required by the adversary to perform the attack; (d) affected systems, e.g., whether the attack can affect a single system or tens or millions; and (e) discoverability. In addition to the attack itself, one needs to analyze the likelihood that the attack can occur on-field, motives of the adversary, etc.

Threat Mitigation: Once the risk is considered substantial given the likelihood of the attack, protection mechanisms are defined, and the analysis must be performed again on the modified system.

Obviously, performing the above activities manually over the range of system assets and policies, in the presence of subtleties related to implicit expectations, potential adversaries that break the risk/mitigation analysis, and complex interplay between functional behavior and security constraints, is a daunting task. Admittedly, there is a host of available tools to assist in the different steps, e.g., tools for documenting steps in threat and severity identification identifying security scenarios, etc. [8, 17]. Nevertheless, the key architectural decisions and analysis still depend highly on human insights.

15.4 Trust Validation on Design and Implementation

Security validation remains the primary tool for trust assurance in modern SoC designs. The goal of this activity is to ensure that protections on security-critical assets work as intended and identify ways in which an adversary can potentially subvert such protection. Obviously, security validation technology forms a crucial component of threat modeling. But validation in general is a complex activity, spanning the entire design life cycle with various interdependent pieces. In current practice, we divide security validation into four categories.

Functional Validation of Security-Sensitive Design Features: This is essentially extension to functional validation but pertains to design elements involved in critical security feature implementations. An example is the cryptographic engine IP. A critical functional requirement for the cryptographic engine is that it encrypts and decrypts data correctly for all modes. As with any other design block, the

cryptographic engine is also a target of functional validation. However, given that it is a critical component of a number of security-critical design features, security validation planning may determine that correctness of cryptographic functionality to be crucial enough to justify further validation beyond the coverage provided by vanilla functional validation activities. Consequently, such an IP may undergo more rigorous testing or even formal analysis in some cases. Other such critical IPs may include IPs involved in secure boot, on-field firmware patching, etc.

Validation of Deterministic Security Requirements: Deterministic security requirements are validation objectives that can be directly derived from security policies. Such objectives typically encompass access control restrictions, address translations, etc. Consider an access control restriction that specifies a certain range of memory to be protected from DMA access; this may be done to ensure protection against code-injection attacks or protect a key that is stored in such location, etc. An obvious derived validation objective is to ensure that all DMA calls for access to a memory whose address translates to an address in the protected range must be aborted. Note that validation of such properties may not be included as part of functional validation, since DMA access requests for DMA-protected addresses are unlikely to arise for "normal" test cases or usage scenarios.

Negative Testing: Negative testing looks beyond the functional specification of designs to identify if security objectives can be subverted or are underspecified. Continuing with the DMA protection example above, negative testing may extend the deterministic security requirement (i.e., abortion of DMA access for protected memory ranges) to identify if there are any other paths to protected memory in addition to address translation activated by a DMA access request and, if so, potential input stimulus to activate such paths.

Hackathons: Hackathons, also referred to as *white box hacking*, fall in the "black magic" end of the security validation spectrum. The idea is for expert hackers to perform goal-oriented attempts at breaking security objectives. This activity depends primarily on human creativity, although some guidelines exist on how to approach them (see discussion on penetration testing in the next section). Because of their cost and the need for high human expertise, they are performed for attacking complex security objectives, typically at hardware/firmware/software interfaces or at the chip boundary.

To perform the spectrum of security validation tasks, various validation technologies are used in current practice. Three common technologies are *fuzzing*, *penetration testing*, and *formal verification*. Here, we briefly summarize these technologies as applied to security validation practice.

Fuzzing: Fuzzing, or fuzz testing [18], is a testing technique for hardware or software that involves providing invalid, unexpected, or random inputs and monitoring the result for exceptions such as crashes or failing built-in code assertions or memory leaks. It was developed as a software testing approach and has since been adapted to hardware/software systems. In the context of security, it is effective for exposing a number of potential attacker entry points, including through buffer

or integer overflows, unhandled exceptions, race conditions, access violations, and denial of service. Traditionally, fuzzing uses either random inputs or random mutations of valid inputs. A key attraction to this approach is its high automation compared to other validation technologies such as penetration testing and formal analysis. Nevertheless, since it relies on randomness, fuzzing may miss security violations that rely on unique corner-case scenarios.

Penetration Testing: Penetration testing, also referred to as *intrusion testing*, is a process of systematically attacking a computing system or platform to identify security vulnerabilities. This is a highly manual process often performed by expert hackers with deep knowledge of the design and implementation. Penetration testing can target the system at any level of abstraction, from architecture to the fabricated IC. In Sect. 15.5, we will discuss penetration testing techniques that target the physical IC. That process involves sophisticated electronic aids, including ion beam, laser attacks, fault attacks, etc. When the target is a design or software (before the IC has been fabricated), the penetration testing job is to identify functional vulnerabilities. This is done in the following three phases:

1. *Attack Surface Enumeration.* The first task is to identify the features or aspects of the system that are vulnerable to attack. This is typically a creative process involving a smorgasbord of activities, including documentation review, network service scanning, and even fuzzing or random testing (see below).
2. *Vulnerability Exploitation.* Once the potential attacker entry points are discovered, applicable attacks and exploits are attempted against target areas. This may require research into known vulnerabilities, looking up applicable vulnerability class attacks, engaging in vulnerability research specific to the target, and writing/creating the necessary exploits.
3. *Result Analysis.* If the attack is successful, then in this phase the resulting state of the target is compared against security objectives and policy definitions to determine if the system was indeed compromised. Note that even if a security objective is not directly compromised, a successful attack may identify additional attack surface which must then be accounted for with further penetration testing.

Note that while there are commonalities between penetration testing and testing done functional validation, there are several important differences. In particular, the goal of functional testing is to simulate benign user behavior and (perhaps) accidental failures under normal environmental conditions of operation of the design as defined by its specification; penetration testing goes outside the specification to the limits set by the security objective and simulates deliberate attacker behavior.

Formal Reasoning: This involves making use of mathematical logic to either derive a security assurance requirement formally or identify flaws in the target system (architecture, design, or implementation). Application of formal methods typically involve significant effort, either in the manual exercise of performing deductive reasoning or in developing abstractions of the security objective which are amenable to analysis by automated formal tools [2, 4]. In spite of the cost, however, the effort is justified for highly critical security objectives, e.g., cryptographic algo-

rithm implementation. Furthermore, for some critical properties, automated formal methods can be used in a lightweight manner as effective state exploration tools. For example, TOCTOU property violations often involve scenarios of overlapping execution of different instances of the same protocol, which are effectively exposed by formal method tools [6]. The efficacy of formal reasoning in particular to find corner-case vulnerabilities in design has resulted in EDA vendors developing targeted "apps" on top of formal tools specifically for security targets and enabled ease of use for those cases. For example, Cadence JasperGold™ provides apps for security path verification that targets information flow properties of hardware [5]. In spite of such efforts—and their successful use in various industrial projects—formal verification remains a complex exercise requiring highly skilled dedicated personnel.

15.5 Platform-Level Trust Assurance

The preceding sections covered trust assurance techniques in the context of integrating untrusted IP and software models during SoC system integration. Unfortunately, these techniques cannot adequately cover *implementation errors* that can lead to side-channel or covert-channel information leaks that depend significantly on the physical characteristics of the device, e.g., leaks through voltage, current, or emission characteristics. To detect such leaks before deployment, a key component of security assurance involves intrusion testing on the physical (silicon) implementation of the device. Typically, such approaches involve removing and adding materials to the integrated circuit (IC) to access and modify information in the chip. Potential attacks include optical inspection, reading internal data, or adding/removing/modifying wires in silicon. In this section, we summarize some of the key activities involved. As with penetration testing on design models, these activities involve essentially mimicking the attack on the device as an external hacker might do, identifying design or implementation vulnerabilities, and performing mitigatory actions. Note of course that these activities are highly nontrivial and creative and often make use of highly sophisticated instruments.

Hardware Reverse Engineering: Hardware reverse engineering refers to the act of inferring the functionality and implementation of a module or system from layout or silicon implementations. This process typically involves the following four steps: (1) chip imaging, through polishing, microscoping, etc., to identify the layout-level structure from silicon, (2) recognizing and interpreting layout-level structures through sophisticated pattern recognition to turn the layout-level design into gates, and (3) inferring higher-level functionality from the resultant gate-level netlist through a combination of gate-level simulation, static analysis, and machine learning techniques. Obviously, each such activity is complex and computation intensive and has a large probability of error. Furthermore, these tests are *destructive*; consequently, care must be taken to apply them judiciously to

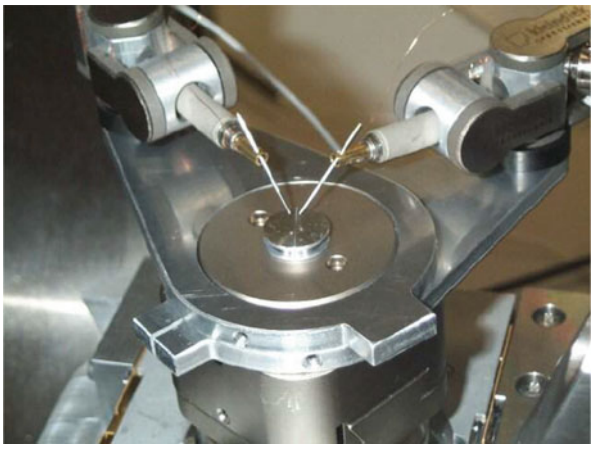


Fig. 15.2 A nanoprobe device. The needles are controlled by motorized arm for precise scanning and modification of IC

maximize the chances of success. Unfortunately, tools in this area that scale to industrial systems are sorely lacking. (This might be considered a benefit, since if available, such tools could potentially be used for malicious hacking once the device is in-field.) At the time of this writing, a viable tool is Degate [15] that facilitates interpretation of circuit images. Developing reverse engineering tools that enable validation while not permitting in-field reverse engineering is a challenging question.

Nanoprobing and Atomic Force Microscopy: This activity entails the use of sophisticated scanning probes (Fig. 15.2) with resolutions of the order of a nanometer or lower. In current practice, we often use nanoprobes with fine probe needles of step size less than 1 nm, with motorized control for effective efficiency. Such devices enable accurate and precise movements on electronic commands as well as extremely precise scanning of the device.

Side-Channel Analysis: Side-channel analysis attacks refer to the inference of information and data in the device through measurement of its nonfunctional characteristics. Side-channel attacks used for trust assurance in industry today include timing attacks (i.e., measurement of runtime differences to statistically extract a secret asset), power attacks (i.e., measurement of power or differential power), electromagnetic emanation attacks, communication attacks (i.e., inferring secrets based on the response of the system to specific commands such as error messages for a failed the user-supplied response to a system challenge), and even heat measurements. As we develop devices and infrastructures with hundreds of devices, the number and diversity of potential side channels are also significantly high. Consequently, side-channel analysis for trust assurance must be judicious in navigating effectively through this space.

Fault Attacks: Fault attacks are used to determine if a faulty or defective behavior of a circuit can be exploited to infer critical information in the device. This attack involves two stages. In the first stage, a fault is introduced in the computation

through an external device. Common fault introduction sources include laser beam, clock jitter, voltage burst, or focused ion beam (among others). The specific device used depends on a variety of factors, including the type of fault being introduced, the target chip, the amount of control available, etc. Once a fault is introduced, in the second phase, one attempts to use the faulty behavior as a side channel to infer secrets in the system.

15.6 Security Certification

Certification is a process by which an organization (typically a product supplier) can provide *some* assurance on the efficacy of security features of a product to the external world. Certification is an essential component of industrial trust assurance. In this section, we provide a flavor of the variety of certification processes employed for security-critical hardware (and software systems).

Security certification involves two additional players (in addition to the supplier or manufacturer of the product being supplier) who are both independent of the product supply chain:

- A laboratory or facility (referred to as *Information Technology Security Evaluation Facility* or "ITSEF") that evaluates the security claims of the product;
- An organization (referred to as *Certification Body*) that approves the security claims based on the evaluation reports from ITSEF.

To illustrate the role and scope of certification, we discuss below two specific certification standards, Common Criteria and FIPS. In addition to those below, there are other (nonstandard and sometimes proprietary) application-specific certification criteria for devices targeted toward specific highly sensitive applications [3], e.g., for credit cards, banking systems, etc.

Common Criteria: Common Criteria for Information Technology Security Evaluation [20], colloquially referred to as *Common Criteria* (CC), is an international standard (ISO/IEC 15408) for guidelines and specifications developed for evaluating information security products, specifically to ensure they meet an agreed-upon security standard for government deployment. The criteria can be used on a diversity of products from operating system down to hardware. As with other certifications, the manufacturer provides the product collateral (e.g., product overview, security requirements, enforcement architecture, validation flow, etc.), ITSEF performs the evaluation, and Certification Body provides the final certification. To standardize the certification process, Common Criteria sets forth a collection of Evaluation Assurance Levels (EAL) for the security assurance claim of the product. The assurance level provides a numerical rating specifying the rigor of evaluation. Each EAL corresponds to a set of *security assurance requirements* (or "SARs") which covers the development of the product. There are seven EALs listed in the common criteria, with EAL1 being the most basic and EAL7 the most rigorous (and hence most expensive to apply).

FIPS 140-2: The Federal Information Processing Standard (FIPS) [10] is a United States government computer security standard applicable specifically to cryptographic modules. Its goal is to coordinate cryptographic components of a design that includes both hardware and software elements. It covers specification, interfaces, and ports of the modules, their roles in service and authentication, physical security, deployment targets, electromagnetic interference, etc. FIPS 140-2 defines four levels of security, where level 1 is the lowest and level 4 is the highest. Level 1 focuses on the basic security requirements, algorithms, and functionality of the module; level 2 deals with tamper resistance, e.g., seals and coatings to protect against attacks due to physical access; level 3 requires mechanisms to prevent a malicious agents from gaining access to the security parameters inside the module; and level 4 requires protection and isolation against unauthorized attempts at physical access.

While certification is a critical component in ensuring trustworthiness of security-critical systems and devices, it is important to carefully understand the evaluation result and exactly what is being certified; a certificate does *not* necessarily ensure that the device is immune to attacks and vulnerabilities. For example, EAL6 of Common Criteria requires a security policy model and evidence that the functional specification satisfies certain security requirements. This is obviously a critical requirement in ensuring trustworthiness of the target device. However, this certificate alone does *not* imply the implementation of the device would indeed adhere to the security policy model used in the certification. A customer deploying the device in a specific environment must comprehend such discrepancies and potential subtleties and determine how to use the result of certification to ensure trustworthiness of the application.

15.7 Conclusion

We have provided an overview of security and trust assurance and validation mechanisms employed in current industrial practice. Trust assurance techniques applied in practice today range over a large class of technologies and model targets, from architecture to silicon implementation. The high complexity and expense in the process is in part because security assurance today is inherently reactive: one identifies security attacks and their mitigation strategies principally by creativity and experience with past attacks, not by a thorough, systematic analysis of deployment requirements and vulnerability sources. Indeed, given the diversity of attacks and complexity of devices, it is unclear how to be more systematic. Furthermore, the expertise required for the various assurance activities varies wildly as we move from architecture to RTL/software models and finally to silicon implementation attacks. This suggests the need to develop a cohesive research on security that can seamlessly move toward various abstraction level. We hope that the exposition here will contribute to that research by showing the gaps in the current industrial practicum.

References

1. S. Bhunia, S. Ray, S. Sur-Kolay (eds.), *Fundamentals of IP and SoC Security: Design, Validation, and Debug* (Springer, Cham, 2017)
2. E.M. Clarke, O. Grumberg, D.A. Peled, *Model-Checking* (The MIT Press, Cambridge, 2000)
3. EMVCo: EMV® Specifications. See http://www.emvco.com
4. A. Gupta, Formal hardware verification methods: a survey. Formal Methods Syst. Des. **2**(3), 151–238 (1992)
5. JasperGold Security Verification Path App. See https://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform/security-path-verification-app.html
6. S. Krstic, J. Yang, D.W. Palmer, R.B. Osborne, E. Talmor, Security of SoC firmware load protocol, in *IEEE HOST* (2014)
7. E. Messmer, RSA security attack demo deep-fries Apple Mac components (2014). See http://www.networkworld.com/news/2014/022614-rsa-apple-attack-279212.html
8. Microsoft Threat Modeling & Analysis Tool version 3.0 (2009)
9. A. Nahiyan, K. Xiao, D. Forte, Y. Jin, M. Tehranipoor, AVFSM: a framework for identifying and mitigating vulnerabilities in FSMs, in *Design Automation Conference* (2016)
10. NIST: Federal information processing standards publication: security requirements for cryptographic modules. See http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf
11. S. Ray, J. Bhadra, Security challenges in mobile and IoT systems, in *IEEE System-on-Chip Conference* (2016)
12. S. Ray, Y. Jin, A. Raychowdhury, The changing computing paradigm with Internet-of-Things: a tutorial introduction. IEEE Des. Test Comput. **33**(2), 76–96 (2016)
13. S. Ray, S. Bhunia, P. Mishra, Security validation in modern SoC designs, in *Fundamentals of IP and SoC security: design, validation, and debug*, ed. by S. Bhunia, S. Ray, S. Sur-Kolay (Springer, Cham, 2017), pp. 9–28
14. S. Ray, E. Peeters, M. Tehranipoor, S. Bhunia, System-on-Chip platform security assurance: architecture and validation. Proc. IEEE (2017). https://doi.org/10.1109/JPROC.2017.2714641
15. Reverse engineering integrated circuits with degate. See http://www.degate.org
16. S. Skorobogatov, C. Woods, Breakthrough silicon scanning discovers backdoor in military chip, in *CHES* (2012), pp. 23–40
17. J. Srivatanakul, J.A. Clark, F. Polac, Effective security requirements analysis: HAZOPs and use cases, in *7th International Conference on Information Security* (2004), pp. 416–427
18. A. Takanen, J.D. DeMott, C. Mille, *Fuzzing for Software Security Testing and Quality Assurance* (Artech House, Norwood, 2008)
19. M. Tehranipoor, U. Guin, D. Forte, *Counterfeit Integrated Circuits: Detection and Avoidance* (Springer, Cham, 2014)
20. The Common Criteria. See http://www.commoncriteriaportal.org/cc/

Chapter 16
Conclusion and Future Work

Swarup Bhunia and Mark M. Tehranipoor

Hardware Trojan horse or hardware Trojan attacks have been the subject of active research for the past decade. The scope of this research has broadened over time – with the possibility of malicious hardware modification extending to different stages of integrated circuit's life cycle (from IP to microchips) as well as to different levels of hardware abstractions (from microchips to printed circuit boards). It now represents a new branch of hardware security. Security concerns caused by hardware Trojans are relevant to various parties involved in the life cycle of electronic hardware – from system-on-chip designer to original equipment manufacturers (OEMs) to end users. The fact that the electronic hardware is subject to malicious alterations is an intriguing departure from conventional wisdom, which assumes that hardware is hard to compromise, is fundamentally more secure and trustworthy than its software counterparts, and hence can be used as trust anchors or "root of trust" for diverse computing and communication systems.

Over the past decades, research on hardware Trojan has followed one of the two broad tracks: (1) exploration of the threat models and attack modalities and (2) approaches to deal with the attacks at different levels. The first track has focused on discovering how hardware can be modified for malicious purpose at different stages of hardware manufacturing process and how different payload can be implemented to satisfy multitude of attack objectives. The second track has focused on countermeasures, which fall into two classes: (1) hardware trust verification and (2) "design-for-security" (DfS) solutions. The problem of hardware Trojan is rapidly creating a new field of *hardware trust verification*, which aims at analyzing a hardware – either IP or chip or PCB – to verify its trustworthiness with respect to malicious design artifacts. The first class of countermeasure comprises of wide-ranging hardware trust verification solutions, which include targeted functional test,

S. Bhunia (✉) • M.M. Tehranipoor
Department of Electrical & Computer Engineering, University of Florida, Gainesville, FL, USA
e-mail: swarup@ece.ufl.edu; tehranipoor@ece.ufl.edu

S. Bhunia, M.M. Tehranipoor (eds.), *The Hardware Trojan War*,
https://doi.org/10.1007/978-3-319-68511-3_16

statistical test, and formal verification approaches. The second class, on the other hand, follows a preemptive approach where protection against Trojan attacks is built into a design before the hardware is fabricated. Such solutions can try to "harden" a design making malicious implantation difficult or infeasible, or make trust verification easier, or allow online monitoring of Trojan activation to detect and/or tolerate Trojan effect during field operation. In particular, design techniques for *Trojan resilience* or *Trojan tolerance* (similar to the concept of fault tolerance) during field operation usher in a promising new direction of Trojan research.

The book has tried to capture all these aspects of hardware Trojan research over the past decade (2007–2017) in a collection of chapters contributed by prominent researchers and practitioners in this field. The chapters try to cover the spectrum of vulnerabilities of modern electronic hardware in terms of Trojan attacks as well as the whole gamut of possible protection approaches. We hope the chapters would collectively serve as a comprehensive resource on hardware Trojan attacks and would provide researchers, students, and practitioners with fundamentals of hardware Trojan attacks. We are glad to be able to collect these high-quality chapter contributions to produce the first book of this kind that includes compendium of materials on this important topic in the field of hardware security.

Over the past decade, Trojan attacks and hardware trust assurance have primarily been of interest to the Department of Defense. This is due to the potential for catastrophic consequence caused by hardware infected with malicious implantation in diverse military applications. Untrusted hardware has emerged as serious concerns for our national security. Such concerns are aggravated by rapidly diminishing control over microelectronic design and fabrication cycle and a complex distributed supply chain for electronics which involves multiple untrusted parties. The relative ease of cloning modern chips and PCBs allows an adversary to incorporate cloned chips with embedded malicious circuits into a supply chain. Recent reports show that cloning of chips adds a new and critical dimension to the hardware trust issue. It is extremely challenging for a system integrator (e.g., OEM) to verify trust of chips acquired from a supply chain primarily due to the lack of golden chip instances or reference designs. Trust verification of electronic components acquired from a supply chain will remain a critical problem and an area of active research in the foreseeable future. Such a problem may trigger key innovations in hardware trust verification fundamentally different from the scenario where a chip designer tries to verify the trust of chips fabricated in an untrusted foundry. In the latter case, the chip designer would have access to the entire design and knowledge of its expected functional/parametric behavior.

The issue of hardware trust is, however, gradually being accepted as a problem of growing significance in the mainstream semiconductor industry. Malicious implants of different forms, e.g., design modifications in untrusted IPs that may leak secret information, are being considered to be a valid concern for the system-on-chip designers. We expect to see increasing emphasis in research and development activities related to hardware Trojan from the major chip design and electronic design automation companies in the coming years.

16.1 Future Work

While the book covers well prior and ongoing research activities in this area, we hope it would stimulate new research explorations in many directions. In particular, it is expected to create new research pathways on the following topics:

1. More Complex Attacks

Research on mounting complex and powerful attacks on a computing system enabled by malicious hardware modification is expected to remain an active research topic. Clever hardware modification that can be leveraged by software or data during field operation needs to be investigated and appropriate countermeasures developed. Such modifications can lead to leaking secret information from inside a hardware unit or cause malfunction, possibly at an adversary-controlled time. Hardware modification can also be studied for its effectiveness in mounting different forms of physical attacks, including side-channel attacks – e.g., fault injection attacks, where a Trojan can facilitate key leakage through fault injection. Finally, a Trojan can be implemented that exploits the collusion across multiple parties – e.g., a rogue physical designer in the design house and an adversary in the foundry. Such Trojans can be significantly more difficult to detect with post-silicon test since they may alter functional or parametric behavior of a design under a very rare operating condition unlikely to be activated during test.

2. Automatic Vulnerability Analysis

There is a growing need to develop CAD tools to automatically analyze a design with respect to its trustworthiness or its vulnerability for malicious modifications. The first tool will be important for evaluating trustworthiness for untrusted IPs that are integrated into an SoC. Such a tool would help us to understand possible malicious behavior of an IP with respect to known Trojan models, before it is used by a SoC designer. Note that it is possible that in addition to deliberate malicious alterations, apparently benign nonfunctional design artifacts, such as design-for-test (DFT) or design-for-debug (DFD) infrastructure, can be exploited by an attacker for malicious intent, such as information leakage. An automatic vulnerability analysis tool should ideally catch these issues too. The second tool is important to analyze how vulnerable a design is with respect to malicious modification in a foundry. It should highlight the components or regions of a design which are more vulnerable than others. There will be a growing need for such automatic analysis tools with increasing reliance of chip designers on third-party IP blocks and growing trust concerns in the chip manufacturing process.

3. Metrics and Benchmarks

Associated with the CAD tools, new research is needed to develop trust metrics and benchmarks for different hardware abstraction levels. For example, ways to quantify trust for hardware IP blocks would involve a number of important research

questions – e.g., (1) what structures can be considered untrusted, (2) what functional behavior of the IP or components of an IP (e.g., the arithmetic logic unit of a processor) can raise trust issues, and (3) can we come up with a set of well-defined properties which can be checked for an IP and the results can be used for trust quantification? A trust metric needs to consider all aspects of trust evaluation to come up with an aggregate trust value. Similarly, there will be growing demand for trust benchmarks to evaluate trust verification or trusted design solutions. While there have been some prominent efforts in this regard, further research is needed to augment the benchmark suites with respect to emerging attack modalities and to cover various hardware abstractions.

4. Trust Verification

It is well accepted that a "silver bullet" solution which can reliably protect against Trojan attacks of all types, forms, and sizes is extremely difficult to achieve. Effective trust verification which can provide high confidence against diverse trust issues would remain to be an active research. Trust verification through logic testing and side-channel analysis has been extensively investigated. The book has dedicated chapters that elaborate on these solutions. However, the current solutions are often effective for very specific types of Trojan or have unrealistic assumptions. Further, golden-free trust verification problem has not received adequate attention from the research community. There is a clear need for significant new research in this area.

5. Design-for-Security (DfS) Approaches

While trust verification solutions are attractive since they do not incur hardware or design overhead and can work for legacy components, they suffer from limited effectiveness and fail to provide complete trust assurance. Trust verification solutions can provide much higher confidence if combined with commensurate design solutions. Trust-aware design approaches try to make Trojan insertion difficult, i.e., effectively "harden" a design, and/or make a Trojan easy to activate/observe. Low-overhead design solutions which are amenable for automation and provide significant benefit in trust assurance are expected to remain attractive research topic in both academia and industry.

6. Trojan Attacks in Nanoscale Devices

Continuous scaling of CMOS technology has brought us to sub-10 nm technology regime, where new nanoscale devices with interesting switching characteristics have emerged. These devices often exploit unique material properties and introduce fundamentally new device structure or state variables (e.g., spins of electrons, mechanical state of a cantilever, or resistive state of a solid dielectric material). They exhibit promising behavior in many aspects – e.g., performance, power, nonvolatility, reliability, manufacturability, and integration density. The operating principles of nanoscale devices are also expected to modify the concepts of hardware Trojans. For example, the current in nanoscale transistors strongly depends on the channel stress, which can be modified through changes in the process steps or even small layout changes. The more challenging problems will be reliability

Trojans where malicious changes are engineered to radically accelerate the device aging process. Similar Trojans are possible in non-charge-based devices, e.g., by marginally modulating spin polarization, or magnetic tunnel junction (MTJ) resistance. Future research is expected to focus on exploring Trojan attacks for nanoscale devices and Trojan-resilient design approaches for these devices.