Novel Approaches for Microelectronics Security and Test

by

Ziqi Zhou

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 7, 2021

Keywords: Hardware Security, Piracy, Logic Locking, Obfuscation,
Design-for-Security, Tampering, Hardware Trojan, Magnetic Skyrmion, Fault Modeling.

Abstract

Due to the globalization in semiconductor industry, the cost of maintaining a foundry is enormous. Hence, most integrated circuit (IC) design houses have become fabless. Typically, a design house acquires multiple third party intellectual property (IP) cores for a system on a chip (SoC) and sends a contract to a foundry/fab for manufacturing and test. The global supply chain of semiconductor design, manufacture, and test opens up a Pandora's box of harmful threats. These can be overproduction or counterfeiting of ICs, piracy of intellectual property (IP), or insertion of hardware Trojans. To prevent these threats, researchers have proposed solutions that include hardware metering, logic locking, IP watermarking, and split manufacturing to address threats.

Logic locking is a widely studied design-for-security (DFS) measure. It protects the IP by inserting logic gates in the design to allow it to become completely functional only when a secret key is programmed in. The inserted logic commonly consists of XOR/XNOR gates, multiplexers (MUXs) or look-up tables (LUTs). The existing logic locking can be disabled using the existing state-of-art methods that include Boolean satisfiability (SAT) based attacks, probing, and tampering attacks. One can obtain the secret key from a functional chip and then unlock any number of locked ones as the secret key is same for every chip.

In this dissertation, we are the first to propose a new secure logic locking method by implementing a design-for-security (DFS) architecture. We modify the scan cell such that it can be set to hold its previous state. To accomplish this the output of the flip-flop (FF) is fed back to its input through a multiplexer (MUX). The proposed infrastructure can prevent the adversary from obtaining the key by accessing the scan chains. Our modification does not affect the testability of the chip during the normal manufacturing flow, which may include the test before activation, post-silicon validation, and debug. Moreover, the proposed secure cell can disable scan dump after functional activation. The proposed design is resistant to various known attacks at a cost lower than 1% area overhead.

Besides the design-for-security (DFS) architecture, we also propose a novel attack that can break any logic locking techniques that rely on stored secret key. This proposed Tampering Attack on

Any Key-based Logic Locked Circuit *TAAL* inserts a malicious hardware Trojan in the netlist, which, when activated, leaks the secret key to an adversary. The attack approach is to tamper with the locked netlist in order to extract the secret key information. The untrusted foundry can extract the netlist of a design from the layout/mask information, which makes it feasible to implement such a hardware Trojan with the adversary's knowledge. Three types of *TAAL* attacks are proposed for extracting the secret key through hardware Trojans placed at various locations in the netlist. Models for both combinational and sequential hardware Trojans are introduced such that they would evade manufacturing tests. An adversary only needs to choose one hardware Trojan out of a large set of possible Trojans to launch the proposed attack.

Given the above-mentioned Trojan attacks, a method to detect this tampering is necessary. In this dissertation, we devise tests that would detect a Trojan in a manufactured chip. Based on the two parts of a Trojan, namely, a trigger derived as a Boolean function of any set of signals and a payload (typically, an XOR gate) inserted on a signal line, we develop a test generation model. A single-line trigger combined with a single payload line gives a set of $2K \times (K{-}1)$ Trojans in this model for a circuit with K signal lines. Tests for these are shown to be the vectors that detect "conditional stuck-at" faults, for which we give a test generation algorithm using standard Automatic Test Pattern Generation (ATPG) tools. This procedure allows us to define and measure a Trojan coverage metric for tests. Results show scalability of these tests, besides being more effective in detecting real Trojans than N-detect stuck-at test vectors or random vectors.

Considering the previous hardware Trojan detection methods, we realize that the fault modeling can both benefit manufacturing tests and hardware Trojan detection. We develop a fault modeling methodology to generate test patterns to detect defects in Skyrmion circuit, which is an emerging technology. We examine breaks, extra material, etching blemishes, bridges in nanotrack interconnects, etc., forming a set of 19 technology-specific defects in the skyrmion gate structures. We believe we are the first to characterize such defects using magnetic simulation. Simulator $\boldsymbol{MuMax^3}$ is used to exhaustively simulate all gates, and each defect is mapped onto an analyzable fault model using the principle of fault equivalence. Experiments on benchmark circuits demonstrate that tests for all nanotrack breaks can be found using the available ATPG and simulation tools. Some defects are classified as technology-specific defects. For example, a bridge between two nanotracks results in simultaneous AND and OR functions on respective nanotracks. This dissertation presents the test generation results for the Skyrmion versions of benchmark circuits for defects that can be expressed as a single stuck-at faults.

This dissertation provides a comprehensive overview of attackers and their attack choices. The proposed DFS structure can provide sufficient security to resist SAT-based attacks, and the proposed hardware Trojan detection method can effectively detect potential risks in the circuit. For emerging technologies, a technology-specific defect to logic-fault modeling approach of testing is proposed. The proposed future work provides definitive paths into new directions for the research community.

Acknowledgments

I would like to express my heartfelt gratitude to Dr. Ujjwal Guin, my graduate advisor, for his encouragement and guidance during my time at Auburn University. His encouragement and guidance paved the way for my successful research projects and thesis completion. I would like to express my gratitude to Dr. Vishwani Agrawal for his experience and insight into research. This dissertation would not have been possible without his constant support and guidance. Also, special thanks to Dr. Adit Singh for his great help and efforts. I would like to thank Dr. Peng Li for leading me into a new field of scientific research, and helping me achieve results in this field. I would also like to thank Dr. Karmaker for being my university reader providing me with valuable comments and kind support. I want to extend my gratitude to the committee members again for their time, support, and advice toward my research and thesis preparation. Thanks to all of my labmates and colleagues for the valuable information I acquired during my course and study work. The lab exercises and brainstorming sessions taught me a lot about my field of study, and I owe much gratitude to them. Finally, I would like to thank my parents and friends for their unwavering support during my academic career.

Contents

List of Figures

List of Tables

List of Abbreviations

3PIP	Third-Party Intellectual Property
AE	Attacker's Effort
ATPG	Automatic Test Pattern Generation
BEOL	Back End of Line
CSP	Conditional SAF Pattern
CSP	Conditional Stuck-at Fault Pattern
DFS	Design-for-Security
DIP	Distinguishing Input Pattern
DMI	Dzyaloshinskii–Moriya Interaction
EDA	Electronic Design Automation
FEOL	Front End of Line
FF	Flip-Flop
FIB	Focused Ion Beam
FM	Ferromagnetic
GDSII	Graphic Database System II
GLN	Gate Level Netlist

HM	Heavy Metal
HT	Hardware Trojan
HTAP	Hardware Trojan Activation Pattern
IC	Integrated Circuit
IP	Intellectual Property
LUT	Lookup Table
MTJ	Magnetic Tunnel Junction
MUX	Multiplexer
NVM	Non-volatile Memory
OASIS	Open Artwork System Interchange Standard
P&R	Place and Route
PC	Percentage of Cones
PI	Primary Input
PO	Primary Output
PUF	Physically Unclonable Function
RE	Reverse Engineering
RTL	Register-Transfer Level
SAF	Stuck-at Fault
SAT	Boolean Satisfiability
SC	Secure Cell
SE	Scan Enable

SEM	Scanning Electron Microscope
SI	Scan In
SoC	System on-chip
SOT	Spin Orbit Torque
SSA	Single Stuck-at
STT	Spin Transfer Torque
TAP	Trojan Activation Pattern
TM	Tamper-proof Memory
TS	Test Suppressor
VCMA	Voltage Controlled Magnetic Anisotropy

Chapter 1

Introduction

Due to the continuing trend of device scaling and the resulting increase in the complexity of the fabrication process, most (SoCs) design companies no longer maintain their own manufacturing units, commonly known as foundries or fabs. The cost of building and maintaining such foundries is reported to be more than several billion dollars [7]. This high cost has forced many design companies to become fabless and adopt a horizontal semiconductor integration model, where the SoC designers contract foundries and assemblies for production. In parallel, the continuous trend of device scaling has enabled designers to compact the design of SoCs and reduce the overall area and cost. As the complexity of modern SoCs is growing exponentially, design of a complete system by a single SoC designer becomes impossible. Instead of designing the whole SoC from scratch, reuse of pre-designed blocks has become a popular solution adopted by the semiconductor industry. SoC designers generally use various third-party intellectual properties (3PIPs) to limit the research and development (R&D) expenses, which may also cause a trust issue from the 3PIPs. The globalization of the semiconductor industry and outsourcing of the design and manufacturing of integrated circuits (ICs) cause IP piracy and IC overproduction to become major threats because of the untrusted entities [8–16].

To prevent IP piracy and IC overproduction, different solutions have been proposed over the years and obfuscation or locking of a circuit netlist was introduced as a viable solution [1,6,17–19]. Logic locking aims to hide the functionality of an IP by inserting additional logic elements into the original design. The intentionally added additional lock elements usually include XOR gates, MUXes, or lookup tables (LUT). No matter what method a designer uses to lock the circuit, without applying the correct secret key, the actual function of the original circuit will be hidden. Attackers cannot analyze the structure of the circuit through the locked netlist and thus cannot copy or modify the circuit design. Once the correct key value is programmed into the chip in a secure environment, the correct output of the locked chip will be generated. The confidentiality of the secret key is indispensable for the security of the lock design, as it is the only barrier to prevent the piracy of the IP. The secret key is stored in tamper-proof non-volatile memory (NVM) to prevent an adversary from accessing it using physical means.

1.1 Motivation

Logic locking has become a prominent method to address the threats incurred from untrusted manufacturing. However, recently a Boolean Satisfiability (SAT)-based attack [20] has demonstrated an effective way of extracting the secret key through iteration of distinguishing input patterns (DIPs). Due to the robustness and efficiency of SAT-based attacks, the latest logic locking research work mainly focused on defending against SAT-based attacks. At the same time, different physical attacks [21–24] have also shown effectiveness in breaking a secure locking technique. These attacks become feasible as an untrusted foundry has the ability to obtain all the layout information since it has access to the GDSII or OASIS file [25]. An untrusted foundry can also initiate a tampering attack through malicious modification by inserting a hardware Trojan without the designers' knowledge. Because the secret key is stored in non-volatile memory (NVM), and connections are made from the NVM to the key gates, the secret key will directly leak the to adversary once the hardware Trojan is activated. Following this motivation, we have proposed a series of methods to prevent IP piracy and IC overproduction.

1.2 Contributions

- A novel design-for-security (DFS) architecture to prevent IP piracy and IC overproduction is proposed. This DFS architecture can achieve complete protection against the state-of-art attacks without modifying the existing IC manufacturing and test flow, only with the cost of a small area overhead. The proposed DFS architecture allows full scan-based structural manufacturing tests for the unlocked design. The unlocked design can be tested with complete functional and structural tests in a secure environment with the secret key.

- A novel tampering attack based on malicious modification of any key-based locked circuit is proposed. The proposed attack is based on implanting a hardware Trojan into the original design to obtain the secret key from any existing logic locking circuit.
- Models for a combinational hardware Trojan and a sequential hardware Trojan have been presented. The model of combinational hardware Trojan is named Type-n Trojan, where n is the number of Trojan's trigger inputs. The model of sequential Trojan is constructed on the basis of combinational hardware Trojan, with an additional state element added in the design as a counter. The Trojan is activated when the activation signal is applied r times.
- A hardware Trojan detection technique based on conditional stuck-at fault patterns (CSP) is proposed. The conditional stuck-at fault test pattern generation starts with Type-1 Trojan. With a reasonable test length, all Type-1 Trojans can be detected. The same test patterns can also be used to detect a large percentage of higher-order hardware Trojans.
- A defect characterization method is proposed for an emerging technology: skyrmion-based logic circuits. The defects are mapped onto an analyzable fault model. Each defect is represented by either a technology-independent single stuck-at fault or a technology-dependent fault. A test pattern generation method is then proposed based on the available EDA tools.

1.3 Organization of this Dissertation

The rest of this dissertation is organized as follows:

- Chapter 2 introduces the basic concept of logic locking and summarizes the state-of-art in this field. The related attack methods on logic locking are presented. In addition, the background of the hardware Trojan is introduced in this chapter.
- Chapter 3 introduces our design-for-security (DFS) architecture to prevent the aforementioned attacks by obfuscating a netlist. The proposed design is resistant to various known attacks including the well-known SAT-based attack. Importantly, the proposed design does not limit the testability of the chip during the normal manufacturing flow in any way, including the post-silicon validation and debug.
- Chapter 4 proposes a new hardware Trojan based attack method on the SAT-resilient design. According to our analysis, even if the locked circuit provides sufficient security protection

against SAT-based attacks, a malicious foundry can be a feasible attacker who would unlock any number of chips and sell overproduced and defective parts.

- Chapter 5 extends the modeling and test generation for combinational hardware Trojans which appeared in Chapter 4. A Trojan may be added to the verified netlist without the knowledge of the designer or user causing unexpected malfunction or data theft when the device is in use. For this new type of hardware Trojan attack, especially the Trojan mentioned in Chapter 4, a method that can effectively detect this type of Trojan is urgently needed. In this chapter, we introduce a Trojan detection method that would detect a Trojan in a manufactured chip.
- Chapter 6 introduces the defect characterization and testing of skyrmion-based logic circuits based on fault modeling. According to the Trojan detection method proposed in Chapter 5, it appears that fault modeling not only adds efficiency to testing, but also plays a critical role in HT detection. Thus, fault modeling has a wider application. In this chapter, we advance fault modeling beyond the conventional CMOS circuits, to a most recent emerging technology.
- Chapter 7 is the conclusion of this dissertation, which also outlines the possible future work. Hardware security has played an important and significant role in CMOS devices and will continue to do so for emerging technologies, especially when the traditional semiconductor device size is approaching the physical limit. We end this chapter with a list of some future research directions.

Chapter 2

Background and Prior Work

In this chapter, we discuss fundamentals essential for understanding the core concepts in this thesis. We give a background of logic locking and the existing attack methods. Previous research on hardware Trojans is also summarized.

2.1 Logic Locking

Logic locking is a widely accepted protection technology against IP Piracy and IC Overproduction. Logic locking modifies the original design by adding additional key gates to the netlist, thereby generating circuits that rely on key functions. The challenges for protecting a circuit against hardware security threats have been the driving force for developing different techniques to limit the amount of circuit information that can be recovered by an adversary. Logic locking has emerged as a field of significant interest from researchers, as it can provide complete protection against IC overproduction and IP piracy [1, 6, 17–19].

The objective of logic locking is to obfuscate the inner details of the circuit and make it infeasible for an adversary to reconstruct the original netlist. Logic Locking hides the circuit's functionality by inserting additional logic gates into the original design, which are termed *key gates*. In addition to the original inputs, the locked circuit needs secret key inputs to key gates from on-chip tamper-proof memory (see Figure 2.1(a) for details). The correct functionality of the design is obtained when the key inputs receive the proper secret key value. Applying an invalid key to the key gates would result in incorrect functionality of the locked design. Note that for a securely locked circuit, the design details cannot be recovered using reverse engineering.

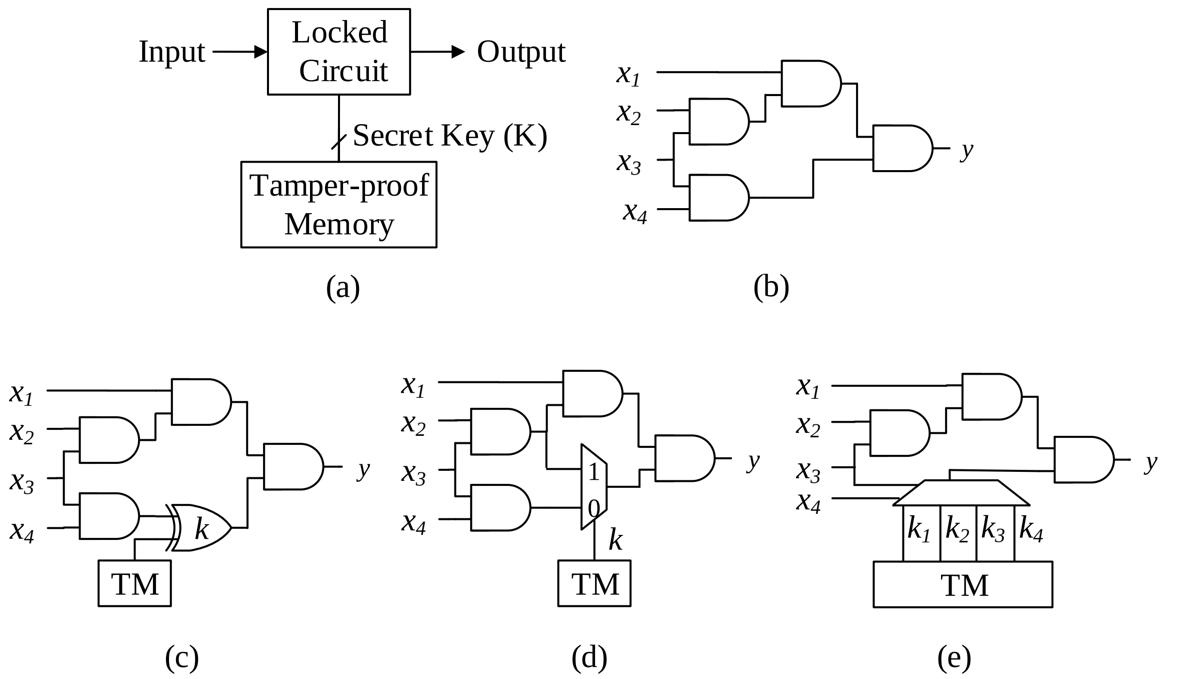


Figure 2.1: Different logic locking techniques. (a) A locked circuit, where the secret key (K) is programmed in a tamper-proof memory (TM). (b) Original circuit. (c) XOR-based locking [1]. (d) MUX-based locking [2,3]. (e) LUT-based locking [4].

Different logic locking methods were devised over the years and can be categorized into three different categories. First, XOR-based logic locking, shown in Figure 2.1(c), has received much attention due to its simplicity. In this technique, a set of XOR or XNOR gates are inserted as key gates [1,5,6,26–30]. The secret key is stored in tamper-proof memory (TM), and connections are made from TM to the key gates. Second, in the MUX-based logic locking technique [2,3], multiplexers (MUX) are inserted so that one of its input is correct, which is the actual net of the circuit. The other input of the MUX is incorrect, which is a dummy net randomly selected from the netlist. This technique is shown in Figure 2.1(d). The select signal of the MUX is associated with the key bit from the tamper-proof memory. The correct signal goes through the MUX upon applying a valid key value; otherwise, the incorrect signal propagates in the netlist. Third, in LUT-based logic locking, [4,31,32], shown in Figure 2.1(e), a look-up table with several key inputs is used to lock the netlist. The LUTs replace a combinational logic in the design, making it difficult to predict the output as it depends on several different key values.

The research community has proposed several attacks to exploit the security vulnerability on a logic locked circuit. Subramanyan et al. [20] first showed that a locked circuit could be broken using Boolean Satisfiability (SAT) analysis. The SAT attack algorithm, attributed as an oracle-guided attack, requires a locked netlist, which can be recovered using reverse engineering and a functional chip with a valid key stored/programmed in its tamper-proof memory. In this attack, an adversary can query an activated chip and observe the response. Note that the SAT attack requires access to the internal nodes of the circuit through the scan chains, which is common in today's netlist for implementing Design-for-Testability (DFT) [33]. The SAT attack works iteratively to eliminate incorrect key values from the key space using distinguishing input patterns (DIPs). A DIP is defined as an input pattern for which two sets of hypothesis keys produce complementary results. By comparing these with the output of an unlocked chip, one set of hypothesis keys is discarded. The SAT attack works efficiently as it discards multiple hypothesis keys in one iteration.

Thereafter, researchers have focused on improving and developing locking techniques to be resilient against the SAT attack. Subsequent work in this direction involved Anti-SAT [26], SAR-Lock [27], SFLL [30], design-for-security (DFS) architecture [6,28,29]. However, Subramanyan et al. has shown that SFLL can be defeated through FALL attack [34]. The attack is built on three primary steps, namely, structural analysis, functional analysis, and key confirmation. The structural analysis is performed to identify the gates that are the output of the cube stripping function in SFLL. After identification of these candidate gates, the functional analysis targets the property of cube stripping functions, which results in a set of potential key values. Finally, the key confirmation algorithm identifies the correct key from the set of potential key values.

As the SAT-attack is based on the availability of accessing the internal states of a circuit through the scan chains, Guin et al. proposed placing multiple flip-flops capturing signals controlled by different key bits at the same level of the parallel scan chains, which were used in the current test compression methodologies [6]. However, a vulnerability existed in this design, when an adversary performs multi-cycle tests, such as delay tests (transition delay faults and path delay faults) [33]. This leads to the necessity for developing a new design-for-security (DFS) architecture to prevent leaking of the key during any manufacturing tests [28,29]. This design prevents scanning out the internal states after a chip is being activated, and the keys are programmed/stored in the circuit.

Apart from SAT-based attacks, probing attacks [21, 35] have also shown serious threats to the security of logic locking, where an attacker makes contact with the probes at signal wires in order to extract sensitive information, mainly, the secret key. With the help of a focused ion beam (FIB), a powerful circuit editing tool that can mill and deposit material with nanoscale precision, an attacker can circumvent protection mechanisms and reach wires carrying sensitive information. However, the countermeasures reflect the complexity of shield-structure and nanopyramid structures as the defense, making it difficult to perform these attacks [36, 37]. Recently, Zhang et al. proposed an oracle-less attack to extract the key from locked circuits [38]. The notion of this attack is to compare the locked and unlocked instances of repeated Boolean functions in the netlist to predict the key. A solution was proposed to countermeasure the attack as well.

2.2 Hardware Trojan

Ensuring the security of integrated circuits becomes a major challenge due to the globalization of the semiconductor industry. Majority of system-on-chip design companies outsource their production across the world to fabrication units (fabs or foundries) due to a massive cost (several billion dollars [7]) for building and maintaining such foundries. This creates the threat of *hardware Trojans* (HT), which is a leading security concern for government and industry [9, 39–45]. A hardware Trojan is a malicious altercation to the original design to modify its functionality such that an adversary can gain control of the system. An adversary may insert a hardware Trojan into a design to interrupt its normal operation in the field. The Trojan would act like a "silicon time bomb" [41]. It can also create a backdoor in a secure system to give access to critical system functionality or leak secret information to an adversary.

The hardware Trojan generally consists of two parts, the trigger and payload, as shown in Figure 2.2. The trigger can activate hardware Trojan when it meets certain activation conditions. The input of the trigger can come directly from the primary input (PI) or from the internal nodes of the circuit. Although shown here as an AND gate, triggers can be any logical function. When the Trojan is activated, for example, when the input B and C are both 1 AND gate output will also change from 0 to 1, it transfers the payload to the circuit by modifying its function. A two

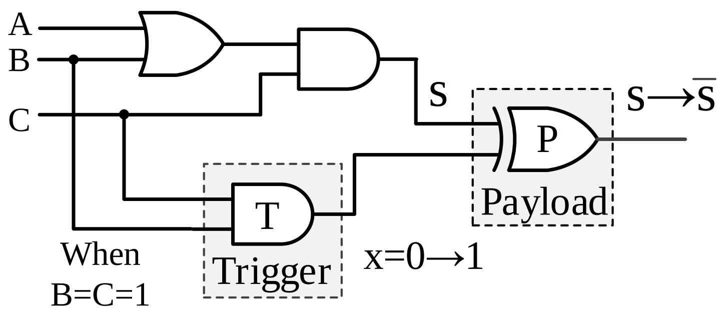


Figure 2.2: A Hardware Trojan.

input XOR gate with inputs from the trigger and a node in the circuit, can be used for this purpose. The output of the payload is brought back to the circuit.

Researchers have proposed numerous techniques to detect and prevent HTs. These techniques are broadly classified into two groups, namely, solutions targeted for the detection of HTs, and solutions designed for preventing an adversary to insert an HT in a design. The detection methods for HTs can further be classified into logic testing [46–50], and side-channel analysis [51–56]. Prevention methods can be grouped into design-for-trust measures [57–60] and split manufacturing [61,62].

The overall aim is to detect HTs in chips manufactured in an untrusted environment and, thus, prevent Trojan infected devices from getting into the electronics supply chain. Logic testing can be used to detect these Trojans, where we apply stimuli to primary inputs (PIs) and observe responses at primary outputs (POs) [43, 46, 48–50]. Detection of an HT occurs when there is a mismatch between the observed and expected responses. Such detection of an HT through logic testing does not have any impact on the process and environmental variations. On the other hand, the side-channel analysis uses physical characteristics such as power [63], temperature [64], delay [65], and radiation [66] to detect the HT. Side-channel detection methods primarily rely on the availability of Trojan-free golden circuits, which may not be available in reality. Moreover, process and environmental variations may mask the side channel leakage, if the Trojan circuitry is small. Despite significant research performed on HT, we still lack methods for modeling and test generation to detect them.

In this chapter, we provided the background knowledge on logic locking, including the state-of-art techniques and existing attack methods of targeting the mentioned locking techniques. Additionally, the background of hardware Trojans is studied to understand how an adversary can implement malicious altercation to the original circuitry without the knowledge of the designer or user.

Chapter 3

A Design-for-Security Architecture to Prevent IP Piracy and IC Overproduction

In the last chapter, we observed that the existing logic locking technology cannot ensure complete safety of the circuit, especially when the SAT-based attack is a possibility. In this chapter, we will show how a proposed design-for-security architecture protects the circuit against IP piracy and IC overproduction.

3.1 The Related Work

The continuous trend of device scaling has enabled designers to fit more and more functionality on an SoC to reduce overall area and cost of a system. As the complexity of modern SoCs grows exponentially, it is virtually impossible to design a complete system by a SoC designer alone. Therefore, the semiconductor industry has shifted gears to the concept of design reuse rather than designing the whole SoC from scratch. In parallel, the increased complexity of the fabrication process has resulted in a majority of SoC designers to no longer maintain a fabrication unit (foundry or fab) of their own. Building and maintaining such fabs for modern SoCs are reported to cost more than several billions of dollars and increasing as technology further scales [7]. Given the increasing cost, the semiconductor business has largely shifted to a contract foundry business model (horizontal business model) over the past two decades. However, the lack of transparency and the resulting lack of trust may lead to the following vulnerabilities:

- *IC Overproduction.* An untrusted foundry/assembly can produce more number of unauthorized chips [1, 4, 6, 18, 19, 67–70], and can make illegitimately larger profits by selling them in the market as no R&D cost is incurred during production. Moreover, they can also practically overbuild chips at zero cost by manipulating the yield information [6, 71–73].

- *Out-of-Specification/Defective ICs from Manufacturing.* Due to the imperfect manufacturing and assembly processes, foundry/assembly discards defective chips and sends defect free chips to the market. In a trusted environment, these defective chips are always scrapped. However, an untrusted entity in the production process (a rogue employee) can source these rejected defective chips to the grey market [6]. The application of these chips in a critical infrastructure can cause significant damage.
- *IP Piracy and Reverse Engineering.* An untrusted foundry or its rogue employee can pirate the details of an SoC (e.g., test patterns, mask information, etc.) to a competitor company or make one or more illegitimate copies of the original IPs [70, 74–76]. The design details of an SoC can be reconstructed from the musk information by reverse engineering, which ultimately help to make cloned ICs [77, 78]. An untrusted foundry can also add some extra features to the SoC to introduce a backdoor or a hardware Trojan into these clone chips.

IC metering aims to prevent all the aforementioned attacks by attempting to give the control over the IC manufacturing to the SoC designer [1,4,6,18,19,67–69,79]. These approaches can be either passive or active. Passive approaches register all new authorized ICs by incorporating physically unclonable functions (PUFs) [80–84] in each copy and then storing their challenge-response pairs in a secure database. Later, any suspect ICs taken from the market can be checked for proper registration. Active metering approaches are designed to automatically lock each new IC that is manufactured by a foundry until it is unlocked (activated) by the authorized SoC designers. Active metering can be efficiently implemented through logic obfuscation. This is a technique where a design is transformed to a different one to obfuscate the inner details of the original design [1, 2, 4, 5, 76, 85]. Only on the application of a programmed secret key can make the transformation reversed, thus preserving the original functionality. Roy et al. first proposed to obfuscate a netlist by using a set of XOR/XNOR gates which can only be unlocked by using a key [1]. Unfortunately, this design is not resistant to reverse engineering (RE) as the key controlled gates are directly related to their key bits (XOR and XNOR gates indicate 0 and 1 at the key location, respectively) and vulnerable to key sensitization attacks [5].

The solutions to prevent key discovery proposed by Rajendran et al. [5] appear to adequately address the above issues. However, Subramanyan et al. have shown that the key in an activated

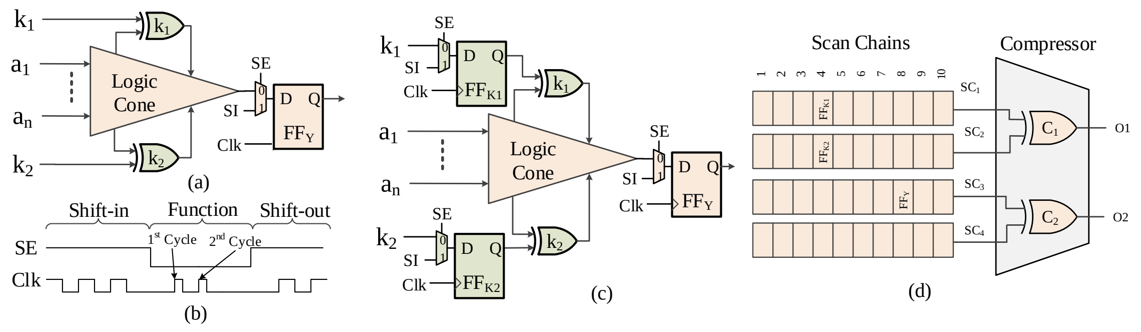


Figure 3.1: Prior obfuscation approaches and their vulnerabilities. (a) Techniques proposed in [1], [5]. (b) Timing diagram for manufacturing tests. (c) Technique proposed in [6]. (d) Attacks in [6].

circuit can always be exposed using scan based manufacturing tests through SAT-based analysis [20]. The SAT-based analysis algorithm [20] finds the correct key by ruling out incorrect ones iteratively, by using distinguishing input scan test patterns (DIP). For simplicity, the logic cone schematic is shown in Figure 3.1(a) is obfuscated by two key bits, k_1 and k_2. Here, a logic cone is a combinational logic unit that represents a Boolean function, and generally bordered by flip-flops and input/output ports. Assume that this cone produces different outputs for $k_1 = 0$ and $k_1 = 1$ for some input pattern $[a_1a_2 \ldots a_n]$. Then by observing the correct response from an activated working chip, the correct key ($k_1 = 0$ *or* 1) can be determined. Guin et al. [6, 73] proposed placing multiple flip-flops capturing signals controlled by different key bits (shown in Figure 3.1(d)) at the same level of the parallel scan chains used in current test compression methodologies [86, 87], thereby exploiting the output compression architecture to address SAT-based attacks. Figure 3.1(d) shows the architecture, where the keys (k_1 and k_2) are placed at the same level (location 4) in scan-chain 1 and 2 (SC_1 and SC_2). It appears impossible to perform SAT-based attacks that discover both k_1 and k_2, as an adversary cannot access individual scan cells from the compressed output O_1. One cannot determine the key bits k_1 and k_2, as they are equally likely in the key.

A vulnerability still remains with this design in view of advances in SAT-based formal tools that can support analysis over multiple sequential clock cycles. The key may be exposed to the adversary through multi-cycle tests, such as delay tests to detect transition delay faults and path delay faults [33]. During these tests, the circuit response is captured multiple times (typically 2 for timing tests), which is shown in Figure 3.1(b). In the first clock cycle, the key bits k_1 and k_2 are captured at FF_{k1} and FF_{k2}. Now, this key information is captured in the second clock cycle at

FF_Y (see Figure 3.1(c)) which can be located in a different scan chain (location 8 in SC_3). Thus, an adversary can perform a multi-cycle attack using a SAT-based approach. So even if the designer attempts to obscure the capture of key FF_{k1} and FF_{k2} at the end of the first cycle, an attacker can capture two or more clock cycles (FF_Y) to perform a SAT-based analysis.

A greater challenge occurs when the designer places the key gates uniformly in an SoC. This is often necessary to obfuscate the netlist to hide most of its functionality. An attacker does not necessarily perform the SAT-based attacks to extract the key when they are distributed throughout the netlist. An adversary can simply search the entire key space (brute-force) to find out the key. In Section 3.2.1, a brute-force attack will be demonstrated to find the key. However, brute-force attacks can be unfeasible when the keys are placed in larger cones (e.g., 128 inputs). An improved version of brute-force attack (greedy attack) can help an adversary to find the key by using a small number of random patterns (see Section 3.2.2 for details). Towards addressing these vulnerabilities, this project focuses on designing an obfuscated circuit such that it can withstand SAT-based, brute-force, and greedy attacks.

3.2 Attacks on Existing Logic Obfuscation Techniques

Modern electronic designs are sequential in nature and consist of combinational logic and memory elements. The outputs of a sequential circuit depend both on the inputs and its internal state. Generating test vectors to test a sequential circuit is extremely challenging as it is required to initialize the internal state before applying a pattern and then carry the response to the primary output [33]. This leads to adopt scan design, where controllability and observability are provided for the memory elements (flip-flops). The basic idea of scan is to convert the sequential circuit to its combinational equivalent. Each combinational block can be tested simultaneously through the scan chains. It is now very relevant to analyze the security of the obfuscated sequential circuits. In this section, two different attacks that can partially (fully) recover the obfuscation key for sequential circuits have been presented.

3.2.1 Brute-Force Attack Based on Logic Cones

For the uniform obfuscation of a netlist, it is required to distribute the key throughout the netlist such that the circuit produces incorrect result most of the time. This can create a new vulnerability that an adversary can estimate the key by using exhaustive search when a key gate is placed in a smaller cone. This kind of attack based on logic cones is named as a brute-force attack, which was first introduced by Lee et al. [3]. Brute-force attack is useful for evaluating the security strength of an obfuscated design.

Brute-force attacks can be performed through the scan chains, which are inserted in a design to provide manufacturing test support [33]. This converts a sequential design to its combinational equivalent and contains hundreds/thousands of cones with varying input sizes. If a key gate is placed in a cone with smaller number of inputs, an adversary can perform an exhaustive search to estimate the key value. In order to get a better understanding of brute-force attack, it is necessary to analyze attacker's effort (AE), which can be defined as the total number of trials to estimate the key. In this attack scenario, an adversary tries all possible combinations of key and input values of a cone, and observes the output of the locked circuit. For a correct key, the output must be equal to the output of that cone of an unlocked functional IC (oracle).

Let us assume a cone with n logic inputs, and m key inputs. Here, $X = \{x_1, x_2, \ldots, x_{2^n}\} \in \{\{0,1\}^n\}$ represents all inputs patterns, $K = \{k_1, k_2, \ldots, k_{2^m}\} \in \{\{0,1\}^m\}$ denotes all possible keys. Now, the input/output relations of the cone is represented by a function F, such that $Y = F(X)$. Similarly, for an obfuscated cone, it becomes $Y = F(X, K)$. For an unlocked circuit $F(x) = F(x, k_O)\ \forall x \in X$, where k_O is the obfuscation key. A brute-force attack verifies for every $k_j \in K$ if

$$F(x, k_O) \overset{?}{=} F(x, k_j)\ \forall x \in X \tag{3.1}$$

The hypothesis key, k_j becomes the obfuscation key, k_O if Equation 3.1 holds. Here, the attacker's effort (AE) becomes $O(2^{n+m})$ for a logic cone. Let us now study the case, where the keys are uniformly distributed across the design. The m-bit obfuscation key is distributed into r cones, where i^{th} cone receives m_i-bit key, and $\sum_{i=1}^{r}(m_i) = m$. The attacker's effort (AE_i) for cone i becomes $O(2^{n_i+m_i})$. The overall attacker's effort will be $AE = max(AE_i)$ as all the cones

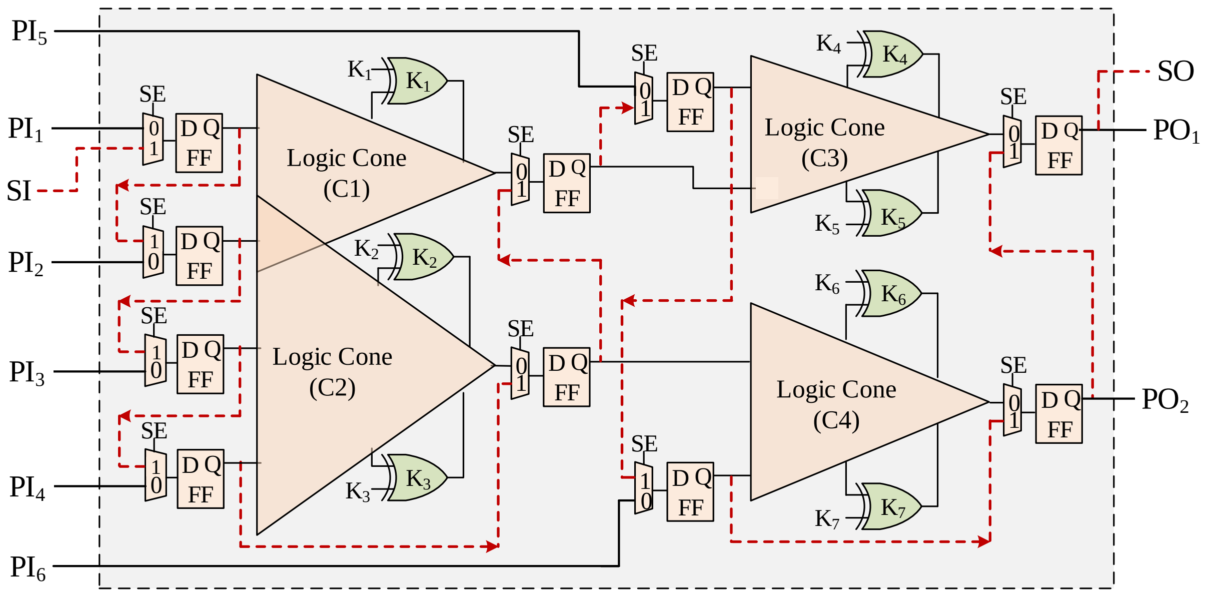


Figure 3.2: An example of a scan inserted sequential circuit.

can be tested simultaneously through the scan chains (see details in McCluskey's verification test paper [88]).

A short example will be presented to illustrate the complexity of this attack. Figure 3.2 shows a sequential circuit, where 7 key gates are placed. It is assumed that the circuit contains four logic cones, namely, C1, C2, C3, and C4, where C1 and C2 have one overlapping input. The circuit has 6 inputs and 2 outputs. For simplicity, assume that it contains one scan chain (highlighted with red broken line). To find the correct key, an adversary will try all possible combinations. Thus, the attacker's effort for C1 (AE_1) will be 2^3. Similarly, AE for cones C2, C3, and C4 will be $AE_2 = 2^5$, $AE_3 = 2^4$, and $AE_4 = 2^4$, respectively. It is interesting to note that an adversary can apply brute-force to all the cones simultaneously by shifting the appropriate patterns through the scan chain. The number of such scan shift operations (the overall attacker's effort) is the $max(A_i) = 2^5$, which is much smaller than the exhaustive key search (2^{6+7}) to find 7-bit obfuscation key. However, an adversary can find some key bits much quickly if they are placed in a smaller cone (e.g., C1).

In summary, an adversary can perform brute-force attacks on all cones simultaneously through scan chains to estimate the complete m-bit key. However, he/she can find a part of key if those keys are placed in a small cone. *The strength of the obfuscation depends only on the cone size, rather than the total number of bits in the obfuscation key and the primary inputs of a complete*

Table 3.1: Percentage of cones (PC) in IWLS benchmarks.

Bench-mark	**# Gates**	**PC ≤ 16**	**16<PC ≤ 32**	**32<PC ≤ 64**	**64<PC ≤ 128**	**PC >128**
S35932	16,065	100.00%	0.00%	0.00%	0.00%	0.00%
S38584	19,253	78.49%	19.88%	1.64%	0.00%	0.00%
S38417	22.179	58.33%	16.25%	9.42%	16.00%	0.00%
b17	37,117	11.06%	4.36%	11.76%	22.43%	50.39%
b18	92,048	6.97%	5.04%	12.91%	14.60%	60.47%
b19	174,157	6.80%	5.14%	12.59%	14.55%	60.92%

netlist. It is thus necessary for a system designer to place all the keys in cones with sufficiently large number of inputs, such that a brute-force attack is infeasible.

The simulations were performed against IWLS benchmarks [89] to analyze the number of cones that can be targeted for brute-force attacks. Table 3.1 shows the cone analysis for six different benchmarks. Based on this table, it can be found that the cone size in the netlist varies from a few inputs up to hundreds of inputs. For a small benchmark (e.g., S35932) all cones have less than 16 inputs. For benchmark S38584, percentage of cones (PC) with less than 16 inputs is 78.49% and PC with less than 64 is 100%. For these smaller benchmarks, an adversary can simulate all input and key combinations to find out the obfuscation key. As mentioned before the objective of an SoC designer is to place the key gates uniformly to have a higher obfuscation impact. Each cone may have very few key gates. For larger benchmarks (e.g., $b19$), PC with less that 16 inputs is 6.8%, whereas PC with less than 32 inputs, and greater than 16 inputs is 5.14%. Thus, an adversary can find few key bits if the keys are uniformly distributed across the circuit. However, an SoC designer can place the keys in larger cones to prevent the attack.

3.2.2 Greedy Attacks on Logic Cones

Brute-force is an efficient approach to obtain the key value especially when the cone size is small. However, when the size of the cone becomes larger, the brute-force attack may not be feasible as the attacker's effort remains exponential complexity with the number of inputs. In this section, a new attack is proposed that can greatly reduces AE for a circuit. This method of attack is referred as *greedy attack*.

Instead of applying every input combinations, an adversary greedily selects a few patterns to recover the secret key.

In greedy attack, an adversary simulates a cone with few random patterns. Then the same patterns are applied to the same cone of an unlocked chip (oracle) to receive the correct response. If the comparison fails, it is guaranteed that the hypothesis key used during simulation is not the obfuscation key. The greedy attacks iterates all possible key combinations to rule out all hypothesis keys. Note that number of key bits are very small for uniform obfuscation. This is a probabilistic attack and it cannot guarantee to find the obfuscation key. However, the experimental results show that a hypothesis key with few random patterns can be ruled out in most cases.

Greedy attack: The hypothesis key, k_j is not the obfuscation key if:

$$\exists x \in X^P : F(x, k_j) \neq F(x, k_O) \tag{3.2}$$

X^P is the set of p randomly selected patterns. The complexity of greedy attack is $O(p \times 2^{m_i}) \approx O(p)$, where m_i (can be very small, e.g., 1) is the key size of the i^{th} cone. This attack is feasible when a designer uniformly distributes the keys in their design to have a greater impact of obfuscation.

To validate this attack, the experiment is performed by using Synopsys Design Compiler [90], and VCS [91] on few IWLS 2005 [89] benchmarks. A wrong key of different sizes (1-bit, 2-bit, 4-bit, and 8-bit) can be found by using only 10k random input patterns for a small cone size (see Table 3.2) and 200k random patterns for large cones (see Table 3.3). A Perl script extracts few cones from benchmarks $b17$, $b18$ and $b19$, and uses VCS to perform the simulation.

Table 3.2 shows the simulation results for few small cones (less than 128 bits) from the ITC'99 benchmarks. Ten thousand random patterns have been applied to observe the responses. Six different cones (C1-C6) are randomly selected to perform the greedy attack. Column 1 represents

Table 3.2: Greedy attacks on small cones.

Key Size	Cone Size	The Number of Get 1 in 10000 Patterns					
		C1	C2	C3	C4	C5	C6
1	44-56	208	11	272	2031	898	87
	94-106	101	57	1742	3	28	15
2	44-56	886	321	461	1760	329	98
	94-106	633	391	29	65	182	59
4	44-56	2715	327	402	911	946	5
	94-106	236	417	62	106	31	169
8	44-56	1958	293	2162	414	1391	1269
	94-106	520	377	3354	378	64	292

Table 3.3: Greedy attacks on large cones.

Key Size	Cone Size	The Number of Getting 1 in 200000 Patterns					
		C1	C2	C3	C4	C5	C6
1	144-156	2	3	412	3	3	6
	194-206	6	14	19	7096	7	9
2	144-156	5	3	5	25	1	6
	194-206	2	1686	6	131	17	13
4	144-156	13	2	6	12623	5	3
	194-206	2	4	11	30	221	906
8	144-156	13	4	1511	11	3	13
	194-206	7	632	12	2726	1573	3408

the number of key gates that are placed in these cones. Column 2 represents cone size. A cone is selected as mentioned in the range. Rest of the columns show the number of times Equation 3.2 are satisfied. Form the table, it is clear that almost all the cones produce incorrect results most of the cases.

Table 3.3 shows the greedy attack on large cones. The larger cones require more random patterns to find a mismatch that satisfies Equation 3.2. When 200k random patterns are applied it generally takes less than a minute to apply all these patterns to perform the attack, assuming the simulation is performed in HP Z840 Workstation with Intel® Xeon® E5-2620 v3 (2.4 GHz/6 cores) processor and 64 GB of RAM. In majority of the cases, the adversary finds an incorrect key effectively in few minutes.

In summary, existing logic obfuscation techniques suffer from three different - brute-force, greedy and SAT-based - attacks. Our objective is to design an obfuscation technique that can effectively circumvent all these attacks. Alternatively, one can state that they require a design solution that prevents access to the response of a logic cone through scan chains. Without an oracle, an adversary cannot compare the simulation results with the oracle and perform such attacks.

3.3 Description of the Proposed Design-for-Security (DFS) Implementation

3.3.1 Requirements of DFS Implementation

This section provides an in-depth analysis for all the requirements for successfully preventing IC overproduction, manufacturing rejection, and IP piracy.

Attack resistance. The netlist must be designed in such a way that the chip never leaks the key (either during tests or normal functions), which makes the design resistant to various known attacks [5,20,28,92,93]. Finding of a key must satisfy NP completeness, and the key must be kept long enough such that brute-force attacks become impractical. In addition, the key must be resistant to reverse engineering (RE) attack, where an attacker must not find the key by looking at the circuit netlist. Direct mapping of the key bits to XOR or XNOR gates are prohibited.

Uniform distribution of the key. The key gates need to be placed uniformly to a design to obfuscate its significant part. As the modern designs are sequential in nature, care needs to be taken to place a key gate. It can be subjected to brute-force attacks (see Section 3.2.1). It can also be vulnerable to greedy attacks (see Section 3.2.2) irrespective of the size of the cone. In addition, any cones are subjected to SAT-based attacks. The obfuscation scheme must address all these attacks.

Structural test capability without the key. Allowing structural tests before the activation is one of the key requirements for preventing the overproduction of chips. It is necessary to add capability which permit a foundry or assembly to perform structural tests right after manufacturing and discard the defective chips. One can argue that tests can be performed at the SoC designer's site. However, it requires additional test setup for the SoC designers, which they may not have. In addition, it is not wise to send chips to the SoC designers without tests which requires addition transportation. However, the greater challenge is that the foundry cannot stabilize the process unless they monitor the outcome. Thus, it is absolutely required that the tests have to be performed at the manufacturing site.

Post-silicon validation and debug capability. The circuits must be modified in such a way that it does not impact the post-silicon validation and debug, where the chips generally run at-speed and scan-dumps may be required to obtain high observability of internal nodes.

Full in-system test capability. The obfuscated circuit must support in-system test capability. It is absolutely necessary that a chip does not leak key information to its primary outputs (POs) while it is in functional mode. In this mode, a set of functional test vectors is required to test a design. While testing it is required that each module (IPs) to be initialized to the desired state. Setting that state of a complex industrial circuit through primary inputs becomes a major challenge and could potentially take millions of clock cycles [94]. Thus, test engineers often shift the state through existing design-for-test (DFT) structure [95]. It is thus required that keys do not impose any limitation to this hybrid testing.

3.3.2 Proposed Design-for-Security (DFS) Architecture

The objective in designing the new DFS architecture is to prevent the key getting exposed during manufacturing tests. As mentioned in Section 3.3.1 that if the key information is captured during a test, it will eventually be exposed to the primary outputs of a working (unlocked) chip and an adversary can effectively retrieve the key.

Figure 3.3 shows the proposed secure cell (SC) architecture used for design-for-security. The scan cell is modified in such a way that it can hold its previous state. The output of FF_k is fed back to the its input through a multiplexer (MUX). As the MUX has four inputs, One additional *Test* pin is needed for the MUX control. Depending on the value of $Test$ and SE pins, a particular

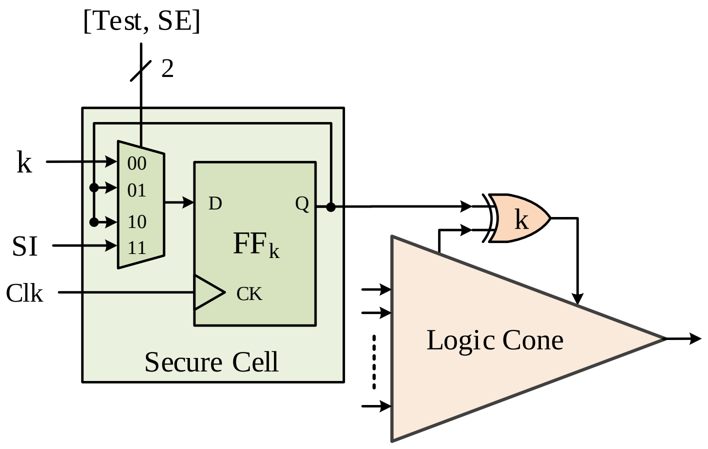


Figure 3.3: Proposed secure cell architecture.

Table 3.4: Modes of operation.

Test	**SE**	**Mode**	**Description**
0	0	M0	The chip is in functional mode. The secure cell applies key to the logic.
0	1	M1	The secure cell holds its previous value. The rest of the circuit is in functional/shift mode depending on the *SE*.
1	0		
1	1	M2	The SC becomes scan cell and it becomes a part of the scan chain.

input is selected. The key bit (k), and scan in (SI) are connected to the first and the fourth inputs of the MUX, respectively. The output of FF_k is connected to the second and third inputs, which provides the capability to hold its previous state.

The SC operates in three different modes based on $Test$ and SE, which is shown in Table 3.4. In mode $M0$, FF_k captures the key k, which represents the normal functionality of the unlocked chip. The chip will be operated in this mode while it is in the field. In mode $M1$, the secure cell continues to hold its previous state. This mode provides test and debug capability without letting the key to be exposed as FF_k continues to hold its previous state. Thus, no key information is captured in $M1$. Note that the rest of the circuit becomes functional mode when $SE = 0$ and scan mode (shift-in or shift-out) when $SE = 1$. Finally, SC becomes the scan cell at mode $M2$ and FF_k becomes a part of a scan chain.

Manufacturing test. The implementation of manufacturing tests using our proposed secure cell does not require any additional modifications in the existing test infrastructure. Note that the key is

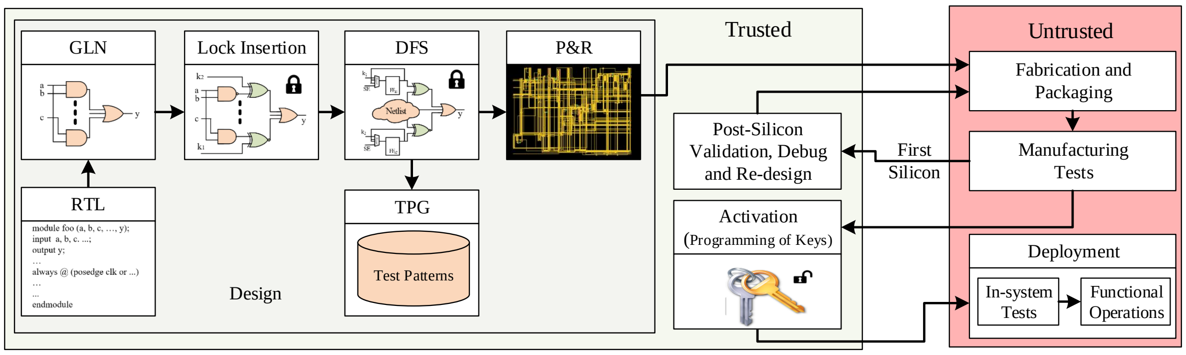


Figure 3.4: Proposed flow for enabling trust in IC manufacturing and test.

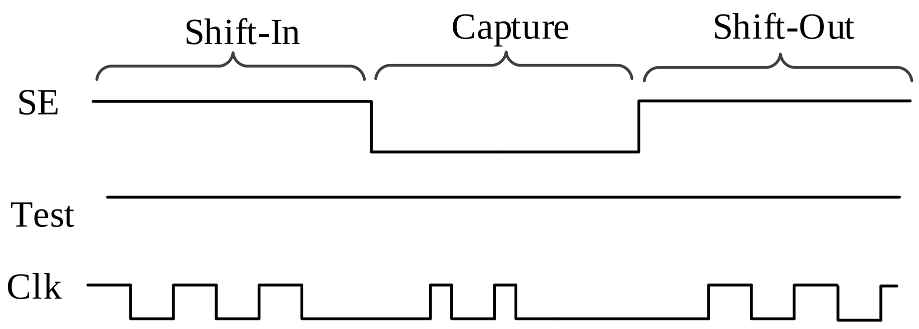


Figure 3.5: Timing diagram for manufacturing tests (delay tests).

not programmed at this stage (see Figure 3.4). It is required to keep $Test$ pin active high (logic 1) during the test. During the scan shift-in phase, the secure cells (SCs) become a part of a scan chain ($\{Test, SE\} = \{1, 1\} = M2$) and receive values generated by the ATPG tool. As the key gate (k) (see Figure 3.3) is directly reachable from the SI.

As shown in the timing diagram of Figure 3.5, during the test response capture, the rest of the circuit becomes functional while the SCs hold their current state (($\{Test, SE\} = \{1, 0\} = M1$). No key bits are captured in the SCs as they continue to hold the states received during scan shift-in phase. This helps us to eliminate all the attacks completely. Finally, the captured functional responses are shifted out through the scan chain ($\{Test, SE\} = \{1, 1\} = M2$).

Post-silicon debug and validation. Complex modern designs can suffer from subtle logic and electrical design bugs that escape design verification and are only discovered in first silicon. This necessitates support for post silicon validation, and if a bug is discovered, its diagnosis followed by design changes to correct the problem. Post silicon debug is extremely challenging, and at a minimum requires both a fully functional test (on the activated design) as well as extensive scan test support. This extent of intrusive testing of the fully functional circuit can make it vulnerable to key discovery through SAT-based attacks or other formal tools. Therefore such full testing is only allowed in a secure design environment, with the key again applied through the scan chain, even if it is already programmed.

Full scan tests on the fully functional circuit are performed in mode $M1$ (Table 3.4). Recall that in this mode, with the scan enable low (functional mode), the programmed key bits are not captured in the secure cells from where they are presented to the logic; instead, the SCs are designed to hold and retain their current value. Thus, if the key bits are shifted into the SCs during the scan shift in $M2$, and the scan enable (SE) is then lowered to the functional mode ($M1$), the scanned in key bits will be retained in the SCs ensuring unlocked functional operation as long as the scan enable stays low. Single or multi-cycle tests can be performed and the results shifted out ($M2$).

Functional Tests. The functional test can only be performed after the activation of the chips. Mode $M0$ supports functional tests. Functional patterns are applied to the primary inputs (PIs) of a chip and the responses are collected at the primary outputs (POs). It is required to initialize the finite state machine of a design before actual tests are performed, and sometimes could lead to millions of clock cycles [94]. Test engineers often shift this initialization state through existing scan architecture. Mode $M2$ can be used to shift this state to the design and then it is switched to mode $M0$.

Mode control. An important restriction on switching between different operation modes for the SC is absolutely necessary for maintaining security. ***Switching from*** $M0$ ***to*** $M2$ ***(***$M0 \rightarrow M2$ ***or*** $M0 \rightarrow M1(Test = 1) \rightarrow M2$***) cannot be permissible. To be specific, any positive transition at the*** $Test$ ***pin will not be permitted***. The key will be captured in $M0$ and be shifted out while the cell is in $M2$, if this is allowed to happen. In addition, the shift out will not allow when $Test$ is not asserted (i.e., $Test = 0$), which will prevent an adversary getting scan data (from SC to end of the scan chain may be shifted out while setting $SE = 1$) during $Test = 0$.

Figure 3.6 shows the proposed architecture to restrict scan data access. A series of OR gates have been added at the output of the compressor (Test data compression is widely accepted by the industry [86, 87]), which is highlighted in green. The output of the test suppressor (TS) block

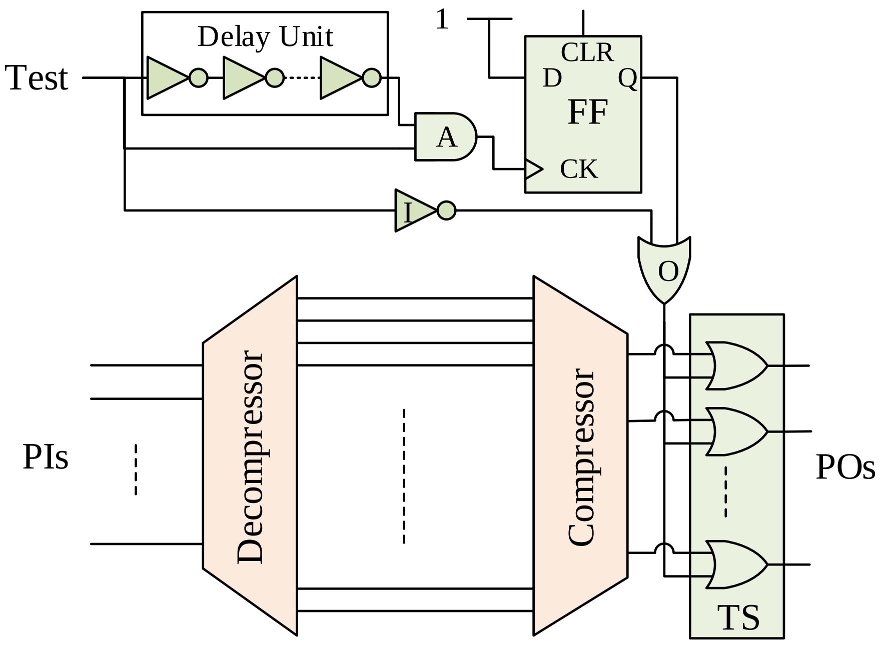


Figure 3.6: Proposed architecture to restrict scan data access.

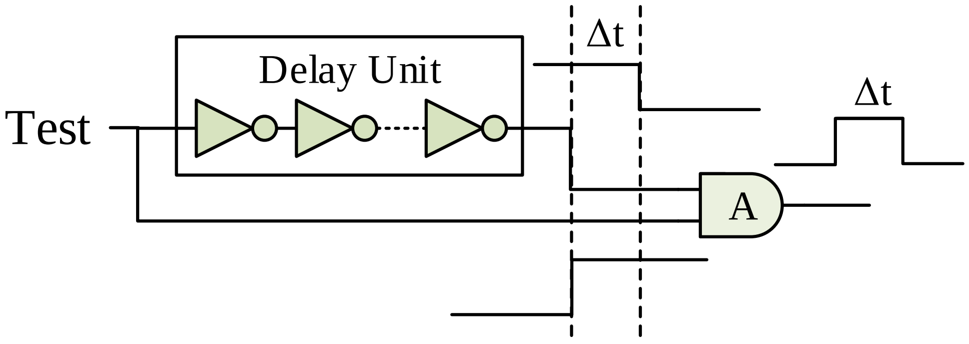


Figure 3.7: Pulse generator module for detecting a positive transition at $Test$ pin.

becomes always 1 when the output of the OR gate (denoted by O) is asserted. One can place TS block before the compressor, however, the number of OR gates will be increased significantly.

The output of the OR gate O becomes 1 when one or both inputs become 1. This ensures that an adversary cannot access scan data while $Test = 0$, which is one of the requirements for protecting the key. Now, there is a need to make sure that there is no positive transition on the $Test$ pin.

Figure 3.7 shows a pulse generator, when it experiences a positive transition of the $Test$ pin. The delay unit consists of odd number of inverters, and is fed to an AND gate, A. A pulse with duration Δt is generated at the output of gate A. The width of this pulse, Δt can be controlled by manipulating the number of inverters.

The output of the AND gate, A is fed to the clock input of the flip-flop (FF) shown in Figure 3.6. When the FF detects a pulse, logic 1 will be captured and its output becomes 1 permanently. This flip-flop can be cleared once during power up or after certain clock cycles (length of the scan chain) depending on one's choice. It is worthwhile to mention here that the $Test$ pin can also be fed to the clock input of FF.

Secure cell placement. Higher fault coverage is often required to ensure the high yield of good chips [33]. In a circuit, there are many untestable faults due to the controllability and/or observability issues. Test point insertion is widely used to detect many of these untestable faults and thus increase the fault coverage of a circuit. Our proposed secure cell can be used as a test point. Thus, the objective of placing a secure cell (e.g., one key bit) in the netlist in such a way that it provides the detection capability of untestable faults.

Algorithm 1 determines the key location such that fault coverage can be increased by sorting the nets (fault locations) based on the number of faults present in them. The two scenarios may arise. First, the number of such nets (L) is greater than key size, $|K|$ (lines 5-9). This may arise when the design has many untestable faults and there are enough nets to place the key gates. Second, the number of such nets, L is less than key size, $|K|$ (lines 11-18). L key gates are placed first (lines 11-14) and the remaining key gates are placed randomly (lines 15-18). Note that the secure cell introduces few new faults to the design that can reduce the overall fault coverage.

3.4 Proposed Flow for Enabling Trust in IC Manufacturing and Test

The primary requirement for preventing IC overproduction and IP piracy is to obfuscate a design with a key which is resistant to all known lines of attack. The design must support all the requirements (mentioned in Section 3.3.1) for the obfuscation key. The manufacturing tests can be performed at the foundry and/or package assembly as these tests do not require any key. It is also important to implement manufacturing tests before the activation of chips as an untrusted foundry can manipulate the yield information (the ratio of the defect free chips to the total number of chips) with the SoC designer and stockpile a large number of chips without contributing any design costs. In addition, it can source defective or out-of-specification chips to the market if the

Algorithm 1: Place all secure cells in a circuit.

```
input  : Scan inserted netlist, key size ($|K|$)
output : Obfuscated and locked netlist

1  Read scan inserted netlist ;
2  Fault simulation to find all untestable faults ;
3  $UF \leftarrow$ Sort the nets based on the number of untestable faults in the descending order ;
4  $L \leftarrow UF.size()$ ;
5  if $N \geq |K|$ then
6      for $i \leftarrow 0$ to $|K-1|$ do
7          Insert a key gate at $net_i$ ;
8          Place a secure cell ;
9      end
10 else
11     for $i \leftarrow 0$ to $|L-1|$ do
12         Insert a key gate at $net_i$ ;
13         Place a secure cell ;
14     end
15     for $i \leftarrow |L|$ to $|K-1|$ do
16         Insert a key gate at the randomly location of the netlist ;
17         Place a secure cell ;
18     end
19 end
```

chips are activated before the manufacturing tests. Thus, it is necessary to activate the chips in a trusted environment such that an untrusted entity cannot get any undue advantages. In summary, manufacturing tests can be performed at any untrusted site, however, activation must be executed at a trusted site. It is also important to note that post-silicon validation and debug require the chip to be fully functional (activated) with full structural test capability enabled.

Figure 3.4 shows the overview of our proposed flow for enabling trust in IC manufacturing and test. This flow is exactly the same as existing IC design and fabrication process, except for the Lock Insertion, DFS, and Activation stages. The process starts with the RTL design phase and then it goes through synthesis to obtain gate level netlist (GLN). A set of key gates are now inserted to lock the GLN, which can only be unlocked through a proper key. It is recommended to use an existing RE-resistant lock insertion technique [5] such that an adversary cannot find the key by simply observing the key gates. Note that the left half of Figure 3.4 highlighted in green (design phase, activation and post-silicon validation and debug) is under designer's control and is trusted. On the other hand, the right half of the figure highlighted in red (fabrication and packaging, manufacturing tests, and deployment) are untrusted.

Even if the keys are resistant to RE, an adversary can still discover the key by using brute-force, greedy and SAT attacks. To avoid the key being exposed to these attacks, the insertion of novel secure scan cells (see Section 3.3.2 for details) to the key gates is proposed. This makes the keys resistant to known manufacturing test related attacks. As all the key gates are reachable through these secure cells, it is not required to provide key information to the automatic test pattern generation (ATPG) tool for generating test patterns. It is worthwhile to mention that the keys now satisfy all the requirements mentioned in Section 3.3.1. After DFS stage, the design is moved to the place and route (P&R) stage, and then Graphic Database System II (GDSII) files are created. Finally, they are sent to foundry for fabrication and packaging.

After manufacturing the first batch of chips, the foundry performs manufacturing tests and sends to the SoC designer for post-silicon validation and debug, where it validates correct behaviour in actual application environments. Any bugs may have been undetected previously during pre-silicon verification. During this stage, the SoC designer performs many different tests (combination of structural and functional) and observes the internal states to detect and diagnose any bugs. The proposed DFS architecture provides the post-silicon validation and debug support which is absolutely required for SoC design and fabrication process.

Once the post-silicon validation and debug is complete, the SoC designer provides the contract to a foundry to fabricate a certain number of chips. After fabrication, the foundry performs manufacturing tests and sends the defect free dies to the assembly for packaging. The assembly performs final tests and sends back the chips to the SoC designer for activation. Finally, SoC designer activates the chips and sends them to the market for deployment. In-system functional tests can be performed on these activated chips in the field to test for their correct functionality.

Note that the left half of Figure 3.4 highlighted in green (design phase, activation and post-silicon validation and debug) is under designer's control and is trusted. On the other hand, the right half of the figure highlighted in red (fabrication and packaging, manufacturing tests, and deployment) are untrusted and the keys may get compromised due to various attacks (SAT, RE, etc.). As the key information is not leaked (see Section 3.3.2) for the proposed design during any tests, it can be safely said that these attacks are ineffective in extracting the key.

3.5 Results and Analysis

First, a security analysis of the proposed scheme is provided, and then we discuss the simulation results.

3.5.1 Security Analysis

Ensuring security by protecting the key being exposed to an adversary is our prime objective. In this section, various known attacks will be examined for evaluation of their impact on security.

Attack Resistance

All types of attacks (e.g., brute-force, greedy and SAT-based) are primarily based on the actual observation of the response of a logic cone through the scan chains of a circuit. As long as the key information is captured during function mode and then dumping the responses through scan chains, the key will be exposed to the aforementioned attacks. Our proposed design is resistant to these attacks as the SC prevents an adversary to capture key information while testing. SC is designed in such a way that it holds it's previous state when the chip experiences tests. In addition, the restrictions for mode switching (mode $M0$ to mode $M2$) to access scan data is imposed. An adversary cannot extract functional response through the scan-chains. He can only observe all 1s, when he tries to dump the scan data which contains the key information.

The security of our proposed approach lies on the length of the key. A key must be long enough such that it can withstand exhaustive key search, as our proposed design is resistant to brute-force, greedy and SAT-based attacks, and maintains NP-completeness. As no key information is captured during the test, an attacker must try $2^{|K|}$ combinations to make the circuit completely functional in worst case. Here, $|K|$ is the length of the key. It is computationally unfeasible to find a correct key when $|K|$ is greater than 128 considering current computing resources. However, one can use 256 or higher bit keys for obfuscating a netlist considering future computing resources.

Tampering

An untrusted foundry can modify the masks to bypass the mode control logic (see Figure 3.6) and write a permanent ”zero” value at the output of the OR gate, O. In this case, an adversary has the full control of changing the modes ($M0$ to $M2$) and perform SAT-based attacks to find the key. Fortunately, this attack can easily be detected by the SoC designers and can be prevented. If the foundry manufacture chips with the tampered masks and send chips to the SoC designer for activation, he can easily detect the tampering by switching the modes and observe data. The scan data will be all 1s if it not tampered.

Now an untrusted foundry can maintain two (one tampered and one genuine) sets of masks, and send chips to the SoC designer those are manufactured with genuine masks. For the worst case, it can send one tampered chip (fabricated with the tampered masks) along all genuine chips hoping that the SoC designer will burn the key and thus can get hold of the scan data (key) from this working chip (bypassed our security measures). To circumvent this attack, the SoC designer needs to verify the chip before activating. It is important to note that the reputation of a foundry will be demolished if the SoC designer detects tampering. Moreover, it is extremely expensive to design a new set of masks, and there is little economic incentive for an untrusted foundry to maintain two different sets of masks.

3.5.2 Area Overhead Analysis

The area overhead for our proposed approach are primarily resulted from four parts:

- *Secure Cell module*: This secure cell contents two parts: a 4 to 1 multiplexer and a scan flip-flop. The secure cell can switch among three modes: function mode, hold mode and scan mode. Based on our proposed structure, this will not disclose the key during any time. For a single secure cell

Table 3.5: Test Metrics Comparison - Test Coverage.

Benchmark	**Key Bits**	**Test Coverage**			
		ORG	**KEY**	**DFS**	**Change**(%)
s35932	128	100.00%	100.00%	100.00%	0.00%
s38584	128	100.00%	100.00%	100.00%	0.00%
s38417	128	100.00%	100.00%	100.00%	0.00%
b17	128	99.92%	99.91%	99.80%	-0.11%
b18	128	99.54%	99.60%	99.58%	-0.02%
b19	128	99.65%	99.65%	99.64%	-0.01%

(a 4 to 1 multiplexer and a scan flip-flop), it usually contents 20 gates. The number of SCs is equal to the key length $|K|$, as each key bit is fed to a different SC. It requires 256 (128) SCs when $|K|$ is 256 (128) to maintain long term security. The approximate gate count for an SC is around 20.

- *Keys gates*: The size due to keys also depends on the length of chip unlock key (CUK). To implement one key bit, one XOR/XNOR gate is required.
- *Test Suppression*: The number of OR gates is equal to the compressor output. It can be safely assumed that 100 OR gates for Test Suppression is require.
- *Mode Control*: It takes approximately 20 gates to implement this module.

From the above analysis, the majority of the overhead results from the secure cells (number of key bits). The total gate count for the proposed approach is approximately 5,200 when considering 256 bit key. This can be reduced significantly to 2,700 when the key is 128 bit long. For a large benchmark (e.g., $b19$) the area overhead is less the 1% (see column 7 of Table 3.6 in the next subsection). However, it can be even smaller ($\ll 1\%$) for a modern industrial design with millions of gates. *Note that one additional pin ($Test$) is needed to provide the DFS support.*

3.5.3 Simulation Results

To evaluate the effectiveness of the proposed DFS architecture, Synopsys tools (Design Compiler [90], TetraMax [96] and VCS [91]) are used to perform the simulation by using Synopsys

Table 3.6: Test Metrics Comparison - Pattern Count.

Benchmark	**Key Bits**	**Pattern Count**				**Area Overhead**
		ORG	**KEY**	**DFS**	**Change(%)**	
s35932	128	56	55	65	18.18%	6.14%
s38584	128	536	549	565	2.91%	7.84%
s38417	128	1,133	1,124	1,115	-0.80%	6.75%
b17	128	2,542	2,516	2,559	1.71%	4.23%
b18	128	5,086	5,112	5,116	0.08%	1.51%
b19	128	9,395	9,387	9,398	0.12%	0.80%

32nm SAED32 EDK Generic Library [97] on IWLS 2005 [89] benchmark circuits. Table 3.5 and Table 3.6 show the test metrics comparison between different methods. By comparing the test coverage and pattern counts among the original netlist with no locks (denoted as ORG), key gate inserted netlist (KEY) and the proposed DFS inserted netlist (DFS). In both tables, column 2 shows the size of the obfuscation key, which represents the number of key gates to be insert into the netlist. In Table 3.5, columns 3-5 represent the test coverage. The test coverage for KEY is computed by applying a preset key (all 0's) during test pattern generation. On the other hand, we do not need any key information for DFS. In Table 3.5, column 6 represents the percentage change of the test coverage from KEY to DFS. No significant change in the test coverage is expected. However, the proposed method causes a small reduction for some benchmarks. This can be due to the added faults from the multiplexer of the DFS architecture. Similar analysis can be performed for the test pattern counts (shown in columns 3-5 of Table 3.6). A minor increase in the pattern counts for moderate to large size benchmarks is observed.

3.6 Summary

In this part, a novel secure cell (SC) design is proposed for implementing design-for-security (DFS) infrastructure that successfully prevents the leaking of obfuscation key to an adversary, and thus establishes trust in semiconductor manufacturing. First, our proposed SC disables scan dump after functional mode. This provides a complete protection against all different attacks that require an oracle (an unlocked functional chip) to compare the simulation responses. As scan dump is not allowed during normal operations using scan chains, an adversary can not have access the functional responses of an unlocked chip. Second, our proposed SC enables manufacturing tests before the activation of chips, which is absolutely necessary to prevent unauthorized overproduction of chips. This allows the fully scan based tests at the foundry on the obfuscated design. Finally, the DFS architecture provides complete support for post-Si validation and debug to ensure perfect compliance to its specifications by correcting subtle logic and electrical design bugs that escape design verification and are only discovered in first silicon. In summary, our proposed solution provides a secure metering and obfuscation technique without modifying the existing IC manufacturing and test flow with the cost of a small area overhead. However, one additional pin ($Test$) is required to provide DFS support.

The proposed structure can provide robust security against state-of-art SAT-based attacks. However, even if the structures are SAT resilient, the design may still not provide absolute security. In the next chapter, a novel attack method that can break any logic locking technique that relies on the stored secret key will be introduced [98].

Chapter 4

TAAL: Tampering Attack on Any Key-Based Logic Locked Circuits

Continuous addition of new functionality in a system-on-a-chip (SoC) has enforced designers to adopt newer and lower technology nodes to manufacture chips primarily to reduce the overall area and the resultant cost of a chip. Building and maintaining such a fabrication plant (foundry) requires a multi-billion dollar investment [7]. As a result, the semiconductor industry has moved towards horizontal integration, where an SoC designer acquires intellectual properties (IPs) from many different vendors and sends the design to a foundry for manufacturing, which is generally located offshore.

The hardware layers that were assumed to be trusted are no longer true with the outsourcing of IC fabrication in a globalized and distributed design flow, including multiple entities. Third-party IPs, fabrication, and test facilities of chips represent security threats to the current horizontal integration of the production. The security threats posed by these entities include – (i) overproduction of ICs [1,4,6,18,19,68,69], where an untrusted foundry fabricates more chips without the consent of the SoC designer in order to generate revenue by selling them in the market, (ii) sale of out-of-specification/rejected ICs [6,28,71,99,100], and (iii) IP piracy [10,74,100–102], where an entity in the supply chain can use, modify and/or sell functional IPs illegally. Over the years, researchers have proposed different techniques to prevent the aforementioned attacks and they are IC metering [1,18,67,68], logic locking [1,5,6], hardware watermarking [75,103,104], and split manufacturing [105,106].

Logic locking has emerged as a promising technique and gained significant attention from the researchers to address the different threats emerging from untrusted manufacturing. In logic locking, the netlist of a circuit is locked so that it produces incorrect results unless it is programmed with a secret key. The locks are generally inserted in the netlist using XOR gates. Traditional logic locking methods involved selection of key gate location as random selection [1, 28, 29], strong interference-based selection [5], and fault-analysis based selection [107]. Over the years, researchers have also proposed different attacks to extract the secret key and undermine the locking mechanisms. Boolean Satisfiability (SAT)-based attacks have demonstrated effective ways of extracting the secret keys. Countermeasures are also proposed so that SAT-based attacks become infeasible.

This chapter shows that any locked circuit, where the secret key is stored in an on-chip memory no matter whether it is tamper-proof or not, can be broken by inserting a hardware Trojan. The attack is presented as **TAAL**: **T**ampering **A**ttack on **A**ny **L**ocked circuit that uses a stored key for locking the netlist. This attack can defeat the security measures provided from any existing logic locking methods. *We believe that we are the first to show that any key-based logic locked circuit can be exploited by inserting a hardware Trojan in the netlist*. The contributions of this chapter are described as follows:

- A novel attack is proposed based on malicious modifications of the netlist to target any key-based locked circuit. The attacking approach is to tamper the locked netlist in order to extract the secret key information. Once the valid key information is extracted from an activated IC, an untrusted foundry can unlock any number of chips and sell overproduced and defective chips. As this attack applies to any key-based locked circuits, an adversary can undermine any secure solutions proposed so far to prevent overproduction, sourcing defective/out-of-spec chips, and IP piracy. Note that different researchers have proposed to use logic locking to prevent hardware Trojans [32, 108–111] as an adversary cannot precisely specify the trigger conditions when a design is locked. However, a Trojan is exploited to obtain the secret key, which is the primary contribution of this chapter.
- Three types of TAAL attacks that extract the secret key differently using hardware Trojans placed at different locations in the netlist are presented. *T1 type TAAL* attack directly leaks the secret key to the primary output of an IC once the Trojan is activated (see Figure 4.1(a)). On the other hand, *T2 type TAAL* and *T3 type TAAL* attacks rely on the activation and propagation of the secret key to the primary output (see Figure 4.1(b) and Figure 4.2, respectively). Note that an adversary has the freedom of choosing one of these three types of TAAL attacks implemented using combinational, sequential, or analog hardware Trojans.

- An existing well-known model for designing a combinational hardware Trojan [112], which can be used to launch the TAAL attacks is presented. An adversary has the freedom to choose the type of hardware Trojan, which can be designed so that it evades manufacturing or production tests and remains undetected. These combinational Trojans can be described as *Type-p* Trojans, as they have p trigger inputs. These triggers can come from the primary inputs and/or internal nodes of a locked circuit, which are not affected by the keys. Research has shown that a very large number of Trojans can be created, and an adversary can use only one such a Trojan. It is practically impossible for an SoC designer to detect all these feasible Trojans using logic tests.
- A model for sequential Trojan is also presented, which is constructed using a combinational one. A state element (a counter) is added to a combinational Trojan to deliver the payload once it is triggered R times consecutively. Note that the trigger inputs for both the Trojans are the same. The combinational Trojan delivers the payload once triggered; on the other hand, a sequential Trojan needs to be triggered R times consecutively. Note that an adversary can choose any existing design of a hardware Trojan proposed so far.

The rest of the chapter is organized as follows: The proposed attacks based on hardware Trojan to implement the malicious design modification for the extraction of the secret key from any locked circuit are described in Section 4.1. An algorithm is provided for designing an *Type-p* combinational Trojan considering the set of manufacturing test patterns in Section 4.2. The number of valid Trojans, along with other factors such as area overhead and leakage power for several benchmark circuits, are presented in Section 4.3. Future directions are provided in Section 4.4. Finally, the chapter is concluded in Section 4.5.

4.1 Proposed TAAL Attack for Extracting Secret Keys

The general hardware security strategy adopted for designing and manufacturing a circuit involves a logic locking, where a chip is unlocked by storing a secret key in the tamper-proof memory. As this secret key is the same for all the chips manufactured with the same design, finding this key from one chip undermines the security resulted from logic locking. Research show that an adversary can easily extract the key for a chip using our proposed TAAL attacks, built on tampering through malicious modification by inserting a hardware Trojan to a locked circuit. In this section, three types of TAAL attacks will be present.

4.1.1 Adversarial Model

The adversarial model is given to defining the capabilities and intentions of an attacker clearly. In this model, the attacker (adversary) is assumed to be an untrusted foundry and possesses the following:

- The attacker has access to the locked netlist of a circuit. An untrusted foundry has access to all the layout information, which can be extracted from the GDSII or OASIS file. The netlist can be reconstructed from the layout using reverse engineering with advanced technological tools [77].
- The attacker can determine the location of the tamper-proof memory. It can also find the location of key gates in a netlist, as it can easily trace the route of the other input of the key gate to the tamper-proof memory.
- The attacker can tamper a netlist for its malicious intentions through inserting additional circuitry, commonly known as hardware Trojans, about which the SoC designer is unaware.
- The attacker has access to all the manufacturing test (e.g., stuck-at-fault, delay fault) patterns. Commonly, the production tests are performed at the foundry.

4.1.2 T1 Type TAAL Attack

The *T1 type TAAL* attack is the most straightforward attack among the other two types, which will be introduced in the successive sections. This attack is beneficial for the attacker who does not intend to gain knowledge regarding the security measures implemented for the circuit. The hardware Trojan assists in extracting out the secret key directly from the tamper-proof memory.

Figure 4.1(a) shows the proposed modification to launch *T1 type TAAL* attack. A *Type-3* combinational Trojan (see details in Section 4.2.1) is designed and inserted in the netlist. A combination of 3-input AND gate with an inverter at one of its input served as the trigger and denoted as T, whereas the 2-input multiplexer delivers the payload to the primary output. One input of the multiplexer is the actual output of the locked netlist. The other input is connected to the line formed between the key gate (K) and the tamper-proof memory as a part of logic locking. Under normal operation for

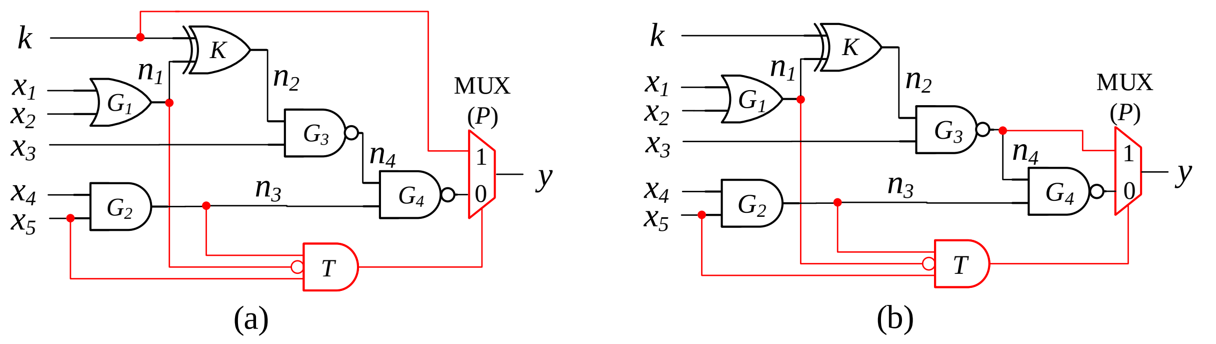


Figure 4.1: (a) *T1 type TAAL attack*, where a *Type-3* combinational Trojan is inserted for key extraction directly from the connection between key gate and tamper-proof memory, (b) *T2 type TAAL attack*, where a *Type-3* combinational Trojan is inserted for the secret key extraction.

any activated chip, the multiplexer propagates the correct circuit functionality at the output. Once the Trojan gets activated with $[x_1\ x_2\ x_3\ x_4\ x_5] = [0\ 0\ X\ 1\ 1]$, the output of AND gate becomes 1, which leads to the extraction of the secret key through the multiplexer at the output. Note that the required number of multiplexers to extract the complete secret key is dependent on the key size.

The *T1 type TAAL* attack is very effective as it does not require any knowledge of the circuit netlist. This attack can also be applied to any logic locking techniques without knowing its implementation details as it directly leaks the key to the primary output. As there are no security measures undertaken in logic locking to protect the connection between the key gates and the tamper-proof memory, any locked circuit can be vulnerable for this attack. Note that an adversary can select any hardware Trojans (combinational or sequential Trojans) of its choice, and one can find the implementation details in Section 4.2.

4.1.3 T2 Type TAAL Attack

Instead of extracting the key directly to the primary output, an adversary can propagate it to the output. In *T1 type TAAL Attack*, once the Trojan is activated, the raw key values are transferred to primary output, which can raise a suspicion of a design being tampered. *T2 type TAAL Attack* primarily addresses this shortcoming of *T1 type attack*, by incorporating logic values in the key, which can easily be separated.

The attack involves tampering a netlist with a *Type-3* combinational Trojan. Similar to *T1 type*, the trigger is constructed using a 3-input AND gate along with an inverter placed before one of the AND gate inputs (see Figure 4.1(b)) and the payload is delivered to the primary output of the circuit using a 2-input MUX. An adversary can choose a net, whose logic value is impacted by the key gate for the input of the MUX. In Figure 4.1(b), net n_4 is selected as the MUX input. One can also select n_2, however, n_3 cannot be selected. The key gate is considered as XOR gate having inputs as secret key (k) and G_1 gate output. In order to propagate the secret key at the output of the key gate (K), G_1 output needs to be specified (either 0 or 1). If $n_1 = 0$, the output of the key gate will be k, otherwise it will be $\overline{k}$. For this example, the trigger requires $n_1 = 0$; else, the Trojan will not be activated. This condition forces $[x_1\ x_2] = [0\ 0]$. Since net n_4 is selected for key extraction, input x_3 also plays a significant role in key propagation as the complementary value at net n_2 can propagate to n_4 only when x_3=1. To launch this attack, an adversary needs to perform the circuit analysis to sensitize the key. This attack requires an adversary to monitor the input pattern to extract the correct key. An adversary extracts $\overline{k}$ when $[x_1\ x_2\ x_3\ x_4\ x_5] = [0\ 0\ 1\ 1\ 1]$, in the example provided in Figure 4.1(b). This attack shows the flexibility to identify individual key gate and target secret key through key propagation from consecutive node selection. The dependency of this attack on primary inputs increases the efficiency of *T2 type* attack where only the adversary knows the logical values of these inputs.

4.1.4 T3 Type TAAL Attack

A locked netlist typically consists of a large number (e.g., 128) of key gates, the effect of one key may affect the propagation of another key to the primary output. A secure logic locking technique can also insert keys in such a way that an adversary cannot propagate the key information to output using manufacturing tests [5]. In such scenarios, *T2 type TAAL attack* may be ineffective in extracting the key. *T3 type TAAL attack* is proposed to addresses this limitation encountered for *T2 type attack*.

Figure 4.2(a) shows the locked netlist, where the propagation of the key (k_1) is prevented by inserting another key (k_2). The output of G_3 cannot be uniquely determined unless an adversary knows either k_1 or k_2 and *T2 type attack* will fail to determine either k_1 or k_2. It is thus necessary to help propagate one key and then determine the other. Figure 4.2(b) shows our proposed *T3 type TAAL attack*, where net n_5 is selected to deliver the payload.

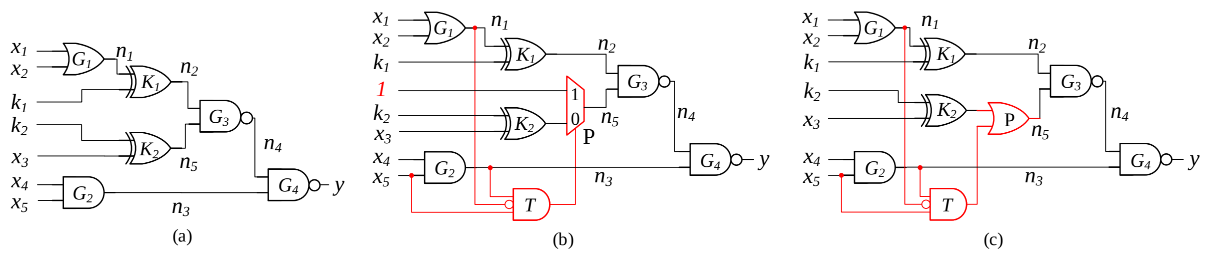


Figure 4.2: T3 Type TAAL attacks. (a) Original netlist with k_2 inserted to prevent the propagation of k_1, (b) *T3 type TAAL attack* with a *Type-3* combinational Trojan with payload as multiplexer (MUX) and (c) *T3 type TAAL attack* with a *Type-3* combinational Trojan with payload as OR gate.

Figure 4.2(b) shows the implantation of *T3 type TAAL attack* using a *Type-3* combinational Trojan. The trigger part for the Trojan can be designed as 3-input AND gate with inverter and payload is delivered through 2-input MUX as before. The key (k_1) propagation requires setting the node n_5 to 1 so that the signal value at node n_2 can propagate through G_3. The other input of the MUX is directly connected to V_{DD}, which is equivalent to logic 1.

The other input of the key gate (K_1) needs to be specified to propagate the key k_1 at node n_2. As one of the inputs to trigger requires $n_1 = 0$ to be satisfied, which results $[x_1\ x_2] = [0\ 0]$, and the output of key gate (K_1) will be k_1. When the trigger condition is met, the output of the AND gate becomes 1, and logic 1 is delivered at the input of G_3 through the MUX. At this point, the Trojan activation nullifies the effect of key value k_2 at the output of $G3$. The signal at n_4 will be $\overline{k_1}$. Finally, setting node n_3 at logic 1 will expose the key at the output, y. As a result, input pattern $[x_1\ x_2\ x_3\ x_4\ x_5] = [0\ 0\ X\ 1\ 1]$ will expose the key k_1 at the output. Once the value of k_1 is known, an adversary can perform the signal propagation analysis to find k_2. Similarly, the payload MUX can be replaced with an OR gate as shown in Figure 4.2(c), the attack works in the same manner where triggering of Trojan would force the output of the payload OR gate to logic 1 irrespective of its other input which will assist in propagating the key value k_1 at the primary output depending on the primary input values as discussed above.

The attacks are explained using a combinational Trojan for the simplicity of understanding. All the attacks proposed can also be implemented using any *Type-p* sequential Trojan, which delivers at its targeted payload once the Trojan is activated R times. The design details for a sequential Trojan is discussed in Section 4.2.2.

Note that the *T3 Type* TAAL attack cannot be applied to DFS structure [28, 29] since DFS architecture prevents leaking of key through the secure cell. However, the adversary can implement either *T1 Type* and *T2 Type* TAAL attacks to bypass the secure cell and be extracted out through scan-chain.

4.2 Designing Hardware Trojans for TAAL Attacks

A hardware Trojan can be described as intentional modifications in the original netlist of a design for malicious purposes [39–43, 113]. A Trojan can be inserted into a circuit during its design or manufacturing stages.

In this chapter, The Trojan is considered, only inserted by an untrusted foundry, which is relevant to logic locking (see the adversarial model in Section 4.1.1).

As logic locking was proposed to address the threat from an untrusted foundry, it can practically thwart the locking mechanism by obtaining the secret key through inserting a hardware Trojan in the design.

A complete hardware Trojan classification can be found in [102]. In this chapter, only combinational and sequential hardware Trojans are been considered to demonstrate the attack. A combinational hardware Trojan generally comprises of a trigger and a payload, the detailed modeling can be found in [112]. On the other hand, sequential Trojans have a state element along with the trigger and payload [43, 114]. Any Trojans can be activated through trigger inputs, which can be taken from the primary inputs and/or internal nodes of a circuit so that manufacturing test patterns cannot trigger a Trojan and remain undetected. The trigger can be implemented as an AND gate. When a Trojan is activated, the output of this AND gate becomes 1 and it delivers the payload (selection input of the multiplexer shown in Figures 4.1-4.2) to the circuit to leak the secret key. The trigger can also be any logic function that provides 1 when activated. Note that a combinational Trojan manifests its effects upon the availability of the trigger inputs and effects the original netlist at the payload, on the other hand, sequential Trojan shows its effect after the occurrence of a sequence or a period of time upon triggered.

4.2.1 Design of a Combinational Hardware Trojan

The primary purpose of a hardware Trojan, once activated, is to modify the original functionality of a circuit to leak the secret key, which is unknown to the SoC designer. It is absolutely necessary that the Trojan must not get activated during scan-based structural or functional tests. In other words, the circuit should not come across any condition during tests that activates the trigger, which can lead to its detection. In this chapter, a step-by-step process is presented for designing a combinational hardware Trojan, initially presented in [112], for a locked netlist.

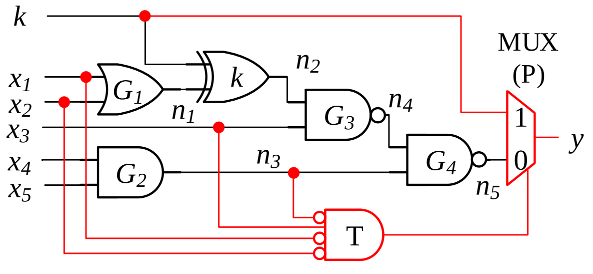
A hardware Trojan can be described based on its trigger inputs, and can be defined as *Type-p* Trojan when it has *p* trigger inputs. The trigger inputs can be selected from primary inputs and/or internal nodes, which are not affected by the key gates. If such a node is selected as a trigger input, an adversary cannot activate a Trojan as it does not know this secret key, and thus the internal signal value for an activation pattern. This pattern is named as hardware Trojan activation pattern (HTAP). The payload of the Trojan (a MUX) can be delivered to a location described in Figures 4.3 for launching our proposed TAAL attack.

Let us determine the number of Trojans, one can insert in a design to extract the secret key. This is basically a selection problem, where an adversary selects p nodes as the trigger inputs from N nodes of a circuit so that the Trojan is not activated during the manufacturing/production tests. The value of N can be determined based on the following equation:

$$N = PI + G + F - M \tag{4.1}$$

where,

PI	:	Number of primary inputs
G	:	Number of gates
F	:	Number of fanout branches
M	:	Number of lines impacted by the key gates



(a)

$$\begin{array}{c|ccccccc|c}
 & P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & \\
k & 1 & 0 & 0 & 1 & 0 & 1 & 1 & X \\
x_1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
x_2 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
x_3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
x_4 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
x_5 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0
\end{array}$$

SAF test patterns (P)

Hardware Trojan activation pattern (HTAP)

(b)

Figure 4.3: Design for a combinational hardware Trojan that evades manufacturing tests. (a) A combinational circuit with a *Type-4* Trojan. (b) Stuck-at fault (SAF) test patterns for manufacturing tests. The hardware Trojan activation pattern (*HTAP*) is $[x_1\ x_2\ x_3\ x_4\ x_5] = [0\ 0\ 1\ 1\ 0]$ while treating the key input as unknown (X).

An upper bound of all possible *Type-p* Trojans (AT_p) can be given by:

$$AT_p = \binom{N}{p} \times 2^p \qquad (4.2)$$

The right-hand side of Equation 4.2 constitutes of two products. The first one represents all possible combinations to select *p* lines from *N*. The second one denotes the trigger combinations, as one line can be applied directly or inverted to the trigger input. Note that the actual number of Trojans (denoted as VT_p) can be less than AT_p as few of them can be detected by the manufacturing test patterns (e.g., stuck-at fault patterns), and few may not be triggered from the primary inputs. However, for a reasonable size circuit, AT_p and VT_p are comparable.

Figure 4.3 shows an example of *TAAL: T1 type* using a *Type-4* Trojan inserted in the netlist. The circuit has five primary inputs (PI). The SoC designer can generate test patterns considering the key as input (the pattern generation is described in detail in [28,29]. To detect all the stuck-at faults (SAFs), seven test patterns (e.g., $P = \{P_1, P_2, \ldots, P_7\}$) are required and they are generated using Synopsys TetraMax [96] ATPG tool. To avoid a Trojan being activated by these manufacturing test patterns, the Trojan's trigger must remain quiet for all these input patterns. A hardware Trojan activation pattern is selected, where $HTAP = (X\ 0\ 0\ 1\ 1\ 0)^T \notin P$. As the logic values of nodes n_2, n_4, and n_5 are impacted by the key, k, these nodes are excluded in designing the Trojan. If one of these nodes is selected, an adversary may not activate the trigger as it does not know the key value. The upper bound of all possible *Type-4* Trojans (AT_4) can be given by:

$$AT_4 = \binom{7}{4} \times 2^4 = 560 \tag{4.3}$$

where, $N = 5 + 5 - 3 = 7$.

Out of these 560 possible Trojans, 188 will be detected by the test patterns P. The remaining 372 will be treated as valid Trojans, and an adversary can select one of them. In Figure 4.3, $\overline{x_1}$, x_2, $\overline{x_3}$ and $\overline{n_3}$ are selected as the trigger. Similarly, one can also design other types (*Type-1* through *Type-6*) of Trojans to launch TAAL attacks. Note that the Trojan needs to be quiet during normal operations so that no functional errors are observed at the primary output. An adversary can select rare nodes, whose value do not become identical with trigger pattern very often under continuous/normal operation of the IC, for the trigger inputs while designing a Trojan. One can perform controllability, and observability analysis [33] to find such rare nodes, and then select them as the trigger inputs. However, such analysis is not included, as including rare nodes in the trigger for a sequential hardware Trojan is not a hard requirement. In this chapter, a sequential Trojan is modeled using a combinational Trojan that needs to be triggered R times consecutively.

An automated process is developed to design a combinational hardware Trojan. Algorithm 2 provides steps to be followed for designing a *Type-p* Trojan that eludes activation during the manufacturing test. The inputs of the algorithm are locked netlist (C), M production/manufacturing test patterns ($P = \{P_1,\ P_2,\ \ldots,\ P_M\}$) and the Trojan type ($p$). The algorithm results in the hardware Trojan activation pattern (HTAP) and the trigger inputs. Initially, it reads the Trojan-free locked netlist (C) and the set of manufacturing test patterns (P) (lines 1-2). Logic simulation is performed using these patterns, and store the internal node values in a matrix, A (line 3). It is unnecessary to store the nodes that are impacted by the key gates, as their values will be unknown (X) during the simulation. Note that A is a $N \times M$ matrix. A hardware Trojan activation pattern of an adversary's choice is selected (line 4). Similarly, matrix H is formed due to logic simulation using HTAP (line 5). Here, H is a $N \times 1$ vector. To select the trigger inputs, one can select p random nodes that are not affected by the key gates of the locked circuit (line 6). To perform the

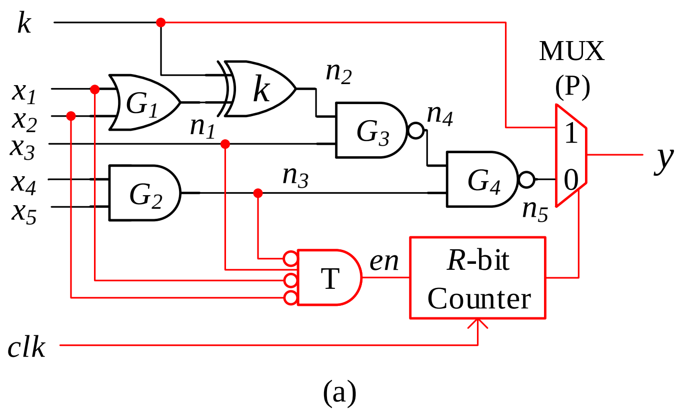
Algorithm 2: Design of Type-p Trojan.

```
input  : Locked Netlist ($C$), test pattern set ($P$), Type-p Trojan
output : Hardware Trojan activation pattern (HTAP), Trigger inputs ($T$)
1  Read the locked netlist ($C$);
2  Read production test patterns ($P$);
3  Perform logic simulation using $P$ to form a matrix ($A$) of all the internal node values;
4  Select a hardware Trojan activation pattern ($HTAP$), where $HTAP \notin P$ ;
5  Perform
    logic simulation using $HTAP$ and form a matrix ($H$) of all the internal node values;
6  Select $p$ random nodes that are not affected by the key gates of C for the trigger inputs;
7  Construct a new matrix $A_p$ that corresponds to the trigger locations for all test patterns;
8  Construct a new vector $H_p$ that corresponds to the trigger locations for $HTAP$ ;
9  if $H_p \notin A_p$ then
10     Choose selected $p$ nodes as trigger, $T$;
11 else
12     Discard selected $p$ nodes, as it would activate the Trojan during tests;
13     Go to Step 6;
14 end
15 Report $HTAP$ and $T$;
```

search whether the trigger values are presented in the production set, the matrix A_p and vector H_p are constructed (lines 7-8). If H_p is not in A_p, the trigger (T) is selected (Line 10), otherwise, drop the selected p locations as the Trojan will be activated during the production tests (lines 12-13) and new p locations are selected (line 6). Finally, the algorithm reports $HTAP$ and T (line 15).

4.2.2 Design of a Sequential Hardware Trojan

A sequential Trojan modifies the functionality of a circuit until a specified time has elapsed after the trigger condition is satisfied. However, in this section, a sequential Trojan has been designed which needs to be triggered R times to deliver the payload. Moreover, the sequential Trojan has



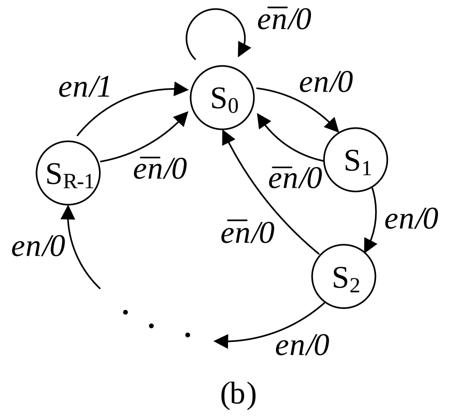


Figure 4.4: (a) The netlist of a sequential Trojan with a *R-bit* counter, (b) The finite state machine (FSM) of the counter used in a sequential Trojan.

been designed in such way that it can be modeled using a combinational Trojan, described in detail in the previous section.

A sequential Trojan also consists of a trigger and payload similar to a combinational Trojan. Additionally, the trigger part contains state elements that ascertain the payload in future time. In our sequential Trojan design, a *R-bit* counter is implemented as the state elements. This counter is enabled (en) once the trigger condition is fulfilled, i.e., the output of the p-input AND gate becomes 1. The counter increments by one unit, every-time the Trojan is triggered using the Trojan activation pattern (HTAP). The Trojan delivers at the payload (MUX) only after reaching the maximum count value (R). The circuit tampered with Sequential Trojan will show the intended malfunction, key extraction in TAAL attacks, only upon applying the activation pattern successively *R*-times to the circuit.

Figure 4.4(a) shows the *TAAL attack* using a sequential Trojan. The trigger consists of a p-input AND gate and a R-bit counter. The finite-state machine (FSM) of the counter is shown in Figure 4.4(b). The FSM goes to the next states when $en = 1$; otherwise, it returns to the initial state, S_0. The counter produces an output of 1, once en is hold to 1 consecutively R clock cycles, as it takes $(R - 1)$ cycles to reach S_{R-1}. Note that this sequential Trojan can be modeled as *R* combinational Trojans. An adversary can also design a different sequential Trojan, already in the literature, to launch the *TAAL attack*. The sequential Trojan increases the complexity compared to a combinational Trojan as it manifests its effect to the payload only after the sequence of repeated application of trigger inputs. Only the adversary has the knowledge regarding the maximum counter value making it very difficult for detection by others.

Note that the Trojan flip-flops (*FFs*) cannot be a part of the scan chain, since a Trojan is implanted after the test pattern generation, which is done at the design house. Any modifications in the scan chain will be detected by the stuck-at fault test. As the Trojan circuitry is extremely small (one counter and an AND gate for the trigger), it is possible to verify the proper operation using a functional test (i.e., applying the trigger pattern R times and observing the response). There is no need to add Trojan *FFs* in the scan chain and perform scan tests. Now, one can argue that test patterns are generated at the foundry, and Trojan *FFs* are a part of the scan chain. In such a situation, the scan shift will not change the state of the original *FFs* in the circuit. However, it is highly unlikely that an adversary will add these malicious *FFs* in the scan chain, as one can easily find the response of the combinational part of the Trojan and determine tampering.

4.2.3 Design of an Analog/RF Trojan

Analog Trojans [115–117] can also be used to launch different TAAL attacks. In this section, a brief description of different analog Trojans is provided. The payload can be implemented using a multiplexer or an OR gate (see Figures 4.1 and 4.2) like in a combinational or sequential hardware Trojan. The implementation of the trigger, however, will be different depending on the Trojan design. The analog or RF-leaking Trojans can detect an extremely rare event with just a handful of transistors added to the circuit. Moreover, analog techniques and characteristics to design the stealthy trigger for Trojans are usually difficult to uncover due to their small footprint [115]. The authors used the switched capacitor to design a trigger circuit, which is activated when the frequency of toggling on a victim wire goes above a certain threshold. When the wire toggles frequently, charge accumulates on the capacitor faster than it leaks away, eventually the voltage of the capacitor rises above the threshold. This deploys the payload to cause intentional malicious activity. A similar notion is utilized to introduce triggers that are activated after some delay or operate on a specific voltage threshold [116]. Kison et al. exploited the capacitor coupling in sub-micron process technologies to design Trojans [117]. This technique uses rerouting and extending existing layout tracks to increase the capacitive coupling between a victim and an

Table 4.1: Circuit parameters.

Benchmarks	**# Gates**	**Key Size ($\|K\|$)**	**# Total Lines ($N+M$)**	**# Net Lines (N)**	**# Test Patterns**	**Fault coverage**
c432	160	30	349	233	58	100%
c499	202	30	491	226	78	100%
c880	383	30	594	350	86	100%
c1908	880	30	552	223	83	100%
c3540	1669	83	1826	1114	173	100%
c6288	2416	128	5621	1335	77	100%

aggressor wire in a way that a low to high transition on the aggressor can adequately affect the victim wire and flip its digital value. Similarly, RF-leaking Trojans leak the information through the Trojan-induced channel without affecting the legitimate signal/channel [118, 119].

4.3 Analysis

The hardware Trojans presented in Section 4.2 pose a unique challenge to the SoC designers for securing their designs.

4.3.1 Complexity Analysis

In this section, it will be shown that an adversary can implement a Trojan in a very large number of ways, and it is practically infeasible to detect all of them with absolute certainty. Six benchmark circuits from ISCAS'85 benchmark suites [120] are chosen to show the complexity of Trojan detection even for these small benchmark circuits.

Table 4.1 shows the design details for different locked benchmark circuits. The number of logic gates and key size for these circuits is shown in columns 2 and 3. The number of key bits is selected in such a way that the total area overhead does not exceed 5%. However, for industrial design with millions of gates, the key gates will merely add any overhead. Column 4 represents the total number of nets in these circuits (see Equation 4.1). The number of nets that are not affected by key gates is shown in column 5. A key gate is selected by analyzing the netlist and forward

Table 4.2: Number of hardware Trojans for launching TAAL attacks.

Benchmarks	**Type-2 Trojan**		**Type-3 Trojan**		**Type-4 Trojan**	
	$\boldsymbol{AT_2}$	$\boldsymbol{VT_2}$	$\boldsymbol{AT_3}$	$\boldsymbol{VT_3}$	$\boldsymbol{AT_4}$	$\boldsymbol{VT_4}$
c432	1.08×10^5	1.04×10^5	1.66×10^7	1.43×10^7	1.91×10^9	1.34×10^9
c499	1.02×10^5	0.27×10^5	1.52×10^7	2.11×10^6	1.69×10^9	1.21×10^8
c880	2.44×10^5	2.25×10^5	5.67×10^7	4.80×10^7	9.83×10^9	7.35×10^9
c1908	0.99×10^5	0.96×10^5	1.46×10^7	1.33×10^7	1.60×10^9	1.27×10^9
c3540	2.48×10^6	2.35×10^6	1.84×10^9	1.57×10^9	1.02×10^{12}	0.74×10^{12}
c6288	3.56×10^6	3.50×10^6	3.17×10^9	3.00×10^9	2.11×10^{12}	1.82×10^{12}

tracing is performed till the primary output(s) is reached. Simultaneously, removing all the nodes that belongs to these paths from the overall list of $N + M$. Note that this includes any fanout branches as well while performing forward tracing. This step is repeated for all the key gates to get the nodes that are not impacted by the key values. Note that these nets cannot be selected for trigger inputs. The manufacturing test patterns are generated using Synopsys TetraMax Automatic Test Pattern Generation (ATPG) tool [96] with targeted 100% fault coverage (columns 6-7). For the $c432$ benchmark, 30 key gates are randomly inserted in the netlist with 160 logic gates. There are 349 nets in the netlist, out of which 233 nets can be selected for the Trojan trigger as the remaining nets are affected by the key gates. The TetraMax ATPG tool generates 58 stuck-at fault patterns and reports 100% fault coverage. An adversary will use these test patterns to design the Trojans such that they are not activated during the manufacturing tests. A similar analysis can be performed for all other benchmark circuits through the details mentioned in respective rows.

Table 4.2 shows the number of combinational hardware Trojans that can be designed to perform *TAAL* attacks (mentioned in Section 4.1) for different benchmark circuits. The upper bound (see Equation 4.2) for all possible Trojans that can be inserted in the circuit is denoted in columns 2, 4, and 6. Out of all possible Trojans, the valid Trojans that will not be detected during manufacturing tests are shown in columns 3, 5, and 7. For $c432$ benchmark circuit, the total number of *Type-2* Trojans is 1.08×10^5, whereas, the number of valid Trojans is 1.0×10^5. The number of Trojans increases exponentially with the increase of the Trojan type (p). Note that AT_p and VT_p are of the same order, which allows an adversary to select a Trojan of choice from a large collection. It is worthwhile to mention that an adversary needs to choose a Trojan whose triggers are selected from the rare nodes such that it does not get activated too frequently during normal operation. However, it is not necessary to impose this condition for designing a sequential Trojan, as it is highly unlikely that a particular trigger condition will arrive R times consecutively during the normal operation of a chip.

Note that the result in Table 4.2 does not show the complexity for sequential hardware Trojans as the ISCAS'85 benchmark circuits are combinational in nature. This table is intended to show the number of possible combinational Trojans resulting from N nets. One can easily extend the results for industrial designs, where an adversary can insert any type of combinational, sequential, or analog Trojans to perform a TAAL attack. Note that we choose small ISCAS'85 benchmark circuits instead of larger benchmarks (e.g., opencores) to demonstrate the difficulty of detecting all the Trojans even for smaller circuits. It is obvious to infer that the difficulty will increase for larger circuits. The number of all possible valid Trojans is on the order of 10^{12}, which is so large even for a small benchmark circuit $c6288$ with 2416 gates, when considering only four trigger inputs. If we show that detecting all possible valid Trojans for a rather small benchmark is a complex problem, it will be sufficient to imagine the complexity of this problem for large circuits.

4.3.2 Overhead Analysis

The area for a hardware Trojan can be varied based on the trigger inputs. A *Type-p* combinational Trojan consists of an AND gate with *p*-trigger inputs and a multiplexer or an OR gate as payload. For a sequential Trojan, it is necessary to add a R-bit counter along with a p-input AND gate to implement the trigger. This subsection presents the area and power overhead for different ITC'99 benchmark circuits [121] locked with a 128-bit key. The simulation is performed by using Synopsys design compiler [90] with 32nm technology [97]. A single sequential hardware Trojan of *Type-2* to Type-4 is added to each benchmark circuits for launching the *T3 Type* TAAL attack. The output of a single trigger is routed to 128 payload locations (OR gates). This is the worst-case area overhead, as we often need less than 128 OR gates to expose all the 128 key bits. For example, shown in Figure 3(c), we need only one OR gate to determine two key bits (e.g., K_1 and K_2). For a given benchmark circuit, A_O represents the original circuit area, whereas A_T represents the area of its Trojan inserted version. The area overhead (AO) is computed using the following formula:

$$AO = \frac{A_T - A_O}{A_O} \times 100\% \tag{4.4}$$

Table 4.3 shows the area overhead analysis for ITC'99 benchmark circuits. The number of logic gates and flip-flops (FFs) for each synthesized benchmark circuits are shown in columns 2 and 3, respectively. The Type of Trojan is represented in column 4. It follows the same definition of *Type-p* Trojan, where p represents the number of trigger inputs. Columns 5 through 8 indicate the percentage area overhead (computed using Equation 4.4). The Trojan consists of a counter and an AND gate as the trigger, and 128 OR gates as the payload. R represents the maximum count for the counter. For example, $R = 8$ means that the trigger has a maximum count of 8.

We observe an area overhead of less than 5% (e.g., $b14$ with only 2064 gates and 215 FFs) for a small benchmark circuit. However, it becomes significantly smaller for larger benchmark circuits. Considering $b19$, the percentage area overhead is 0.13$\%$ for *Type-2* sequential Trojan with $R = 8$. The area overhead remains almost constant even with an increased count (e.g., $R = 16$) and different Trojan Types for a large benchmark circuit. A similar analysis can be performed for all the benchmark circuits as well. Note that the majority of the overhead results from the payload as we require 128 OR gates to determine 128 key bits.

Table 4.4 shows the power overhead analysis for the same benchmark circuits. We have computed two overhead for the dynamic power and the leakage power using Equation 4.5 and Equation 4.6, respectively. As the Trojan remains quiet most of the time unless triggered, leakage power overhead is of the Trojan designers' concern so that it evades detection. Dynamic power overhead is,

$$DPO = \frac{DP_T - DP_O}{DP_O} \times 100\% \tag{4.5}$$

where, DP_T and DP_O represent the dynamic power for the Trojan inserted and Trojan free circuits, respectively. Leakage power overhead is,

$$SPO = \frac{SP_T - SP_O}{SP_O} \times 100\% \tag{4.6}$$

Table 4.3: Area overhead for ITC'99 benchmark circuits.

Benchmarks	**# Gates**	**# FFs**	**Trojan Type**	**Area Overhead (%)**			
				R=2	**R=4**	**R=8**	**R=16**
b14	2064	215	Type-2	4.03	4.23	4.41	4.56
			Type-3	4.05	4.23	4.41	4.59
			Type-4	4.06	4.27	4.41	4.62
b15	2722	418	Type-2	3.02	3.16	3.30	3.41
			Type-3	3.03	3.17	3.30	3.43
			Type-4	3.04	3.19	3.30	3.45
b20	4190	430	Type-2	1.92	2.01	2.10	2.17
			Type-3	1.93	2.02	2.10	2.19
			Type-4	1.94	2.03	2.10	2.20
b21	4225	430	Type-2	1.90	1.99	2.08	2.15
			Type-3	1.91	1.99	2.08	2.16
			Type-4	1.91	2.01	2.06	2.17
b17	8629	1328	Type-2	0.89	0.93	0.97	1.00
			Type-3	0.90	0.94	0.97	1.01
			Type-4	0.90	0.94	0.97	1.02
b18	39845	3168	Type-2	0.24	0.25	0.26	0.27
			Type-3	0.24	0.25	0.26	0.27
			Type-4	0.24	0.25	0.26	0.27
b19	78076	6337	Type-2	0.12	0.12	0.13	0.13
			Type-3	0.12	0.12	0.13	0.13
			Type-4	0.12	0.12	0.13	0.13

where, SP_T, SP_O represent the leakage power for the Trojan inserted and Trojan free circuits, respectively.

Columns 1 and 2 of Table 4.4 represent different benchmark circuits and Trojan types. Columns 3-6 show dynamic power overhead for a sequential Trojan with $R = 2, 4, 8$ and 16, respectively. On the other hand, columns 7-10 show leakage power overhead for sequential Trojan with $R = 2, 4, 8$ and 16, respectively. For example, For a small benchmark circuit, we observe a dynamic power

Table 4.4: Power overhead for benchmark circuits.

Bench-marks	**Trojan Types**	**Dynamic Power Overhead (%)**				**Leakage Power Overhead (%)**			
		R=2	***R=4***	***R=8***	***R=16***	***R=2***	***R=4***	***R=8***	***R=16***
b14	*Type-2*	9.61	8.46	8.63	8.86	6.02	6.63	6.89	7.05
	Type-3	9.70	8.34	8.60	8.80	5.91	6.62	6.77	7.03
	Type-4	9.63	8.38	8.52	8.70	5.95	6.64	6.82	7.05
b15	*Type-2*	10.22	8.99	9.18	9.42	4.78	5.27	5.48	5.60
	Type-3	10.31	8.87	9.14	9.36	4.69	5.26	5.37	5.58
	Type-4	10.24	8.90	9.05	9.24	4.73	5.28	5.42	5.60
b20	*Type-2*	7.42	6.53	6.67	6.84	3.01	3.32	3.45	3.53
	Type-3	7.49	6.44	6.64	6.80	2.96	3.31	3.39	3.52
	Type-4	7.44	6.47	6.58	6.71	2.98	3.33	3.41	3.53
b21	*Type-2*	7.67	6.75	6.89	7.07	2.99	3.30	3.43	3.51
	Type-3	7.74	6.66	6.86	7.03	2.94	3.29	3.36	3.49
	Type-4	7.69	6.68	6.80	6.94	2.96	3.30	3.39	3.51
b17	*Type-2*	3.36	2.96	3.02	3.10	1.52	1.67	1.74	1.78
	Type-3	3.39	2.92	3.01	3.08	1.49	1.67	1.71	1.77
	Type-4	3.37	2.93	2.98	3.04	1.50	1.68	1.72	1.78
b18	*Type-2*	0.24	0.21	0.21	0.22	0.38	0.42	0.44	0.45
	Type-3	0.24	0.21	0.21	0.22	0.37	0.42	0.43	0.44
	Type-4	0.24	0.21	0.21	0.21	0.38	0.42	0.43	0.45
b19	*Type-2*	0.12	0.10	0.11	0.11	0.20	0.22	0.22	0.23
	Type-3	0.12	0.10	0.11	0.11	0.19	0.21	0.22	0.23
	Type-4	0.12	0.10	0.10	0.11	0.19	0.22	0.22	0.23

overhead is around 10% (e.g., $b15$) and leakage power overhead is around 7% (e.g., $b14$). However, for larger benchmark circuits, the power overhead becomes very small. The percentage dynamic power and leakage power overheads for $b19$ are 0.10% and 0.22% for Type-2 sequential Trojan with $R = 4$, respectively. Note that for a large circuit, an increased count (e.g., $R = 16$) and a different type of Trojan will not have a significant impact on dynamic power and leakage power overheads.

4.4 Future Research Direction for Secure Logic Locking

The security of a logic locking technique can be tied together with the hardware Trojan detection problem. Developing SAT-resistant logic locking is not sufficient enough to prevent IC overproduction or to protect IPs. It is required to address the detection of Trojans inserted at an untrusted manufacturing site. Researchers have already proposed techniques to detect and prevent hardware Trojans. The detection methods can be grouped into two categories, namely, logic testing [46–50], and side-channel analysis [51–56]. On the other hand, prevention methods can be categorized as design-for-trust measures [57–60] and split manufacturing [61, 62].

Logic testing by applying stimuli to primary inputs (PIs) and observing responses at primary outputs (POs) can be used to detect these Trojans [43, 46, 48–50]. The decision being made is whether a chip is tampered with a hardware Trojan by observing a mismatch between the observed and expected responses. Note that the accuracy of the detection process does not depend on the manufacturing process variations. However, the detection will be extremely difficult as it is practically impossible to detect all types of combinational Trojans. In addition, it is not feasible to trigger a sequential Trojan, as it requires to apply the same trigger pattern at the input R times.

Side-channel information, such as, power [63], temperature [64], delay [65], and radiation [66] can be used to detect a hardware Trojan. The side-channel fingerprinting technique for detecting Analog/RF hardware Trojans in a wireless cryptographic IC has also been proposed [122, 123]. These detection methods rely on the availability of Trojan-free golden circuits for creating Trojan free signature. It can be very difficult to acquire a golden sample as all manufactured chips may have Trojans. Path delay-based testing has limitations on detecting a hardware Trojan as long as the Trojan is inactive. Note that activating a Trojan is challenging as an adversary can select a hard to activate Trojan. In addition, the path delay does not depend on the size of the trigger circuit or the number of payloads attached with each trigger as the Trojan remains quiet during the logic testing. In addition, process and environmental variations may mask the side-channel leakage, when a Trojan circuitry is small.

Researchers have also proposed measures to prevent a Trojan from being inserted into the design in the first place. These solutions involved characterization of ring-oscillator [124], shadow registers [55], or delay elements [125] to detect the delay deviation caused by hardware Trojans. Reducing the rare signal in the circuitry is another proposed method for designers to reduce the risk of being implanted with a Trojan [58, 126]. Camouflage fill techniques [127, 128] to create indistinguishable layouts for different gates by adding dummy contacts and connections can prevent the attacker from extracting the correct gate-level netlist of a circuit for Trojan insertion. Xiao et al. proposed to fill all the unused spaces using filler cells so that an untrusted foundry cannot insert a Trojan [59, 129]. However, this direction still lacks any firm solution and more emphasis has been given to Trojan detection.

Split Manufacturing can be an effective way to thwart Hardware Trojan insertion at an untrusted foundry. In split manufacturing, the production of ICs is carried out in two different foundries [130]. The design is divided into two parts – Front End of Line (FEOL) and Back End of Line (BEOL). An untrusted foundry is provided with the FEOL design, which contains partial information regarding the design that requires complex steps for fabricating and involves higher cost. Fabrication of BEOL does not incorporate complex fabrication steps and can be done by a smaller trusted foundry. The untrusted foundry sends the fabricated wafers directly to the smaller foundry for the complete fabrication. This way the untrusted foundry can be restricted to make any Trojan based modification as it does not have the complete information regarding the design. However, several attacks undermining the security achieved through split manufacturing have also been proposed in the past [131].

Recent research contributions show that machine learning and image processing can also be incorporated to detect hardware Trojans in the chip. Vashistha et al. presented Trojan scanner [132], which uses a trusted GDSII layout (golden layout) and scanning electron microscope (SEM) images to identify the malicious modifications made in the netlist during the manufacturing of a circuit. A unique descriptor for each type of gate is prepared based on different features using computer vision algorithms along with a machine-learning model of a golden layout and SEM images of an IC under authentication. These descriptors, when compared with each other can detect any modifications either in the form of additional gates or modified gates, which might raise the suspicion for a potential hardware Trojan. Moreover, Trojan scanner also presents the trade-off between the accuracy and SEM parameters. The authors demonstrated the effectiveness of the scheme using a smart card die as a test sample (generally manufactured with 90 *nm* technology). Its effectiveness for detection when a chip is fabricated using recent technology nodes (10 *nm* and beyond) is yet to be validated.

Research groups have also investigated vulnerabilities in the analog/RF front-end of a wireless device that can facilitate hardware Trojan attacks. Subramani et al. proposed defense to distinguish between channel-induced and Trojan-induced impact on the legitimate signal through its traits [118]. Acknowledging excessive toggling activity on a victim wire of Trojans such as A2 [115], Hou et al. proposed to add on-chip monitors to measure switching activity on potential victim wires during an adjustable time period and raise an alarm if such activity goes above a certain threshold [133]. This method can be effective for detecting A2 Trojan which comprises of a capacitor as a trigger. Reverse engineering based detection methods have also been proposed by sorting trace lengths to realize the minimum capacitance needed for such Trojans [134]. However, it is possible to mitigate this requirement by an adversary, e.g., by using multiple layers or higher voltages to evade detection by this kind of approaches.

Despite significant research has been performed on detecting hardware Trojans, the researcher still lack efficient and accurate methods for modeling them and generating tests for their detection. Once the detection of hardware Trojans is ensured, an SoC designer can choose a SAT-resistant logic locking to prevent IC overproduction and IP piracy.

4.5 Summary

In this chapter, the vulnerability of logic locking techniques through a set of tampering attacks with hardware Trojans has been demonstrated. Three types of proposed *TAAL attacks* can defeat any logic locking techniques that rely on storing the secret key in a tamper-proof memory. In *T1 type TAAL Attack*, An adversary could extract the key from a locked netlist without knowing the details of the logic locking technique used to protect the circuit. For *T2 type* and *T3 type TAAL* attacks, the complexity of detecting an attack has been improved. Only the attacker knows the specific values that can lead to key extraction, increasing the identification of a *TAAL* attack. To launch a *TAAL* attack, models for combinational and sequential hardware Trojans are developed. An algorithm for designing a hardware Trojan is also proposed, which would not be detected by manufacturing tests. The results depict the range of Trojans selected by an adversary, whose number has a very high order of magnitude. Finally, the avoidance strategies for hardware Trojans to make logic locking secure are described.

According to this chapter, hardware Trojan (HT) added during manufacturing process may cause great harm to the semiconductor industry. Compared with other attackers, untrusted foundry has more resources. As an attacker, the foundry has the ability to access all the layout information and all the manufacturing test patterns. Thus, it can easily insert HT based on this information without the knowledge of the designer. Therefore, the method of detecting HT is also particularly important in the semiconductor industry. In next chapter, a modeling and test generation method for combinational hardware Trojan is proposed [112].

Chapter 5

Modeling and Test Generation for Combinational Hardware Trojans

A generalized model of a combinational hardware Trojan is proposed and this could be the first time such a model for a Trojan is being presented. Meanwhile, a generalized method based on conditional stuck-at fault patterns (CSP) to detect the modeled Trojans is also proposed. The contributions of this chapter are:

- *Design of a combinational hardware Trojan:* We have proposed a generalized model of hardware Trojan based on the circuit netlist. We call this Type-n Trojan, where n is the number of the trigger inputs. The payload of this Trojan can be delivered to a location where a stuck-at fault (SAF) is detectable by a Trojan activation pattern (TAP). Because TAPs may detect several stuck-at faults, the location of the Trojans is not unique. Any such fault site inserted with payload will result in Trojan behavior for the TAP. We believe this is the first approach to model a generalized Type-n Trojan.
- *Detection based on conditional SAFs:* We have proposed a hardware Trojan detection technique based on conditional detection of SAFs. With reasonable test length, we can detect all Type-1 Trojans. These conditional SAF patterns (CSPs) also detect higher order Trojans with reasonable confidence. It is reasonable to assume that an adversary will not have access to the CSP since they would not be included in manufacturing data.

The rest of this chapter is organized as follows: Section 5.1 describes the generalized model for a combinational hardware Trojan, termed as Type-n Trojan. Section 5.2 details our approach for detecting Type-n Trojans. Section 5.3 describes simulation results to demonstrate the effectiveness of our proposed conditional SAF patterns for hardware Trojan detection. Section 5.4 concludes the chapter.

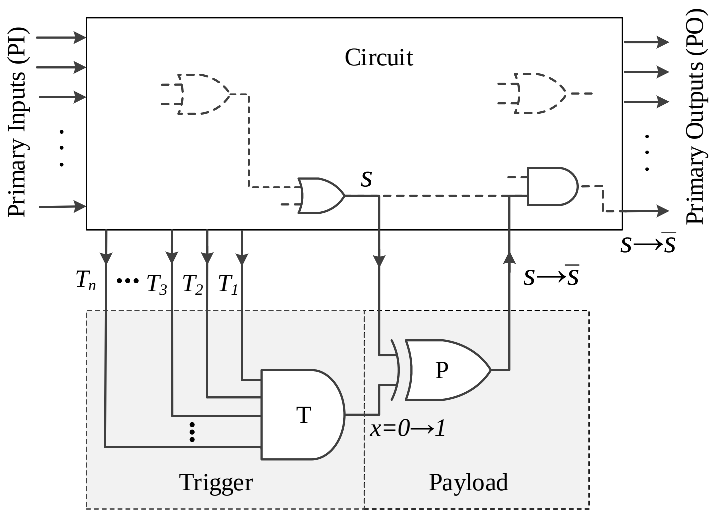


Figure 5.1: A model for a combinational hardware Trojan.

5.1 Modeling a Hardware Trojan

A hardware Trojan has two parts, namely, a trigger and a payload as shown in Figure 5.1. The trigger activates the hardware Trojan when a certain condition is satisfied. Inputs to the trigger can directly come from primary inputs (PI) or from internal nets of the circuit. Although shown here as an AND gate, the trigger can be any logic function. When the Trojan is activated, e.g., when the AND gate output becomes 1, it delivers the payload to the circuit by modifying its functionality. A two-input XOR gate with inputs from the trigger and a net in the circuit, can be used for such purpose. The output of the XOR gate is taken back to the circuit.

Trojans, added for malicious purposes, consist of circuitry (trigger and payload) that has been added to a VLSI chip without the knowledge of the designer or the user. A hardware Trojan must have the following properties:

- **Property 1:** A Trojan modifies the logical function of a chip, although the modification may be subtle. For certain inputs, termed as *activation vectors* or *activation patterns*, the output of the chip then deviates from its correct value. This incorrect result may help an adversary to fulfill his/her malicious purpose.
- **Property 2:** A Trojan must not be activated by production (scan-based structural or functional) tests. This leads the Trojan circuitry to remain undetected during production testing of the chip.

- **Property 3:** Although the effect of a Trojan may appear similar to a design error, the Trojan distinctly differs from a design error. In case of a remaining error in a completed design, production tests are generated for the chip with error and these tests aim at preserving the error in the manufactured chip. Since the Trojan is inserted in the chip design after the production tests were generated, the Trojan circuitry is, by design, made transparent to tests. Thus, the design of a Trojan must consider its function (activation inputs and modified outputs), the chip function (typically, a netlist), and the production tests.

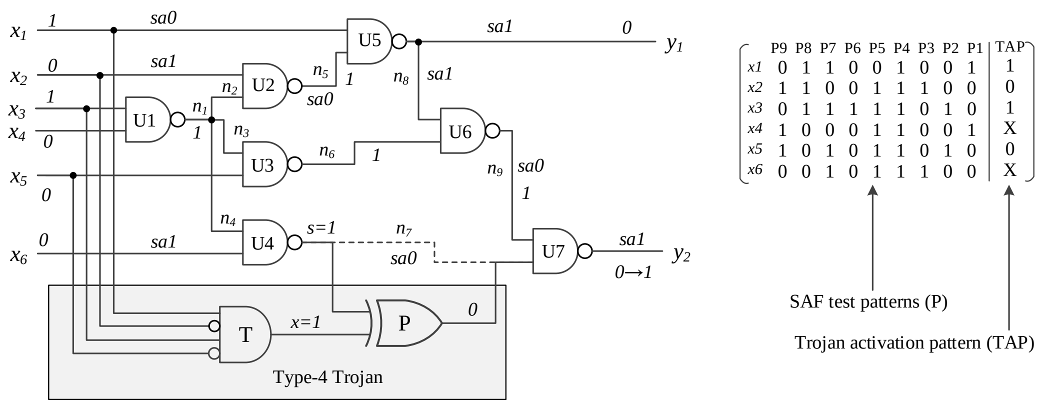
5.1.1 Hardware Trojan Model

The Trojan circuitry may be designed for malicious purposes, such as, expose some secret key to an adversary, transmit unencrypted data to an unsecured channel, disable a circuitry, or incorrectly execute an intended function. A hardware Trojan modifies the input-output characteristics of the chip and thus provides an adversary to gain undue advantage. In this chapter, Assuming that the chip is sequential, is implemented with flip-flops and combinational logic, and is tested through the scan technique [33]. Typically, manufacturing tests are generated for single stuck-at faults of the combinational logic. These tests are digital vectors applied to primary inputs (PIs) of the combinational logic and the results at primary outputs (POs) are verified against expected responses. Functional tests and delay tests are also performed at the manufacturing site. Without loss of generality, the focus of discussion is on stuck-at fault (SAF) manufacturing tests for designing a Trojan and its detection.

Two single stuck-at (SSA) faults, namely, stuck-at-0 ($sa0$) and stuck-at-1 ($sa1$), are modeled on every *signal* or *line*, where a signal can be a primary input (PI), a gate output, or a fanout branch. Thus, the number K of fault sites is given by:

$$K = \#PI + \#Gates + \#Fanout\ branches \tag{5.1}$$

A test for a fault on a signal assumes all other signals to be fault-free. The test activates the fault by setting the signal to an appropriate value, for example, 0 for a $sa1$ fault, and propagates the



	P9	P8	P7	P6	P5	P4	P3	P2	P1	TAP
x1	0	1	1	0	0	1	0	0	1	1
x2	1	1	0	0	1	1	1	0	0	0
x3	0	1	1	1	1	1	0	1	0	1
x4	1	0	0	0	1	1	0	0	1	X
x5	1	0	1	0	1	1	0	1	0	0
x6	0	0	1	0	1	1	1	0	0	X

Figure 5.2: An 18-line $(K = 18)$ combinational circuit with Type-4 Trojan $(n_7|x_1, \overline{x_2}, x_3, \overline{x_5})$. For Trojan activation pattern (TAP) 101000, logic states of lines and detectable SAFs are marked on the circuit.

state of the signal to a primary output (PO). In addition, there are specific test sequences to verify the function of the scan shift register [33].

To facilitate the testing of a hardware Trojan, we propose a model shown in Figure 5.1 with following attributes:

1. Trigger: The objective of a Trojan designer is to evade manufacturing tests, otherwise every chip will fail at the testing site. The trigger circuit must remain quiet (e.g., output of the trigger x remains "0") during the tests. The selection of the trigger inputs (T_i) can be from the primary inputs or internal nets of the circuit.
2. Payload: A net (s) is selected in the circuit to deliver the payload of the Trojan. The original signal s, shown with broken line in Figure 5.1, is rerouted through a two-input XOR gate whose other input is either the trigger x as shown in Figure 5.1 or $\overline{x}$. We define this net as Trojan location and assume that it is distinctly different from the set $\{T_i\}$ used to generate the trigger. Two conditions must be satisfied by a vector at PI to activate the Trojan. First, the vector should activate a path from s to a PO, hence it should be a test for either a $sa0$ or $sa1$ fault on s. Second, this vector should place a logic 1 on x (or logic 0 if $\overline{x}$ is connected to the XOR). As a result the PO will experience a signal inversion, changing the true function of the circuit.

Definition 1. A *Type-n Trojan* is defined as a combinational hardware Trojan of order n and has n trigger inputs.

Definition 2. The *location* of a Type-n Trojan is defined as a site (signal or line) in the circuit where the payload is delivered.

A Type-1 Trojan has only one trigger input that can come from any part of the netlist or a primary input. Similarly, a Type-2 Trojan has two trigger inputs. We note that for lower order Trojans, in general, the trigger inputs may come from low switching nets to keep the Trojans mostly quiet.

Figure 5.2 shows an example of a Type-4 Trojan inserted in a 6-input, 2-output circuit, where we select trigger inputs directly from PIs. We specify this Trojan as $(n_7|x_1, \overline{x_2}, x_3, \overline{x_5})$, where n_7 is the payload site and $x_1, \overline{x_2}, x_3$ and $\overline{x_5}$ are the trigger input signals. Note that, in general, triggers can be tapped from any line in the circuit. Nine test vectors $\{P1, P2, \cdots, P9\}$ are generated using the ATPG tool TetraMax [96] as manufacturing tests to provide 100% stuck-at fault (SAF) coverage. As long as the Trojan is not activated by these test vectors, the circuit will pass the production test. For this example, we have Trojan activation pattern, $TAP = (101X0X)^T \notin \{P1, P2, \ldots, P9\}$, where "X" denotes *don't care* state. As the order of this Trojan ($n = 4$) is less than the number of PI (6), the Trojan may be activated by multiple input patterns. It is thus necessary to verify that the trigger output (x) remains 0 for all test vectors ($P1, P2, \ldots, P9$).

One can deliver the payload at any site, where a SAF is detected by the TAP. Because the two X's can be enumerated in four ways, the TAP 101X0X corresponds to four vectors. Simulating [96] two SAFs at the payload site n_7 we find that 101001 detects $sa1$ and the other three patters, 101000, 101100 and 101101 detect $sa0$ at n_7. Thus, the Trojan in Figure 5.2 will produce four errors at y_2, $0 \rightarrow 1$ for 101001 input and $1 \rightarrow 0$ for 101000, 101100 and 101101.

In this example, the Trojan delivers the payload at signal n_7, where a SAF fault is detected by one or more TAPs. By simulating any TAP, alternative locations for delivering the payload can be found. Figure 5.2 shows SAFs detectable at x_1, x_2, x_6, n_1, n_2, n_3, or n_5 the TAP 101000. Thus, any of these lines can be used as an alternative payload site.

Table 5.1: Modeled Hardware Trojans in Circuit of Figure 5.2.

Trojan Category	**Type-1**	**Type-2**	**Type-3**	**Type-4**
All possible Trojans (Eq. 5.2)	612	9,792	97,920	685,440
Feasible Trojans	605	8,097	60,905	294,538
Trojans removed by SAF tests	586	6,985	43,852	17,4114
Valid Trojans	19	1,112	17,053	120,424

5.1.2 Finding All Type-*n* Trojans

An upper bound on the number of Type-*n* Trojans (T_n) is given by:

$$T_n \leq \begin{pmatrix} K \\ n \end{pmatrix} \times 2^n \times (K - n) \tag{5.2}$$

where K is the number of lines in the Trojan-free circuit. From Equation 5.1, $K = 18$ for the circuit of Figure 5.2. Numbers of all possible Trojans of types 1 through 4, computed from Equation 5.2, are shown in Table 5.1. Note that the number of Trojans goes up by one order for each higher type. However, all Trojan structures (payload and trigger) do not modify the truth table of the circuit; when trigger is active, a path from payload site to PO may or may not be sensitized. Those modifying the truth table are shown as feasible Trojans in Table 5.1. These were determined using the exhaustive set of $2^6 = 64$ patterns, a fault simulator [96] to identify sensitized paths, and a logic simulator [91] to examine the trigger states. We find that a significant number of the feasible Trojans is detectable by the set of nine SAF manufacturing test patters $P1$ through $P9$ shown in Figure 5.2. We do not consider those as valid Trojans because chips containing them will be eliminated during production testing. Removing them from feasible Trojans gives us the number of valid Trojans.

Although the Type-n Trojan model seems general, even for a small circuit $(K = 18)$, the number of valid Trojans grows rapidly with n. For generating tests for Trojan detection and for coverage analysis, we will use Type-1 Trojans assuming their number equals the upper bound of Equation 5.2:

$$T_1 = 2K(K - 1) \tag{5.3}$$

Algorithm 3: Design of a Type-n Trojan.

```
input  : Circuit Netlist ($C$), Manufacturing test patterns ($P$), Order of a Trojan ($n$)
output : Trojan activation pattern ($TAP$), Trigger Inputs ($T$)

1  Read the netlist ;
2  Read manufacturing test patterns ($P$);
3  Select a random pattern as Trojan activation pattern, $TAP \notin P$;
4  Perform logic simulation using $P$ to obtain all internal node values ($M_K$);
5  Perform logic simulation with $TAP$ to obtain all internal node values ($S_T$);
6  Select a $n$ random locations of the netlist to form the trigger inputs ;
7  Form a new matrix $M_n$, $M_n \leftarrow mod(M_K)$ ;
8  if $S_n \in M_n$ then
9      Drop the selection as it will activate the Trojan;
10     Go to Step 6;
11 else
12     Choose $T$ as trigger input;
13 end
14 Perform fault simulation and logic simulation with $TAP$ ;
15 Select a fault site (from Step 14) for delivering the payload, i.e., Trojan location.
```

This number of target Trojans is $O(K^2)$, or quadratic in circuit size, K. The number would be $O(K^{n+1})$ for Type-n Trojans, where $n \leq K - 1$. Thus, our methodology parallels SAF whose tests are known to detect multiple stuck-at and many other types of faults [33].

Trigger circuitry of a Type-n Trojan model is an n-inputs AND gate. The payload is delivered through an XOR gate to any site activated by the Trojan activation pattern (TAP).

Algorithm 3 designs a Type-n Trojan that will not be activated the manufacturing tests. The inputs are original netlist (C), manufacturing test patterns (P), and the order of the Trojan (n). The algorithm reports the TAP and trigger inputs (T). It reads the Trojan free circuit netlist and manufacturing test patterns (Lines 1-2). A random TAP is selected (Line 3), which is not present the manufacturing test pattern set ($TAP \notin P$). Logic simulation gives the internal node values (M_K) of the circuit for all manufacturing test patterns (Line 4). M_K is a $K \times p$ matrix where

K and p denote number of circuit nodes and number of manufacturing test patterns, respectively. We simulate the circuit with TAP for all internal node values (S_T). Here, S_T is a $K \times 1$ vector. Select n locations from K nets, either randomly or by some given criterion, to form an $(n \times 1)$ vector S_n from S_T (Line 6). The $mod()$ function returns a new matrix M_n corresponding to the selected n nodes (Line 7). Selection of these n nets is not valid if $S_n \in M_n$, as one of the test pattern will trigger the Trojan (Line 9), so select a new set of n nodes (go to Step 6). Finally, fault simulation with TAP gives possible sites for payload (Lines 14-15).

5.2 Test Generation for Type-*n* Trojans

Definition 3. CSP-n, CSP-1 or CSP, and CSP-0: For a signal s in a digital circuit, two *type-n conditional stuck-at fault (SAF) patterns* (CSP-n) detect $sa0$ and $sa1$ faults, respectively, while setting specified [0,1] values on n other signals $t_1, \cdots t_n$. CSP-1, or simply CSP, are conditional tests with a condition on a single signal. CSP-0 are tests without any condition and are identical to the classical SAF tests.

Type-n HT's with signal s as payload are detectable by CSP-n we denote as $(s \text{ sa0} \mid C_1, \cdots C_n)$ or $(s \text{ sa1} \mid C_1, \cdots C_n)$, where $C_i = t_i$ or $\overline{t_i}$. CSP-n is a generalization of the CSP-1 defined in the literature [135].

Clearly, a Type-n Trojan is detectable by any of the two CSP-n's for which s is the payload site and $t_i's$ are trigger inputs. For example, the Type-4 Trojan in Figure 5.2 is detected by CSP-4 $(n_7\ sa0 \mid x_1, \overline{x_2}, x_3, \overline{x_5})$ and $(n_7\ sa0 \mid x_1, \overline{x_2}, x_3, \overline{x_5})$. Considering complexity, we restrict to test generation for CSP-1 and evaluate their coverage for higher type of Trojans.

5.2.1 Conditional SAF Pattern (CSP-1 or CSP) Generation

Algorithm 4 generates conditional SAF patterns (CSP) for detecting hardware Trojans. It is necessary to determine the total number of nets (K) according to Equation 5.1 (Line 2). The algorithm initializes CSP-n as an empty set (Line 3) and then iterations with their signal values ($Nets$) are stored in $TempNets$. A CSP-1 is generated for a specific SAF with signal values for specific nets (Line 11). Note that, a CSP-n generation capability in the ATPG tool TetraMAX [96]

Algorithm 4: Conditional stuck-at fault (SAF) pattern generation for hardware Trojan detection.

```
input  : Circuit Netlist, $C$
output : Conditional SAF detection pattern set, CSP

1  Read the netlist $C$ ;
2  Determine number of nets in $C$, $K \leftarrow \#PIs + \#Gates + \#fanout\_branches$;
3  Initialize empty set of conditional SAF patterns, CSP $\leftarrow \phi$ ;
4  Initialize count $c \leftarrow 0$ ;
5  for $i \leftarrow 0$ to $K$ do
6      for $j \leftarrow 0$ to $1$ do
7          for $k \leftarrow 0$ to $1$ do
8              Initialize $TempNets \leftarrow Nets$ ;
9              Initialize count $l \leftarrow 0$;
10             while $TempNets \neq \phi$ do
11                 $CSP[c] \leftarrow$ Test pattern for stuck-at $j$ fault with $net_l = k$ ;
12                 Invoke logic simulation with $CSP[c]$ to get internal node values ;
13                 Remove nets with signal
                     value from $CSP[c]$, $TempNets \leftarrow update(TempNets)$;
14                 $c \leftarrow c + 1$, and $l \leftarrow l + 1$ ;
15             end
16         end
17     end
18 end
19 Report CSP for Trojan detection;
```

is invoked by specifying an SAF target with n other signals and their values. Logic simulation is performed with this pattern to find the internal node values (Line 12). All {net, signal value} pairs corresponding to this pattern are dropped from $TempNets$ (Line 13). Repeat this process until $TempNets$ is empty (Line 10). Once all iterations are complete, the algorithm reports CSP-n's for Type-n Trojan detection.

5.2.2 An Example: Circuit of Figure 5.2.

Once again, considering the high complexity due to large number of higher type of Trojans, explained in Section 5.1 C, we generated tests for all 612 Type-1 Trojans using Algorithm 4. As a result, 48 CSP-1 detected 605 feasible Trojans confirming the data in Table 5.1. Without confusion, we simply call them CSP. Assuming that 9 SAF vectors would have already tested for 586 Trojans during production, those were removed leaving as set of 48 vectors that detect all 19 valid Type-1 Trojans. Manufacturing tests are applied during production to all chips to eliminate defective ones. Assuming that a Trojan remains undetected (we define this as a *valid Trojan*, all passing chips must have the same Trojan. Hence it is sufficient to test just one chip for Trojans and the Trojan tests can be much longer than the manufacturing tests. For any type (n), the quality of Trojan tests is their coverage of valid Trojans, which, in turn depends on the manufacturing tests. Thus,

$$\text{Trojan Coverage} = \frac{\#\ of\ detected\ valid\ Trojans}{\#\ of\ all\ valid\ Trojans} \times 100\ \% \tag{5.4}$$

We generated two other sets, each with 48 vectors, an N-detect set and a random set. Valid Trojans of types 1 through 4 (Table 5.1) were simulated. Results are given in Table 5.2 (Rows 1, 4, 7 and 10). Coverage of CSP was always higher and dropped slower with increasing n. Four Type-2 Trojans, $(x_5 \mid \overline{x_2}, x_4)$, $(x_5 \mid x_4, n_5)$, $(n_4 \mid \overline{x_1}, x_4)$ and $(n_3 \mid x_1, x_4)$, were only detected by CSP. Their payloads are closer to PI. Besides, they indicate superior capability of CSP in covering trigger combinations.

5.3 Benchmark Circuits

To study the effectiveness of the proposed conditional SAF patterns (CSP), we used a simulation setup for ISCAS 85 benchmark circuits [120]. TetraMax [96] provided manufacturing tests for each circuit covering 100% of all detectable SAFs. Next, we generate the CSP for each circuit using Algorithm 4. As the number of valid Trojans is large, we perform the coverage analysis of CSP based on four random sample sets of 20,000 Trojans of Type-1 through Type-4, respectively. Some Trojans cannot be triggered from inputs, nor do they affect outputs. Excluding these, we get feasible

Table 5.2: HT Test Coverage (%) of Valid Trojans (V_n).

Trojan type	**Circuit**	**Lines, K Eq (5.1)**	**All Trojans T_n, Eq (5.2)**	**SAF tests**	**Valid Trojans V_n, Eq (5.5)**	**HT tests**	**V_n Coverage (%)**		
							CSP	**N-det.**	**RDM.**
Type 1	Fig. 5.2	18	612	9	19	48	100	100	100
	c432	307	187,884	69	16,684	99,991	100	96.86	70.29
	c880	577	664,704	76	152,583	531,846	96.25	95.88	94.81
Type 2	Fig. 5.2	18	9,792	9	1,112	48	100	99.60	99.73
	c432	307	5.73×10^{7}	69	1.27×10^{7}	99,991	93.48	89.71	61.93
	c880	577	3.82×10^{8}	76	1.22×10^{8}	531,846	93.72	92.98	91.62
Type 3	Fig. 5.2	18	97,920	9	17,053	48	99.80	98.46	97.64
	c432	307	1.16×10^{10}	69	3.96×10^{9}	99,991	89.62	83.89	59.16
	c880	577	1.46×10^{11}	76	6.25×10^{10}	531,846	90.08	89.99	88.06
Type 4	Fig. 5.2	18	685,440	9	120,424	48	99.30	97.40	95.95
	c432	307	1.76×10^{12}	69	7.27×10^{11}	99,991	85.28	78.67	53.11
	c880	577	4.19×10^{13}	76	2.176×10^{13}	531,846	87.87	87.51	85.62

Trojans. Feasibility within each sampled set was assessed using the TetraMax conditional ATPG capability [96]. Some feasible Trojans are detectable by the manufacturing tests. Excluding those, we get valid Trojans (v_n), any of which an adversary may insert in the netlist. It is economical to estimate the total number of valid Trojans (V_n) in a circuit based on the 20,000-Trojan sample. This sample size is large enough for reasonable accuracy [33]. Total number of valid Trojans is,

$$V_n = \frac{v_n}{20,000} \times T_n \tag{5.5}$$

Table 5.2 shows the results. Trojan sampling was not used for the circuit of Figure 5.2 (Rows 1, 4, 7 and 10). Next, Rows 2, 5, 8 and 11 show data for $c432$ benchmark. For $K = 307$, Equation 5.2 gives $T_1 = 187,884$ Type-1 Trojans (column 4). This circuit has 712 SAFs detected by 69 manufacturing test patterns (column 5). Algorithm 4 generated 99,991 CSP, beyond 69 SAF patterns, shown as HT tests in column 7. From 187,884 Type-1 Trojans, we take a random sample of 20,000 Type-1 Trojans to estimate the Trojan coverage. Among these, 513 Trojans could not be triggered from inputs, leaving 19,487 feasible Trojans. In addition, 17,711 Trojans were detected by 69 SAF patterns. Hence, number of valid Trojans, $v_n = 19,487 - 17,711 = 1,776$. From Equation 5.5, number of valid Type-1 Trojans, $V_n = 16,684$ (column 7). Next three columns of row 2 give Type-1 Trojan coverage by CSP, N-detect patterns, and random patterns, respectively, each containing 99,991 patterns. Similarly, results for Trojans of Type 2 (row 5), Type 3 (row 8) and Type 4 (row 11) were obtained. Notably, the CSP coverages are consistently higher.

Results for $c880$ benchmark in rows 3, 6, 9 and 12 were obtained in a similar manner with one exception. The number of HT tests is 531,846 and will grow significantly larger for bigger circuits. We randomly sampled 5,000 patterns from 531,846 HT tests to estimate the coverage of Trojans of Types 1 through 4 [136]. The results are given in columns 8-10 (rows for $c880$). Once again, CSP coverages are higher.

5.4 Summary

The Type-n Trojan is a generalized model that facilitates test generation and coverage analysis. A Consideration of the complexity issue leads to the Type-1 Trojan model and its test by conditional stuck-at fault patterns (CSP). Thus, the number of Trojans to be modeled is $O(K^2)$ for a circuit with K signal lines. Although the detection coverage is measured over valid Type-1 Trojans, not detectable by manufacturing tests, tests are generated for all Type-1 Trojans. This is because our "real" targets include higher types as well. We find that both Trojan sampling and vector sampling are beneficial for coverage estimates. For larger circuits, CSP generation for a random sample of Type-1 Trojans may also be used [137].

According to this chapter, by model conditional stuck-at faults, HT can be effectively detected. A fault modeling can help predict the consequences of particular faults and make an analysis of the circuit possible. Fault modeling is an analyzable approximation of defects and is essential for a test methodology. Fault modeling is not only essential in conventional CMOS circuits but also plays a pivotal role in emerging technologies. In emerging technologies, if these unique defects can be mapped to the fault model in the existing CMOS, then the existing ATPG tool can be used to detect these emerging technologies without having to pay more effort. In the next chapter, a defect characterization and testing method for the skyrmion-based logic circuit will be introduced in detail [138].

Chapter 6

Defect Characterization and Testing of Skyrmion-Based Logic Circuits

Spintronic devices offer a feasible choice for post-Moore devices [139, 140]. Magnetic skyrmion is a possible choice for implementing various logic designs and non-volatile memories [141, 142]. It has been experimentally demonstrated that skyrmions can be stabilized in various material systems, including noncentrosymmetric chiral-lattice magnets such as MnSi/MnGe and $Fe_{0.5}Co_{0.5}Si$ [143, 144] as well as at heavy metal/perpendicular magnet interfaces with strong Dzyaloshinskii–Moriya interaction (DMI) [145, 146]. They can also be created, moved and annihilated by magnetic fields and low electrical current pulses [147]. Though the non-linear motion caused by skyrmion Hall effect poses an issue with skyrmions, their properties such as nanometer diameter for high-density storage, room-temperature stability, current-controlled motion, topological charge, and protection against large defects have made skyrmion-based devices promising candidates for beyond-Moore systems [148, 149].

Skyrmion logic gates utilize effects of skyrmion movement to implement such functions as spin-orbit torque-induced motion [150, 151], skyrmion Hall effect [152–156], skyrmion-edge repulsion [157, 158], and voltage control of magnetic anisotropy [159–161]. Skyrmion-based gates are also known to implement reversible computing [162].

Due to minimal power consumption and small physical size, a spin-based device like magnetic skyrmion is a promising candidate for a beyond CMOS technology. We believe a significant amount of work is still necessary to make these devices feasible for commercial applications. To the best of our knowledge, little is available on their testing [163]. In manufacturing, a wide variety of defects may occur, and finding tests for them is often impractical. Thus, modeled faults serve as the basis for tests, and their coverage measures the effectiveness of tests in detecting the manufacturing defects. Main contributions of this chapter are as follows:

- *Defect characterization:* As technology-specific defects, we examine breaks, extra material, etching blemishes, bridges among interconnects, and a set of 19 physical defects in the skyrmion gate structures. *We believe we are the first to characterize such defects using magnetic simulation.*
- *Developing fault models:* Each defect is simulated for an exhaustive set of signals to map it onto an analyzable fault model using the principle of fault equivalence [33]. Thus, each defect is represented by either a technology-independent single stuck-at fault or a technology-dependent fault.
- *Testing of skyrmion-based circuits:* Single stuck-at, transition and certain bridge faults are directly analyzable by the available tools. Others are, as far as possible, represented by combinations of analyzable faults. This chapter gives test generation results for skyrmion versions of benchmark circuits for defects that could be represented as single stuck-at faults. The ongoing future research will address other faults, shown to require complex representations.

The approach outlined above can be extended to other emerging circuit technologies (e.g., memristors [164]). This chapter is organized as follows. The background of skyrmion-based designs is provided in Section 6.1. Physical defects is characterized, likely to appear during the manufacturing process, in Section 6.2 using magnetic simulation. A fault models for skyrmion circuits is presented in Section 6.3. In Section 6.4, the testing strategy for detecting manufacturing defects is developed, with some results given in Section 6.5. Finally, the chapter is concluded in Section 6.6 .

6.1 How Skyrmion Works?

In this section, we will study the movement of skyrmion in a nanotrack and simulate basic logic gates.

6.1.1 Skyrmion Motion in Nanotrack

Skyrmion is a stable magnetic field that acts like a pseudoparticle. It moves through a structure called *nanotrack*, which is made of a heavy metal (HM) layer and a ferromagnetic (FM) layer [162]. Figure 6.1 shows the 3D cross sectional view of a nanotrack consisting of a FM layer (gray) and a HM layer (green). The HM layer is wrapped around by the FM layer at the top and two sides.

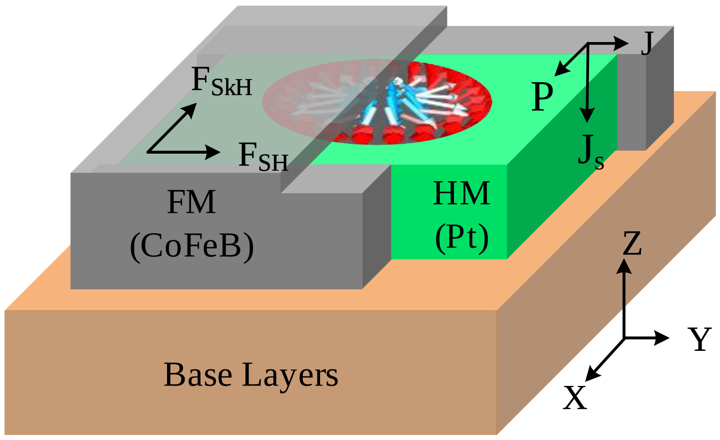


Figure 6.1: Nanotrack structure for skyrmion movement.

The skyrmion can be hosted at the FM/HM interface due to the strong interfacial DMI [145, 146]. Simulation shows that the side wall wrapping eliminates transverse motion of the skyrmion caused by the Hall effect, allowing only linear motion. The dimensions of the FM and HM layers can be controlled based on the device geometry. The HM layer consists of platinum (*Pt*), and the FM layer consists of *CoFeB*.

To drive a skyrmion in the nanotrack, a continuous electric current J is required in the HM layer. Due to spin Hall effect, J generates a spin current J_s in the FM layer. At the FM/HM interface, the spin current applies a spin-orbit torque to the skyrmion, driving it along y-axis. At the same time, Hall effect tends to move the skyrmion transversely along x-axis. However, the transverse motion is prevented by the HM side wall.

The dynamics of skyrmion is governed by the Landau–Lifshitz–Gilbert–Slonczewski equation [165]:

$$\frac{dm}{dt} = -\left|\gamma\right| m \times H_{eff} + \alpha \left(m \times \frac{d_m}{d_t} \right) + \tau_{SOT} \tag{6.1}$$

where *m* is the normalized magnetization M/M_s, M being magnetization and M_s the saturation magnetization. H_{eff} is the effective magnetic field associated with magnetocrystalline anisotropic energy and the DMI energy [145, 146]. Further, γ is gyromagnetic ratio, α is damping coefficient, and τ_{SOT} represents the spin-orbit torque determined by multiple parts, namely, gyromagnetic ratio, effective field, spin polarization rate, permeability of vacuum, driving current density and thickness of magnetic film.

A skyrmion inside nanotrack is driven by a current flowing in the heavy metal (HM) layer via spin orbit torque (SOT). The forces on the micromagnetic skyrmion can be modeled by Thiele equation [166]:

$$G \times v - \alpha D \cdot v + F_{SOT} - \nabla V = 0 \tag{6.2}$$

where the first term describes the Magnus force, G is the gyromagnetic coupling vector, and *v* is velocity of the skyrmion. The second term represents a dissipative force, α is the damping coefficient, and D is the dissipative tensor. The third term represents the driving force F_{SOT} generated by the spin Hall effect. The last term gives the resultant force acting on the skyrmion, and V is the confining potential due to boundaries, process impurities and other textures.

6.1.2 Micromagnetic Simulation Platform

The micromagnetic simulation tool $MuMax^3$ is a GPU-accelerated program that analyzes the dynamic behavior of skyrmions [167]. Skyrmion movement in the track is modeled based on equation 6.2 as the electrical current in the HM layer drives the skyrmion. Parameters used in simulation are: Gilbert damping factor α = 0.3, nonadiabatic STT factor $\beta = 0.1$, exchange stiffness $A_{ex} = 15 \times 10^{-12}$ J/M, perpendicular magnetic anisotropy $K_u = 0.6$ MJ/m^3, saturation magnetization $M_s = 5.8 \times 10^5$ A/m, and DMI constant = 3.5 mJ/m^2. Mesh sizes are $1nm \times 1nm \times 0.4nm$, along x, y and z axes. The parameters here are typical values for the magnetic layers [168].

6.1.3 Skyrmion Logic Gates

A traditional skyrmion gate combines phenomena such as spin Hall effect, skyrmion Hall effect, skyrmion-skyrmion repulsion and skyrmion curb repulsion [158, 169]. We have adopted the reversible gates of Friedman and coworkers [162] by modifying them as non-reversible logic gates. We simulated the basic two-input AND and OR gates, and an inverter. In addition, in this technology a specific fanout element is needed. Figure 6.2 shows the gates and fanout. For simplicity, we only show the top view of the nanotracks with the bottom HM layer and top FM layer to illustrate the design of gates. Other gates including complex gates can be similarly constructed.

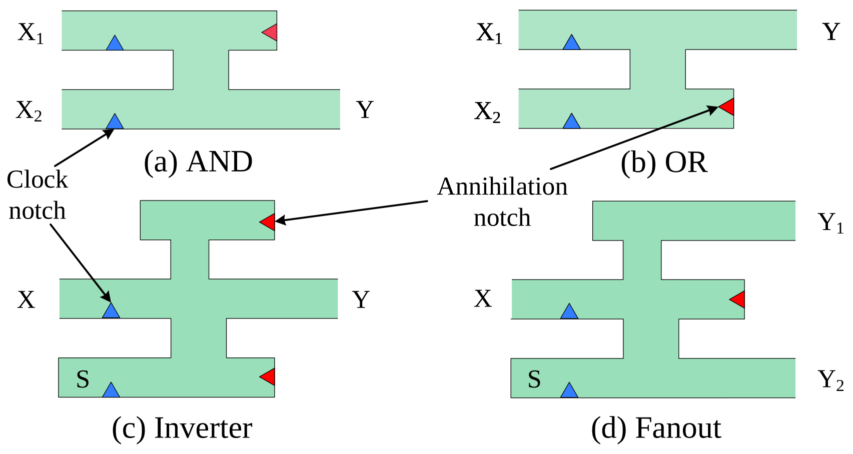


Figure 6.2: Structure of skyrmion gates. (a) AND gate (b) OR gate (c) Inverter, and (d) Fanout.

Figure 6.2(a) is an AND gate consisting of two nanotracks with a junction. This makes the gate a transversely H-shape structure. The blue triangle on the inputs side is a clock notch to synchronize the input skyrmions so that the output of the gate is properly evaluated based on skyrmion-skyrmion interaction. The clock notch has the same material like the ferromagnetic layer. One can also implement voltage controlled magnetic anisotropy (VCMA) structure to synchronize the skyrmion [170]. The clock notch can hold/block the skyrmion movement when a standard current is applied. When a high current pulse is applied, the skyrmions can simultaneously cross the notchs, ensuring proper logic operation of the gate. The red triangle at the end of the upper nanotrack is an annihilation notch, which eliminates any arriving skyrmion. The three forces mentioned above, two of which shown in Figure 6.1, are responsible for the operation of the gate: (1) Spin-Hall force F_{SH} moves skyrmion along the nanotrack toward output (shown on the right in our diagrams); (2) Skyrmion-Hall force F_{SkH} moves skyrmion from one to other nanotrack whenever a junction becomes available; and (3) Skyrmion-skyrmion repulsion prevents the movement when another skyrmion is present in the other nanotrack.

For OR gate we swap the output (Y) and the annihilation (X_1) tracks of the AND gate as shown in Figure 6.2(b). Figure 6.2(c) gives the structure of an inverter. We need to add a source S, where a skyrmion is injected every clock cycle. Our source (S) is the same as the control $(C = 1)$ used by others [162]. The inverter can be regarded as an OR gate with an annihilation track added through a junction. For $X = 1$, skyrmion from X prevents the upward movement of the skyrmion from S, which is then annihilated. Meanwhile, the skyrmion from X goes up and is annihilated as

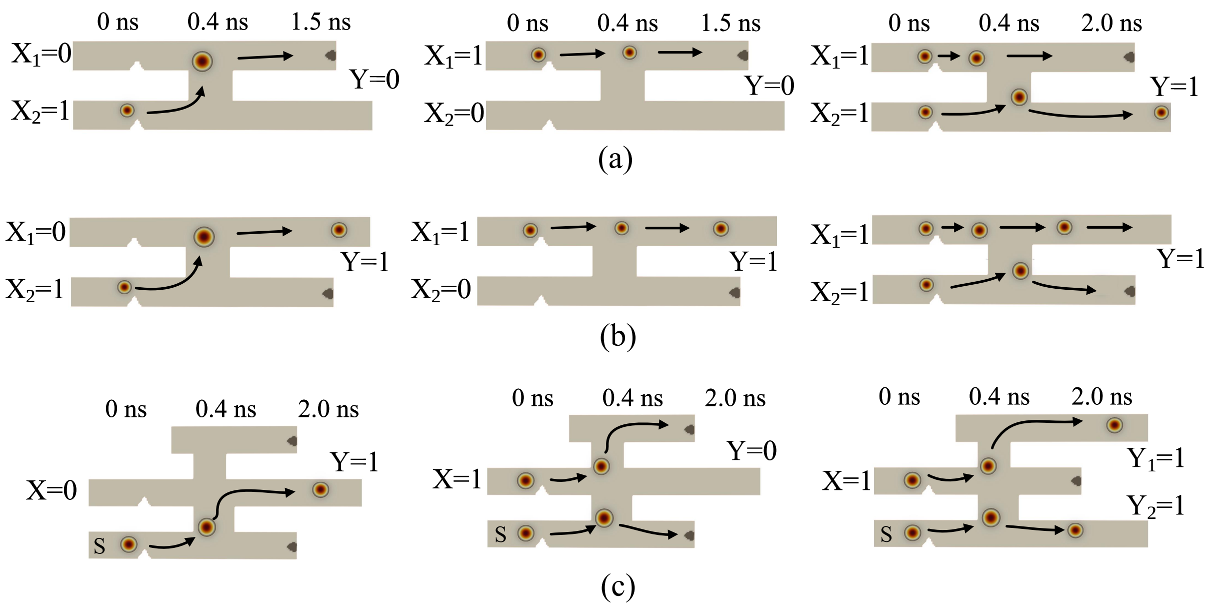


Figure 6.3: Simulation of skyrmion gates: (a) AND, (b) OR, and (c) Inverter and fanout.

well, leaving the output Y with no skyrmion, i.e., at logic 0. If $X = 0$, there will be no skyrmion from X to prevent that from S from moving up and appearing at Y.

The inverter is modified as a fanout gate in Figure 6.2(d). Since the source S is always 1, the two nanotracks Y_1 and Y_2 will output 1 only when there is a skyrmion at input X. Otherwise, the skyrmion from S will move to the middle track and get annihilated, leaving no skyrmion in the two output nanotracks.

Figure 6.3(a) shows the simulation of AND gate with various input combinations. Before clock pulse arrives, the skyrmions are held at the clock notch. After the clock arrival, the skyrmions cross the notches and keep moving in respective nanotracks. For inputs $X_1 = 0$ and $X_2 = 1$ the lower skyrmion will travel up through the junction under skyrmion Hall effect and will be annihilated. "Upper" and "lower" here refer to the left and right nanotracks when viewing in the direction of skyrmion motion. When $X_1 = 1$ and $X_2 = 0$, the skyrmion in the upper nanotrack will be directly annihilated. When the input pattern is 11, skyrmions will meet in the middle of the junction but skyrmion-skyrmion repulsion will keep them in their original tracks, sending the upper one to annihilation and the lower one to the output Y. Not shown is the simulation of input 00, which has no skyrmion and hence no action occurs, leaving $Y = 0$.

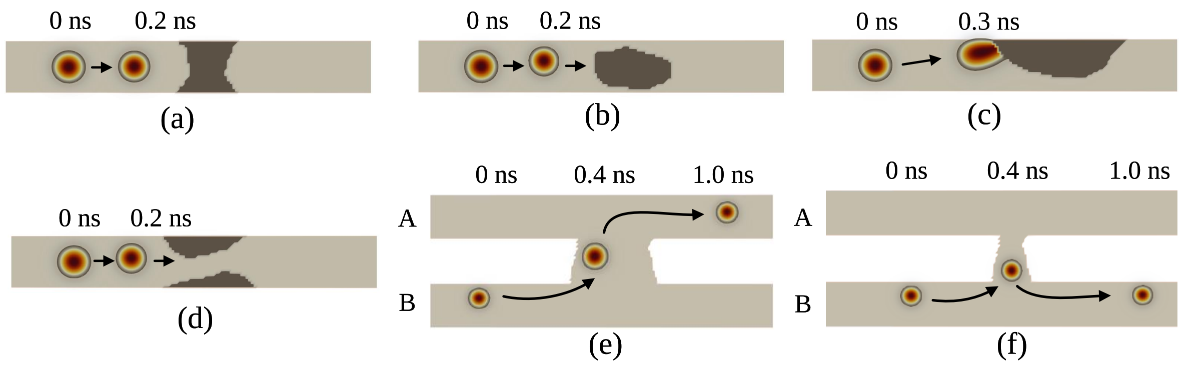


Figure 6.4: Simulation of interconnect defects: (a) break, (b) void, (c) and (d) etching blemishes, (e) wide bridge, and (f) narrow bridge. Defects (a) through (d) map onto stuck-at faults, (e) causes a bridging fault, and (f) causes no fault.

Figure 6.3(b) shows the simulation of OR gate. Figure 6.3(c) shows the simulation of inverter and fanout gates.

6.2 Defect Characterization

This section explores the testability aspects of skyrmion circuits with physical defects using $MuMax^3$ tool [167].

6.2.1 Interconnect Faults

Similar to other technologies (e.g., CMOS), these defects are not associated with gate implementations. However, the skyrmion interconnects are nanotracks and not simple wires. Also, logic 1 or 0 state is represented by presence or absence of a single skyrmion and not by high or low voltage. Not considering any influence of these two attributes, the interconnect defects can be regarded as technology-independent.

The interconnect defects include material void, crack in a nanotrack, and bridging between the two nanotracks. Figures 6.4(a)-(f) show snapshots of the skyrmion movement at three different simulation times. A break in the nanotrack is shown in Figure 6.4(a) such that the skyrmion cannot move along the nanotrack. A hollow structure/void appears on nanotrack and is shown in Figure 6.4(b). Although the FM or HM layers are not completely disconnected, the skyrmion still cannot propagate through the void. However, the effect of a void defect will be different depending

Table 6.1: Wide interconnect bridge of Figure 6.4(e).

Correct inputs and outputs		**Faulty outputs**	
A	**B**	**A**	**B**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

on the speed of skyrmion. When the speed is low, the skyrmion will stop before the void defect. When the skyrmion is moving at a high speed, it will collide with the void and vanish. As shown in Figures 6.4(c) and (d), there are etching blemishes on one or both surfaces of the nanotrack, respectively. The uneve surface will block the propagation and destroy the skyrmion. Thus, all of these defects can potentially cause stuck-at-0 faults. We have not found any defect that could permanently trap a skyrmion in an interconnect to cause a stuck-at-1 fault.

Another possible defect is a bridge between two nanotracks, possibly where nanotracks are physically close to each other. As skyrmions move in one direction, which is along the electric field, it is unlikely that a feedback bridging will occur. Figures 6.4(e) and 6.4(f) show two types of bridging defects. When the bridge is wide, the skyrmion will cross over to the upper nanotrack as shown in Figure 6.4(e). However, a narrow bridge will not affect the skyrmion movement and will not produce incorrect response as shown in Figure 6.4(f). Interconnect response for wide bridge is, as Table 6.1 shows, logical OR, i.e., A+B, for interconnect A and logical AND, i.e., AB, for interconnect B. This differs from the conventional OR-bridging or AND-bridging faults [33]. We classify the interconnect bridge in a skyrmion circuit as technology-specific because the circuit layout must determine which interconnect will be OR and which will be AND.

6.2.2 Technology-Specific Defects in Gates

The defects inside a gate are technology-specific because their bahavior depends upon the device characteristics and gate structure. We present a comprehensive taxonomy of defects in two-input functions, $Y = X_1X_2$ and $Y = X_1 + X_2$, inverter, $Y = \overline{X}$, and fanout $(Y_1, Y_2) = X$, described

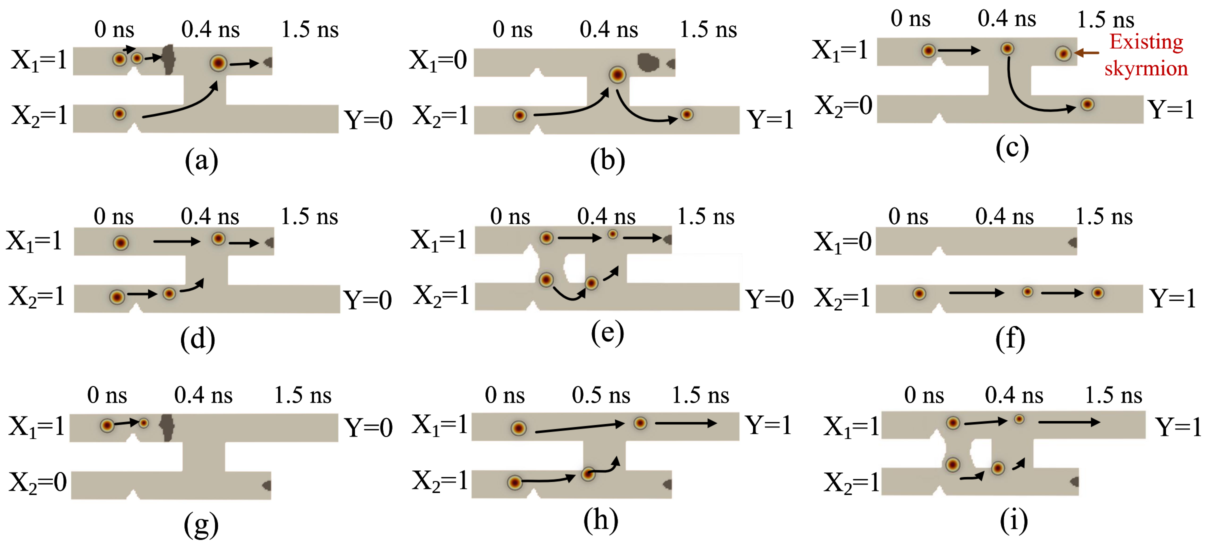


Figure 6.5: Some technology-specific defects in skyrmion gates. AND gate - (a) through (f), OR gate - (g), (h) and (i).

below as T_1 through T_{19}. These were analyzed as single defects by a technology simulator [167] and the faulty outputs for exhaustive set of inputs are recorded in Table 6.2:

- $\boldsymbol{T_1 - T_5}$: Breaks at different locations in nanotrack effect the gate function differently. For example, if the break is located before the junction (Figure 6.5(a) for AND gate or Figure 6.5(g) for OR gate), repulsion from the break will change the original trajectory of the skyrmion. This repulsion may either stall a slow moving skyrmion or destroy a fast moving skyrmion. Defects T1 through T4 are breaks at X_1, X_2, Y, or the dummy channel (with annihilation notch) in AND or OR gate. For an inverter or fanout having an additional nanotrack, only the T5 of Figures 6.6(a) and 6.6(g) are considered.
- $\boldsymbol{T_6 - T_{10}}$: A void may appear at different locations in the nanotrack. Unlike T_1 through T_5, a void can sometimes change the trajectory of skyrmion due to the skyrmion-edge repulsion. Defects T_6, T_7 and T_8 are voids in tracks X_1, X_2 and Y, respectively. T_9 in Figure 6.5(b) corresponds to a void in the dummy track. For inverter and fanout gates, defects T_6 through T_{10} represent voids in nanotracks X, S, Y, Y_1 and Y_2 shown in Figures 6.6(b) through 6.6(h), respectively.
- $\boldsymbol{T_{11} - T_{13}}$: The annihilation notch of a gate can be absent due to T_{11}. The skyrmion in the previous computation will not vanish as expected and the skyrmion-skyrmion repulsion due to the old skyrmion will alter the original trajectory of the skyrmion, as shown in Figure 6.5(c).

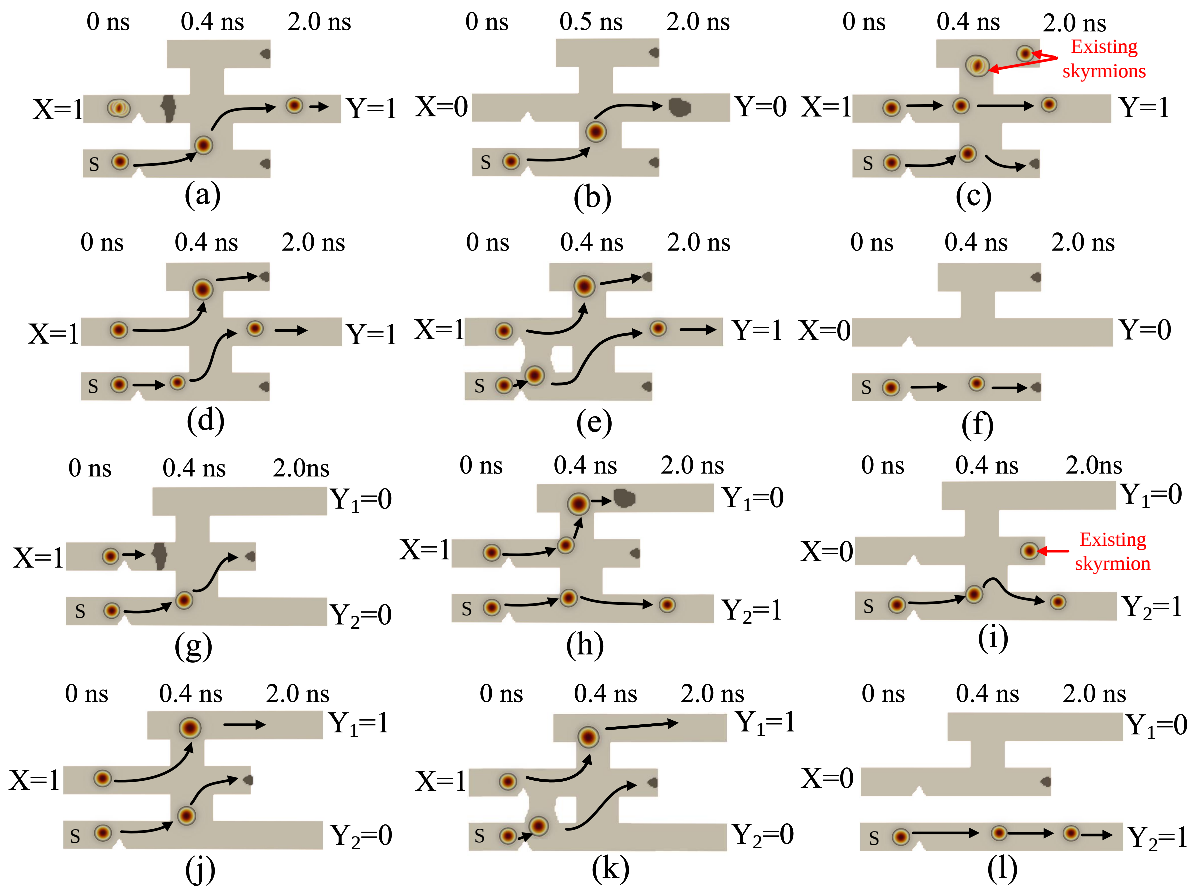


Figure 6.6: Technology-specific defects in skyrmion inverter and fanout gates.

Also, T_{11} and T_{12} represent missing annihilation notches from upper and lower nanotracks of the inverter in Figure 6.6(c). T_{13} is the missing notch of fanout gate in Figure 6.6(i).

- $\boldsymbol{T_{14} - T_{15}}$: These are missing clock notches, or the defects where the clock is absent at input tracks X_1 and X_2, respectively. Due to these defects, skyrmions enter logic gate at different times and the synchronization mismatch can cause logic malfunction. Figures 6.5(d) and 6.5(h) and Figures 6.6(d) and 6.6(j) show T_{14} defects in four types of gates, respectively.
- $\boldsymbol{T_{16}}$: T_{16} is a bridging defect between two input tracks of a gate, after the clock notch. This will cause the skyrmion to either change speed or directly pass through the bridge, thereby affecting the function of the gate. Figures 6.5(e) and 6.5(i) show T_{16} for AND and OR gates, and Figures 6.6(e) and 6.6(k) show T_{16} for an Inverter and fanout, respectively.
- $\boldsymbol{T_{17} - T_{18}}$: There can be breaks in the nanotrack that links the two input nanotracks. It might be missing as well. The two possible defects are denoted as T_{17} and T_{18}, respectively. Figure 6.5(f) shows T_{17} for an AND gate. Figures 6.6(f) and 6.6(l) show breaks between middle and lower

tracks of inverter and fanout gate. T_{18} is a missing bridge in the upper track of the inverter and fanout gates.

- $\boldsymbol{T_{19}}$: This is a missing skyrmion source S, which is supposed to produce a skyrmion every clock. T_{19} will cause the inverter and fanout to function incorrectly.

Figure 6.5 shows the magnetic simulation using $MuMax^3$ tool [167] for technology dependent defects in AND and OR gates. We apply an input pattern so that a faulty response can be observed at the output. Figure 6.5(a) shows a break in the nanotrack of input X_1 of an AND gate. The skyrmion at X_2 moves to the upper track and causes a faulty response, i.e., $Y = 0$. Figure 6.5(b) shows a void defect located at the upper nanotrack of AND gate. When the input pattern $X_1X_2 = 01$ is applied, the skyrmion at X_2 is repulsed from the void and produces a faulty response $Y = 1$. As shown in Figure 6.5(c), a missing annihilation notch will cause redundant skyrmion, which was supposed to have been eliminated. The repulsion from the extra skyrmion will cause the new skyrmion to change the original trajectory and the skyrmion from X_1 will enter the lower nanotrack, to cause a faulty response. To detect this defect, it is necessary to provide a skyrmion initially, requiring a two pattern test. Figure 6.5(d) shows a missing clock notch at input X_1, which will cause the input skyrmions to arrive at different times and produce incorrect results. Figure 6.5(e) shows the defect with an additional bridging between X_1 and X_2. The extra bridging part will cause the lower skyrmion to either enter the upper layer or change its speed, thereby changing the function of the gate. Figure 6.5(f) shows the defect with a missing connection between the lower and upper nanotracks, and input $X_1X_2 = 01$ produces an incorrect result. Figures 6.5(g)-(i) show different defects related to an OR gate.

Figure 6.6 shows the magnetic simulation using the $MuMax^3$ tool [167] for technology-specific defects for an inverter/fanout gate. Figure 6.6(a) shows the simulation of a break in the input of an inverter, which will help move the skyrmion in the lower nanotrack to the output nanotrack and produce a faulty response $Y = 1$. Figure 6.6(b) shows the simulation for a void in the output nanotrack. Input pattern $X = 0$ produces a faulty response $Y = 0$. All other defects (Figures 6.6(c)-(l)) can be described based on skyrmion-skyrmion or skyrmion-edge repulsion.

In Figures 6.5 and 6.6, we observe the erroneous outputs under different input conditions. Table 6.2 summarizes the results for all defects under exhaustive input conditions. Column 1

Table 6.2: Exhaustive simulation of skyrmion-based gates with defects.

Gate Type	**Input Pattern**	**Correct Output**	**Output in presence of defect T_i**																		
			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}	T_{19}
AND	00	0	0	0	0	0	-	0	0	0	0	-	0	-	-	0	0	0	0	-	-
	01	0	0	0	0	0	-	0	0	0	1	-	$\ast$	-	-	0	0	0	1	-	-
	10	0	0	0	0	0	-	0	0	0	0	-	$\ast$	-	-	0	0	0	0	-	-
	11	1	0	0	0	1	-	0	0	0	0	-	$\ast$	-	-	0	0	0	1	-	-
OR	00	0	0	0	0	0	-	0	0	0	0	-	0	-	-	0	0	0	0	-	-
	01	1	1	0	0	1	-	1	0	0	1	-	$\ast$	-	-	$\ast$	$\ast$	1	0	-	-
	10	1	0	1	0	1	-	0	1	0	1	-	$\ast$	-	-	$\ast$	$\ast$	1	1	-	-
	11	1	1	1	0	1	-	1	1	0	1	-	$\ast$	-	-	$\ast$	$\ast$	$\ast$	1	-	-
INV.	0	1	1	0	0	1	1	1	0	0	1	1	$\ast$	$\ast$	-	1	1	0	0	1	0
	1	0	1	0	0	0	0	1	0	0	1	1	$\ast$	$\ast$	-	1	1	1	0	1	0
Fan-out	0	00	00	00	00	00	00	00	00	01	00	00	-	-	$\ast$	00	00	10	01	00	00
	1	11	00	10	11	01	10	00	10	11	01	10	-	-	$\ast$	10	10	10	11	01	10

gives the gate type, and columns 2 and 3 provide the input pattern and the correct output response, respectively. Columns 4 through 22 show responses of defective gates. The asterisk "$\ast$" marks the defects that will produce faulty response only with a pair of input patterns. Also, "$-$" denotes a "no fault" response.

6.3 Fault Modeling

To deal with technology-dependent defects, it is beneficial to model them as stuck-at faults, whenever possible, so that we can take advantage of traditional EDA tools. When analyzing the results of Table 6.2, it can be found that some special patterns can detect defects in skyrmion circuits just like they detect stuck-at faults in CMOS circuits. An input pattern 01 applied to an AND gate will produce a faulty output 1 when T_{17} is present. Similarly, the test pattern 01 can be used to detect X_1 and Y_1 stuck-at-1 faults. Also, the input pattern 11 can detect defects T_1 through T_3, T_6 through T_9 and T_{14} through T_{16}. This pattern detects the fault stuck-at-0 at X_1, X_2 and Y. Thus, defects T_1 through T_3, T_6 through T_9 and T_{14} through T_{16} can be modeled as X_1, X_2 or Y stuck-at-0. Note that these three faults are equivalent [33]. The skyrmion-based circuit defects can be mapped onto traditional stuck-at faults of a logic circuit. Table 6.3 shows how we

Table 6.3: Skyrmion gate defect mapping onto single stuck-at faults in AND $(Y = X_1X_2)$ and OR $(Y = X_1 + X_2)$logic gates.

Gate	**Inputs**	**Defects equivalent to single stuck-at faults**					
Type	$\boldsymbol{X_1X_2}$	$\boldsymbol{X_1}$ **sa0**	$\boldsymbol{X_1}$ **sa1**	$\boldsymbol{X_2}$ **sa0**	$\boldsymbol{X_2}$ **sa1**	$\boldsymbol{Y}$ **sa0**	$\boldsymbol{Y}$ **sa1**
AND	01		T_{17}				
	11	T_1, T_2, T_3, T_6 $T_7, T_8, T_{14}, T_{15}T_{16}$		T_1, T_2, T_3, T_6, T_7 $T_8, T_{14}, T_{15}T_{16}$		T_1, T_2, T_3, T_6, T_7 $T_8, T_{14}, T_{15}T_{16}$	
OR	01			T_2, T_7, T_{17}		T_3, T_8	
	10	T_1, T_6				T_3, T_8	
	11					T_3, T_8	

converted the skyrmion defects to equivalent stuck-at faults. First, we find the test patterns for the skyrmion-based circuit that will produce faulty results. Then, for those patterns we list out the stuck-at faults detected in the logic gate of the same function. This way the skyrmion-based defects have been converted into conventional stuck-at fault of CMOS (logic) gates. This internal defect based fault modeling will guarantee that when the circuit is analyzed by a conventional ATPG tool, the tool will generate patterns to detect the defects in the skyrmion-based circuit.

Table 6.4 shows that defects of inverter and fanout can also be represented by stuck-at faults. To detect defects T_{11} through T_{13} (missing annihilation notch), at least two patterns are required. The first pattern is for presetting the skyrmion in missing notch defect and then produce the incorrect operation in the second pattern. These defects cannot be modeled as stuck-at faults. To test defects T_{11} through T_{13}, we model defect T_{11} as a delay fault, for which at least two test patterns are required. Defects T_{11}, T_{12}, T_{16} of inverter and T_{13}, T_{16} of the fanout are considered as technology-dependent faults, Defects T_4, T_5 of inverter and T_3 of the fanout are considered as no faults.

6.4 Test Pattern Generation

To generate patterns for testing of skyrmion logic circuits, it is necessary to first covert a CMOS gate level netlist to skyrmion-based netlist. We use a commercial synthesis tool (e.g., Synopsys Design Compiler [171]) to synthesize the RTL code with specifying cell preferences

Table 6.4: Skyrmion gate defect mapping onto single stuck-at faults in inverter $(Y = \overline{X})$ and fanout $((Y_1, Y_2) = X)$.

Gate Type	**Input X**	**Defects equivalent to single stuck-at faults**			
		X sa0	**X sa1**	**Y sa0, Y_1 sa0, Y_2 sa0**	**Y sa1, Y_1 sa1, Y_2 sa1**
INV.	0		T_2, T_3, T_7 T_8, T_{17}, T_{19}	$T_2, T_3, T_7, T_8, T_{17}, T_{19}$	
	1	T_1, T_6, T_9, T_{10} T_{14}, T_{15}, T_{18}			T_1, T_6, T_9, T_{10} T_{14}, T_{15}, T_{18}
Fan-out	0				T_8, T_{17}
	1	T_1, T_6		$T_1, T_2, T_4, T_5, T_6, T_7, T_9$ $T_{10}, T_{14}, T_{15}, T_{18}, T_{19}$	

with conventional CMOS cell library. In this chapter, Only AND, OR, inverter and fanout gates are used to realize circuits. However, in traditional CMOS circuits, interconnect crossover is common through vias. But, this is not permitted in skyrmion-based circuits.

To achieve crossover in skyrmion-based circuits, an additional element MTJ is required. The magnetic tunnel junction (MTJ) [172] consists of two layers of magnetic metal separated by an ultra-thin insulating layer. The insulating layer is very thin, and if a bias voltage is applied between the two metal electrodes, electrons will pass through the barrier. In MTJ, the tunneling current depends on the relative direction of the magnetization of the two ferromagnetic layers, which can be changed by the applied magnetic field. In spintronics, MTJ is often used to generate or read skyrmion. The addition of element magnetic tunnel junction (MTJ) may also cause defects in the circuit, but because we currently only focus on gate level netlist, the impacts of crossover and MTJ are not to consider at this time. However, our future work will include complex gates (e.g., more than 2 input gates, XOR, and AOI) and node crossover, the overlap of two nanotracks, to mimic the traditional synthesis including MTJ-induced defects. As we only use simple gates, we synthesize circuits with AND, OR and inverter gates. During the synthesis process, we restrict the EDA tool to use only these gates. We use set_prefer command to indicate preferred cells and set_dont_use command to exclude cells from the target library [171]. In addition, skyrmion fanout gates are added, when we encounter a fanout in the gate level netlist.

Table 6.5: Testing stuck-at faults in CMOS and Skyrmion circuits.

Circuit	**Number of collapsed faults**		**Number of test patterns**		**Fault coverage %**	
	CMOS	**Skyrmion**	**CMOS**	**Skyrmion**	**CMOS**	**Skyrmion**
c17	26	16	7	7	100	100
c432	534	461	84	80	100	100
c499	1398	1199	112	109	100	100
c880	982	779	72	62	100	100
c1355	1460	1209	142	125	99.94	99.96
c1908	1262	1060	112	99	100	100
c3540	2536	2272	184	170	100	100
c6288	6766	6538	82	71	100	100

The test pattern generation for skyrmion logic circuits is straightforward. Once the circuit is synthesized and mapped with skyrmion gates, a commercial test pattern generation tool (e.g., Synopsys TestMAX ATPG [173]) is invoked to generate test patterns. It is necessary to add all faults (resulted from both technology dependent and independent defects, see Section 6.2 for details) and to provide this fault list to the ATPG tool. At this point, we only demonstrate test pattern generation for single stuck-at faults, which requires just one test pattern to detect a fault. However, some skyrmion defects (e.g., T_{11}, T_{14}, T_{15} and T_{16}) require two pattern test (like transition delay fault test [33]). Our future work will address the detection of these defects using delay fault tests.

6.5 Results and Discussion

To evaluate the effectiveness of the proposed test generation process, we used Synopsys tools, Design Compiler [171] for synthesis, and TestMAX ATPG [173] for test pattern generation. We used Synopsys 32nm SAED32 EDK Generic Library [97] to synthesize ISCAS'85 benchmark circuits [174]. Table 6.5 shows the results, which include collapsed fault count, test pattern count and fault coverage for skyrmion logic circuits and compares them with traditional CMOS circuits. The first column of this table indicates the benchmark circuit name. The second and third columns

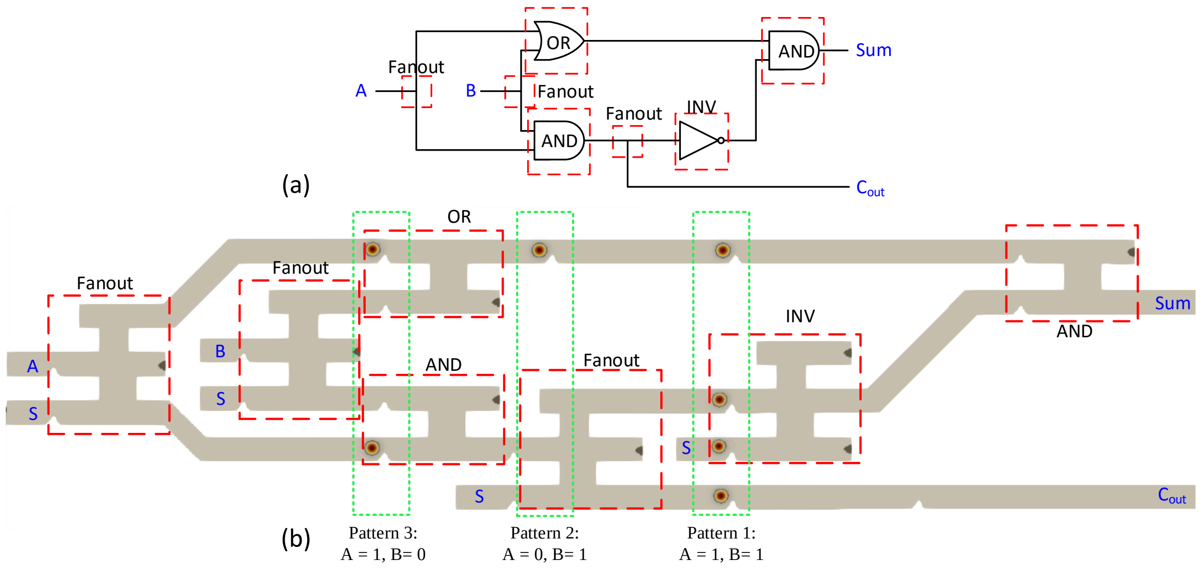


Figure 6.7: Skyrmion circuit design for a half adder.

show the pattern count for both conventional CMOS and skyrmion logic circuits. Fourth and fifth columns show the fault coverages for the same circuit pair. It can be inferred that our pattern generation method for skyrmion logic circuits has reduced the number of test patterns and increased fault coverage compared to the traditional CMOS circuits due to smaller number of stuck-at-1 faults in the netlist.

Figure 6.7 shows a gate implementation of half adder and the design of a skyrmion circuit using AND, OR and fanout gates. Our future work includes designing of NAND, NOR, and complex gates such as 3-input AND and OR, and AND-OR-Invert (AOI) gates so that traditional EDA tool can synthesize a scalable skyrmion circuit.

6.6 Summary

We have considered defect scenarios for skyrmion based digital circuits. We describe these defects under two separate categories, technology-independent and technology-specific defects. The defects include break in nanotrack, extra material, etching blemishes, and bridges between pairs of nanotracks. Those defects are analyzed by exhaustively simulating small structures (interconnects or single gates) using a technology simulator [167]. They are mapped onto analyzable fault models (single stuck-at, for now) using the fault equivalence [33].

The defects are classified into three categories: (1) Technology-independent such as single stuck-at faults; (2) Technology-specific faults that are not analyzable by available tools; and (3) No faults that do not cause error but may lead to aging or latent failure.

ISCAS'85 benchmark circuit results on single stuck-at faults using commercial tools show higher coverage than in CMOS circuits. A possible reason is fewer faults in the skyrmion version, in which many stuck-at-1's do not exist.

Although we have laid the groundwork, the test methodology for the skyrmion circuits is not complete. Remaining work includes technology-specific defects in Table 6.2 that could not be placed in Table 6.3 or 6.4. Included among those are the "no fault" defects whose latent effects must be examined.

Chapter 7

Conclusion and Future Work

This chapter provides the summary of this dissertation and some suggestions for the future work.

7.1 Conclusion of Dissertation

As the density of modern SoC integration grows, most ASIC design companies no longer maintain their foundries. As a result, IP piracy and IC overproduction became a severe issue for the modern semiconductor industry. Over the years, researchers have proposed countermeasures to enable trust in outsourced IC manufacturing, and logic locking has become a popular choice due to its simple structure and low overhead. Thus a circuit will work functionally only when the correct key is applied. However, SAT-based attacks have been shown to efficiently break key-based obfuscation methods with a rather small number of distinguishing input patterns (DIPs) to iteratively refine the key.

In Chapter 3, a novel secure cell (SC) design for implementing design-for-security (DFS) infrastructure has been proposed that successfully prevents the leaking of obfuscation key to an adversary at any time. Due to the unavailability of scan data that contains the obfuscation key, existing attacks become unfeasible for the proposed design-for-security structure. Besides, this structure allows scan-based tests, post-silicon validation, and debug to ensure perfect compliance at the cost of a small area overhead.

In Chapter 4, a new attack method was introduced. Over the years, researchers have proposed different techniques to prevent SAT-based attacks. However, even if the circuit achieves SAT attack resiliency, it is not entirely secure. There are still other ways to obtain the secret key value. This chapter shows that even if the circuit can effectively block the SAT-based attack, a tampering attacker can still defeat the security measures provided from any existing logic locking method to obtain the secret key.

In order to prevent the above kind of tampering attack, Chapter 5 proposed a method of modeling and test generation for combinational hardware Trojans. Based on the two basic components of hardware Trojan, a trigger and a payload, a combinational hardware Trojan model with n triggers and one payload has been developed. Based on this model, a detection method using conditional stuck-at tests has also been proposed. According to experimental results, both Trojan sampling and vector sampling are beneficial for detection coverage estimates. The proposed detection tests are more effective than the N-detect stuck-at tests or random vectors in detecting hardware Trojans.

In Chapter 6, a defect characterization method of skyrmion-based digital circuit that belongs to an emerging technology is proposed. Depending on the fault type a defect maps on to, we classified defects into three categories: Technology-independent, Technology-specific, and no fault. Using the principle of fault equivalence, a defect to fault mapping technique is proposed. A test method for the mapped faults is introduced as well. The simulation result shows that compared with a traditional CMOS circuit, the skyrmion logic circuit pattern generation method reduces the number of test patterns and increases fault coverage when considering single stuck-at faults.

7.2 Future Work

This section discusses possible extensions of the work presented in this dissertation.

7.2.1 Hardware Trojan Detection

In the future, the scope of modeling and test generation should be expanded to solve diagnostic problems. Another aspect to explore is the minimization of Trojan tests. Despite the fact that Trojan tests do not need to be applied to all chips, the numbers of Type-1 Trojans (with single trigger) and their CSP for large circuits can be enormous. A third aspect to explore is the behavior of conditional stuck-at fault patterns (CSP) in detecting Trojans of type $n > 4$.

The proposed Trojan detection method can only be applied to the combinational hardware Trojan. Due to the stealthiness, the detection of sequential hardware Trojan has more challenges than combinational hardware Trojan [175]. The trigger condition of sequential hardware Trojan is more difficult to satisfy, rendering logic testing ineffective in detecting the hardware Trojan. In the future research, it is necessary to find an effective way to detect sequential hardware Trojan.

7.2.2 Emerging Technologies

As the CMOS device size is approaching the physical limit, it is not practical to reduce the device size using the traditional structure. Many research institutes and semiconductor companies are trying to improve the design of semiconductor devices to follow Moore's law up to the hilt. Magnetic skyrmion devices are considered a promising alternative to the current CMOS technology in the post-Moore era due to their small size, extremely low power consumption, and ultra-high operation speed. In CMOS technology, the electric charge decides the logic level, while in skyrmion gate, the state of a logic signal is represented by the presence (logic-1) or absence (logic-0) of a single skyrmion, a magnetic pseudoparticle.

For this new type of circuit technology, security protection is also necessary. Because skyrmion circuit and conventional CMOS circuit are different in structure and characteristics, testing and verification processes are being addressed for the new technology. In addition, security will be an essential concern for the future skyrmion circuits and must be investigated.

Like traditional CMOS circuits, skyrmion circuits will be affected by IP piracy, IC overproduction, and even hardware Trojan. There is an urgent need for a solution that establishes trust for protecting IPs and ICs in skyrmion circuits. In the future research, the security mechanism should also be taken into consideration, e.g., designing logic locking structure, and developing methods to detect hardware Trojans in skyrmion circuits.

Bibliography

[1] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *Proc. Design, Automation and Test in Europe*. IEEE, 2008, pp. 1069–1074.

[2] S. M. Plaza and I. L. Markov, "Solving the Third-Shift Problem in IC Piracy with Test-Aware Logic Locking," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 961–971, Jun. 2015.

[3] Y.-W. Lee and N. A. Touba, "Improving Logic Obfuscation via Logic Cone Analysis," in *Proc. 16th Latin-American Test Symposium (LATS)*. IEEE, 2015, pp. 1–6.

[4] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 66–75, 2010.

[5] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," in *Proc. of ACM/IEEE on Design Automation Conference*, June 2012, pp. 83–89.

[6] U. Guin, Q. Shi, D. Forte, and M. M. Tehranipoor, "FORTIS: A Comprehensive Solution for Establishing Forward Trust for Protecting IPs and ICs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 21, no. 4, p. 63, 2016.

[7] A. Yeh, "Trends in the Global IC Design Service Market," DIGITIMES Research, March 2012. [Online]. Available: http://www.digitimes.com/news/a20120313RS400.html?chid=2

[8] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

[9] U. Guin, D. DiMase, and M. Tehranipoor, "Counterfeit Integrated Circuits: Detection, Avoidance, and the Challenges Ahead," *Journal of Electronic Testing*, vol. 30, no. 1, pp. 9–23, Feb. 2014.

[10] M. M. Tehranipoor, U. Guin, and D. Forte, "Counterfeit Integrated Circuits," in *Counterfeit Integrated Circuits*. Springer, 2015, pp. 15–36.

[11] P. Chowdhury, U. Guin, A. D. Singh, and V. D. Agrawal, "Estimating Operational Age of an Integrated Circuit," *Journal of Electronic Testing*, no. 1, pp. 25–40, Feb. 2021.

[12] P. Cui, J. Dixon, U. Guin, and D. Dimase, "A Blockchain-based Framework for Supply Chain Provenance," *IEEE Access*, vol. 7, pp. 157 113–157 125, 2019.

[13] Y. Zhang and U. Guin, "End-to-end Traceability of ICs in Component Supply Chain for Fighting Against Recycling," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 767–775, 2019.

[14] W. Wang, U. Guin, and A. Singh, "A Zero-cost Detection Approach for Recycled ICs Using Scan Architecture," in *Proc. IEEE 38th VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[15] U. Guin, W. Wang, C. Harper, and A. D. Singh, "Detecting Recycled SOCs by Exploiting Aging Induced Biases in Memory Cells," in *Proc. IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 72–80.

[16] P. Chowdhury, U. Guin, A. D. Singh, and V. D. Agrawal, "Two-pattern $\Delta\ I_{DDQ}$ Test for Recycled IC Detection," in *Proc. 32nd International Conference on VLSI Design and 18th International Conference on Embedded Systems (VLSID)*. IEEE, 2019, pp. 82–87.

[17] J. Lee, M. Tehranipoor, and J. Plusquellic, "A Low-Cost Solution for Protecting IPs Against Scan-Based Side-Channel Attacks," in *Proc. of the IEEE VLSI Test Symposium (VTS)*, 2006, pp. 94–99. [Online]. Available: http://dx.doi.org/10.1109/VTS.2006.7

[18] Y. M. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," in *Proc. of USENIX Security Symposium*, 2007, pp. 20:1–20:16.

[19] R. Chakraborty and S. Bhunia, “Hardware Protection and Authentication Through Netlist Level Obfuscation,” in *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, November 2008, pp. 674 –677.

[20] P. Subramanyan, S. Ray, and S. Malik, “Evaluating the Security of Logic Encryption Algorithms,” in *Proc. IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2015, pp. 137–143.

[21] M. T. Rahman, S. Tajik, M. S. Rahman, M. Tehranipoor, and N. Asadizanjani, “The Key is Left Under the Mat: On the Inappropriate Security Assumption of Logic Locking Schemes,” in *Proc. IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 262–272.

[22] H. Lohrke, S. Tajik, C. Boit, and J.-P. Seifert, “No Place to Hide: Contactless Probing of Secret Data on FPGAs,” in *Proc. International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 147–167.

[23] S. Tajik, H. Lohrke, J.-P. Seifert, and C. Boit, “On the Power of Optical Contactless Probing: Attacking Bitstream Encryption of FPGAs,” in *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1661–1674.

[24] C. Helfmeier, D. Nedospasov, C. Tarnovsky, J. S. Krissler, C. Boit, and J.-P. Seifert, “Breaking and Entering Through the Silicon,” in *Proc. ACM SIGSAC Conference on Computer & Communications Security*, 2013, pp. 733–744.

[25] A. J. Reich, K. H. Nakagawa, and R. E. Boone, “OASIS vs. GDSII Stream Format Efficiency,” in *Proc. 23rd Annual BACUS Symposium on Photomask Technology*, vol. 5256. International Society for Optics and Photonics, 2003, pp. 163–173.

[26] Y. Xie and A. Srivastava, “Mitigating SAT Attack on Logic Locking,” in *Proc. International Conference on Cryptographic Hardware and Embedded Systems*, 2016, pp. 127–146.

[27] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, “SARLock: SAT Attack Resistant Logic Locking,” in *Proc. IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2016, pp. 236–241.

[28] U. Guin, Z. Zhou, and A. Singh, “A Novel Design-for-Security (DFS) Architecture to Prevent Unauthorized IC Overproduction,” in *Proc. of the IEEE VLSI Test Symposium (VTS)*, 2017, pp. 1–6.

[29] U. Guin, Z. Zhou, and A. D. Singh, “Robust Design-for-Security Architecture for Enabling Trust in IC Manufacturing and Test,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 5, pp. 818–830, 2018.

[30] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, “Provably-secure Logic Locking: From Theory to Practice,” in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1601–1618.

[31] S. Khaleghi, K. Da Zhao, and W. Rao, “IC Piracy Prevention via Design Withholding and Entanglement,” in *Proc. Asia and South Pacific Design Automation Conference*, 2015, pp. 821–826.

[32] B. Liu and B. Wang, “Embedded Reconfigurable Logic for ASIC Design Obfuscation against supply chain attacks,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1–6.

[33] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Science & Business Media, 2004.

[34] D. Sirone and P. Subramanyan, “Functional Analysis Attacks on Logic Locking,” *arXiv preprint arXiv:1811.12088*, 2018.

[35] H. Wang, D. Forte, M. M. Tehranipoor, and Q. Shi, “Probing Attacks on Integrated Circuits: Challenges and Research Opportunities,” *IEEE Design & Test of Computers*, vol. 34, no. 5, pp. 63–71, 2017.

[36] H. Wang, Q. Shi, D. Forte, and M. M. Tehranipoor, "Probing Assessment Framework and Evaluation of Antiprobing Solutions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1239–1252, 2019.

[37] H. Shen, N. Asadizanjani, M. Tehranipoor, and D. Forte, "Nanopyramid: An Optical Scrambler Against Backside Probing Attacks," in *Proc. Int. Symposium for Testing and Failure Analysis (ISTFA)*, 2018, p. 280.

[38] Y. Zhang, P. Cui, Z. Zhou, and U. Guin, "TGA: An Oracle-less and Topology-Guided Attack on Logic Locking," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, 2019, pp. 75–83.

[39] S. Adee, "The Hunt for the Kill Switch," *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, 2008.

[40] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design & Test of Computers*, vol. 27, no. 1, 2010.

[41] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy Hardware: Identifying and Classifying Hardware Trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[42] M. Tehranipoor, H. Salmani, X. Zhang, M. Wang, R. Karri, J. Rajendran, and K. Rosenfeld, "Trustworthy Hardware: Trojan Detection and Design-for-Trust Challenges," *Computer*, vol. 44, no. 7, pp. 66–74, 2010.

[43] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[44] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[45] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware Trojans: Lessons Learned After One Decade of Research," *ACM Trans. Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, p. 6, 2016.

[46] N. Lesperance, S. Kulkarni, and K.-T. Cheng, “Hardware Trojan Detection Using Exhaustive Testing of k-bit Subspaces,” in *Proc. of Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2015, pp. 755–760.

[47] A. Waksman, M. Suozzo, and S. Sethumadhavan, “FANCI: Identification of Stealthy Malicious Logic Using Boolean Functional Analysis,” in *Proc. ACM SIGSAC Conf. on Computer & Communications Security*, 2013, pp. 697–708.

[48] R. S. Chakraborty, F. G. Wolff, S. Paul, C. A. Papachristou, and S. Bhunia, “MERO: A Statistical Approach for Hardware Trojan Detection,” in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2009, pp. 396–410.

[49] M. Banga and M. S. Hsiao, “Odette: A Non-Scan Design-for-Test Methodology for Trojan Detection in ICs,” in *Proc. IEEE Int. Symp. Hardware-Oriented Security and Trust*, 2011, pp. 18–23.

[50] S. K. Haider, C. Jin, M. Ahmad, D. M. Shila, O. Khan, and M. van Dijk, “Advancing the State-of-the-Art in Hardware Trojans Detection,” *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 18–32, 2017.

[51] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, “Trojan Detection Using IC Fingerprinting,” in *Proc. IEEE Symp. Security and Privacy (SP)*, 2007, pp. 296–310.

[52] M. Banga and M. S. Hsiao, “A Region Based Approach for the Identification of Hardware Trojans,” in *Proc. IEEE Int. Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 40–47.

[53] M. Banga and M. Hsiao, “A Novel Sustained Vector Technique for the Detection of Hardware Trojans,” in *Proc. 22nd International Conference on VLSI Design*. IEEE, 2009, pp. 327–332.

[54] R. Rad, J. Plusquellic, and M. Tehranipoor, “Sensitivity Analysis to Hardware Trojans Using Power Supply Transient Signals,” in *Proc. IEEE Int. Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 3–7.

[55] J. Li and J. Lach, “At-Speed Delay Characterization for IC Authentication and Trojan Horse Detection,” in *Proc. IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 8–14.

[56] Y. Liu, K. Huang, and Y. Makris, “Hardware Trojan Detection Through Golden Chip-Free Statistical Side-Channel Fingerprinting,” in *Proc. of Design Automation Conference*, 2014.

[57] R. S. Chakraborty and S. Bhunia, “Security Against Hardware Trojan Through a Novel Application of Design Obfuscation,” in *Proc. Int. Conf. Computer-Aided Design*, 2009, pp. 113–116.

[58] H. Salmani, M. Tehranipoor, and J. Plusquellic, “A Novel Technique for Improving Hardware Trojan Detection and Reducing Trojan Activation Time,” *IEEE Trans. Very Large Scale Integration Sys.*, vol. 20, no. 1, pp. 112–125, 2012.

[59] K. Xiao and M. Tehranipoor, “BISA: Built-In Self-Authentication for Preventing Hardware Trojan Insertion,” in *Proc. IEEE Int. Symp. Hardware-Oriented Security and Trust*, 2013, pp. 45–50.

[60] X. T. Ngo, S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, “Linear Complementary Dual Code Improvement to Strengthen Encoded Circuit Against Hardware Trojan Horses,” in *Proc. IEEE Int. Symp. Hardware Oriented Security and Trust*, 2015, pp. 82–87.

[61] K. Vaidyanathan, B. P. Das, and L. Pileggi, “Detecting Reliability Attacks During Split Fabrication Using Test-Only BEOL Stack,” in *Proc. of Design Automation Conf.*, 2014, pp. 1–6.

[62] J. J. V. Rajendran, O. Sinanoglu, and R. Karri, “Is Split Manufacturing Secure?” in *Proc. Conf. Design, Automation and Test in Europe (DATE)*, 2013, pp. 1259–1264.

[63] S. Wei, S. Meguerdichian, and M. Potkonjak, “Malicious Circuitry Detection Using Thermal Conditioning,” *IEEE Trans. on Information Forensics and Security*, vol. 6, no. 3, pp. 1136–1145, Sep. 2011.

[64] A. N. Nowroz, K. Hu, F. Koushanfar, and S. Reda, "Novel Techniques for High-Sensitivity Hardware Trojan Detection Using Thermal and Power Maps," *Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1792–1805, 2014.

[65] Y. Jin and Y. Makris, "Hardware Trojan Detection Using Path Delay Fingerprint," in *Proc. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 51–57.

[66] J. He, Y. Zhao, X. Guo, and Y. Jin, "Hardware Trojan Detection Through Chip-Free Electromagnetic Side-Channel Statistical Analysis," *IEEE Trans. Very Large Scale Integration Sys.*, vol. 25, no. 10, pp. 2939–2948, 2017.

[67] F. Koushanfar and G. Qu, "Hardware Metering," in *Proc. IEEE-ACM Design Automation Conference*, 2001, pp. 490–493.

[68] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote Activation of ICs for Piracy Prevention and Digital Right Management," in *Proc. of IEEE/ACM Int. Conf. on Computer-Aided Design*, 2007, pp. 674–677.

[69] J. Huang and J. Lach, "IC Activation and User Authentication for Security-Sensitive Systems," in *Proc. IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 76–80.

[70] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. Springer, 2012.

[71] G. K. Contreras, M. T. Rahman, and M. Tehranipoor, "Secure Split-Test for Preventing IC Piracy by Untrusted Foundry and Assembly," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, 2013, pp. 196–203.

[72] M. T. Rahman, D. Forte, Q. Shi, G. K. Contreras, and M. Tehranipoor, "CSST: Preventing Distribution of Unlicensed and Rejected ICs by Untrusted Foundry and Assembly," in *Proc. Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2014, pp. 46–51.

[73] U. Guin and M. M. Tehranipoor, "Obfuscation and Encryption for Securing Semiconductor Supply Chain," in *Hardware Protection through Obfuscation*. Springer, 2017, pp. 317–346.

[74] E. Castillo, U. Meyer-Baese, A. García, L. Parrilla, and A. Lloris, "IPP@ HDL: Efficient Intellectual Property Protection Scheme for IP Cores," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 5, pp. 578–591, 2007.

[75] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-Based Watermarking Techniques for Design IP Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 10, pp. 1236–1252, Oct. 2001.

[76] R. S. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.

[77] R. Torrance and D. James, "The State-of-the-Art in IC Reverse Engineering," in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems*, 2009, pp. 363–381.

[78] M. M. Tehranipoor, U. Guin, and S. Bhunia, "Invasion of the Hardware Snatchers," *IEEE Spectrum*, vol. 54, no. 5, pp. 36–41, 2017.

[79] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[80] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2002, pp. 148–160.

[81] G. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Proc. of ACM/IEEE on Design Automation Conference*, 2007, pp. 9–14.

[82] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 63–80.

[83] R. Maes and I. Verbauwhede, "Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions," *Towards Hardware-Intrinsic Security*, pp. 3–37, 2010.

[84] A. Maiti and P. Schaumont, "The Impact of Aging on a Physical Unclonable Function," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 9, pp. 1854–1864, 2014.

[85] X. Zhuang, T. Zhang, H.-H. S. Lee, and S. Pande, "Hardware Assisted Control Flow Obfuscation for Embedded Processors," in *Proc. of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2004, pp. 292–302.

[86] Synopsys, "Compression for Highest Test Quality and Lowest Test Cost," 2015. [Online]. Available: https://www.synopsys.com/Tools/Implementation/RTLSynthesis/Test/Pages/dftmax-ultra-ds.aspx

[87] P. Nagaraj and S. P. P. Engineer, "Choosing the Right Scan Compression Architecture for Your Design," *Cadence Design Syst*, 2015.

[88] E. J. McCluskey, "Verification Testing? A Pseudoexhaustive Test Technique," *IEEE Transactions on Computers*, no. 6, pp. 541–546, 1984.

[89] C. Albrecht, "IWLS 2005 Benchmarks," in *Proc. International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*, 2005.

[90] DC Ultra: Concurrent Timing, Area, Power, and Test Optimization, "[online] available at: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html," 2019.

[91] VCS: Industry's Highest Performance Simulation Solution, Synopsys, Inc., 2017.

[92] Y. Shen and H. Zhou, "Double Dip: Re-evaluating Security of Logic Encryption Algorithms," in *Proceedings of the Great Lakes Symposium on VLSI*, 2017, pp. 179–184.

[93] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security Analysis of Anti-SAT," in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 342–347.

[94] R. Kuppuswamy, P. DesRosier, D. Feltham, R. Sheikh, and P. Thadikaran, "Full Hold-Scan Systems in Microprocessors: Cost/Benefit Analysis," *Intel Technology Journal*, vol. 8, no. 1, pp. 63–72, Feb. 2004.

[95] U. Guin, T. Chakraborty, and M. Tehranipoor, "Functional Fmax Test-time Reduction Using Novel DFTs for Circuit Initialization," in *Proc. IEEE International Conference on Computer Design (ICCD)*, Oct. 2013, pp. 1–6.

[96] Synopsys Inc., Mountain View, CA, USA, "TetraMAX ATPG: Automatic Test Pattern Generation," 2017.

[97] Synopsys 32/28nm Generic Library for Teaching IC Design, https://www.synopsys.com/community/university-program/teaching-resources.html.

[98] A. Jain, Z. Zhou, and U. Guin, "TAAL: Tampering Attack on Any Key-based Logic Locked Circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 4, pp. 1–22, 2021.

[99] M. T. Rahman, D. Forte, Q. Shi, G. K. Contreras, and M. Tehranipoor, "CSST: An Efficient Secure Split-test for Preventing IC Piracy," in *Proc. IEEE 23rd North Atlantic Test Workshop*, 2014, pp. 43–47.

[100] D. Zhang, X. Wang, M. T. Rahman, and M. Tehranipoor, "An On-Chip Dynamically Obfuscated Wrapper for Protecting Supply Chain Against IP and IC Piracies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 11, pp. 2456–2469, 2018.

[101] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. Springer Science & Business Media, 2011.

[102] S. Bhunia and M. Tehranipoor, *Hardware Security: A Hands-on Learning Approach*. Morgan Kaufmann, 2018.

[103] E. Charbon, "Hierarchical Watermarking in IC Design," in *Proc. of the Custom Integrated Circuits*, 1998, pp. 295–298.

[104] G. Qu and M. Potkonjak, *Intellectual Property Protection in VLSI Designs: Theory and Practice*. Springer Science & Business Media, 2007.

[105] R. W. Jarvis and M. G. McIntyre, "Split Manufacturing Method for Advanced Semiconductor Circuits," 2007, US Patent 7,195,931.

[106] K. Vaidyanathan, R. Liu, E. Sumbul, Q. Zhu, F. Franchetti, and L. Pileggi, "Efficient and Secure Intellectual Property (IP) Design with Split Fabrication," in *Proc. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014, pp. 13–18.

[107] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault Analysis-based Logic Encryption," *IEEE Transactions on computers*, vol. 64, no. 2, pp. 410–424, 2015.

[108] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2015.

[109] Q. Yu, J. Dofe, and Z. Zhang, "Exploiting Hardware Obfuscation Methods to Prevent and Detect Hardware Trojans," in *Proc. International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 819–822.

[110] A. Sengupta and S. P. Mohanty, "Functional Obfuscation of DSP Cores Using Robust Logic Locking and Encryption," in *Proc. Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 709–713.

[111] Q. Yu, J. Dofe, Z. Zhang, and S. Kramer, "Hardware Obfuscation Methods for Hardware Trojan Prevention and Detection," in *The Hardware Trojan War*. Springer, 2018, pp. 291–325.

[112] Z. Zhou, U. Guin, and V. D. Agrawal, "Modeling and Test Generation for Combinational Hardware Trojans," in *Proc. IEEE 36th VLSI Test Symposium (VTS)*, 2018, pp. 1–6.

[113] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating UCI: Building Stealthy and Malicious Hardware," in *Proc. IEEE Symp. Security and Privacy (SP)*, 2011, pp. 64–77.

[114] X. Wang, S. Narasimhan, A. Krishna, T. Mal-Sarkar, and S. Bhunia, "Sequential Hardware Trojan: Side-Channel Aware Design and Placement," in *Proc. International Conference on Computer Design (ICCD)*, 2011, pp. 297–300.

[115] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog Malicious Hardware," in *Proc. IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 18–37.

[116] K. Nagarajan, M. N. I. Khan, and S. Ghosh, "ENTT: A Family of Emerging NVM-based Trojan Triggers," in *Proc. International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 51–60.

[117] C. Kison, O. M. Awad, M. Fyrbiak, and C. Paar, "Security Implications of Intentional Capacitive Crosstalk," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 12, pp. 3246–3258, 2019.

[118] K. S. Subramani, A. Antonopoulos, A. A. Abotabl, A. Nosratinia, and Y. Makris, "ACE: Adaptive Channel Estimation for Detecting Analog/RF Trojans in WLAN Transceivers," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 722–727.

[119] D. Chang, B. Bakkaloglu, and S. Ozev, "Enabling Unauthorized RF Transmission Below Noise Floor with No Detectable Impact on Primary Communication Performance," in *Proc. VLSI Test Symposium (VTS)*, 2015, pp. 1–4.

[120] D. Bryan, "The ISCAS'85 Benchmark Circuits and Netlist Format," *North Carolina State University*, 1985.

[121] S. Davidson, "ITC'99 Benchmark Circuits - Preliminary Results," in *Proc. International Test Conference*. IEEE Computer Society, 1999, pp. 1125–1125.

[122] Y. Jin and Y. Makris, "Hardware Trojans in Wireless Cryptographic ICs," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 26–35, 2010.

[123] Y. Liu, G. Volanis, K. Huang, and Y. Makris, "Concurrent Hardware Trojan Detection in Wireless Cryptographic ICs," in *Proc. International Test Conference (ITC)*, 2015, pp. 1–8.

[124] Y. Qin and T. Xia, “Sensitivity Analysis of Ring Oscillator Based Hardware Trojan Detection,” in *Proc. IEEE 17th International Conference on Communication Technology (ICCT)*. IEEE, 2017, pp. 1979–1983.

[125] A. Ramdas, S. M. Saeed, and O. Sinanoglu, “Slack Removal for Enhanced Reliability and Trust,” in *Int. Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2014, pp. 1–4.

[126] B. Zhou, W. Zhang, S. Thambipillai, and J. Teo, “A Low Cost Acceleration Method for Hardware Trojan Detection Based on Fan-out Cone Analysis,” in *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis*, 2014, p. 28.

[127] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, “Circuit Camouflage Integration for Hardware IP Protection,” in *Proceedings of Annual Design Automation Conference*, 2014, pp. 1–5.

[128] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, “Security Analysis of Integrated Circuit Camouflaging,” in *Proc. of ACM SIGSAC Conference on Computer & Communications Security*, 2013, pp. 709–720.

[129] Q. Shi, K. Xiao, D. Forte, and M. M. Tehranipoor, “Obfuscated Built-in Self-Authentication,” in *Hardware Protection Through Obfuscation*. Springer, 2017, pp. 263–289.

[130] Intelligence Advanced Research Projects Activity, “Trusted Integrated Chips (TIC) Program,” 2011, [online] Available at : https://www.iarpa.gov/index.php/research-programs/tic.

[131] J. Magaña, D. Shi, J. Melchert, and A. Davoodi, “Are Proximity Attacks a Threat to the Security of Split Manufacturing of Integrated Circuits?” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3406–3419, 2017.

[132] N. Vashistha, H. Lu, Q. Shi, M. T. Rahman, H. Shen, D. L. Woodard, N. Asadizanjani, and M. Tehranipoor, “Trojan Scanner: Detecting Hardware Trojans with Rapid SEM Imaging combined with Image Processing and Machine Learning,” in *Proc. Int. Symposium for Testing and Failure Analysis*, 2018, p. 256.

[133] Y. Hou, H. He, K. Shamsi, Y. Jin, D. Wu, and H. Wu, “R2D2: Runtime Reassurance and Detection of A2 Trojan,” in *Proc. International Symposium on Hardware Oriented Security and Trust (HOST)*, 2018, pp. 195–200.

[134] X. Guo, H. Zhu, Y. Jin, and X. Zhang, “When Capacitors Attack: Formal Method Driven Design and Detection of Charge-Domain Trojans,” in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2019.

[135] O. E. Cornelia, “Conditional Stuck-At Fault Model for PLA Test Generation,” Master’s thesis, McGill University, Montreal, Canada, 1987.

[136] K. Heragu, V. D. Agrawal, and M. L. Bushnell, “FACTS: Fault Coverage Estimation by Test Vector Sampling,” in *Proc. of VLSI Test Symp.*, 1994, pp. 266–271.

[137] V. D. Agrawal, H. Farhat, and S. C. Seth, “Test Generation by Fault Sampling,” in *Proc. Int. Conf. on Computer Design (ICCD)*, 1988, pp. 58–61.

[138] Z. Zhou, U. Guin, P. Li, and V. D. Agrawal, “Defect Characterization and Testing of Skyrmion-Based Logic Circuits,” in *Proc. IEEE 39th VLSI Test Symposium (VTS)*, 2021, pp. 1–7.

[139] T.-C. Chen, “Overcoming Research Challenges for CMOS Scaling: Industry Directions,” in *Proc. IEEE 8th International Conference on Solid-State and Integrated Circuit Technology*, 2006, pp. 4–7.

[140] A. Hirohata, K. Yamada, Y. Nakatani, L. Prejbeanu, B. Diény, P. Pirro, and B. Hillebrands, “Review on Spintronics: Principles and Device Applications,” *Journal of Magnetism and Magnetic Materials*, p. 166711, 2020.

[141] A. Fert, V. Cros, and J. Sampaio, “Skyrmions on the Track,” *Nature Nanotechnology*, vol. 8, no. 3, pp. 152–156, 2013.

[142] T. H. R. Skyrme, “A Unified Field Theory of Mesons and Baryons,” *Nuclear Physics*, vol. 31, pp. 556–569, 1962.

[143] X. Yu, Y. Onose, N. Kanazawa, J. H. Park, J. Han, Y. Matsui, N. Nagaosa, and Y. Tokura, "Real-Space Observation of a Two-dimensional Skyrmion Crystal," *Nature*, vol. 465, no. 7300, pp. 901–904, 2010.

[144] N. Kanazawa, Y. Onose, T. Arima, D. Okuyama, K. Ohoyama, S. Wakimoto, K. Kakurai, S. Ishiwata, and Y. Tokura, "Large Topological Hall Effect in a Short-Period Helimagnet MnGe," *Physical Review Letters*, vol. 106, no. 15, p. 156603, 2011.

[145] C. Moreau-Luchaire, C. Moutafis, N. Reyren, J. Sampaio, C. Vaz, N. Van Horne, K. Bouzehouane, K. Garcia, C. Deranlot, P. Warnicke *et al.*, "Additive Interfacial Chiral Interaction in Multilayers for Stabilization of Small Individual Skyrmions at Room Temperature," *Nature Nanotechnology*, vol. 11, no. 5, pp. 444–448, 2016.

[146] S. Woo, K. Litzius, B. Krüger, M.-Y. Im, L. Caretta, K. Richter, M. Mann, A. Krone, R. M. Reeve, M. Weigand *et al.*, "Observation of Room-Temperature Magnetic Skyrmions and Their Current-Driven Dynamics in Ultrathin Metallic Ferromagnets," *Nature Materials*, vol. 15, no. 5, pp. 501–506, 2016.

[147] K. M. Song, J.-S. Jeong, B. Pan, X. Zhang, J. Xia, S. Cha, T.-E. Park, K. Kim, S. Finizio, J. Raabe *et al.*, "Skyrmion-Based Artificial Synapses for Neuromorphic Computing," *Nature Electronics*, vol. 3, no. 3, pp. 148–155, 2020.

[148] G. Chen, "Skyrmion Hall Effect," *Nature Physics*, vol. 13, no. 2, pp. 112–113, 2017.

[149] T. Dohi, S. DuttaGupta, S. Fukami, and H. Ohno, "Formation and Current-Induced Motion of Synthetic Antiferromagnetic Skyrmion Bubbles," *Nature Communications*, vol. 10, no. 1, pp. 1–6, 2019.

[150] W. Jiang, P. Upadhyaya, W. Zhang, G. Yu, M. B. Jungfleisch, F. Y. Fradin, J. E. Pearson, Y. Tserkovnyak, K. L. Wang, O. Heinonen *et al.*, "Blowing Magnetic Skyrmion Bubbles," *Science*, vol. 349, no. 6245, pp. 283–286, 2015.

[151] G. Yu, P. Upadhyaya, X. Li, W. Li, S. K. Kim, Y. Fan, K. L. Wong, Y. Tserkovnyak, P. K. Amiri, and K. L. Wang, "Room-Temperature Creation and Spin–Orbit Torque Manipulation

of Skyrmions in Thin Films with Engineered Asymmetry," *Nano Letters*, vol. 16, no. 3, pp. 1981–1988, 2016.

[152] J. Zang, M. Mostovoy, J. H. Han, and N. Nagaosa, "Dynamics of Skyrmion Crystals in Metallic Thin Films," *Physical Review Letters*, vol. 107, no. 13, p. 136804, 2011.

[153] N. Nagaosa and Y. Tokura, "Topological Properties and Dynamics of Magnetic Skyrmions," *Nature Nanotechnology*, vol. 8, no. 12, p. 899, 2013.

[154] W. Jiang, X. Zhang, G. Yu, W. Zhang, X. Wang, M. B. Jungfleisch, J. E. Pearson, X. Cheng, O. Heinonen, K. L. Wang *et al.*, "Direct Observation of the Skyrmion Hall Effect," *Nature Physics*, vol. 13, no. 2, pp. 162–169, 2017.

[155] G. Chen, "Spin–Orbitronics: Skyrmion Hall Effect," *Nature Physics*, vol. 13, no. 2, pp. 112–113, 2017.

[156] K. Litzius, I. Lemesh, B. Krüger, P. Bassirian, L. Caretta, K. Richter, F. Büttner, K. Sato, O. A. Tretiakov, J. Förster *et al.*, "Skyrmion Hall Effect Revealed by Direct Time-Resolved X-Ray Microscopy," *Nature Physics*, vol. 13, no. 2, pp. 170–175, 2017.

[157] J. Iwasaki, M. Mochizuki, and N. Nagaosa, "Current-Induced Skyrmion Dynamics in Constricted Geometries," *Nature Nanotechnology*, vol. 8, no. 10, p. 742, 2013.

[158] X. Zhang, G. Zhao, H. Fangohr, J. P. Liu, W. Xia, J. Xia, and F. Morvan, "Skyrmion-Skyrmion and Skyrmion-Edge Repulsions in Skyrmion-Based Racetrack Memory," *Scientific Reports*, vol. 5, p. 7643, 2015.

[159] T. Maruyama, Y. Shiota, T. Nozaki, K. Ohta, N. Toda, M. Mizuguchi, A. Tulapurkar, T. Shinjo, M. Shiraishi, S. Mizukami *et al.*, "Large Voltage-Induced Magnetic Anisotropy Change in a Few Atomic Layers of Iron," *Nature Nanotechnology*, vol. 4, no. 3, p. 158, 2009.

[160] P. Upadhyaya, G. Yu, P. K. Amiri, and K. L. Wang, "Electric-Field Guiding of Magnetic Skyrmions," *Physical Review B*, vol. 92, no. 13, p. 134411, 2015.

[161] X. Liang, J. Xia, X. Zhang, M. Ezawa, O. A. Tretiakov, X. Liu, L. Qiu, G. Zhao, and Y. Zhou, “Antiferromagnetic Skyrmion-Based Logic Gates Controlled by Electric Currents and Fields,” *arXiv preprint arXiv:1909.10709*, 2019.

[162] M. Chauwin, X. Hu, F. Garcia-Sanchez, N. Betrabet, A. Paler, C. Moutafis, and J. S. Friedman, “Skyrmion Logic System for Large-Scale Reversible Computation,” *Physical Review Applied*, vol. 12, no. 6, p. 064053, 2019.

[163] R. Bishnoi, F. Oboril, and M. B. Tahoori, “Design of Defect and Fault-Tolerant Nonvolatile Spintronic Flip-Flops,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1421–1432, 2016.

[164] A. Troiano, F. Corinto, and E. Pasero, “A Memristor Circuit Using Basic Elements with Memory Capability,” in *Recent Advances of Neural Network Models and Applications. Smart Innovation, Systems and Technologies*, S. Bassis, A. Esposito, and F. Morabito, Eds., vol. 26. Springer, 2014.

[165] W. F. Brown, *Micromagnetics*. Interscience Publishers, 1963, no. 18.

[166] A. A. Thiele, “Steady-State Motion of Magnetic Domains,” *Physical Review Letters*, vol. 30, no. 6, p. 230, 1973.

[167] A. Vansteenkiste, J. Leliaert, M. Dvornik, M. Helsen, F. Garcia-Sanchez, and B. Van Waeyenberge, “The Design and Verification of MuMax3,” *AIP Advances*, vol. 4, no. 10, p. 107133, 2014. [Online]. Available: http://doi.org/10.1063/1.4899186

[168] M. Xu, M. Li, P. Khanal, A. Habiboglu, B. Insana, Y. Xiong, T. Peterson, J. C. Myers, D. Ortega, H. Qu *et al.*, “Voltage-Controlled Antiferromagnetism in Magnetic Tunnel Junctions,” *Physical Review Letters*, vol. 124, no. 18, p. 187701, 2020.

[169] S.-Z. Lin, C. Reichhardt, C. D. Batista, and A. Saxena, “Particle Model for Skyrmions in Metallic Chiral Magnets: Dynamics, Pinning, and Creep,” *Physical Review B*, vol. 87, no. 21, p. 214419, 2013.

[170] M. Weisheit, S. Fähler, A. Marty, Y. Souche, C. Poinsignon, and D. Givord, "Electric Field-Induced Modification of Magnetism in Thin-Film Ferromagnets," *Science*, vol. 315, no. 5810, pp. 349–351, 2007.

[171] DC Ultra: Concurrent Timing, Area, Power, and Test Optimization. See details at - https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html.

[172] S. Mangin, D. Ravelosona, J. Katine, M. Carey, B. Terris, and E. E. Fullerton, "Current-Induced Magnetization Reversal in Nanopillars with Perpendicular Anisotropy," *Nature materials*, vol. 5, no. 3, pp. 210–215, 2006.

[173] TestMAX ATPG, Synopsys, https://www.synopsys.com/support/training/ signoff/tmax1-fcd.html.

[174] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Targeted Translator in FORTRAN, Special Session on ATPG and Fault Simulation," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS'85)*, Jun. 1985, pp. 663–698, *Benchmark circuit netlists are available at* http://www.pld.ttu.ee/∼maksim/benchmarks/iscas85/.

[175] A. Jain, Z. Zhou, and U. Guin, "Survey of Recent Developments for Hardware Trojan Detection," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.