Optimized ATPG for Hardware Trojan Detection

Animesh Basak Chowdhury

July 2016

Optimized ATPG for Hardware Trojan Detection

Dissertation submitted in partial fulfilment of the requirements for the degree of

Master of Technology
in
Computer Science

by

Animesh Basak Chowdhury
[Roll No: CS1424]

under the guidance of

Bhargab B. Bhattacharya
Professor
Advanced Computing and Microelectronics Unit

Nanotech Research Triangle
Advanced Computing and Microelectronics Unit
Indian Statistical Institute
Kolkata-700108, India

July 2016

To my supervisor and Satyapriya

CERTIFICATE

This is to certify that the dissertation entitled **"Optimized ATPG for Hardware Trojan Detection"** submitted by **Animesh Basak Chowdhury** to Indian Statistical Institute, Kolkata, in partial fulfilment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Bhargab B. Bhattacharya
Professor,
Advanced Computing and Microelectronics Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgements

I would show my deepest gratitude to my advisor and mentor, *Prof. (Dr.) Bhargab B. Bhattacharya*, Advanced Computing Microelectronics Unit, Indian Statistical Institute, Kolkata, for his guidance and continuous support and encouragement. He has literally taught me how to do good research, and motivated me with great insights and innovative ideas. He has guided me the ways to visualize the problem with all possible angles, and to get a deep insight about the topic. His directions have helped me to go beyond conventional methods, and come up with a solution which has the ability to compete with current standards. His presence always motivated me to work harder and be simple.

I would also like to thank *Dr. Ansuman Banerjee*, Assoc. Prof., Advanced Computing and Microelectronics Unit, ISI Kolkata, for his continuous pursuit in making me self sufficient with skill-sets required to do good research. He imparted the philosophy of doing research that can be implemented and contributed for benefit of society. His valuable and constructive comments has helped me a lot in shaping my research.

I am thankful to all the research fellows of Nanotech Research Triangle, Advanced Computing and Microelectronics Unit, with whom I have discussed various innovative ideas, and worked in interaction. Those valuable sessions helped me a lot, to make my research technically sound.

I would also like to show my gratitude towards the technical review committee members of "Hardware Trojan Research Project", CAIR Lab, DRDO, INDIA, for providing vital comments and review about the methodology taken and practical implementations.

My deepest thanks to all the teachers of Indian Statistical Institute, for their valuable suggestions and discussions which added an important dimension to my research work.

Finally, I am thankful to my parents and family for their everlasting supports.

Last but not the least, I would like to thank all of my friends for their help and support. I thank all those, whom I have missed out from the above list.

Animesh Basak Chowdhury
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

Combating the threats imposed by hardware Trojans that are covertly inserted in hardware has surfaced as a challenging problem in recent times. An adversary may maliciously tamper the given SoC design at netlist level to insert Trojans for the purpose of extracting secret information from the chip via output pins, or for launching denial-of-service attack. Because of several scalability issues, the detection of hardware Trojans in a large circuit turns out to be extremely difficult when the Trojan sizes are small ($\sim 8-10$) gates. Such threats may not only degrade the reliability of the system but also endanger its security, especially when these chips are used for mission-critical applications such as defence, space, biological, and in nuclear stations. As the possible instances of Hardware Trojans present in design are exponential, we propose ATPG-binning technique, using divide-and-conquer strategy, to lessen the search-complexity and then use a SAT-solver to derive a test set. Simulation results demonstrate the effectiveness and superiority of the proposed method to prior work in terms of Trojan-coverage, size of test set, and CPU-time.

Keywords: *Hardware Trojan, Trigger Instances, Trojan Instances, Trigger, Payload, ATPG-Binning.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Electronic systems have rapidly progressed over the past few decades to the point that most aspects of daily life are aided or affected by the automation, control, monitoring that are provided by Integrated Circuits (ICs). With the emergence of Information Technology and its critical role in these electronic systems, Electronic Design Automation have played a major role in shaping the world of 21st Century. With this advent, the risk of cyber attacks have also been larger than ever before. In 1980s, there was war between software developers and hackers, where software was prone to attack with malicious intent. However, as time progressed, the complexity of the design, fabrication, and distribution of electronics has caused a shift throughout the industry toward a global business model, thereby creating new sources of attack. In such a model, untrusted entities participate either directly or indirectly in all phases in the life of an electronic device or integrated circuit (IC). This unprecedented access to hardware has been a major cause for concern, resulting in very plausible conspiracy theories. The ability to trust these ICs to perform their specied operation (and only their specified operation) has therefore become a security concern and an active topic of research. ICs are the building blocks for all modern electronic systems, they form the information backbone of many critical sectors including the nancial, military, industrial, and transportation sec- tors. Without trust in these ICs, the systems they support cannot necessarily be trusted to perform as specied and may even be susceptible to attack by a malicious adversary.

In 2008, it was reported that a critical failure in Syrian radar might have been intentionally triggered through a “back door hidden within a commercial off-the-shelf (COTS) microprocessor. According to a U.S. defence contractor who spoke on condition of anonymity, a “European chip maker recently built such microprocessors with remote kill switches for just such purposes. Given the dire consequences associated with such weaknesses, the so-called hardware Trojan issue has received considerable attention from academia, industry, and government over the last decade [1, 2, 3, 5].

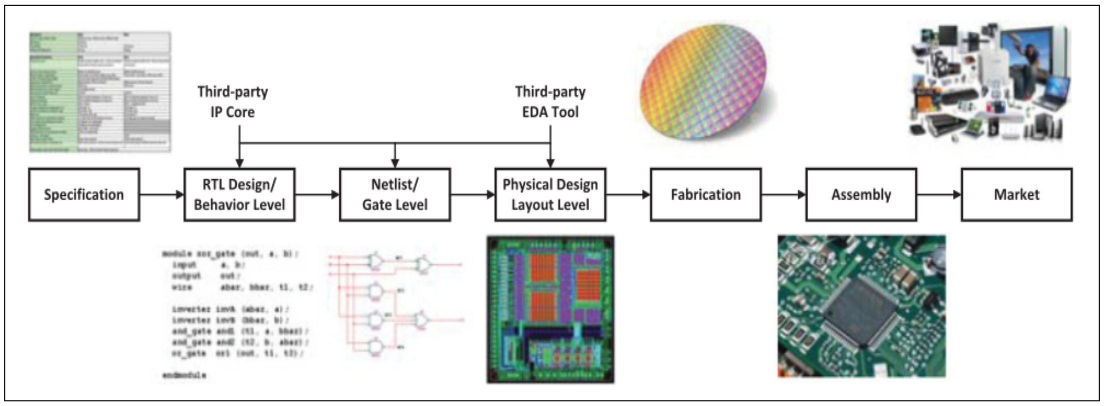


Figure 1.1: Various phases of IC development cycle[1].

With semiconductor scaling to very deep sub micron levels, the complexity and cost of IC design and fabrication have increased dramatically. An SoC component will typically go through design cycle starting from specification to retail marketing. The first step of the process involves the process of translation of the specifications into a behavioural description, typically in a hardware design language (HDL) such as Verilog or VHDL. Next, synthesis is performed to transform the behavioural description into a design implementation in gate level netlist using technological library provided by Semiconductor Companies like TSMC, Intel. After implementing the netlist as a layout design, the digital GDSII files are then handed to a foundry for IC fabrication. Once the foundry produces the ICs, the testing step ensures the chips are free from manufacturing defects. Those ICs that pass testing are then assembled, retested, sent to the market, and eventually deployed in systems.

In most of the countries, setting up foundry, fabrication facilities and R&D services for latest semiconductor technology requires a huge investment. Also, to design and setup a complex system, a company outsource most of the requirements in order to reduce cost of skilled manpower and domain experts across all the subsystems required to build the whole design. These basically involve outsourcing fabrication to third party foundry, purchase and use third-party intellectual property (IP) cores, and/or use EDA tools from outside vendors. Use of third-party tools and IPs, and integrating them into the design though speed up the development cycle, but introduce potential security concerns. The supply chain is therefore susceptible to various kind of attacks like Hardware Trojan insertion, reverse engineering, IP piracy, IC tampering and so forth. Amongst all the challenges, hardware trojans are undoubtedly the biggest concern and have earned considerable attention.

A hardware trojan is defined as malicious, undesired and intentional modification of given SoC design or electronic circuit, resulting in modified functionality of the device, when in operation. It is like a back-door that can be used by attackers to gain unauthorized access to the system, or to destroy the original functionality of the system, or to leak out important information from the system. Several previous research papers and articles have proposed various modelling of potential Trojans that can be possibly inserted in the design. In [2, 4, 10], the authors have classified Trojans into five different categories, based on the attributes: insertion phase, abstraction level, activation mechanism, region and effects. Trojans are inherently stealthy in nature, owing to the fact, that these should not be easily detectable during normal operation. Intelligent adversaries try to devise such trojans which are extremely hard-to-detect but moderately triggerable. Trojans are totally different from manufacturing defects. manufacturing defects are totally random and are unintentional. Defects can be easily modelled with stuck-at fault, delay fault, transition fault, and are comparatively easy to detect. But, the same is not valid for trojans. Trojans cannot be easily modelled to some pre-defined or existing models, hence the modified functionality are not easily detectable. Also, as the trojans can be inserted at various phase of IC development cycle, worst case would be exhaustive verification and testing to detect the presence, which is generally infeasible and time consuming.

1.1 Motivation of this dissertation

Given the threat posed by Hardware trojan inserted in the SoC design, owing to the stealthiness and malicious nature, a deep and thorough understanding is vital while developing next generation sophisticated systems, which are required for critical systems like defence, healthcare, and nuclear stations. In India, especially in defence sectors, most of the state-of-the-art technology related to electronic warfare, are either imported from foreign countries or Transfer-of-Technology is done. Ensuring trust and reliability on these imported technologies, is a major requirement and should not be taken lightly. Research over a decade in this area, have come up with lots of methodology involving Trojan Detection and countermeasures [1, 3, 4, 5, 12].

Though a lot of research have been done on trojan detection and diagnosis, and preventive measures have been taken, the major shortcomings amongst the techniques is that, they are able to detect the trojans, which have been modelled as per the philosophy defined in insertion of trojan. Latest research have shown, that if the trojan modelling is changed, the technique fail to detect the trojan. Though the researchers and academicians in this area have been considerate enough to provide some benchmarks about the trojan-free and trojan-injected circuitry, the real attackers and adversaries are well aware of the modelling, and would try to devise such trojans that can evade current detection schemes.

So, in this cat-and-mouse game between attackers and the people involved in detecting trojans, there has to be a solid framework, which has the ability to detect most of the trojans, modelled on any philosophy. The systems, which are generally required in defence, space, nuclear stations and Health centres require cent percent assurance of the system to be trojan free. Motivated by the need, we decided to have a survey of current techniques and methodology adapted for detection, which has the ability to cover all possible flavour of trojans. The authors in [16, 18, 20, 23, 24] have used current signature and power analysis, to detect the presence of extra and redundant circuitry in the given design. In [25, 35], authors have presented layout and region based approach for detecting Hardware Trojans. However, in [17], authors have taken best of all the worlds, and used multiple side channel parameters, to detect trojans. The technique was immensely successful in detection of majority of trojans, irrespective of any modelling of trojan. It monitors all the side channel parameters, and compare them with the parameters obtained from golden trojan free design. Any deviation from accepted threshold values is reported as presence of trojan in the design. However, the method issue false positive reports, when the trojan sizes are extremely small, i.e. trojans are inserted by additional insertion of 3-4 gates, and nets.

All the methodology related to side channel parameters suffer from this issue of false positive, when comes to small sized trojans. A path breaking work was done by authors in [31], where logic testing based approach was used and trojan coverage was phenomenal. The technique, added with multiple side channel parameters seemed to be a perfect framework, for trojan detection. But, there was still an underlying problem related to technique discussed in [31]. The method failed to provide cent percent coverage, and when sizes of circuit increased, the trojan coverage failed miserably. Our motivation was to propose an optimised ATPG , which would be able to detect exhaustively all possible small trojans in given design, within reasonable time.

1.2 Contribution of this dissertation

Our test generation framework SATBiST, is able to provide optimised test patterns, which would be able to provide a cent percent *trigger coverage* and a wider *trojan coverage*. The framework is scalable, and generates optimized test pattern, providing exhaustive coverage. The framework assumes the generic philosophy of trojan insertion, i.e. stealthiness and trojan activation is rare occurring event. The framework employs divide-and-conquer strategy called ATPG Binning, which in turn generate test patterns for a considerable number of trojan instances. We have taken the help of already existing optimised tools, and modelled the problem of test pattern generation in such way, that can provide a good trigger and trojan coverage. Test-patterns for hard-to-detect trigger instances are generated with the help of SAT Solver. In the end, optimized ATPG is reported providing cent percent coverage of all possible trigger instances. Simulation results show effectiveness of the technique in terms of trojan-coverage, CPU time, and test-length.

1.3 Organization of the dissertation

The rest of the dissertation is organized into 6 chapters. A summary of the contents of the chapters is as follows:

Chapter 2: A detailed study of relevant research is presented here.

Chapter 3: This chapter describes the mathematical calculations related triggering instances of Hardware Trojan, taken into consideration.

Chapter 4: This chapter formalizes the ATPG Binning approach, divide-and-conquer strategy to provide scalability.

Chapter 5: This chapter describes the approach for test generation for Hard-to-detect trigger instances using SAT.

Chapter 6: This chapter describes the detailed framework implementation and results.

Chapter 7: We summarize with conclusions on the contributions of this dissertation.

Chapter 2

Background and related work

In this chapter, we present a few background concepts needed for developing the foundation of our framework. We also present an overview of different schemes proposed in literature for Hardware trojan detection.

2.1 Background

In this section, we discuss a few background concepts.

2.1.1 Trojan Taxonomy

Before going to the methods used for trojan detection and countermeasure techniques, we first go through few fundamental concepts about nature of Hardware Trojan. When trojans are non-destructively inserted in any phase of IC design cycle, the threat remains whenever the system is powered on. It depends on the adversary and the stage of insertion, about the kind of modification a trojan would be carrying out, after infecting the design. Researchers and Academicians have gone through a rigorous research, and have come down to a broad level of trojan taxonomy, that is widely accepted amongst the community. Fig. 2.1 and 2.2, shows a detailed classification of types of trojan, based on various attributes. The detailed classification help us to understand, the possible areas and phases which are vulnerable, and the behaviour of trojan, i.e. whether the trojan is used for modifying functionality, or gaining unauthorized access to the system, or for leaking out sensitive information. In order to design a strong methodology and framework to detect trojans, irrespective of any modelling or assumption used for inserting the malicious logic, it is a necessary thing to find out common parameters.

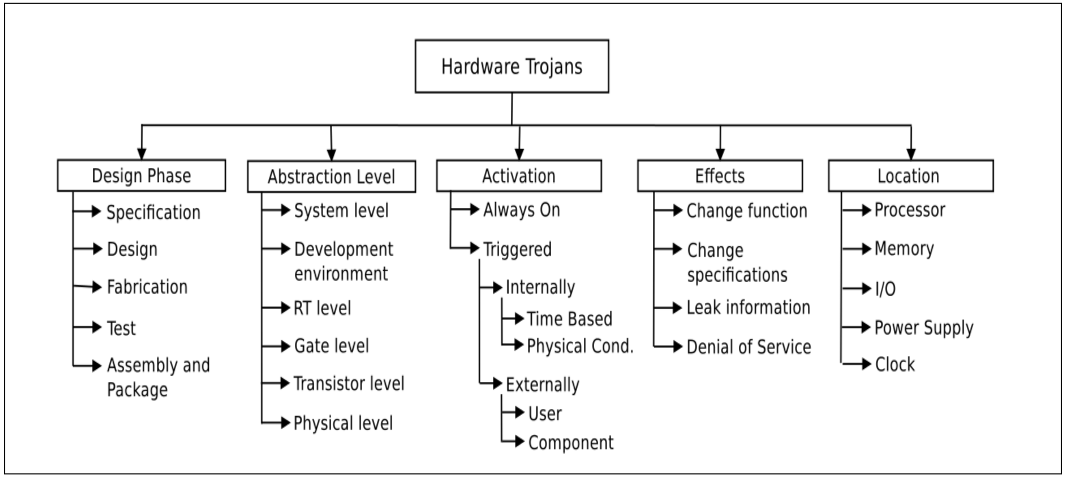


Figure 2.1: Trojan Taxonomy[4]

There have been several Hardware Trojan taxonomies proposed to describe such common attributes, with the aim to enable a systematic study of Hardware Trojan characteristics, to aid the development of detection and mitigation techniques for given classes, and to facilitate benchmarking for detection and mitigation strategies. Chakraborty et. al. [4], proposed a classification shown in figure 2.1, extended from Wolf et. al. [9], that is based upon trigger and payload mechanisms. Rajendran et al. [10] extend this taxonomy further , by consideration of design phase, abstraction level and location. In our dissertation, we have followed the trojan taxonomy proposed by authors in [4].

A trojan is basically designed using a suitable trigger-payload combination. Fig. 2.1 shows various areas of mixed signal SoC design, which can constitute possible trigger and payloads. Before going to the details of trojan designing, let us first clarify the two terms: **Trigger** and **Payload**.

TRIGGER : The malicious functionality introduced into the design, which is responsible for activation of trojan infection. *Trigger* is used to feed the trojan to start corrupting the original functionality or the job intended to be done by the attackers.

PAYLOAD : The portion of the design, which is going to be affected by activation of trojan. Selection of payload depends on the type of attack intended to be launched into the design. An attacker's viewpoint would be finding out a good trigger-payload combination, which create such trojan instance, which is very hard-to-detect.

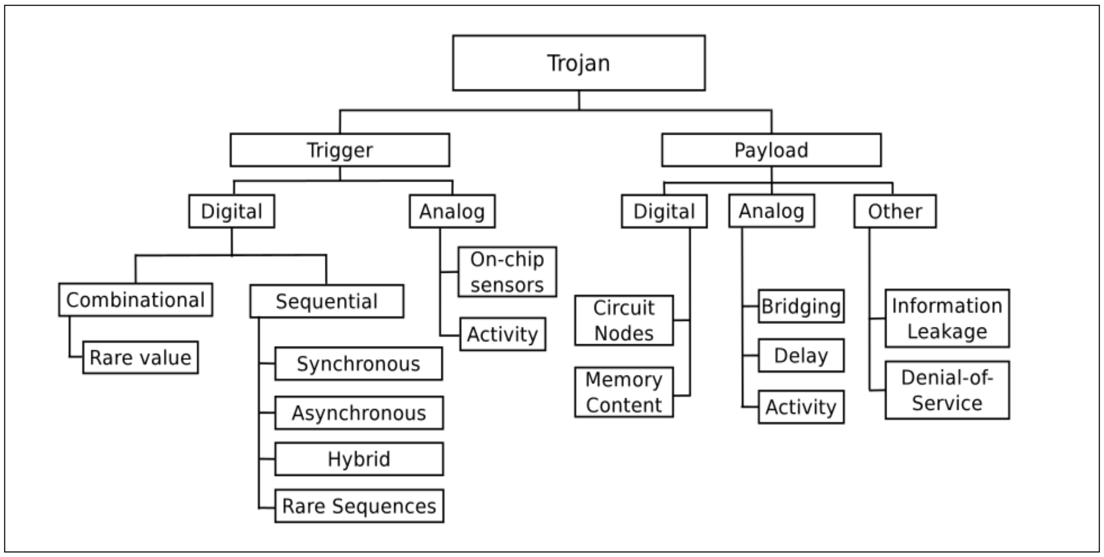


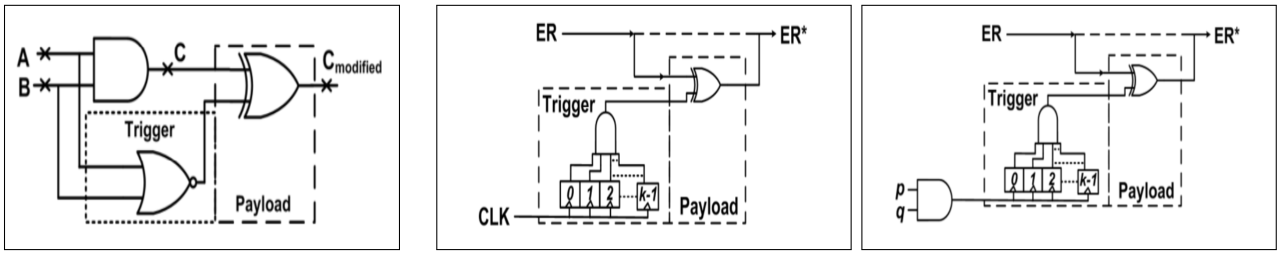
Figure 2.2: Trojan Taxonomy[10]

2.1.2 Trigger Mechanism

As already discussed in previous sections, trojan can be inserted into the design in variety of ways, and they pose many direct and leveraged threats. Once the design is infected with trojan, most of them will lie dormant until triggered to corrupt the activity. Activation can be of any technique, mostly covert, random, directed or predetermined that elicits a change in state or behaviour of the trojan. This activation phase is important as it provide crucial state of the system, for detecting and countering trojan. During various testing and verification phase of an IC development cycle, every attempt is made to activate the trojan which are stealthily inserted in the design. Typically, it requires intelligent state-space exploration and optimized functional validation techniques, involving input, output and current state of the system. Triggering a trojan increase a huge chance of a trojan getting activated and start its functionality. Most of the research on trojan detection heavily rely on activating the trigger pattern. It is important to understand the manner in which Hardware Trojan can be triggered, considering the level of threat posed by them. A brief taxonomy of various trigger mechanism has been presented in consequent sections. Figure 2.2, gives a detailed classification of types of trigger mechanism.

Digital logic Trigger

These trigger basically rely on some specific internal state of the circuit. The most common types are combinational and sequential activation. Most of the digital logic trigger are internally triggered i.e. the trigger gets activated when certain values at specific internal nodes.



(a) (b) (c)

Figure 2.3: Various kind of Trigger-payload combination of a Trojan. (a) Combinationally triggered trojan. (b) Synchronous counter based trojan. (c) Asynchronous counter based trojan.

The trigger is constituted by combination of these specific internal (nodes/states) only. Digitally triggered trojans can again be classified into two broad categories - **Combinational** and **Sequential** triggered trojans.

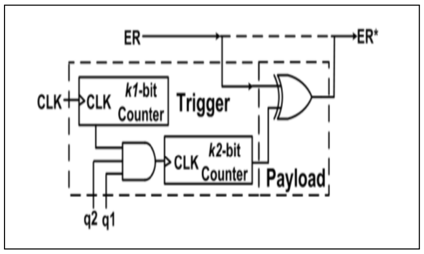
Combinational Trojan: In combinational trojan, inputs to trigger instance comes from internal nodes of a circuit, which is a pure combinational logic. In general, attacker tries to identify the possible areas, from RTL (Register Transfer Level) verilog/vhdl code, which can be fused together, to form a "combinational" trigger. Typically, a particular set of lines from address/data bus, combined with some logic values coming out from ALU of a processor, can form a combinational trigger. In figure 2.3a, a combinationally triggered trojan has been presented. When $A = 0$ and $B = 0$ occur at respective nodes, triggering happens, causing payload node C, to have a corrupted value $\mathrm{C}_{\mathrm{modified}}$. Typically, an attacker will try to find such rare combination, which does not get detected during conventional manufacturing test.

Sequential Trojan: This kind of trojan are also known as "Time-Bombs". They are triggered after a certain sequence of operation. The simplest sequential Trojan can be generated from simple clock based counter, which activates a trojan after reaching a certain count. In figure 2.3b, a synchronous k bit counter gets triggered, when the count value reaches $2^k - 1$. The actual value ER is corrupted to ER^*. The asynchronous counterpart of synchronous sequential trojan, is shown in figure 2.3c. Here, the counter value gets incremented, with the number of times, the condition $P = 1$ and $Q = 1$ is satisfied. After the threshold value is reached, which was set by the attacker, the trigger gets activated, resulting in corruption the value of ER to ER^*.

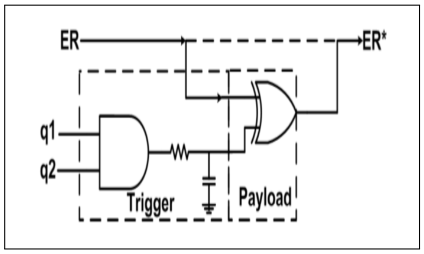
Other than combinational and sequential trojans, there can be trigger mechanism, which incorporates both types into a single trojan. In figure 2.4a, $k2$-bit counter, is fed by a synchronous counter and a combinational logic function of P and Q. These trojans are called **hybrid trojans**. now, when the size of design becomes large and more complex, various sub-circuits/sub-regions can be used to create a trigger instance based on a sequence of rare events. In general, owing to the stealthy nature, sequential trojans are very hard-to-detect using conventional test generation, as it require to satisfy the internal state of the circuit, used to create the trojan instance, which is very rare. The number of such sequential triggers are undoubtedly large and conventional state-space exploration would not be able to provide cent percent coverage of trojan detection.

Analog Trigger

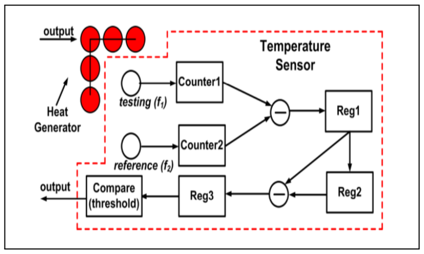
These trigger can be broadly classified into two categories : **internally triggered and externally triggered**. In internally triggered, analog trojan, addition of some capacitance value to power source or load, would act as a trigger, whereas, in externally triggered trojan, on-chip sensors can be used to trigger a malfunction. In figure 2.4b, the capacitor which has been inserted in the circuit, gets charged via load resistor, if the state, q_1=1 and q_2=1 is reached. Other than that, the capacitor, remains discharged. The charged capacitor causes the logic threshold to cross, after large number of clock-cycles. An externally triggered analog trojan has been shown in 2.4c. Here, higher circuit activity and high temperature was used by the adversary, to trigger the malfunction, through a pair of ring oscillator and counter.



(a)



(b)



(c)

Figure 2.4: Various kind of Trigger-payload combination of a Trojan. (a) Hybrid nature trojan. (b) Ananlog trojan (Trigger based on logic value at AND gate). (c) Analog trojan (Trigger based on input from sensor).

2.2 Related Work

Research on hardware trojan started way back in 2005-06, with one of the most promising field considering the security threat posed by systems at hardware level. After that, the field has seen significant progress since the first remarkable report of Agrawal et. al. [15], at IBM T.J Watson Research center, New York, US. In order to exploit potential threat posed by trojans, various trojan modelling have been developed. As discussed in previous sections and following the taxonomy described in [18], a trojan basically consist of two parts: *trigger* and *payload*. As inherent nature of trojan, trigger is expected to be activated under extreme rare condition, therefore, payload remains inactive for most of the time. And in presence of inactive payload, there is no point in malfunctioning of system, and behave as trojan-free circuit.

Existing research for hardware trojan design has been much more focussed on trigger and payload mechanisms. Since, trigger and payload determine the difficulty of detection and activation, researchers have explored and modelled novel trigger designs and payload mechanism. Some of the newest trigger mechanisms have been described in [13, 14], where trigger model consist of don't care terms, or silicon wear out mechanisms, for trojan activation. New payloads have been modelled such that, the side channel parameters leak out important information from the design. The major takeaway is for trojan infected circuitry, due to trigger and payload, inevitably some side effects are introduced, such as additional power consumption, change in path-delay, additional area, or radiation. These parameters can be effectively used by defenders for detecting trojan. Thus, researchers in the field, already aware of attacker's philosophy, have tried to model and optimize trojan design in such way, that trojans are much more stealthy in nature and affect normal operation in a minimal way possible. Various approaches to the mentioned philosophy have been presented in [1, 2, 3, 4, 6, 8, 9]. Finally, in 2014, the research community felt the need to present some standardized benchmark, to compare effectiveness of various techniques fairly. The standard benchmarks in various formats such as RTL, gate level, and layout form, and trojan affected designs have been presented in website *trust-hub.org* [52].

Current research trend focusses on various countermeasure techniques that are able to deal with trojan threats, inserted at various phases of supply chain. Figure 2.5, shows various categories of countermeasures, that have been proposed against the threat of hardware trojan. We will describe more about two major schemes : Trojan Detection and Design-for-Trust methodology.

Trojan Detection

This category has been pretty straightforward and commonly used methodology to determine the existence of trojan. They are performed generally at two stages : pre silicon and post silicon stage. The detailed explanation and philosophy of techniques falling under different category are described in subsequent sections.

Post Silicon Detection Schemes

As shown in figure 2.5, post silicon techniques can be further classified into two categories, destructive approach, and non-destructive approach. In destructive approach, an IC is de packaged using reverse engineering techniques, and obtain images of the original components

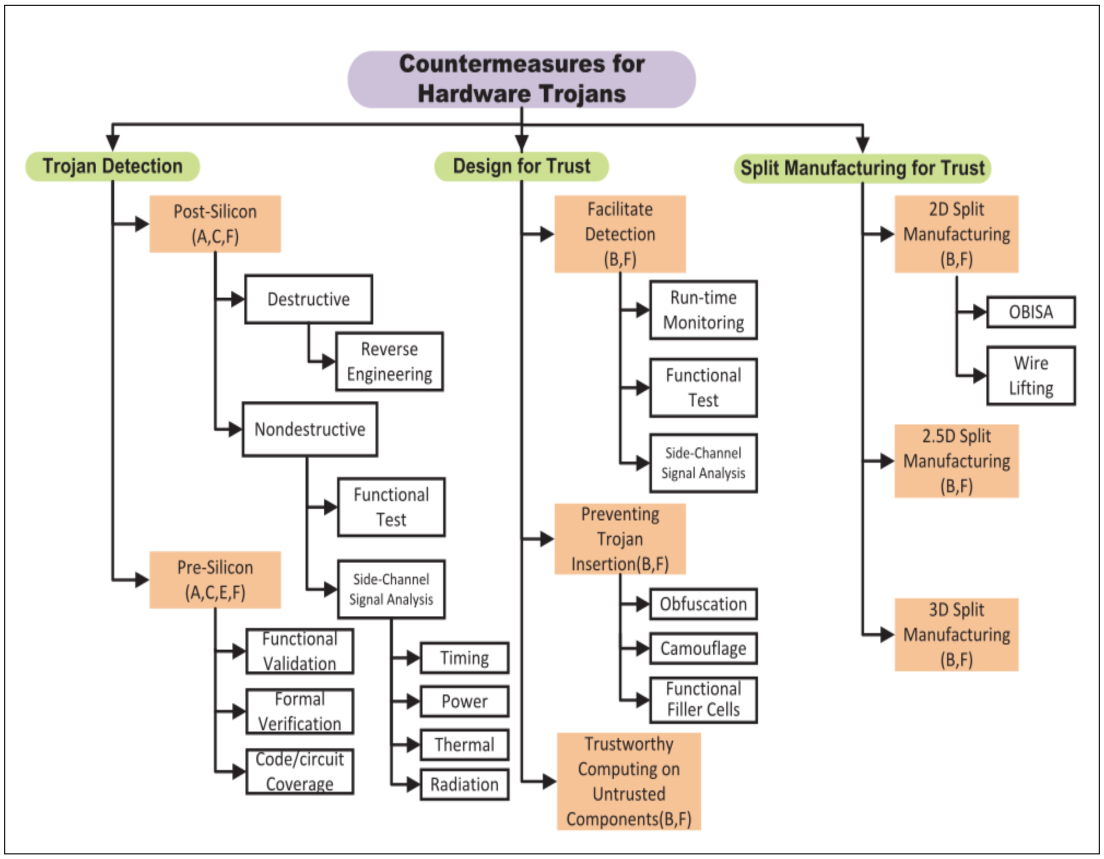

Countermeasures for Hardware Trojans
Trojan Detection
Design for Trust
Split Manufacturing for Trust
Post-Silicon (A,C,F)
Destructive
Reverse Engineering
Nondestructive
Functional Test
Side-Channel Signal Analysis
Pre-Silicon (A,C,E,F)
Functional Validation
Formal Verification
Code/circuit Coverage
Timing
Power
Thermal
Radiation
Facilitate Detection (B,F)
Run-time Monitoring
Functional Test
Side-Channel Signal Analysis
Preventing Trojan Insertion(B,F)
Obfuscation
Camouflage
Functional Filler Cells
Trustworthy Computing on Untrusted Components(B,F)
2D Split Manufacturing (B,F)
OBISA
Wire Lifting
2.5D Split Manufacturing (B,F)
3D Split Manufacturing (B,F)

Figure 2.5: Countermeasure for Hardware Trojan[1]

to reconstruct design-for-trust validation. Though the technique has the potential to give 100% assurance, that the chip is malicious or not, but it takes several weeks or months to determine the trust, for an IC of reasonable complexity. The technique is suitable, to get the golden model of trojan-free ICs. Bao et. al. [28], proposed reverse-engineering based solution, using machine learning technique, for identification of golden IC. Non destructive techniques try to authenticate the chip, after fabrication. These include functional tests, which are performed with manufacturing tests, and side channel analysis technique.

In functional tests, appropriate test vectors are applied to IC under test, and the results are compared with response of golden IC. While, the technique sounds familiar with manufacturing tests, the test patterns optimized for various fault-models provides poor coverage on trigger and thereby on trojan coverage, as mentioned in [5, 9]. Attackers try to model the trigger-payload combination in such a way, that it remains hidden, and has the ability to bypass manufacturing tests. As, trojan activation is a rare event, Chakraborty et. al. [31], Banga et. al. [35], Jin et. al. [33], proposed various optimized test patterns, which comparatively increases the trigger and trojan activation. These patterns help to catch the malfunctioning which would have been occurred by trojan injection. However, these techniques fail to scale up, thereby exhaustive search on whole state space is not possible. Authors in [32, 34] have provided comparable improvements, still the issue of exhaustive coverage remains a critical problem.

In side channel analysis based methods,involve monitoring the physical parameters like path-delay by Jin et. al.[18], power based analysis given by Agrawal et. al.[15], Rad et. al.[23]. Rad and Jim et. al.[24] and current signature based methods given by Wang et. al. [19] and Aarestad et. al. [20]. All these techniques mentioned carefully monitors the parameters mentioned, and check for abnormalities w.r.t the parameters of golden IC. Any variation beyond a tolerable threshold would be reported as presence of Hardware Trojan in the design. However, until recently, it has been observed that the technique suffers from process and environmental variations. Still, the side channel parameter technique is one of the best method in hand today, for trojan detection, where sizes of trojan induced inside the circuit is typically big.

However, combining testing based methodology and multiple parameter side channel analysis, can prove to deliver a reliable and trustworthy tool, which would be able to detect trojans of all possible sizes. The only requirement of developing such framework is to have a golden IC/trojan free circuit of the design, which has to be checked for reliability and trustworthiness.

Pre Silicon Detection Schemes

The main idea of functional validation is same as logic testing based methodology described earlier. the functional validation is done using simulation in pre-silicon stage, while in post-silicon stage, the same is done, by applying test patterns, and collecting output response from chip. HDL analysis can also be done at pre-silicon stage from the codes which are written at behavioural level [43], or at structural level [41]. Using structural analysis of the code, various metrics can be defined and used, to find the probable areas, of trojan injection in the design. The limitations of code/structural analysis is that, they don't guarantee trojan detection. Formal verification method is mathematical and formal approach to logic verification, that has the potential to exhaustively prove predefined set of properties, where properties are written in security point of view. The design, is converted into a finite-state-machine (FSM) and using satisfiability problem, it is checked whether the properties are satisfied during the run. The technique suffers from scalability issues, and fail to detect additional functionalities introduced by trojan in design.

Design for trust

Detection of stealthy and low overhead trojan is quite a challenging problem, in the area of hardware trojan detection. An effective way to deal with this problem, is to facilitate various modification and introduction of additional circuits during design phase itself. A way of introducing design-for-trust is to introduce additional test points in the circuits, to those nodes which have low testability measures. Another approach is via runtime monitoring, which can significantly increase the level of trust. Runtime monitoring introduce supplemental on-chip circuit, to monitor the behaviour during runtime. Jin et. al. [44] proposed a post-silicon based method, which uses a neural network, that can be trained using on-chip parameters, and used for distinguishing the circuit operating in normal mode, and working under trojan attacked mode.

Another approach to implement design-for-trust in IC, is to apply preventive measures that attempt to thwart insertion of hardware trojan inside the design. This is for those attackers, who are not present in design phases, but they are present in later phases of development cycle. one way is by Logic Obfuscation. In this method, unless the key is not provided, during the operation of circuit, the exact functionality of the design is not revealed. The circuit produces correct function, when correct key is applied. Chakraborty et. al. [27] devised such method in 2009, which was further improved by many researchers of the domain.

Split-manufacturing-for-Trust is a more recent technique as a countermeasure to trojan detection. It uses current state-of-art foundries while minimizing the risks to an IC Design. The technique is very new, and theoretical solutions have been proposed. However, research is still going on, with scope of delivering promising solutions.

Chapter 3

SATBiST Framework : Analysis and foundation

In this chapter, we formally explain the mathematical analysis of the foundation and assumption, we have taken into consideration for developing our test framework. In a subsequent chapters, we elaborate the algorithms involved in the framework and experimental results.

3.1 Basic Definations and Terminologies

We now present a brief overview about the rare event occurrences we have taken into account, while detection of trojan present in a given circuitry.

Here, in figure 3.1, a simple combinational circuit's gate level netlist has been shown. Under normal and ideal condition, the circuit is expected to produce the output as desired. Now, suppose, we have introduced a trojan in the design, and now the circuit looks like in figure 3.2. What we have done is, we have ANDed up some of the internal nodes of the circuit, and XORed the output of AND gate to some randomly chosen primary output. Now, as shown in the figure 3.2, when $G7 = 0$, $G8 = 1$ and $G11 = 1$, occurs simultaneously, the ANDed output becomes one, and hence, the value at primary output, $G15$ is inverted from the expected output.

Now, the question which comes to mind, that whether simultaneous occurrence of $G7 = 0$, $G8 = 1$ and $G11 = 1$ is really a RARE event or not. As its the inherent nature of trojan, to be much more stealthy in nature, we have to find those state of the design, which are very rare to occur. To determine rare events at netlist level, we have followed the method given by Salmani et. al. [30]. We determine the probability of occurrence of 0 and 1, at

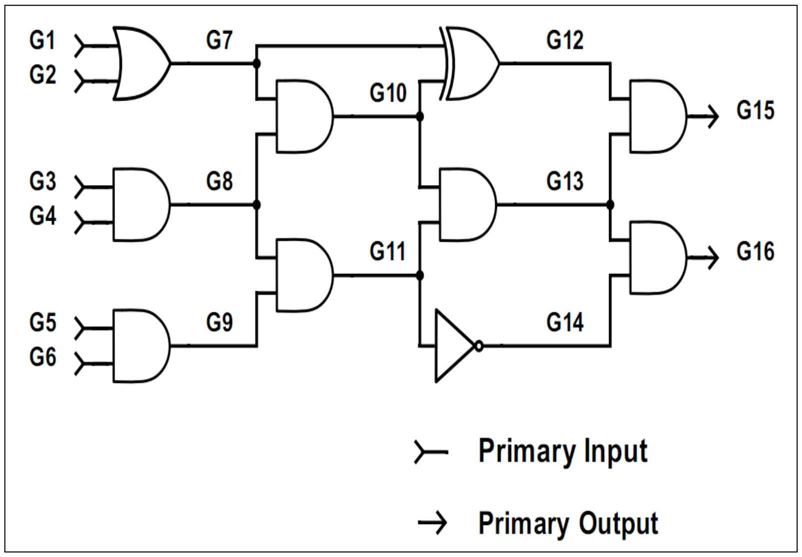


Figure 3.1: A simple combinational circuit

each of the node of the netlist. This can be done in two ways. First, using some arbitrary high number of random test vectors are applied to the design, and the occurrences of 0s and 1s are determined. This method is simulation based. Another technique is by static analysis. In this methodology, we have applied the probability of occurrence of 0 and 1, 0.5 at each of the primary input, and then, determine Probability of Occurrence of 0 (P_{zero}) and 1 (P_{one}). Figure 3.3, shows the values of (P_{zero}) and (P_{one}) at each of the nodes.

Now, consider a node, where probability of occurrence of a logic is very low. We now, introduce traditional concepts of VLSI testing, **controllability** and **observability**. Suppose, a line is having the value of P_{zero} very low. So, given a set of random test vectors applied on CUT(Circuit-under-test), the chances node/logic line having logic value zero is very less, i.e. given a test vector set, it is very difficult to get logic zero on that node. So, the node is having very less controllability of logic zero. We can clearly assume that, logic 0 on this particular node can be defined as **RARE event**, since a less controllable line will ultimately mean that there exist very few test vectors, for occurrence of rare logic to that node.

We now define some important terminologies, which we would be using further for foundation of our test framework.

Definition 3.1 Rare Event: *A state of the circuit/design which occurs very rarely, given a set of input patterns.* ■

Definition 3.2 Rareness Threshold: *A threshold probability value set, below which a*

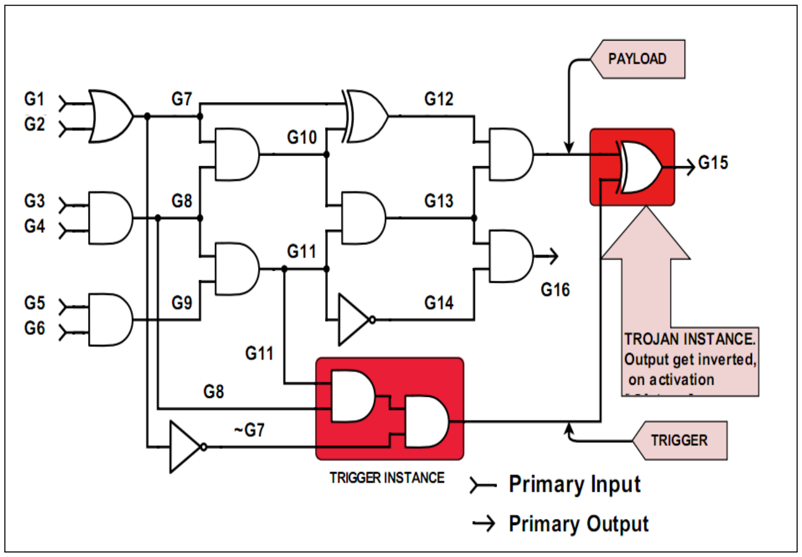


Figure 3.2: A trojan infected combinational circuit

node, along with its logic value would be considered as rare event of the circuit. Denoted by θ. ■

Definition 3.3 Activation Node: *A node where either P_{zero} or P_{one} is less than θ. It is denoted as tupple (i, L), where i, represents node number, and L is the boolean logic for which $P_L < \theta$.* ■

Definition 3.4 Trigger Instance: *A set of Activation Nodes, ANDed together to form trigger of hardware trojan, is called Trigger Instance.* ■

Definition 3.5 Q value: *Number of Activation Nodes, ANDed together to form trigger of hardware trojan, is called Q value.* ■

Definition 3.6 Trojan Instance: *A suitable trigger-payload combination to form a hardware trojan, is called Trojan Instance.* ■

Definition 3.7 Controllability: *It is a parameter related to each node of the circuit. Defined as difficulty in sensitizing the node to given logic value, given a set of randomly chosen input test vectors.* ■

Definition 3.8 Observability: *It is a parameter related to each node of the circuit. Defined as the difficulty in propagating the logic value to one of the primary output of the circuit.* ■

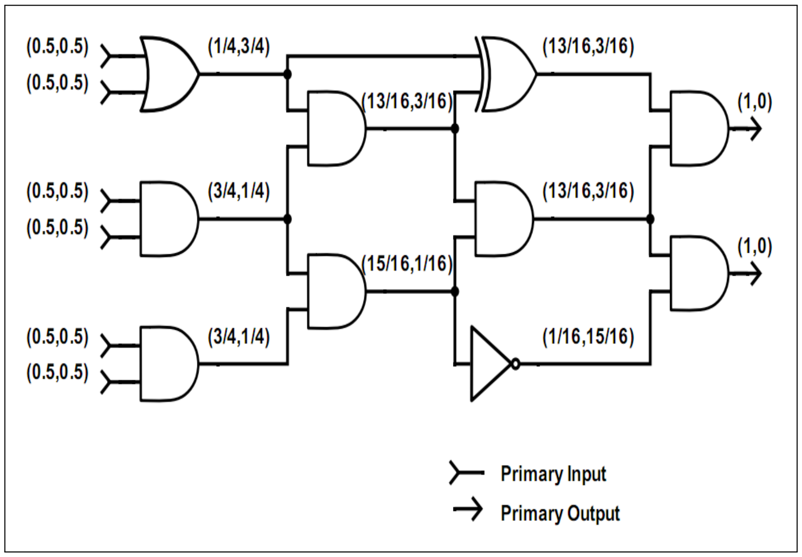


Figure 3.3: Static Analysis showing testability measures (P_{zero},P_{one})

In subsequent sections, we now describe, how the trigger instances have been modelled in our designs, and how our testing techniques have enable us to perform generate test vectors to perform exhaustive coverage.

3.2 Hardware Trojan Modelling and Analysis

We now discuss how we have modelled our trojans and triggers, that can be present in the given design, and our contribution to design a framework, ensuring reliability and trust of a design. We have considered that trojan activation is a *RARE event*, and in a netlist level, a *RARE event*, can be modelled by a combination of *Activation Nodes*. A defender, whose primary role, is to detect hardware trojan, would like to set the value of *Rareness Threshold*, very low, such that only a smaller percentage of nodes can be termed as activation nodes. This is done, in order to evade the manufacturing tests, that are primarily to detect *defects* in the CUT. After the set of activation nodes is available, *Q value* is set, to determine the number of nodes participating to form a trigger instance.

Generally, any *Q value* can be taken form a trigger instance. Typically, this Q value depends on choice of attacker. However, existing side-channel parameter techniques [17, 18, 22, 24], are capable of monitoring the power, current and various signatures, and would be able to detect the trojan presence, when value of $Q > 8$, as cited in [31]. So, logic testing based approaches come to the rescue, in detecting trojans, which are of small sizes, just by introduction of one gates and couple of nodes from netlist. Side-channel techniques fail to detect these, as the parameters are within tolerable limit of trojan-free ICs.

MERO test patterns proved to be quite good in comparison to existing ATPG patterns and random test vectors. The technique, first proposed in 2009, was to be used along with side-channel analysis, to detect trojans of any *Q value*. However, MERO test patterns lack scalability and exhaustive coverage of all trojans that can be present in the CUT. Motivated by the drawback, we decided to give set of test patterns, which are capable of providing exhaustive coverage over all trigger instances, and trojan instances. We now discuss about how we have modelled trigger instances, into existing fault models.

3.2.1 Fault Modelling

We now explain how errors and defects in electronic circuits is suitably mapped to various fault models. Without going into vast details, we would like to explain the portion that are relevant to this dissertation. Rest of the things are out of scope of this dissertation.

After post-silicon stage, the chip which are fabricated, need to under manufacturing tests. This is required in order to detect the defects and errors that has been introduced in the design, during fabrication. We have assumed that the chip performs corrected functionality and has been verified extensively at pre-silicon stage. Now, during fabrication, it may happen, that some of the internal logic lines, has been shorted to +5V line or ground line. Due to this, irrespective of any boolean operation that is being performed, does not get reflected in the line, as it has been mistakenly shorted to some logic value.

In order to cope up with this problem, test patterns are being applied, to check whether these kind of defects are present in the circuit or not. The defect, mentioned above is effectively modelled with **stuck-at fault model**. In *stuck-at fault model, (s-a-f)*, either the line is said to "stuck-at" value 0 or value 1. So, ATPG (Automatic Test Pattern Generator) derives test vectors, that is able to cover all possible "stuck-at" faults at each of the lines. Now, it depends on controllability and observability of node, whether stuck-at fault is easily detectable, or hard-to-detect. In general, for a node to have a good testability, it is required to have both, i.e. good observability and good controllability.

In stuck-at fault model, suppose we need to find "stuck-at-0" (s-a-0) fault at a certain internal node of a circuit. So, in order to check the fault, we need to apply boolean logic in such way, that the faulty is sensitized to the opposite value, i.e. for s-a-0 fault at a node, we need to apply such test vector, that sensitize that node to 1. Now, this 1, needs to be propagated to be visible in one of the primary outputs. For sensitization, the controllability of the node needs to be good, and for propagation, the observability needs to be good enough. We now, formally define the stuck-at fault model, which would be used by us, for effective mapping of trigger instances to testing methodologies.

Definition 3.9 Stuck-at Fault Model (s-a-f): *This defect oriented fault model is defined by assigning a fixed boolean logic (*0 *or* 1*) to a node in a circuit. A node can be primary input, internal signals, or primary output of gate level netlist of a circuit. For* *manufacturing tests, two stuck-at faults are used per node, i.e. s-a-0 fault and s-a-1 fault.* ■

3.2.2 Mapping trigger instances to stuck-at-fault model

In earlier sections, we described, how we have modelled our trigger and trojan instances. In our dissertation, all the assumptions, considerations, and trojan modelling have been strictly described below.

- *Trigger Instances*: We have created trigger instances, by combination of activation nodes in the circuit.
- *Q value*: For logic testing based methodology, we have defined that $Q < 10$.
- *Trojan Instances*: For now, we have generated test vectors considering payload to be primary output of a circuit.

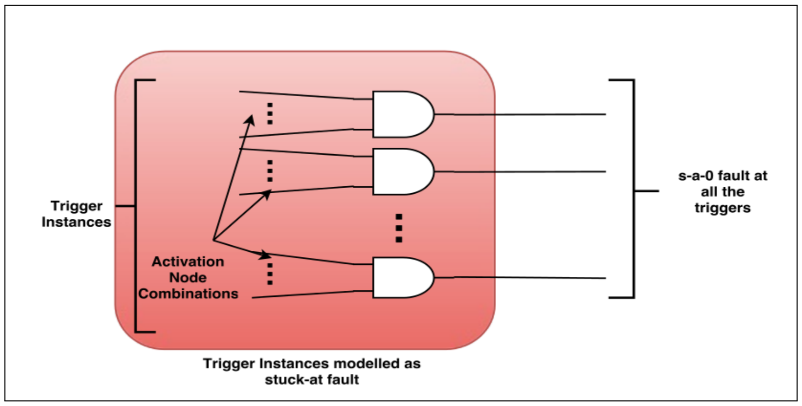

Trigger Instances
Activation Node Combinations
s-a-0 fault at all the triggers
Trigger Instances modelled as stuck-at fault

Figure 3.4: Trigger Instances modelled as stuck-at fault model

In figure 3.4, it has been shown, how the trigger instances has been modelled using stuck-at fault model. In order to activate the trigger, the output of AND gate has to be 1. So, we would apply ***s-a-0***, at the output of trigger instances, and will generate the test patterns, for that. In our modelling, as we have taken the payload to be one of the primary outputs, test pattern, which would be activating the trigger instances, will automatically activate the trojan. Now, in order to provide exhaustive coverage, we need to obtain all possible combinations of activation nodes, we are possible trigger instances. Given the value of θ, we would be having a list of activation node tuples, and Q, i.e. number of activation nodes in one trigger instance. We now, present the algorithm, which would report, all possible trigger instances.

Algorithm 1 ENUMERATING TRIGGER INSTANCES

```
Input:
    $N$ : Gate level Netlist
    $\theta$ : Activation Node threshold Probability
    $q$ : No. of Activation Nodes in Trigger Instance Instance
Output: $T$ : Trigger Instance set
 1: Read gate level netlist of design.
 2: for $\forall$ node $i \in N$ do
 3:     Calculate $P_{\text{zero}}(i)$ and $P_{\text{one}}(i)$
 4:     if $P_{\text{zero}}(i) < \theta$ then
 5:         $R \leftarrow R \cup (i, 0)$
 6:     else
 7:         if $P_{\text{one}}(i) < \theta$ then
 8:             $R \leftarrow R \cup (i, 1)$
 9:         end if
10:     end if
11: end for
12: Set $R$ is reported as set of tuples $(i, L)$, where $i$ is the node and $L$ is its associated Rare
    logic value.
13: Initialize $T \leftarrow \phi$.
14: for $\forall$ tuple $(i, L) \in R$ do
15:     Generate a trigger instance $TRIG$ , taking $q$ tuples at a time.
16:     if $TRIG \notin T$ then $T \leftarrow T \cup TRIG$
17:     end if
18: end for
19: Report Set T.
```

As shown in algorithm 1, set T is reported, containing all possible trigger instances. As the number of trigger instances, are exponential in terms of number of activation nodes, we provide a two-step approach, which would be able to report minimum possible testset, which would be able to provide cent-percent coverage. In next chapters, the algorithms related to test generation framework has been described in detail.

Chapter 4

SATBiST Framework : ATPG Binning Based Test Generation Method

In this chapter, we discuss the ATPG Binning based test generation methodology, where we used divide and conquer based method, and parallely test generation along with fault simulation based technique. We use the results obtained in the previous chapter to intuitively illustrate the objectives of our work. As already discussed in previous chapters, the number of trigger and trojan instances are exponential in terms of the number of activation nodes, and feasible payloads present in the given design. Reporting a minimal testset, to provide cent-percent coverage of all trigger instances, is a real challenge, and is require, to ensure reliability and trust of a design. We present our first approach out of two-step approach we have taken to address the problem.

4.1 Appropriate Disjoint Binning

We are available with set T, containing all possible trigger instances, for a given value of Q and θ. We appropriately divide the trigger instances into disjoint bins, each containing B trigger instances. Let S be called the set, containing all these bins. Now, $\forall$ bin $b \in S$, a modified netlist is created. A modified netlist can be created, by taking original gate level netlist N, and additionally inserting trigger instances into the netlist, and make the trigger, primary output. Let, *numBins*, be the cardinality of set S. Now, as described, for each bin $b \in S$, a modified netlist k' is created, where along with the original netlist, all trigger instances of that bin b, is also inserted. Let the collection of all these modified netlists created for each bin, be called K.

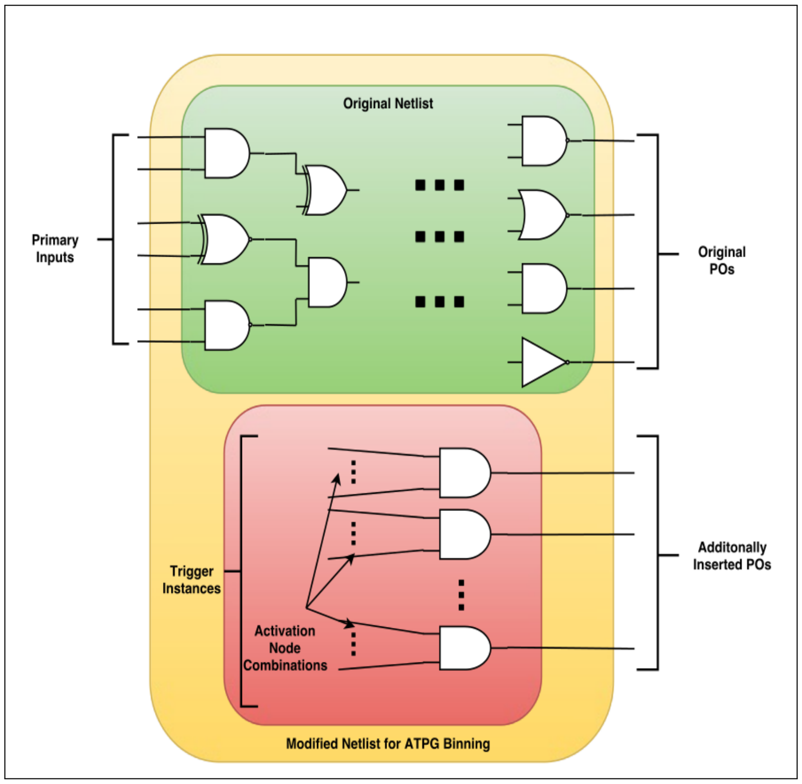


Figure 4.1: Trigger injected netlist k'.

In figure 4.1, a typical structure of modified netlist for each bin, has been shown. Here, each modified netlist $k' \in K$, contain B trigger instances. As discussed previously, we have modelled the trigger, using stuck-at fault modelling, with s-a-0 fault at each of the trigger, acting as primary output. We need to create a fault file corresponding to each modified netlist, containing all the trigger with s-a-0 fault. This fault file f, is required to supplied to the existing ATPG tool, to generate test vectors targeting the faults given in fault list. This is done to generate test patterns specifically which can excite the trigger instances.

4.2 Test generation with Parallel fault simulation

While in previous methods given for logic testing targeting this problem, statistical methods were given accompanying N-detect test [31], and using Genetic Algorithm [34] guided approach to develop test patterns. Our objective was different in sense, that we want to model the problem in such a way, so that we can use existing optimized ATPG algorithms used in VLSI testing to address the problem. So we appropriately created the modified netlist along with corresponding fault files. The set containing all these is called K.

Now, randomly, a modified netlist is picked from set K, and using established ATPG tool, test vectors are generated, targeting the faults given in fault file. Implicitly it derived all the test vectors, which could activate all triggers, that are injected in the netlist. The ATPG tool also report two important thing along with test vectors. They are :

- *Aborted Fault List*
- *Redundant Fault List*

Aborted fault list denotes all those faults, for which the algorithms used in ATPG tool, failed to generate test vectors, under given constraint of time and other implicit parameters. **Redundant fault list** denotes all those faults, for which no test pattern exist to detect them. Now, the way we have modelled the trigger instances, into existing fault model, the report generated gives important information about nature of trigger and trojan instances. As, in our modified netlist, all the trigger instances are set as primary output, there is no role of observability. The requirement is about controllability of 1 at the trigger node. So, all the faults, which has been reported as redundant fault by the ATPG tool, conclude the fact that all the corresponding trigger instances are infeasible, i.e. the instances formed by the combination of activation nodes cannot be simultaneously activated to corresponding rare logic value. So, redundant fault list are infeasible trigger instances, which can be ignored, since they represent illegal state of the circuit. Considering aborted fault list, the ATPG tool failed to determine whether there exist a test vector for detecting the fault. We can classify these trigger instance, as hard-to-detect, and we will employ our second technique of test generation based on SAT methodology, which we will discuss in subsequent chapters.

Now, after generation of test vector for one randomly chose netlist k' from set K, let the test vectors be included into a master test vector set TA. Let the trigger instances corresponding to each aborted fault be included in set I, and redundant faults in set U. Remove netlist k' from set K. Now, using test vectors available in TA, perform fault simulation $\forall$ netlist $k' \in K$. The faults which are detectable corresponding to each modified netlist, are removed, and faultfile is updated with undetectable fault. In this fashion, we reduced the common test vectors, which would have generated, if parallel fault simulation was not performed. This step also ensured that in subsequent runs, the ATPG tool don't have to generate test vectors containing B faults, unless there were no common test vectors between the bin under test, and the bins already explored. After one run, all fault files, are updated corresponding to each modified netlist, each containing $\leq B$ faults. Again, the subset of modified netlist is chosen from K, containing the highest number of undetected faults, and out of them, one is chosen randomly, and ATPG tool is run for that netlist. The test vectors generated in included into the master testset TA, and the netlist is removed from K. The whole process is repeated again, unless set K becomes empty. in this way, we presented a divide-and-conquer based strategy to lessen search complexity, and testset length.

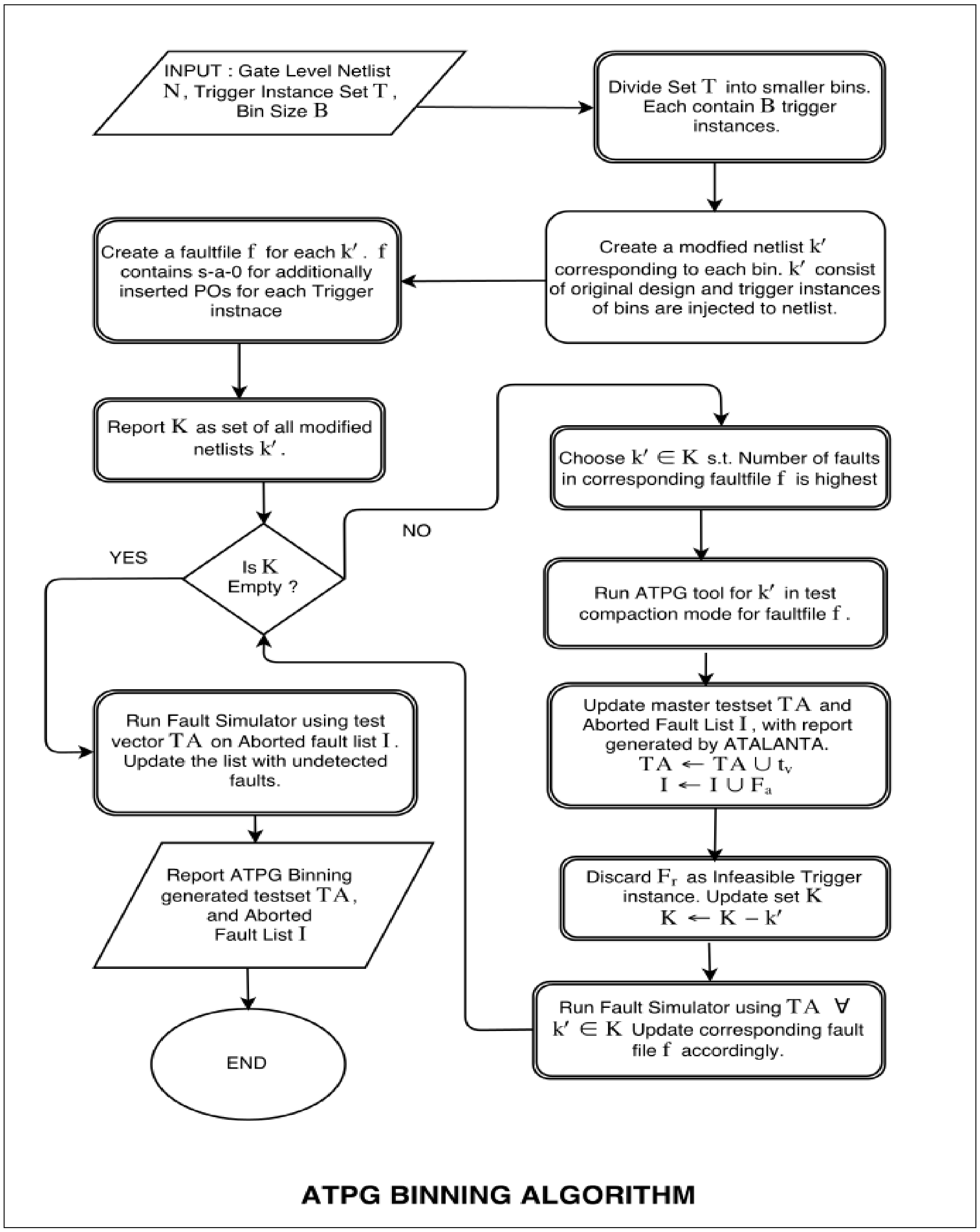

INPUT : Gate Level Netlist N, Trigger Instance Set T, Bin Size B
Divide Set T into smaller bins. Each contain B trigger instances.
Create a modfied netlist k' corresponding to each bin. k' consist of original design and trigger instances of bins are injected to netlist.
Create a faultfile f for each k'. f contains s-a-0 for additionally inserted POs for each Trigger instnace
Report K as set of all modified netlists k'.
Is K Empty ?
YES
NO
Choose $k' \in K$ s.t. Number of faults in corresponding faultfile f is highest
Run ATPG tool for k' in test compaction mode for faultfile f.
Update master testset TA and Aborted Fault List I, with report generated by ATALANTA. $TA \leftarrow TA \cup t_v$ $I \leftarrow I \cup F_a$
Discard F_r as Infeasible Trigger instance. Update set K $K \leftarrow K - k'$
Run Fault Simulator using TA $\forall$ $k' \in K$ Update corresponding fault file f accordingly.
Run Fault Simulator using test vector TA on Aborted fault list I. Update the list with undetected faults.
Report ATPG Binning generated testset TA, and Aborted Fault List I
END
ATPG BINNING ALGORITHM

Figure 4.2: Flowchart showing ATPG Binning Algorithm

4.2.1 Reporting Hard-to-detect trigger instances

Figure 4.2, shows a flowchart based presentation of ATPG Binning algorithm. After set K becomes empty, master testset TA is reported containing all minimal test vectors generated by the algorithm. I contains all the trigger instances of the aborted faults generated during the ATPG run. Now, it may happen, that the test vectors generated in later stage, might cover some of the aborted faults that were generated in previous rounds of ATPG Binning. So, a freshly modified netlist k' is created as previously defined, containing all trigger instances from set I. Using fault simulator, this modified netlist k' is simulated with master testset TA. All detectable faults are removed from set I. Now, set I consist of hard-to-detect faults, which may or may not have a test vector. We will present our second SAT based methodology in next chapter which would be used to generate test vectors for the instances present in set I.

Chapter 5

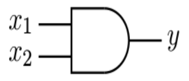
SATBiST Framework : SAT-Based Test Generation Method

The problem of "Boolean Satisfiability", also well known as SAT-problem, is one of the well known NP-complete problem in Computer Science. Basically SAT-problem is defined as "Given a boolean function $F(x_1, x_2, x_3 \ldots. x_n)$, does there exist an assignment of boolean logic to variables $x_1, x_2, x_3 \ldots. x_n$, such that the function F evaluates to $TRUE$? ". If such as assignment exist, then it is called *satisfying assignment*, and function F is called *satisfiable*, otherwise, *unsatisfiable*. A SAT-solver is a tool which solves satisfiability problem, and determine whether a boolean function is satisfiable or not. Here, the variables associated are boolean variable, i.e. they can take value of either 0, or 1. The most common logical operators that are acceptable in SAT formulae are described below.

- AND($\bigwedge$)
- OR($\bigvee$)
- NEGATION ($\sim$)
- IMPLICATION ($\rightarrow$)
- EQUIVALENCE($\longleftrightarrow$)

There are also other logical operators used in SAT formulae. In practice, SAT-solver accept the format of DIMACS CNF, where CNF stands for Conjunctive Normal Form. In a CNF formula, a *literal* is an instance of a variable x, or its negation, $\bar{x}$. A *clause* is *disjunction* ($\vee$) of one or more literals. A boolean formula is said to be in CNF format, if and only if it is expressed as *conjunction* ($\wedge$) of clauses. Today, various efficient SAT-solver tools are available, which implements good heuristics to solve the SAT-problem in polynomial time in average case. There are efficient polynomial time algorithms which can convert arbitrary

x_1	x_2	y	χ_{AND}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$$\chi_{\text{AND}}(x_1, x_2, y) = (x_1 + \bar{y})(x_2 + \bar{y})(\bar{x}_1 + \bar{x}_2 + y)$$

Figure 5.1: Truth table representation and CNF formula of a two-input AND gate

boolean formulae into CNF format. Logical gate level netlist can also be converted into CNF boolean formulae, which will be shown in subsequent sections.

5.1 Transforming Circuits into CNF

In this section, we present the technique to map a gate level netlist to an equivalent SAT instance. This technique is popularly known as Tseytin transformation. In a gate level netlist, the logic value assigned to internal nodes are constrained by the gates attached to those lines. The output of an AND gate will be 1, only when all of its inputs are set to logic 1. Similarly, in OR gate, the output will be set to logic 0, when all of the input values are set to logic 0, and so on. Given any logic circuit, an equivalent CNF formula can be constructed which expresses the same constraints. This can be easily generated by writing clauses which impose the same restrictions of each gate in the circuit, and then taking the conjunction of all the clauses.

A CNF formula representing the constraints for a logic gate can be generated from its truth table and characteristic function. This can be mapped to a function $F(.)$, which evaluates to TRUE value, if the values assigned to the input and output of the logical gates are consistent, and 0 otherwise.The characteristic function of fundamental gates can be transformed to a minimal CNF formula by means of K-Map or Karnaugh map, or using Quine-McClusky technique. For example, in figure 5.1, the function for an AND gate with inputs x_1 and x_2 and output y is shown, which can be easily written in CNF format.

This process can be repeated for any type of fundamental gates, including multi-input and multi-output gates. The construction will be polynomial in the number of inputs and outputs, as long as "counting"-type gates are assumed to have a fixed maximum number of inputs. In the table presented below, the CNF formulas for all gate types which have been used in this thesis have been presented. From the table, it can be easily concluded, that the CNF for gates like NAND and NOR, can be made from their non-inverting counterparts, by negation of the literal which is output of the gate.

Gate	Function	CNF Formula
AND	$y = x_1.x_2.x_3..x_n$	$\left(\prod_{i=1}^{n}(x_i + \bar{y})\right)\left(\sum_{i=1}^{n}(\bar{x_i} + y)\right)$
NAND	$y = \overline{x_1.x_2.x_3..x_n}$	$\left(\prod_{i=1}^{n}(x_i + y)\right)\left(\sum_{i=1}^{n}(\bar{x_i} + \bar{y})\right)$
OR	$y = x_1 + x_2 + x_3 + .. + x_n$	$\left(\prod_{i=1}^{n}(\bar{x_i} + y)\right)\left(\sum_{i=1}^{n}(x_i + \bar{y})\right)$
NOR	$y = \overline{x_1 + x_2 + x_3 + .. + x_n}$	$\left(\prod_{i=1}^{n}(\bar{x_i} + \bar{y})\right)\left(\sum_{i=1}^{n}(x_i + y)\right)$
XOR	$y = x_1 \oplus x_2$	$\left(\bar{x_1} + \bar{x_2} + \bar{y}\right)\left(x_1 + x_2 + \bar{y}\right)\left(x_1 + \bar{x_2} + y\right)\left(\bar{x_1} + x_2 + y\right)$
XNOR	$y = \overline{x_1 \oplus x_2}$	$\left(x_1 + x_2 + y\right)\left(\bar{x_1} + x_2 + \bar{y}\right)\left(\bar{x_1} + \bar{x_2} + y\right)\left(x_1 + \bar{x_2} + \bar{y}\right)$
NOT	$y = \bar{x}$	$\left(x + y\right)\left(\bar{x} + \bar{y}\right)$
BUFFER	$y = x$	$\left(x + \bar{y}\right)\left(\bar{x} + y\right)$

Table 5.1: CNF formulation of boolean logic gates

5.2 SAT Based Test generation methodology

In previous section, it has been shown, how to convert boolean logic gates into its equivalent CNF formulation. Now, given a design, from gate level netlist of the circuit, we can easily get the CNF formulation of the whole circuit, by mapping each gates input/output to its equivalent CNF clauses. From previous chapters, we know that, we tried to cover all feasible trigger instances. The hard-to-detect trigger instances are reported by set I. Now, the original netlist N, is taken, and Tsyetin transformation is applied to get the equivalent SAT formulation of a given netlist. Let this be called C. Now, for each trigger instance, generate the CNF for the ANDing of activation nodes of trigger instance, and add the clause obtained to original SAT instance C. Let the SAT instance, we get after injection of trigger be called C'. Now, using efficient SAT solvers, we check for the satisfiable instnace of C'. If there exist such instance, then the boolean values assigned to the input lines are taken, and added to testset $\text{TEST}_{\text{reduced}}$. In this fashion, for every trigger instance present in set I, test vector are generated using SAT solver and added to testset $\text{TEST}_{\text{reduced}}$. In the end, $\text{TEST}_{\text{reduced}}$ is merged with the test vectors generated by ATPG Binning TA, and minimal testset is reported, covering all feasible trigger instance. Note that, as we have considered the trojan instance, where the payload is Primary Output, the test vectors, which activate the trigger instance, would also be able to cover trojan instances, which we have considered.

The framework, thus able to derive a minimal test pattern, within considerable CPU-time, providing cent-percent coverage. The detailed algorithm has been presented below.

Algorithm 3 TEST GENERATION WITH SAT SOLVER

```
Input:
    $N$ : Gate level Netlist
    $TA$ : Testset $TA$ generated by ATPG Binning
    $I$ : Aborted Trigger Instance Set
Output: $TEST_{\text{reduced}}$ : Reduced Testset covering all possible Trigger and Trojan as mentioned in Section-I
 1: Read netlist $N$, do Tseytin transformation and generate CNF of netlist, $C$.
 2: for each trigger instance $t \in I$ do
 3:     Generate CNF for trigger instance $t$, $C'$.
 4:     $C' \leftarrow C \bigwedge C'$
 5:     Check for satisfiable instance for $C'$, through SAT Solver.
 6:     Report Test vector of SAT instance, $t_{\text{SAT}}$
 7:     $TEST_{\text{reduced}} \leftarrow TEST_{\text{reduced}} \cup t_{\text{SAT}}$
 8: end for
 9: $TEST_{\text{reduced}} \leftarrow TEST_{\text{reduced}} \cup TA$
10: Report Reduced Testset $TEST_{\text{reduced}}$.
```

Chapter 6

Experimental Results and Implementation

In this section, we present experimental results of our implementation on standard ISCAS85 [45] combinational circuits and ISCAS89 [46] sequential benchmark circuits.

6.1 Implementation

The whole SATBiST framework, has been developed with the following intentions :

- The test generation framework can be used alongside multiple parameter side channel analysis, which is capable of detecting trojans, modelled on the philosophy of activation under extremely rare conditions. The test based methodology would be able to detect smaller trojans, with Q value less than 10.
- The test generation framework can be modelled in such a way, that already existing good ATPG algorithms can be used to derive test patterns specifically for trojan detection.
- The test generation framework provides exhaustive coverage, ensuring greater reliability of the design.
- The test generation framework can be easily integrable into existing tools, without much effort.

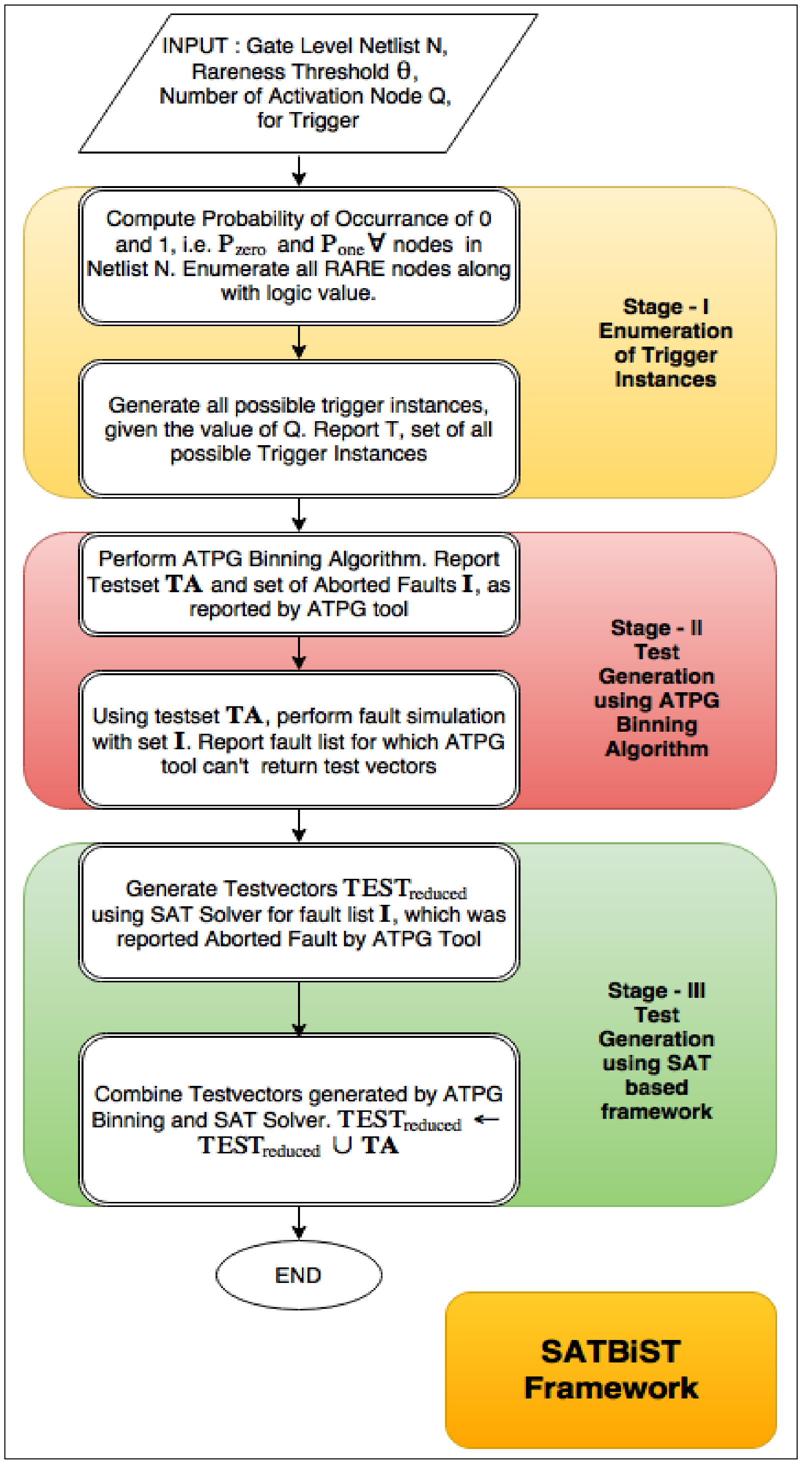

INPUT : Gate Level Netlist N, Rareness Threshold θ, Number of Activation Node Q, for Trigger
Compute Probability of Occurrance of 0 and 1, i.e. P_{zero} and P_{one} ∀ nodes in Netlist N. Enumerate all RARE nodes along with logic value.
Generate all possible trigger instances, given the value of Q. Report T, set of all possible Trigger Instances
Stage - I Enumeration of Trigger Instances
Perform ATPG Binning Algorithm. Report Testset **TA** and set of Aborted Faults **I**, as reported by ATPG tool
Using testset **TA**, perform fault simulation with set **I**. Report fault list for which ATPG tool can't return test vectors
Stage - II Test Generation using ATPG Binning Algorithm
Generate Testvectors $\mathbf{TEST}_{reduced}$ using SAT Solver for fault list **I**, which was reported Aborted Fault by ATPG Tool
Combine Testvectors generated by ATPG Binning and SAT Solver. $\mathbf{TEST}_{reduced} \leftarrow \mathbf{TEST}_{reduced} \cup \mathbf{TA}$
Stage - III Test Generation using SAT based framework
END
SATBiST Framework

Figure 6.1: SATBiST Framework

6.2 Results

We have run the simulation experiment on ISCAS85 [45] combinational circuits, and ISCAS89 [46] sequential circuits in full scan mode. The whole framework is coded in C++ and python, and experiments were done on an Intel core i5 processor, 3Ghz Clock and 4GB RAM with Linux OS. We have a tool named Transition Probability Calculator (TPC) [52], which gives an approximate value of testability measures, like controllability and observability of a node. TPC internally uses Synopsys 90nm Educational Development Kit(EDK) technology library [54]. The pre synthesized verilog code is synthesized using Synopsys Design Compiler [53], and using TPC, the probability values, $\mathrm{P_{zero}}$ and $\mathrm{P_{one}}$ is determined for all circuit nodes.

We select the value of rareness threshold $\theta = 0.1$, for combinational circuits and 0.01 for sequential circuits. The number (q) of activation nodes present in the given instance is assumed to be three. This can later be extended to values less than 10. The value $q = 3$ is taken, to show a comparative analysis of existing state-of-the-art technique for test generation of hardware trojan detection. The θ values are generally chosen in such way, that probable activation nodes constitute $5-10\%$ of total nodes present in the netlist. The list of activation nodes, containing tuples (i, L), is fed to program *gen-TrojanComb*, which generates all possible combinations of trigger instances. Next, the design and the trigger instances are provided as inputs to the program *TrojanInjection*, which generates the modified netlist, considering B trigger instances at a time. We have set the value of $B =$ 2000, for use in the ATPG tool, ATALANTA [50] in default test-compaction mode. The test set thus produced can activate 2000 trigger instances and the corresponding Trojan instances that corrupt POs directly. We now use HOPE [51] fault simulator to identify those trigger instances in other bins that are activated by the present test set, and update the fault list in each bin accordingly. The binning-algorithm terminates when all the modified netlists are covered. The faults, reported redundant by ATALANTA, denote false trigger combinations, and are deleted from the list. For each of the aborted faults left by ATALANTA, the trigger combination is fed to the *createSATInstance* tool. The SAT-formulation for the trigger instance thus produced along is fed to the SAT-solver, zChaff [49] along with the netlist. If a solution is found, it is added to the test set.

The results presented in Table 1 show the trigger instances for initial set-parameters. For sequential circuits, the trigger instances are in the range of hundreds of millions and we have presented the result only for 1 million instances. Table 2 draws a comparative analysis of the proposed methodology with MERO [31]. Since we have assumed that the payload only affects primary outputs, we have focussed only on generating tests that activates various trigger combinations. The results show significant improvements in terms of Trojan-coverage and the size of test sets over previous results.

Bench-mark Circuits	No. of Activation Nodes	No. of Possible Trigger Instances	Feasible Trigger Instances	Testset Length			CPU Time (in seconds)
				Testset generated by ATPG Binning	Testset generated by SAT	Total	
c432	40	9880	9648	236	0	236	0.377
c499	48	17296	32	32	0	32	1.69
c880	62	37280	36211	91	0	91	2.799
c1355	112	227920	224	84	0	84	94.37
c1908	65	43680	39747	161	1140	1301	1149.05
c2670	67	47905	25568	448	0	448	12.002
c3540	196	1235780	359811	2039	3937	5976	16308.002
c5315	176	893200	725814	2353	5211	7564	20712.658
c7552	232	2054360	1734842	11929	59621	71550	96398.282
s15850	748	1000000	402421	6425	323	6748	90258.39
s38417	1254	1000000	943951	24501	64	24565	223210.89

Table 6.1: SATBiST test patterns with $\theta = 0.1$ (Combinational) and 0.01 (Sequential) and $q = 3$. For Sequential circuits, 1 million instances are taken to show the order and magnitude of test length and CPU time, compared to previous techniques.

Benchmark Circuits	MERO Patterns		SATBiST Patterns	
	Trigger Instances	Testset length	Trigger Instances	Testset length
c3540	100000	15207	1235780	5976
c5315	100000	12429	893200	7564
c7552	100000	14785	2054360	71550
s15850	10000	35112	1000000	6748
s38417	10000	51885	1000000	24565

Table 6.2: Comparative analysis of MERO Test patterns and SATBiST test patterns with $\theta = 0.1$ (Combinational) and 0.01 (Sequential) and $q = 3$. For Sequential circuits, 1 million instances are taken to show the order and magnitude of test length and CPU time, compared to previous techniques. SATBiST Patterns provide cent percent coverage over all feasible trigger instances, compared to MERO patterns.

Chapter 7

Conclusion and Future Work

In this work, we have proposed a novel strategy for based on logic testing based approach for trojan detection. Experimental results show promising improvements in terms of scalability, CPU time, testset length and trigger coverage. With hardware trojan detection gaining widespread attention in hardware security community in recent times, we believe our work will have interesting applications.

In this work we have restricted our analysis to the detection and cent percent coverage of all possible trigger instances. We have shown here how we can exploit the existing good ATPG tools, algorithms and SAT-solvers, and make a scalable framework which is able to provide exhaustive coverage over trojan search space.

As our future work, we are extending our research to report a minimal testset where internal nodes of design can be possible payload of trojan. In particular, we are interested to investigate, the effectiveness of test vectors, about their trigger coverage, and ability to cover stuck-at faults in the circuit. We would like to apply a good heuristic of selecting a test vector, that would be able to provide maximum trigger coverage and maximum stuck-at fault coverage. In our problem, the trigger controllability has been successfully achieved. We need to see the observability of the payload, which can be any internal node of a circuit. We believe, that the framework developed with this inclusion, would be able to provide more coverage to hard-to-trigger trojan patterns.

Chapter 8

Disseminations out of this work

- A. B. Chowdhury, A. Banerjee, and B. B. Bhattacharya, “HARDWRE Trojan : threats and Countermeasures”. Presentation at Technical Review Committee Meet, DRDO funded Project “Hardware Trojan Research”, 2016.
- A. B. Chowdhury, A. Banerjee, and B. B. Bhattacharya, “SATBiST : Scalable ATPG Binning and SAT Based approach to Hardware Trojan Detection” under review *25th Asian Test Symposium, 2016.*

Bibliography

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

[2] Tehranipoor, Mohammad, and Farinaz Koushanfar. "A survey of hardware Trojan taxonomy and detection." *IEEE Design and Test of Computers* 27, no. 1 (2010): 10-25.

[3] Beaumont, Mark, Bradley Hopkins, and Tristan Newby. "Hardware trojans-prevention, detection, countermeasures (a literature review)." No. DSTO-TN-1012. *DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA) COMMAND CONTROL COMMUNICATIONS AND INTELLIGENCE DIV, 2011.*

[4] Chakraborty, Rajat Subhra, Seetharam Narasimhan, and Swarup Bhunia. "Hardware Trojan: Threats and emerging solutions." In *High Level Design Validation and Test Workshop, 2009.* HLDVT 2009. IEEE International, pp. 166-171. IEEE, 2009.

[5] Bhunia, Swarup, Michael S. Hsiao, Mainak Banga, and Seetharam Narasimhan. "Hardware Trojan attacks: threat analysis and countermeasures." *Proceedings of the IEEE 102, no. 8 (2014)*: 1229-1247.

[6] Tehranipoor, Mohammad, and Cliff Wang. "Introduction to hardware security and trust". *Springer Science and Business Media*, 2011.

[7] Bhunia, Swarup, Miron Abramovici, Dakshi Agrawal, Paul Bradley, Michael S. Hsiao, Jim Plusquellic, and Mohammad Tehranipoor. "Protection Against Hardware Trojan Attacks: Towards a Comprehensive Solution." *IEEE Design and Test 30*, no. 3 (2013): 6-17.

[8] Karri, Ramesh, Jeyavijayan Rajendran, Kurt Rosenfeld, and Mohammad Tehranipoor. "Trustworthy hardware." *Computer 43*, no. 10 (2010): 39-46.

[9] Wolff, Francis, Chris Papachristou, Swarup Bhunia, and Rajat S. Chakraborty. "Towards Trojan-free trusted ICs: Problem analysis and detection scheme." In Proceedings of the conference on *Design, automation and test in Europe*, pp. 1362-1365. ACM, 2008.

[10] Rostami, Masoud, Farinaz Koushanfar, Jeyavijayan Rajendran, and Ramesh Karri. "Hardware security: Threat models and metrics." In Proceedings of the *International Conference on Computer-Aided Design*, pp. 819-823. IEEE Press, 2013.

[11] Jacob, Nisha, Dominik Merli, Johann Heyszl, and Georg Sigl. "Hardware Trojans: current challenges and approaches." *IET Computers and Digital Techniques* 8, no. 6 (2014): 264-273.

[12] Li, He, Qiang Liu, Jiliang Zhang, and Yongqiang Lyu. "A Survey of Hardware Trojan Detection, Diagnosis and Prevention." In 2015 14th *International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*, pp. 173-180. IEEE, 2015.

[13] Dunbar, Carson, and Gang Qu. "Designing trusted embedded systems from finite state machines." *ACM Transactions on Embedded Computing Systems (TECS) 13, no. 5s (2014): 153.*

[14] Zhang, Xuehui, Kan Xiao, Mohammad Tehranipoor, Jeyavijayan Rajendran, and Ramesh Karri. "A study on the effectiveness of Trojan detection techniques using a red team blue team approach." In *VLSI Test Symposium (VTS), 2013 IEEE 31st, pp. 1-3. IEEE, 2013.*

[15] Agrawal, Dakshi, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. "Trojan detection using IC fingerprinting." In *2007 IEEE Symposium on Security and Privacy (SP'07), pp. 296-310. IEEE, 2007.*

[16] Wang, Xiaoxiao, Hassan Salmani, Mohammad Tehranipoor, and Jim Plusquellic. "Hardware Trojan detection and isolation using current integration and localized current analysis." In *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pp. 87-95. IEEE, 2008.

[17] Narasimhan, Seetharam, Rajat Subhra Chakraborty, Dongdong Du, Somnath Paul, Francis G. Wolff, Christos A. Papachristou, Kaushik Roy, and Swarup Bhunia. "Multiple-Parameter Side-Channel Analysis: A Non-invasive Hardware Trojan Detection Approach." In *HOST*, pp. 13-18. 2010.

[18] Jin, Yier, and Yiorgos Makris. "Hardware Trojan detection using path delay fingerprint." In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pp. 51-57. IEEE, 2008.

[19] Wang, Xiaoxiao, Hassan Salmani, Mohammad Tehranipoor, and Jim Plusquellic. "Hardware Trojan detection and isolation using current integration and localized current analysis." In *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pp. 87-95. IEEE, 2008.

[20] Aarestad, Jim, Dhruva Acharyya, Reza Rad, and Jim Plusquellic. "Detecting trojans through leakage current analysis using multiple supply pad s." *IEEE Transactions on information forensics and security 5, no. 4 (2010)*: 893-904.

[21] Potkonjak, Miodrag, Ani Nahapetian, Michael Nelson, and Tammara Massey. "Hardware Trojan horse detection using gate-level characterization." In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pp. 688-693. IEEE, 2009.

[22] Narasimhan, Seetharam, Xinmu Wang, Dongdong Du, Rajat Subhra Chakraborty, and Swarup Bhunia. "TeSR: A robust temporal self-referencing approach for hardware Trojan detection." In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pp. 71-74. IEEE, 2011.

[23] Rad, Reza M., Xiaoxiao Wang, Mohammad Tehranipoor, and Jim Plusquellic. "Power supply signal calibration techniques for improving detection resolution to hardware Trojans." In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 632-639. IEEE Press, 2008.

[24] Rad, Reza, Jim Plusquellic, and Mohammad Tehranipoor. "Sensitivity analysis to hardware Trojans using power supply transient signals." In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pp. 3-7. IEEE, 2008.

[25] Salmani, Hassan, and Mohammad Tehranipoor. "Layout-aware switching activity localization to enhance hardware Trojan detection." *IEEE Transactions on Information Forensics and Security 7, no. 1 (2012)*: 76-87.

[26] Banga, Mainak, and Michael S. Hsiao. "A region based approach for the identification of hardware Trojans." In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop* on, pp. 40-47. IEEE, 2008.

[27] Chakraborty, Rajat Subhra, and Swarup Bhunia. "Security against hardware Trojan through a novel application of design obfuscation." In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pp. 113-116. ACM, 2009.

[28] Bao, Chongxi, Domenic Forte, and Ankur Srivastava. "On application of one-class SVM to reverse engineering-based hardware Trojan detection." In *Fifteenth International Symposium on Quality Electronic Design, pp. 47-54. IEEE, 2014.*

[29] Salmani, Hassan, Mohammad Tehranipoor, and Jim Plusquellic. "New design strategy for improving hardware Trojan detection and reducing Trojan activation time." In *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on*, pp. 66-73. IEEE, 2009.

[30] Salmani, Hassan, Mohammad Tehranipoor, and Jim Plusquellic. "A novel technique for improving hardware trojan detection and reducing trojan activation time." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 20*, no. 1 (2012): 112-125.

[31] Chakraborty, Rajat Subhra, Francis Wolff, Somnath Paul, Christos Papachristou, and Swarup Bhunia. "MERO: A statistical approach for hardware Trojan detection." In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pp. 396-410. Springer Berlin Heidelberg, 2009.

[32] Banga, Mainak, and Michael S. Hsiao. "Odette: A non-scan design-for-test methodology for trojan detection in ics." In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pp. 18-23. IEEE, 2011.

[33] Jin, Yier, Nathan Kupp, and Yiorgos Makris. "DFTT: Design for Trojan test." In *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference* on, pp. 1168-1171. IEEE, 2010.

[34] Saha, Sayandeep, Rajat Subhra Chakraborty, Srinivasa Shashank Nuthakki, and Debdeep Mukhopadhyay. "Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability." In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 577-596. *Springer Berlin Heidelberg, 2015.*

[35] Banga, Mainak, Maheshwar Chandrasekar, Lei Fang, and Michael S. Hsiao. "Guided test generation for isolation and detection of embedded trojans in ics." In *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pp. 363-366. ACM, 2008.

[36] Banga, Mainak, and Michael S. Hsiao. "A novel sustained vector technique for the detection of hardware Trojans." In *2009 22nd International Conference on VLSI Design*, pp. 327-332. IEEE, 2009.

[37] Xiao, Kan, Domenic Forte, and Mohammed Tehranipoor. "A novel built-in self-authentication technique to prevent inserting hardware Trojans." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 33, no. 12 (2014)*: 1778-1791.

[38] Waksman, Adam, Matthew Suozzo, and Simha Sethumadhavan. "FANCI: identification of stealthy malicious logic using boolean functional analysis." In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security*, pp. 697-708. ACM, 2013.

[39] Zhang, Jie, Feng Yuan, Linxiao Wei, Yannan Liu, and Qiang Xu. "VeriTrust: verification for hardware trust." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34*, no. 7 (2015): 1148-1161.

[40] Zhang, Jie, Feng Yuan, and Qiang Xu. "Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans." In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 153-166. ACM, 2014.

[41] Hicks, Matthew, Murph Finnicum, Samuel T. King, Milo MK Martin, and Jonathan M. Smith. "Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically." In *IEEE Symposium on Security and Privacy*, pp. 159-172. 2010.

[42] Yang, Kaiyuan, Matthew Hicks, Qing Dong, Todd Austin, and Dennis Sylvester. "A2: Analog malicious hardware." (2016): 2016.

[43] Salmani, Hassan, and Mohammed Tehranipoor. "Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level." In *2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pp. 190-195. IEEE, 2013.

[44] Jin, Yier, Dzmitry Maliuk, and Yiorgos Makris. "Post-deployment trust evaluation in wireless cryptographic ICs." In *2012 Design, Automation and Test in Europe Conference & Exhibition (DATE), pp. 965-970. IEEE, 2012.*

[45] Brglez, Franc. "Neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN." In *Special session on ATPG and fault simulation, Proc. IEEE Int. Symp. Circuits and Systems*, June 1985, pp. 663-698. 1985.

[46] Brglez, Franc, David Bryan, and Krzysztof Kozminski. "Combinational profiles of sequential benchmark circuits." In *Circuits and Systems, 1989., IEEE International Symposium on*, pp. 1929-1934. IEEE, 1989.

[47] Eggergluss, Stephan, and Rolf Drechsler. Improving test pattern compactness in SAT-based ATPG." In *16th Asian Test Symposium (ATS 2007)*, pp. 445-452. IEEE, 2007.

[48] Shi, Junhao, Grschwin Fey, Rolf Drechsler, Andreas Glowatz, Friedrich Hapke, and Jurgen Schloffel. "PASSAT: efficient SAT-based test pattern generation for industrial circuits." In *IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design (ISVLSI'05)*, pp. 212-217. IEEE, 2005.

[49] Fu, Zhaohui, Yogesh Marhajan, and Sharad Malik. "Zchaff sat solver." 2012-01-31]. *http:///www. princeton, edu/chaff (2004).*

[50] Lee, H. K., and D. S. Ha. "Atalanta: An efficient ATPG for combinational circuits." *Deptartment of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, Technical Report (1993).*

[51] Lee, Hyung Ki, and Dong Sam Ha. "HOPE: An efficient parallel fault simulator for synchronous sequential circuits." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15*, no. 9 (1996): 1048-1058.

[52] Salmani, Hassan and Mohammad Tehranipoor. " Transition Probability Calculation (TPC)" *https://www.trust-hub.org/index.php?option=comcontentview=articleid=75Itemid=83.*

[53] " Synopsys Design Compiler v2014.03". *http://www.synopsys.com/Tools/Implementation /RTLSynthesis/DesignCompiler/Pages/default.aspx.*

[54] "Synopsys 90nm Technoloy Library: Educational Development Kit (EDK)". *http://www.synopsys.com/Community/UniversityProgram/Pages/Library.aspx.*