Testing and Verification Strategies for Enhancing Trust in Third Party IPs

Mainak Banga

Dissertation submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Michael S. Hsiao (Chair)
Lynn A. Abbott
Leyla Nazhandali
Mark Shimozono
Sandeep K. Shukla

December 7^{th}, 2010
Blacksburg, Virginia

Keywords: Trojans, Third party IP, Side-Channel Analysis, Sequential Equivalence Checking, Design for Testability

Testing and Verification Strategies for Enhancing Trust in Third Party IPs

Mainak Banga

ABSTRACT

Globalization in semiconductor industry has surged up the trend of outsourcing component design and manufacturing process across geographical boundaries. While cost reduction and short time to market are the driving factors behind this trend, the authenticity of the final product remains a major question. Third party deliverables are solely based on mutual trust and any manufacturer with a malicious intent can fiddle with the original design to make it work otherwise than expected in certain specific situations. In case such a backfire happens, the consequences can be disastrous especially for mission critical systems such as space-explorations, defense equipments such as missiles, life saving equipments such as medical gadgets where a single failure can translate to a loss of lives or millions of dollars. Thus accompanied with outsourcing, comes the question of trustworthy design - **"how to ensure that integrity of the product manufactured by a third party has not been compromised"**.

This dissertation aims towards developing verification methodologies and implementing non-destructive testing strategies to ensure the authenticity of a third party IP. This can be accomplished at various levels in the IC product life cycle. At the design stage, special testability features can be incorporated in the circuit to enhance its overall testability thereby making the otherwise hard to test portions of the design testable at the post silicon stage. We propose two different approaches to enhance the testability of the overall circuit. The first allows improved at-speed testing for the design while the second aims to exaggerate the effect of unwanted tampering (if present) on the IC. At the verification level, techniques like sequential equivalence checking can be employed to compare the third-party IP against a genuine specification and filter out components showing any deviation from the intended behavior. At the post silicon stage power discrepancies beyond a certain threshold between

two otherwise identical ICs can indicate the presence of a malicious insertion in one of them. We have addressed all of them in this dissertation and suggested techniques that can be employed at each stage. Our experiments show promising results for detecting such alterations/insertions in the original design.

Dedicated to

Swapna Banga,

Ranendranath Banga

and

Mou Banga

whose inspiration

have always motivated me

towards my goal.

Acknowledgments

It gives me great pleasure to thank my advisor Dr. Michael S. Hsiao for his sustained guidance and valuable advice throughout the duration of my research. His insight in my field of research and timely suggestions enabled me to comprehend and approach the target problem in a very methodical way. His continuous encouragement always inspired me to strive for the best.

I would like to thank Dr. Sandeep Shukla for his tedious effort to go through my entire thesis and pointing out areas of improvement. I would like to thank Dr. Leyla Nazhandali for her suggestions to broaden the scope of my approaches. I would like to thank Dr. Lynn Abbott and Dr. Mark Shimozono for their consent of being a part of my Doctoral Committee and providing valuable suggestions to improve my thesis composition and research direction. I would also like to thank Sanjay Sengupta and Matthew. C. Youell for giving me opportunities of internship in Intel which helped me gain breadth in my field of research, especially from the implementation point of view.

I am indebted to my friends in the PROACTIVE Lab. viz. Vishnu Vimjam, Lei Fang, Weixin Wu, Shrirang Yardi, Maheshwar Chandrasekar, Karthik Channakeshava, Ankur Parikh, Anupam Shrivastava, Xueqi Cheng, Nannan He, Harini Jagdeeshan, Swapneel Dong-likar, Sandesh Prabhakar, Min Li, Neha Goel, Wei Hu, Nikhil Rahagude, Supratik Mishra, Sarvesh Prabhu, Dhrumeel Bakshi and Huy Nguyen for sharing their different viewpoints in my research work and constructive amendments that helped in shaping up the solution in a more complete way. They made my stay in PROACTIVE a memorable and enjoyable

experience.

I am grateful to my roommate Sumit Ahuja for his constant feedbacks on my thoughts which guided my research in a big way. I will cherish the good times and light moments that I have had with my friends Vipul Chawla, Anupam Shrivastava and Ananya Ojha during the course of my research. I would like to thank Mike Henry for his help on backend processing required for some of my experiments. Finally, I would like to extend my heartiest thanks to my mother Swapna Banga, father Ranendranath Banga, my sister Mou Banga and all my relatives and friends whose well wishes always motivated me towards my endeavor.

Mainak Banga

December 7^{th}, 2010.

List of Publications

1. **Design-for-Test Methodology for Non-Scan At-Speed Testing**, Mainak Banga, Nikhil P. Rahagude and Michael S. Hsiao, IEEE International Conference on Design, Automation and Testing in Europe (DATE'11)[Grenoble, FRANCE], March 2011.

2. **Trusted RTL: Trojan Detection Methodology in Pre-Silicon Designs**, Mainak Banga and Michael S. Hsiao, IEEE International Symposium on Hardware Oriented Security and Trust (HOST'10)[Annaheim, USA], June 2010.

3. **Kiss the Scan Goodbye: A Non-Scan Architecture for High Coverage, Low Test Data Volume and Low Test Application Time**, Michael S. Hsiao and Mainak Banga, IEEE Asian Test Symposium (ATS'09)[Taichung, TAIWAN], Nov 2009.

4. **Fast Circuit Topology Based Method to Configure the Scan Chains in Illinois Scan Architecture**, Swapneel Donglikar, Mainak Banga, Maheshwar Chandrasekar and Michael S. Hsiao, IEEE International Test Conference (ITC'09) [Austin, USA], Nov 2009.

5. **VITAMIN: Voltage Inversion Technique to Ascertain Malicious Insertions in ICs**, Mainak Banga and Michael S. Hsiao, IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'09) [San Fransisco, USA], July 2009.

6. **A Novel Sustained Vector Technique for the Detection of Hardware Trojans**, Mainak Banga and Michael S. Hsiao, IEEE International Conference on VLSI Design (VLSID'09) [New Delhi, INDIA], Jan 2009.

7. **A Region Based Approach for the Identification of Hardware Trojans**, Mainak Banga and Michael S. Hsiao, IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'08) [Annaheim, USA], June 2008.

8. **Guided Test Generation for Isolation and Detection of Embedded Trojans in ICs**, Mainak Banga, Maheshwar Chandrasekar, Lei Fang, and Michael S. Hsiao, Proceedings of the IEEE/ACM Great Lakes Symposium on VLSI (GLSVLSI'08) [Orlando, USA], May 2008.

Contents

List of Figures

List of Tables

List of Abbreviations

3PIP	Third Party IP
ATPG	Automatic Test Pattern Generation
BDD	Binary Decision Diagram
BIST	Built-In Self-Test
CAD	Computer Aided Designing
CEC	Combinational Equivalence Checking
CMOS	Complimentary Metal Oxide Semiconductor
CNF	Conjunctive Normal Form
CRP	Challenge Response Pair
CSA	California Scan Architecture
CUT	Circuit Under Test
DARPA	Defense Advanced Research Project Agency
DFF	D Flip-Flop
DFM	Design For Manufacturability

DFT	Design For Testability/Design For Trust
DNF	Disjunctive Normal Form
DOD	Departemnt of Defense
DUT	Design/Device Under Test
EC	Equivalence Checking
EDA	Electronic Design Automation
FSM	Finite State Machine
I/O	Input-Output
IC	Integrated Circuit
IP	Intellectual Property
ILS	Illinois Scan
LSB	Least significant Bit
LFSR	Linear Feedback Shift Register
MB	Megabyte
MC	Model Checking
MSB	Most significant Bit
NSA	National Security Agency
PPI	Pseudo Primary Input
PPO	Pseudo Primary Output
PUF	Physically Unclonable Functions

R&D	Research and Development
RAM	Random Access Memory
ROBDD	Reduced Order Binary Decision Diagram
RSA	Rivest, Shamir and Adleman (Encryption Algorithm)
RTL	Register Transfer Level
SAT	Propositional Satisfiability
SCOAP	Sandia Controllability/Observability Analysis Program
SEC	Sequential Equivalence Checking
SoC	System On Chip
SRAM	Static Random Access Memory
TSMC	Taiwan Semiconductor Manufacturing Company

Chapter 1

Introduction

1.1 How it all started

The unprecedented growth of semiconductor industry over the past four decades has been largely governed by the Moore's Law [59] which states that *"the number of components on the silicon die (Integrated Circuit or IC) will double every year"*. This law which was predicted in 1965 based on contemporary trends, later became a self-fulfilling prophecy that helped the industry gain the magnitude it is today. Till date, competency of any chip manufacturing organization is judged by the fact whether is it able to keep pace with *Moore's Law* or not. The paranoia that their competitors are going to achieve it sooner has pushed designers in every semiconductor organization uphold the validity of Moore's prediction.

While increase in the number of transistors squeezing in more functionalities in an IC has been one of the growth trends in the semiconductor industry, decreasing cost per unit component has been another one! Path breaking improvements in wafer processing technology, innovative breakthroughs in chip manufacturing materials and creation of sophisticated tools for Electronic Design Automation (EDA) are some of the important factors that have accelerated and eased chip designing/manufacturing process thereby boosting the cost mini-

mization trend. Semiconductor companies rely on bulk consumption to ensure a profitable business. To this end, reducing the overall production cost translates into maximizing profit on the product thereby prompting the IC design organizations to seek for locations that are economically most viable for designing/manufacturing their product. Since manpower cost contributes a significant portion in the overall cost of the product, design houses have resorted to outsourcing of various manual labor intensive stages to other geographical locations where the cost of operation is cheaper.

1.2 Outsourcing - The Inevitable Problem

Since a third party is involved in outsourcing the most important question that arise is *the question of security and trust*; i.e. how to ensure that the integrity of the intended design has not been compromised overseas during the process of manufacturing. Business relations are driven by political and socio-economic factors. If a manufacturer has a malicious intent, he/she may introduce subtle alterations in the original design to make it work contrary to the normal in special rare situations. Under the presence of such malicious modifications the design is just like a ticking *time bomb* waiting for the trigger to explode. A faulty part is acceptable in the sense that most of them can be detected during production testing and post silicon validation phase. For those that escape testing, a random operational failure is unlikely to cause any catastrophic consequence. To the contrary, an intelligently tampered IC is really severe to handle. Once triggered off, it can cause catastrophic consequences. Thus, ensuring the sanity of the parts imported from the overseas third party manufacturer emerges as a prominent concern for the parent IC design company. More so, with the shrinkage of the technology nodes and increase in the process variation and leakage, distinguishing a behavioral anomaly from a process variation effect is proving to be even more difficult.

In an effort to mitigate the risk arising from structural and technical vulnerabilities of third party IPs, DARPA came up with the “Trusted Foundry Program" in 2007 [24]. The ICs used

by DOD are very specialized ones and the national labs in SANDIA, NSA and Honeywell etc. cannot provide the performance, volume and variety of DOD needs. Consequently, they need to be desiged/manufactured by parties/foundries which are not under direct surveillance from national security agency. Thus DARPA came up these five hard to answer questions:

• How do you trust the design cycle to faithfully generate only the microelectronics desired?
• How do you trust microelectronics chips when they are manufactured in a non-trusted facility, such that they will faithfully perform only the function they are designed for?
• How do you trust that the testing on the microelectronic chips will faithfully determine that the chip will operate only as designed. (no more - no less)
• How do you know that the packaging of the chip does not introduce features into or misidentify the chip?
• How do you determine that the packaged chip has not been tampered with after installation, and how do you communicate the fact of tampering?

No instance of actual tampering has been yet found, yet instances of numerous unexplained military mishaps remains unsolved! [1] In [47] the author has given a detailed description of how reverse engineering is applied on chip production masks to decipher the original design. Once a design is deciphered it is relatively easy for an adversary to tamper it. Labs specializing in reverse engineering do exist which includes *Chipworks*, *IBM* etc. Advent of sophisticated CAD tools have relatively eased the human effort on reverse engineering process. Taking these factors into consideration, DARPA undertook the program for "Trusted ICs" wherein the aforementioned questions needed to be addressed.

1.3 Existing Security Techniques - The Drawback

Security features in ICs have existed from long time to prevent IP theft. *Watermarking* entails sculpting the information directly and imperceptibly into the original data. *Water-*

marking can be visible wherein the watermarked part is a visible picture or invisible where the watermarked data cannot be perceived. *Watermarking* is employed for copyright protection, distribution tracing, authentication, and authorized access control [79].

Testing features like *scan-chains* can be used to extract out the sensitive data from the designs. Thus techniques like *scan-chain encryption* have been proposed to mask the original test data with a predetermined mask and only the authorized persons who have the key for the mask are able to decipher the test results.

More recently, variations introduced in the systems because of physical randomness of processing technology have been exploited to characterize an IC. Since reproducing the exact physical conditions for creating a *clone* of an IC is virtually impossible, this technique is called as *Physically Unclonable Function* (PUF) protection technique [27, 33, 78]. A PUF entails a challenge-response authentication. The applied stimulus is called the challenge, and the reaction of the PUF is called the response. A specific challenge and its corresponding response together form a challenge-response-pair or CRP. When a physical stimulus is applied to the PUF structure, it reacts in an unpredictable way due to the presence of randomness [55, 56, 60]. Randomness in PUFs can be created externally or internally. While optical and coating PUFs belong to the former category, the later category includes silicon PUF, SRAM PUF, butterfly PUF and magnetic PUFs. Although PUFs are very reliable in ensuring that the internal data embedded in the ICs are not compromised in any way, they cannot prevent on-chip intrusions. An expert adversary can still contaminate the original design with an undesirable logic of own to make it act contrary to the normal.

Figure 1.1 shows the steps where infiltrations can occur. Starting from the specification phase, a third party can introduce alterations while converting the specification into a behavioral model, a behavioral model into a gate level netlist, while scan-chain insertion or while converting the final design into a physical layout. Tampering can be done even in the fabrication process by tampering the actual silicon. Recently, almost any portion of the design flow is subject to outsourcing and so the threat of tampering is not only restricted to

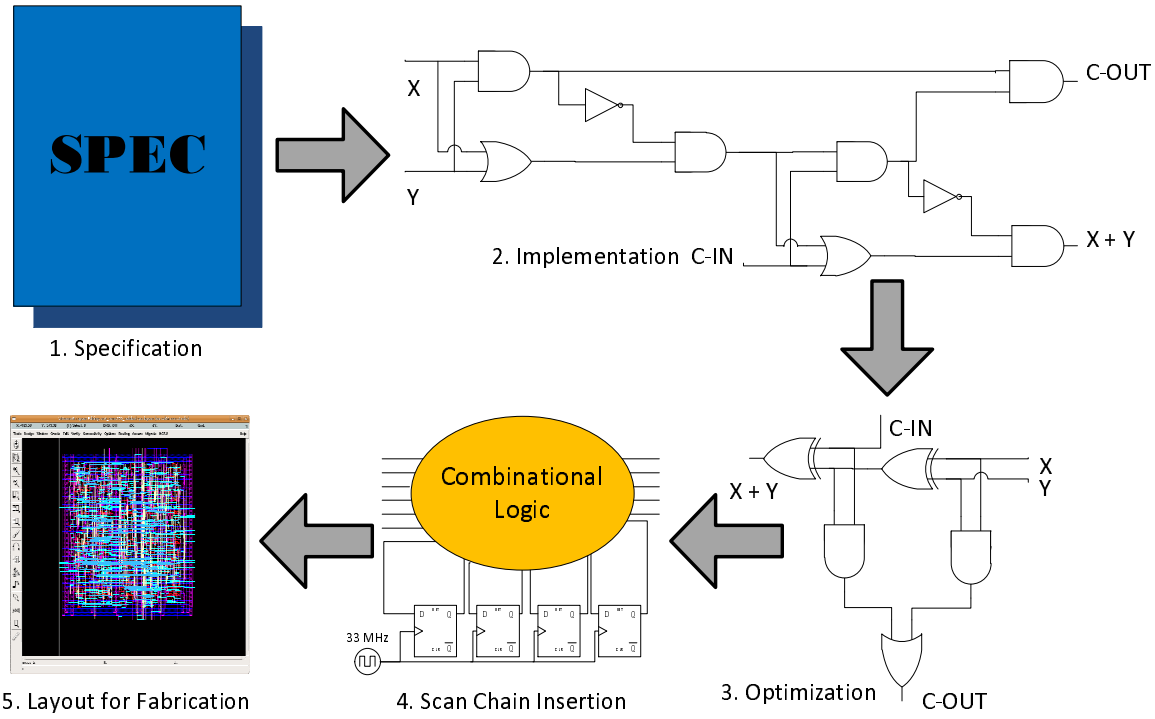


Figure 1.1: Stages where Trojan can be implanted/Test mechanism can be enhanced in a design cycle

the post silicon stage.

Hence checking a third party design for the presence of intentional tampering is an issue of key importance before the parts can be certified as safe to be used in a real application. Else, operational disasters can be imminent. Since such tampering are selectively implanted, so it is really difficult to distinguish the malicious parts from the good ones with conventional testing mechanisms. With outsourcing gaining so popularity in the IC production flow, the obvious direction is to look for ways preventing such tampers and is emerging as a critical and exciting area of research. Devising non-destructive ways of distinguishing the tampered parts from the genuine ones is the requirement of the industry today.

Ensuring integrity is essential at both pre and post silicon stages in the IC design flow. A modern IC may contain one or more software IPs delivered by a third party vendor. Thus there is a chance of intrusion at the software IP level. At this level using formal techniques like satisfiability (SAT) based equivalence checking can be utilized to sieve out the suspicious

signals and then check more thoroughly if the suspicious signals are indeed malicious! Also it is possible that the final silicon is manufactured by an overseas fabrication unit. This creates a chance of tampering at the post silicon level. Conventional testing like stuck-at-faults, delay faults, IDDQ testing are not effective against detecting such alterations. This means that the designer should either incorporate special test capabilities in the design that are intended for checking the authenticity of the final product or they devise ways to exaggerate the behavioral difference of a tampered third party IC from a genuine one beyond the normal range of process variation. In this dissertation, we have discussed the problem in detail at all levels and have proposed/will propose techniques that can be incorporated in the IC design and production road map to ensure a risk free operation.

1.4 Contributions of this Dissertation

This dissertation addresses all the three levels of intrusion we pointed out in the previous section.

In our **first contribution** we architect a new design-for-testability approach that accelerates the testability of the circuit to a great extent while maintaining very high fault coverage without using scan. This approach ensures that all the portions of circuit has been exercised against any manufacturing defects before proceeding to check for Trojan implantation. Later, we also show that the same technique can be employed to create an obfuscated design making the insertion of Trojans in such circuits as non-trivial. Finally our experiments demonstrate that functionally hard-to-detect implantations are easily observable at the outputs under a partitioned state-space scheme.

Our **second contribution** proposes a novel design for manufacturability (DFM) methodology that uses a *voltage inversion scheme* to expose Trojan activity on primary output(s). Apart from that, in *inverted voltage* mode the power profile for the genuine and tampered ICs is shown to markedly different, while it is not so in the normal mode.

Our **third contribution** employs a *functional and N-detect ATPG* technique followed by *suspect signal guided sequential equivalence checking(SSG-SEC)* scheme which is very effective in indicating a Trojan behavior in pre-silicon designs. We reinforce the whole methodology with a region identification scheme to filter out a subset of probable candidates for Trojan activity.

In our **fourth contribution** we devised a post silicon test methodology using a *sustained vector simulation* technique. Experimental results show that this method is very effective in keeping the over all circuit activity low thereby increasing relative activity of Trojan portion making them observable. We also weigh the gates based on their activity to assess whether it is tied to a Trojan or not.

1.5 Organization of this Document

The rest of the dissertation is organized as follows:

• Chapter 2: This chapter introduces Trojans and their classification based on different parameters. A categorized description of the previous works on Trojan detection is presented. Thereafter we describe the key concepts and techniques that are/will be used to formulate the Trojan detection techniques. A mathematical formulation illustrates why Trojan Detection is considered such a difficult problem. Finally, we discuss the construction of Trojan circuits for a given design in detail.

• Chapter 3: This chapter details the 3-stage design-for-test methodology for accelerating the test procedure of sequential circuits without using scan. This method can be applied at-speed and hence reduces the test application time by orders of magnitude. Here we inflate the functional state space of the design by using the $\bar{Q}$ signal of a DFF. The extended state space is able to detect many otherwise hard-to-detect faults. At-speed testing helps catching delay related defects as well. Also this technique is shown to be useful in detecting Trojans. The extended state-space is useful for creating necessary excitation/propagation condition

which is otherwise very difficult in functional mode.

• Chapter 4: This chapter proposes a methodology that combines *diluted sequential equivalence* checking combined with a *N-detect Automatic Test Pattern Generation(ATPG)* to distinguish two software versions of the same IP which differ substantially in netlist structure. In this method, we progressively filter out a very small subset of signals that are prime suspects for the Trojan. Further we plan to do a region mapping on the gates represented by the remaining faults and attempt to predict the possible location(s) of Trojan in the IP.

• Chapter 5: This chapter explains how inverting the operating voltages of the IC can reveal the presence of malicious insertions in it. We call this method as the *voltage inversion technique*. Theoretical analysis shows that voltage inversion is very effective in activating the Trojans and hence there is a high chance that their effect will be much more visible on the IC outputs. Moreover, in *inverted voltage* mode power difference between the genuine and tampered IC can be substantial as compared to the normal operational mode.

• Chapter 6: This chapter describes a *sustained vector simulation method* that proves to be very effective in pronouncing the Trojan effect in a design. This technique also helps pointing out suspect locations in the IC where the Trojan is most likely to be inserted. Experimental results show order of magnitude improvers in behavioral discrepancies using this technique.

• Chapter 7: This chapter discusses the future research that is possible in this directions and concludes the dissertation.

Chapter 2

Background

This chapter provides a lucid overview of the concepts and existing techniques that we have incorporated to formulate our theoretical analysis and explain the results obtained. We have defined the terms and notions that have been used throughout the rest of the thesis. The chapter starts with the definition of a Trojan, its characteristics and detailed classification. Then we explain side-channel signal analysis which is a key non-destructive methodology for predicting IC behavior. Side-channels can include power information, delay information or the current consumed to characterize IC behavior. A brief overview of the works that have already been accomplished in the field of Trojan detection using these parameters follows next. We discuss why the problem of Trojan detection is difficult and challenging with a mathematical formulation. Finally, we conclude the chapter by describing methodologies to create *combinational* and *sequential* Trojans that we used in our work.

2.1 Trojan and Characteristics of Trojan

Trojans can be defined as - *A subtle alteration in the intended design structure aimed to alter its performance (it can be a change in physical behavior or functionality) drastically under*

certain rare and critical situations. Such alterations can be addition of extra logic, addition of extra wires to make undesired connections, removal of existing logic or even varying the design geometries. In its most basic structure, Trojans are mainly categorized as:

• **Combinational Trojan**: A combinational circuit that is triggered under a specific condition of certain internal signals.

• **Sequential Trojan**: An finite state machine (FSM) that monitors a portion of the internal circuit signals and triggers the output upon the occurrence of specific sequence(s).

In general, *Trojans* share some common attributes. They are:

1. *Trojans* are minuscule when compared to the size of the design in which they are implanted. They can be very well accommodated within the non-used area of the IC thereby keeping the IC dimensions intact even with their presence in it. Consequently they do not incur any silicon area overhead on the die. Since they are activated by internal signals and inject their effect in some other internal signals, they do not require additional output for taking effect. Hence they do not increase the pin count either.

2. They are stealthy in nature. Activation of a Trojan depends on very rare triggering scenarios that are difficult to produce during conventional testing. Even scan based testing cannot expose the Trojan because the triggering condition is unknown.

3. They are intentionally malicious. Triggering of an *Active Trojan* affects one or more output(s) of the IC, changing the desired behavior of the device in a way such that the end result is detrimental. A *Passive Trojan*, on the other hand, do not affect the functionality of the device. Instead it may transmit secret information to an adversary [1] who can use it to deter the performance of the device or may induce some physical changes like excessive heat generation to make the device fail.

4. The Trojans may be dormant for the most part of their life. Under normal operating conditions of the IC, *Active Trojans* do not interfere with the logical behavior of the

device unless the triggering condition arises. So there is no difference in the outputs of a tampered IC from that of a genuine one.

2.2 Trojan Classification

Apart from the broad classification stated earlier, depending on the point of view, Trojans are classified in many different ways. A nice explanation of such a hierarchical distribution has been presented in [80] and a part of it is reproduced here for a quick overview. As shown in Fig 2.1, Trojans can be distinguished based on their *physical characteristics*, *activation characteristics* or *action characteristics*. *Physical characteristics* can be further based on *type*, *size*, *distribution* and *structure*.

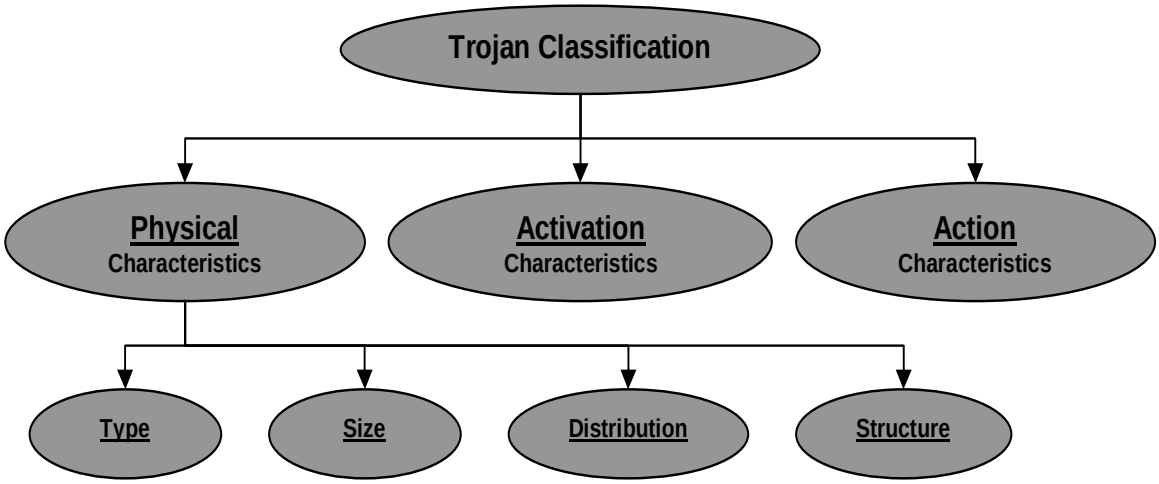


Figure 2.1: Taxonomy of Trojans

2.2.1 Trojan Classification Based on Physical Characteristics

The detailed classification of Trojan based on its physical characteristics has been shown in Figure 2.2.

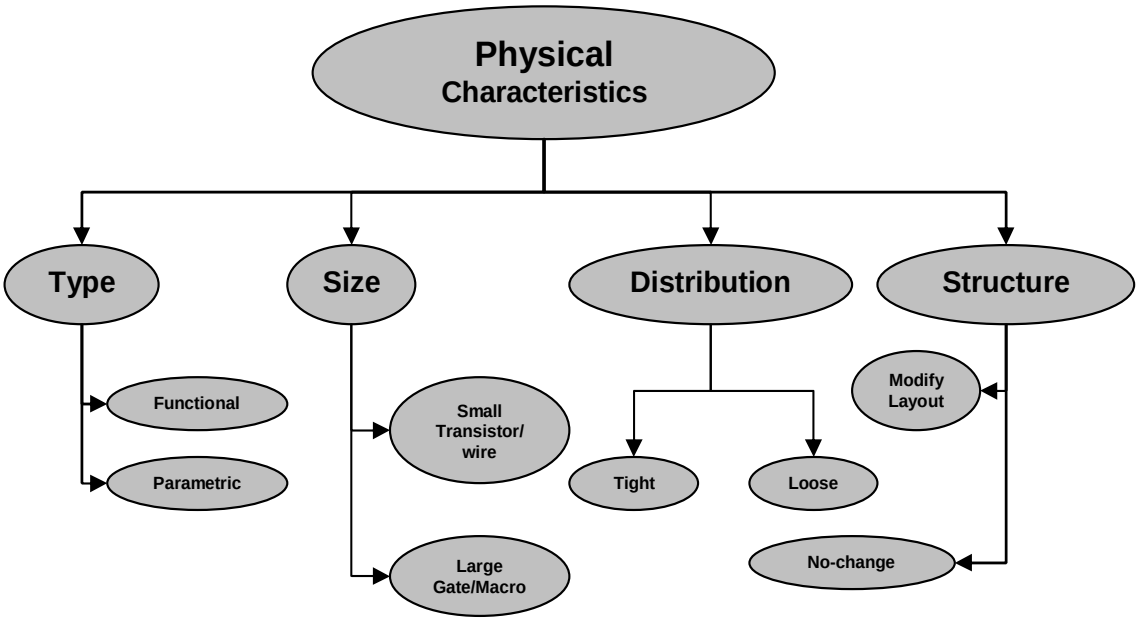


Figure 2.2: Trojan classification based on physical characteristics

• **Type** : There are two sub categories in *type* section. The *functional type* of Trojan includes Trojans that are framed by manipulating the original structure of the design. This includes addition or deletion of gates from the circuit. *Parametric type* of Trojan involves modifications of existing wires and logic. This can include thinning of interconnect wires, the weakening of transistor strength by varying the length-width ratio.

• **Size** : This accounts for the overhead incurred in implanting the Trojan. Size can be an important factor for Trojan activation and detection. *Large/macro* Trojans are relatively easier to detect at lower frequencies because the leakage current consumed by Trojans is directly proportional to its size and hence a substantial Trojan will have a considerable leakage power consumption and the variations are more easily observed at a lower operating frequency [12]. On the other hand, *small transistor/ wire* Trojans may be easier to activate because they require less constrained conditions to trigger them.

• **Distribution** : This refers to the location(s) of Trojan on the IC. While a *tight* Trojan consists of a few gates topologically coalesced together in a localized area, *loose* Trojan consists of gates distributed all over the circuit or portions of a circuit. Flexibility of Trojan fabrication depends on the available space in the original layout. Hence a malicious third party manufacturer has to choose a proper distribution of the required components to design the Trojan.

• **Structure** : Changing the structure of the IC affect the power, delay characteristics of the device. Structural changes especially that leading to a *layout modification* is very difficult to achieve. Thus to incorporate such changes which inevitably requires the layout to be reconfigured, the adversary is likely to use a Trojan with a very small physical *footprint. Physical footprint* is measured in terms of the area, power consumption, delay characteristics and other such factors. Since for a functional Trojan size and distribution have significant impact on the original footprint of the Trojan, for larger Trojan sizes, distributing the components across the layout can assist in reducing the impact on the power and delay characteristics thereby making it more stealthy. Trojans which require *no-change* in the original layout uses the existing spare cells in the original design to fabricate the Trojan circuit in which case there will be *no change* in the original circuit structure.

2.2.2 Trojan Classification Based on Activation Characteristics

Trojans can be classified according to their activation characteristics. This has been shown in Figure 2.3. Activation characteristics refer to the conditions which triggers the Trojan towards its objective.

Broadly speaking, there are two types of Trojans based on activation -

- **Externally Activated Trojans** - These Trojans are those whose triggering conditions are controlled using the input pins of the IC. Thus it is on the part of the operator as to when he/she wants to trigger the Trojan. This can be done by monitoring the

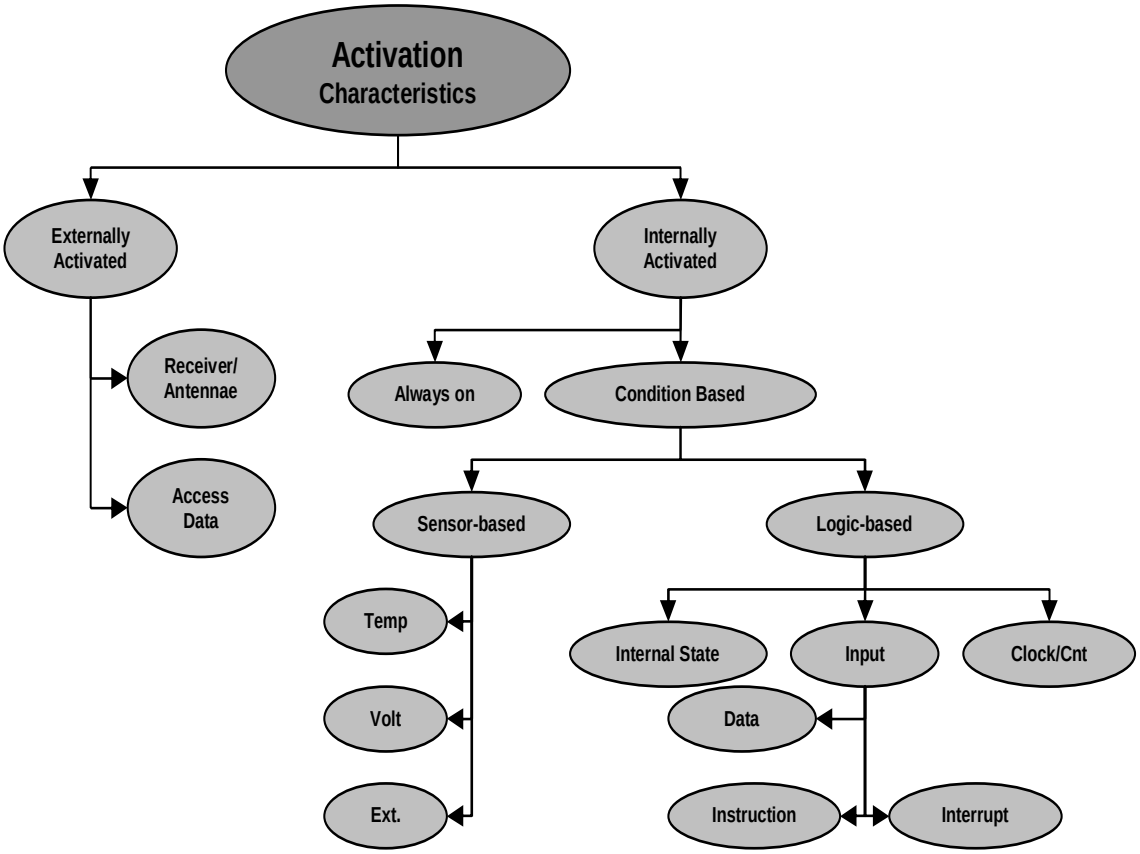


Figure 2.3: Trojan classification based on activation characteristics

device conditions using a side channel signal that transmits the internal information of the device and using that information to appropriately trigger the device.

- **Internally Activated Trojans** - These Trojans monitor the internal configuration of the system for its operation i.e. they derive their condition for activation from the existing internal environment. *Internally Activated Trojans* are subdivided into two categories:
  - *Always-on Trojans* are perennially active and can get triggered any time inside the operating device. This includes Trojans like thinning of transistor wires, changing the drive strength of the transistors by tampering their length-width ratio etc. A

device with such change may start working exactly as a normal device but it will fail whenever the internal conditions the exceed the physical threshold supported by the fabricated device. Thus their failure cannot be accurately predicted but only a statistical probability of occurrence of such a failure can be estimated.

- *Condition based Trojans* are much more intelligent. They wait for the circuit to enter a specific configuration or the state bits to attain a specific value to get activated. *Condition based* Trojans are further classified based upon the conditions that triggers them. Thus they can be -
  - *sensor based Trojans* whose activation depends on values/thresholds of some physical parameters like temperature, voltage or any types of external environmental conditions like pressure, humidity, electromagnetic interference that is monitored by the sensor.
  - *logic based Trojans* are those where the logic inside the Trojan intelligently monitors the internal circuit environment to get triggered. Example of *logic based Trojans* are counter Trojans, sequence-detector Trojans etc.

2.2.3 Trojan Classification Based on Action Characteristics

The third classification of Trojans is on the basis of their *action characteristics*. This has been shown in Figure 2.4. *Action characteristics* describe the effect of triggering of the Trojan on the underlying design. These are of three types:

• **Modify Function** - These Trojans change the original functionality of the logic. This can imply removal of a portion of the logic to remove a property, disable some functionality to cause an operational failure or addition of extraneous logic to realize something additional to what is intended.

• **Modify Specification** - This class of Trojans change the properties of the chip such as delay to realize their intended objective. These are similar to *parametric Trojans* discussed

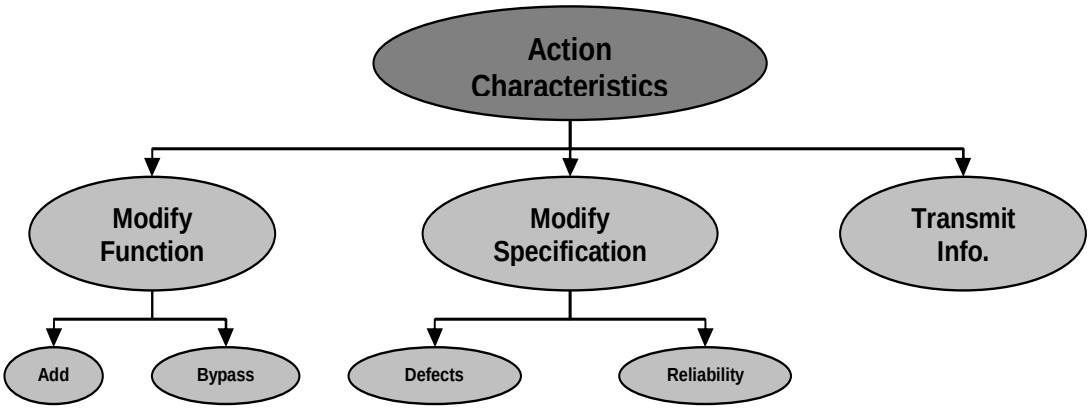


Figure 2.4: Trojan classification based on action characteristics

earlier. *Parametric Trojans* tamper the strength like the fanout supporting capability of an output of a gate, ability to supply a desired current through a wire etc. Changing the strength of the wires or gates affects the delay of the combinational path and hence fall within this category.

• **Transmit Info** - This type of Trojans don't interfere with the operation of the device. It has been proved that side-channel signals can be decrypted to reveal important internal information embedded within the device. Trojans under this class emit signals containing such key information. This information can be misused by an adversary.

2.3 Related Works in Trojan Detection

In this section, we detail the works that has been accomplished so far in this direction by different researchers. Trojans can affect different circuit parameters as was explained in its classification. So their detection mechanisms also vary. Based on detection mechanism, we can classify the previous works to target on *delay* based Trojans, *logic* based Trojans or *parametric* Trojans.

2.3.1 Trojan Detection Based on Delay

One of the first works on *delay* Based Trojans was authored by Li and Lach [50]. In this work, authors use a negatively skewed clock to characterize the delay of the functional paths. As shown in Fig. 2.5 any combinational path in the circuit is contained between a *Source* register and a *Destination* register. To characterize the delay signature of this path, a third register called as the *Shadow Register* is kept in parallel to the *Destination* register. But unlike the *Source* and *Destination* registers which are clocked by the system clock, the *Shadow* register is clocked by a different clock which is skewed negatively with respect to the system clock. The comparator compares the output of the *Destination* and the *Shadow* registers and the *Result Bit* is set to 1 if the outputs differ.

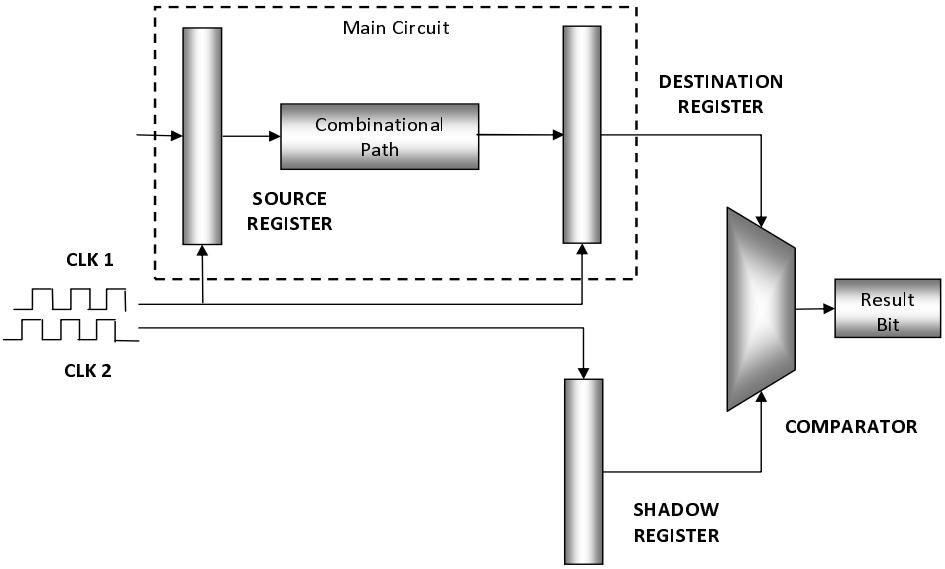


Figure 2.5: Delay characterization with negatively skewed clock

Initially when the clocks are perfectly in phase, the *Result Bit* is 0. Then progressively the skew of the second clock is increased and a set of vectors exercising different portions of the circuit are applied. For each skew step, the result of the chain of *Result Bits* are scanned out and observed. The paths for which the result bit are set to 1 are characterized by the set of vectors and the amount of applied skew. This process is repeated until all the paths are

characterized (i.e. the *Result Bit* is set to 1). The characterization has to be carried out on a large number of samples to ensure that the on-die process variation effects smoothen out.

If a Trojan consisting of an additional logic is inserted in the existing logic, the delay of the corresponding combinational path increases invariably. In such a case, the path signature shows a 1 in a lesser value of applied clock skew to the second clock. Thus the delay characterization fails to meet the desired criteria and the chip will be considered as a tampered one. The exact *Result Bit* that shows the discrepancy in signature also indicates the portion where the Trojan might have been implanted.

This method of Trojan isolation might work if one is able to trigger the Trojan so that the controlling event is able to reach the *Destination* flip-flop. Trojan by nature remain at the non-triggered value for most part of their operational life cycle. Unless a transition is propagated through the path containing the Trojan, it very difficult even to assess its presence at that location. In Fig. 2.6, it is very unlikely that the transition delay of the single OR gate is going to affect the total delay of the path on which the Trojan resides. Since the Trojan input remains at 0 for most part of the operation, the path is transparent to its effect.

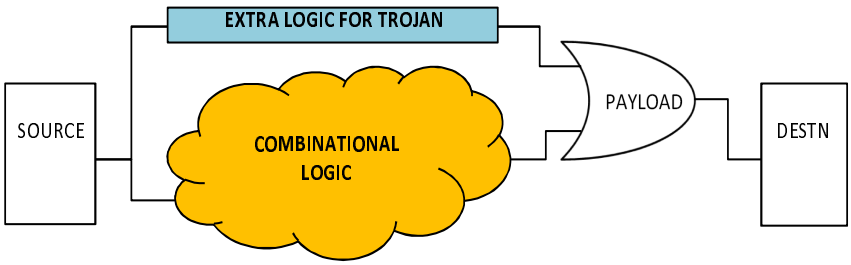


Figure 2.6: Problem of detecting the Trojan with delay testing - the effect of payload is difficult to characterize

In another work reported by Jin and Makris [42], they use the delay patterns of all the paths in the circuits at any particular output as the signature. The level of authentication that can be guaranteed is directly proportional to number of paths that can be delay tested at the outputs. They use a genuine circuit netlist to produce a high coverage delay fault test

set. The effect of each of these patterns are observed at each output. The experiment is repeated with a sample set of test ICs which are then destructively tested to assure they were indeed genuine. Experiments with multiple ICs ensure that the effect of process variation gets canceled out.

Based on the data obtained from the experiments, the points are plotted on a three dimensional chart. Since managing so much data is cumbersome, so a convex hull that contains all the test data inside it is constructed to represent the delay profile of the IC. The authors have used Quickhull algorithm to construct the convex hull. Once the hull is ready, the delay profile of any other sample of the same IC is compared with it. If the delay profile points mostly lie outside the convex hull, it indicates a suspicious behavior and the IC is likely to be infected with Trojans.

A third work reported by Rai and Lach [69], the authors have attempted to assess the effectiveness of the already proposed delay based Trojan detection technique in [50]. First they perform a static timing analysis to extract design specific facts that can be used in the next stages of testing. One of such information is the skew step value. In the second stage, they perform a dynamic analysis - first on the design without a Trojan and then on the design with a Trojan. The results at this stage redefines the skew step that detects the Trojan. Then this process is repeated with parameter variation to determine the new value of skew step at which the design fails. Results show that the statistical distribution of delay profile differs significantly for those ICs which are affected by Trojans as compared to those which are genuine, even in the presence of process variation. This led the authors conclude that delay based techniques using statistical analysis are indeed an effective measure to detect the presence of a Trojan in an IC.

2.3.2 Trojan Detection Based on Power

One of the first works on logic based Trojans was proposed by Agarwal et. al. [12]. This work attempted to construct a signature of the ICs based on the power consumption under varying

process variations. The base circuit was RSA algorithms and they inserted several types of Trojans to tamper its functionality. The process variation was set at three values viz. 2%, 5% and 7.5%. Owing to its minuscule area, the power consumed by the Trojan is very small. It is quite obvious that such a small variation will be overshadowed by the system noise and process variations. The authors used Karhuen-Loeve (KL) expansion method to filter out the noise from the observed power profile of the IC. However the results from their experiments revealed that unless the process variation is really small, it is difficult to conclude the presence of any Trojans. This is turn means that the effectiveness of power estimation based Trojan detection techniques also depend on the accuracy of power estimation techniques. Lot of works on accurate power estimation at various levels of design flow has been done recently. In [9], the authors present a methodology, which utilizes the power estimation knowledge to guide power reduction. The premise to their work is to first provide an accurate and efficient power estimation framework at higher abstraction level and then utilize this information to guide the power reduction algorithm while generating the hardware RTL. In [5], the authors provide a rational for their approach and show that based on modeling style at system-level the speedup in estimation time can be achieved. They experimentally show on variety of benchmarks the range of speedup in power estimation process reaches upto 15 times as compare to RTL power estimation techniques. Such techniques with more accuracy can be used for relative power estimations to detect Trojans. More specifically work in [4], [3] present characterization based power estimation methodology, they utilize GEZEL based co-simulation environment. The error or loss of accuracy on their proposed methodology was less than 10% with respect to lower level power estimates. The estimates from the power estimation methodology/tools are no good if they are not supplied with good representative test vectors. The same authors in their work in [7], [6] present methodology to utilize verification collateral to enhance the accuracy of power estimation at higher level. Similar case studies can help us in finding out trojans or relatively peak power test vectors. In [8], the authors present a power reduction based on clock-gating and sequential clock-gating from high level model description. Reducing the overall power consumption has a positive effect

in Trojan detection, since it becomes more difficult to hide the excess power consumption due to the Trojan.

In [13], we proposed a partition based mechanism to attack the logic based Trojans. In this scheme, we selectively excite specific portions of the state space while ensuring that the rest of the portion of the state space remains as unexcited as possible. Trojans being small, their effect can only be observed if the activity of the entire circuit can be kept at a minimal level. At the same time we needed to ensure that the circuit does not enter a sleep mode with no activity going inside it. For this we kept on stimulating a portion of the state space. If the Trojan is associated with one or more of such portions, then it was observed that their effect is felt when the corresponding portion(s) are excited. Our experiments showed that this method can project the power discrepancies in excess of the process variation.

In [14], we proposed our second partition based method to isolate Trojan infected portions of the circuit. Logic based Trojans are intelligent in the sense that they are activated by specific rare sequence of logic values in the circuits. These logic values are monitored by minuscule state machines which keep track of the sequences encountered. Obviously such state machines are formed by gates and flip-flops that are physically connected. To clusters out such entities, we partitioned the circuit to what we called as *regions*. Each region is centered around a gate in the circuit and is defined by a specific radius. So the total number of regions in a circuit is equal to the total number of gates in it. The concept of a region is shown in Fig. 2.7.

Since the total number of gates in a circuit can be prohibitively large, we needed an effective way to filter out the regions that we want to test. Logic based Trojans wait for specific conditions to trigger them up which means that it is most likely associated with circuit flip-flops. Accounting for this we considered only those regions for testing which contained a specific number of flip-flops within it. We followed the previous approach of selectively exciting each region and then observing the power variations in the genuine and tampered ICs. Our results indicate that this method is successful is sieving out circuit portions that

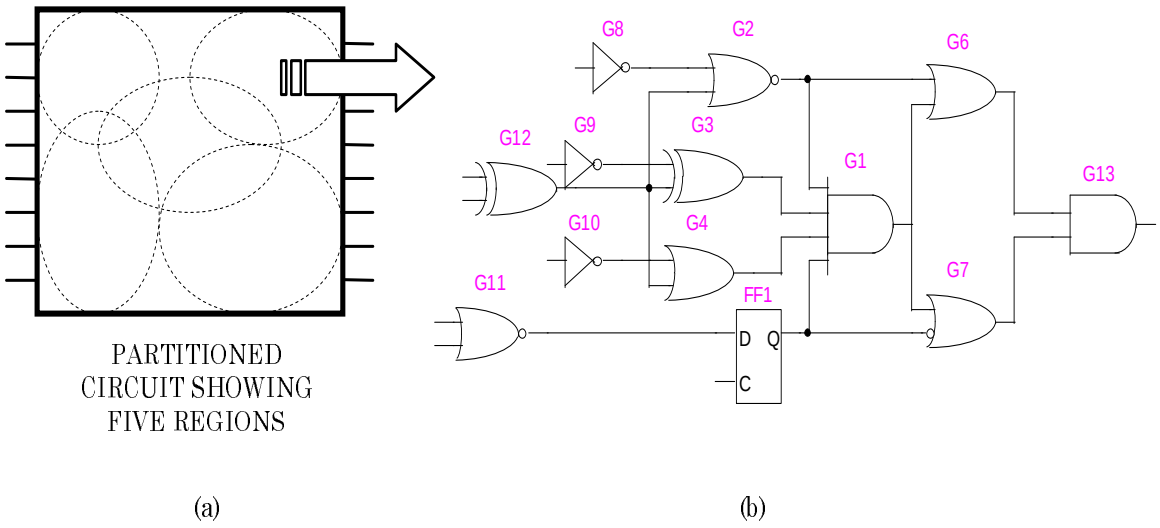


Figure 2.7: Circuit partitioned into regions

are actually associated with the Trojan.

The problem we faced here is that if in a circuit it is difficult to contain the activity of the gates (some circuits are inherently high toggling!), then these methods are not effective for detecting Trojans in such circuits. This has been shown in the results of circuit $s3271$ in [14], although $s3271$ is not a big circuit at all!

2.3.3 Trojan Detection Based on Charge Consumption

Comparing the charge consumption on different ICs to detect Trojans in them was first proposed by Salmani et. al. in [72]. In this work the authors have used a current integration technique to compute the total charge consumed by the circuit over a period of time. Trojans, howsoever small they might be, consumes some amount of current. This can be the leakage current if the gates in the Trojan are not toggling or the dynamic current if a portion of the gates are toggling. Obviously more the number of gates in the Trojan toggles, more will be the current consumption. But since the triggering condition of the Trojan is unknown, one cannot be assured to activate the Trojan fully.

On chip variations affect the current consumption profile. So if the chip is tested from the global power pads, chances are there that the process variation will overshadow the minute additional current consumed by the Trojan. To avert this problem the authors have suggested testing the chip from different power pint simultaneously, each of which is dedicated to a particular region inside the IC. The instantaneous current graph is then integrated (area under the current consumption graph) to compute the cumulative charge that is consumed over a period of time for a given set of predefined test vectors.

First the authors obtain a golden charge consumption profile by testing a set of genuine ICs. Then the effect of process variation is included to reflect the change in the charge consumption curve. Finally the IC under test is subjected to the same testing mechanism and if the charge consumption curve differs from the golden one by a threshold value, the IC is declared to be malicious. Sine there are multiple ports that needs to be accounted, any one port that fails to conform to the observed behavior is sufficient to indicate an anomaly. Besides, we also get a good approximation of the spatial location of the Trojans on the ICs as indicated by the power ports that consume more current! This method is also effective for detecting distributed Trojans where a Trojan circuit is spread out over the entire IC area to make its detection difficult. In such a case, multiple power ports show a behavioral discrepancy.

The limitation of this approach is the assumption that the Trojan is constructed within a localized space. Give that fact that Trojans monitor very specific signals in the circuit for their operation, their locations need not be lumped at one place. If the Trojan gates are distributed over the circuit so that different gates receive power from different voltage rails, then the effect of extra charge consumption will be averaged out over the entire chip instead of a specific portion showing an excess charge consumption. As a result it will be difficult to predict the location of the Trojan. Moreover, distribution of Trojan logic over the chip area will result in a slow growth of the charge consumption curve and it will be difficult to predict if the excess charge consumption is due to the effect of Trojan or process variation.

2.4 Preliminary Concepts

2.4.1 Side Channel Analysis - Power Profile

In manufactured ICs, we normally do not have access to the internal signals within the circuit. Therefore, to assess the internal behavior of such a device during operation, one can analyze parameters like electromagnetic radiation, I/O timing behavior or power profile of the overall system. Such parameters that act like a signature for the device are commonly known as the *side channel signals*. The method of using *side channel signals* to extract internal information of a device is known as the *side channel analysis*, and *side channel analysis* have been effectively used to detect the anomalies in the behavior of a circuit [2,46]. For our approach, we compute the power profile of the genuine circuit under test (CUT). The dynamic power for an IC is proportional to the operating frequency *f*, switching capacitance **C**, and supply voltage **V**, shown in the following expression [67]:

$$\mathbf{P} \propto \mathbf{CV^2f} \tag{2.1}$$

Total power consumed in a circuit is the sum of the dynamic power (given by equation 2.1) and the leakage power. Leakage power consumed by the Trojan depends on its size. Since dynamic power depends on clock frequency, a lower frequency will result in a lower dynamic power consumption. In such a case, if the leakage power is high enough, it will be reflected as a discrepancy in the power numbers between the two CUTs. This was illustrated in [66] by an experiment in which a large Trojan could not be detected when the circuit was operated at 100 MHz, whereas it was detected at 500 KHz. But Trojans are mostly small and their leakage power consumption is negligible submerging it within the process variation and so their response to the clock frequency change is not practically observable. Hence we need a different approach to uncover their presence. All the parameters except the switching capacitance C in the Trojan circuit is same under normal operating condition of an IC. This

means that variation in the switching capacitance should be the distinguishing parameter for analyzing two distinct ICs - one genuine and the other maligned. Effect of switching capacitance is directly proportional to the total number of gates toggling in a circuit for a particular vector pair. Thus our ultimate goal is to induce maximum toggles in the Trojan portion of the circuit.

A power profile represents the pattern of power consumption in a system. Power consumption for any pair of vectors is dependent on the total number of gates that switch which accounts for the changing switching capacitance (other factors in Equation 2.1 remaining the same). In our work we have used the terms activity profile or power profile interchangeably because number of gate switches in a circuit is directly proportional to the dynamic power consumed by it.

Dynamic power profile indicates the variation in the power consumed by a circuit. Variations in power profile may arise because of multiple reasons. Of these, the switching gates and process variation are noteworthy. This is a common observation that power profile of identical design will not be exactly be the same for two different chips. That is, they will differ within a certain range which we call as the *process variation*. In order for the extraneous activity generated by the Trojan to be observable, we must be able to highlight it above the process variation. In [12], they have assumed three distinct values of process variation, viz. 2%, 5% and 7.5%. In our work we have assumed the process variation to be 5% although in some cases, as our results will show, we are able to generate power profile differentials in excess of 7.5%.

2.4.2 Equivalence Checking

Equivalence Checking(EC) entails the formal proof that different representations of the same circuit are indeed equivalent cycle by cycle either from a known reset state (reset-equivalent) or from an unknown state after the application of synchronization sequence. As a design progresses down the flow path, optimizations such as clock tree optimizations, delay path

optimizations, placement and routing optimizations etc. and transformations like behavioral to gate level, gate level to layout level etc. affects the design structure. It is therefore essential to ascertain that the final design that is ready to be fabricated is indeed logically equivalent to what was originally intended.

There are two major strategies for proving equivalence of two circuits i.e. either using a Binary Decision Diagram (BDD) or using the Satisfiability(SAT) solver.

BDDs represent the output(s) of circuit for any truth value assignment to its inputs graphically [17,19,57]. It consists of a binary tree where each node has two branches corresponding to the *TRUE* and *FALSE* assignment of the variable represented by the node. Bryant showed that ROBDDs are canonical structures and so different representations of the same circuit should eventually produce the same ROBDD given the variable ordering remains the same [58]. Hence they are highly preferred for equivalence checking. On the flip side, they have the problem of memory explosion if the circuit size is large.

Propositional satisfiability (SAT) represents a boolean formula in a *Conjunctive Normal Form*(CNF) or *Disjunctive Normal Form*(DNF). The DNF form can be derived from the truth table minimization using *Karnaugh maps*. The CNF can be derived by converting each gate in the circuit in its equivalent CNF representation and conjoining all such gates. Most of the current day SAT algorithms manipulate the CNF. We use the CNF representation in our work.

A CNF is formally represented as:

$$Z = \prod(C) \tag{2.2}$$

$$C = \sum(L) \tag{2.3}$$

where Z is the propositional SAT formula, C represents a *clause* and L represents a *literal*. A *literal* is any Boolean variable in positive or negated form. Disjunction of one or more

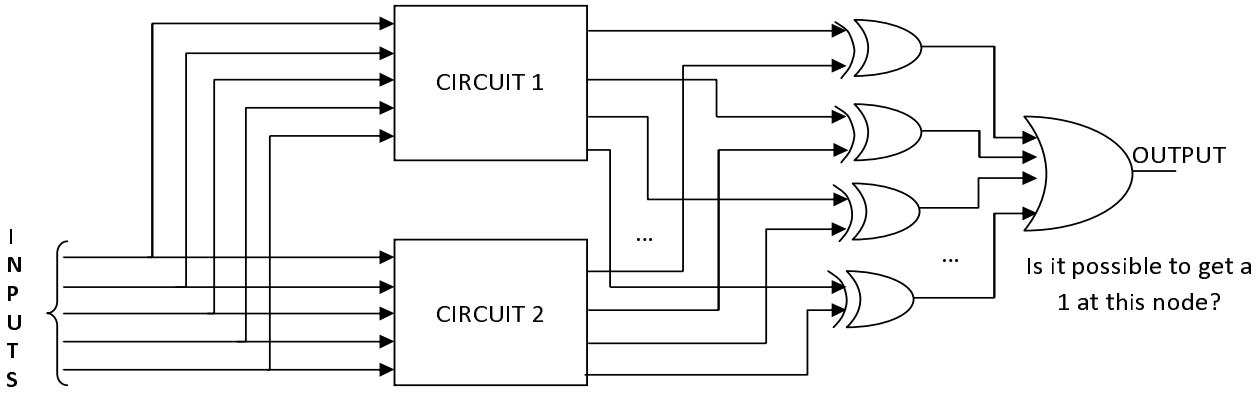


Figure 2.8: Miter to check the equivalence of CIRCUIT1 and CIRCUIT2

literals form a *clause*. Conjunction of one or more clauses constitute a CNF. SAT solvers return an assignment to the variables of a propositional formula that satisfies it, provided such an assignments exists [31, 61, 75, 83].

SAT solvers can be used to check the equivalence between two circuits. Let us consider *CktOrig* and *CktOpt* as the two structurally different instances of the same design. To check the equivalence of these two circuits we conjoin them to create what is called as the *miter* circuit. A *miter* [18] consists of the two circuits with each corresponding inputs tied and each corresponding output XORed as shown in Figure 2.8. The XOR values of all the outputs are ORed to get the final output. Clearly the output of this OR gate will be a '1' if any of the output pair differs in value. In other words, if after constraining the $OUTPUT$ to 1 the SAT solver returns a solution, we can conclude that the circuits are not equivalent. The input assignment returned by the SAT solver run represents the counter example, a vector that can cause one or more pair of corresponding outputs to differ. Else the SAT solver returns $UNSAT$ which means that it is unable to find an input assignment that can cause any of the outputs in the two circuits to differ and so the circuits are equivalent. SAT solvers are more powerful in proving the non-equivalence of two circuits than equivalence. This notion of equivalence is applicable to combinational circuits [21, 25, 41, 54, 68, 73, 76].

Proving sequential equivalence is more involved. For such systems we need to take care of the state variables also. Further, since any sequential circuit behavior is distributed over

different time frames, proving that the circuit instances are equivalent over a single time frame may not suffice. To prove the equivalence of two sequential circuits model checking is used. In Model Checking(MC), the circuits are unrolled k time frames and the *miter* is constructed as described earlier. The initial states in both the circuits are constrained to ensure that both circuits are logically concurrent. Such a state correspondence can be created by using an initialization sequence. The resulting CNF is given to the SAT solver. If the SAT solver returns a solution, we get a counter example. If it returns $UNSAT$, then we cannot conclude anything about their equivalence. Thus we need to increase the bound on k and reiterate until we get a counterexample or reach the diameter of the circuit. Tuning the SAT checks for Equivalence checking has been exploited in several ways. Examples include Robust Boolean reasoning [30], Recursive Learning [48, 74], Exploiting local relations [65], Signal correlation guided solving [52, 53], etc.

2.4.3 Levelized Circuit Representation

Any digital circuit constituting of gates and flip-flops can be modeled as a levelized structures. Any gate in the circuit is assigned its level as per the following equation:

$$\begin{aligned} L(g) &= 0 \mid g \,\epsilon \text{ INPUT or FLIP-FLOP} \\ &= max\{L(\text{FANIN}(g))\} + 5 \text{ otherwise} \end{aligned} \tag{2.4}$$

where $L(g)$ represents the level of gate g and FANIN(g) represents the set of fanins to gate g. Thus all the inputs are assigned the level number 0, all the flip-flops are assigned level number of 5 and level of any other gate in the circuit is 5 more than the maximum level of any of its fanins. The levelized representation of an ISCAS'85 combinational benchmark circuit ($c17$) is shown in Fig. 2.9 where the gate levels are designated as L0 through L20.

Intuitively, gates that are higher up in the level are progressively farther from the input. It is not required that the successive levels be distinguished with an increment of five in the

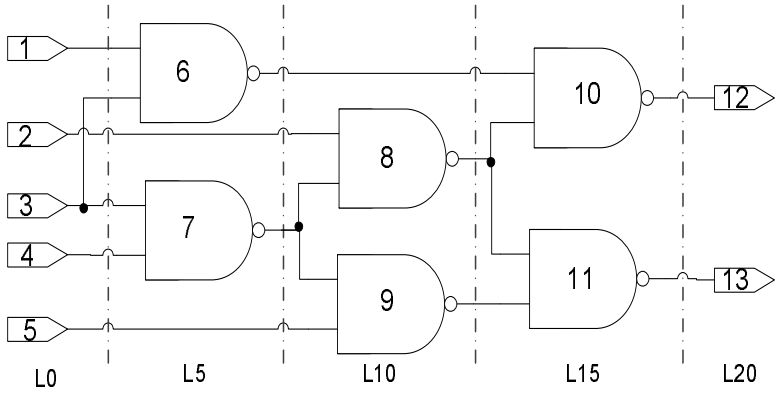


Figure 2.9: Levelized representation of ISCAS benchmark c17

level number but for the other considerations it has been formulated that way. For our purpose, a simple level increment of 1 will suffice provided the level numbers are monotonically increasing as we traverse from the input to the output.

2.4.4 Controllability and Observability

Testability of a circuit depends on the ease with which we can control and/or observe its internal nodes. Thigpen and Goldstein in 1979 proposed the first systematic and efficient algorithm for computing these values [32] which they called **SCOAP** (an acronym for *Sandia Controllability/Observability Analysis Program*). As per **SCOAP** there are six values assigned to each signal s in the circuit. They are:

1. Combinational 0-controllability, CC0(s)
2. Combinational 1-controllability, CC1(s)
3. Combinational observability, CO(s)
4. Sequential 0-controllability, SC0(s)
5. Sequential 1-controllability, SC1(s)
6. Sequential observability, SO(s)

The controllability values range from 1 to ∞ while the observability values range from 0 to ∞. A higher value of controllability/observability on a signal represents a difficult to control/observe signal.

First we shall discuss about these measures for a combinational circuit. Since primary inputs are fully controllable to be assigned as 0 or 1, their 0-controllability (CC0) and 1-controllability value is set to 1. Then we traverse the circuit in a levelized order and successively compute the controllability of all other gates in the circuit until we reach the primary outputs. The controllability value of any gate in the circuit is computed as per the following two conditions:

- If the output value of the gate is determined by a single controlling input then - $Output\ Controllability = \min(InputControllabilities) + 1$
- If the output value of the gate is determined by setting all inputs to non-controlling value then - $Output\ Controllability = \sum(InputControllabilities) + 1$
- If the output can be controlled by a set of inputs then - $Output\ Controllability = \min(ControllabilitiesofInputSets) + 1$

In the above equations 1 is added to account for the increasing logic depth of the circuit as we progressively traverse from the input towards the output. In case a gate has multiple fanouts, 1 is replaced with the fanout count. For a two input gate G with inputs A and B and output Z, the equations for computing controllability values for different gate types is represented in Table 2.1. Here $CC0(A)$, $CC0(B)$ and $CC0(Z)$ represent the combinational 0-controllability and $CC1(A)$, $CC1(B)$ and $CC1(Z)$ represent the combinational 1-controllability of signals A, B and Z respectively.

Observability of the gates of a circuit are computed in the reverse levelized order i.e. starting from the outputs one traverses towards the inputs. Since the outputs are fully observable they are assigned observability value of 0. In observability computation, no distinction is

Table 2.1: Combinational Controllability Equations using SCOAP

Gate	**0-Controllability Equation CC0(Z)**	**1-Controllability Equation CC1(Z)**
AND	$\min(CC0(A), CC0(B)) + 1$	$CC1(A) + CC1(B) + 1$
OR	$CC0(A) + CC0(B) + 1$	$\min(CC1(A), CC1(B)) + 1$
NAND	$CC1(A) + CC1(B) + 1$	$\min(CC0(A), CC0(B)) + 1$
NOR	$\min(CC1(A), CC1(B)) + 1$	$CC0(A) + CC0(B) + 1$
NOT	$CC1(A) + 1$	$CC0(A) + 1$
BUF	$CC0(A) + 1$	$CC1(A) + 1$
XOR	$\min((CC0(A) + CC0(B)), (CC1(A) + CC1(B))) + 1$	$\min((CC0(A) + CC1(B)), CC1(A) + CC0(B)) + 1$
XNOR	$\min((CC1(A) + CC0(B)), (CC0(A) + CC1(B))) + 1$	$\min((CC0(A) + CC0(B)), CC1(A) + CC1(B)) + 1$

made between 0 or 1. The observability of any input signal to a gate in the circuit is computed as the sum of the output observability and the controllability of all other input signals to non-controlling values. This in turn means that the observability of an input net of a gate in the circuit depends on how controllable the other input nets to the same gate are. The observability equations for both the inputs of a two input gate for different gate types are shown in Table 2.2. Here CO(A), CO(B) and CO(Z) represent the combinational observability for signals A, B and output Z respectively.

Table 2.2: Combinational Observability Equations using SCOAP

Gate	**A-Observability Equation CO(A)**	**B-Observability Equation CO(B)**
AND	$CO(Z) + CC1(B) + 1$	$CO(Z) + CC1(A) + 1$
OR	$CO(Z) + CC0(B) + 1$	$CO(Z) + CC0(A) + 1$
NAND	$CO(Z) + CC1(B) + 1$	$CO(A) + CC1(A)) + 1$
NOR	$CO(Z) + CC0(B) + 1$	$CO(Z) + CC0(A) + 1$
NOT	$CO(Z) + 1$	
BUF	$CO(Z) + 1$	
XOR	$CO(Z) + \min(CC0(B), CC1(B)) + 1$	$CO(Z) + \min(CC0(A) + CC1(A)) + 1$
XNOR	$CO(Z) + \min(CC0(B), CC1(B)) + 1$	$CO(Z) + \min(CC0(A) + CC1(A)) + 1$

Sequential **SCOAP** measures are computed in a similar way. However the depth in the combinational portion of the circuit is not considered. The controllability and observability values are incremented by 1 only when a signal crosses a timing boundary i.e. at the flip-flop interface. Also since flip-flop introduce feedback loops in the system, the computation has to be iteratively done so as to reach a stabilized value. Sequential controllability roughly measures how many times the flip-flops need to be clocked to set the net to a required value. Sequential observability measures the number of times the flip-flops should be clocked to observe a particular value.

In its simplest form, let us consider a flip-flop with a data input D, a clock inputCLK and an output Q. Since the clock input is fully controllable, it is assigned sequential 0-controllability and sequential 1-controllability value of 0 each. This is unlike the combinational controllability computation where the initial values are set to 1. The sequential 0-controllability and sequential 1-controllability of the signal Q is given by the following equations:

- $SC0(Q) = SC0(D) + SC1(CLK) + SC0(CLK) + 1$
- $SC1(Q) = SC1(D) + SC1(CLK) + SC0(CLK) + 1$
- $SO(D) = SO(Q) + SC1(CLK) + SC0(CLK) + 1$

For the clock signal CLK both the 0 and 1 controllabilities are considered because it needs to be ticked to pass the value from the input to the output. The process of iterative computation converges because controllability/observability values are monotonically non increasing between each successive iterations.

2.5 Challenges in Trojan Detection - Why is it a difficult problem?

Trojans are hard-to-detect using conventional testing mechanisms. In hardware domain, cryptographic algorithms based on *public and private key* concept and approaches based on LFSR and Logic BIST [20, 26, 37] have been proposed to monitor the proper operation of the internal hardware. However, Trojans can be intelligently built to deter the advantages of such vigilant approaches. In addition, Trojans can be selectively implanted and its absence in one IC does not guarantee its absence in another. So destructive testing is also not a viable option. On one hand, destructive testing incurs a yield loss where a chip that has been cut open for analysis has to be discarded, while on the other it cannot guarantee that parts not subjected to such testing are genuine. Trojans can have varying spatial locations on the IC and different logical behaviors (counter-based Trojans, sequence-detector Trojans etc.) [80] which complicates the detection mechanism. In software, Trojans (a class of viruses) have been prevalent and many software solutions (anti-virus) exist for their detection. In [82] the authors propose a method for identifying the Trojan software running on a microprocessor. This method uses a digital signature to validate the authenticity of the software before running it on the machine. But there is a difference between a software virus and the

hardware Trojan. *Viruses* are usually malicious and interfere with the normal operation of the host on which they reside, whereas *Trojans* are passive monitors for most part of their operational life cycle until they are triggered.

An intelligent adversary can insert the Trojans such that their detection probability using test patterns (functional or random) turns out to be extremely low. Assume a Trojan with n inputs. Also assume p_i is the probability of justifying a 0 or 1 on the i^{th} input of the Trojan circuit. In case of deeply embedded Trojans p_i will be extremely small. The probability of activating this n input Trojan and propagating its effect to an observable point is given by [80]:

$$\mathbf{P} = \mathbf{P}_{activation}.\mathbf{P}_{propagation} \tag{2.5}$$

Also the probability of activation is given by:

$$\mathbf{P}_{activation} = \mathbf{\Pi}_{i=1}^{n} p_i \tag{2.6}$$

Assuming p_i=10^{-3} and n=10, P turns out to be 10^{-30}. In addition to this, $P_{propagation}$ can further worsen the value of P. This clearly indicates that test patterns can assure a very low reliability in detecting embedded Trojans.

2.6 Constructing a Trojan

One of the most challenging task in devising mechanisms of Trojan detection is to come up with a minute circuit that satisfies all the properties of a Trojan. Our experimentation have shown that selecting random nodes for creating Trojan inputs is not effective as much as a random payload point is not good either. Moreover the Trojans should be stealthy enough to escape functional simulation, scan testing etc.

2.6.1 Framing Combinational Trojans

Intuitively combinational Trojans seem to be easy to construct than sequential Trojans, but actually it is just the reverse. Combinational Trojans are small (as small as a single gate) but they are highly testable with randomly filled scan patterns if not constructed properly. A proper construction means selecting a very rare condition for excitation and then deploying the payload to a gate such that their effect is non-observable using normal scan patterns.

We follow a specific procedure to construct nice combinational Trojans for our experimentation:

- In the first stage, we simulate the original circuit with a large set of functional patters and filter out those nodes which are rarely toggling. Within these nodes we construct an appropriate pattern that is rarely occurring to act a trigger for the Trojan.
- In the second stage we build up the Trojan gate with the selected input and try to see the effect of payload on the appropriate gates. For example if the Trojan is an AND gate the minority value at the output is 1. So we feed this payload to an OR ar a NOR gate for which 1 is a controlling value. Similarly for an OR Trojan the minority value is 0 and the output is fed to a an AND or a NAND gate.
- We try to scan test the Trojan circuit with a set of ATPG generated scan vectors without fault dropping and filled with random bits for the don't cares. The gates for which the Trojan effect is exposed at the outputs or the flip-flops of the next cycles are discarded as weak payloads which can be detected by scan testing.
- The final payload is deployed to one/more of those gates which are not easily exposed at the output. Here this is to not that the scan patterns contain the illegal states when filled in randomly. So the chances that the Trojans are triggered by the scan patters are much more than those with functional vectors. Our experimental results show that those Trojans which are stealthy against the scan patterns are stealthy for the functional patters as well.

2.6.2 Framing Sequential Trojans

Sequential Trojans are realized as state machines. To construct a sequential Trojan we random simulate the circuit with a large set of vectors and collect the state sequence. From the state sequence we select a rarely occurring unique pattern. It is not necessary that the unique patters needs to be selected from the state bits only. Internal signals from other gates can be used as well. But just a proof-of-concept we chose to use the state bits. Once the sequence is selected, we construct a finite state machine to create the Trojan circuit.

There are a few things to note here. Sequential Trojans require extra flip-flops to implement their logic. These flip-flops cannot be a part of the normal scan chain because otherwise a simple scan chain testing mechanism will reveal their presence because the scan length will be different. More so the reset state of such flip-flops will be set to the one which is farthest from the triggering state. This is because the adversary will try to make the Trojan as stealthy against conventional tests as possible. Letting the Trojan come up to a random state actually increases the potential risk of making them easy to detect as shown by the following argument.

Let us consider a Trojan with 2 flip-flops going through a sequence of 00,01,10 and 11. Let 11 be the triggering state and the sequence of transition is in the order specified. If the flip-flops are allowed to come up randomly during reset, there is a chance that the bits can be set to 11 which is a triggering state and the effect will be exposed. To avoid such incidents, the adversary will set the Trojan flip-flops to 00 (the starting state in the state transition diagram), so that scan test is ineffective is exposing the Trojan as well as scan testing followed by a partial functional patterns simulation is also ineffective.

Chapter 3

Improving At-Speed Testing

While scan-based testing achieves a high fault coverage, it requires long test application times and substantial tester memory, in addition to the overhead in chip area and high test power. Functional testing, on the other hand, suffers from low coverage but can be applied at-speed. In this chapter, we propose a novel three-step design-for-test (DFT) methodology which enhances the performance of functional testing to a great extent. In the first step we expand the state space of the circuit beyond functionally reachable space without scan or reset. These new states create conditions to activate/propagate fault effects that are otherwise hard-to-detect. Since structural correlation between D flip-flops (DFFs) of a circuit restricts its state space variation, the second step consists of partitioning the DFFs into different groups that helps to break such correlations. In the third step, we make internal hard-to-observe points in the circuit more observable by directly XORing them with selected primary outputs. This method can be applied at-speed (since no scan shifting is involved) saving significant amount of test application time, with comparable area overhead as scan-based DFT. Later we also show that this technique can be employed for an improved Trojan detection scheme. Our experiments with minuscule Trojans yield promising results with this approach.

3.1 Motivation

There are two major DFT approaches in practice. *Scan based techniques* achieve a very high *fault coverage* at the cost of large test application time, test data volume, and additional hardware overhead. In such a technique, all the DFFs are connected back-to-back in a chain during test mode. Every scan pattern needs to be shifted into the scan chains before they can be applied to the circuit. This shifting is done on the *Test Clock* (T_{CK}) which is often slower than *Functional Clock* (F_{CK}) and hence translates to increased test application time. Variations in scan architecture have been proposed to improve test application time of traditional scan architecture. For example, in [40], [36] and [49], the authors have proposed *Illinois Scan Architecture* (ILS) to speed up scan testing. In ILS, the single scan chain is partitioned into multiple smaller chains. Test data can be broadcast and shifted in parallel to these scan chains thereby reducing the total test application time. Since parallel shifting of test data can make some of the otherwise detectable faults undetectable, variations in ILS exist to support single scan chain mode as well. In [23] the authors proposed *California Scan Architecture* (CSA). CSA employs a traditional scan-architecture to shift in test patterns with adjacent filling such that circuit power consumption remains low during scan shift. In test application, it employs selected $\overline{Q}$ signals instead of all Qs from the DFFs. Experimental results show that their strategy helps in improving defect coverage.

On the other hand, *functional testing* can be applied at-speed because no scan shifting is involved which is more economical from test application point of view. But often, values required on DFFs for exciting/propagating many fault effect(s) are very difficult to achieve in the functional mode. In [64], [38], [51] and [28], the authors have proposed novel sequential ATPG(Automatic Test Pattern Generation) algorithms targeted to improve the fault coverage of sequential circuits. Their results show that achieving a high level of fault coverage for sequential circuits is still a major problem, especially for large circuits with huge state spaces. In [22] and [39], the authors have proposed non-scan DFT techniques for improving the *fault coverage* of non-scan sequential circuits. Our current work falls under this theme,

although it involves less overhead in terms of additional hardware used with respect to these works.

In this work, we propose a novel three-step non-scan DFT approach to enhance the *fault coverage* of a DUT. We use both Q and $\overline{Q}$ signals of a subset of DFFs to help activating/propagating many more fault effects to primary output(s). We propose a heuristic based approach to partition the DFFs into multiple groups and select a specific group using test inputs which we call as *ENABLE* pins. Incorporating the $\overline{Q}$ values of only a subset of DFFs (instead of all DFFs) sufficiently help to diversify the state space. Normal scan based testing uses *scan enable*(SE), *scan in*(SI), *scan out*(SO) and *scan mode*(SM)(Single chain or ILS mode) input pins to test the DUT. We also use a maximum of 4 input pins for our proposed DFT architecture so that there is no additional overhead in terms of pin count. Finally, by selecting a set of hard-to-observe points and making them observable at outputs, we further enhance the *fault coverage*. No additional output pins are needed as we use existing outputs for observing fault effects. Since no scan shifting is involved, this methodology can be applied at-speed that helps saving a significant amount of test application time, as well as allowing for catching delay-related defects.

3.2 Approach

Our approach consists of three steps. In the first step, we modify the DFFs to make $\overline{Q}$ usable for creating new state vectors in the circuit. In the next step, we partition the DFFs into multiple groups to selectively use their $\overline{Q}$ signal, thereby breaking the DFF correlations and enriching the state space. In the final step, we increase the observability of a set of selected hard-to-observe internal nodes by making them directly observable at existing outputs of the circuit.

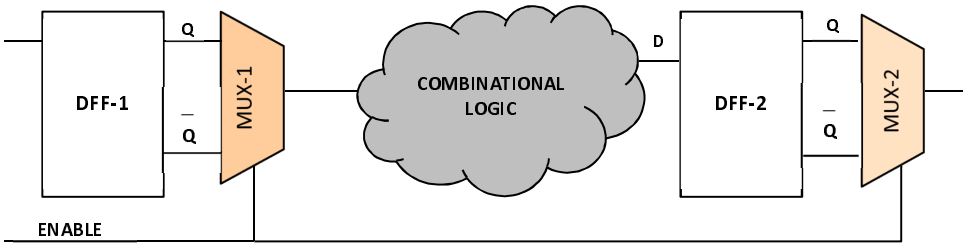


Figure 3.1: Modifying flip-flops to enable D-BAR

3.2.1 Enabling $\overline{Q}$

In this work, we expand the state space of the DUT by utilizing $\overline{Q}$ values of DFFs along with Q. We modify the output of a conventional DFF by multiplexing the Q and $\overline{Q}$ signals through a 2×1 multiplexer and selecting either of them based on *ENABLE* signal of the multiplexer (refer to Fig. 3.1). The *ENABLE* signal is determined by input pin(s) of the DUT and is used to select the normal or complemented value from the DFFs. This modification does not add any hardware overhead as compared to normal scan DFF. Essentially, we are taking the multiplexer which was placed in front of the DFF in normal scan architecture to the output end of the DFF. By allowing for either Q or $\overline{Q}$ values, we effectively increase the controllability of many hard-to-control signals, thereby significantly enhancing the testability of the logic. In other words, the inclusion of $\overline{Q}$ along with Q helps creating new conditions in the combinational logic for excitation/propagation of fault effects which were otherwise difficult to detect.

3.2.2 Flip-flop Partitioning

Controlling all the DFFs using a single *ENABLE* restricts the state space variation to an extent. In a circuit, many DFFs are interdependent, i.e., their values are correlated. Such correlations must be broken to create more variations in state space. For example, if a

pair of DFFs A and B are equivalent, $Q_A \leftrightarrow Q_B$; then $(1,1)$ and $(0,0)$ are the only two possible combinations possible using a single *ENABLE* pin. $(1,0)$ and $(0,1)$ would not be achievable under this single *ENABLE* configuration. To resolve this problem, we partition the DFF set into different groups so that a minimum number of correlated DFFs are placed in a group. We use multiple input pins and use a decoder to create several *ENABLEs* each of which is used to select a specific group. Selecting $\overline{Q}$ from a part of the DFFs instead of the entire DFF set helps in creating variation in states obtained on DFF outputs which are not achievable using a single *ENABLE*. Conceptually the achievable state space under all such configuration(s) (single/multiple *ENABLES*) lies in between the total state space and functional state space. This is shown in Fig. 3.2

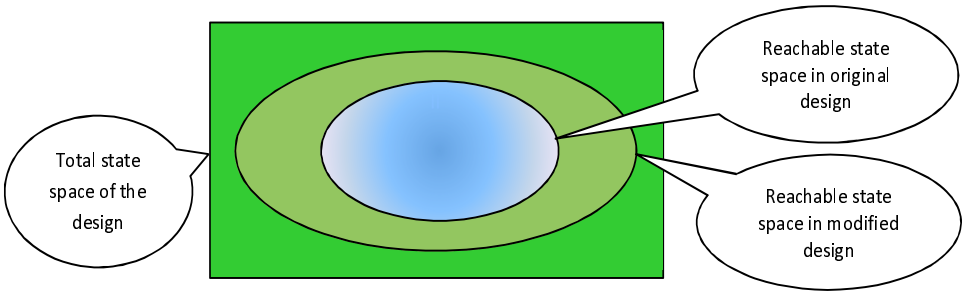


Figure 3.2: Extent of reachable state space using our approach

Fig. 3.3 shows the fanin cone of three DFFs viz. 4, 9 and 13. It is evident that the overlap in fanin cones of DFFs 4 and 9 is more than that between DFFs 9 and 13. This implies DFFs 4 and 9 have more common inputs than DFFs 9 and 13. When two gates are controlled by a common set of inputs, chances are high that their logic values are correlated. To break this structural correlation we place DFF 4 in one group and 9 in the other. Since DFF 13 has no fanin cone overlap with DFF 4, it is placed in the same group containing DFF 4. DFF partitioning has been used in the past to reduce test data volume in scan based testing [11].

The DFF grouping procedure is explained using ISCAS'89 benchmark circuit s298. This circuit has 14 DFFs viz. 1 through 14 and their fanin cone matrix is shown in Fig 3.4 (i). An entry of 1 at the intersection of DFF A (represented as row index) and DFF B (represented as column index) indicates that DFF B is in the fanin cone (FC) of DFF A. The fanin cone

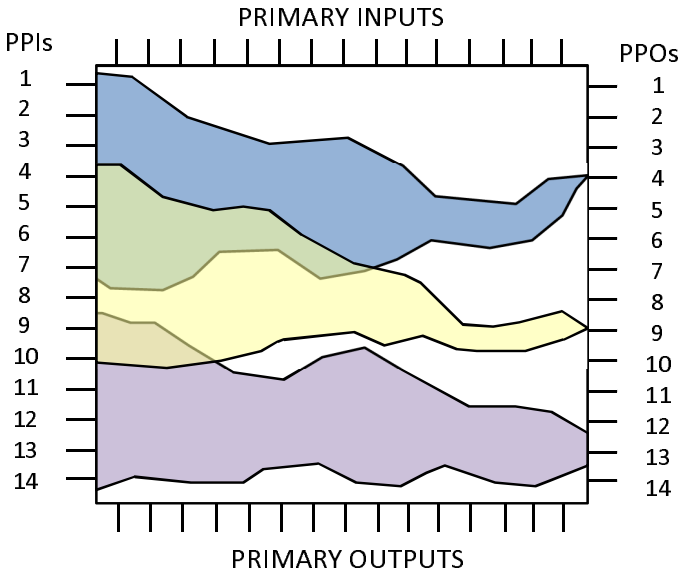


Figure 3.3: Partitioning scheme for flip-flops into different groups

of a DFF X is defined by the following set:

$$FC_X = \{g \mid g\, is\, a\, gate\, in\, the\, fanin\, cone\, of\, X\} \tag{3.1}$$

Thus the fanin cone of DFF 9 consists of DFFs $FC_9 = \{2, 3, 4, 5, 6, 9, 13\}$ and is represented by the following vector $\langle 01111100100010 \rangle$ where the index 9 refers to the DFF number.

For a group of DFFs G, the fanin cone (FC) of G is defined as the union of the fanin cones of individual DFFs that comprise G.

$$FC_G = \bigcup_{n=1}^{n=i} FC_n \tag{3.2}$$

We define the fanin count C of a DFF or a group of DFFs as the cardinality of its fanin cone.

$$C_X = |FC_X|\ or\ C_G = |FC_G| \tag{3.3}$$

For the DFFs in s298, their cardinalities are $C_1 = 1, C_2 = 4, C_3 = 3, C_4 = 4, C_5 = 6, C_6 =$

$6, C_7 = 7, C_8 = 7, C_9 = 7, C_{10} = 8, C_{11} = 8, C_{12} = 7, C_{13} = 1, C_{14} = 1.$

Overlap between two DFF groups G_x and G_y is defined as the summation of the dot product of their fanin cone vectors

$$O_{G_x}^{G_y} = \Sigma FC_{G_x} \cdot FC_{G_y} \tag{3.4}$$

This is to note here that a group can consist of a single DFF as well.

Given a target number of partitions to be formed within a set of DFFs, we begin by placing the DFF with maximum cardinality in GROUP 1 (G_1). For example, in s298 both DFFs 10 and 11 have cardinality of 8. For creating 3 different groups we select the first one, i.e., DFF 10 and place it in G_1. We define *selected group* (SG) as the group of DFFs that have already been selected in some group. Currently SG for s298 consists of DFF 10 only. Thus the fanin-cone $FC_{SG} = \langle 11111100010010 \rangle$. For selecting the first DFF of all subsequent groups G_2 and G_3 in our running example), we compute the overlap of all remaining DFFs with SG individually. For s298, the fanin cone overlap for DFF 1 with current SG is given by $O_{SG}^1 = \Sigma\langle 11111100010010 \rangle \cdot \langle 10000000000000 \rangle = 1$. Similarly the overlap with all other DFFs with SG is $O_{SG}^2 = 4, O_{SG}^3 = 3, O_{SG}^4 = 4, O_{SG}^5 = 5, O_{SG}^6 = 6, O_{SG}^7 = 6, O_{SG}^8 = 6, O_{SG}^9 = 6, O_{SG}^{11} = 7, O_{SG}^{12} = 6, O_{SG}^{13} = 1, O_{SG}^{14} = 0$. This is to note that there is no entry for DFF 10 as it has already been selected in G_1. The highest overlap of SG is with DFF 11 and so DFF 11 is placed in G_2. SG now consists of DFF 10 and 11. Thus, $FC_{SG} = \langle 11111100011010 \rangle$. The next DFF is selected based on the individual overlap of the remaining DFFs with current SG. This overlap comes out to be $O_{SG}^1 = 1, O_{SG}^2 = 4, O_{SG}^3 = 3, O_{SG}^4 = 4, O_{SG}^5 = 5, O_{SG}^6 = 6, O_{SG}^7 = 6, O_{SG}^8 = 6, O_{SG}^9 = 6, O_{SG}^{12} = 6, O_{SG}^{13} = 1, O_{SG}^{14} = 0$. Hence the first DFF of G_3 is DFF 6. (Although DFF 7, 8, 9 and 12 have the same overlap, we select the first one).

Once each group has been assigned a DFF, the overlap criteria for selecting subsequent DFFs in any group is reversed from maximum overlap to minimum overlap. For selecting the next DFF in G_1, we compute the overlap of the remaining DFFs (those which are yet to be selected) with the existing DFF(s) in G_1, selecting one with minimum overlap. For s298, the overlap of remaining DFFs with G_1 is given by $O_{G_1}^1 = 1, O_{G_1}^2 = 4, O_{G_1}^3 = 3, O_{G_1}^4 =$

$4, O^5_{G_1} = 5, O^7_{G_1} = 6, O^8_{G_1} = 6, O^9_{G_1} = 6, O^{12}_{G_1} = 6, O^{13}_{G_1} = 1, O^{14}_{G_1} = 0$. Since DFF 14 has the minimum overlap, it is placed in G_1 along with DFF 10. All the rest of the DFFs are selected in the same manner. Selecting the DFFs with minimum overlap helps in breaking structural correlations among DFFs in a group. In other words, DFFs which are structurally correlated get segregated into different groups. Since each group has independent *ENABLE* signals, such a grouping scheme ensures that their $\overline{Q}$ output are independently selected irrespective of the DFF to which it is correlated. Fig 3.4 (iii) shows the composition of the groups for a partition count of 7.

FF IDs	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	0	0	0	0	0	0	0	0	1
6	0	1	1	1	1	1	0	0	0	0	0	0	1	0
7	0	1	1	1	1	1	1	0	0	0	0	0	1	0
8	0	1	1	1	1	1	0	1	0	0	0	0	1	0
9	0	1	1	1	1	1	0	0	1	0	0	0	1	0
10	1	1	1	1	1	1	0	0	0	1	0	0	1	0
11	1	1	1	1	1	1	0	0	0	0	1	0	1	0
12	0	1	1	1	1	1	0	0	0	0	0	1	1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(i) Fanin-cone matrix for flip-flops in s298

G1	10	14	3	5	9
G2	11	1	2	7	12
G3	6	13	4	8	

(ii) Flip-flop groups with enable count 2

G1	10	14
G2	11	1
G3	6	13
G4	7	3
G5	8	2
G6	9	4
G7	12	5

(iii) Flip-flop groups with enable count 3

Figure 3.4: (i) Flip-flop fanin cone matrix for s298 (ii) Flip-flop grouping for 2 enable pins for s298 (iii) Flip-flop grouping for 3 enable pins for s298

3.2.3 Observability Enhancement

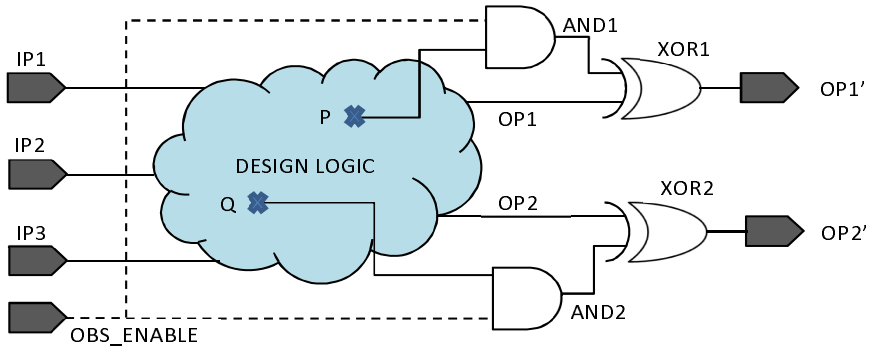


Figure 3.5: Observability modification for hard-to-observe nodes

In this step, we attempt to increase the observability of a selected set of hard-to-observe internal nodes. The concept of *Observability Enhancement* is shown in Fig. 3.5. P and Q are two hard-to-observe points in the given circuit. For each signal, we use a AND gate to select the point in the test mode and XOR them with an existing primary output to make them observable. The first input of the AND gate is the hard-to-observe signal and the second input is OBS_ENABLE signal. In the normal functional mode, OBS_ENABLE is set to 0 and hence only functional logic value reaches the final output $OP1'$ and $OP2'$. In the test mode, OBS_ENABLE is set to 1. If any fault effect appears on either P or Q, it will immediately propagate to $OP1'$ or $OP2'$ because the XOR gate will not block the fault effect (unless the other input of the XOR gate also contains the fault effect from the same fault).

Selecting suitable hard-to-observe points in a circuit can be tricky. It may be the case where there are more hard-to-observe points than the number of outputs in a circuit. One approach could be building multiple XOR trees to include all such points and associating each XOR tree with one of the outputs. But XOR trees involve much hardware overhead and is not preferred. Alternatively, if we associate each output with single internal hard-to-observe point, the number of points we can observe is exactly the number of outputs (which can be small)!

To tackle this problem we select our hard-to-observe points based on hard-to-observe *regions*. For each gate in the circuit we compute its fanin cone till a depth of 3. We run a random + sustained random vector set on the circuit with modified and partitioned DFFs and obtain a list of excited but undetected faults. We grade the gates present in this list by the number of fault instances that are undetected in its *region*. For example, if a 2 input gate T has 2 faults on its input and 1 fault on its output as excited but undetected, then the *fault weight* (FW) assigned to this gate is 3. For a *region* we define its *fault weight* as the summation of fault weights of all gates included in that *region*.

$$FW_R = \sum_{i=1}^{i=N} FW_{T_i} | T_i \epsilon R \tag{3.5}$$

where FW_R represents the *fault weight* of *region* R, FW_{T_i} represents the *fault weight* of i^{th} gate in *region* R and R contains a total of N gates.

As an example, let's consider the *region* shown by the dotted line in Fig. 3.6. This *region* stems out from the fanin cone of gate 9 till a depth of 3 levels and contains gates $2, 3, 4, 5, 6, 7, 8, 9$ and $DFF - 2$. Gates which have undetected faults on them are tabulated in top right corner of Fig. 3.6. Gates $2, 3, 4$ and 5 have undetected faults on them and *fault weight* corresponding to each one of them is shown under the $FaultWeight$ column in the figure. So, *fault weight* of the *region* is computed as $FW_{R_9} = FW_2 + FW_3 + FW_4 + FW_5 = 1 + 2 + 3 + 3 = 9$.

We arrange all the *regions* in the circuit in descending order of their *fault weights* and select regions from the top of the list. In our implementation, the number of *regions* selected is equal to the number of outputs. In essence, we are trying to maximize the number of hard-to-observe points that can be made observable without incurring much hardware overhead compared to an XOR tree.

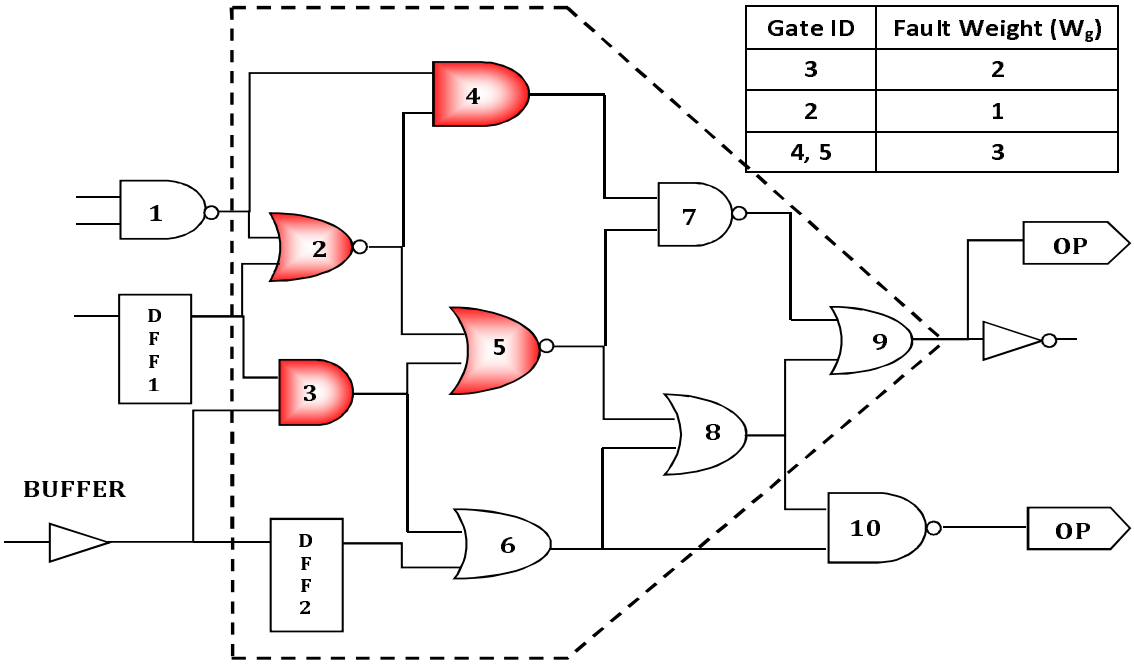


Figure 3.6: Computing *fault weight* of a *region*

3.3 Application in Trojan Detection

So far we have discussed how this approach can be used for improving the testability of hard-to-test circuits. Alternatively, we can use the same approach towards Trojan detection as well. Since Trojans are hard to activate portions of the circuit, the expanded state space can be used to create more variations in the internal circuit nodes thereby forcing the gates to attain combination of values which were otherwise very hard to achieve. At the same time, the propagation criteria we discussed earlier still holds. As a result of this, it is possible to activate and observe many otherwise hard-to-detect Trojans.

The state-space partitioning scheme acts like a obfuscation technique as well. The $ENABLE$ signals that we use to selectively invert portions of the state space can be used as a *security key*. Considering we have 3 $ENABLE$ pins we can operate the circuit in 8 different modes. In our setup, we have used the $ENABLE\_1 = 0, ENABLE\_2 = 0 \, and \, ENABLE\_3 =$

0 as the condition of normal operation of the device. This need not necessarily be like that. So we can have a circuit configuration in which $ENABLE\_1 = 0, ENABLE\_2 = 1 \, and \, ENABLE\_3 = 1$ is the condition for correct operation. Since the DFFs belonging to the groups which are assigned a value of 1 uses $\overline{Q}$ signals at the output, we need to make sure that the input is inverted when fed to the DFF; i.e. if the input of a DFF is originally and AND gate, it needs to be changed to a NAND gate when this DFF is supposed to work with $\overline{Q}$ in functional mode. It is very difficult to guess at about the correct operating mode of the circuit without the prior knowledge about its functionality. Consequently, it is difficult to come up with a Trojan specifically intended to affect for the functional mode.

3.4 Experimental Setup

We used ISCAS'89 and ITC'99 sequential circuits to evaluate the effectiveness of our approach. We report the results for most of these circuits with varying number of DFFs, viz. from 5 DFFs in s820/s832 t0 1426 DFFs in s38584.1 i.e., a potential state space ranging from 32 states to $> 2^{1426}$ states. For each circuit, we modified the existing DFFs by adding a multiplexer, thus allowing Q and $\overline{Q}$ signals to propagate to the combinational logic. We experimented using 1, 2 and 3 input *ENABLE* pins for creating the *ENABLE* signals to select DFF groups. A decoder structure is used to create the actual *ENABLEs* for each group. Thus, for k *ENABLE* input pins, we can uniquely select 2^k groups. Out of these 2^k possible selects, we reserve one for the functional mode in which none of the groups are selected to use $\overline{Q}$ signal. The remaining are assigned to address different groups. Thus the number of groups addressable for k *ENABLE* input pins is $2^k - 1$.

We create a vector set consisting of random and sustained random vectors. In sustained random vectors we do not change the input vector for a specific number of time frames (we sustain it)! and allow current state in the circuit to create the next state. This concept was introduced in [63] in which the authors sustained pseudo random sequence generated by a

LFSR to make the DUT transition through different states. Later in [34], the authors showed it to be very effective in enhancing *fault coverage* of some hard-to-test circuits. For exercising each group (including the group containing entire DFF set in normal functional mode), we generate 25,000 random vectors and 1000 sustained random vectors, each sustained 25 times making a vector set size of 50,000 vectors. Thus, total vector set for a circuit with single *ENABLE* pin consists of 100,000 (once with *ENABLE* high and once with *ENABLE* low) vectors. To make the comparison of *fault coverage* fair, for the original circuit (which doesn't have an *ENABLE* input) we repeat the same random + sustained random vector set twice (also making it 100,000 vectors). The runtime of our algorithm is in the order of minutes for large circuits but considering this is a one time procedure it is not a major concern.

Our experiments are conducted in two stages. In stage 1 we modify the DFFs, create the groups, add *ENABLE* pins at the input and create the decoder to utilize the *ENABLE* pins for selecting different DFF groups. We fault simulate the generated vector set to create a list of faults that are excited but undetected. In the second stage, we use the list of excited but undetected faults remaining from stage 1 to select the observation points using step-3 of 3.2. After we modify the hard-to-observe points to make them observable, we again fault simulate the new circuit. Our experimental results report the final coverage achieved after all the modifications have been done on the circuit. We also run the circuit with 3 *ENABLE* input pins through a state-of-art sequential ATPG tool to improve the *fault coverage* even more. The entire algorithm is written in C++.

3.5 Experimental Results

Table 3.1 shows the *fault coverage*, *fault efficiency* and reduction in test time for large ISCAS'89 and ITC'99 benchmarks using our proposed methodology. "CKT STATS" contains the circuit attributes viz. the circuit name represented under "CKT NAME" column, number of inputs and outputs in the original circuit under the "INPUT/OUTPUT" column and

the number of DFFs under the "DFF COUNT" column. "FAULT COVERAGE" reports the *fault coverage* achieved by using different number of *ENABLE* pins as proposed in the theory. Under this section, "NO-ENABLE", "1-ENABLE", "2-ENABLE" and "3-ENABLE" represent the *fault coverage* achieved in % by using 0, 1, 2 and 3 *ENABLE* pins respectively. We run the sequential ATPG only in the "3-ENABLE" configuration. For this, we report both *fault coverage* and *fault efficiency* which are the sub-columns under "SEQ ATPG W/ 3-ENABLE". The column "TEST TIME RED." represents the reduction in test application time for our methodology relative to a scan configuration (single scan chain or ILS structure depending on DFF count of the circuit). For all the configuration with one or more *ENABLE* pins, we have one OBS_ENABLE pin to control the observability of hard-to-observe points. In comparison to the original circuit, additional gates are required to enable observability of internal nodes in the modified circuits. For three different configurations of modified circuits, the number of gates in the decoder logic is different. Hence total fault count in the fault list of a given circuit is different for all the four different configurations (*NO-ENABLE*, *1-ENABLE*, *2-ENABLE* and *3-ENABLE*). To make fair comparisons we report the total *fault coverage* including all those new faults which are introduced in the circuit because of the addition of extra hardware. For circuits like s820, s832, s1488 and s1494 where the DFF count is less than 8, it is not feasible to form 8 groups out of it. So we don't test them with *3-ENABLE* configuration and the corresponding coverages are denoted by X. For computing the reduction in test application time, we use our state-of-art sequential ATPG tool with the *2-ENABLE* configuration for these circuits.

In Table 3.1, results on "FAULT COVERAGE" show the general trend that *fault coverage* increases as we increase the number of *ENABLE* pins. This is expected because as we increase the granularity of the DFF sets it creates new states which help exciting/propagating more fault effects to the output. In general, sequential ATPG is able to achieve a higher *fault coverage* for all the circuits with 3 *ENABLE* signals.

For circuits like $s1423$, $s3330$, $s5378$, $s38584.1$, $b12$, $b13$ and $b14$ which were otherwise hard to test using traditional sequential ATPG, we have been able to improve the *fault coverage*

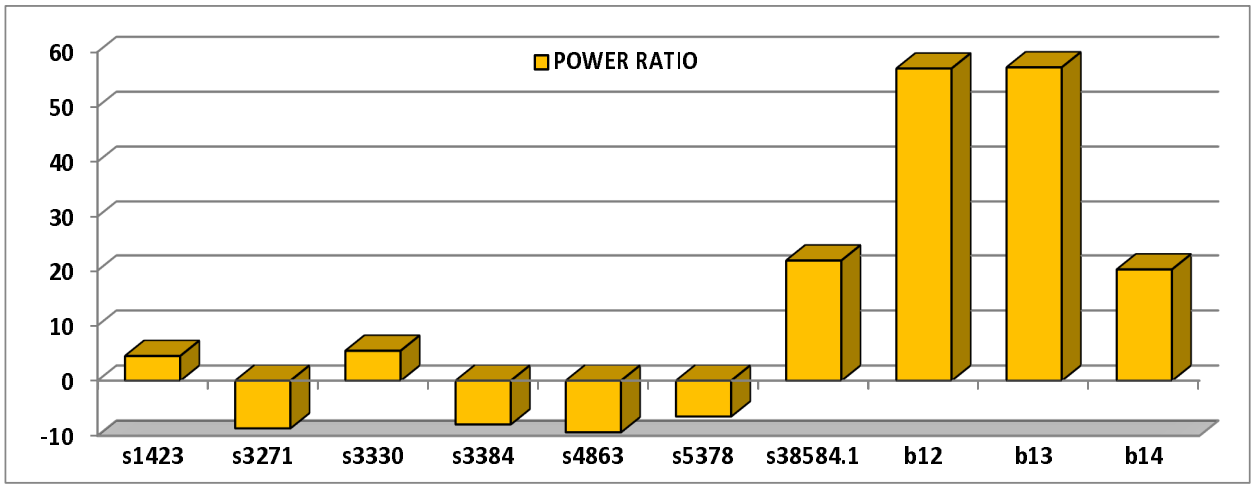
Table 3.1: Fault Coverage and Test Time Reduction for ISCAS'89 and ITC'99 Benchmarks using our methodology

CIRCUIT STATS			FAULT COVERAGE						TEST
CKT NAME	INPUT/ OUTPUT	DFF COUNT	NO-EN FC(%)	1-EN FC(%)	2-EN FC(%)	3-EN FC(%)	SEQ ATPG W/ 3-EN FC(%)	SEQ ATPG W/ 3-EN FE(%)	TIME RED.
s298	3/6	14	86.04	91.74	95.73	96.22	**97.35**	97.35	**88.48**
s344	9/11	15	96.20	95.85	97.20	97.38	**99.35**	99.35	**107.33**
s349	9/11	15	95.71	95.55	96.90	97.09	**99.03**	99.35	**113.38**
s382	3/6	21	91.23	91.14	95.91	**97.03**	94.66	94.66	**106.25**
s400	3/6	21	89.49	90.00	94.88	**96.17**	95.33	96.32	**145.03**
s444	3/6	21	88.82	88.15	95.66	95.75	**96.02**	97.08	**73.33**
s526	3/6	21	79.64	90.84	95.35	95.56	**97.12**	97.24	**73.31**
s641	35/24	19	86.51	94.44	90.81	94.24	**99.54**	99.54	**105.10**
s713	35/23	19	81.93	90.19	86.71	90.55	**95.48**	99.59	**87.81**
s820	17/19	5	50.12	70.90	94.64	X	**99.44**	99.44	**34.31**
s832	17/19	5	48.97	70.22	94.56	X	**99.26**	99.35	**35.90**
s1196	14/14	18	98.47	98.86	98.94	98.97	**99.81**	99.81	**135.62**
s1238	14/14	18	93.28	95.84	96.19	96.04	**96.94**	97.78	**134.36**
s1423	17/5	74	80.00	86.42	96.04	96.11	**96.42**	97.06	**167.48**
s1488	7/19	6	95.56	95.69	97.01	X	**99.53**	99.53	**33.44**
s1494	7/19	6	94.82	95.09	96.53	X	**99.19**	99.19	**33.00**
s3271	26/14	116	99.20	98.49	99.40	99.43	**99.98**	99.98	**58.62**
s3330	40/73	132	73.45	89.32	92.47	94.94	**96.65**	97.34	**90.05**
s3384	26/14	183	91.15	96.64	96.79	96.99	**97.94**	97.94	**165.83**
s4863	49/16	104	95.89	96.51	98.14	98.81	**99.83**	99.93	**66.37**
s5378	35/49	179	69.56	83.62	81.97	86.65	**96.96**	97.81	**58.51**
s15850.1	77/150	534	40.80	65.07	69.08	72.48	**90.08**	93.15	**88.77**
s38584.1	38/304	1426	66.79	88.57	91.50	91.42	**96.25**	98.90	**301.70**
b10	12/6	17	85.94	95.86	97.21	98.15	**98.15**	98.15	**55.81**
b11	8/6	30	84.11	85.01	97.36	97.35	**97.49**	97.98	**137.53**
b12	6/6	121	30.17	39.19	57.81	66.04	**98.18**	98.18	**12.67**
b13	11/10	53	60.41	78.65	91.00	91.81	**96.85**	97.58	**194.94**
b14	33/54	247	64.29	85.71	87.60	88.87	**95.15**	97.49	**19.24**
AVERAGE			79.23	87.27	92.26	93.09	**97.43**	**98.11**	**97.29**

by more than 10% using our modified circuit configuration; sometimes the improvement is as high as 68% for $b12$, nearly 30% for s38584.1 and 27% for s5378. We didn't report the results for $s35932$ because it is already highly testable without any modification. For such circuits, adding testability features would be an overhead. Circuits such as s15850.1 have less than 95% *fault efficiency* since they are not initializable; thus, both Q and $\overline{Q}$ remains to unknown value. On an average, the final test set obtained by sequential ATPG achieves a *fault coverage* of 97% and *fault efficiency* of 98%.

For computing the reduction in test application (shown in "TEST TIME RED." column) we made the following assumptions. Circuits with less than 100 DFFs are assumed to have a single scan chain and those with more than 100 DFFs have 16 balanced scan chains. Thus $s3271$, $s3330$, $s3384$, $s4863$, $s5378$, $s38584.1$, $b12$ and $b14$ have 16 chains for broadcasting the test patterns. If there are N patterns in the functional ATPG test set, we would need $N * F_{CK}$ time to run the complete test set, where F_{CK} is the period of the functional clock. For a circuit with single scan chain, assuming we have M test patterns and D DFFs in the circuit; total test time required is $((M+1) * D * T_{CK}) + M * F_{CK})$. For circuits with multiple scan chains, the time required will be $((M+1) * D/16 * T_{CK}) + M * F_{CK})$. The first part of the above expression(s) $((M+1) * D * T_{CK}))$ represent scan shift-in time (also shift-out which is multiplexed with shift-in) while the second part $(M * F_{CK}))$ represent test application time. For M test patterns we need to scan in all of them and scan out the last pattern after the test is applied. So the multiplying factor is $(M+1)$. Since T_{CK} is much slower than F_{CK}, the scan-shift in time dominates the total test time required. Test clock T_{CK} in our experiments is assumed to be 10 times [35] slower than the functional clock. The test sets for full scan testing is obtained using a combinational ATPG tool with fault dropping. Although for ILS mode we have assumed that the scan chains can be partitioned without dropping *fault coverage*, this may not be entirely possible. Thus the actual time required for scan testing in ILS mode will be little higher because all faults are not testable in ILS broadcast mode. For our functional test set generated by sequential ATPG, we have used compaction techniques to reduce the size of the test set before computing the test application time. In our results,

we can see orders of magnitude reduction in the test application time for circuits like $s1423$, $s3384$, $s38584.1$ and $b13$. When millions of ICs are subjected to test, such a reduction in testing time of a single part translates to a huge saving in overall test time.



Note: The disproportionate power in b12 and b13 is due to the fact that original circuits were very untestable with low signal toggles in the circuit

Figure 3.7: Ratio of average power in test mode using our approach to normal functional mode

Fig. 3.7 shows the increase/decrease in average test power using our approach. In [10], [71] the authors have shown that scan shift in power could be significantly high as compared to the functional power. For most circuits, our experiments show that the test power is comparable to the power in functional mode. In fact, the test power could sometimes be lower, as in s3271, s3384, s4863 and s5378. In two circuits, $b12$ and $b13$, the test power was unusually higher. This is because these two circuits are originally very hard to test (as shown by their low *fault coverage* without using our approach), as certain portions of logic in these circuits are very hard to toggle. With our modified circuit structure, those portions become sufficiently active, hence improving *fault coverage* and thereby adding to the average circuit power.

When compared to [22] and [39], they required several more input and output pins; on the other hand, our approach uses a fixed count (4) of input pins and no additional output pins. For example, in $s38584.1$, 11 additional inputs and 12 outputs have been used in [39] which

added to considerable hardware overhead. With less pins, we are still able to achieve a higher *fault coverage* and *fault efficiency* for the same circuit. In our case the test application time is improved by a factor of 300× as compared to 10× improvement in the previous works. For the remaining circuits, the *fault coverages* are comparable, but with a speed up in test application time in our approach.

For Trojan detection using the same approach, we employed only step-I and step-II. Our Trojan circuits consists of single gate Trojans (AND Trojan and OR Trojan). These Trojans are very hard-to-detect using functional mode and scan based tests. The results of our experiments are shown in Table 3.2. For the cases where we could not could not create a suitable Trojan, the corresponding Table entry has been marked with an *X*. In s5378 there are no AND/NAND gates which can act like a payload point for OR Trojans. In s35932, the high testability of the circuit makes all the payload points ineffective against AND Trojans with scan vectors. i.e. whatever payload point we select the effect of Trojan can be captured in some DFF using the scan vectors. So we couldn't frame a Trojan for these two cases. We chose to use the $3 - ENABLE$ configuration to partition the DFFs. We used a set of random and sustained random vectors and the various combination of $ENABLE$ signals. Specifically, we generated 10,000 random and 1000 sustained random vectors each of them sustained for 10 clocks making a total of 20,000 vectors for each mode. So for all 8 modes we had a total of 160,000 vectors to apply.

In Table 3.2 column one shows the circuit names, columns two and three shows the result for AND Trojans and columns four and five shows the results for OR Trojans under functional and test modes respectively. In functional mode the circuits is operated keeping the $ENABLE$ pins at functional mode values. In test mode the values are changed periodically to select the opposite values from the DFFs of the selected group. The vectorset used for both the functional and test modes are the same (except for the change in values of the $ENABLE$ pins in test mode). We find that all the Trojans are non-detectable at the output under the functional mode but are mostly detectable when the test mode is used. A few of them are not detected at all. This is expected because we partition our DFFs according to

Table 3.2: Detection Directly at the Primary Outputs for Scan-Resistant Trojans using Improved At-Speed Testing Technique

Trojans	**AND Trojan**		**OR Trojan**	
Op. Mode	Functional	Test	Functional	Test
Circuits				
s1196	UnObs	UnObs	UnObs	Obs
s1238	UnObs	Obs	UnObs	Obs
s1423	UnObs	Obs	UnObs	Obs
s3271	UnObs	UnObs	UnObs	Obs
s3330	UnObs	UnObs	UnObs	Obs
s5378	UnObs	Obs	X	X
s13207	UnObs	Obs	UnObs	Obs
s15850	UnObs	Obs	UnObs	Obs
s35932	X	X	Unobs	Obs

a particular heuristic and under such a paritition scheme it is not possible to generate the activation/propagation condition for these Trojans. A different partitioning scheme would yield a different result.

3.6 Summary

In this work, we proposed a three step technique to boost the performance of non-scan DFT substantially. In the first step we modified the existing usage of DFFs in a circuit by utilizing $\overline{Q}$ and thus extending the state space of the DUT beyond the functional limit. Partitioning the DFFs into different groups and using their $\overline{Q}$ selectively helps create more state variations helping in enhanced *fault coverage*. In stage three we select hard-to-observe points in the circuit and make them easily observable at the output so that any fault effect propagated to these points are not masked. Experimental results on ISCAS'89 and ITC'99 benchmarks shows marked improvement in the coverage of those circuits which were earlier hard to test. At-speed testing is possible using this methodology thus reducing the test application time by 2 orders of magnitude. We showed that the same technique can be used for creating an obfuscated design in which implanting a Trojan becomes non-trivial. Finally, the variation

in circuit conditions created by the variation in the state-space enhances the detection of Trojans in one/more of the test modes.

As a future work, variation in group selection heuristic would be interesting to investigate. Selection of hard-to-observe point using other heuristic is also possible. Moreover, constructing a XOR tree to include a larger set of hard-to-observe points should further enhance the *fault coverage*, but one has to strike a balance between the hardware overhead incurred in building such a tree and the improvement achieved.

Chapter 4

Voltage Inversion Technique

This chapter proposes an *inverted voltage* scheme for exciting and pronouncing the behavior of any undesirable logic that may be inserted in the IC manufactured abroad. The *inverted voltage* scheme is coupled with a *sustained vector simulation* technique to further enhance the behavioral difference between the genuine and targeted test IC. Our experimental Trojans consist of a single gate only. Experimental results show that we are able to significantly magnify the difference between the genuine and tampered designs. For most of the smaller benchmarks, our inverted voltage method is able to detect the effect of the tamper directly at the primary outputs of the IC, especially when existing techniques fail to make any observable distinction. For the larger circuits, our technique is able to augment the power consumption of the tampered circuit by several times.

4.1 Motivation

Most of the contemporary Trojan detection methods monitor one/more side-channel signal(s) for assessing behavior of the circuits under test (CUTs). Extraction of internal information like security keys using side-channel signal analysis [2], [45], [44] can lead to misuse of an IC

[77], [43]. For protecting the rights of IP designer, features are embedded in the IC to shield it from piracy or side-channel signal attacks. Security schemes based on *Physically Unclonable Functions* (PUF) [33], [27], *Digital Watermarking* etc. have been proposed towards this end. Nevertheless, if any third party intends to degrade the performance of underlying design tactfully, he/she can implant *Trojan(s)* even in the presence of these security features.

In the recent past, a number of methods have been proposed to detect the presence of *Trojan(s)* inside a design. In [12] the authors simulate a set of random vectors on the genuine and Trojan affected ICs and observe the power profile in both cases. Circuits affected with *Trojan(s)* consume more power but this difference is not truly discernible unless the circuit activity is kept really low. In [70] the authors have used a current integration technique to capture the prolonged effect of *Trojan(s)* on total charge consumption of the device. Every single toggle inside the Trojan causes a little variation in the instantaneous current which adds to the charge consumption. Over a period of time this cumulative charge consumption is shown to be sufficient to differentiate between the genuine and maligned ICs. The effectiveness of such an approach depends on the size of the Trojan as compared to the size of the host circuit on which it resides because the incremental current drawn by Trojan is proportional to its size. In [13], [14] and [15], the goal is to minimize the circuit activity so that the total circuit power does not drown out the Trojan power.

In [13] and [14], partition based schemes are employed to target and activate circuit portions individually. In [15], a sustained vector set is used to ensure that the circuit activity is generated by state transitions only. Results show that these methods are effective for circuits whose switching activity can be kept low. In [42] the authors have used a sampling based method to create a golden delay profile of the genuine ICs. Delay profile of any manufactured IC is plotted with the existing golden delay profile map. If the profile of CUT is outside a specified zone (called as the *convex-hull*), it is declared to be tampered. While delay profiling can be useful for capturing Trojan effects which affects the *critical paths*, it may not be equally effective for other *Trojan(s)*.

In this work we propose an inverted voltage scheme that aims to activate the *Trojan(s)* with a much higher triggering frequency. Once triggered/activated, the effect of a Trojan becomes more prominent. It either leads to an increased power consumption or causes an observable alteration in the output of the circuit. To strengthen its effectiveness, we integrate this approach with the sustained vector technique introduced in [15] to magnify the power profile difference. To make Trojans hard to detect, our experimental Trojan consists of a single gate only. For the smaller benchmarks we were able to detect the effect of Trojan directly on the primary outputs using our approach. For the rest of the benchmarks, the power profile behavior shows significantly increased variations between the genuine and the Trojan-affected designs, the differences were enhanced by many times, sometimes in orders of magnitude.

To study the effect of voltage inversion on normal CMOS circuits we conducted some experiments using HSPICE. Our experiments were conducted on 180nm TSMC library obtained from the VTVT lab in Virgina Tech. We performed two sets of experiments, first on a series of cascaded NOT gates and second on a series of cascaded NAND gates. For each set of gates we simulated the configuration using two approaches - one in which the voltage of all the gates are inverted simultaneously and other in which the voltages are inverted on alternating gates.

Fig 4.1 shows the schematic diagram of the three different setups for cascaded NOT gates. In Fig 4.1 (i) we can see the logic of the circuit under normal operating conditions. Fig. 4.1 (ii) represents the logic of the gates when all the NOT gates are simultaneously inverted. Fig 4.1 (iii) represents the logic of the gates when gates on alternating levels are subjected to inverted supply.

The resulting output waveforms for a pulse input at various output points is shown in Fig 4.2 and Fig 4.3. When the supply voltage of all the gates are inverted simultaneously, we can see from Fig 4.2 that after 3 stages the output voltage goes beyond recognition to the next level. So this idea of inverting all the voltage rails simultaneously is not feasible to

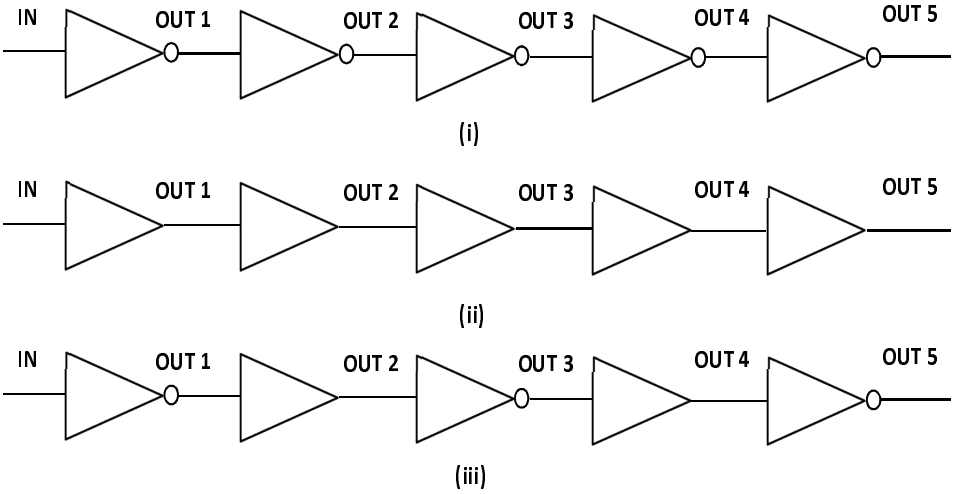


Figure 4.1: Schematic for cascaded INVERTERs for different supply voltage operations (i) Normal operation (ii) Voltage inverted for all stages simultaneously (iii) Voltage inverted for alternate stages

implement. So we consider applying the inverted voltage at alternating gates. The resultant waveform is shown in Fig 4.3. Here we can see that the output voltage drops by a small fraction when the gate operates in the inverted voltage mode. But in the next level (which is in the normal mode of operation), the output is restored back to a strong 0 or strong 1 depending on the input.

The schematic for similar kinds of experiments on two input NAND gates is shown in Fig. 4.4 (i), (ii) and (iii). The corresponding plots for output signals are shown in Fig 4.5 and Fig4.6. In this setup we have assumed that input B of the NAND gate is tied at a constant non-controlling value of 1. Input A is varied using a pulse waveform and the effect is studied at the output of successive gates. The behavior is similar to that of cascaded inverters. Notably under the setup where all the gates are inverted simultaneously, the signal degradation rate is more than in case of inverters.

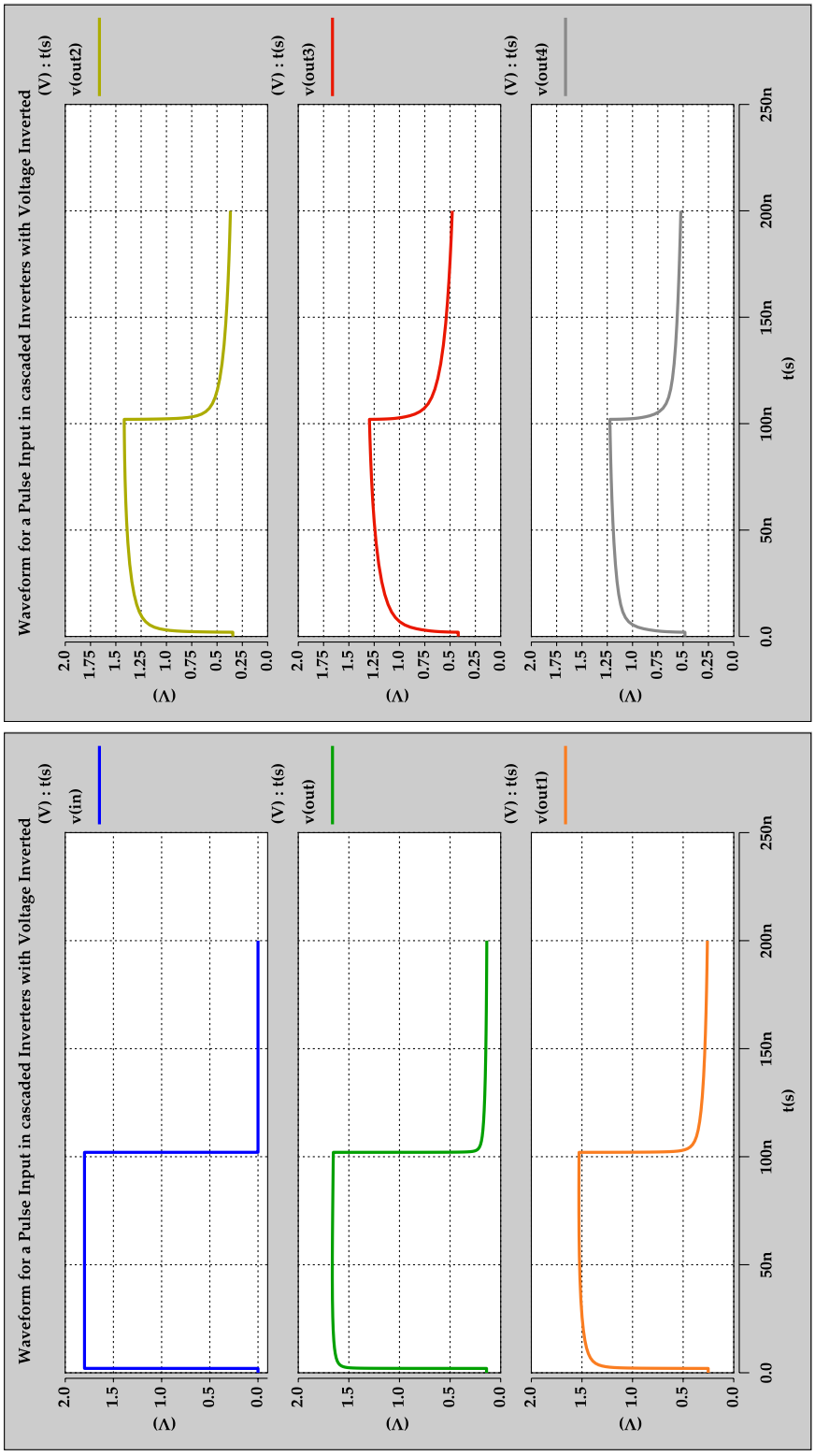


Figure 4.2: Plot of outputs of cascaded INVERTERs with voltage inverted in all stages

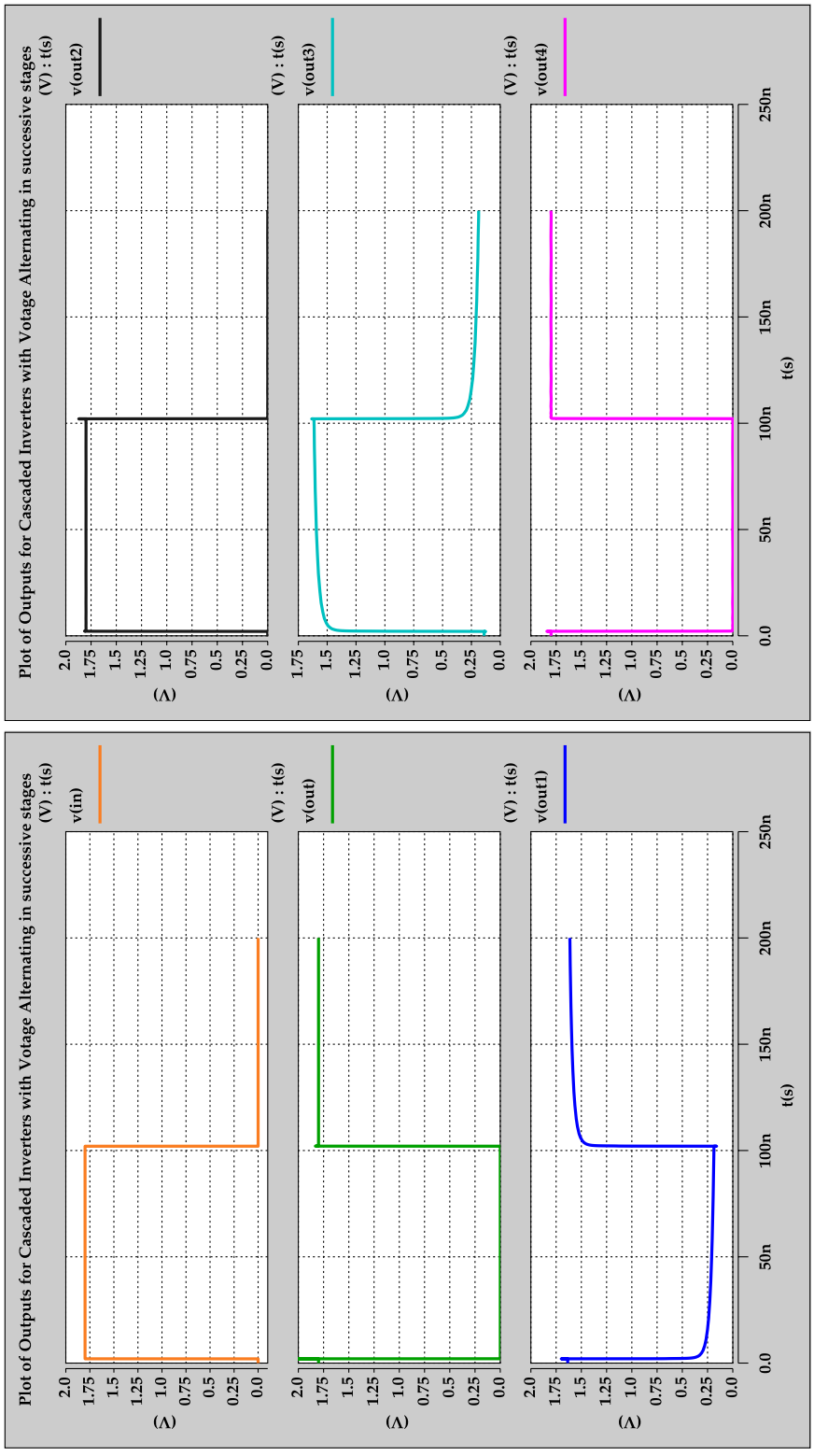


Figure 4.3: Plot of outputs of cascaded INVERTERs with voltage inverted in alternate stages

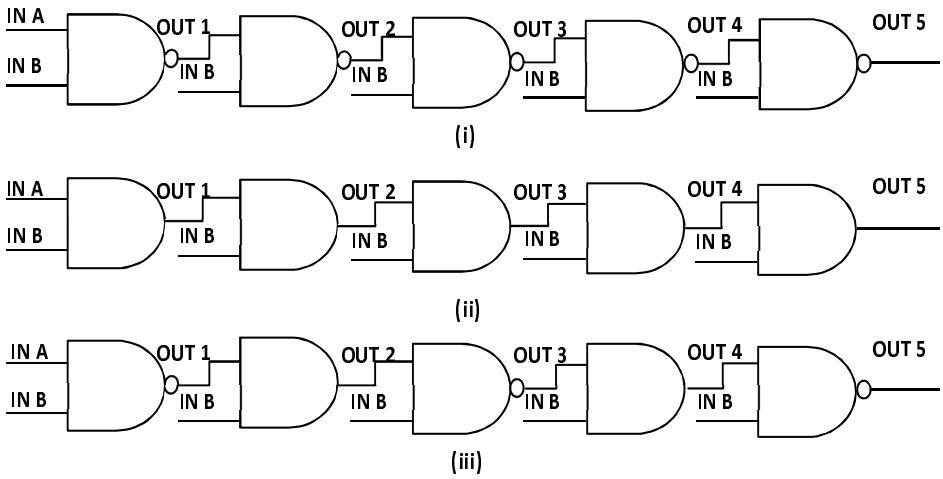


Figure 4.4: Schematic for cascaded NAND gates for different supply voltage operations (i) Normal operation (ii) Voltage inverted for all stages simultaneously (iii) Voltage inverted for alternate stages

4.2 Approach

Our approach is based on two principles - (1) *Gate Logic Inversion* creates a frequent triggering scenario for the Trojan gate and (2) *Sustained Vector Simulation* aims towards minimizing the overall circuit activity to magnify the extra toggles created by the Trojan.

For any gate g, if a logic value $L(L\epsilon\ \{0,1\})$ appears more frequently at its output under a vector set T than the other logic value $\overline{L}$, we call L as the *majority value* and $\overline{L}$ as the *minority value* for g under T. For instance, under a random test set, logic 0 is generally the *majority value* for an AND gate and *minority value* for an OR gate, while logic 1 is the *minority value* for an AND gate and *majority value* for an OR gate. Triggering of a Trojan generally requires the attainment of *minority value* at its output, otherwise it will be frequently triggered and easily detectable. Furthermore, probability of attainment of the *minority value* by a Trojan depends on setting the necessary values on its inputs. Under normal operating conditions, a Trojan is difficult to trigger because the input combination that triggers it is difficult to attain.

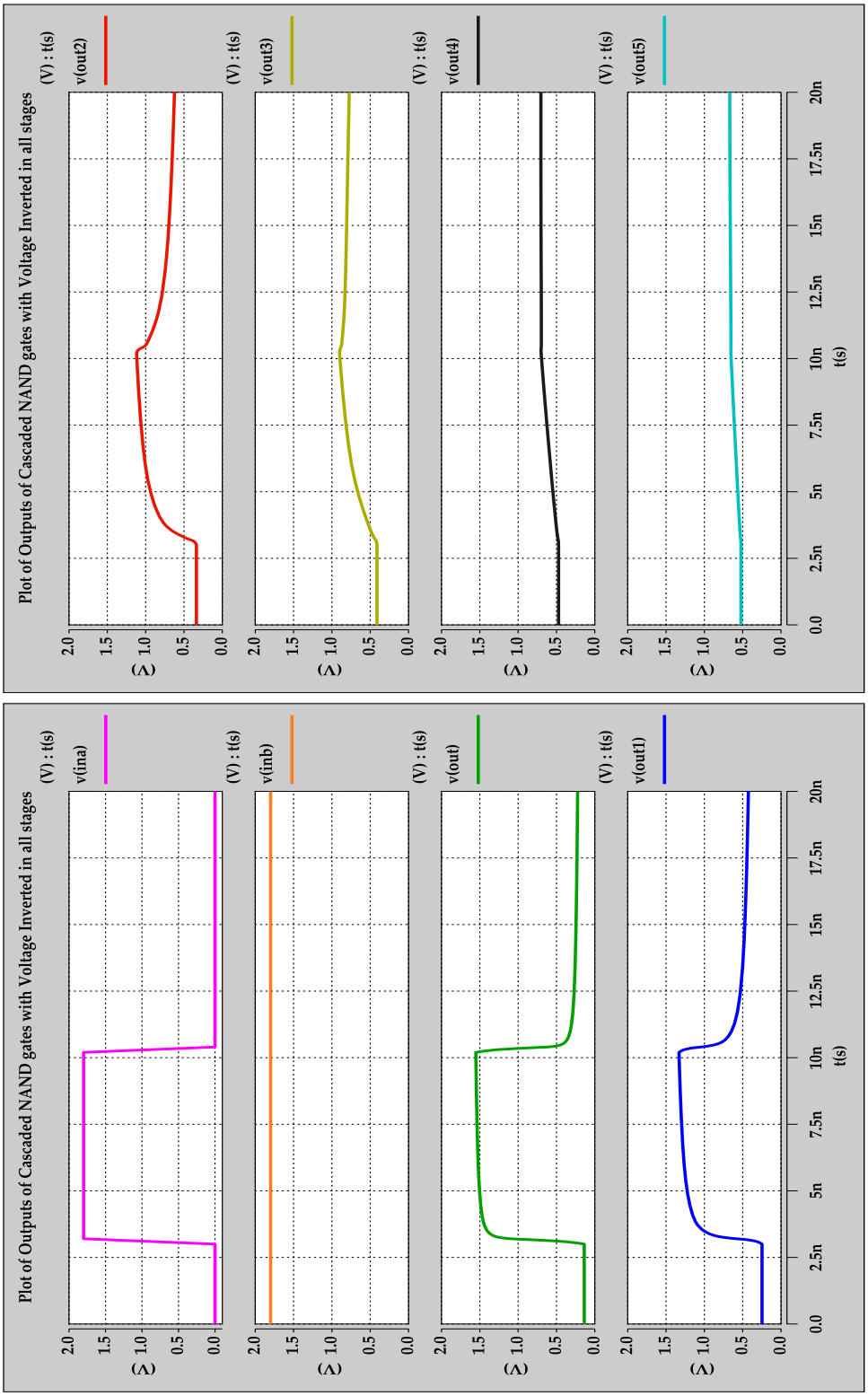


Figure 4.5: Plot of outputs of cascaded NAND gates with voltage inverted in all stages

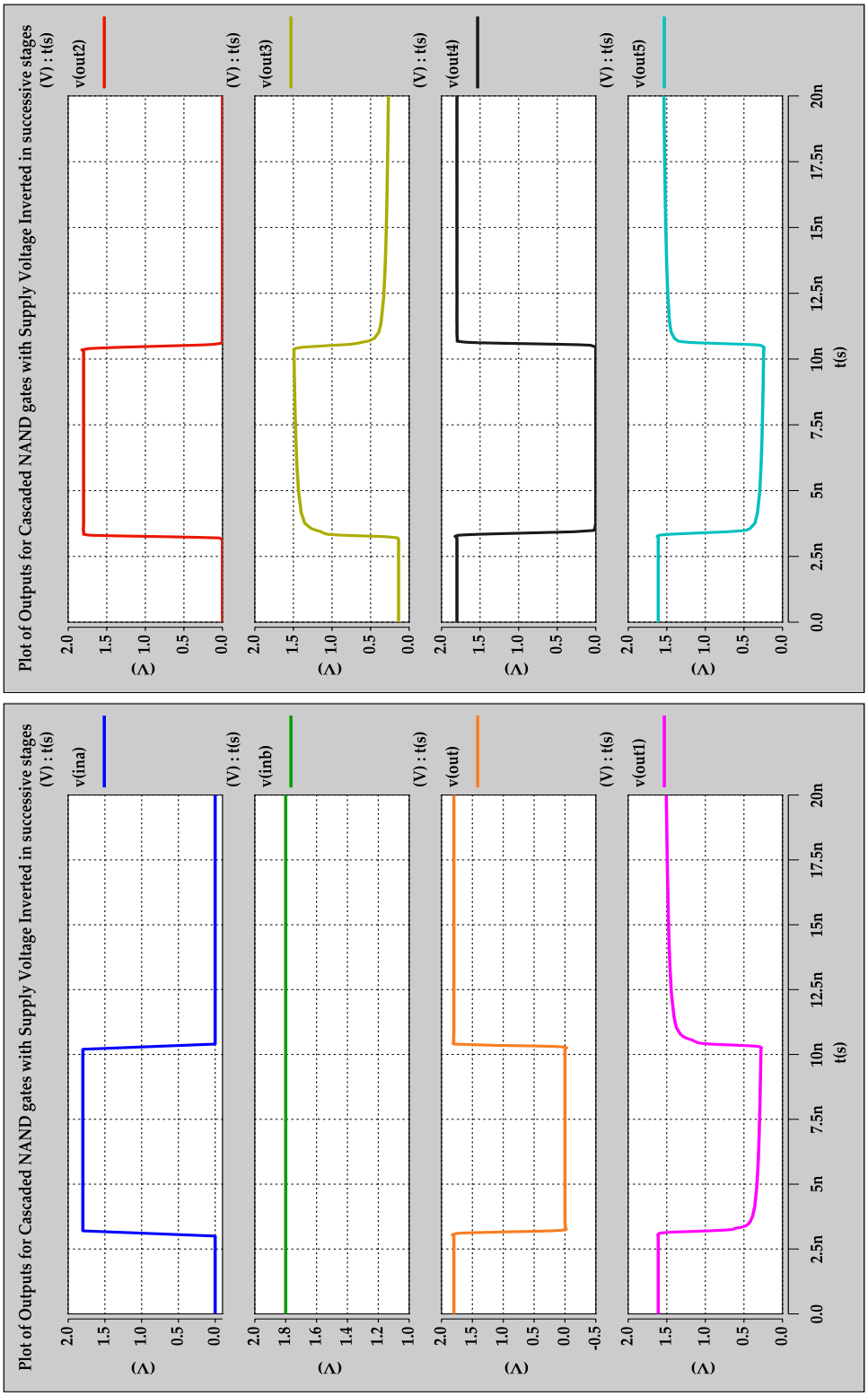


Figure 4.6: Plot of outputs of cascaded NAND with voltage inverted in alternate stages

4.2.1 Gate Logic Inversion

Let us consider a 4-input *Trojan AND* gate shown in Fig. 4.7 (a). It feeds its output to an existing NOR gate in the genuine circuit which we call as the *Payload* gate. Let $P1$, $P2$, $P3$ and $P4$ be the probabilities that the respective inputs of the Trojan gate can be set to non-controlling value 1. So the probability that the output is set to *minority value* 1 can be approximated as $P1 \times P2 \times P3 \times P4$, assuming signal independence of the four inputs of the AND gate. For a general Trojan gate with n inputs, if $P[i]_{NC}$ be the probability that an input i to the Trojan attains a non-controlling value, then the probability that the Trojan is triggered for a given combination of inputs can be approximated by:

$$P(trigger) = \prod_{i=0}^{i=n} P[i]_{NC} \tag{4.1}$$

Now let us consider the case when the supply voltage to the *Trojan AND* gate is inverted while the supply of the *Payload* gate remains same. From the preceding discussion we know that when we invert the supply voltage of a gate, it behaves as its complement. This situation is shown in Fig 4.7 (b). Thus the AND gate is converted into a NAND gate detonating its payload to the NOR gate. The controlling value for the NOR gate is 1 which is the *majority value* for NAND gate. Thus under the inverted voltage conditions, the triggering scenario occurs more frequently (due to the Trojan gate) than in the normal operation. The situation where both the Trojan and *Payload* gate gets inverted simultaneously is shown in Fig. 4.7 (c). In this case, the NOR gate is converted to an OR gate at the output of the Trojan NAND gate. Since 1 is the controlling value for OR gate as well, in this case also, the Trojan is more likely to provide the triggering value.Although the probability of a *minority value* assignment at the output of the Trojan in the *inverted voltage* mode will not be exactly $(1-P)$ (owing to change in the input probabilities because of voltage inversion), the *minority value* and *majority value* gets swapped causing more triggers.

Inverting the voltage for the entire circuit causes the voltage drop to build up across succes-

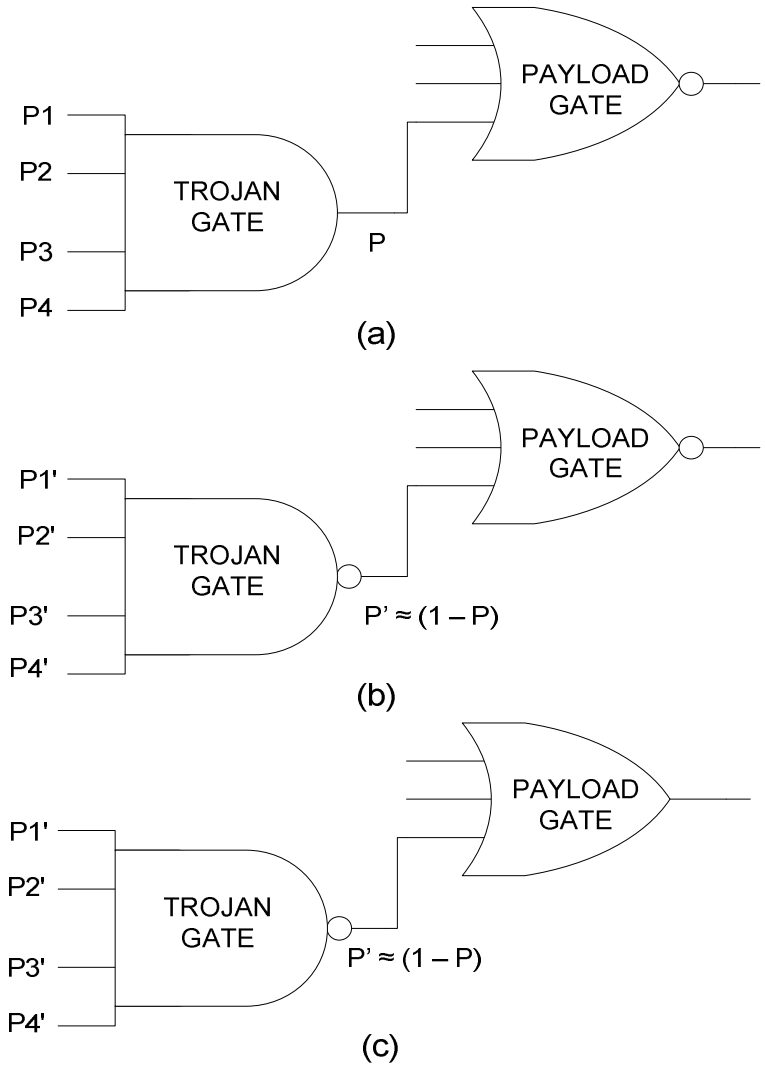


Figure 4.7: (a) Trojan logic under normal voltage supply (b) Trojan logic where only *Trojan* gate is affected by inverted voltage supply (c) Trojan logic where both Trojan and *Payload* gate is affected by inverted voltage supply

sive levels, i.e., the degraded gate potential of one stage of CMOS gates degrades the gate potential on the next stage. This way, after a few stages, the signal voltage may degrade beyond the capability of further propagating any activity, stalling the circuit. So we apply *inverted voltage* to alternate levels in the circuit. Feeding gates in alternate levels with different supply voltages ensure that the drop in voltage at the output of a gate operating in *inverted voltage* domain is compensated by gate(s) in its immediate fanout cone (which operate(s) in the normal voltage domain and hence swings their output rail-to-rail). In our experiments, we first simulate the circuit without any inversion (phase I). Then, we let the odd-level gates be inverted, apply test vectors (phase II), followed by making the odd-level gates normal and even-level gates inverted (phase III). Note that in both cases the flip-flops are never inverted. The Trojan gate, if connected to the supply of the odd level or the even level will be inverted in at least one of the phases of testing (phase II or phase III). Since the output of the Trojan gate attains *majority value* for most of the time whenever it is in *inverted voltage* mode, it will inject a controlling value into the *Payload gate* that propagates down its fanout cone to create more extraneous activities. This is reflected in the power profile of the Trojan affected circuit because in genuine circuit such toggles will not occur. We reinforce the voltage inversion technique with a *sustained vector simulation* to further enhance the behavioral discrepancies. For a detailed description of *sustained vector technique* interested readers should refer to [15].

Threshold voltage(V_{TH}) is a device characteristics. Its value depends on the gate oxide thickness. In [62] the authors have shown the variation of threshold voltage on the channel length as well as on the drain voltage. It is possible to control the device processing parameters so that the *threshold voltage* is only a small fraction of the supply voltage. This ensures that the drop occurring at the output of a *pull-up* or *pull-down* network is not high enough to degrade the signal strength beyond recognition at the input of the next stage.

4.2.2 Methodology

There are three phases in our approach. In the first phase the entire circuit is configured to work under normal operating condition (Fig. 4.8 (a)), i.e., the *pull-up* and *pull-down* networks of all the gates are connected to the V_{DD} and the GND signals respectively. The CUT is exercised by the set of random and sustained random vectors and the power profile is captured. In the second phase the gates in the odd level are supplied with inverted voltages while the rest of the gates are kept under normal operating conditions (Fig 4.8 (b)). This setup is exercised by the same set of random and sustained vectors and again the power profile is obtained. In phase three the same procedure as phase two is repeated but the *voltage inversion* is applied to the even level gates (Fig. 4.8 (c)). Again, the flip-flops are never inverted.

When a gate is in the *inverted voltage* domain, the output voltage doesn't swing rail to rail because of the *threshold voltage* drop. So the power consumed for a transition in *inverted voltage* domain is somewhat smaller than that in a normal voltage domain. For a gate in the normal voltage domain we count a transition as 1. For a gate in the inverted voltage domain a transition is scaled down by a factor to compensate for the reduced voltage swing. The voltage scaling factor SF is determined as per Equation 4.2.

$$SF = \frac{V_{DD(INV)} - V_{GND(INV)}}{V_{DD} - V_{GND}} \tag{4.2}$$

where $V_{DD(INV)}$ and $V_{GND(INV)}$ correspond to the maximum and minimum voltage for a gate in the *inverted voltage* mode and V_{DD} and V_{GND} denote the maximum and minimum voltage for a gate in the normal voltage mode respectively. Assuming a $1.8V$ supply as V_{DD}, $0V$ for GND and $0.2V$ for the *threshold voltage* V_{TH} and substituting $V_{DD(INV)} = V_{DD} - V_{TH}$ and $V_{GND(INV)} = V_{TH}$ in Equation 4.2, we get $SF \simeq 0.75$. Thus in our experiment we count a toggle at the output of a gate in the *inverted voltage* domain as 0.75.

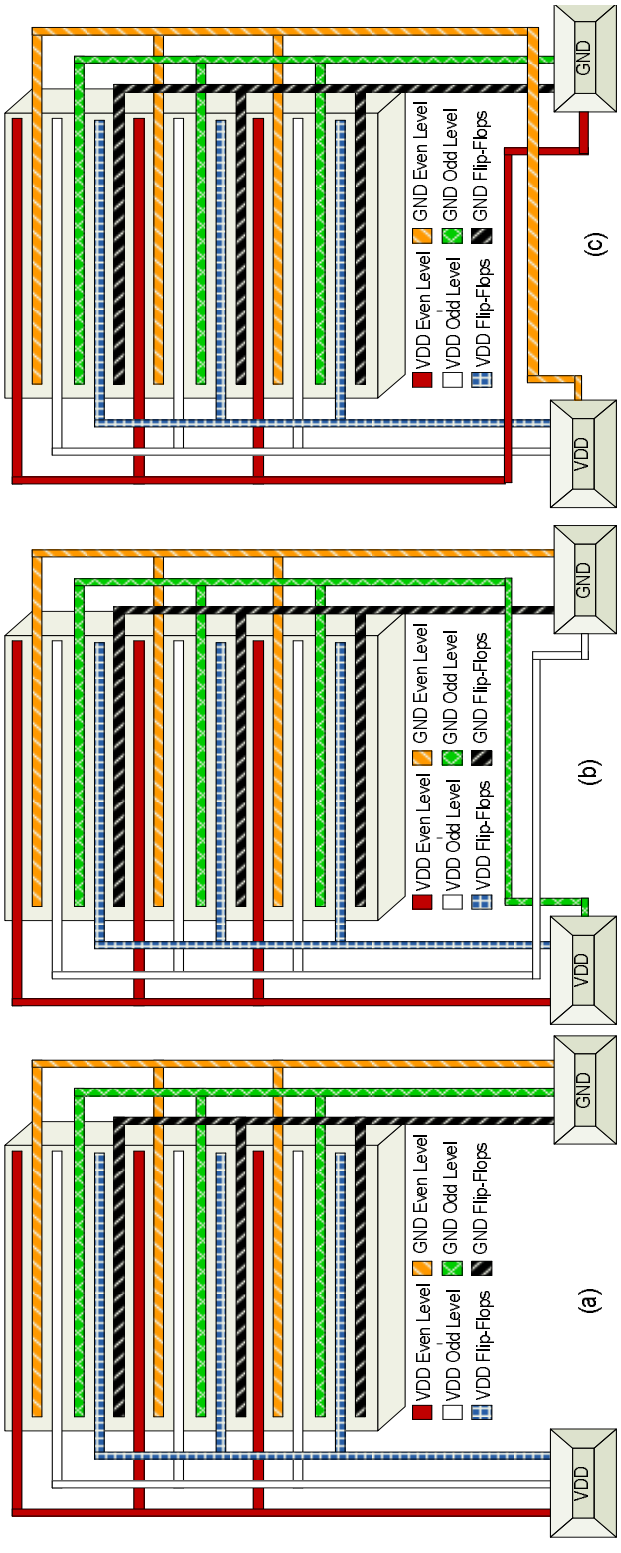


Figure 4.8: Voltage supply for different simulation phases (a) normal operation (b) Odd level gates on inverted voltage (c) Even level gates on inverted voltage

Algorithm 1 Compute maximum % relative power difference using 3-phase simulation

Require: *GenuineCkt*, *TrojanCkt*

1: $SimulationPhases \Leftarrow 3, SimulationModes \Leftarrow 2$
2: $RandVecSet \Leftarrow GenerateRandomVector()$
3: $SustainedRandomVectorSet \Leftarrow GenerateSustainedRandomVector(RandVecSet)$
4: **for all** $SimulationModes$ **do**
5: $VectorSet \Leftarrow SelectVectorSet(SimulationMode)$
6: **for all** $SimulationPhases$ **do**
7: $ComputeOutput(Circuit, VectorSet, SimPhase)$
8: $ComputePowerNumbers(Circuit, VectorSet, SimPhase)$
9: **for all** $(V_i, V_{i+1}) \epsilon VectorSet$ **do**
10: $Compute\%Powerdifferential(GenuineCkt, TrojanCkt)$
11: **end for**
12: $ComputeMaxPowerDifference(GenuineCkt, TrojanCkt)$
13: **if** $Outputs(GenuineCkt) \neq Outputs(TrojaCkt)$ **then**
14: $TrojanDetectedAtOutput = TRUE$
15: **end if**
16: **end for**
17: **end for**

4.2.3 Algorithm

The basic steps of our methodology is given in Algorithm 1 and the functions used in it are listed and explained in Table 4.1.

4.3 Experimental Results

Fig. 4.9 to Fig. 4.12 shows relative increase in percentage activity of Trojan affected circuit over the genuine circuit for an AND/OR Trojan when the CUT is simulated with random vectors and sustained random vectors respectively. In both these cases, magnification obtained by using random vectors on the *inverted voltage* set up are much higher when compared to the *normal voltage* mode. In almost all cases, the random vectors failed to create a difference in excess of 5% which is typically considered as the process variation threshold [12]. Sustained vector technique can achieve substantial improvement in creating a measurable

Table 4.1: Functions of Algorithm 1

Function	**Purpose**
GenerateRandomVector()	Generates a random vector set
GenerateSustainedRandomVector(*RandVecSet*)	Generates a sustained random vector set based on the *RandVecSet*
SelectVectorSet(*SimulationMode*)	Selects the vector set as per the *SimulationMode*
ComputeOutput(*Circuit*, *VecSet*, *SimPhase*)	Compute the corresponding *Circuit* output for a given *VectorSet* under chosen *SimPhase*
ComputePowerNumbers(*Circuit*, *VecSet*, *SimPhase*)	Compute the corresponding *Circuit* activity numbers for a given *VecSet* under chosen *SimPhase*
Compute%PowerDifferential(*GenuineCkt*, *TrojanCkt*)	Compute the % activity difference between *GenuineCkt* and *TrojanCkt*
ComputeMaxPowerDifference(*GenuineCkt*, *TrojanCkt*)	Compute maximum power difference between *GenuineCkt* and *TrojanCkt*

power difference when applied on top of the proposed *inverted voltage* methodology. In most cases, as shown in the Fig. 4.11 and Fig. 4.12, the difference is magnified by many times, sometimes by several orders of magnitude. The cases in which the *Trojans* were detectable at the output, the corresponding bar is marked by a **D**. The experiments were carried out on a 3GHz Intel dual-core quad processor machine with 2GB RAM. The entire code for test generation is written in C++. The typical test generation time varied from a few seconds for the small circuits to a few minutes for the larger benchmarks. All *Trojans* used in our experiments were tested to be non discernible at the output using the set of 10000 random vector sequence.

Exposure at a primary output usually happens for the smaller benchmarks because of shorter propagation paths of triggered Trojan output signal to the circuit primary output. In larger benchmarks, triggered values are restricted from reaching the output by a controlling value on some other gate in its propagation path. When the primary outputs can differentiate the genuine and Trojan circuits, no power analysis is needed. Yet for the circuits where we could detect the *Trojans* at the primary output, we still report the power difference values because it may not always be possible to observe *Trojans* at the output even for smaller circuits ($s1423$). But our results show that power differential is indeed a robust measure to detect malicious activity irrespective of the output values!

For circuits such as $s3271$, $s3330$ and $s35932$, which are inherently very high toggling, previous method has proved not so effective for Trojan detection. As clear from the results, our proposed technique proves to be very effective is creating an observable difference in these circuits as well. For $s9234$ with OR-type Trojan and for $s13207$ with AND-type Trojan, they cannot be detected at the output. However, the sustained random technique with normal voltage mode provides highest activity magnification. This is because changing the voltage supplies changes the logic and it is possible that such an alteration renders the circuit less sensitive to activities from the Trojan. Nevertheless, voltage inversion was still able to detect the difference, albeit with a smaller magnification. In such a case there will be reduction in the magnification ratio.

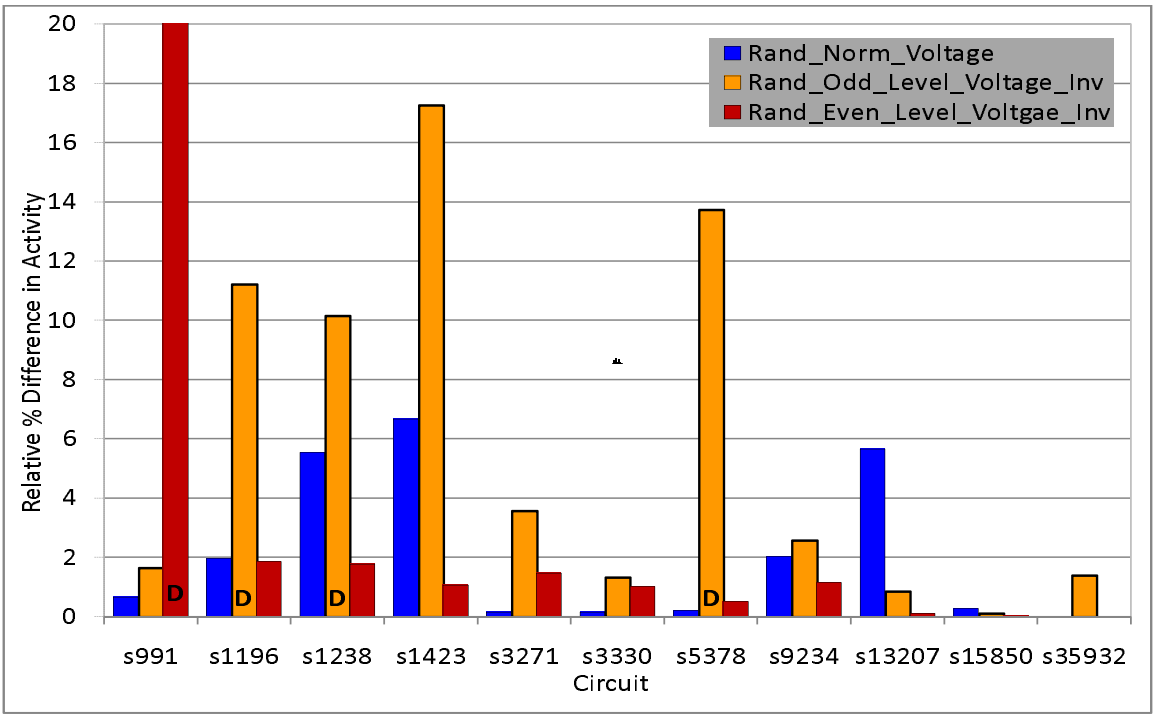


Figure 4.9: Relative % difference in activity for AND Trojans using random vectors

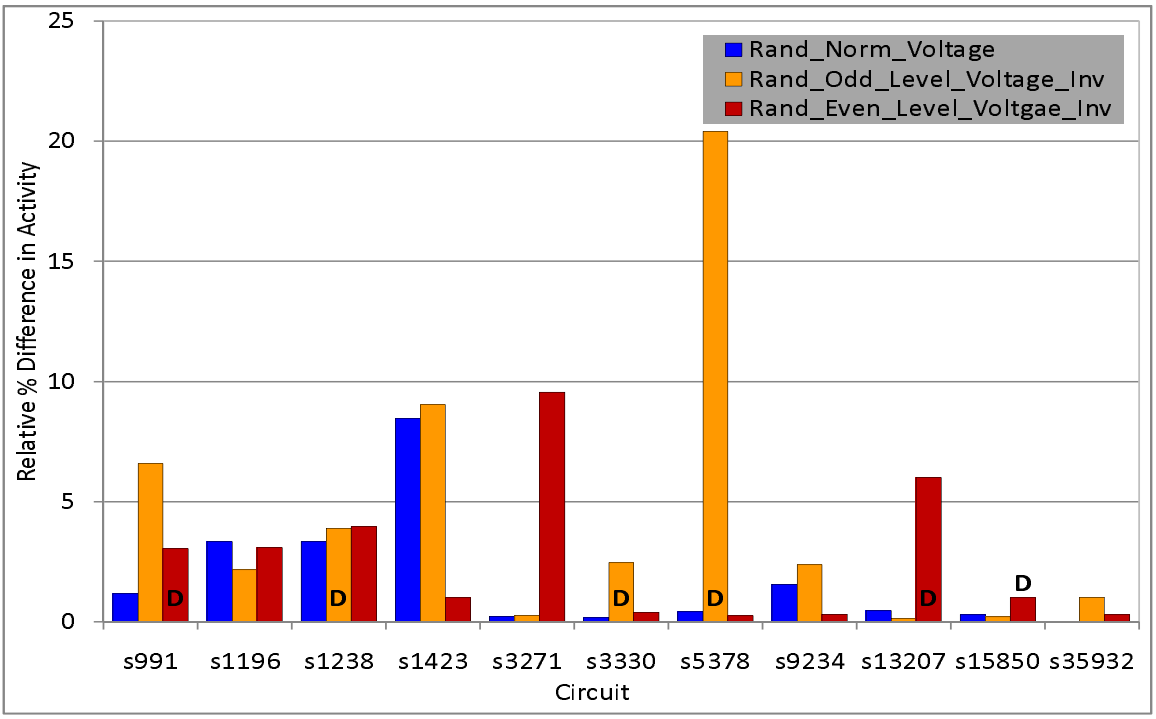


Figure 4.10: Relative % difference in activity for OR Trojans with random vectors

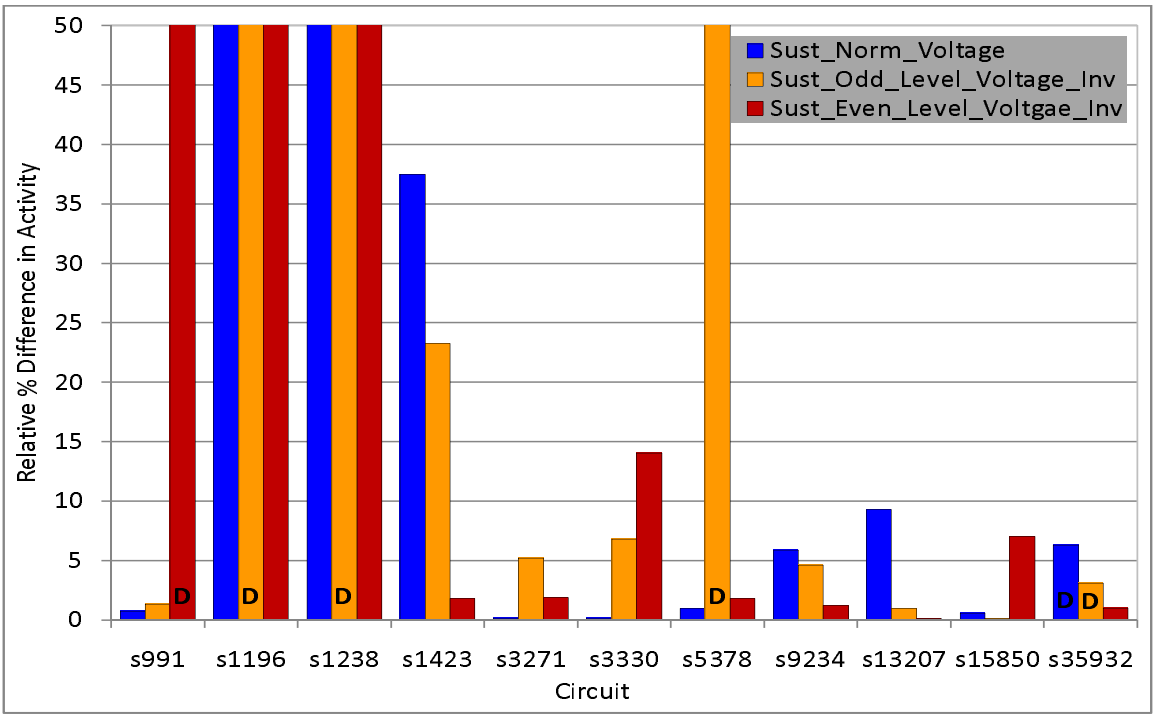


Figure 4.11: Relative % difference in activity for AND Trojans with sustained random vectors

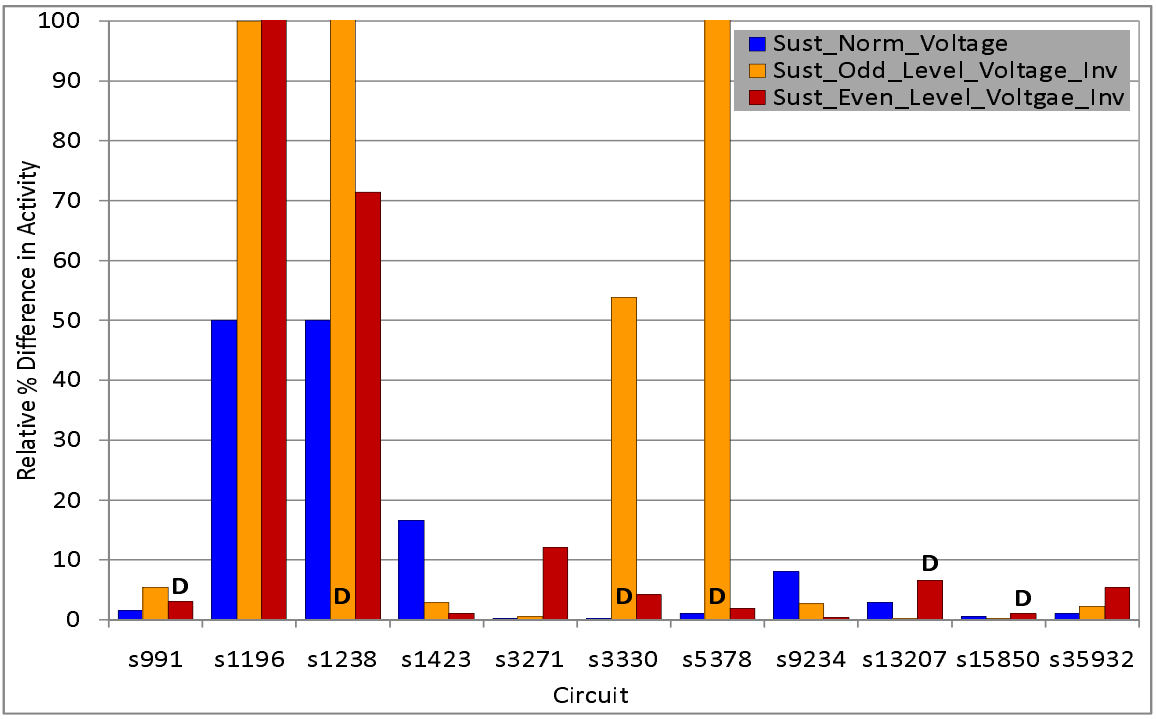


Figure 4.12: Relative % difference in activity for OR Trojans with sustained random vectors

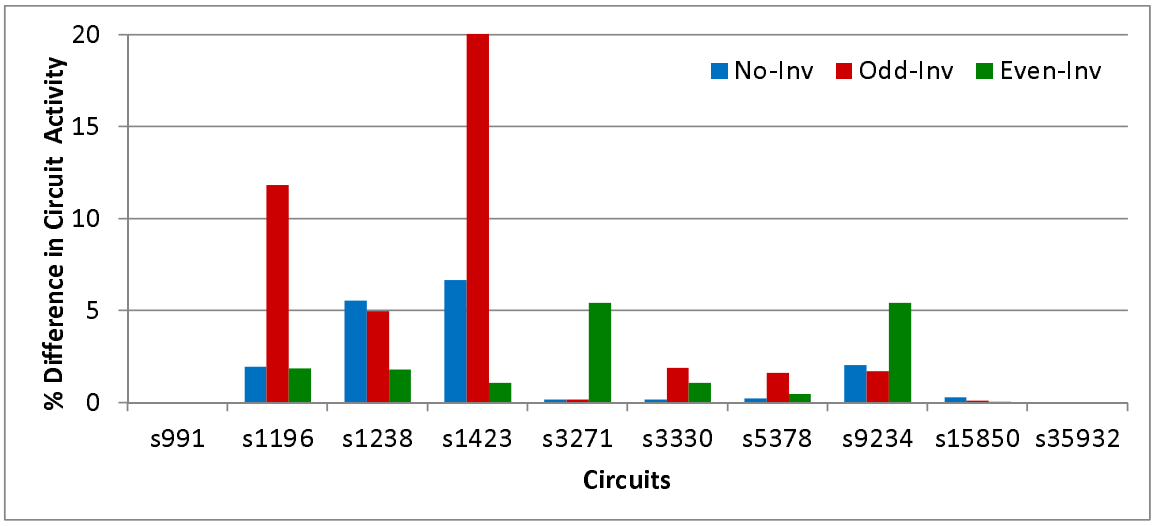


Figure 4.13: Relative % difference in activity for scan resistant AND Trojans using random vectors

Our first set of experiments were done on *functional* Trojans i.e. they were functionally very hard to activate/detect. In another related experiment, we tried testing the same Trojans against scan patterns. Our experiments showed that these Trojans were not resistant to scan patterns. So we came up with another set of Trojans which were functionally hard to activate as well as scan resistant. We used the approach mentioned in section 2.6.1 in the Background chapter.

Our experiments with this new set of Trojans proved that this methodology is equally effective for scan resistant Trojans. In Table 4.2, we have shown the detection status for AND and OR Trojans, both for random vector simulation as well as for sustained random vector simulation. An entry of 'X' is used when we could not come up with a Trojan for that circuit. The power profile analysis for the same has been shown in Fig. 4.13, Fig. 4.14, Fig. 4.15 and Fig. 4.16.

We analyzed the area impact of the proposed technique and the results are shown in Fig. 4.17 and Fig 4.18 respectively. We used $c432$ ISCAS'85 benchmark to study the area analysis. In the first experiment, we used a single module flat netlist and a standard design flow to come up with the pre-routing and post routing layouts shown in Fig 4.17 (i) and (ii) respectively.

Table 4.2: Detection Directly at the Primary Outputs for Scan-Resistant Trojans

Type	**AND Trojan**						**OR Trojan**					
Vectors	Random			Sustained			Random			Sustained		
Circuits	Norm Volt Sup.	OL Volt Inv.	EL Volt Inv.	Norm Volt Sup.	OL Volt Inv.	EL Volt Inv.	Norm Volt Sup.	OL Volt Inv.	EL Volt Inv.	Norm Volt Sup.	OL Volt Inv.	EL Volt Inv.
s991	X	X	X	X	X	X	UnObs	UnObs	Obs	UnObs	UnObs	Obs
s1196	UnObs	Obs	UnObs	UnObs	Obs	UnObs	UnObs	UnObs	Obs	UnObs	UnObs	Obs
s1238	UnObs	UnObs	UnObs	UnObs	UnObs	UnObs	UnObs	Obs	UnObs	UnObs	Obs	UnObs
s1423	UnObs	Obs	UnObs	UnObs	Obs	UnObs	UnObs	Obs	UnObs	UnObs	Obs	UnObs
s3271	UnObs	UnObs	Obs	UnObs	UnObs	Obs	UnObs	UnObs	Obs	UnObs	UnObs	Obs
s3330	UnObs	Obs	Obs	UnObs	Obs	Obs	UnObs	Obs	UnObs	UnObs	Obs	UnObs
s5378	UnObs	Obs	UnObs	UnObs	Obs	UnObs	X	X	X	X	X	X
s9234	UnObs	UnObs	Obs	UnObs	UnObs	Obs	UnObs	UnObs	UnObs	UnObs	UnObs	UnObs
s15850	UnObs	UnObs	UnObs	UnObs	UnObs	UnObs	UnObs	Obs	Obs	UnObs	Obs	Obs
s35932	X	X	X	X	X	X	UnObs	Obs	UnObs	UnObs	Obs	UnObs

Obs = Observable at Output, *UnObs* = Not Observable at Output
OL = Odd Level, *EL* = Even Level

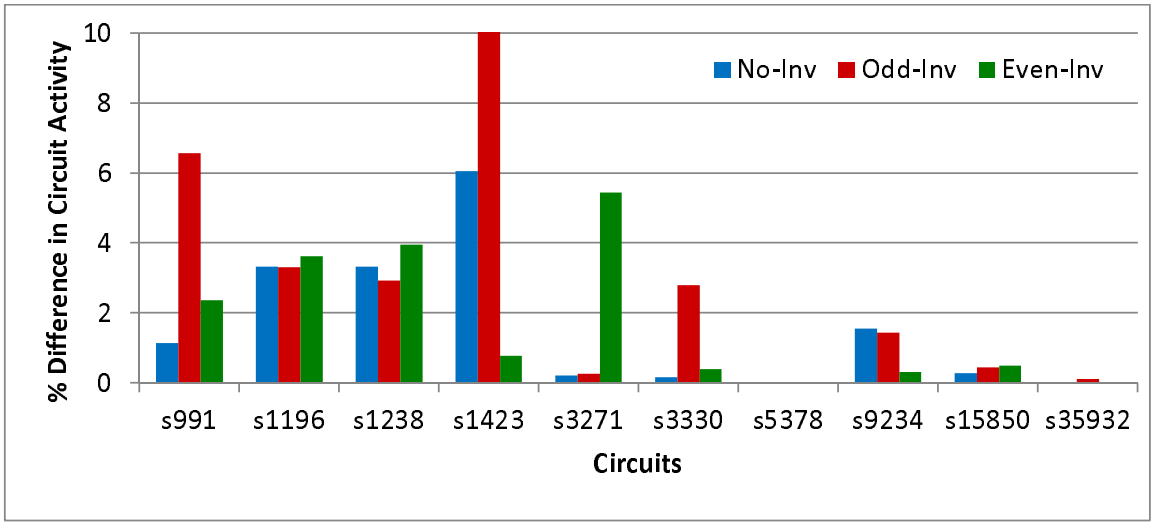


Figure 4.14: Relative % difference in activity for scan resistant OR Trojans with random vectors

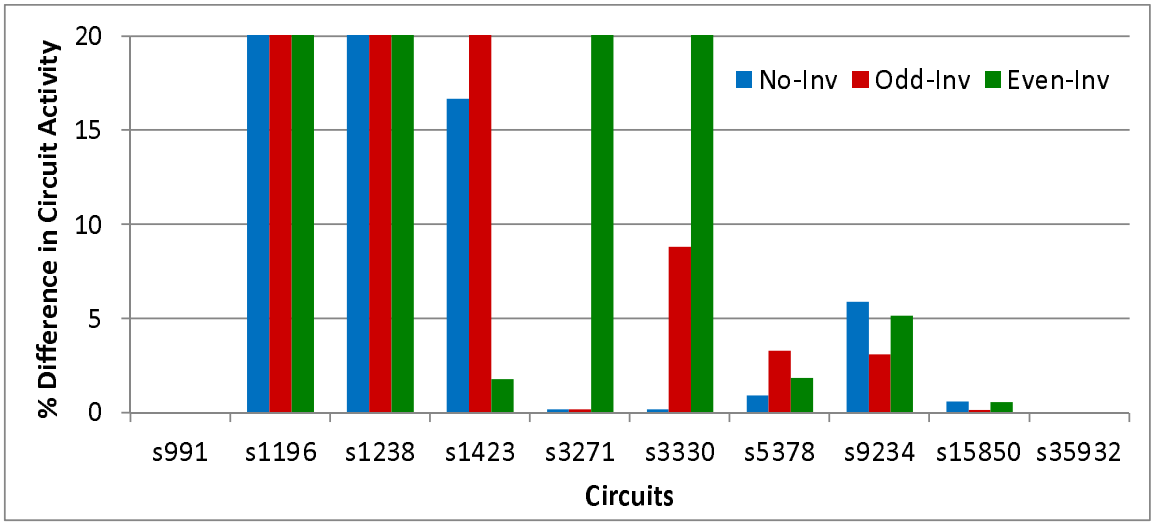


Figure 4.15: Relative % difference in activity for scan resistant AND Trojans with sustained random vectors

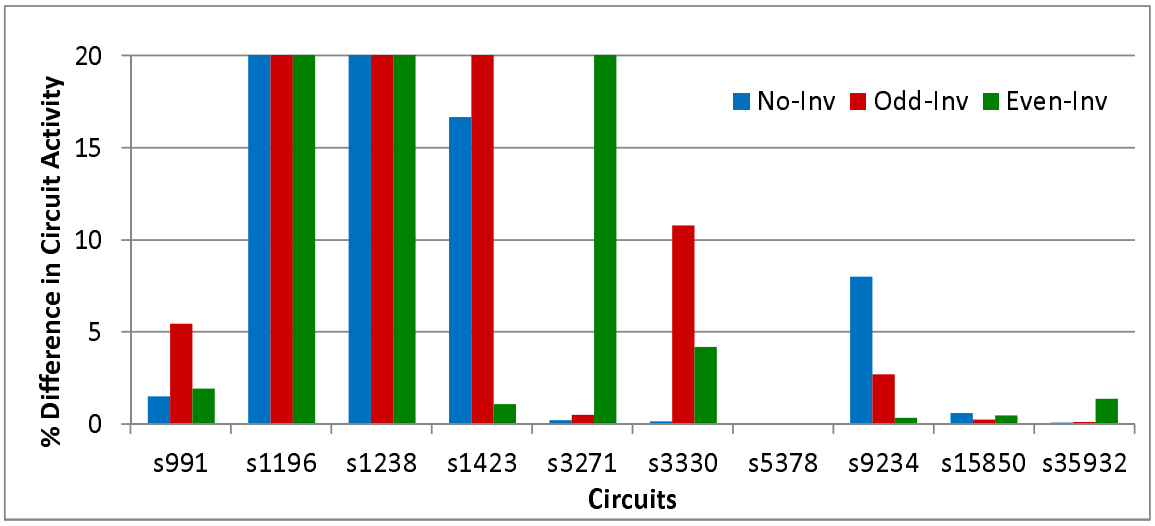


Figure 4.16: Relative % difference in activity for scan resistant OR Trojans with sustained random vectors

In the second experiment, we partitioned the netlist into two submodules containing the gates in odd and even level separately. While doing placement, we placed the odd level and even level modules separately so that the voltage distribution is easy. The pre-routing and post routing layouts for this case are shown in Fig. 4.18 (i) and (ii) respectively. We used an IBM 130 nm standard performance technology, and an industry-quality standard cell library from ARM. The synthesis tool was Synopsys Design Compiler, and the place and route tool was Synopsys IC Compiler. We attempted to do a minimum-area place and route flow with a square core for the original circuit. For the partitioned circuit we split the core into a section for each partition and increased the core size until the router could succesfully complete.

The pre-routing area for the two cases came out to be $800.6\mu m^2$ and $1074\mu m^2$ respectively. Thus there is a penalty of around 30% area overhead. There are two main reasons for this viz. (1) Optimizations cannot be done by the Design Compiler across voltage domains and (ii) Different voltage levels required creation of separate rails to supply standard cells with similar fanins which were otherwise placed under same voltage rails. This is evident from Fig. 4.17 (i) and Fig. 4.18 (i) where we can see that there is an increase in the number of rails from the non-partitioned scheme to the partitioned scheme.

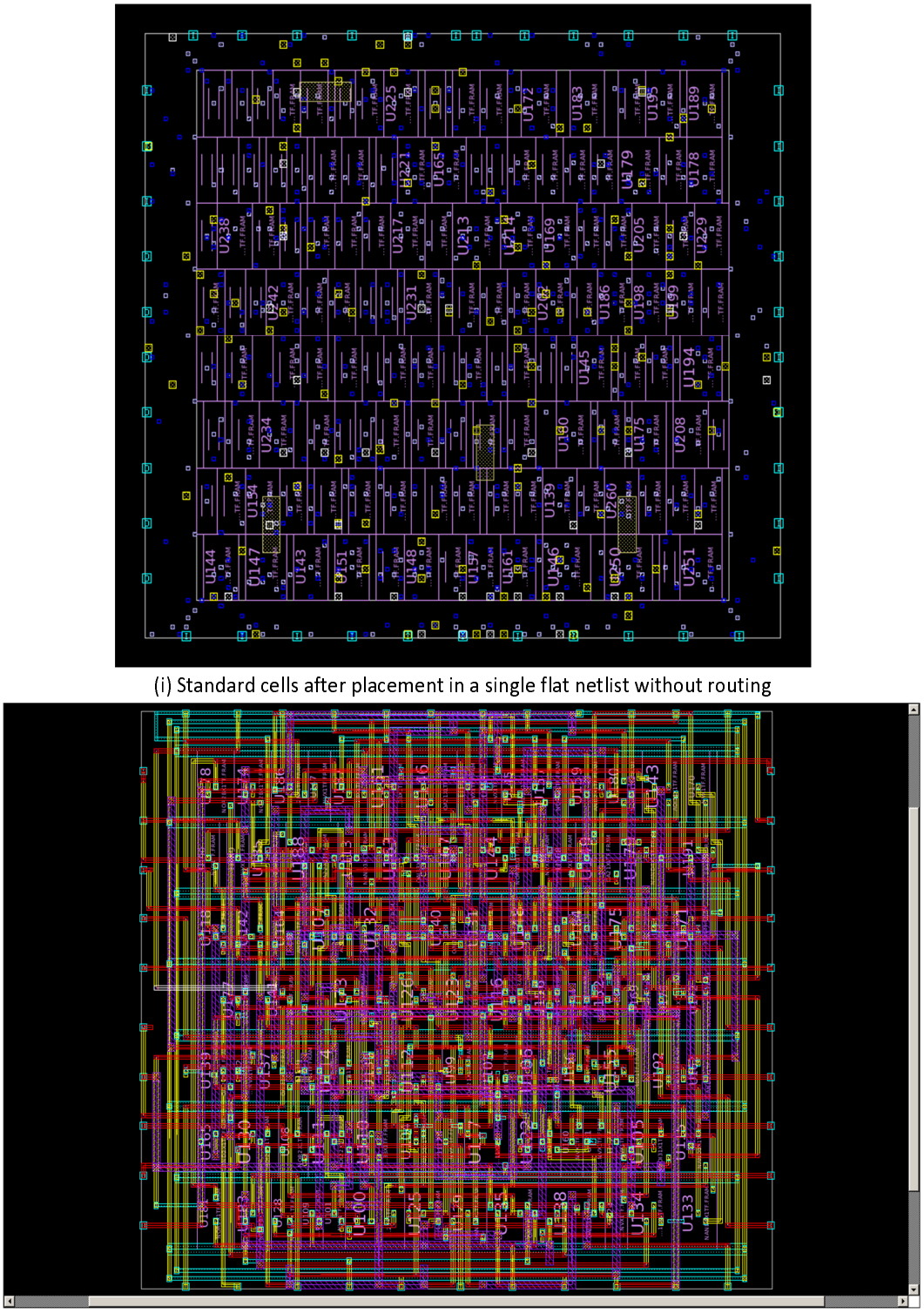
The post-routing area for the same circuit comes out to be $887\mu m^2$ and $1880\mu m^2$ respectively.

Thus we find that the area doubles when we do the routing. This is because of increased length of routing wires. When all the cells are placed in a single voltage domain, the routing tool can maximally optimize the routing lengths. But when the odd level and even level gates are separated into distinct areas, each cell in the even level has to be routed to some cell in the odd level area and vice versa. Partitioning increases the routing congestion, which therefore requires a larger core to route succesfully. This adds substantially to the routing overhead.

An optimization that would help reducing the area overhead both in pre-routing and post-routing phases is interleaved voltage distribution in dual voltage mode. In our current scenario, we have separated even and odd level cells in two distinct chunks and created a separate voltage distribution scheme. In an interleaved distribution, these chunk of cells would be further distributed into smaller groups and the voltage rails need to be placed alternatively. Consequently, routing tool can optimize the routing area more effectively because it will have multiple options for placing each group of cells.

4.4 Summary

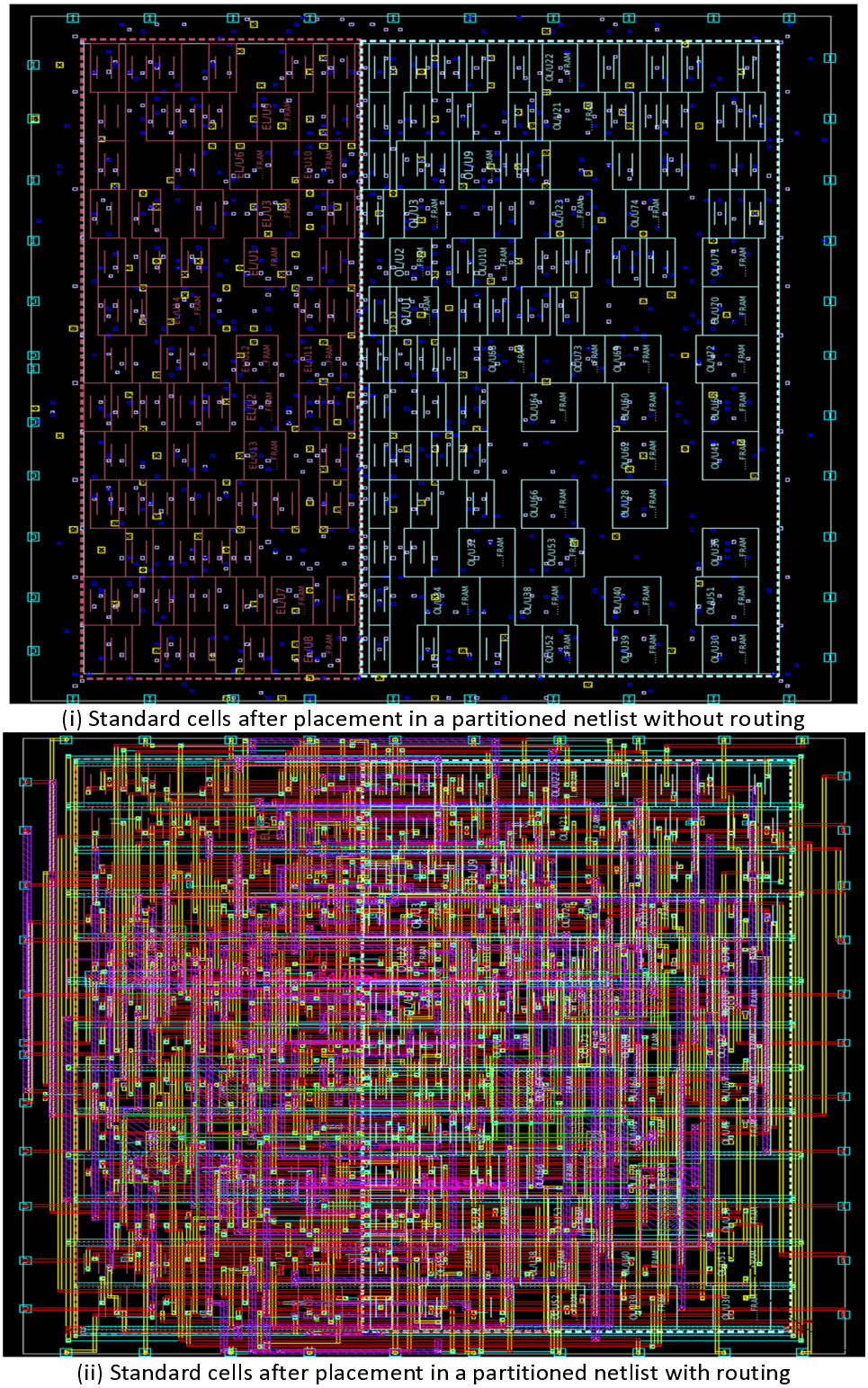
We presented a new *inverted voltage* technique to better activate and detect the malicious insertions in third party ICs. This setup is also coupled by a *sustained vector set* ensuring minimum genuine activity inside the design so that extraneous activity created by *Trojan* gets exaggerated. Experimental results on ISCAS'89 benchmarks prove that the approach is very effective in detecting the presence of very small *Trojans* inside the designs. For almost half of the instances, change in logic created by inverting the voltage supply readily exposes the Trojan directly at the output. For those which are not detected at the primary outputs, there is a magnification in activity difference by many times. Future work in this direction will be an actual estimation of the overhead involved in laying out the supply rails in order to meet the requirements of providing controlled voltages to the even and odd level gates.



(i) Standard cells after placement in a single flat netlist without routing

(ii) Standard cells after placement in a single flat netlist with routing

Figure 4.17: Layout of c432 under no voltage inversion scheme



(i) Standard cells after placement in a partitioned netlist without routing

(ii) Standard cells after placement in a partitioned netlist with routing

Figure 4.18: Layout of c432 under voltage inversion scheme

Chapter 5

Suspect-Signal Guided Sequential Equivalence Checking

This chapter presents a step-by-step approach for isolating and detecting malicious insertion(s) implanted in a third party Intellectual Property (3PIP). Such malicious implantation(s) is/are beyond simple, direct recognition. When considered from a fault detection point of view, one of the faults (either stuck-at-0 or stuck-at-1) on such nodes are very hard to be detected. Moreover, distinguishing a malicious insertion is not as straightforward as simply recognizing the hard-to-detect faults. In our approach, we first remove those easy-to-detect faults in the 3PIP using *functional vectors*. This is followed by a optional *N-detect full-scan Automatic Test Pattern Generation (ATPG)* to identify those fault which are functionally hard to excite and/or propagate. These identified faults (either from step-1 or from step-2) are then processed by a modified *sequential equivalence checking* (SEC) setup to compare behavioral consistency between the original design and the 3PIPs. Essentially, we intelligently use the selected faults in the circuit to direct the SEC to give valuable information about such implantations. Finally, a *region isolation* approach is applied on the filtered faults to map the possible location(s) of the insertion(s).

5.1 Motivation

Various post-silicon techniques have been proposed in the past few years to tackle the detection of malicious ICs manufactured abroad. In [12], the authors have compared power profile of the circuit under test (CUT) with the genuine circuit to assess any unwanted intrusion in the CUT. Since the extra power consumed by the Trojan is very small, they have used a noise filtering technique to pronounce the possible extraneous behavior. This was leveraged in [14] where the authors have augmented the power profile mapping with partition based mechanism so as to specifically exaggerate the activity in different regions of the circuit while reducing the same for the rest of the circuit. In [15], self transitional behavior of a system has been exploited to minimize circuit activity. In a self transitional system, states undergo self-transitions inducing toggles in the circuit when the inputs are retained at constant value for successive cycles. In [70], the authors have used variation in the current drawn from supply ports on different power pads to locate the Trojan. In [16] voltage switching on supply rails have been proposed to alter the circuit logic thereby making the *Trojan(s)* more observable on outputs. In [42, 50], the authors have exploited additional gate delay introduced by the Trojan to alter the delay signature of the path on which it resides. [69] shows that such delay testing mechanisms are effective for Trojan detection even in the presence of process variation.

While all the aforementioned methods are targeted to scrutinize ICs for the presence of *Trojan(s)* in the post-silicon stage, research is severely lacking in the pre-silicon stages. Since future designs may increasingly involve third party contributions in the form of soft IPs, it is equally likely that an adverse change may be introduced during these design or optimization stages. Our work attempts to address this problem of checking the suspicious netlist (which is optimized version of a design and we call this as *sus circuit*) against an original behavior (which we can quickly synthesize and call this unoptimized synthesized circuit as *spec circuit*). Since these two netlists may bear no/little resemblance in terms of the number and/or type of gates/flip-flops used as well as their connectivity, ensuring their

equivalence by straightforward inspection is unlikely. Thus, the problem maps to performing a SEC between the *spec circuit* and the *sus circuit*. However, in most of the cases such a full blown SEC approach is infeasible. Moreover, the fact that Trojan triggering conditions can take millions of clock cycles to achieve makes SEC impractical for these circuits.

Since a Trojan is stealthy, a node that represents the final output (payload) of the Trojan is hard to excite and observe simultaneously. Thus, it is almost impossible to detect the stuck-at-fault that requires the Trojan triggering value for fault excitation at this node in the functional mode! In our discussion, when we refer to a fault we imply the corresponding signal associated with it. Since this extraneous signal is not present in the *spec circuit*, the malicious behavior will be absent in it. Our analysis shows that if we can intelligently identify candidate signals (faults) of the circuit that are most likely to correspond to potential discrepancies, a diluted SEC can be fine-tuned to give valuable information about such implantations.

5.2 Approach

Our methodology consists of four steps. In step-I we use a functional test set to exclude the so called functionally easy-to-detect faults. In step-II (which is optional as we will discuss in the experimental results section), an N-detect full scan ATPG is employed to identify from those remaining faults which are detected only a limited number of times. In step-III we formulate a suspect-signal guided SEC on this filtered fault list to check if their behavior can be mapped to the *spec circuit*. Finally, in step-IV we perform a weighted region isolation to sieve out suspect locations. Our experiments with ISCAS'89 and ITC'99 benchmarks show that this methodology is very effective in distinguishing a stealthy, malicious 3PIP with a clear indication of the region of tamper.

5.2.1 Functional Vectors

In the first step, we use functional vectors (could be generated by a quick run of sequential ATPG, random non-scan vectors, or from designer-provided sequence) to detect and drop as many stuck-at-faults as possible from the *sus circuit*. In general, functional vectors activate and propagate/observe many internal signals corresponding to the fault under consideration. These easy-to-detect signals can be omitted for future consideration. On the other hand, since *Trojan(s)* are hard to excite and propagate (even in the full-scan mode), it is rare that functional vectors would detect them. Otherwise, the functional vectors themselves serve as a witness to differentiating the two circuits. Those signals corresponding to faults undetected by functional vectors are categorized as *suspect candidates*. Our functional vectors set consist of 200,000 vectors out of which 100,000 vectors are purely random and the remaining are sustained random vectors (4000 $\times$ 25 cycles for each vector = 100,000 sustained random vectors). The motivation behind supplementing the test set with sustained random vectors is that they are particularly effective in increasing fault coverage for circuits in which the states have a self-transitional nature [15].

5.2.2 N-Detect Full Scan ATPG

Initially all the signals are equally suspicious as Trojan candidates. After the easy-to-observe signals (faults) are removed from the suspect candidate list in the previous step, we employ a full-scan N-detect ATPG to process the remaining faults. In a sequential circuit, detection of a fault depends as much on the state as on the primary inputs. In the functional mode, it is possible for a fault to have one (or very few) detection vector(s) because of the difficulty in achieving the required state configuration. In full scan mode the state bits which were otherwise controlled by the internal nets become fully controllable. So the same fault can be detected by multiple full-scan vectors.

Since *Trojans* have rare triggering conditions. This is because a Trojan is a very specific

logical interconnection and a few specific full-scan vectors can set its output to the triggering value. A vector cube is an incompletely specified vector in which some of the values are 'X's. Consequently, if there exists multiple unique vector cubes (inputs + states) that can activate and observe fault on a signal, this signal is less likely to correspond to a Trojan. Nevertheless, an unconstrained initial state in the full-scan mode produces functionally illegal states making Trojan observable at the output by some vectors which are not functionally reachable. Thus, we resorted to a N-detect full-scan ATPG instead of selecting only those faults which are uniquely excitable in full-scan mode.

Signals corresponding to combinationally untestable stuck-at-faults are removed from the *suspect candidate* list in this step. To keep the filtered list manageable, the threshold N should be suitably chosen. Keeping N too low will remove the Trojan-related signals (faults) whereas making it high will result in a large set of *suspect candidates* to process in the next step. In our experiments we have set N to 4, which means we consider only those faults which are detected less than four times in the full-scan mode. However, we note that this threshold can be tuned for different circuits.

5.2.3 Diluted Sequential Equivalence Checking

The third step is *diluted SEC*. It consists of two conjoining parts viz. PART-I and PART-II represented in Fig. 5.1.

In PART-I we ensure that the corresponding stuck-at fault on the selected signal S (from the *suspected candidate* list) is indeed observable at the output (and not on the pseudo primary outputs(PPO-2)). To realize this, we set up a miter shown in Fig 5.1 PART-I. It contains two instances of the *sus circuit*, each of which has been unrolled a specific number of time frames. The fault on S is injected in the middle time frame of one of the instances. Since we inject the fault in the middle time frame, our unroll depth is always an odd number. The initial state ($PPI\ SUS$) as well as the $INPUTS$ are left unconstrained. The effect produced by S is captured at the output of the miter (gate OR-1). The pseudo primary

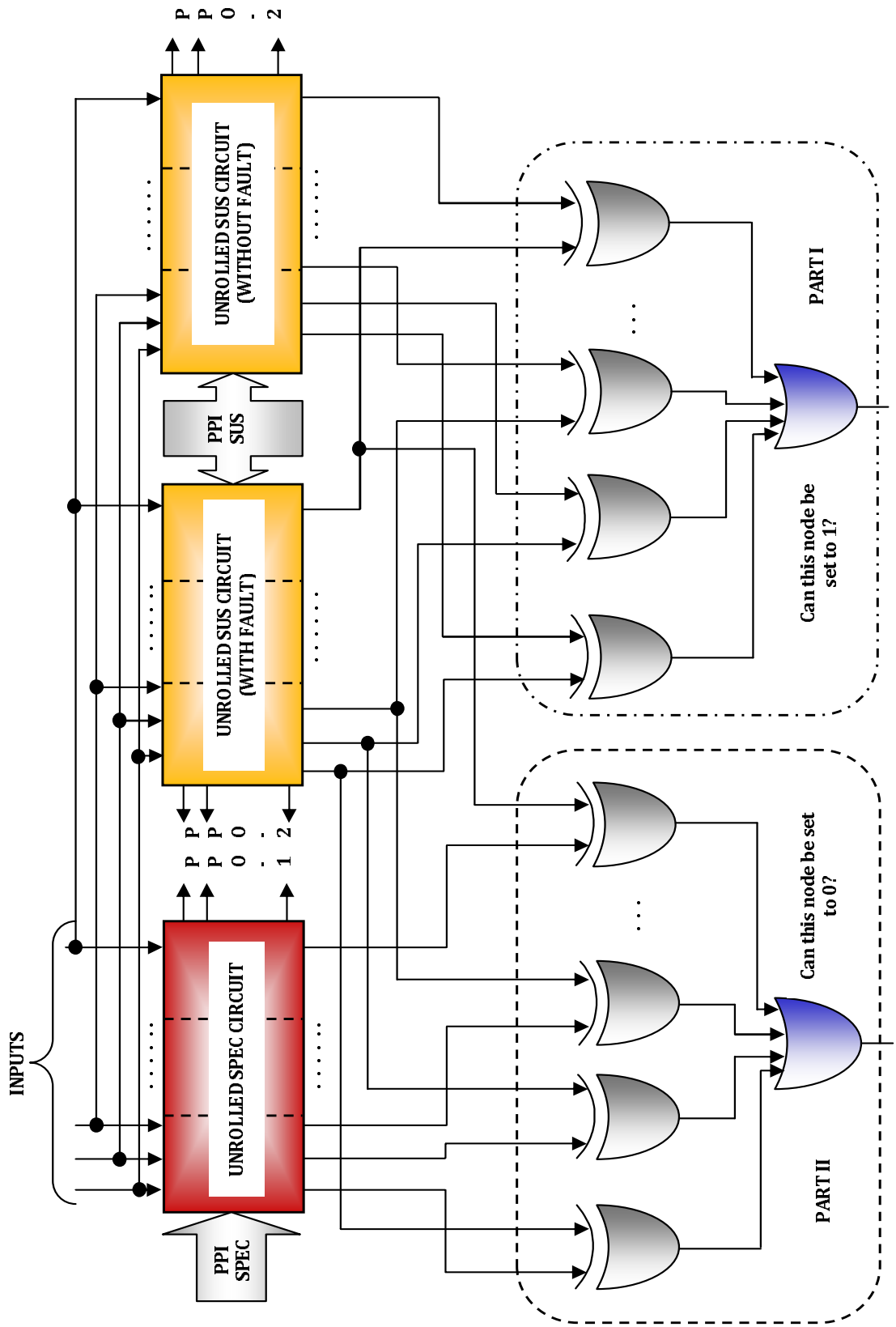


Figure 5.1: Miter circuit set up for performing diluted Sequential Equivalence Checking

outputs (PPO-2) are not considered for comparison because we want to ensure that the selected signal specifically affects the output.

Three advantages of this setup are - (1) faults untestable over multiple frames get filtered out; (2) successive states appearing at the internal state boundaries are progressively filtered by circuit logic and hence tend to converge towards reachable state space; (3) faults which are excitable but not propagable to the output within the given unroll depth are also discarded thereby further pruning the *suspect candidate* list. Although a completely unconstrained initial state will eventually allow a portion of the unreachable state space to penetrate the successive states, this does not discard any functionally possible state. This is to note that generally it is not difficult to come up with an initial state along with an input sequence that can activate and propagate the fault on S to a primary output. However, this is not our ultimate goal. We want to use the vector sequence thus obtained to constrain our search.

First, let's consider the k-time frame unrolled version of the *spec/sus circuit* shown in Fig. 5.2. Without loss of generality, let the circuits have corresponding initial states (which are assumed to be reachable) represented by S_0 and S_0' respectively. Starting with S_0', say, we activate and observe the hard to detect stuck-at fault on signal S in the *sus circuit* using the vector sequence $< V_x >$ where $1 \leq x \leq k$. Thus if S is indeed a Trojan signal, its effect will be absent in the *spec circuit* and hence the outputs in the k^{th} time frame between *spec* and *sus circuits* will differ for the same input sequence.

Now, we shall move to PART-II of this diluted SEC setup. We use the line of thought explained above. We unroll the *spec circuit* the same number of time frames as that of the *sus circuit* (shown as N in Fig. 5.1 PART-II). We constrain the outputs of the *spec circuit* to have the same value as that in the *sus circuit* by forcing the output of the miter in PART-II to 0 (OR-2). We simultaneously constrain the primary inputs ($INPUTS$) to a sequence that activated and observed signal S in *sus circuit* (obtained from the miter set-up in PART-I). Thus, we are effectively checking if the *spec circuit* can produce the same behavior as the *sus circuit* when signal S has been activated and observed in the sus circuit (within the given

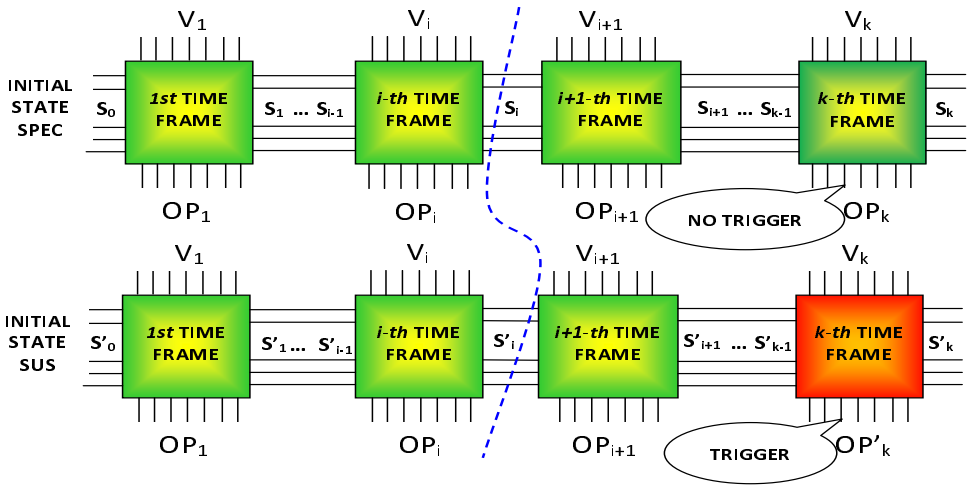


Figure 5.2: Concept of triggering the Trojan in unrolled circuit

unroll bound).

This combined 2-part set-up is converted into a CNF instance and given to the SAT solver. If the solver returns a SAT result, we cannot conclude about the signal S because we don't know the reachability of the state in the initial time frame in the *spec circuit*. Optimistically, it is most likely not a malicious signal and just an infrequently activated signal. However, if it is UNSAT, it can be due to two reasons. Either it is due to the effect of the Trojan and conditions produced by it in *sus circuit* that makes it trigger (which is absent in the *spec circuit*) or due to the effect of an illegal state at any time frame in *sus circuit* for which no corresponding state exists in the *spec circuit*. But with unrolling the state illegality factor diminishes and so it is more likely that an UNSAT solution arises due to the effect of the Trojan. Our results show that for the cases where the SAT solver could not solve the instance, majority of the signals belonged to the Trojan.

5.2.4 Infected Region Isolation

The final step in our approach aims to isolate the region(s) infected by the Trojan. We start with the *suspect candidate* list obtained from the N-detect ATPG. In general, the *suspect*

candidate list contains multiple faults associated with some gates. For example, there can be a two-input logic gate G for which several stuck-at faults around G are present in the list. We refer to each gate in the *suspect candidate* list as G_i, where i is the gate ID. The weight of each gate W_i corresponds to the frequency of occurrence of G_i in the *suspect candidate* list, each associated with a different stuck-at fault on that gate. We then expand the *region* around each of these *suspect candidates*. Since the number of such gates are small, the number of regions are also small. We progressively go on increasing the *radius* of each region. If there are n gates in the *region* centered around gate i within a radius of x (denoted by $R_i(x)$), the *suspect count* SC_i of the *region* is computed by:

$$SC_i(x) = \sum_{1}^{n} W_i | G_i \in R_i(x) \tag{5.1}$$

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

$$SI_i(x) = SC_i(x)/|R_i(x)| \tag{5.2}$$

where $|R_i(x)|$ denotes the total number of gates within *region* centered around gate i and x is the *radius* of the *region* around gate i.

The rationale behind using the *suspect index* as a diagnosis parameter is as follows. As we progressively go on expanding a *region*, the number of gates in that region will also increase. Ultimately every region will have sufficiently large radii to contain all the gates in the *suspect candidate* list. But our aim is to pin-point a very small fraction of the gates that are susceptible as *Trojan(s)*. For this, we use the *suspect index* as the parameter of interest, which represents a weighted value of each gate in a *region* to be associated with the Trojan. For the *region* not containing the Trojan, with a growing radius the number of gates in it increases without adding to the *suspect count* thereby reducing the *suspect index*.

We select the gates from the suspect candidate list obtained in N-Detect ATPG (step-II) and not after the SSG-EC (step-III). This is because SSG-EC tends to remove a large number

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

Let's consider again the circuit fragment in Fig. 5.3. The shaded gates are the ones associated with the Trojan. After processing the faultlist through steps I and II, let the *suspected candidate* list contain $\{G1, G2, G3, G5, G6, G9\}$. The weight distribution for these gates are as follows: $W_1 = 2, W_2 = 3, W_3 = 3, W_5 = 1, W_6 = 2$ and $W_9 = 1$. All other gates which are not in the *suspect candidate* list receive a weight of 0. Noticeably $W_4 = 0$ (belongs to Trojan) and $W_9 = 1$ (doesn't belong to the Trojan). To explain the significance of *suspect index*, let's observe the effect of expanding the region around two of these candidates viz. $G3$ and $G6$. A *region* around gate $G3$ with *radius* of 1 unit gives a suspect index of $SI_{G3} = (W_1+W_2+W_3+W_5+W_6)/5 = 2.2$. The corresponding suspect index centered around gate $G6$ with the same *radius* of 1 is given by $SI_{G6} = (W_3 + W_6 + W_7 + W_8 + W_9)/5 = 1.2$. Based on these values, the *region* around $G3$ is more likely to contain the Trojan. Now, if we increase the *radius* around gate $G3$ to 2, it will add gates $G7, G8, G10$ along with $DFF1$ and $BUFFER$ without increasing the *suspect count* thereby reducing the *suspect index* to 1.1. Thus, the *radius* parameter needs to be balanced to get a maximum overlap with Trojan region. Our experimental results show a remarkable match between the gates contained in the region with highest *suspect index* and those in the implanted Trojan.

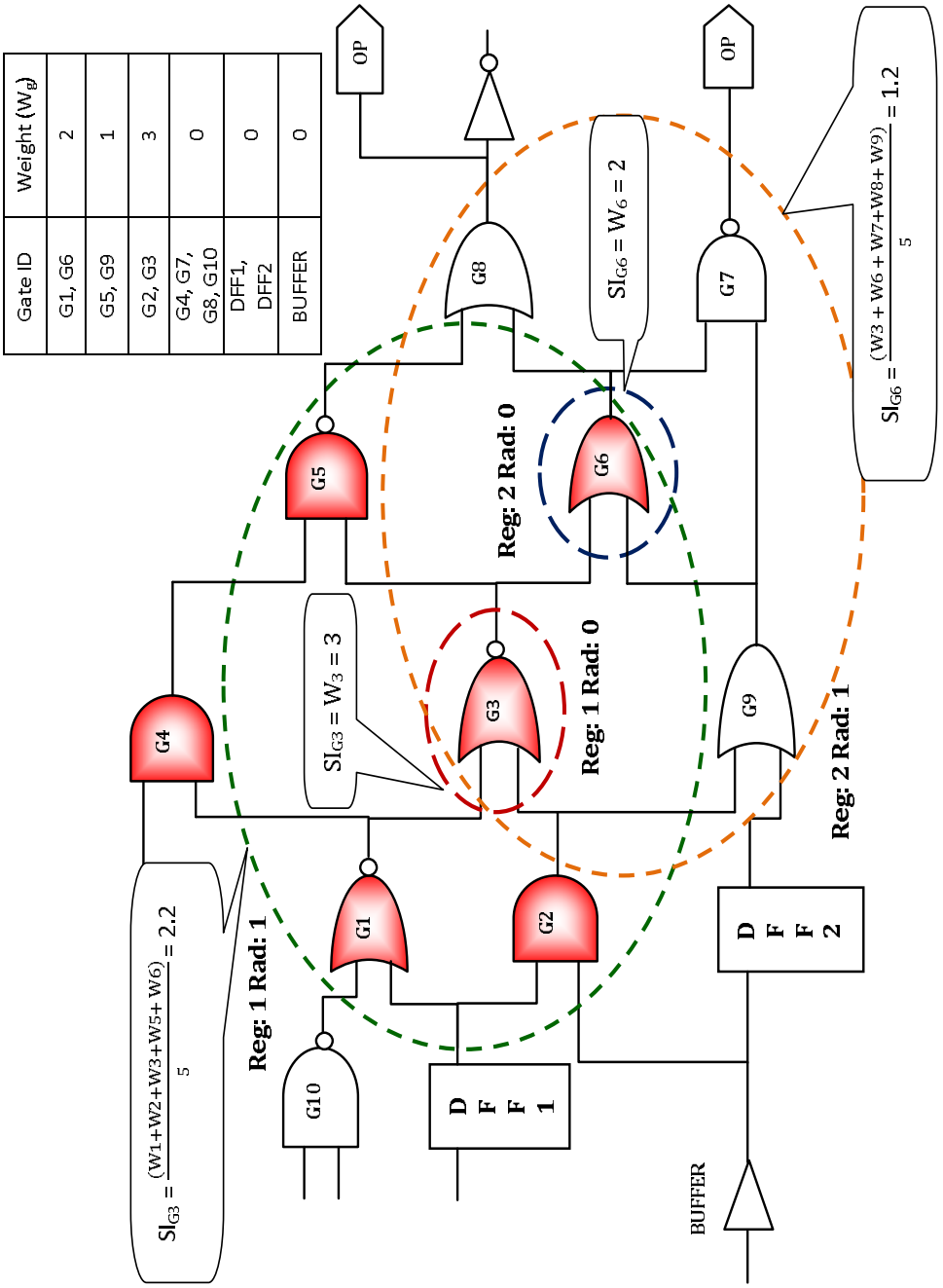


Figure 5.3: Region isolation based on gate weights

Table 5.1: Circuit Parameters for *spec* and *sus* circuits for ISCAS'89 and ITC'99 benchmarks (with the Trojan inserted)

Attr	IP/OP	Total Gate Count		Flip-Flops	
Ckt	Spec/Sus	Spec	Sus (Tro)	Spec	Sus (Tro)
s298	3/6	142	230(41)	14	38(10)
s344	9/11	195	247(37)	15	31(10)
s349	9/11	196	253(36)	15	31(10)
s526	3/6	223	380(41)	21	39(10)
s641	35/24	457	346(37)	19	27(10)
s713	35/24	470	359(36)	23	27(10)
s1196	14/14	575	792(37)	18	28(10)
s1238	14/14	554	867(38)	18	28(10)
s1423	17/5	753	726(42)	74	86(10)
s3330	40/73	2034	1687(36)	132	83(10)
s5378	35/49	3042	3120(39)	179	213(10)
b03	6/4	316	344(35)	29	40(10)
b04	13/8	1013	2087(38)	69	80(10)
b05	3/36	1101	2374(38)	32	44(10)
b08	11/4	280	316(37)	21	33(10)
b09	3/1	290	490(38)	28	40(10)
b10	13/6	364	362(36)	17	34(10)
b11	10/7	1390	1436(36)	35	50(10)
b12	6/6	3245	3108(38)	199	230(10)
b13	4/10	281	320(37)	29	44(10)

5.3 Malicious 3PIP Setup

We have used ISCAS'89 and ITC'99 sequential benchmark circuits for our experiments. The sequentially equivalent versions of these circuits were generated via circuit re-synthesis. The original circuit is the *spec circuit*. The circuit resulting after re-synthesis is injected with a Trojan, called as the *sus circuit*. The *sus circuit* is checked against the *spec circuit* with a set of 100,000 random vectors to ensure that the Trojan is not detected at the output, ensuring it is stealthy. The circuit statistics after inserting the Trojan is given in Table 5.1. The two versions of ITC'99 circuits correspond to a *gray encoding* and *one-hot encoding* of the corresponding states.

For each circuit, IP/OP represents the number of inputs and outputs in the circuit. Under the *Total Gate Count* column we report the total number of gates and flip-flops in *spec* and *sus circuits*. This includes the entire Trojan present in the *sus circuit*. The number of gates in the Trojan is given in *(Tro)* under *sus* column. The column under *Flip-Flops* shows the number of state elements in *spec* and *sus circuits*. The counts in *(Tro)* under *sus* column of *Flip-Flops* represents the total number of flip-flops in the Trojan portion. Our experimental Trojan consists of a 10-bit counter (sequential Trojan). To make it stealthier, we control the ticking of the counter by a specific state which occurs, say 1 in 10,000.

Let's consider a k bit counter which is ticked by a rare state whose probability of occurrence is 1 in S (where S is very large number). To compute the probability that the Trojan is triggered in a simulation run of N cycles (where $N \gg 2^k$), we use the following formula:

$$P_{trigger} = \binom{N}{2^k}(1/S)^{2^k}(1-1/S)^{N-2^k} \tag{5.3}$$

So even by the random simulation the probability of the Trojan getting triggered is 1 in 10,240,000 assuming that the Trojan triggers when all the counter bits are set to 1. To complicate the detection in ISCAS'89 circuits, we have incorporated conditions to propagate the effect to any observation points after the Trojan has been triggered.

5.4 Experimental Results

Our experimental results are summarized in Table 5.2. The first column represents the ISCAS'89 and ITC'99 benchmarks used in our experiments. The total number of collapsed stuck-at-faults in the *sus circuit* for each benchmark is given in column two. Column three (Step-I) represents the total number of faults detected using the functional vectors, with the remaining undetected faults in parentheses. Sustained random vectors are particularly helpful in increasing the fault coverage for circuits like $s1423$, in which 1329 of the original 1600 collapsed stuck-at faults in the *sus circuit* were detected by functional vectors. Only the

Table 5.2: Trojan detection result for ISCAS'89 and ITC'99 benchmarks using our 4-step methodology

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

remaining 271 faults are then passed to Step II. Column four gives the fault count which are uniquely detected less than $N = 3$ times using the full-scan N-detect ATPG. For the same s1423 circuit, only 46 out of 271 faults were identified as *suspect candidates* by this step. In the fifth column we report the percentage of faults associated with the Trojan portion of the circuit that are contained in the list represented by column four. For example, if the list produced by step II contains 10 faults out of which 3 belong to the Trojan portion, this figure is 30%. For s1423, this is 82.6%. Next, the result for PART-I of step-III is tabulated under column seven. Here, we achieve further reduction in the size of the *suspect candidate* list because of the factors mentioned earlier: (1) non-propagation to output and (2) sequential untestability. This represents a very small fraction of total faultlist. For s5378, as small as $132/4826 \times 100 = 2.7\%$ of the total faults. Columns seven through ten report the result of the diluted SEC. The *SAT* columns contain the number of faults for which the SAT solver returned a solution, the *UNSAT* columns contain the faults which for which the SAT solver could not solve the instance. These columns are further partitioned to represent how many signals belong to the Trojan portion and how many belong to the genuine circuit.

For s1196 the majority of faults classified as UNSAT belonged to the Trojan. For most of the other circuits (except for $s3330$) where most of the instances return a SAT solution, Trojan nets constitute the majority of the filtered faults. In $s5378$, although the number of Trojan-related nets is small, the implications are significant. This is because when compared to the Trojan size, the number of signals that have been removed is quite substantial. Column eleven shows the result of step-IV in our approach. The results have been tabulated for a *radius* of 2. For most of the ISCAS'89 circuits (except for $s349$) and more than half of the ITC'99 circuits, all the gates in the *region* that produced highest *suspect index* belonged to the Trojan, i.e., all gates in the region are signals associated with the Trojan!

For the circuits like $b04$, $b05$, $b12$ and $b13$ in which our methodology failed to sieve out the region of tamper we found out that these circuits contain numerous untestable *regions* and the one getting the highest *suspect index* is not necessarily the one that contains major portion of the Trojan nets. On further analysis, we came to the conclusion that because of

the unconstrained initial state, step-II (N-Detect ATPG) sometimes remove major portion of the Trojan related nets from the *suspect candidate* list. To verify our hypothesis, we repeated the same experiments on the same circuits, but this time skipping the N-Detect ATPG stage. The results are tabulated in Table. 5.3.

From Table 5.3, it is clear that we have got a marked improvement in filtering out the *suspicious regions* as compared with the 4-step methodology discussed earlier. Along with that a lot of signals related to the Trojan setup create an unsatisfiable *triple miter* instance as discussed in 5.2.3. This is because functional ATPG cannot detect the Trojan nets, so they remain in the *suspect candidate* list and are accounted for in the *diluted SEC* step. On the flip side, here we need to analyze more signals than in the earlier setup.

As a supporting experiment, we wanted to verify that the Trojan instances we built are indeed stealthy enough, we performed two different analysis. In the first experiment, we did a functional simulation of both the *spec circuit* and the *sus circuit* starting from a know initial state. We wanted to check that the Trojan effect is not visible at the output very easily. In the second experiment, we did a bounded model checking (BMC) on the *spec circuit* and the *sus circuit* at different unroll depths.

The results are summarized in Table 5.4. The first column shows the circuit names. For both random functional simulation as well as for BMC we had a timeout limit of 3 hours. The second column shows the number of random vectors simulated on the instances in 3 rous. Columns 3 through 7 show the result of BMC on the same circuits with different unroll bounds.

None of the instances were detectable by functional simulation for 3 hours. In BMC almost all of the instances (except for $s298$) were shown to be unsatisfiable or timeout. For $s298$, we found out that the original circuits have a discrepancy and they are not equivalent to begin with (even when the Trojan is absent in the *sus circuit*). For the rest, experimental evidences stands to the testimony for the fact that the Trojans used are not easily detectable.

Table 5.3: Trojan detection result for ISCAS'89 and ITC'99 benchmarks using our 3-step methodology (step-II skipped)

Ckt.	# Flts Sus Ckt.	Step-I Func. vec (# Und. Flts.)	Step-III (I) Sel. for SSG-EC	Step-III (II) SAT		Step-III (II) UNSAT		Step-IV Rad-2 % SI_{max}
				Tro	*Non-Tro*	*Tro*	*Non-Tro*	
s298	502	340(162)	118	107	11	0	0	**100**
s344	507	353(154)	128	74	4	**46**	4	**100**
s349	517	361(156)	117	60	4	**49**	4	**100**
s526	728	512(216)	147	115	23	0	9	**100**
s641	633	427(206)	176	111	47	**10**	8	**100**
s713	656	433(223)	178	120	50	0	8	**100**
s1196	1414	1259(155)	125	72	0	**48**	5	**100**
s1238	1557	1332(225)	117	60	1	**50**	6	**100**
s1423	1600	1329(271)	150	113	37	0	0	**100**
s3330	2878	2106(772)	309	103	182	0	24	**100**
s5378	4826	3496(1330)	919	79	840	**45**	79	**100**
b03	628	351(277)	65	50	15	0	0	**92.8**
b04	3308	2581(727)	268	99	168	0	1	**100**
b05	3762	1197(2565)	2064	70	1813	**37**	144	**100**
b08	576	371(205)	129	112	11	0	0	**100**
b09	846	2(844)	377	103	266	1	7	**100**
b10	672	467(205)	84	50	31	0	3	**92.8**
b11	2154	1169(985)	60	50	10	0	0	**92.8**
b12	4558	1287(3271)	282	50	228	0	4	**92.8**
b13	622	380(242)	83	50	30	0	3	**92.8**

Table 5.4: Trojan detection result for ISCAS'89 and ITC'99 benchmarks using functional vector simulation and BMC

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

TO = Timeout. Timeout limit for both functional simulation as well as BMC is 3 hours.

5.5 Summary

In this work, we proposed a novel methodology to detect and isolate *Trojans* in pre-fabricated 3PIPs. The first two stages involving functional vectors and full-scan N-detect ATPG remove a large number of signals from consideration, leaving only a very small subset of suspect signal candidates. The *diluted SEC* and *region isolation* steps help to pinpoint the actual gates associated with the Trojan. Experimental results on ISCAS'89 benchmarks show that the methodology is very effective in distinguishing a tampered 3PIP from a genuine one, able to pin-point the region containing the stealthy Trojan. We performed two distinct set of experiments to isolate the Trojan affected regions in the circuits, one of which skipped step-II. Also, we did some post analysis to ensure that the Trojans used are not easily detectable either by functional simulation or by formal methods. Future directions will involve the use of implication-based clauses in the unrolled circuits to further prune out illegal states. This will speed up the SAT solver performance. In addition, it is possible that many Trojan related signals which appear as SAT in stage III (Part II) are rendered UNSAT.

Chapter 6

Sustained Vector Technique

In this chapter, we propose a sustained vector methodology that magnifies the power consumption differences between the actual and the Trojan circuitry to a value which is much higher than the process variation. In certain cases the power differential can vary by more than an order of magnitude. Each vector is repeated multiple times to both the genuine and the Trojan-embedded circuits that ensures the reduction of toggles within the genuine circuit. This is needed so that the power dissipation outside of the Trojan will not drown out the extra power from the Trojan. In addition, we propose a scheme to suggest the locations susceptible to Trojan implantations. Regions showing wide variations in the power behavior are analyzed to isolate the infected gate(s). Our method is generalized for detecting Trojans that may be connected to any gate(s) in the circuit. There is no pre-silicon on-chip processing requirement which means that the methodology has no silicon overhead.

6.1 Motivation

One of the inherent difficulties that makes a Trojan difficult to detect is its dormancy. In normal operational conditions it is very difficult to create the scenario that triggers the

Trojan. So at most one can expect to trigger only a portion of the Trojan at a time. The impact of such a small extraneous activity is really small in terms of the total power consumed by the circuit during normal operation. So we need some strategy to drastically reduce the original circuit activity so that the incremental activity in the triggered portion of the Trojan can be highlighted.

6.2 Approach

This methodology is applicable to static CMOS circuits. There are two steps in our approach. The first step aims to detect the presence of a Trojan while the second tries to isolate the region within the circuit that may contain it. We call the first step as *Toggle Minimization* and the second as *Infected Region Isolation* respectively.

6.2.1 Step 1: Toggle Minimization

In the context of earlier discussion, power consumed in a circuit depends on the amount of signal switchings for any given vector pair. Since Trojans are minuscule circuits relative to the entire circuit, it is intuitive that the power consumed by the Trojan will also be very small. To observe the extra power that is contributed by the Trojan circuit over the genuine circuit, it is essential that the overall power consumption in the genuine circuit should be minimized. This would highlight the power contributed from the Trojan circuit which is the key to detect its presence. In the process, we also need to ensure that there is at least some kind of activity going on inside the circuit. In other words, the circuit should not be allowed to enter some *sleep mode*, which may also make the Trojan dormant.

Circuit activity within the combinational frame of the circuit is induced in two ways: (1) with the changing inputs and (2) with the changing state. While primary inputs are fully controllable, the state variables are not. In order to limit the switching activity within the

circuit, we can restrict the input variations to an extent such that the state variables are the only factor for inducing toggles. This is achievable by sustaining the same vector at the input pins over multiple clock cycles. Statistically in a purely random scenario each new vector will have at least half of the input bits toggled from the previous vector. These toggles will propagate through the transitive fanout cones of the respective inputs to generate further toggles in the circuit. If we ensure not to create any toggles at the input itself, it helps us reduce the circuit activity to a good extent because in such situation the state bits are the only factor for generating activity in the circuit. Moreover, we prefer scenarios where fewer state bits change as we keep the input vector at a stable value. Additionally, it also helps in reducing the synergistic transitions. That is, there are gates in the circuit which derives its inputs from the state-bits as well as from the inputs and they transition when more than a single input changes. If the changing state is the only dynamic variable during the operational mode, chances are less that such synergistic transitions will occur as compared with the random scenarios. Naturally these help in minimizing the overall circuit activity and keep the power consumption of the overall circuit low, which is our primary objective. We note that without sustaining a vector, the power consumption generally is much higher, closer to consuming an average power level of the circuit. The concept of *Toggle Minimization* by sustaining a vector is shown in Figures 6.1 (a) and (b).

For a pair of vectors $\{V_i, V_{i+1}\}$ applied to the circuit C at clock cycles i and $i+1$ respectively, the total number of gates undergoing toggle is dependent on two factors

1. $\triangle V = V_{i+1} - V_i$

2. $\triangle S = S_{i+1} - S_i$

where S_i and S_{i+1} represent the state of the circuit i^{th} and $i+1^{th}$ clock cycles respectively. If we denote the total activity generated in circuit C at the $(i+1)^{th}$ clock cycle by A^C_{i+1} then

$$A^C_{i+1} \propto \triangle V + \triangle S + \triangle V \dot{\triangle} S \tag{6.1}$$

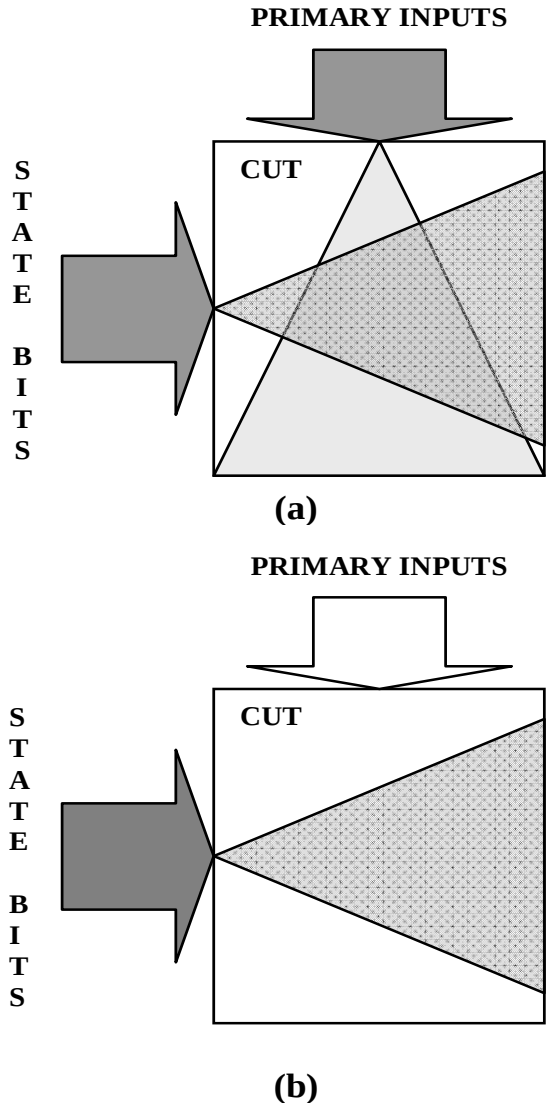


Figure 6.1: Concept of activity minimization. In (a) circuit activity is created by both flip-flops and PIs whereas in (b) only by flip-flops

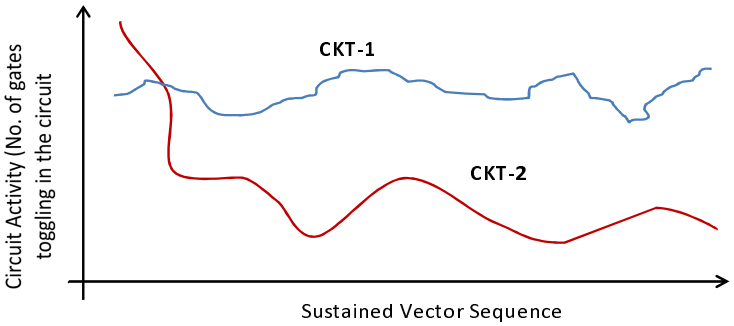


Figure 6.2: Conceptual representation of circuit activity behavior under sustained vector simulation for CKT-1 and CKT-2

When we sustain the vectors at the input $\triangle V = 0$ and $\triangle S$ is the only factor that generates activity in the circuit. This will eliminate the activity generated by $\triangle V$ as well as the activity generated by the combined effect of $\triangle V$ and $\triangle S$ thereby reducing the total circuit activity.

Our experiments on different circuits subjected to sustained vector simulation led to the following conclusion. In some circuits, the activity remains fairly constant at a high value irrespective of the input value (sustained or non-sustained); while in some other circuits, sustaining a vector over multiple clock cycles results in a continuous decrease in the circuit activity. This observation has been conceptually shown in Fig. 6.2 where $CKT-1$ belongs to the first class and $CKT-2$ belongs to the second. This graph is not generated from any real simulation but is based on what we observed from our experiments on different circuits. The approach we develop here is very effective for the second class of circuits, as will be shown in the experimental results; although it works for the first class in some cases as well.

In our experiments, we generated a set of 1000 random input vectors, each of which is sustained to a maximum of 25 cycles. For a vector V, after sustaining it k times ($k < 25$), if we find that the system has reached a stable state where no further change in the state variables occurs, we move on to the next vector. Thus, a vector set for any particular circuit contains a maximum of 25000 input sequences. Since we need to exercise all portions of the

circuit, we change the input vectors to a different value after a period of sustenance which helps the circuit acquire a different state and explore some other regions of the state space. We apply this test sequence to both the genuine and the Trojan circuits in our experiments and obtain the *differential power* numbers (expressed in %) between them. The resultant plot is the *Differential Power Profile Plot* for the CUT.

6.2.2 Step 2: Infected Region Isolation

Let G represent the set of gates in a circuit C

$$G = \bigcup g \mid g\, is\, a\, gate\, in\, circuit\, C \tag{6.2}$$

Let G^T represent the gates in the *genuine* circuit that are used to control the Trojan. Upon simulation of a sustained vector V for n clock cycles we see appreciable power differences at specific time instances. If W_g denote the weight of a gate $g \epsilon G$ being associated with the Trojan then

$$\forall g \epsilon G \;\; W_g = W_g + \sum_{i=1}^{i=n} 1 \mid there\, is\, an\, appreciable\, power\, dif\, ference\, at\, instance\, i \tag{6.3}$$

The difference in the activity between the *genuine* and the Trojan circuits appears whenever activity is induced inside the Trojan portion by the gates in G^T. For all such instances, the counter $W_g \mid g \epsilon G^T$ is incremented. This results in a higher weight assignment to the gates in G^T

We use the *Differential Power Profile Plot* information from Step 1 to identify the region(s) of the circuit that are likely to be insertion points of the Trojan. We focus on vector pairs that produced high *differential power* as starting points.

Let us consider a sustained vector V_1 with which the CUTs shows a noticeable difference in the power profile in simulation cycles t and $t+1$. This is shown in Figure 6.3. Let g_1

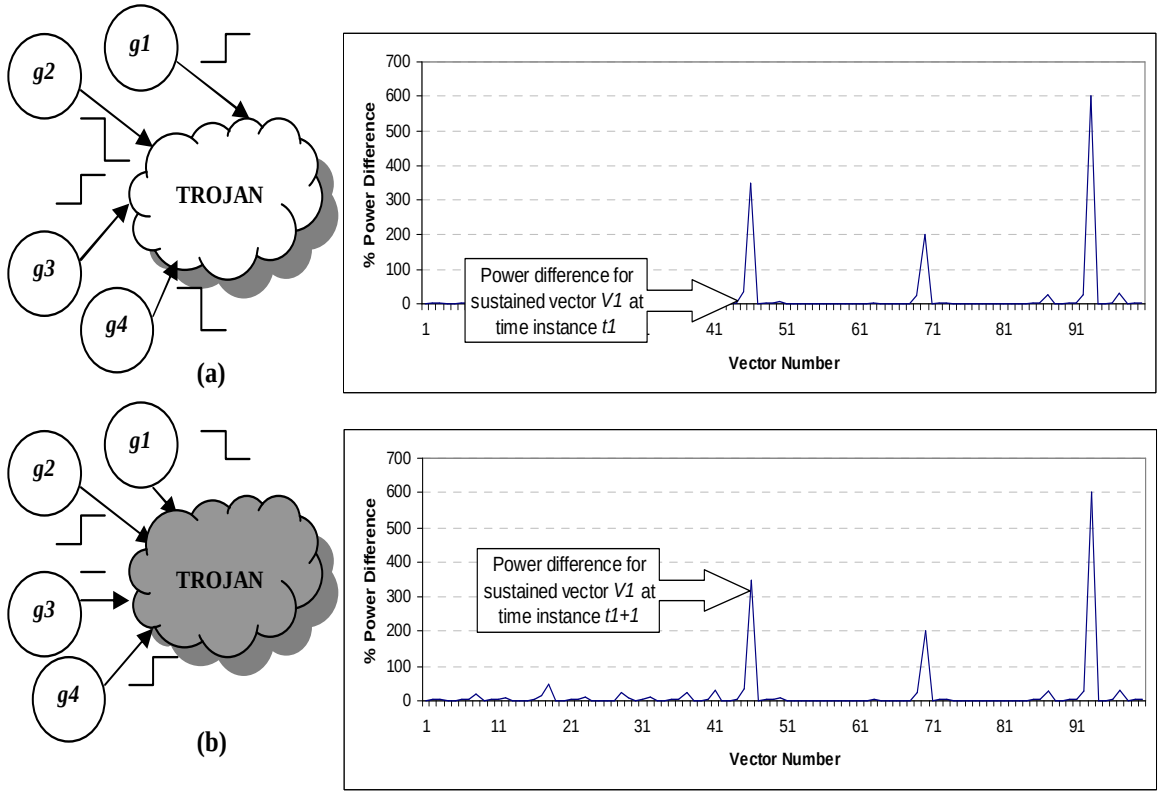


Figure 6.3: Power differential measurement between vectors to isolate the gates connected to the Trojan

Algorithm 2 Isolate gate(s) accountable for Trojan activity

Require: *GenuineCkt*, *TrojanCkt*, *InputVector*

Ensure: Plot of *gate weights* corresponding to their probability of *Trojan association*

1: $PowerDifferentialThreshold \Leftarrow 5.0$
2: $TrojanCount \Leftarrow 0$
3: $NonTrojanCount \Leftarrow 0$
4: $ToggledGateList \Leftarrow Simulate(GenuineCkt, InputVector)$
5: $PowerNumbers(GenuineCkt) = PowerSimulate(GenuineCkt, InputVector)$
6: $PowerNumbers(TrojanCkt) = PowerSimulate(TrojanCkt, (InputVector))$
7: **for all** $(V_i, V_{i+1}) \epsilon InputVector$ **do**
8: $\%PowerDifferential \Leftarrow$ $(\textbf{abs}\frac{(PowerNumbers(TrojanCkt,(V_i,V_{i+1}) - PowerNumbers(GenuineCkt,(Vi,V_{i+1}))))}{PowerNumbers((GenuineCkt),(V_i,V_{i+1}))} * 100)$
9: **if** $\%PowerDifferential > PowerDifferentialThreshold$ **then**
10: $IncrementWeight(TrojanCount, ToggledGateList(V_i, Vi + 1))$
11: **else**
12: $IncrementWeight(NonTrojanCount, ToggledGateList(V_i, Vi + 1)$
13: **end if**
14: **end for**
15: $BuildPowerProfilePlots(PowerNumbers(GenuineCkt), PowerNumbers(TrojanCkt))$
16: **for all** $g_i \epsilon GenuineCkt$ **do**
17: $GateWeight = \frac{TrojanCount(g_i)}{NonTrojanCount(g_i}$
18: **end for**

to g_4 be the four internal gates in the circuit that are actually connected to the Trojan. Since the Trojan derives its inputs from the gates in its transitive fanin cone, any activity produced in the Trojan implies activity in its inputs. From the lower portion of the figure it shows that as the gates g_1, g_2, g_3 and g_4 underwent transition from 1010 to 0111, there is an observable difference in the *differential power* in the circuit. Thus a transition in gates g_1, g_2 and g_4 gives rise to a significant increase in the differential power ratio between the Trojan circuit and the genuine circuit, marking these gates as a potential suspect for the Trojan implantation.

We keep two counters for each gate: *TrojanCount* and *NonTrojanCount*. Each time a gate toggles by a vector pair that shows *differential power* greater than the *PowerDifferentialThreshold* (which is set to 5%, a typical value of process variation [12]), its *TrojanCount* is incremented and vice versa. After this analysis is over we compute the ratio *Trojan-*

Table 6.1: Functions for the algorithm of Sustained Vector technique

Function	**Purpose**
Simulate(*Ckt*, *Vector*)	Simulate *Vector* set on given *Ckt*
PowerSimulate(*Ckt*, *Vector*)	Simulate *vector* set on given *Ckt* to compute the power numbers
BuildPowerProfilePlots(PowerNumbers (*Ckt1*), PowerNumbers(*Ckt2*))	Plot the differential power between *Ckt1* and *Ckt2*
IncrementWeight (*CountArray*, ToggledGateList(V_i, V_{i+1})	Increment weights of all gates that toggled between V_i and V_{i+1} in *CountArray*

Count/NonTrojanCount which is called as *gate weight*. A high value of the *gate weight* indicates that the gate(s) are most likely to be associated with the Trojan.

The entire procedure is outlined in Algorithm 2 and the functions used in the algorithm are explained in Table 6.1. *PowerDifferentialThreshold* is the maximum differential power accounted for process variation. The two counters *TrojanCount* and *NonTrojanCount* are as described above. Evidently, a highly active gate which is not associated with the Trojan will most likely toggle between vectors when power differential is above the threshold. In that case its *TrojanCount* will increase as per our algorithm. To compensate for such spurious increments, we keep the *NonTrojanCount* which increments the count of a gate which toggles when the *Differential Power* is below *PowerDifferentialThreshold*. Whenever the number of points above the threshold is less than the points below the threshold for a given gate, the ratio of these two parameters would turn out to be small, thereby filtering out highly active signals that may not be associated with the Trojan.

6.3 Experimental Results

In this section we shall detail the process of our experimentation and discuss the observed results. The first section describes about the attributes of the Trojan and the considerations

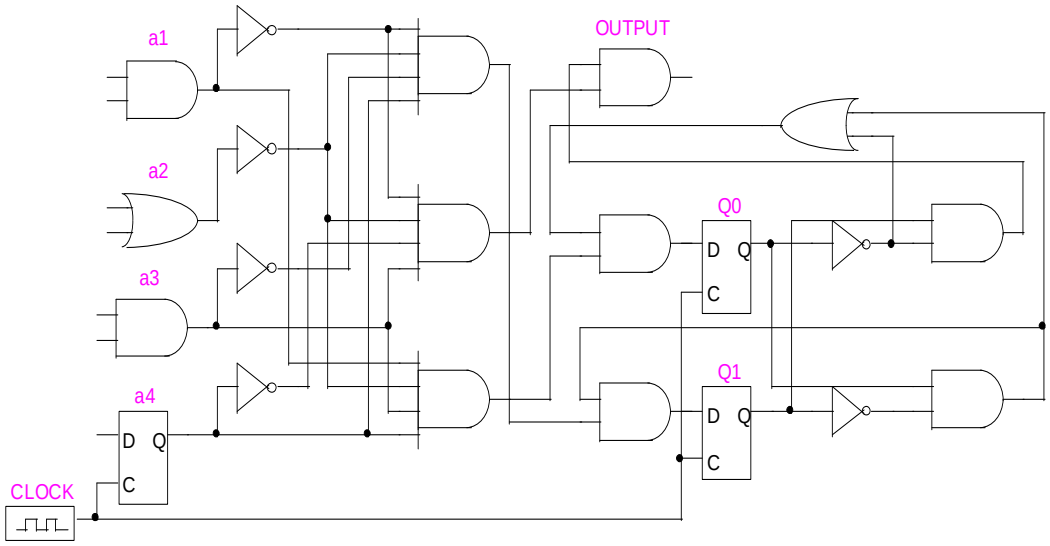


Figure 6.4: Example of a Trojan circuit

that were taken into account for constructing them. The second section represents the results obtained by applying our technique on the Trojan setup.

6.3.1 Trojan Description

Typical Trojan structures used in our experimental setup is shown in Figure 6.4, is less than 1% of the original circuit for large circuits. For small circuits, we keep it less than 3%. The Trojan inputs as well as the Trojan output is connected to internal nets. Table 6.2 shows the sizes of the Trojans in terms of percentage gate counts used in our experiments.

Table 6.2: Trojan Size (% of total gate count)

Circuit	**High-activity**	**Medium-activity**	**Low-activity**
s1196	2.61	2.96	2.61
s5378	0.66	0.56	0.56
s9234	0.29	0.32	0.36
s15850	0.19	0.20	0.18
s38584	0.09	0.09	0.10
b14	0.19	0.18	0.18

We also insert the Trojan to different locations in the original circuit. We classify the gates in the circuit into three different classes depending on their activity. To compute the activity of a gate in the circuit we apply a sequence of 10000 non-sustained random vectors to the circuit and then compute the toggle count of each gate. Each signal is classified as High-activity, Medium-activity or Low-activity. After classifying the signals, we embed Trojans with the groups identified. The inputs to a Trojan can come from any of the three categories. Table 6.3 shows the average percentage activity at the inputs of the Trojans for our setup.

In order to make sure that the Trojan is really stealthy we choose a triggering sequence which is very rarely occurring within the set of signals chosen. To make it undetectable we connect the Trojan output(s) to an internal gate for which the non-triggered value at the output(s) of the Trojan is non-controlling for the gate to which it is input. For instance, if the Trojan produces an output of 0 under normal operating conditions, this net is connected to an OR/NOR gate so that it does not produce any effect unless the Trojan is triggered. Further, in Step 1 of the algorithm, we check that even when the Trojan is triggered, the effect does not reach any primary output, which otherwise would be a functionally detectable Trojan. This is possible because any controlling value in the propagation path of the gate affected by the Trojan output would mask its effect.

Table 6.3: Average % activity of gates associated with Trojan

Circuit	**High-activity**	**Medium-activity**	**Low-activity**
s1196	40.4	18.8	5.3
s5378	48.2	37.4	10.6
s9234	49.7	25.5	11.6
s15850	37.7	27.7	9.9
s38584	41.8	31.7	0.4
b14	49.2	35.8	25.8

6.3.2 Trojan Detection Results

The results are presented in two parts corresponding to the two steps in our methodology. Table 6.4 shows the maximum *percentage power differential* between the genuine and Trojan circuits obtained from both random vectors and vectors generated by our approach. We embedded Trojans three different ways as discussed before: Trojans monitoring high-activity signals, medium-activity signals, and low-activity signals, respectively. *Rnd* and *Ours* used in the table stand for *Random Approach* and *Our Approach* respectively. It is evident that for nearly all cases our approach enhances the *percentage differential power*.

For example, for both s1196 and s5378, more than an order of magnitude improvement was achieved using our approach when compared with random vectors. The enhanced power differential helps us to isolate the region where the Trojan may be embedded, which is discussed next.

Table 6.4: Comparison of % Power Differential between Genuine and Trojan Circuits achieved by Random and Our Approach

Trojan Type	High-activity		Medium-activity		Low-activity	
Circuit	Rnd	Ours	Rnd	Ours	Rnd	Ours
s1196	10.71	**300**	27.08	**650**	18.75	**600**
s5378	4.18	**133.33**	2.24	**18.18**	2.79	**7.46**
s9234	30.77	**50**	30.77	**50**	22.22	**66.66**
s15850	13.04	**38.7**	**10.07**	9.46	3.28	**5.68**
s38584	0.74	**4.62**	0.8	**1.66**	0.68	**6.52**
b14	10.34	**10.29**	1.1	**1.26**	1.18	**1.55**

The results of the second part of our analysis is shown in Figures 6.5, 6.6, 6.7,6.8 and 6.9. The *x-axis* represents the gate numbers and the *y-axis* represents the computed*gate weight*. Recall that relative *GateWeight* is a direct indication of the probability that the gate is connected to the Trojan.In larger circuits, gates have higher count value in the *NonTrojanCount* because *PowerDifferentialThreshold* is exceeded less frequently during the application of the sustained vectors. As a result, the fraction *GateWeight* generally decreases as the circuit size increases. Nevertheless, it is the relative weights that count. As we can

see from Figure 6.9, only a fraction of Total gates of the circuit is actually assigned a weight which means that the algorithm sieves out most of the gates from the larger circuits. The dotted circle in the figures correspond to the gates that are indeed connected to the Trojan in the actual circuit. As evident, in most cases the Trojan gates weigh out to a high value relative to other gates.

For circuits like $b14$ our approach didn't work as effectively. This is because $b12$ is a highly active circuit and so it is not possible to minimize the overall circuit activity using a *sustained* vector simulation. This is evident from the fact that the lowest average activity for gates in $b12$ is 25% which is much more higher than the rest of the circuits. Nevertheless, for *high activity* Trojans it was still distinguished appreciably. For the cases where our method could not make a distinction, we concluded the following two reasons for such behavior:

1. **Varying** W_i: Not all signals connected to the Trojan undergo transition when the Trojan is switching. As an example, for signals g_1, g_2, g_3 and g_4 connected to the Trojan if the triggering sequence is 1010 followed by 1111, then corresponding to every such sequence occurring during simulation g_2 and g_4 will have higher count in *TrojanCount* than g_1 and g_3. So during *GateWeight* computation, g_2 and g_4 will get more weight than g_1 and g_3, and this effect will show up in the *GateWeights* plot.

2. **Reverse transitions in the Trojan FSM**: The FSM nature of the Trojan may be such that the internal state of the Trojan may change even if the inputs to the Trojan are stable. In such a case, if the overall circuit activity happens to be low, the *differential power* between the CUTs will be highlighted.

   Let's consider a Trojan FSM represented by $S_0 \leftrightarrow S_1 \leftrightarrow S_2$, where S_0 is the initial state of the Trojan and S_2 is the triggering state. This transition is triggered by a specific set of values appearing on the internal circuit nodes in succession. Of course, these transitions are brought about by the toggling of the Trojan gates. If at any internal state (e.g. S_1), the next sequence is not the appropriate triggering sequence then the FSM returns back to the initial state S_0. This induces some activity in the

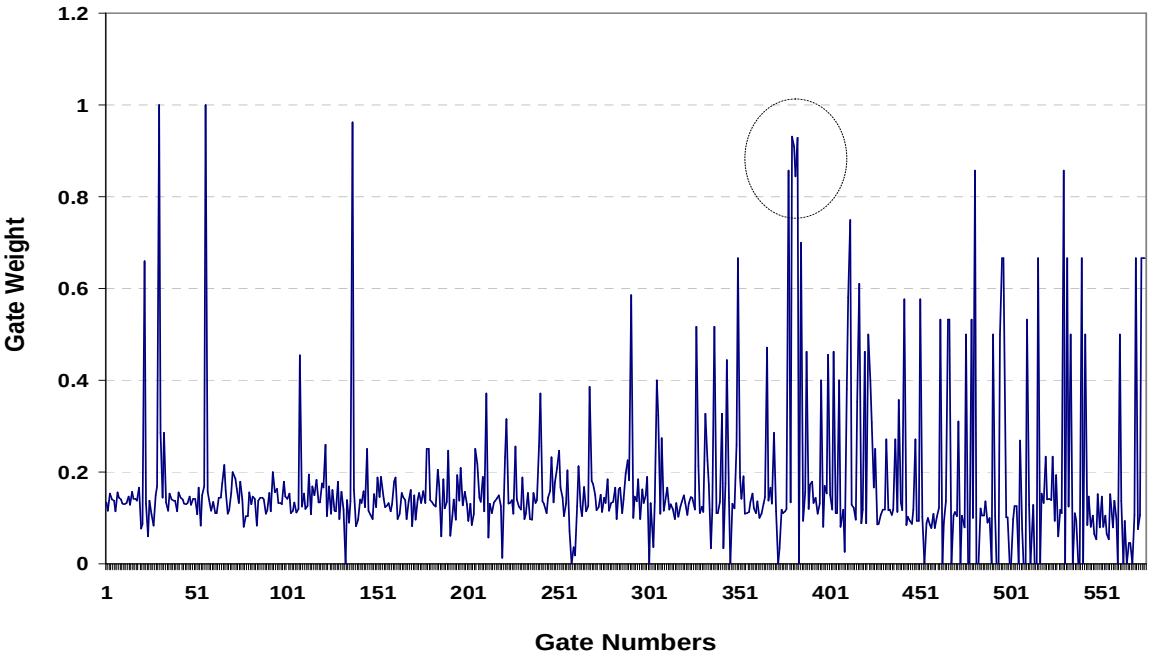


Figure 6.5: Plot of Gate Weights for s1196 Medium-active Trojan

Trojan portion of the circuit which can be pronounced if the overall circuit activity is low. During this transition most of the gates that toggle are unrelated to the Trojan and these gates get an incremented weight.

We note that for nearly all the test cases, our approach was able to identify the signals responsible for the Trojan, and the anomalies mentioned above occur infrequently.

As discussed above, our experiments were conducted on ISCAS'89 benchmarks which are not as large as industrial designs. For industrial designs, since they contain millions of gates, it might not be feasible to contain the activity to a certain minimum level. For such SOC designs, we can use the module *ENABLE* signals to selectively pick out the portion to test. In Fig. 6.10, we have 6 different modules, each with its own *ENABLE* signal. While testing any one of them we should keep the other ENABLES to low value thereby making the rest of the circuit non-functional. This ensures minimal power consumption for the portion of the circuits which is not under test.

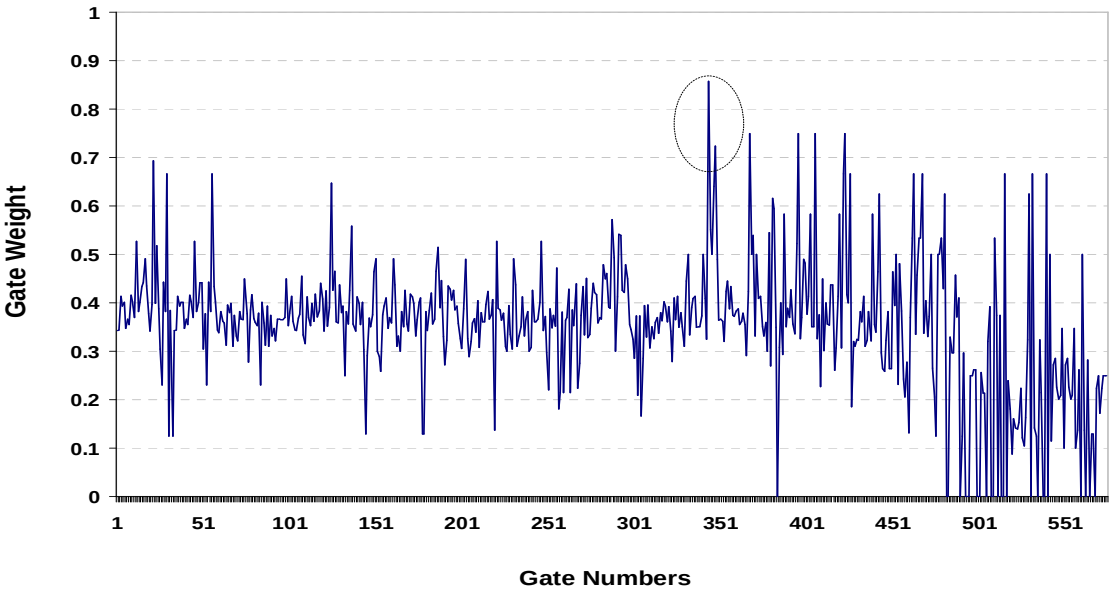


Figure 6.6: Plot of Gate Weights for s1196 High-active Trojan

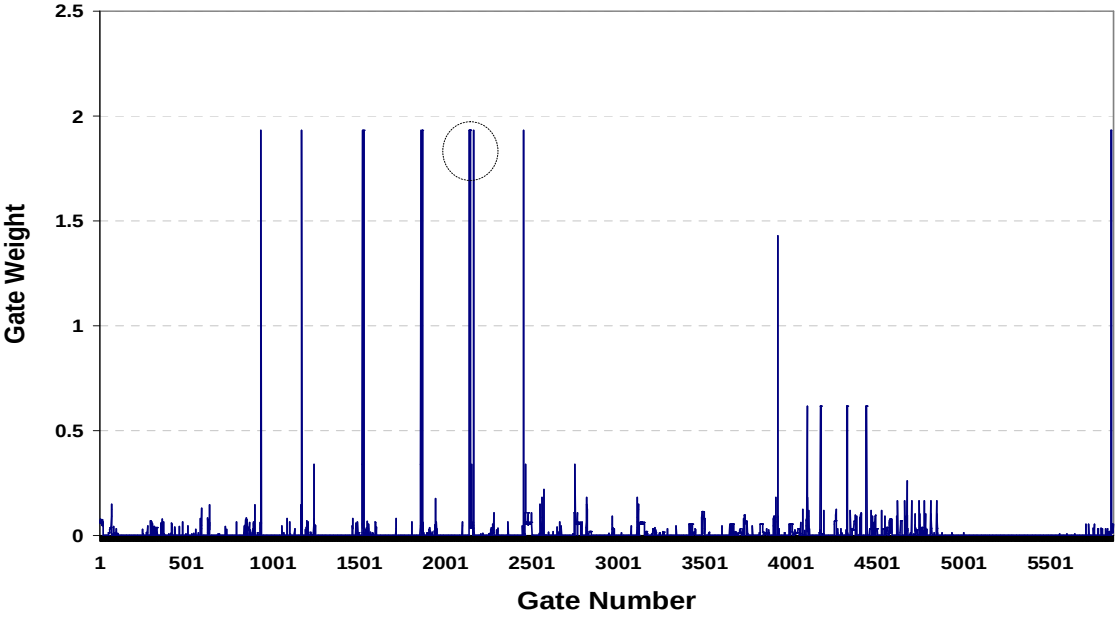


Figure 6.7: Plot of Gate Weights for s9234 Low-active Trojan

6.4 Summary

We have presented a novel sustained vector methodology that is very effective in detecting the presence of a Trojan. Even if the Trojan constitutes only a tiny fraction of the chip area, our

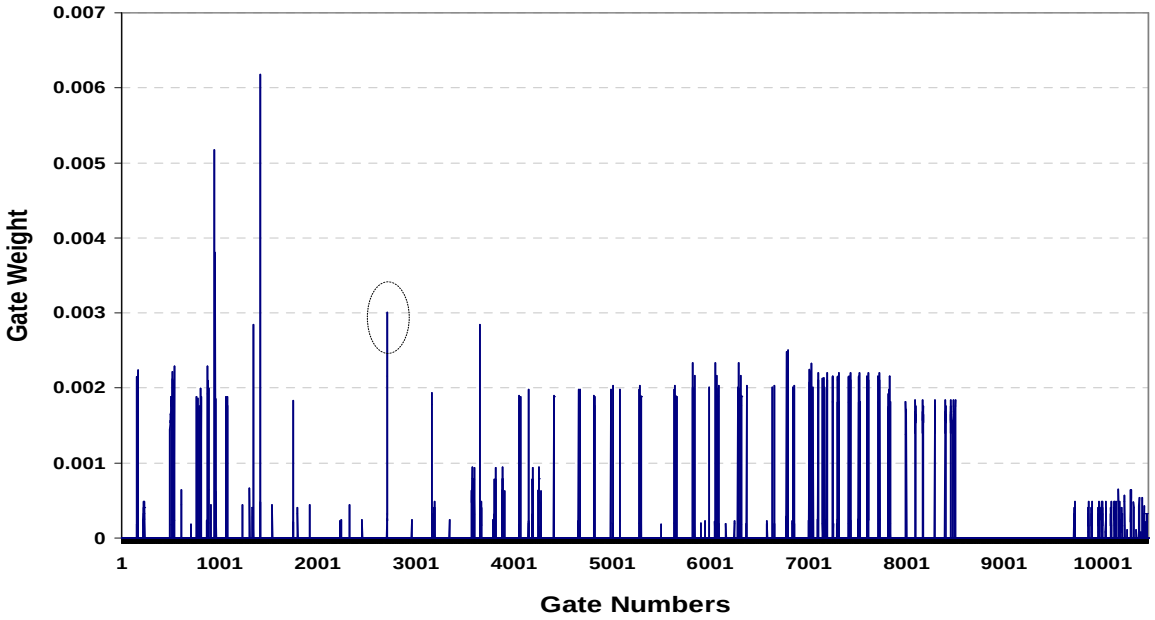


Figure 6.8: Plot of Gate Weights for s15850 Low-active Trojan

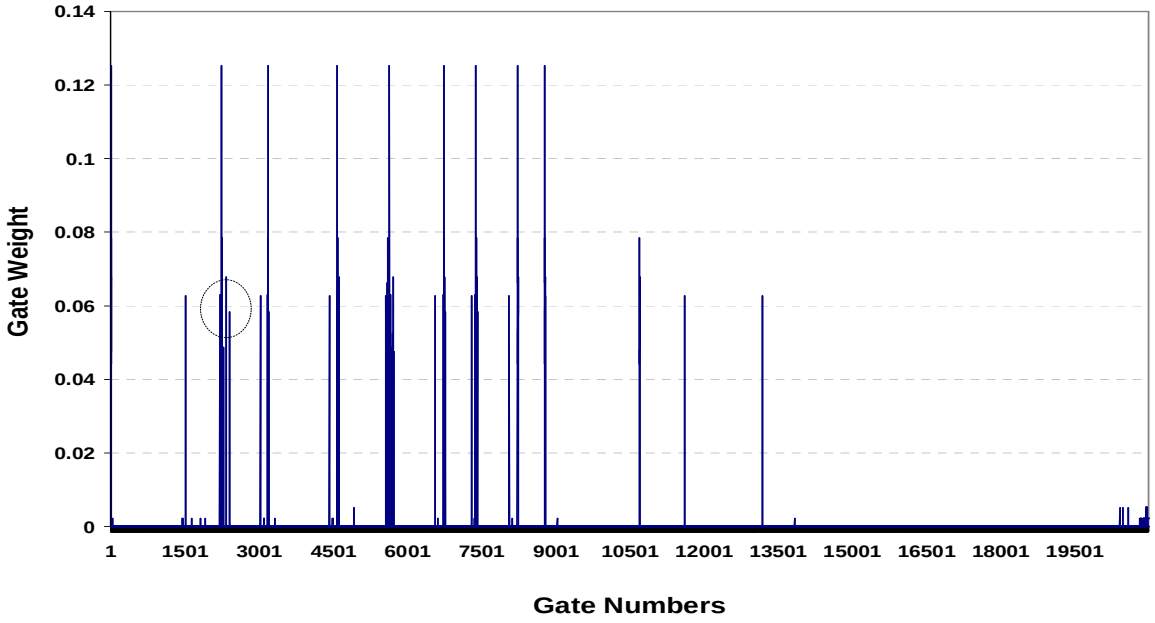


Figure 6.9: Plot of Gate Weights for s38584 High-active Trojan

experimental results show that this technique enhances the power profile difference between the genuine and Trojan circuits by up to more than one order of magnitude as compared with the random vectors. Furthermore, this method is effective irrespective of the activity

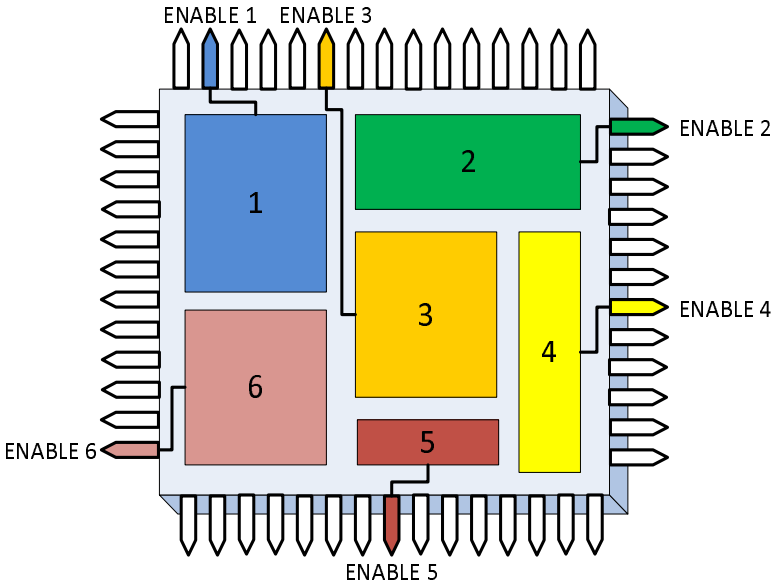


Figure 6.10: Use of module ENABLEs to selectively activate portions of an SOC

behavior of the gates which is evident from the fact that it is successful in detecting most of the High-activity, Medium-activity and Low-activity Trojans in the benchmark circuits. Finally, in many cases, our approach was able to pinpoint the actual location of the Trojan in the circuit.

Chapter 7

Conclusion

In this dissertation we analyzed potential threats that arise due to the involvement of a third party manufacturer in design or fabrication process of an IC. With growing complexity and diverse applicability of today's ICs, it is becoming increasingly difficult to contain the entire design cycle within trusted entities. Since third party transactions are based on mutual trust and faith, we need to implement an additional level of cross examination before the parts are deployed for actual field operation. This is to make sure that the original intent of the design has not been tampered with Trojans that will make the design work contrary to the expected in future, consequences of which can be calamitous. We have addressed this issue at various levels in the IC design flow. Some of the methods strengthen the existing test techniques to specifically target malicious insertions while others are aimed to uncover the Trojans, if present, using non-destructive approaches.

In chapter 3, we devised a technique to improve the overall testability of the design with a substantial speed-up in test application time. We use both Q and $\bar{Q}$ signals of a DFF to enlarge the state space of the DUT. These additional states are effective in activating and propagating fault effects to the output which were otherwise hard to detect in functional mode. We proposed a heuristic based approach to partition the DFFs into different groups and thereafter selectively choose a group to use $\bar{Q}$ instead of Q to enhance the variation in

state space. Lastly, we implemented test point insertion scheme to further improve the fault coverage.

In chapter 4, we proposed a *Design for Trust* methodology that is very effective in pronouncing the effect of Trojans implanted in any circuit. Using the fact that changing the supply voltage of a static CMOS gates inverts its behavior, we are able to detect the presence of Trojans at the primary outputs. This is because under voltage inverted condition the circuit logic changes and so the Trojan which was originally built to be rarely active is no longer stealthy. Since threshold drop builds up across circuit stages, we consider inverting voltage at alternate levels thereby replenishing the drop incurred on one stage at the other. Additionally, power profiles of Trojan infected and original designs show substantial difference under voltage inversion scheme, which are otherwise not distinguishable under normal operation.

In chapter 5, we proposed a methodology for detecting the Trojans in 3PIPs. Here we use the rare activation condition of Trojan related signals in the suspected design obtained from the third party along with a sequential equivalence checking framework to filter out nodes related to Trojan logic. Under the condition that the Trojan related signals are activated and propagated to output, we have shown that optimized version of a design can differ with the original design to an extent that cam make the resultant miter unsatisfiable, thereby indicating that the Trojan behavior is not replicable in the original circuit. Finally a region based filtering approach is used to sieve out the Trojan related gates.

In chapter 6, we proposed a post silicon non-destructive methodology based on power difference of the Trojan infected and genuine ICs. Circuit activity is generated in two different ways, from inputs and from state bits, out of which we can minimize the activity generated by the inputs by holding them at a constant value for several clock cycles. This results in pronounced Trojan effect and based on the gates that toggle internally we are able to isolate the Trojan implantation locations.

Future works on this area is possible in the following directions:

1. An analysis of different types of Trojans that can be detected by existing techniques and a categorization telling which techniques are suited for what types of Trojans.

2. Devising more non-destructive test mechanisms that can further strengthen the existing Trojan detection techniques.

3. Combination of two or more existing techniques to find a synergistic improvement in Trojan detection approaches.

4. Novel *Design For Trust* mechanisms for improved testability against Trojan implantations.

5. Techniques directed for other stages in the IC design flow which are not considered in this work. This is important because increasing trend of outsourcing has raised the issue of trust at every stage in the design where a third party is involved.

Bibliography

[1] S. Adee, *The hunt for the kill switch*, IEEE Spectrum, Vol. 45, Issue 5, 2008, pp 34-39.

[2] D. Agarwal, B. Archambeault, J. R. Rao and P. Rohatgi, *The EM side-channel(s)*, Workshop on Cryptographic Hardware and Embedded Systems, Vol. 2523, 2002, Lecture Notes in Computer Science, pp. 29-45.

[3] S. Ahuja, D. A. Mathaikutty, A. Lakshminarayana and S. K. Shukla, *SCoPE: Statistical Regression Based Power Models for Co-processors Power Estimation*, Journal of Low Power Electronics, Vol. 5, Issue 4, 2009, pp. 407-415.

[4] S. Ahuja, D. A. Mathaikutty, A. Lakshminarayana and S. K. Shukla, *Accurate Power Estimation of Hardware Co-Processors using System Level Simulation*, IEEE International SOC Conference, 2009, pp. 399-402.

[5] S. Ahuja, D. A. Mathaikutty, G. Singh, J. Stetzer, S. K. Shukla and A. Dingankar, *Power estimation methodology for a high-level synthesis framework*, International Symposium on Quality Electronic Design, 2009, pp. 541-546.

[6] S. Ahuja, D. A. Mathaikutty and S. K. Shukla, *Applying Verification Collaterals for Accurate Power Estimation*, International Workshop on Microprocessor Test and Verification, 2008, pp. 61-66.

[7] S. Ahuja, D. A. Mathaikutty, S. K. Shukla and A. Dingankar, *Assertion-Based Modal Power Estimation*, International Workshop on Microprocessor Test and Verification, 2007, pp.3-7.

[8] S. Ahuja, W. Zhang, A. Lakshminarayana and S. K. Shukla, *A Methodology for Power Aware High-Level Synthesis of Co-Processors from Software Algorithms*, International Conference on VLSI Design, 2010, pp. 282-287.

[9] S. Ahuja and S. K. Shukla, *MCBCG: Model Checking Based Sequential Clock-Gating*, IEEE International workshop on High Level Design Validation and Test, 2009, pp. 20-25.

[10] S. Almukhaizim and O. Sinanoglu, *Peak power reduction through dynamic partitioning of scan chains*, IEEE International Test Conference, 2008, pp 1-10.

[11] M. Ashouei, A. Chatterjee, A. Singh, *Test volume reduction via flip-flop compatibility analysis for balanced parallel scan*, IEEE International Workshop on Current and Defect Based Testing, 2004, pp.105-109.

[12] D. Agarwal, S. Baktir, D. Karakoyunlu, P. Rohatgi and B. Sunar, *Trojan detection using IC fingerprinting*, IEEE Symposium on Security and Privacy, 2007, pp. 296-310.

[13] M. Banga, M. Chandrasekar, L. Fang and M. Hsiao, *Guided test generation for isolation and detection of embedded Trojans in ICs*, ACM Great Lake Symposium on Very Large Scale Integration, 2008, pp. 363-366.

[14] M. Banga and M. Hsiao, *A region based approach for the detection of hardware Trojans*, IEEE International Workshop on Hardware Oriented Security and Trust, 2008, pp. 43-50.

[15] M. Banga and M. Hsiao, *A Novel Sustained Vector Technique for the Detection of Hardware Trojans*, IEEE International Conference on VLSI Design, 2009, pp. 327-332.

[16] M. Banga and M. Hsiao; *VITAMIN: Voltage inversion technique to ascertain malicious insertions in ICs*, IEEE International Workshop on Hardware Oriented Security and Trust, 2009, pp. 104-107.

[17] K. S. Brace, R. L. Rudell, and R. E. Bryant, *Efficient implementation of a BDD package*, IEEE Design Automation Conference, 1990, pp. 40-45.

[18] D. Brand, *Verification of large synthesized designs*, IEEE International Conference on Computer Aided Design, 1993, pp. 534-537.

[19] R. E. Bryant, *Graph-based algorithms for boolean function manipulation*, IEEE Transaction on Computers, Vol. 35, 1986, pp. 677-691.

[20] W. T. Cheng, M. Sharma, T. Rinderknecht and C. Hill, *Signature based diagnosis for logic BIST*, IEEE International Test Conference, 2006, Oct. 2006, pp. 1-9.

[21] K. T. Cheng, *Redundancy removal for sequential circuits without reset states*, IEEE Transactions on Computer Aided Design, Vol. 12, 1993, pp. 13-24.

[22] V. Chickermane and J. H. Patel, *An optimization based approach to the partial scan design problem*, IEEE International Test Conference, 1990, pp. 377-386.

[23] K. Y. Cho, S. Mitra, E. J. McCluskey, *California scan architecture for high quality and low power testing*, IEEE International Test Conference, 2007, pp. 1-10.

[24] D. R. Collins, *TRUST, A proposed plan for trusted integrated circuits*, Government Microcircuit Applications and Critical Technology Conference, 2006, pp. 276-277.

[25] C. A. J. van Eijk, *Sequential equivalence checking without state-space traversal*, IEEE Design, Automation and Test in Europe, 1998, pp. 618-623.

[26] C. Fagot, O. Gascuel, P. Girard and C. Landrault, *On calculating efficient LFSR seeds for built-in self test*, IEEE European Test Workshop, 1999, pp 7-14.

[27] B. Gassend, D. Clarke, M. van Dijk and S. Devadas, *Controlled physical random functions*, IEEE Computer Security Applications Conference, 2002, pp. 149-160.

[28] A. Giani, S. Sheng, M. Hsiao and V. Agrawal, *Efficient spectral techniques for sequential ATPG*, IEEE Design, Automation and Testing in Europe, 2001, pp. 204-208.

[29] P. Goel, *An implicit enumeration algorithm to generate tests for combinational logic circuits*, IEEE Transactions on Computers, Vol. C-30, 1981, pp. 215-222.

[30] E. I. Goldberg, M. R. Prasad, and R. K. Brayton, *Using SAT for combinational equivalence checking*, IEEE Design Automation and Test in Europe, 2001, pp. 114-121.

[31] E. Goldberg and Y. Novikov, *BerkMin: a fast and robust SAT-solver*, IEEE Design Automation and Test in Europe, 2002, pp. 142-149.

[32] L. H. Goldstein and E. L. Thigpen, *SCOAP: Sandia controllability/observability analysis program*, IEEE Design Automation Conference, 1979, pp 190-196.

[33] J. Guajardo, S. S. Kumar, G. J. Schrijen and P. Tuyls, *Physical unclonable functions and public-key crypto for FPGA IP protection*, IEEE International Conference on Field Programmable Logic and Applications, 2007, pp. 189-195.

[34] R. Guo, S. Reddy and I. Pomeranz, *PROPTEST: A property based test pattern generator for sequential circuits using test compression*, IEEE Design Automation Conference, 1999, pp. 653-659.

[35] H. Hashempour, F. J. Meyer and F. Lombardi, *Test time reduction in a manufacturing environment by combining BIST and ATE* IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2002, pp 186-194.

[36] I. Hamzaoglu and J. .H. Patel, *Reducing test application time for full scan embedded cores*, IEEE Annual International Symposium on Fault Tolerant Computing, 1999, pp. 260-267.

[37] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan and J. Rajski, *Logic BIST for large industrial designs: real issues and case studies*, IEEE International Test Conference, 1999, pp. 358-367.

[38] M. S. Hsiao, E. M. Rudnick and J. H. Patel, *Sequential circuit test generation using dynamic state traversal*, IEEE Design, Automation and Testing in Europe, 1997, pp. 22-28.

[39] M. S. Hsiao and M. Banga, *Kiss the scan goodbye: A non-scan architecture for high coverage, low test data volume and low test application time*, IEEE Asian Test Symposium, 2009, pp. 225-230.

[40] F. F. Hsu, K. M. Butler and J. H. Patel, *A case study on the implementation of the Illinois Scan Architecture*, International Test Conference, 2001, pp. 538-547.

[41] S. Y. Huang, K. T. Cheng, and K. C. Chen, *Verifying sequential equivalence using ATPG techniques*, ACM Transactions on Design Automation of Electronic Systems, Vol. 6, Issue 2, 2001, pp. 244-275.

[42] Y. Jin and Y. Markis, *Hardware Trojan detection using path delay fingerprint*, IEEE International Workshop on Hardware Oriented Security and Trust, 2008, pp. 54-60.

[43] C. H. Kim and J. J. Quisquater, *How can we overcome both side channel analysis and fault attacks on RSA-CRT?*, IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography, 2007, pp. 21-29.

[44] P. C. Kocher, J. Jaffe and B. Jun, *Differential power analysis*, Proceedings of CRYPTO, Vol. 1666, 1999, Lecture Notes in Computer Science, pp. 388-397.

[45] P. C. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Proceedings of CRYPTO, Vol. 1109, 1996, Lecture Notes in Computer Science, pp. 104-113.

[46] L. J. Kohout, A. Yasinsac and E. McDuffie, *Activity profiles for intrusion detection*, IEEE Fuzzy Information Processing Society, 2002, pp. 463-468.

[47] J. Kumagai, *Chip detectives*, IEEE Spectrum, Vol. 37, Issue 11, 2000, pp. 43-48.

[48] W. Kunz and D. K. Pradhan, *Recursive learning: A new implication technique for efficient solutions to CAD problems - test, verification and optimization*, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 13, 1994, pp. 1149-1158.

[49] K J. Lee, J J. Chen and C H. Huang, *Using a single input to support multiple scan chains*, IEEE International Conference on Computer Aided Design, 1988, pp. 74-78.

[50] J. Li and J. Lach, *At-speed delay characterization for IC authentication and Trojan Horse detection*, IEEE International Workshop on Hardware Oriented Security and Trust, 2008, pp. 8-14.

[51] X. Lin, I. Pomeranz and S. M. Reddy, *MIX: a test generation system for synchronous sequential circuits*, IEEE International Conference on VLSI Design, 1998, pp. 456-463.

[52] F. Lu, L. C. Wang, K. T. Cheng and R. C. Y. Huang, *A circuit SAT solver with signal correlation guided learning*,IEEE Design, Automation and Test in Europe, 2003, pp. 892-897.

[53] F. Lu, L. C. Wang, K. T. Cheng, J. Moondanos and Z. Hanna, *A signal correlation guided ATPG solver and its applications for solving difficult industrial cases*, IEEE Design Automation Conference, 2003, pp. 436-441.

[54] F. Lu and K. T. Cheng, *Sequential equivalence checking based on k-th invariants and circuit SAT solving*, IEEE High Level Design Validation and Test Workshop, 2005, pp. 45-51.

[55] A. Maiti, R. Nagesh, A. Reddy and P. Schaumont, *Physical unclonable function and true random number generator: a compact and scalable implementation*, IEEE/ACM Great Lakes Symposium on VLSI, 2009, pp. 425-428.

[56] A. Maiti and P. Schaumont, *Improving the quality of a Physical Unclonable Function using configurable Ring Oscillators*, IEEE International Conference on Field Programmable Logic and Applications, 2009, pp. 703-707.

[57] S. Malik, A. R. Wang, R. K. Brayton and A. Sangiovanni-Vincentelli, *Logic verification using Binary Decision Diagrams in a logic synthesis environment*, IEEE International Conference on Computer Aided Design, 1988, pp. 6-9.

[58] I. H. Moon and C. Pixley, *Non-miter based combinational equivalence checking byc-Comparing BDDs with different variable orders*, Formal Methods in Computer Aided Design, Vol. 3312, Lecture Notes in Computer Science, 2004, pp. 144-158.

[59] G. E. Moore, *Cramming more components onto integrated circuits*, Electronics Magazine, Vol. 38, Number 8, April 1965.

[60] S. Morozov, A. Maiti and P. Schaumont, *An Analysis of Delay Based PUF Implementations on FPGA*, International Symposium on Applied Reconfigurable Computing, Lecture Notes in Computer Science, 2010, pp. 382-387.

[61] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, *Chaff: Engineering an efficient SAT-solver*, IEEE Design Automation Conference, 2001, pp. 530-535.

[62] S. Narendra, V. De, S. Borkar, D. Antoniadis, A. Chandrakasan, *Full-chip sub-threshold leakage power prediction model for sub-0.18nm CMOS*, IEEE International Symposium on Low Power Electronics and Design, 2002, pp. 19-23.

[63] L. Nechman, K. K. Saluja, S. Upadhyaya and R. Reuse, *Random pattern testing for sequential circuits revisited*, IEEE Annual International Symposium on Fault Tolerant Computing 1996, pp. 44-52.

[64] T. Niermann and J. H. Patel, *HITEC: a test generation package for sequential circuits*, IEEE Design, Automation and Testing in Europe, 1991, pp. 214-218.

[65] Y. Novikov, *Local search for boolean relations on the basis of unit propagation*, IEEE Design, Automation and Test in Europe, 2003, pp. 810-815.

[66] K. Nowaka, G. Carpenter, F. Gebara, J. Schaub, D. Agarwal, P. Rohatgi, W. E. Hall, S. Baktir, D. Karakoyunlu and B. Sunar; *IC Fingerprinting and Stable IC Sensors for Enhanced IC Trust*,Government Microcircuit Applications and Critical Technology Conference, March 2007.

[67] S. Pilli and S. Sapatnekar, *Power estimation considering statistical IC parametric variations*, IEEE International Symposium on Circuits and Systems, 1997, Vol. 3, pp. 1524-1527.

[68] C. Pixley, *A theory and implementation of sequential hardware equivalence*, IEEE Transactions on Computer Aided Design, Vol. 11, 1992, pp. 1469-1494.

[69] D. Rai and J. Lach, *Performance of delay based Trojan detection techniques under parameter variations*, IEEE International Workshop on Hardware Oriented Security and Trust, 2009, pp 58-65.

[70] R. Rad, J. Plusquellic and M. Tehranipoor, *Sensitivity analysis to hardware Trojans using power supply transient signals*, IEEE International Workshop on Hardware Oriented Security and Trust, 2008, pp. 3-7.

[71] S. Remersaro, X. Lin, Z. Zhang, S. M. Reddy, I. Pomeranz and J. Rajski, *Preferred fill: a scalable method to reduce capture power for scan-based designs*, IEEE International Test Conference, 2006, pp 1-10.

[72] X. Wang, H. Salmani, M. Tehranipoor and J. Plusquellic, *Hardware Trojan detection and isolation using current integration and localized current analysis*, IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, 2008, pp. 87-95.

[73] V. Singhal, C. Pixley, A. Aziz, and R. K. Brayton, *Theory of Safe Replacements for Sequential Circuits*, IEEE Transcations on Computer Aided Design of Integrated Circuits and Systems, Vol. 20, 2001, pp. 249-265.

[74] J. P. Marques-Silva and T. Glass, *Combinational equivalence checking using satisfiability and recursive learning*, IEEE Design, Automation and Test in Europe, 1999, pp. 145-149.

[75] J. P. Marques-Silva and K. A. Sakallah, *GRASP: A search algorithm for propositional satisfiability*, IEEE Transactions on Computers, Vol. 48, 1999, pp. 506-521.

[76] M. Syal and M. Hsiao, *VERISEC: Verifying equivalence of sequential circuits using SAT*, IEEE High-Level Design Validation and Test Workshop, 2005, pp. 52-59.

[77] I. Verbauwhede, K. Tiri, D. Hwang, and P. Schaumont, *Circuits and design techniques for secure ICs resistant to side-channel attack*, IEEE International Conference on Integrated Circuit Design and Technology, 2006, pp. 1-4.

[78] V. Vivekraja and L. Nazhandali, *Circuit-Level Techniques for Reliable Physically Uncloneable Functions*, IEEE International Workshop on Hardware-Oriented Security and Trust, 2009, pp. 30-35.

[79] S. Voloshynovskiy, S. Pereira, T. Pun, J. J. Eggers and J. K. Su, *Attacks on digital watermarks: classification, estimation based attacks, and benchmarks* IEEE Communications Magazine, Vol. 39, Issue 8, Aug. 2001, pp. 118-126.

[80] X. Wang, M. Tehranipoor and J. Plusquellic, *Detecting Malicious Inclusions in Secure Hardware: Challenges and Solutions*, IEEE International Workshop on Hardware Oriented Security and Trust, 2008, pp. 15-22.

[81] N. H. E. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Addison-Wesley, Third Edition, 2005.

[82] M. A. Williams, *Anti-Trojan and Trojan Detection with In-Kernel Digital Signature testing of executables*, Technical report, Security Software Engineering: NetXSecure NZ Limited, 2002, pp. 1-12.

[83] H. Zhang, *SATO: An Efficient Propositional Prover*, International Conference on Automated Deduction, Vol. 1249, 1997, Lecture Notes in Computer Science, pp. 272-275.