J Hardw Syst Secur (2017) 1:85–102
DOI 10.1007/s41635-017-0001-6

Benchmarking of Hardware Trojans and Maliciously Affected Circuits

Bicky Shakya[1] · Tony He[1] · Hassan Salmani[2] · Domenic Forte[1] · Swarup Bhunia[1] · Mark Tehranipoor[1]

Received: 27 July 2016 / Accepted: 8 March 2017 / Published online: 10 April 2017

Abstract Research in the field of hardware Trojans has seen significant growth in the past decade. However, standard benchmarks to evaluate hardware Trojans and their detection are lacking. To this end, we have developed a suite of Trojans and 'trust benchmarks' (i.e., benchmark circuits with a hardware Trojan inserted in them) that can be used by researchers in the community to compare and contrast various Trojan detection techniques. In this paper, we present a comprehensive vulnerability analysis flow at various levels of abstraction of digital-design, that has been utilized to create these trust benchmarks. Further, we present a detailed evaluation of our benchmarks in terms of metrics such as Trojan detectability, and in the context of different attack models. Finally, we discuss future work such as automatic Trojan insertion into any arbitrary circuit.

Keywords Hardware Trojan · Benchmarks · Hardware security

✉ Bicky Shakya
bshakya@ufl.edu

Tony He
tonyhe@ufl.edu

Hassan Salmani
hassan.salmani@howard.edu

Domenic Forte
dforte@ece.ufl.edu

Swarup Bhunia
swarup@ece.ufl.edu

Mark Tehranipoor
tehranipoor@ece.ufl.edu

[1] ECE Department, University of Florida, Gainesville, FL 32611, USA

[2] ECE Department, Howard University, Washington, DC 20059, USA

1 Introduction

The past decade has seen great advancement in research for hardware Trojan detection and prevention. Many techniques have been proposed for Trojan detection at several stages in the supply chain. However, such techniques, while bearing merits of their own, have several shortcomings. We highlight some of these shortcomings below.

- **Ad-hoc Trojans:** For each detection technique, researchers have mostly resorted to using 'home-grown' hardware Trojans to demonstrate the advantages and accuracy of the proposed techniques. While such Trojans might be suited to a particular detection approach, the results might vary vastly when they are used in the context of other detection approaches. Due to this, when comparing results between different detection techniques, there is never a baseline for comparing the merits of one technique to another. Further, it is not clear if these Trojans satisfy the basic characteristic of a hardware Trojan, i.e. it must be able to bypass absolutely all commonly used manufacturing test methods such as functional, structural, fault-based tests etc.
- **Varying Assumptions:** The simulation/implementation environment and factors such as the amount of process variation allowed, difficulty of triggering

Trojans, Trojan switching activity, size of design (no. of gates, Trojan size) etc. also vary greatly from one technique to another. This further compounds the problem of comparing various Trojan detection schemes.

– **Ad-Hoc Metrics:** The metrics used for evaluating detection techniques have been mostly ad-hoc as well. Some researchers may choose to evaluate their technique in terms of an arbitrary percentage detection rate, some may present false positive/false negative rates and some may explain their results in terms of test coverage. While these techniques may be working with the same Trojan attack model, comparison between them becomes difficult with ad-hoc figures of merit.

Thus, there is a need for a unifying approach for hardware Trojan detection. In order to address this need, we have developed tools to assess the vulnerability of designs that can be exploited for insertion of various types of Trojans, as well as tools to evaluate the stealthiness or difficulty of detecting Trojans. Using these tools, we have designed an array of Trojans that have been carefully integrated into various circuits to create "trust benchmarks". Benchmark circuits are commonplace in many different fields. Various circuit benchmarking efforts such as the ITC '02 [1], ISCAS '85 [2] and '89 [3] benchmark sets have allowed comparative research in the field of modular SoC testing and combinational/sequential logic synthesis. Further, research fields such as multimedia systems, computer architecture, digital signal processing and machine intelligence have greatly benefited from their own set of benchmarks, which have allowed objective comparison of different techniques [4, 5]. This widespread use of benchmarks in various fields only emphasizes the need for specific benchmarks in the field of hardware Trojan detection. Our hope is that these benchmarks will allow researchers to implement their Trojan detection schemes and compare them to others on a level playing field.

The Trojans that we have developed are (i) not detectable by standard manufacturing tests; (ii) vary in size and distribution to fit different Trojan insertion scenarios; and (iii) are implemented at different levels of abstraction from RTL, netlist, to layout. As we have seen in Table 1, a large number of potential Trojans can be inserted at different levels in the supply chain. Taking this into consideration, we have developed trust benchmarks that cover each stage in the supply chain, and have been implemented in different platforms from microcontrollers, cryptographic IP cores to JTAG controllers. This can help the research community and industry to prioritize their efforts for developing defenses against hardware Trojans. A further merit of these trust benchmarks is that it will help in the reproducibility of results.

Table 1 Comprehensive adversarial models for hardware Trojans

Model	3PIP vendor	SoC developer	Foundry
A	Untrusted	Trusted	Trusted
B	Trusted	Trusted	Untrusted
C	Trusted	Untrusted	Trusted
D	Untrusted	Untrusted	Untrusted
E	Untrusted	Untrusted	Trusted
F	Untrusted	Trusted	Untrusted
G	Trusted	Untrusted	Untrusted

This is vital for transferring these detection techniques from research to real-world implementation. Further, each Trojan we have developed has been analyzed by concrete metrics, which establishes a sound basis for analyzing the hardness of detecting Trojans in each benchmark instance. Note that for these trust benchmarks, the Trojans have been manually inserted into the benchmark circuits after vulnerability analysis. We do not perform an automatic payload identification for any arbitrary circuit, which would be part of our future work (Section 7). This would aid in developing "Trojan benchmarks", which are custom-designed Trojan circuitry that can be inserted into any arbitrary circuit, as opposed to the specific benchmark circuits we have used for developing the trust benchmarks. Compared to [6] which only looked at gate-level benchmarks, this paper explores trust benchmarks and vulnerability analysis at different levels of abstraction (RTL, gate, layout, FPGA), introduces attack models for benchmark evaluation and thus, takes a holistic approach towards trust benchmark development.

The rest of the paper is organized as follows. In Sections 3 and 4, we will introduce a comprehensive Trojan taxonomy and review the benchmarks we have developed over the course of the past few years. In Section 5, we will describe the tools we have developed to perform vulnerability analysis on a design. In particular, we will talk about a vulnerability analysis flow that can determine which parts of a circuit are more susceptible to Trojan insertion, at the layout, gate and behavioral level. In Section 6, we will discuss the Trojan evaluation suite we have developed, which is based on the efficiency of test patterns in activating the Trojan and a Trojan's resiliency to side-channel analysis. The evaluation suite will help us to explain the trust benchmarks in terms of concrete metrics such as 'detectability'. In the same section, we will also present results on the trust benchmarks, based on our Trojan evaluation suite and the attack models we presented earlier. In Sections 7 and 8, we will provide remarks on future work for further development of trust benchmarks and conclude the paper.

2 Background

2.1 Hardware Trojans

A hardware Trojan is defined as a malicious, undesired and intentional modification made to an electronic circuit. Such a modification can potentially bring about a variety of effects [7], such as:

- **Change of functionality:** A hardware Trojan can alter the functionality of a circuit and cause it to perform malicious, unauthorized operations, such as bypassing of encryption algorithms, privilege escalation, denial of service etc.
- **Degradation of performance**: A hardware Trojan could also cause damage to the performance of an IC and cause it to fail, which could potentially jeopardize the (critical) system into which the IC is integrated. Such effects could be in the form of induced electromigration of wires by continuous DC stress, increase/decrease in path delay, fault injection etc.
- **Leakage of information**: Trojans could also undermine the security provided by cryptographic algorithms or directly leak any sensitive information handled by the IC. This could involve leakage of cryptographic keys or other sensitive information through debug or I/O ports, side-channels (delay, power) etc.

2.2 Adversarial/Attack Models

In order to accomplish one or more of the effects shown above, a malicious entity present in any given stage of the IC design/manufacturing process (Fig. 1) can insert the Trojan at various levels of abstraction. This brings up the need to define hardware Trojan in the context of different adversarial models. In Table 1 and below, we present a comprehensive list of adversarial models that show exactly when, where and how a Trojan can be inserted into an IC [8]. The unique sources of Trojans are 3PIP vendor, SoC integrator, and foundry. A Trojan can be inserted by one or more of these entities as follows:

- **Model A:** In this attack model, the third-party IP that a SoC developer or design house buys may contain hardware Trojans. This is a considerable threat in today's semiconductor design landscape, where SoCs are made by integrating several 3PIPs in order to reduce complexity, cost and shorten time-to-market. The effect and design of the Trojan can vary depending on whether the IP is soft (RTL-level), firm (netlist-level) or hard (GDSII).
- **Model B:** Attack model B relates to the threat of untrusted foundry/assembly. Since a foundry has access to all layers of the design, they can perform reverse-engineering of the design, in order to add/modify/delete gates and create Trojans in the design. This attack model is especially significant in today's horizontal semiconductor business model, where the design house has little to no control of the off-shore foundries.
- **Model C:** Attack model C relates to an untrusted design house. This can be either due to an untrusted EDA tool used by the design house or a rogue employee (malicious insider) that maliciously modifies the design.
- **Model D:** This attack model relates to the threat faced by most consumers or system integrators (e.g. PCB developers), who are forced to buy commercial off-the-shelf (COTS) in order to minimize costs. Since they have no control over any aspect of the design/manufacturing process, the threat of a Trojan insertion can occur at all three sources.
- **Model E:** In this attack model, all parties except the foundry are assumed to be trusted. This can be the case

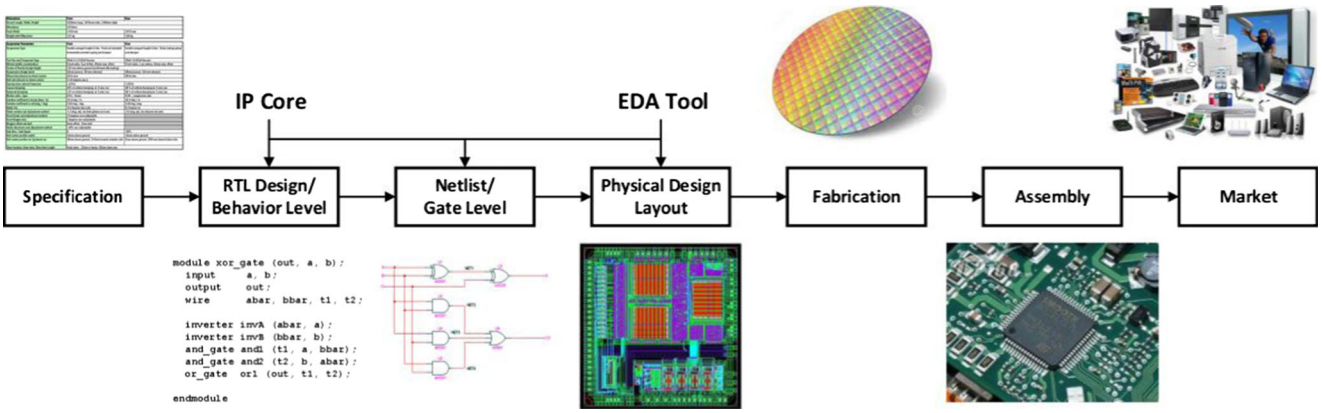

Fig. 1 Supply chain for IC production

when the foundry and the manufacturing process are trusted but the development process is not. This model can also account for cloned ICs, where malicious parties could reverse engineer a Trojan-free chip and create designs with Trojans inserted.

- **Model F:** This adversarial model applies to a majority of trusted design houses today who are forced to rely on untrusted 3PIPs and foundries.
- **Model G:** This attack model relates to companies who have designed their own proprietary IPs but need to rely on an untrusted design house and foundry to manufacture their final ICs.

2.3 Detection Techniques

With such an array of vulnerabilities from one or more untrusted entities, there is a pressing need to develop hardware Trojan detection techniques for assessing a design at various levels of abstraction. To address this need, the hardware security research community has proposed a plethora of Trojan detection techniques over the past decade. These efforts can be broadly separated into three categories.

- **Post-silicon detection** includes destructive and non destructive techniques to detect Trojans in manufactured chips. In destructive techniques, a fabricated chip is completely reverse-engineered layer-by-layer in order to reconstruct the design and compared with a Trojan-free or 'golden' design to detect Trojans. While such techniques offer high probability of Trojan detection, the time and cost required to perform reverse-engineering can be prohibitively high. Non-destructive techniques focus on detecting Trojans using functional tests or side-channel analysis. In *functional tests*, test-vectors are applied to the design and the responses/outputs are compared to the correct results to find anomalies (which could potentially be Trojans). However, a Trojan designer will, more likely than not, make sure that the Trojan is activated only under very rare conditions (such as an extremely rare test pattern), so that it can evade standard manufacturing tests. Added to this problem, modern designs may have a very high number of test patterns, making it impossible to conduct exhaustive functional tests. Prior work in Trojan detection has looked at techniques for generating test-patterns that can specifically target rarely activated nets [9–12]. However, the large number of states/inputs in modern designs limit the accuracy of these approaches, especially for Trojans that have extremely low trigger probability. In *side-channel analysis*, the impact of Trojans on circuit delay, transient current, leakage power, thermal profiles etc. are used for detection [13–19]. Most, if not all of these techniques require a golden model for comparison/detection, which might not always be available. Further, with the large process variation experienced at advanced nodes, such side-channel analysis based techniques might produce a significant amount of false positives/false negatives, limiting their applicability.
- **Pre-silicon detection** is necessary to detect Trojans that could have been inserted in 3PIP cores, by untrusted EDA tools, and/or by rogue employees in the design itself. Netlist-level IP cores can be tested by using the same functional testing techniques, as described above. For soft IP cores, various techniques such as code or structural analysis [20, 21] have been proposed, which analyze hardware description languages for redundant lines of code, conditional statements that rarely trigger etc. which could be possible locations where Trojans have been coded down. Formal verification has also been recently adapted for Trojan detection, where IP cores are tested by proof-checking/model-checking to make sure they perform the intended functionality and nothing else [22–25]. Such techniques are limited due to two issues. Firstly, most 3PIP blocks come as hard macros or in encrypted form, whose internal implementations are often inaccessible and can only be used as black-boxes. Secondly, for functional verification, there might always exist Trojans that could satisfy proof-checking constraints and evade detection.
- **Design for Trust** techniques are also necessary in order to make Trojan detection easier and/or Trojan insertion prohibitively difficult. Towards facilitating detection, techniques such as facilitating functional test by increasing the observability of nodes [26], increasing side-channel activity caused by Trojans [27] and run-time monitoring [28] have been proposed. On the other hand, in order to make Trojan insertion difficult, researchers have proposed techniques such as logic obfuscation to lock the functionality of the IC [29–31], functional filler cell insertion [32] for layout protection, IC camouflaging [33] and split-manufacturing [34, 35] to prevent reverse-engineering of the design. These design-for-trust techniques can potentially help to detect and prevent highly stealthy Trojans that can otherwise evade pre and post-silicon detection techniques, albeit at the cost of area, power and timing overhead.

2.4 Hardware Trojans in FPGA Designs

FPGAs are widely used today in an array of embedded applications ranging from telecommunications and data centers to missile guidance systems. Unfortunately, the

outsourcing of FPGA production and the use of untrusted third party IPs has also given rise to the threat of Trojan insertion in them. FPGA-based Trojans can be in the form of IP blocks (hard, soft or firm), which get loaded onto a generic FPGA fabric and cause malicious activity (denial-of-service, leakage etc.) on the system in which the FPGA is deployed. Such FPGA IP-based Trojans are more or less similar to their counterparts in an ASIC design flow (with the exception of layout-based Trojans, which are not applicable to FPGAs). However, Trojans that 'pre-exist' in an FPGA fabric and could potentially be inserted by an untrusted foundry or vendor pose unique threats and challenges of their own. FPGAs contain a large volume of reconfigurable logic in the form of lookup tables, block RAM and programmable interconnects, which can be used to realize any arbitrary sequential or combinational design. However, there might be a significant amount of reconfigurable logic open to a malicious party (e.g. the FPGA foundry or even the FPGA vendor) who can load a hardware Trojan and affect the FPGA-integrated system or compromise the IP loaded onto the FPGA. These FPGA device-specific hardware Trojans and their effects are explained in [36] and summarized below.

2.4.1 Activation Characteristic

Hardware Trojans in FPGAs can have activation characteristics similar to the ones described in Section 3.1 such as always-on or triggered. However, a unique characteristic of FPGA device-based hardware Trojans is that they can either be IP dependent or independent.

- *IP-dependent Trojans:* A malicious foundry or FPGA vendor may implement a hardware Trojan that can monitor the logic values of several LUTs in the FPGA fabric. Once triggered, such Trojans can corrupt other LUT values, load incorrect values into BRAMs or sabotage configuration cells. Since any arbitrary IP may be loaded onto the FPGA, the malicious foundry or vendor could distribute trigger LUTs throughout the FPGA so that the probability of the Trojan triggering and causing malfunction may increase.
- *IP-independent Trojans:* A malicious foundry or vendor may also implement a Trojan into an FPGA chip that is completely independent of the IP loaded onto it. Such Trojans can occupy a small portion of FPGA resources and malfunction IP-independent but critical FPGA resources such as digital clock managers (DCM). One potential mode of attack would be the Trojan increasing or decreasing the design clock frequency by manipulating the configuring SRAM cells of the DCM unit, which can cause failure in sequential circuits.

2.4.2 Payload Characteristics

FPGA device-based Trojans can also bring about unique malicious effects, such as causing malfunction of FPGA resources or leakage of the IP loaded onto the FPGA.

- *Malfunction:* Hardware Trojans in FPGA devices can either cause logical malfunction by corrupting LUT or SRAM values, thereby affecting the functionality of the implemented IP, or by causing physical damage to the FPGA device. For example, a triggered hardware Trojan could reprogram an I/O port set as an input, as an output while suppressing the configuration cells that prevent it from being programmed as such. This would cause a high short-circuit current to flow between the FPGA and the system it is connected to, thereby leading to physical device failure.
- *IP Leakage:* FPGAs today offer bitstream encryption capabilities in order to protect the IP loaded onto an FPGA device. However, such encryption only prevents a direct, unauthorized readback by software. A hardware Trojan may circumvent such protection by either leaking the decryption key or even the entire IP. The Trojan may tap the decryption key as it comes out of non-volatile memory, or the actual decrypted IP , which could then be exfiltrated either via covert side-channels (e.g. power traces) or through JTAG, USB or I/O ports.

3 Trojan Benchmarks

3.1 Trojan Benchmark Taxonomy

It is difficult to model a Trojan similar to faults in a circuit. Although both faults and Trojans can potentially cause errors in a circuit, one is based on random or systematic manufacturing defects, and the other is based on malicious intent. Defects can be modeled but intention cannot. Thus, to address this issue, we have developed a comprehensive Trojan taxonomy based on the vulnerabilities in the modern horizontal design and test processes and the opportunities adversaries may have at different levels of abstraction.

Over the past few years, there have been efforts to develop comprehensive hardware Trojan taxonomies based on Trojan implementation and effect [7]. We have further improved the taxonomies of the earlier works by including the physical characteristics of Trojans. Presented in Fig. 2, the Trojan taxonomy is broken down into the following categories:

Insertion Phase The design flow consists of several phases from determining design specification to its assembly and

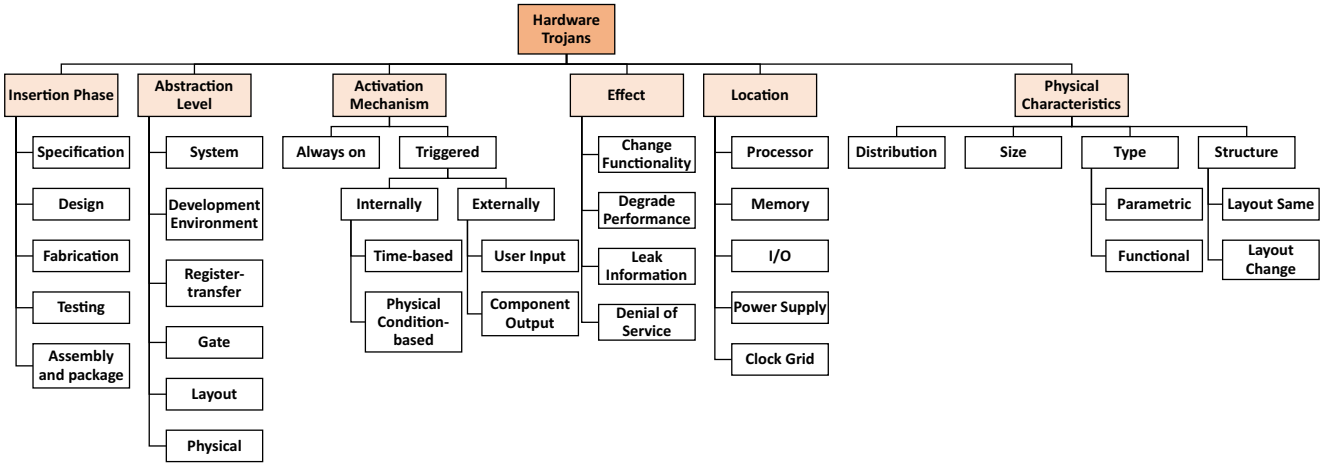

Fig. 2 A Comprehensive hardware Trojan taxonomy for Trojan benchmark development

packaging. Due to the globalization, circuit tampering can occur at different stages. For example, a Trojan can be realized by adding some extra gates to a circuit netlist at the design phase or by changing its masks during the fabrication step.

Abstraction Level The level of abstraction determines the control and flexibility an adversary may have on Trojan implementation. At the system level, a circuit is defined in terms of modules and the interconnections between them, limiting an adversary to the modules' interfaces and their interactions. On the other hand, all circuit components, their dimensions, and locations are determined at the physical level. A Trojan can be inserted in the white/dead spaces of circuit layout with the least impact on circuit characteristics.

Activation Mechanism Trojans may always function, or they can be conditionally activated. Always-on Trojans start as soon as their host designs are powered on, while conditional Trojans seek specific internal or external triggers to launch.

Effect Trojans can be characterized based on their effects. They may change a circuit's functionality, for example, by modifying the data path of the processor. Trojans can also reduce a circuit's performance or degrade its reliability by changing its physical parameters.

Location Every part of a circuit can potentially be subjected to Trojan insertion. A Trojan can be distributed over several regions or focused in one region. A Trojan can tamper with a processor to manipulate its controller or data path units. For example, on a printed circuit board (PCB) including several chips, an inserted Trojan on these chips' interfaces can disturb chip-to-chip communication.

Physical Characteristic Trojans can alter a circuit's physical characteristics, an assault that has many hardware manifestations. A Trojan can be a functional or parametric type. Functional Trojans are realized by the addition or deletion of transistors/gates, and parametric Trojans by modification to wire thickness or any other circuit parameter. Trojan cells can be distributed loosely or tightly in the physical layout in white spaces or spaces created by displacing the cells of a main circuit.

3.2 Sample Trojan Benchmarks

We have developed several Trojan benchmarks that have been inserted into a variety of benchmark circuits. For a given circuit, different Trojans can cause different effects, such as those shown in the effects taxonomy in Section 3.1. For example, a gate-level Trojan in a circuit may leak an internal net-value to the primary outputs. On the other hand, a different Trojan in the same circuit may inject erroneous values into the internal nets. Thus, for one given benchmark, we can have several different Trojans inserted into it. Keeping this in mind, we have developed a naming convention for each unique Trojan inserted in a benchmark circuit. Each of such 'Trojan benchmarks' is named as T_i, where i (a two-digit number) denotes a unique Trojan. For example, for the $s35932$ benchmark, the $T1$ Trojan benchmark maliciously activates the scan enable of the circuit and leaks an internal value through a test output pin. On the other hand, the $T2$ Trojan benchmark applies a dominant value to the same design in functional mode, thereby bypassing four gates of the main design. Note that $T1$ for one benchmark does not necessarily mean that the exact same Trojan appears

in a different benchmark with a Trojan benchmark labeled $T1$.

4 Trust Benchmarks

A "trust benchmark" is a benchmark circuit (generic circuits at the RTL, gate or layout level) which has Trojan(s) deliberately added to it at hard-to-detect, impactful and/or opportunistic locations (e.g. rare nodes, layout white-space etc.), for the purpose of comparing impacts of Trojans and the effectiveness of different Trojan detection techniques. Our initial efforts have focused on "static" trust benchmarks, which we define as those in which the location and size of the Trojan do not change. Our current trust benchmarks are available at http://www.trust-hub.org/taxonomy.

Each benchmark comes ready with documentation, that lists down important features of the trust benchmark such as trigger probability (for gate/layout level Trojans), exact effect of the Trojan, input combination required to trigger Trojan (for RTL/gate level), Trojan-induced delay or capacitance etc, size of Trojan/overall circuit etc. Additionally, for some benchmarks, we have provided a 'golden model', i.e. a version of the same circuit without Trojans, which can be handy for analyzing the trust benchmarks in terms of different attack models (see Section 6.4). Finally, for most of the trust benchmarks, we have included two testbenches, one of which can be used with the golden model (for debugging/test purposes) and the other which can be used to trigger the Trojan. For RTL level trust benchmarks, the testbench is in the form of Verilog/VHDL testbenches that have the Trojan trigger specified. For netlist/gate level benchmarks, exact test patterns to trigger the Trojan are provided. Finally, the documentation for each trust benchmark contains the exact form and location of the inserted Trojan. For example, for RTL level Trojans, the part of the RTL code that implements the Trojan has been documented. For gate-level circuits, a snippet of the Trojan netlist has also been provided. We have disclosed the exact location and implementation of the Trojan to make it easier for researchers to present results in terms of detection accuracy. However, it should be noted that such information should only be used 'a posteriori', as taking into account the Trojan implementation and location beforehand might unfairly bias detection techniques. Lastly, we note that this is an ongoing effort, and we are continuously generating various trust benchmarks to cover the Trojan taxonomy and improve on existing ones. We encourage the community to submit Trojans to us as well for inclusion on the website.

In the following, we explain our naming convention for trust benchmarks. We then present some of the representative benchmarks from the lot of approximately a hundred benchmarks developed so far.

4.1 Benchmark Naming Convention

We have developed benchmarks of different sizes, different types (ASIC, microprocessor, etc.), and with different Trojans covering the taxonomy shown in Fig. 2. Moreover, a Trojan can be inserted in several circuits and also placed in different locations within each circuit. Further, it is possible to modify and update a Trojan to a new version over time. Based on the above, we develop the following naming convention to assign a unique name to each Trojan benchmark in a trust benchmark circuit:

DesignName-Tn#$

where

- **DesignName**: The name of the main design without a Trojan. There is no limit on the number of letters or characters for the design name.
- **Tn** (Trojan number): It is of a maximum two digits. Note that the same Trojan number in different designs does not represent the same Trojan.
- **#** (Placement number): The second to last digit indicates the different placement of the same Trojan in a circuit and ranges from 0 to 9.
- **$** (Version number): The last digit in a benchmark name indicates the version of the Trojan and ranges from 0 to 9. This is added as a feature in case a new version of the same Trojan with the same placement has been developed. The version number will differentiate the older version from the new one.

For example, MC8051-T1000 indicates that Trojan number 10 (T10) was inserted in the micro-controller 8051 (MC 8051) at the location number 0, and its version is 0. As another example, dma-T1020 means that Trojan number 10 (T10) was inserted in the DMA circuit at the location number 2 and its version is 0. As aforementioned, Trojan T10 in DMA is not necessarily the same as Trojan T10 in MC8051.

4.2 Sample Trust Benchmarks

In the following, we present some of the benchmarks with a brief description of their enclosed Trojan.

- **Insertion Phase - Fabrication:** Trojans can also be realized by adding/removing gates or changing the circuit layout during GDSII development, and the mask during fabrication.
  Sample Benchmark: EthernetMAC10GE-T710 contains a Trojan triggered by a combinational comparator circuit which seeks a specific 16-bit vector. The probability of Trojan activation in this case is 6.4271e-23. When the Trojan is triggered, its payload gains control over an internal signal in the circuit.

- **Abstraction Level - Layout:** Trojans can be realized by varying circuit mask, adding/removing gates, or changing gate and interconnect geometry to impact circuit reliability.
  Sample Benchmark: EthernetMAC10GE-T100 contains a Trojan on a critical path. The net fault_sm0/n80 is widened to increase coupling capacitance, enabling crosstalk.
- **Activation Mechanism - Triggered Externally:** Trojans become activated under certain external conditions, such as by an external enable input.

  Sample Benchmark: RS232-T1700 contains a Trojan triggered by a combinational comparator. The trigger input probability is 1.59e-7 and it is externally controlled. Whenever the Trojan gets triggered, its payload gains control over the output xmit_doneH signal.
- **Effect - Change Functionality:** After activation, a Trojan will change the functionality of a circuit.
  Sample Benchmark: RS232-T1200 contains a Trojan triggered by a sequential comparator with probability is 8.47e-11. Whenever the Trojan gets triggered, its payload gains control over the output xmit_doneH signal.
- **Location - Power Supply:** A Trojan can be placed in the chip power network.

  Sample Benchmark: EthernetMAC10GE-T400 is modified with narrow power lines in one part of the circuit layout.
- **Physical Characteristic - Parametric:** A Trojan can be realized by changing circuit parameters like wire thickness.

  Sample Benchmark: EthernetMAC10GE-T100 contains a Trojan on the critical path. Specifically, the net fault_sm0/n80 is widened.

Table 2 presents a complete list of trust benchmarks that have been developed so far. They are categorized based on the Trojan taxonomy. The number of trust benchmarks available for each type, along with the names of the main circuits/benchmarks the Trojans have been inserted into are also presented. For instance, the table shows that 25 Trojans are inserted at the gate-level, 51 at the register-level, and 12 at the layout level, under the row 'Abstraction Level'. As another example, the 'effect' row shows that 35 Trojans change circuit functionality, 3 degrade circuit performance, 24 leak information to the outside of a chip, and 34 will perform a denial-of-service attack when activated. Note that some benchmarks fall under more than one category. Currently, there are a total of 91 trust benchmarks on the Trust-Hub website.

5 Design Vulnerability Analysis

In order to create the Trojans and trust benchmarks, we have created a suite of tools that first evaluate a design for vulnerability to Trojan insertion. This analysis has been performed at the RTL, gate and layout levels. In each case, vulnerabilities such as rare events, transition probabilities, white spaces etc. have been extracted and subsequently exploited to insert the hardware Trojan circuitry as part of the trust benchmarks.

5.1 Vulnerability Analysis at the RTL Level

Due to the increased use of 3PIP in designs, it is also necessary to conduct vulnerability analysis at the behavioral or RTL level. Given a RTL IP, we need to carefully analyze statements that are rarely activated and check if certain signals are propagated to the primary outputs. A malicious 3PIP can use such features to create a behavioral Trojan that is difficult to trigger and whose effects cannot be observed at the outputs on regular functional testing. In order to quantify such vulnerabilities, we have proposed two broad metrics [20]:

- **Statement Hardness:** This metric analyzes the rarity of the conditions under which a statement executes in RTL code. This metric is quantified by $(\frac{U-L+1}{U_0-L_0+1})^{-1}$, where U, L are the upper and lower limits of the value range for a control signal (set by a conditional statement such as IF) and U_0, L_0 are the declared upper and lower limits of the value range for the same control signal (set by an assignment/declaration statement). This indicates that statements/control signals that are highly nested by multiple conditional statements are harder to execute and are the most likely targets for Trojan insertion at the behavioral level.
- **Observability:** This metric denotes the ease with which a signal can propagate to the primary output of a design. In order to quantify this metric, a data graph is constructed from the RTL code. Each node in the graph represents a control signal in the RTL code, and the edges represent the flow of data in the code. Each signal (with respect to a destination node) is assigned a '0′ for observability if its destination node is not a primary output. If a signal is considered with respect to a destination node that serves as a primary output, it is assigned a observability figure equal to the sum of the weight of the assignment statements where the signal is assigned to the destination node.

Using these two metrics, the *b*01 benchmark from the ITC'99 benchmark set was analyzed. The maximum statement hardness for this benchmark was 64, with a maximum observability of 0.5. In contrast, the *b*05 benchmark, a

Table 2 Trust benchmark details

Category	Trojans		Main Circuits
	Trojan Type	# of Trust Benchmarks	
Insertion Phase	Specification	0	–
	Design	80	AES, BasicRSA, EthernetMAC10GE, MC8051, PIC16F84, RS232, s15850, s35932, s38417, s38584, vga_lcd, wb_conmax
	Fabrication	8	EthernetMAC10GE, MultPyramid
	Testing	0	–
	Assembly and Package	0	–
Abstraction Level	System	0	–
	Development Environment	0	–
	Register Transfer	51	AES, b19, BasicRSA, MC8051, PIC16F84, RS232, wb_conmax
	Gate	25	b19, EthernetMAC10GE, RS232, s15850, s35932, s38417, s38584, vga_lcd, wb_conmax
	Layout	12	EthernetMAC10GE, MultPyramid, RS232
	Physical	0	–
Activation Mechanism	Always On	11	AES-T100, EthernetMAC10GE, MultPyramid
	Triggered	79	AES, b19, BasicRSA, EthernetMAC10GE,MC8051, MultPyramid, PIC16F84l, RS232,s15850, s35932, s38417, s38584, vga_lcd,wb_conmax
Effect	Change Functionality	35	b19, EthernetMAC10GE, MC8051, RS232, s15850, s35932, s38417, s38584, vga_lcd,wb_conmax
	Degrade Performance	3	EthernetMAC10GE, MultPyramid, s35932
	Leak Information	24	AES, BasicRSA, PIC16F84, s35932, s38584
	Denial of Service	34	AES, BasicRSA, EthernetMAC10GE, MC8051, MultPyramid, PIC16F84, RS232, s15850, s35932, s38417, s38584, vga_lcd, wb_conmax
Location	Processor	51	AES, b19, BasicRSA, MC8051, MultPyramid, PIC16F84, s15850, s35932, s38417, s38584, vga_lcd
	Memory	0	–
	I/O	4	MC8051, wb_conmax
	Power Supply	2	MC8051-T300, wb_conmax
	Clock Grid	2	EthernetMAC10GE
Physical Characteristics	Distribution	2	b19
	Size	0	–
	Type	86	AES, b19, BasicRSA, EthernetMAC10GE, MC8051, MultPyramid, PIC16F84, RS232,s15850, s35932, s38417, s38584, vga_lcd,wb_conmax
	Structure	8	b19, EthernetMAC10GE, MultPyramid
Total	NA	88	NA

considerably large benchmark, has a statement hardness of 4.4×10^5 with a observability of 0. This indicates that compared to the *b*01 benchmark, the *b*05 benchmark has heavily nested statements along with primary inputs that do not propagate all the way to the primary output, making it more vulnerable to Trojan insertion.

5.2 Vulnerability Analysis at the Netlist Level

Functional hardware Trojans are realized by adding or removing gates; therefore, the inclusion of Trojan gates or the elimination of circuit gates affects circuit side-channel signals such as power consumption and delay

characteristics, as well as the functionality. To minimize a Trojan's contribution to the circuit side-channel signals, an adversary can exploit hard-to-detect areas (e.g. nets) to implement the Trojan. Hard-to-detect areas are defined as areas in a circuit not testable by well-known fault-testing techniques (stuck-at, transition delay, path delay, and bridging faults) or having negligible impact on the circuit side-channel signals. We propose a vulnerability analysis flow to identify such hard-to-detect areas in a circuit. These areas provide opportunities to insert hard-to-detect Trojans and invite researchers to develop techniques to make it difficult for an adversary to insert Trojans.

As Fig. 3 shows, our proposed vulnerability analysis flow performs power, delay, and structural analyses on a circuit to extract the hard-to-detect areas. Any transition inside a Trojan circuit increases the overall transient power consumption; therefore, it is expected that Trojan inputs or triggers will be supplied by nets with low transition probabilities to reduce activity inside the Trojan circuit.

The *Power Analysis* step in Fig. 3 is based on analyzing switching activity; it determines the transition probability of every net in the circuit assuming the probability of 0.5 for '0' or '1' at primary inputs and at memory cells' outputs. More details regarding the transition probability calculation can be found in [26]. Then, nets with transition probabilities below a certain threshold are considered as possible Trojan inputs. The *Delay Analysis* step performs path delay measurement based on gates' capacitance. This allows us to measure the additional delay induced by the Trojan, by knowing the added capacitance to circuit paths. The Delay Analysis step identifies nets on non-critical paths as they are more susceptible to 'smart' Trojan insertion, which would not change the circuit delay. To further reduce Trojan impact on circuit delay characteristics, the delay analysis tool also reports the paths to which a net belongs to avoid selecting nets belonging to different sections of the same path. The *Structural Analysis* step executes the structural transition delay fault testing to find untestable blocked and untestable redundant nets. Untestable redundant nets are not testable because they are masked by a redundant logic, and they are not observable through the primary outputs or scan cells. Untestable blocked nets are not controllable or observable by untestable redundant nets. Creating Trojan inputs/triggers using these untestable nets hides the Trojan impact on delay variations.

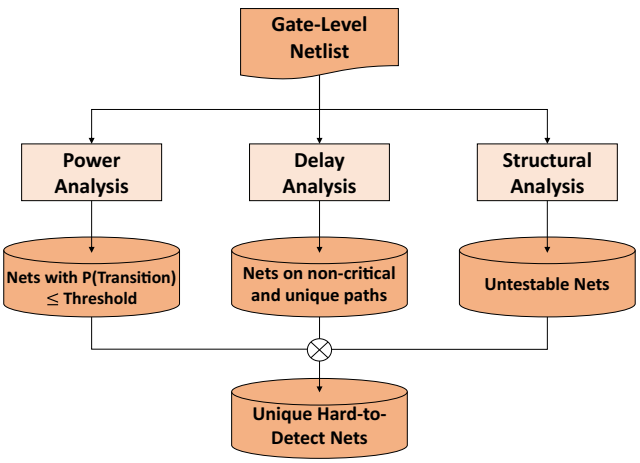

Fig. 3 The vulnerability analysis flow

At the end, the vulnerability analysis flow reports a list of unique hard-to-detect nets in a circuit. This list includes untestable nets with low transition probabilities and also those nets with low transition probabilities on unique non-critical paths. Note that when a Trojan impacts more than one path, it provides greater opportunities for detection. Using unique paths and avoiding shared ones make a Trojan's contribution to the affected paths' delay minimal. This means that the Trojan impact on delay could be masked by process variations. The reported nets are also ensured to be untestable by structural test patterns used in production tests. They also have low transition probabilities so that the Trojans will negligibly affect circuit power consumption. As the nets are chosen from non-critical paths without any shared segments, it would also be extremely difficult to detect Trojans by practical delay-based techniques.

The vulnerability analysis flow can be implemented using most electronic design automation (EDA) tools, and the complexity of the analysis is linear with respect to the number of nets in the circuit. We have applied the flow to the Ethernet MAC 10GE circuit from http://opencores.org, which implements 10Gbps Ethernet Media Access Control functions. Synthesized at 90nm Synopsys technology node, the Ethernet MAC 10GE circuit consists of 102,047 components, including 21,830 flip-flops. The Power Analysis shows that out of 102,669 nets in the circuit, 23,783 of them have a transition probability smaller than 0.1, 7003 of them smaller than 0.01, 367 of them smaller than 0.001, and 99 of them smaller than 0.0001. The Delay Analysis indicates that the largest capacitance along a path, representing path delay, in the circuit is 0.06572 pF, and there are 14,927 paths in the circuit whose path capacitance are smaller than 70% of the largest capacitance, assuming that paths longer than 70% in a circuit can be tested using testers. The Structural Analysis finds that there is not a single untestable fault in the circuit. By excluding nets sharing different segments of one path, there are 494 nets in the Ethernet MAC 10GE circuit considered to be areas where Trojan inputs could be used while ensuring the high difficulty of detection based on side-channel and functional test techniques.

5.2.1 Creating the Trojans

We have also created a separate flow that can create and validate hardware Trojans, given a flattened netlist of a circuit

design [9]. In order to create these Trojans, the flow first applies a random set of test vectors to the design, from which a list of rare nodes are identified. These rare nodes are extracted by calculating the signal probabilities at each node and picking from those nodes whose probabilities are less than a pre-defined threshold (e.g., 0.0001). A hardware Trojan is then created by randomly sampling from these rare nodes. For example, if a 2-trigger hardware Trojan is required, two nodes from the list of rare nodes are randomly picked and an instance of a hardware Trojan is created in a separate file. The same process can be repeated for creating a list of Trojans with an arbitrary number of triggers and also, an arbitrary number of Trojans. Once these Trojans are created, they are functionally validated using Synopsys Tetramax. In this step, we make sure that the Trojans we have created are in fact triggered by selected test patterns (even though those patterns may be extremely rare, which is actually desirable for a Trojan). This step is necessary because an adversary will never insert a functional Trojan that will never get triggered. After validation, the Trojans can be manually inserted into the victim netlist and linked to a desirable payload that can cause malicious alternations to the design.

5.3 Vulnerability Analysis at the Layout Level

In addition to performing vulnerability analysis at the netlist level, we have also implemented a unique flow to identify Trojan-insertion prone areas in a circuit layout [37]. Such a vulnerability analysis flow at the layout level is critical as an untrusted foundry will likely launch a Trojan-insertion attack at the layout level after it gains the entire design from a system integrator. A malicious foundry will likely look for empty regions or 'whitespaces' in a circuit layout, with available routing channels in metal layers above the empty regions. Such whitespaces are very common in many designs today. For example, we have analyzed the ITC99 b15 benchmark layout, which consists of 3296 cells and evaluated the amount of available white space in the design. Figure 4 shows the distribution of whitespace in the benchmark, in units of INVX0, which is the smallest gate in the b15 benchmark layout. The entire layout was divided into grids, with the grid area equal to W^2, where W is the width of the largest cell in the synthesized design library. The layout example shows that there is considerable amount of white spaces in areas closer to the layout boundaries, which averages to about 41.41 units of INVXO. Since such whitespaces are inevitable in any layout design, the threat of Trojan insertion is very realistic as a Trojan size could be as small as a few gates. Further, along with the whitespace, we have also analyzed the average routing channels available above the whitespaces, which might be required to complete the Trojan design. Out analysis shows that for a 9-metal layer implementation, the b15 benchmark layout has 0.84 average unused routing channels available per unit grid.

However, it should be noted that the mere presence of whitespace and routing channels is not a vulnerability that a smart adversary will exploit. In order to bypass detection techniques, a malicious party, whether it is an untrusted foundry with the complete layout or a malicious insider in the design house, will make sure that he/she designs a 'stealthy' Trojan. In order to do this, a vulnerability analysis similar to the one we described in Section 5 must be performed in conjunction with the white space/available routing channel analysis we described above. The first step in doing so is to identify the non-critical paths in the circuit netlist and match those nets to the paths in the layout. A quick analysis of the b15 benchmark circuit shows that, on average, there are about 17 nets per grid whose delay is less than 75% of the critical path of the circuit. Such nets can potentially be exploited to insert a Trojan, as the Trojan-induced capacitance (due to the added wiring connections) and the resulting delay can be evaded from Trojan detection techniques based on path delay. The second step is to insert the Trojan in such a location that its trigger is attached to nets with very low transition probabilities, so that the probability of Trojan activation gets reduced. Figure 5 shows the distribution of nets whose transition probability is less than 10^{-4}, and the delay is less than 75% of the critical path delay, for the b15 benchmark. Clearly, such nets at the layout level, which are also near to layout regions with sufficient amount of whitespace, indicates the vulnerability of the circuit to layout-level Trojan insertion. For example, the nets around row 8 and column 6 in the layout grid in Figs. 4 and 5 have > 10 nets available for implementing a Trojan and its trigger.

6 Trust Benchmark Evaluation

6.1 Trojan Evaluation Suite

Hardware Trojans have a stealthy nature; they rarely activate and make limited contributions to circuit characteristics. To ensure that each Trojan inserted in a trust benchmark does not get activated by test patterns used in production tests, we develop an automatic Trojan Evaluation Suite (TES) to investigate the effectiveness of different types of test patterns at detecting hardware Trojans. TES synthesizes a circuit, generates structural test patterns, applies the patterns to the circuit, and monitors circuit switching activity, including transitions inside Trojan circuits. The flow is implemented using Synopsys tools with additional in-house ones, though it is also possible to develop the same flow with other commercial tools.

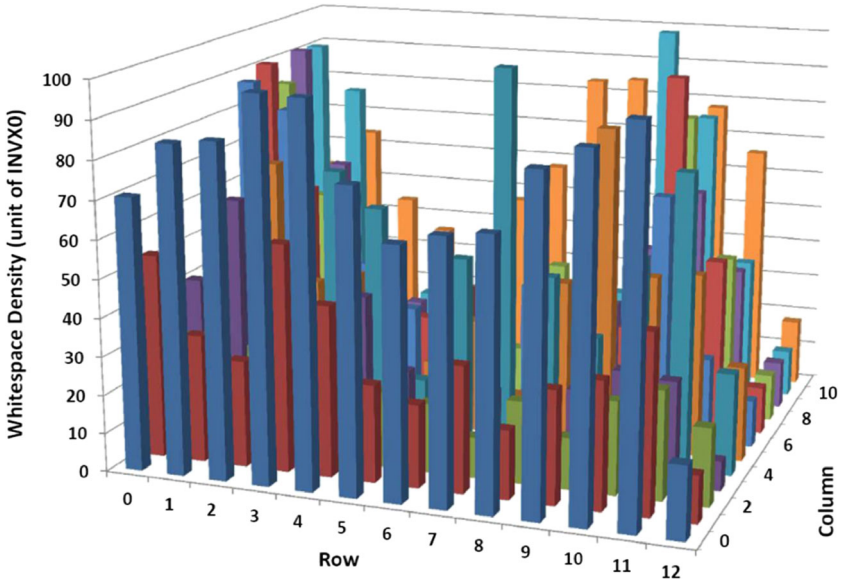

Fig. 4 Distribution of whitespace across the b15 layout

As shown in Fig. 6, the TES flow consists of two main steps: (1) pattern generation and (2) pattern evaluation. To generate structural test patterns, a circuit is first synthesized by the Synopsys Design Compiler. The synthesized netlist is passed to the Synopsys Automatic Test Pattern Generation (ATPG) tool, TetraMax, to generate structural stuck-at-fault (SAF), transition-delay-fault (TDF), and path-delay-fault (PDF) fault test patterns. To generate path-delay fault test patterns, the synthesized netlist is also passed to Synopsys PrimeTime to obtain circuit timing information and critical paths.

To automatically apply patterns and observe switching activity in the main circuit and Trojan circuits during the pattern evaluation step, we develop a program in the Synopsys Verilog Compiler Simulator (VCS) by using Programming Language Interface (PLI) routines. The program fetches patterns generated by TetraMax and applies them to the circuit. Every transition on every net in the circuit is recorded during the simulation, and the test application also determines whether the Trojan is ever activated.

A Trojan can be inserted into a circuit before or after synthesizing the circuit. TES makes the detailed analysis of Trojan circuits possible, and any transition in the Trojan circuit can be recorded. A combinational comparator Trojan, shown in the example in Fig. 7, was inserted into the Ethernet circuit. The Trojan trigger sought a specific 16-bit vector. Whenever the Trojan was triggered, its payload gained control over an internal net. 5218 random functional vectors were applied. Simulation took 104386 ns, and in total there were 106,664,486 transitions in the circuit and

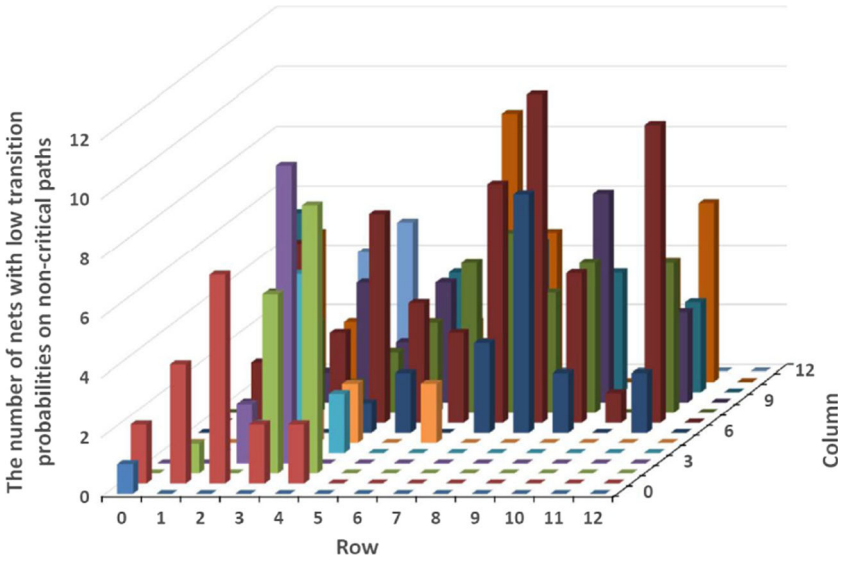

Fig. 5 Distribution of nets with low transition probability and minimal impact on critical path delay in the b15 layout

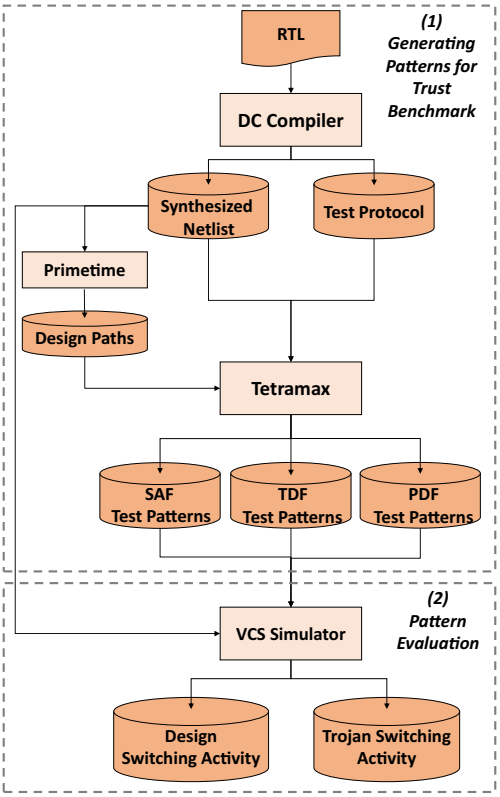

Fig. 6 The Trojan evaluation suite (TES)

4,229 transitions inside the Trojan circuit, though the Trojan never became fully activated (i.e., Trojan's payload never changed the circuit net value).

Without having any knowledge about Trojan insertion, it is possible to use TES to evaluate the effectiveness of a test pattern in generating switching on nets with low transition probability. This information can be used to reduce authentication time by selecting test vectors which create maximum switching on hard-to-detect nets.

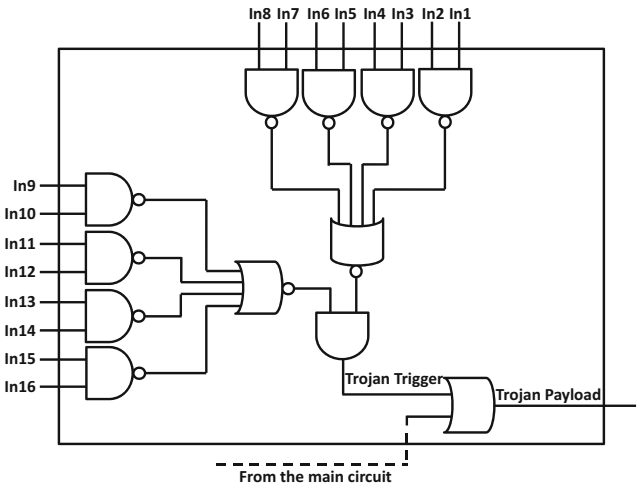

Fig. 7 An example comparator Trojan

6.2 Trojan Detectability

A Trojan's impact on circuit characteristics depends on its implementation. Trojan inputs tapped from nets with higher transition probabilities will aggrandize switching activity inside the Trojan circuit and increase its contribution to circuit power consumption. Furthermore, the Trojan might affect circuit delay characteristics due to additional capacitance induced by extra routing and Trojan gates. To quantitatively determine the difficulty of detecting a gate-level Trojan, a procedure is developed to determine Trojan detectability based on a Trojan's impact on delay and power side-channels across different circuits.

The proposed Trojan detectability metric ($T_{detectability}$) is determined by (1) the number of transitions in the Trojan circuit and (2) extra capacitance induced by Trojan gates and their routing. This metric is designed to be forward-compatible with new approaches for Trojan detection by introducing a new variable, for example a quantity related to the electromagnetic field.

Transitions in a Trojan circuit reflect Trojan contribution to circuit power consumption, and Trojan impact on circuit delay characteristic is represented by measuring the added capacitance by the Trojan. Assuming A_{Trojan} represents the number of transitions in the Trojan circuit, S_{Trojan} the Trojan circuit size in terms of the number of cells, A_{TjFree} the number of transitions in the Trojan-free circuit, S_{TjFree} the Trojan-free circuit size in terms of the number of cells, TIC the added capacitance by Trojan as Trojan-induced capacitance, and C_{TjFree} the Trojan-affected path with the largest capacitance in the corresponding Trojan-free circuit, Trojan detectability ($T_{Detectability}$) at the gate-level is defined as

$$T_{Detectability} = |t| \tag{1}$$

where

$$t = \left(\frac{A_{Trojan}/S_{Trojan}}{A_{TjFree}/S_{TjFree}}, \frac{TIC}{C_{TjFree}} \right) \tag{2}$$

$T_{Detectability}$ at the gate-level is calculated as follows:

1. Apply random inputs to a Trojan-free circuit and obtain the no. of transitions in the circuit (A_{TjFree}).
2. Apply the same random vectors to the circuit with a Trojan and obtain the number of transitions in the Trojan circuit (A_{Trojan}).
3. Perform the Delay analysis on the Trojan-free and Trojan-inserted circuits.
4. Obtain the list of paths whose capacitance are changed by the Trojan.
5. Determine the Trojan-affected path with the largest capacitance in corresponding Trojan-free (C_{TjFree}) and the added capacitance (TIC).
6. Form the vector t (2) and compute $T_{Detectability}$ (the absolute value/modulus of the vector t) as defined in

Eq. (1). Note that Trojan detectability represents the difficulty of detecting a Trojan.

As an example, the comparator Trojan, shown in Fig. 7, is inserted at four different locations, namely TjG-Loc1, TjG-Loc2, TjG-Loc3, and TjG-Loc4 (G represents "gate level"), in the Ethernet MAC 10GE circuit, and Table 3 shows their detectability. The Ethernet MAC 10GE circuit consists of 102047 cells, Column 3 S_{TjFree}, while the Trojan size with 12 cells, Column 5 S_{Trojan}, is only about 0.011% of the entire circuit. TjG-Loc4, in Row 5, experiences the largest switching activity (13484 in Column 4) and relatively induces high TIC (0.00493 pF in Column 6). It is expected that TjG-Loc4 will be the easiest Trojan to be detected due to more impact on circuit side-channel signals, and in turn the detectability of TjG-Loc4 ($T_{Detectability} = 1.07911$ in Column 8) is higher than the others. Although the induced capacitance by TjG-Loc2 (0.00497 pF), in Row 3, is more than the capacitance induced by TjG-Loc1 (0.00029 pF), in Row 2, TjG-Loc1 has more significant contribution into circuit switching activity, 10682 versus 4229 in Column 4. Therefore, TjG-Loc1 has the second largest detectability (0.85166) after TjG-Loc4. Among TjG-Loc2 and TjG-Loc3, although TjG-Loc3, in Row 4, has slightly larger induced capacitance (0.00501 pF), TjG-Loc2 experiences more switching activity (4229 versus 3598 in Column 4). The two Trojans have close detectability where TjG-Loc2 stands above and TjG-Loc3 remains the hardest Trojan to be detected with the lowest Trojan detectability.

6.3 Vulnerability and Detectability

Using the vulnerability analysis flow and the detectability metric proposed in Section 6.2, we have evaluated the vulnerability of the benchmark circuits to Trojan insertion and have presented detectability results for Trojans inserted into these vulnerable circuits. Tables 4 and 5 show detailed analysis of a selected number of gate-level benchmarks. In Table 5, Column 3 indicates that b19 circuit, in Row 2, is the largest circuit in size (62835) among the selected circuits. Table 4 also shows the number of nets with transition probability less than 0.0001 in b19, 4530 in Row 2 and Column 6, is larger than the other circuits, and b19 has considerable number of paths whose capacitances are less than 70% of its critical path' capacitance, 474358 in Column 8. Further, there are eight untestable faults is b19, in Column 9. These provide significant opportunity for implanting Trojans resilient against power and delay side-channel analyses in b19. Table 5 confirms that b19-T100 with $T_{Detectability} = 0.02498$, in Column 8, is the second most difficult Trojan to detect as no transition inside the Trojan is observed, 0 in Column 4, and it induces a small capacitance, 0.00095 pF in Column 6, on a non-critical path, 0.03785 pF in Column 7. s38584-T200, in Row 7, has the lowest detectability, 0.01390 in Column 8; similar to b19-T100, there is no switching activity in s38584-T200, 0 in Column 4, and s38584-T200 induces less capacitance, 0.00041 pF in Column 6, on a shorter path, 0.02984 pF in Column 7, compared to b19-T100. We can also note that trust benchmark $s35854 - T300$ has high detectability (2.84578), as the Trojan gate count is fairly high (731) and the resulting Trojan induced capacitance (TIC) is also high. This makes the Trojan easier to detect through delay side-channels. To summarize, Fig. 8 shows the $T_{detectability}$ metric figures for all the gate-level benchmarks on Trust-Hub, which show varying levels of detection difficulty, based on the aforementioned metric.

6.4 Benchmarks and Attack Models

As we discussed in Section 1, the threat of hardware Trojans can be explained in the context of various adversarial/attack models. In this section, we highlight how each benchmark that we have developed can fit a particular attack model. This is necessary as the motivation behind any Trojan detection/prevention technique cannot be fully understood without specifying its context/scope in terms of an attack model. Below, we present our methodology for classifying the trust benchmarks in the form of a case study on a benchmark. Additionally, we also point out what changes/additions can be made to a benchmark in order to fit it to a particular attack model.

EthernetMAC10GE-T200 The benchmark only comes with the .def (Layout) file and does not include a golden layout. However, there is a golden netlist available. We

Table 3 The detectability of the comparator Trojan placed at four different locations in Ethernet MAC 10GE circuit

Trojan	A_{TjFree}	S_{TjFree}	A_{Trojan}	S_{Trojan}	$TIC(pF)$	$C_{TjFree}(pF)$	$T_{Detectability}$
TjG-Loc1	106,664,486	102,047	10,682	12	0.00029	0.04136	0.85166
TjG-Loc2	106,664,486	102,047	4,229	12	0.00497	0.07211	0.34413
TjG-Loc3	106,664,486	102,047	3,598	12	0.00501	0.04969	0.30403
TjG-Loc4	106,664,486	102,047	13,484	12	0.00493	0.05260	1.07911

Table 4 Design vulnerability analysis of a selected number of Trojan-free circuits

Circuit	Power Analysis					Delay Analysis		Structural Analysis
	# Nets	< 0.1	< 0.01	< 0.001	< 0.0001	Critical path Capacitance(pF)	$< 70\%$ of Critical Path Capacitance	# Untestable faults
b19	70,259	14,482	8,389	5,533	4,530	0.37724	474,358	8
s38417	5,669	589	291	219	69	0.05015	41,901	0
s38584	7,203	817	197	85	30	0.04467	27,689	0
s35932	6,269	0	0	0	0	0.00851	3,156	0

can classify this particular benchmark with different attack models as follows:

- A: No (This benchmark lacks any RTL level description, and going from a gate-level description to a behavioral RTL description is non-trivial. Thus, it does not fit attack model A.)
- B: Possible to adapt (The golden netlist can be used to generate a golden layout, which is required in attack model B, since a design house will always have a golden layout if only the foundry is untrusted.)
- C: Possible to adapt (A separate Trojan can be inserted into the golden netlist at the gate-level, and a layout can be generated. However, the original Trojan-inserted layout would not be needed to fit the benchmark to this attack model.)
- D: No (The RTL level description is not available, due to which a Trojan insertion at the RTL level, emulating an untrusted 3PIP vendor, cannot be performed.)
- E: No (Same as above, due to lack of RTL level description)
- F: No (Same as above.)
- G: Possible to adapt (In order to fit this attack model, a gate-level Trojan can be inserted into the golden netlist and converted to a layout. From the T200 benchmark, the particular Trojan can be observed and inserted into the newly generated layout, thereby emulating both an untrusted SoC developer and an untrusted foundry.)

A count of the number of trust benchmarks which are available on the Trust-Hub website and fit the seven attack models, is shown in Fig. 9. Clearly, most of the benchmarks can be further expanded to emulate the scenario where multiple entities in the supply chain are untrusted.

7 Future Work

Although we have developed a total of 91 trust benchmarks and the corresponding tools, there is still room for more improvement. As part of future work, we plan to explore the following scenarios and develop appropriate solutions.

- **Including additional trust benchmarks:** The set of benchmarks we have developed are only a small set of possible hardware Trojans that can be designed. We plan to update/add more Trojans and trust benchmarks to the Trust-Hub website as part of our ongoing work.
- **Fitting new attack models:** As seen from Fig. 9, there are already a large number of trust benchmarks that can be adapted to fit attack models E, F and G, which would involve inserting Trojan(s) at two or more levels of abstraction into a single benchmark circuit. For example, a design starting at the RTL level can be inserted with Trojan 1 and synthesized to a netlist. After layout generation from the netlist, Trojan 2 can then be inserted at the layout level. This would capture

Table 5 The detectability ($T_{Detectability}$) of a selected number of gate-level Trojans inserted in the circuits in Table 4

Trojan	A_{TjFree}	S_{TjFree}	A_{Trojan}	S_{Trojan}	$TIC(pF)$	$C_{TjFree}(pF)$	$T_{Detectability}$
b19-T100	4,037,383	62,835	0	83	0.00095	0.03785	0.02498
s38417-T100	2,717,682	5,329	59	11	0.00417	0.03234	0.13939
s38417-T200	2,717,682	5,329	1,328	11	0.00531	0.03052	0.41085
s38417-T300	2,717,682	5,329	257	15	0.00046	0.03078	0.04847
s38584-T100	423,986	6,473	705	9	0.00095	0.01504	1.25877
s38584-T200	423,986	6,473	0	83	0.00041	0.02984	0.01390
s38584-T300	423,986	6,473	16	731	0.01246	0.00438	2.84578
s35932-T100	353,304	5,426	354	15	0.00049	0.00699	0.86644
s35932-T200	353,304	5,426	733	12	0.00316	0.00973	1.26280
s35932-T300	353,304	5,426	738	36	0.00050	0.00823	0.37533

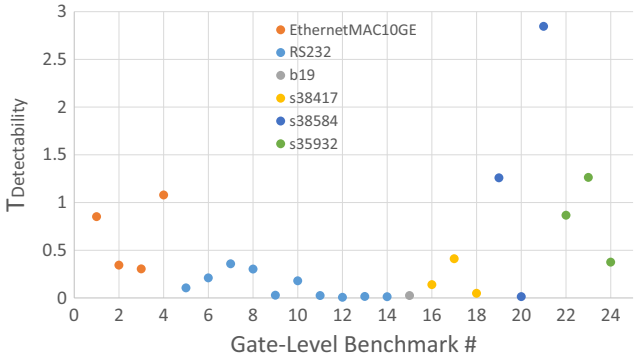

Fig. 8 Detectability metric for gate-level Trojans in trust benchmarks

the essence of attack model F (fabless SoC design house).

- **Larger benchmarks:** Most of the benchmarks currently available are fairly small in size and might not capture the Trojan detection problem in the context of real-life industrial designs, at which point some detection strategies might become completely infeasible. Thus, it would make sense to develop larger trust benchmarks.
- **SoC scenario:** We also plan to put up SoC designs that combine various IPs (at different abstraction levels) into one single design. This emulates a scenario where multiple IP vendors are untrusted, and is representative of the SoC development process today.
- **Dynamic trust benchmark generation:** While the Trojans we have created are inserted only into specific benchmark circuits, Trojan benchmarks, which are arbitrary Trojan circuits (at the RTL, gate or layout level) with an arbitrary number of gates/cells, should ideally be applicable for insertion into any given circuit. The Trojan benchmarks we have developed have been manually inserted into benchmark circuits after performing the vulnerability analysis flow. In order to create dynamic trust benchmarks, we require automatic payload and trigger identification of a circuit at different levels (RTL, gate and layout). While trigger identification is straight-forward (as we have shown in Section 5), automating payload identification for any arbitrary circuit is a non-trivial problem, which must be addressed. Nonetheless, the detectability metric we have proposed would still be useful in characterizing the Trojan benchmarks after they have been inserted into a circuit.
- **Testbench development:** In order to help in the use of the benchmarks, we are constantly adding and refining testbench programs that can help researchers to run the benchmark with or without the Trojan and exactly activate it to observe the effects on delay, power and other side-channels.
- **FPGA device Trojans:** In Section 2.4, we delved into hardware Trojans that can be inserted into the FPGA fabric, while being dependent or independent of the IP on the FPGA. Currently, the FPGA-implementable trust benchmarks on Trust-Hub only cover the threat of an untrusted IP vendor, where the Trojan is a part of the IP. Trojans that are part of the FPGA device (e.g. DCM corruptors, IP leak), which would need to be implemented through the FPGA firmware, is part of our future work.

8 Conclusion

The hardware trust community needs common ground to more effectively address the Trojan detection problem. As no standard measurements, benchmarks, or tools have previously been developed, we have put our effort into developing tools that can aid us in generating trust benchmarks.

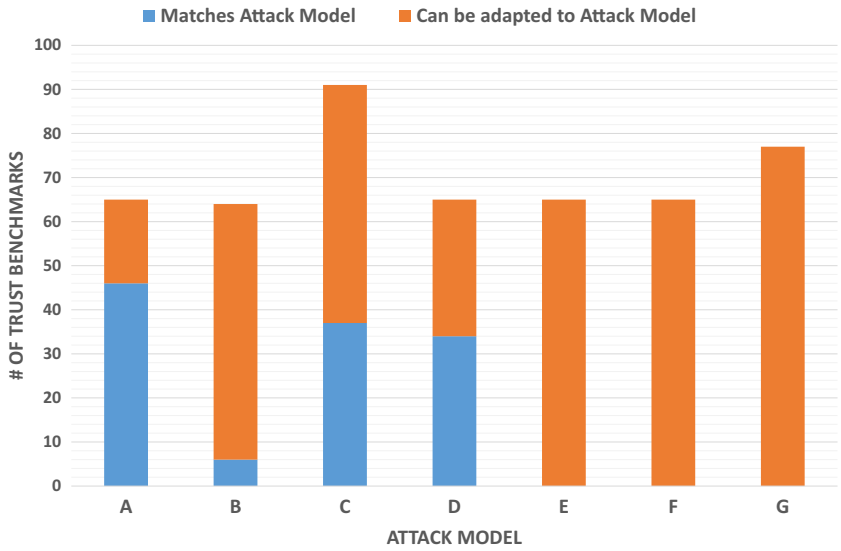

Fig. 9 Classifying the number of trust benchmarks that fit different attack models

As part of this toolset, the vulnerability analysis flow determines areas in a circuit that are more probable to be used for Trojan implementation. Further, we developed an automatic Trojan evaluation suite to measure the resiliency of hardware Trojans. Then, we defined the Trojan detectability metric to quantify Trojan impact on circuit power consumption and circuit performance, thereby giving us an idea of its stealthiness. Using these tools and metrics, we generated a large number of trust benchmarks, which are available at trust-hub.org for researchers to evaluate their detection techniques. These benchmarks are now currently being used in developing the state-of-the-art in hardware Trojan detection techniques [12, 21, 25, 28, 38, 39]. As part of our future work, we plan to expand the current benchmarks to fit more attack models, develop Trojan benchmarks and also, add newer benchmarks to the repository to enhance outcomes of Trojan detection research.

Acknowledgments This work was supported in part by the National Science Foundation (NSF) under grant 1513239.

References

1. Marinissen E, Iyengar V, Chakrabarty K (2002) A set of benchmarks for modular testing of socs. In: Proceedings International Test Conference, 2002, pp 519–528
2. Brglez F (1985) A neutral netlist of 10 combinational benchmark circuits and a target translation in fortran. In: ISCAS-85
3. Brglez F, Bryan D, Kozminski K (1989) Combinational profiles of sequential benchmark circuits. In: IEEE international symposium on circuits and Systems, 1989, vol 3
4. Lee C, Potkonjak M, Mangione-Smith WH (1997) Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In: Proceedings of the 30th annual ACM/IEEE international symposium on microarchitecture, ser. MICRO 30. IEEE Computer Society, Washington, DC, pp 330–335. [Online]. Available: http://dl.acm.org/citation.cfm?id=266800.266832
5. Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB (2001) Mibench: a free, commercially representative embedded benchmark suite. In: 2001 IEEE international workshop on workload characterization, 2001. WWC-4, pp 3–14
6. Salmani H, Tehranipoor M, Karri R (2013) On design vulnerability analysis and trust benchmarks development. In: 2013 IEEE 31st international conference on computer design (ICCD), pp 471–474
7. Tehranipoor M, Koushanfar F (2013) A survey of hardware trojan taxonomy and detection. IEEE Design Test 99:1–1
8. Xiao K, Forte D, Jin Y, Karri R, Bhunia S, Tehranipoor M (2016) Hardware Trojans: Lessons learned after one decade of research. ACM transactions on design automation of electronic system (To Appear)
9. Chakraborty RS, Wolff F, Paul S, Papachristou C, Bhunia S (2009) Mero: A statistical approach for hardware trojan detection. In: Proceedings of the 11th international workshop on cryptographic hardware and embedded systems, ser. CHES '09. Springer, Berlin, pp 396–410. [Online]. Available: doi:10.1007/978-3-642-04138-9_28
10. Banga M, Hsiao MS (2010) Trusted rtl: Trojan detection methodology in pre-silicon designs. In: 2010 IEEE international symposium on hardware-oriented security and trust (HOST), pp 56–59
11. Banga M, Hsiao M (2009) A novel sustained vector technique for the detection of hardware trojans. In: 2009 22nd international conference on VLSI design, pp 327–332
12. Waksman A, Suozzo M, Sethumadhavan S (2013) Fanci: identification of stealthy malicious logic using boolean functional analysis. In Proceedings of the 2013 ACM SIGSAC conference on computer & communications security, ser. CCS '13. ACM, New York, pp 697–708. [Online]. Available: doi:10.1145/2508859.2516654
13. Xiao K, Zhang X, Tehranipoor M (2013) A clock sweeping technique for detecting hardware trojans impacting circuits delay. IEEE Design Test 30(2):26–34
14. Wang X, Salmani H, Tehranipoor M, Plusquellic J (2008) 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
15. Narasimhan S, Du D, Chakraborty RS, Paul S, Wolff F, Papachristou C, Roy K, Bhunia S (2010) Multiple-parameter side-channel analysis: a non-invasive hardware trojan detection approach. In: 2010 IEEE international symposium on hardware-oriented security and trust (HOST), pp 13–18
16. Zhang X, Tehranipoor M (2011) Ron: an on-chip ring oscillator network for hardware trojan detection. In: 2011 Design, automation test in Europe, pp 1–6
17. Hu K, Nowroz AN, Reda S, Koushanfar F (2013) High-sensitivity hardware trojan detection using multimodal characterization. In: Design, automation test in europe conference exhibition (DATE), 2013, pp 1271–1276
18. Stellari F, Song P, Weger AJ, Culp J, Herbert A, Pfeiffer D (2014) Verification of untrusted chips using trusted layout and emission measurements. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST), pp 19–24
19. Li J, Lach J (2008) At-speed delay characterization for ic authentication and trojan horse detection. In: IEEE international workshop on hardware-oriented security and trust, 2008. HOST 2008, pp 8–14
20. Salmani H, Tehranipoor M (2013) 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 (DFT), pp 190–195
21. Zhang X, Tehranipoor M (2011) Case study: detecting hardware trojans in third-party digital ip cores. In: 2011 IEEE international symposium on hardware-oriented security and trust (HOST), pp 67–70
22. Love E, Jin Y, Makris Y (2012) Proof-carrying hardware intellectual property: A pathway to trusted module acquisition. IEEE Transactions on Information Forensics and Security 7(1):25–40
23. Love E, Jin Y, Makris YG (2011) Enhancing security via provably trustworthy hardware intellectual property. In: 2011 IEEE international symposium on hardware-oriented security and trust, pp 12–17
24. Guo X, Dutta RG, Jin Y, Farahmandi F, Mishra P (2015) Pre-silicon security verification and validation: a formal perspective. In: Proceedings of the 52Nd annual design automation conference, ser. DAC '15, ACM, New York. [Online]. Available: doi:10.1145/2744769.2747939
25. Rajendran J., Vedula V, Karri R (2015) Detecting malicious modifications of data in third-party intellectual property cores. In: 2015 52nd ACM/EDAC/IEEE design automation conference (DAC), pp 1–6
26. Salmani H, Tehranipoor M, Plusquellic J (2012) A novel technique for improving hardware trojan detection and reducing trojan activation time. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 20(1):112–125

27. Salmani H, Tehranipoor M (2012) Layout-aware switching activity localization to enhance hardware trojan detection. IEEE Transactions on Information Forensics and Security 7(1):76–87
28. Forte D, Bao C, Srivastava A (2013) Temperature tracking: an innovative run-time approach for hardware trojan detection. In: Proceedings of the international conference on computer-aided design, ser. ICCAD '13. IEEE Press, Piscataway, pp 532–539. [Online]. Available: http://dl.acm.org/citation.cfm?id=2561828.2561931
29. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Logic encryption: A fault analysis perspective. In: Proceedings of the conference on design, automation and test in europe, ser. DATE '12. EDA Consortium, San Jose, pp 953–958. [Online]. Available: http://dl.acm.org/citation.cfm?id=2492708.2492947
30. Chakraborty RS, Bhunia S (2009) Security against hardware trojan through a novel application of design obfuscation. In: Proceedings of the 2009 international conference on computer-aided design, ser. ICCAD '09. ACM, New York, pp 113–116. [Online]. Available: doi:10.1145/1687399.1687424
31. Chakraborty R, Bhunia S (2009) Harpoon: an obfuscation-based soc design methodology for hardware protection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28(10):1493–1502
32. Xiao K, Tehranipoor M (2013) Bisa: built-in self-authentication for preventing hardware trojan insertion. In: 2013 IEEE international symposium on hardware-oriented security and trust (HOST), pp 45–50
33. Rajendran J, Sam M, Sinanoglu O, Karri R (2013) Security analysis of integrated circuit camouflaging. In: Proceedings of the 2013 ACM SIGSAC conference on computer & communications security, ser. CCS '13. ACM, New York, pp pp 709–720. [Online]. Available: doi:10.1145/2508859.2516656
34. Imeson F, Emtenan A, Garg S, Tripunitara M (2013) Securing computer hardware using 3d integrated circuit (ic) technology and split manufacturing for obfuscation. In: Presented as part of the 22nd USENIX security symposium (USENIX Security 13), USENIX, Washington. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/imeson
35. Vaidyanathan K, Das BP, Sumbul E, Liu R, Pileggi L (2014) Building trusted ics using split fabrication. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST), pp 1–6
36. Mal-Sarkar S, Krishna A, Ghosh A, Bhunia S (2014) Hardware trojan attacks in fpga devices: threat analysis and effective counter measures. In: Proceedings of the 24th edition of the great lakes symposium on VLSI, ser. GLSVLSI '14. ACM, New York, pp 287–292. [Online]. Available: doi:10.1145/2591513.2591520
37. Salmani H, Tehranipoor MM (2016) Vulnerability analysis of a circuit layout to hardware trojan insertion. IEEE Transactions on Information Forensics and Security 11(6):1214–1225
38. Dupuis S, Di Natale G, Flottes M-L, Rouzeyre B (2013) On the effectiveness of hardware trojan horse detection via side-channel analysis, vol 22. [Online]. Available: doi:10.1080/19393555.2014.891277
39. Hu W, Mao B, Oberg J, Kastner R (2016) Detecting hardware trojans with gate-level information-flow tracking. Computer 49(8):44–52. [Online]. Available: doi:10.1109/MC.2016.225