A Side Channel Delay Analysis for Hardware Trojan Detection

A proposal submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Ashkan Vakil

Director: Dr. Avesta Sasan
Department of Electrical and Computer Engineering

Fall Semester 2020
George Mason University
Fairfax, VA

Table of Contents

List of Tables

List of Figures

Abstract

A SIDE CHANNEL DELAY ANALYSIS FOR HARDWARE TROJAN DETECTION
Ashkan Vakil, PhD
George Mason University, 2020

This research proposal introduces a learning assisted modeling technique for the purpose of Hardware Trojan detection. Our proposed model, unlike the prior art, does not require a Golden fabricated chip as a fingerprint to compare the side channel signals. Instead, by modeling the voltage drop and voltage noise pre-fabrication, and with training a Neural Network post-fabrication, our proposed technique can improve the timing model collected during timing closure and produces a Neural assisted Golden Timing Model (NGTM) for side channel delay-signal analysis.

The Neural Network acts as a process tracking watchdog for correlating the static timing data (produced at design time) to the delay information obtained from clock frequency sweeping test. Proposed modeling technique enables Hardware Trojan detection close to 90% in the simulated scenarios.

This page intentionally left blank

Chapter 1: Introduction

In the past decade, to reduce the fabrication cost and for economic feasibility, the manufacturing supply chain of Integrated Circuits (IC) has adopted a globally distributed model [2]. The use of untrusted entities in this global supply chain has raised pressing concerns about the security of the fabricated ICs with threats including IP theft, counterfeiting [3], over-production of ICs [4,5], and hardware Trojan insertion. There has been research works to both improve the threat models, and also provide a better remedy for each aspect of hardware security [4,6,7]. To protect ICs against first three threats, logic encryption methods have been investigated [4]. Although, these techniques are vulnerable to attacks using Boolean satisfiability (SAT) solvers [8]. Several methods have been proposed to create SAT resilient logic encryption [9–15]. These methods work by increasing the number of SAT iterations or by inserting elements such as cycles that make the SAT solver stuck in an infinite loop [16, 17].

The last security threats is the adversarial infestation of fabricated ICs with a Hardware (HW) Trojan which usually targets devices with sensitive applications. A Trojan can be broadly defined as a malicious modification to a circuit to control, modify, disable, or monitor its logic. The spectrum of harm caused by HW Trojans is broad. It can range from passive Trojans for activity monitoring or stealing information to weaponized Trojans that could cause catastrophic consequences in the critical military, space, or medical applications [18]. Thereby, detecting HW Trojans is highly crucial, and it has become a significant concern for governments and industries.

One solution for detecting HW Trojan is through destructive reverse-engineering schemes to check and ensure that the manufactured chips' logical structure and functional integrity

is untouched. Relying on Reverse Engineering (RE) to produce a golden model from fabricated ICs has severe limitations. The destructive process of de-layering, combined with advanced image processing techniques could be used for the generation of a netlist, but not a golden model containing all process information (such as doping levels and the extent of parametric variation in that process) as such information can not be extracted using imaging techniques. Besides, IC reverse engineering requires significant investment, is extremely challenging in advanced geometries, and is quite time and resource consuming. One may argue that a Trojan-induced logic-change can be detected during Manufacturing test process. However, HW Trojans are stealthy in nature, and are designed such that they are rarely activated. This makes detecting the Trojans during manufacturing testing highly difficult if not impossible.

Conventional manufacturing VLSI test and verification methodologies fall short in detecting HW Trojans due to the different and un-modeled nature of these malicious alterations. This has led many researchers to investigate solutions for detection of HW Trojans through statistical analysis of side-channel information collected from ICs, including side-channel power analysis [1, 19–23], power supply transient signal analysis [24, 25], regional supply currents analysis [26], temperature analysis [27], wireless transmission power analysis [28], and side-channel delay analysis [29–35].

The problem with many of the previous HW Trojan detection solutions is a need for some sort of a golden model from which the parametric signature of the fabricated ICs are collected and used to define a decision boundary (power, delay, temperature, etc) for separating the Trojan-infested ICs. However, building a golden IC is extremely difficult or even impossible: In many cases, especially in advanced technology node, the choice of the foundry is limited to one or a very few, none of which may be trusted. Even if a trusted foundry exists, fabrication of a small volume of ICs for obtaining a golden IC is usually cost prohibitive[22]. Moreover, the process used in each foundry is quite different and a golden IC that is fabricated in one foundry can not be used for assessing an IC fabricated in another foundry.

For these reasons, we do not assume the existence of a golden IC. Instead, we develop and train a learning-assisted timing-adjustment model post-fabrication that combined with a voltage variation modeling during Static Timing Analysis (STA) and make a golden model. This work is motivated by two previous papers: The side-channel power analysis in [22] and side-channel delay analysis in [29], a short description of which is given next:

The side-channel statistical power analysis solution for Trojan detection in [22] proposed that the trusted region for the operation of a Trojan free IC can be learned using a combination of a trusted simulation model, measurements from the carefully engineered and distributed Process Control Monitor (PCM) structures, and advanced statistical tail modeling techniques. This work, however, relies on side-channel power analysis for the detection of hardware Trojan. For observing a meaningful change in leakage or dynamic power, the size of HW Trojan has to be large. Hence, this technique falls short of detecting Hardware Trojans implemented using a small number of gates. This is when our proposed solution can detect even a single added logic gate in a tested timing path. Besides, [22] relies on the usage of PCMs (with a defined structure that is repeated and distributed over the IC) for extracting the process parameters. However, the number and accuracy of PCMs are limited. Although PCM can roughly track the process corner from chip to chip and could be used for the rough calibration of timing and spice models, they fall short of accurately characterizing the behavior of different gates and metal layers. This is when in our proposed solution, every timing path could be used as a PCM for training the neural assisted timing augmentation engine, and therefore the impact of different timing path topology, different gate types/sizes, and the change in the capacitive or resistive load of different metal layers are taken into account.

The side channel delay analysis solution in [29] uses Clock Frequency Sweeping Test (CFST) to detect the hardware Trojan. However, it relies on the existence of a Golden IC for delay comparison. Our proposed side-channel Trojan detection scheme is inspired by this work (and used CFST for the generation of label data points for each feature set), however, our proposed mechanism does not need a Golden IC.

Chapter 2: Background

In practice, a HW Trojan can be inserted at any stage of the design flow[37–41]. Upon activation of the Trojan, the Trojan delivers its payload which can result in a denial of service in the whole or part of the circuit, corruption of the circuit's functionality, an alteration in the characteristic of the circuit such as aging factors, or leaking secret information [40, 41].

Countermeasures against HW Trojans can be categorized into the design-for-security, run-time monitoring, and detection solutions [41]. The design-for-security approaches opt to reduce the chances of Trojan insertion (e.g., through hardware obfuscation). However, they can neither guarantee a Trojan free IC nor detect them. The run-time techniques monitor the functionality of the IC (usually through snapshots of its operation) at run-time [42], and compare it against known behavior signatures. However, if the Trojan impact does not persist, it does not create the expected signature, or affects the IC's behavior in a way that is not modeled (in the monitoring solution), the monitoring schemes will fail to detect the Trojan. On the other hand, detection approaches aim to directly or indirectly detect the presence of HW Trojans. Detection solutions could be destructive or non-destructive [41]. The destructive solutions, that could provide an ultimate proof for Trojan's presence in the selected IC, require full reverse engineering of the IC.

The non-destructive detection approaches can detect the Trojans by either activating them or via side-channel signal analysis [40,41,43]. The former relies on finding a set of input patterns that trigger the possible Trojan such that the Trojan results in a noticeable impact (e.g., change in expected output). On the other hand, the side-channel based detection methods attempt to identify the Trojan presence through side-channel information obtained from an IC, e.g., power consumption [1, 20, 21, 44], electromagnetic emanations [45], or path delays [29–31].

Detecting Trojans by activating them during manufacturing test is significantly challenging. In principal, Trojans are designed to be activated through a rare sequence or combination of events, only known to the adversary [29, 41]. Testing an IC exhaustively using all sets of possible patterns is not practically feasible [29]. Note that not all HW Trojans alter the functionality. For example, a HW Trojan may be designed to leak secret information (with antenna or noise); making such Trojan immune to activation-based solutions as the functional impact of such HW Trojan is not observable.

Trojan activation solutions' limited coverage has encouraged the research to focus on side-channel analysis-based detection techniques. One widely studied Trojan detection direction is through side-channel power analysis [1,46–49] that focuses on power consumed by The Trojan Circuit (TC). However, for side-channel power analysis, TC should be partially or fully activated. Therefore it is better suited for Trojans, trigger of which is connected to shorter timing paths with a higher degree of controllability. At the same time, the power signature of the TC should be significant enough to stand out (make a noticeable change in the power consumption of the IC) as the demanded current of an IC can be monitored with limited precision (through package balls or, at best case, through power delivery networks pads). Hence, the observed current signature is the accumulation of the transistors' current needed for the normal function of the IC and those added for implementation of TC. Therefore, the size of a TC should be large enough to be observable using such techniques.

The delay side-channel test, on the other hand, focuses on the change of the delay and measures path delays to detect a Trojan [19]. The delay analysis proposed in [30] monitors the critical timing-paths to detect Trojans. However, it fails to consider the near-critical or shorter timing paths. The authors of [31] insert shadow registers to measure the delay of each timing-path. However, this results in a large area overhead. Finally, the solution suggested by [29] uses Clock Frequency Sweeping Test (CFST) to detect Trojans. However, it relies on the existence of a Golden IC for delay comparison. This work inspires our proposed side-channel Trojan detection scheme; however, our proposed solution relaxes the need for the presence of a Golden (trusted) fabricated IC.

Chapter 3: Threat Model and Challenges

3.1 Trojan Threat Model

The adversary in this paper is an untrusted foundry with access to GDSII (Graphic Database System format). The goal of the adversary is to insert a Trojan that is triggered based on a combination, or a sequence of rare events. A Trojan, As illustrated in Fig. 3.1, consists of 1) Trojan's Trigger inputs (TT), 2) Trojan's Triggering (which could be sequential or combinational) Circuit (TTC), and 3) Trojan Payload (TP). Upon activation, the TP alters the circuit functionality. We assume that no Golden IC exists, and the Trojan is inserted in all fabricated ICs.

3.2 Trojan Detection Challenge: Variability

The TT of an HW Trojan poses an additional capacitive load on its driving cell, resulting in a slower rise and fall, while its TP adds a gate delay to its victim timing path. In a perfect world, A Trojan can be detected by tracking and analyzing the changes in the delay of timing-paths compared to that predicted by STA. The challenge for this solution is that STA suggested delay information can be significantly different from delay information that is collected at the test time. This is due to several factors most notable of which are: 1- voltage noise, 2- Process Variation (random and systematic variations), and 3- process drift.

Voltage Noise: In an ASIC chip, the current flow to and return from transistors via the Power Delivery Network (PDN), which consist of a sequence of resistive, inductive and capacitive elements. A flow of current through a resistive element manifests a voltage (IR) drop that is proportional to the current flow (I) and element's resistance (R). Furthermore, the current flow is orchestrated by the die switching activity that changes per clock cycle

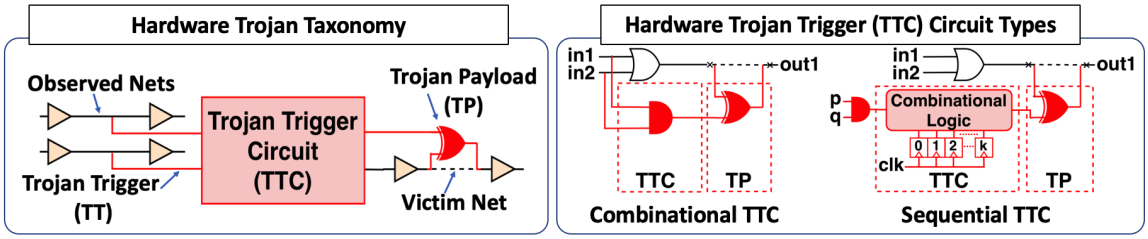


Figure 3.1: (left): Trojan taxonomy, (right): Trojan trigger circuit types

[50]; Hence, due to inductive nature of PDN, transistors also experience an inductive voltage drop which is proportional to the PDN's inductance profile (L) and the rate of change in the current, denoted by $d(i)/d(t)$, which exacerbates at scaled geometries with increased current demand and higher frequencies [51]. In addition, there are both intentional and device/metal topology related decoupling capacitance (DECAP), that decouple the power and ground lines. This results in the PDN to act as an RLC network. In the result, the voltage that a transistor experience is below the voltage supplied from the voltage regulator and also changes dynamically from cycle to cycle causing variation in the delay of timing paths [52]. The cycle to cycle voltage variation, which in physical design flow is denoted by voltage noise, causes clock jitter [53] leading to uncertainty in clock arrival time to the clock pin of registers.

During STA the IR drop and voltage noise are modeled by (1) specifying a rail voltage value below supplied voltage to account for IR drop, and (2) using register-endpoint uncertainty to guard against voltage-variation-induced clock network jitter [53]. The chosen values for the rail-voltage and uncertainty should be pessimistic to capture the worst-case (to prevent setup/hold timing failure). However, the majority of timing-paths experience smaller IR-drop and voltage noise [54]. This poses a security threat; the pessimistic margins build large unused timing slack into the majority of timing-paths, which is not visible to the physical designer and test engineer. The unused timing slacks can be used by an adversary in an untrusted foundry to design a Trojan and hide its delay impact.

Random Process Variation: The random process variation refers to the variations in the physical and electrical properties of transistors due to the physical limitations faced

during the fabrication process [55]. The random process variation impacts the delay and drive strength of fabricated transistors and makes Trojan detection more difficult as the test engineer needs to differentiate between the delays imposed by random process variation and the timing impact of an HW Trojan. Figure 3.2 illustrates the effect of the random process variation on the slack of timing paths.

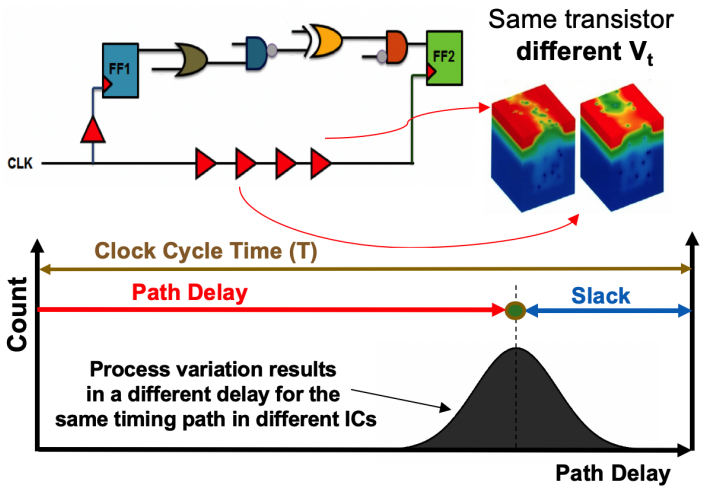


Figure 3.2: The impact of random process variation on the delay of a timing-path when sampled across multiple dies (after fabrication).

Systematic Process Variation: Systematic Process Variation is the result of imperfection in one or several process steps, as a result of which, a systematic shift occurs in the behavior of transistors or wires. For example, the systematic shift may speed up all NMOS transistors, increase the capacitance of a given metal layer, or reduce PMOS transistors' strength. Unlike random process variation (mitigation of which is disclosed when we describe our IC classification methodology), the systematic (inter-die) process variation affects all devices similarly. Therefore, systematic process variation behaves similarly to process drift, with the difference that process drift is the intended consequence of improving the fabrication process. On the other hand, the *systematic process variation is an unintended consequence of imperfection in one or several processing steps.* For example, if during the Chemical Mechanical Polishing step, the height of a specific metal layer, e.g., M4, was less or more than the process defined height, the expected resistance, and capacitance of all M4

net segments systematically shifts. In practice, the systematic process drift can be treated similarly to process drift.

Process Drift: The SPICE model for the fabrication process in a new technology node is released soon after the process is stabled and is used to characterize the standard cell libraries deployed in a physical design house. The SPICE model and standard cell libraries are padded with carefully crafted margins to guarantee a high yield. Furthermore, the foundry keeps improving the process over time to improve yield and reduce cost and may update the process by deploying newer and more capable stepping devices. Hence, the fabrication process and the released SPICE model drift apart over time. The improvement in the process builds large unused slacks in a fabricated IC that is designed using the older SPICE model. This practice poses a security problem as these unused and hidden timing slacks (to the test engineer) can be used by an adversary in the untrusted foundry to design stealthy HW Trojan(s). Figure 3.3 illustrates the impact of the Process Drift on the slack of timing paths.

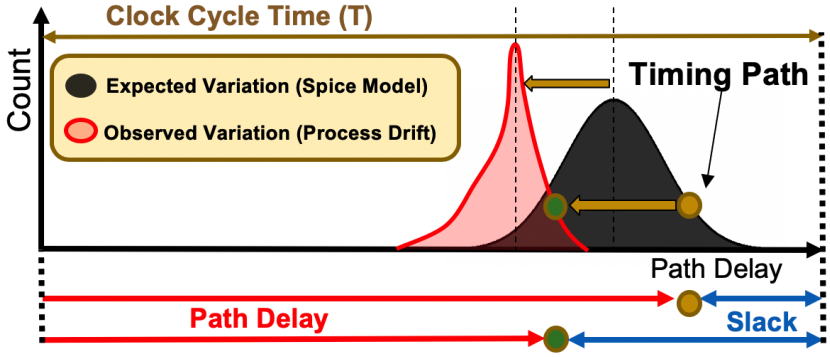


Figure 3.3: Improvement in the process over time non-linearly changes the delay of different timing-paths (process drift). The process drift affects each timing-path differently.

Chapter 4: Proposed HW Trojan Detection Solutions

Proposed modeling technique integrates multiple variation modeling and mitigation techniques into a side-channel delay analysis solution for the purpose of HW Trojan testing. Using our proposed model, we characterize and mitigate the impact of voltage noise, process variation, and process drift to improve the correlation between the adjusted timing model and the fabricated ICs' timing behavior. We first describe how each of these variation sources is modeled and mitigated, and then explain how each mitigation technique is integrated into the proposed scheme to improve the chances of detecting an HW Trojan.

4.1 Modeling and Tracking the Process Drift

Process drift results in a non-uniform shift in the delay of different timing-paths. To model the timing impact of process drift, we design and train a Neural Network (NN) to act as a process tracking watchdog (NN-Watchdog) [56]. This NN-Watchdog is used to predict the difference between the slack reported by STA at design time and that sampled from fabricated IC at test time. To train the NN-Watchdog, we need a labeled data-set. Each data point in our data-set is a collection of 48 input features and a label value. The input features, detail of which is in Table 4.1, are extracted from physical design EDA and the STA engine. The label for each data point is the difference between the slack reported by STA (at design time), and that obtained by CFST [29] (at test time).

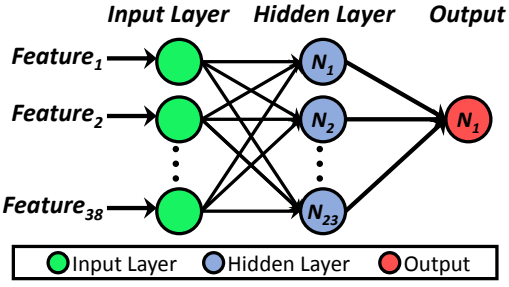
To assess the effectiveness of NN-Watchdog (and for lack of access to fabricated ICs), we modeled the process drift by extracting the shift in delay values from SPICE simulations performed using a skewed SPICE model. For this purpose, we first extracted the SPICE model for each timing-path in the input training. Then, to mimic a systematic process drift, the SPICE model was skewed such that the NMOS and PMOS transistors were ∼X%

Table 4.1: Description for each of 48 features, extracted from each timing-path for building the NN training set. (LP: Launch portion of timing-path, CP: Capture portion of timing-path, DP: Data portion of timing-path, M: Metal Layer, x: drive strength of the gate)

Total of 48 Features, 3 Feature Extracted from each timing-path		
Setup Time	Path delay reported in STA	Sum of fanout over cells in DP
45 Feature Extracted, 15 from each sub-path (CP, LP and DP)		
number of gates	subpath Delay	# cells of x0 strength
# cells of x1 strength	# cells of x2 strength	# cells of x4 strength
# cells of x8 strength	# cells of x16 strength	# cells of x32 strength
Total Length of M1	Total Length of M2	Total Length of M3
Total Length of M4	Total Length of M5	Total Length of M6

faster, and the Metal capacitance for Metal layers 1 to 7 was derated by Y%. Selection of X and Y gives us a consistently faster or slower process model. For example, the selection of $(X, Y) = (5, 5), (0, 0), (-5, -5)$ produces Fast, Typical, and Slow process models in our simulations. The resulting database was then divided into 1) training-set for training the NN (60% of timing-paths), 2) verification-set used for assessing the trained model accuracy while training (20% of timing-paths), and 3) test-set used for reporting the results (20% of timing-paths).

We then design and train a fully connected feed-forward NN (Fig. 4.1.(left)) as a process tracking watchdog that predicts the difference between the slack reported by STA and slack measured by the tester. To find a NN architecture with high accuracy, we utilized Keras [57] and trained a large number of models by sweeping various model parameters. The number of hidden layers was swept between 1 to 3, and the number of nodes in each hidden



Feature	Selection
Number of Input features from Table. 2	48
Number of Output Neurons (regression model)	1
Number of hidden layers	1
Number of Neurons in the hidden layer	23
Activation function	tanH
Optimizer	Adam
Loss function	MSE
Utilized Machine Learning Library	Keras

Figure 4.1: The configuration of the feed-forward fully-connected NN trained in this work to serve as a test-time process watchdog.

layer was swept from $[\frac{input+output}{2}$ to $\frac{2\times(input+output)}{3}]$. We also tested different activation functions including: $tanH$, $Sigmoid$, $ReLU$, $PReLU$, $Power$, Log, and Exp. The object of the training was defined to reduce the sum of squared distances (MSE) between the model's (slack shift) prediction and labeled data. The latency of the model can be also improved by designing dedicated accelerators like TCD-NPE [58] and NESTA [59].

We separately trained each model for AES128, Ethernet, and S38417 (from IWLS benchmark suit[60]) benchmarks using 21K, 20K, and 4K timing-paths for training, respectively. Data collected from the training of over 5K models revealed that the configuration that is shown in Fig. 4.1.(right) achieves the highest regression accuracy in most cases. When using a single "NVIDIA Tesla k80" GPU, the training time of this network for s38417, Ethernet and AES128 was approximately 1,4 and 5 hours respectively.

4.2 Modeling and Mitigating process variation

We divide the process variation into two categories: 1) *Random* Class that includes the independent intra-die process variation, and 2) *systematic* class including all forms of inter-die and correlated intra-die variation. We use two different mechanisms to deal with random and systematic process variation: (1) We perform speed binning on fabricated ICs and divided them into different speed bins (Fast, Normal, and Slow), arguing that ICs in the same bin are similarly affected by the systematic process variation. Then for each bin, we train an NN-Watchdog. (2) To reduce the impact of random process variation, using the formulation presented in Fig. 4.2, we collect the delay of each timing path (in our test set) from many ICs and compute their average delay to be used in our HW Trojan detection solution. When the timing-path delay is averaged across N different dies, the standard deviation of the random variable representing the average delay is N times smaller than the standard deviation of individual samples ($\sigma_{AVG} = \sigma_{sample}/N$). Note that the mean value is computed from discrete delay samples obtained from CFST test, and the tester's size (S), as illustrated in Fig. 4.2, affects the value of the computed mean.

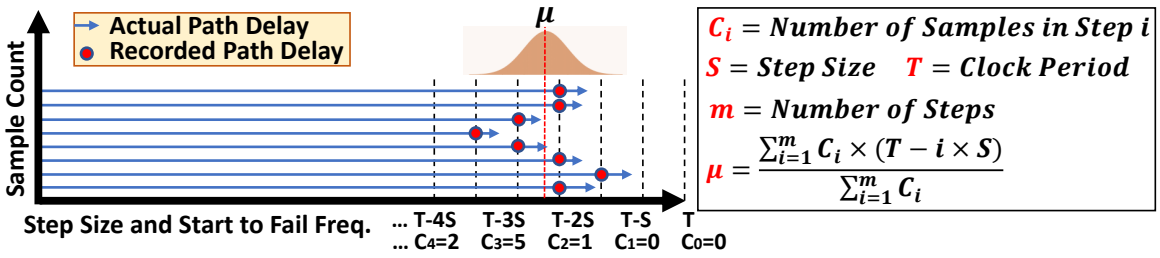


Figure 4.2: Computing the mean delay of a path using CFST delay measurements with step size S, clock period T, over m samples (dies).

To emulate the systematic process variation (within the same process corner), we created 2 additional derivatives (slightly modified copy) for each of our skewed SPICE models. Each skewed SPICE model was altered to make the transistors in the first derivative 1% slower, and in the second derivative 1% faster. To model Random process variation, each SPICE simulation is subjected to 100 Monte Carlo simulations (modeling CFST performed on 100 different dies in the same speed-bin), where the threshold voltage (V_{th}), Oxide thickness (T_{ox}) and channel Length (L) are varied (based on a normal distribution) to model the variation of path delays from chip to chip according to the expected variation in 32nm technology node.

4.2.1 Modeling Timing Impact of Voltage Noise

To improve the accuracy of our timing model (GTM), we utilize a methodology [54] that models the voltage drop and voltage noise. The voltage modeling flow models the voltage drop and endpoint uncertainty (due to the voltage-induced clock jitter) using a differential voltage pair (different voltages for launch and capture path of a timing-path). The differential voltage pair is obtained based on a statistical analysis performed on the design-specific IR simulation results. By using this voltage modeling scheme, the voltage-induced clock jitter uncertainty becomes path specific. This removes the need for a large hard margin, resulting in the majority of timing-paths to benefit from the smaller and dynamically computed margins. We first introduce a new metric, coined as **Delay Equivalent Voltage** (V_{DEV}) that could be used to express the effective voltage of a timing path. Then we

illustrate how we can extract and use this metric to margin a design against IR drop and voltage noise, and illustrate how the computed IR drop and voltage noise generated from this flow track the physical and PDN changes.

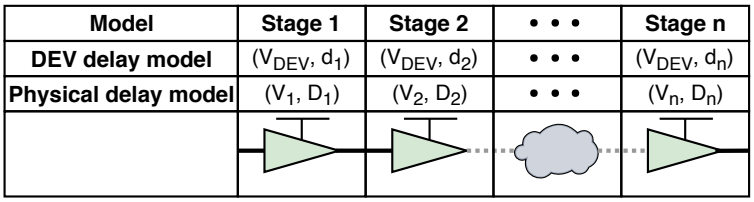


Figure 4.3: Inverter chain delay based on individual cell voltages when modeled by actual and V_{DEV} voltages.

4.2.2 Delay Equivalent Voltage

Consider the inverter-chain in Fig. 4.3. Each inverter, after physical placement, is connected to a different point of the on-chip PDN and experience a unique voltage signature. The Timing Window (TW) of a cell is defined as the time interval in which the cell propagates an arriving input signal to its output. *The supplied voltage to a cell can only affect the delay of a cell during its TW.* Let's assume that during its TW the inverter at stage i experiences the average voltage V_i, and accumulated delay of the inverter and next stage wire when voltage V_i is supplied, is D_i. Hence, the total delay of the inverter chain is $\sum D_i$. The V_{DEV} is now defined as a single voltage that when applied to all inverters in the chain, the delay of the chain remains unchanged. In another word, the application of V_{DEV} changes the delay of inverter i from D_i to a new delay d_i, such that: $\sum_{i=1}^{N} D_i = \sum_{i=1}^{N} d_i$.

Using the alpha power model, the delay of a cell is defined as:

$$D_i \approx \frac{k_i V_i}{(V_i - V_{th(i)})^{\alpha}} \tag{4.1}$$

In this equation k_i is a technology dependent constant, V_i and $V_{th(i)}$ are respectively the voltage and threshold voltage of i^{th} cell, and α is the velocity saturation constant, where based on choice of technology node and process is bounded by $1 < \alpha < 2$ [61]. By differentiating this equation over voltage, the delay impact of small variation in the supplied

voltage can be expressed as:

$$dD_i = -\frac{k_i - \frac{\alpha k_i V_i}{V_i - V_{th(i)}}}{(V_i - V_{th(i)})^\alpha} dV_i \tag{4.2}$$

Let's consider dD_i as the difference in delay of a cell when instead of V_i, the voltage V_{DEV} is applied. Additionally, let's define dV_i as:

$$dV_i = V_{DEV} - V_i \tag{4.3}$$

Based on its definition, when applying the voltage V_{DEV} to all cells in a logic path we expect the same delay (d_{path}) as of when each cell is annotated with its own unique voltage. More precisely, if the application of V_{DEV} to i^{th} cell in a logic path causes delay variation $dD(i)$, the overall path delay variation should be zero. Hence:

$$\Delta d_{path} = \sum_{i=1}^{N} dD_i = \sum_{i=1}^{N} -\frac{(k_i - \frac{\alpha k_i V_i}{V_i - V_{th(i)}}) \times (V_{DEV} - V_i)}{(V_i - V_{th(i)})^\alpha} = 0 \tag{4.4}$$

Let's define voltage headroom as $\Omega_i = \frac{V_i}{V_{th(i)}}$. After simplification and by using (4.1), we can rewrite the equation for V_{DEV} as:

$$V_{DEV} = \frac{\sum_{i=1}^{N} \frac{D_i[\Omega_i(1-\alpha)-1]}{\Omega_i - 1}}{\sum_{i=1}^{N} \frac{D_i[\Omega_i(1-\alpha)-1]}{V_i(\Omega_i - 1)}} \tag{4.5}$$

Based on this equation, the V_{DEV} of a timing path, by knowing the individual voltages of each cell could be calculated. In this equation, the $V_{th(i)}$ and V_i are known, and we only need the D_i, which is the delay of a cell when voltage V_i is applied. In order to compute the D_i quickly and effectively, we could use linear interpolation between delays specified in the library at different voltages. However, to improve the accuracy of the result, as illustrated in Fig. 4.4, we use non-linear interpolation between PVT corners defined using Composite

Current Source (CCS) delay libraries [62]. In order to enable CCS non-linear interpolation, we need to have at least 3 CCS-enabled standard cell libraries at different voltages (for the same process and temperature). With this setup, we can generate timing sessions for each voltage within the range of voltages covered by CCS libraries. Using this setup, we perform timing analysis for multiple voltages in the desired range of IR drop and generate additional voltage-delay reference points. Now, let's consider that after IR analysis we obtain a cell voltage to be V_i. We first identify two closest timing sessions whose applied rail voltage value encapsulates the V_i. Then we obtain the delay of the desired cell in each of the timing session. Let's denote the two timing sessions' voltages by V_{Slow} and V_{Fast} and the delay of the desired cell in each timing session by D_{Slow} and D_{Fast} respectively. The delay of the desired cell with voltage V_i could be obtained using the linear interpolation in equation 4.6. As illustrated in Fig. 4.4, by generating additional timing sessions (generated using CCS nonlinear interpolation), the error of the final linear interpolation for D_i is considerably reduced.

$$D_i = D_{Fast} + \frac{D_{Slow} - D_{Fast}}{V_{Slow} - V_{Fast}} \times (V_i - V_{Fast}) \tag{4.6}$$

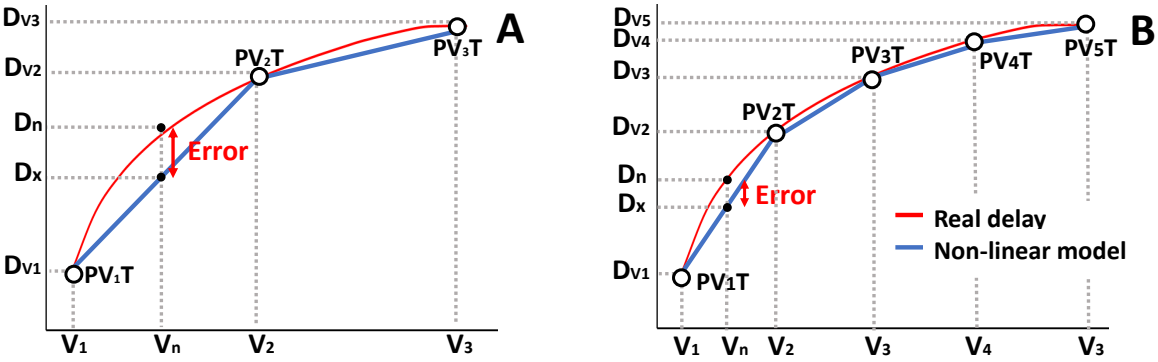


Figure 4.4: (left): Larger error for linear interpolation of a cell delay when using three timing session; (right): Generating two additional timing sessions using CCS non-linear interpolation followed by non-linear interpolation of the cell delay which resulting in a smaller interpolation error.

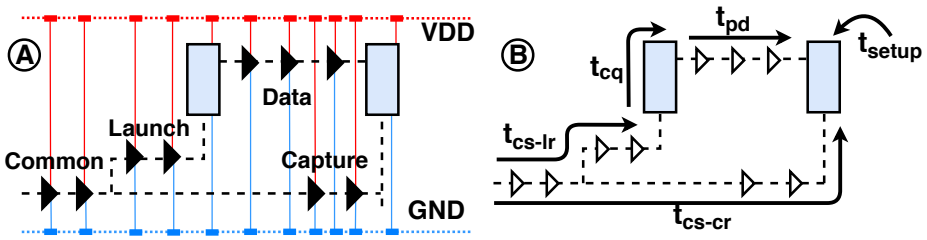


Figure 4.5: (left) The naming convention for different sections of timing path, (right): Delay components of a timing path

4.2.3 Using V_{DEV} for STA annotation

In order to improve the accuracy of GTM, we replace the IR drop and uncertainty hard margins, with a statistical representation of V_{DEV}, and bound the voltage of launch and capture sub-paths in each timing path separately. To do this, we rely on a recently supported feature of modern timing engines that support two different voltages for launch and capture paths when performing setup and hold timing checks.

The most contributing factor to clock Jitter is the dynamic change of voltage (voltage noise) from cycle to cycle. Hence, by providing a set of two different voltage for launch and capture path, we can capture the worst case clock jitter effectively and remove the related endpoint register's uncertainty altogether. Let's consider the timing path in Fig. 4.5 during a setup check in two consecutive clock cycles; In the first clock cycle, the voltage of the cells in the common, and launch portion of clock path, leading to the launch register, will determine how fast the clock reaches the clock pin of launch register. In the second cycle, the voltage of the cells in the common and capture portion of the clock, determine how fast the clock signal reaches the capture register's clock pin. Considering that voltage changes from cycle to cycle, the arrival time of the clock to the launch and capture registers changes at each cycle. The worst-case arrival time of the clock, leading to the worst-case jitter is when the voltage in the first cycle is low (late lunch), and in the next cycle is high (early capture). Hence, if based on a dynamic IR drop analysis, we can provide the expected worst-case rail voltage values (the IR drop), and could statistically determine the worst-case change in rail voltage value from a cycle to the next, we could completely remove the

uncertainty for the endpoint register, relying on the two (min and max) voltages to compute the worst-case jitter for each timing path. In this case, instead of using a fixed uncertainty value for the entire design, the amount of jitter is automatically computed for each timing path based on its launch and capture topology (the type and number of cells in each sub-path). Note that the original formulation for protecting the timing path against voltage noise jitter is overly pessimistic, as the degree of uncertainty was computed when worst case voltage noise was applied to the worst case topology (longest clock path). However, in our proposed formulation, the impact of voltage noise on clock jitter is determined based on topology of each timing path, reducing the extent of pessimism by avoiding the double margining (worst case voltage noise + worst case path) against voltage noise.

In order to derive the proper voltages to perform the setup and hold check in a timing engine, we need to set up an IR simulation. For this, the IR drop is divided into that of the $package + die$, and that of the *board*. The IR drop in the *board* is of both resistive and inductive nature. However, the frequency of the RLC oscillation of the *board* is usually in the range of KHz to few MHz, which is much smaller than die frequency. Additionally, due to the large difference in their time constant (and frequency), the cycle to cycle variation on the die and RLC oscillation on the board could be considered as independent. Hence, using ANSYS Redhawk [63] we only simulated the $package + die$ and used the worst-case of IR drop in the board as a constant IR drop. The worst-case IR drop in the board, in typical industrial designs, based on the quality of board and DECAP engineering, is between 2% to 4% [64]. The package s-parameter model is then extracted and used in IR simulation. The starting voltage for IR analysis (at package balls) was then set to voltage regulator's voltage minus 4% drop in the board.

For IR simulation, due to time-consuming nature of IR analysis, we are constrained to perform the IR analysis for no more than a couple of hundreds of cycles. Hence, in order to capture the worst-case scenarios in our IR simulation (worst case IR drop and cycle to cycle voltage variation), we resort to the following methodology: For a given netlist, we find its max-power vector, and fast profile the power consumption across thousands of cycle

using PrimeTime PTPX or Redhawk. From the obtained cycle-accurate power trace, we identify the following 5 scenarios: the part of the trace that contains (1) cycles resulting in the highest sustained power consumption (for at least 5 cycles), (2) cycles with largest increase in the power consumption from one cycle to the next, (3) cycles with largest drop in power consumption from one cycle to the next, (4) cycles with largest increase in the average power over two 10 cycles segments of the trace, and (5) cycle trace containing the largest decrease in the average power over two 10 cycle segments of the power trace. These 5 scenarios are chosen to detect the worst case IR drop and worst case cycle to cycle voltage variation (possibly due to phase change in the input vector) of the design. Each of this scenarios is padded with 30 cycles of pre-simulation (10 of which is ignored when computing the V_{DEV} resulting in a total simulation of less than 200 cycles). By simulating a netlist for 200 cycles, we will obtain an effective voltage per cell per cycle. Using this simulation environment, the timing engine's voltages for launch and capture rail values could be computed as follows:

Using taxonomy in Fig. 4.5.(right) the $Common{+}Launch{+}Data$ portion was considered as Launch-Path (LP) and the $Common + Capture$ portion was considered as Capture-Path (CP). Using methodology described in section 4.2.2, we extracted the V_{DEV} for each of LP and CP of each timing path. Let $V_{DEV_L}(C_i, LP_j)$ be the V_{DEV} of the LP of the j^{th} timing path at cycle $i->(i+1)$. The mean value of V_{DEV} across all cycles for all measured paths is obtained from:

$$\mu_{V_{DEV_L}} = \frac{1}{C \times P} \sum_{i=1}^{C} \sum_{j=1}^{P} V_{DEV_L}(C_i, LP_j) \tag{4.7}$$

In this equation, C is the number of simulated cycles; and P is the number of timing paths. The standard deviation of V_{DEV_L} is then used to capture the extent of launch voltage variation, which is obtained from:

$$\sigma_{V_{DEV_L}} = \sqrt{\frac{\sum_{i=1}^{C}\sum_{j=1}^{P}(V_{DEV_L}(C_i, LP_j) - \mu_{V_{DEV_L}})^2}{C \times P}} \tag{4.8}$$

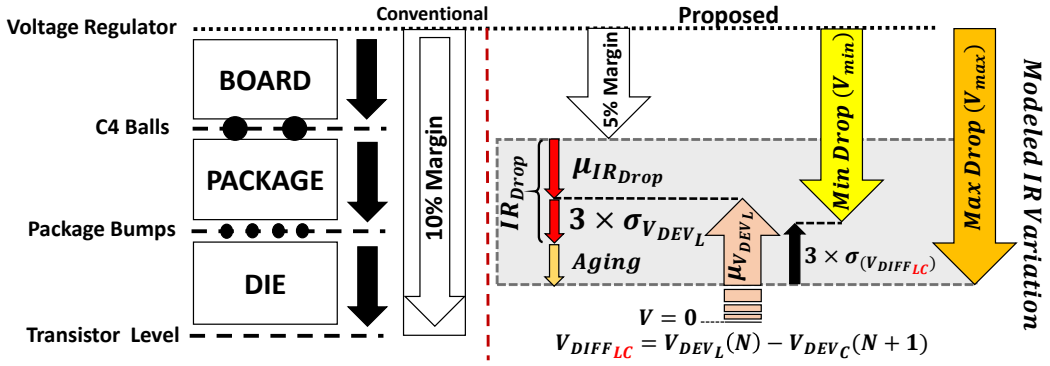


Figure 4.6: Modeling rail voltages, considering IR drop across board, package and die, for STA annotation.

To protect the circuit against aging effects such a Negative-Bias and Positive-Bias Temperature Instability (NBTI and PBTI) and Hot Carrier Injection (HCI), we could also include a measure $V_{Aging} = V_{NBTI} + V_{HCI} + V_{PBTI}$ to account for this phenomenon. At this point, as illustrated in Fig. 4.6, we compute the voltage representing the largest drop on a sub-path (V_{max}) using:

$$V_{max} = \mu_{V_{DEV_L}} - K \times \sigma_{V_{DEV_L}} - V_{Aging} \tag{4.9}$$

Where K (e.g. K=3) is the guardband factor. We then, need to determine the extent of the voltage noise. The voltage noise of timing path P, from cycle i to cycle $i+1$ is obtained from:

$$V_{Diff}[C_i, P_j] = V_{DEV_L}[C_i, P_j] - V_{DEV_C}[C_{i+1}, P_j] \tag{4.10}$$

Considering P timing paths, and C cycles, there exist $P \times (C-1)$ data points for the voltage noise for all investigated timing paths. Using this data points the $\sigma_{V_{Diff}}$ and $\mu_{V_{Diff}}$, similar to equations (4.7) and (4.8) are extracted. Using these values, the voltage

representing the maximum recovery from V_{max} within one cycle, denoted by V_{min} (minimum voltage drop) is obtained from:

$$V_{Min} = V_{Max} + 3\sigma_{V_{Diff}} + \mu_{V_{Diff}} \tag{4.11}$$

Note that V_{min} represents the maximum voltage departure of a timing path from V_{max} that is achievable in a single clock cycle. *By using V_{min} and V_{max} as voltage values for LP and CP of a timing path, both IR drop and voltage noise are modeled. The voltage difference allows the timing engine to effectively compute the jitter per timing path and there is no longer a need for using an uncertainty margin for register endpoints for this purpose. Additionally, the jitter will be unique for each timing path depending on its topology, reducing the unnecessary jitter margin for the majority of timing paths.* Note that if we had input test vectors covering all worst scenarios, and we were able to simulate the IR drop for very large input vectors, we could have annotated each timing paths, based on its own distribution of V_{DEV}. However, in reality, for IR analysis and for practical purposes, we are limited to IR simulation for 100s of cycles. For this reason, we still need to statistically margin the voltage drop and voltage variation. However, in this case, the computed voltages (1) are computed based on realistic worst-case values observed in the design and not the rule of thump. (2) track the changes in PDN and physical design. Hence, not only the physical designer can safely reduce the margins to voltages suggested by proposed voltage modeling scheme, but could also observer the IR and timing impact of changes in PDN and physical design, and make more informed decisions. Algorithm1 captures the process explained previously.

Algorithm 1 Computing V_{max} and V_{min} Rail Voltage Values

1: $\mu_{V_{DEV_L}} \leftarrow$ mean of $(V_{DEV_L}[Paths][Cycles])$; $K \leftarrow 3$;
2: $\sigma_{V_{DEV_L}} \leftarrow$ Standard deviation of $(V_{DEV_L}[Paths][Cycles])$;
3: $V_{max} \leftarrow \mu_{V_{DEV_L}} - (k \times \sigma_{V_{DEV_L}}) - V_{Aging}$;
4: **for all** C in $(Cycles - 1)$ **do**
5: **for all** P in $Paths$ **do**
6: $V_{diff}[P][C] \leftarrow V_{DEV_L}[P][C] - V_{DEV_C}[P][C+1]$;
7: $\mu_{V_{Diff}} \leftarrow$ mean of $(V_{diff}[Paths][Cycles])$;
8: $\sigma_{V_{Diff}} \leftarrow$ Standard deviation of $(V_{diff}[Paths][Cycles])$;
9: $V_{min} \leftarrow V_{max} + (K \times \sigma_{V_{Diff}}) + \mu_{V_{Diff}}$;

Usage of computed voltages is straightforward; State-of-the-art timing engines support dual rail voltages for LP and CP. For example, using Synopsys PrimeTime [65], the V_{max} and V_{min} could be applied using the following command:

$$PT > set\_rail\_voltage \quad -dynamic \quad -vmin < V_{min} > \tag{4.12}$$

$$PT > set\_rail\_voltage \quad -dynamic \quad -vmax < V_{max} > \tag{4.13}$$

Note that the proposed rail voltage modeling is far less pessimistic that annotating each cell with its worst case and best case voltage (as adopted by recent EDA tools) for the purpose of setup and hold timing check. This is because by using V_{DEV} we have accounted for the accumulated impact of delay variation due to individual cell voltage drops across all cells in a timing path. In addition, this voltage modeling technique accounts for the maximum voltage difference between the launch and capture portion of a timing path that **could be developed within one clock cycle**. Whereas if the cell voltage is annotated with their highest and lowest observed voltage (as adopted by several EDAs), such differential voltage is substantially exaggerated and is not based on physical reality, as such differential voltage could not be developed within one clock cycle.

4.3 Trojan Detection Flow

Fig. 4.7 shows the overall flow of the proposed Trojan detection flow. We augment the design stage with an additional step for statistical modeling of the voltage noise and IR drop using proposed voltage modeling. Accordingly, the STA reports the timing slack of each timing path based on its estimate of voltage drop and voltage noise (as opposed to a global pessimistic margin). This, as we will illustrate in the result section, will improve the correlation between timing slack predicted by timing engine, and the timing slack observed at test time using CFST. The final GDSII is then sent to the foundry for fabrication. The fabricated ICs may be tested in the untrusted foundry for functionality. The working ICs are then sent to a trusted facility for Trojan detection.

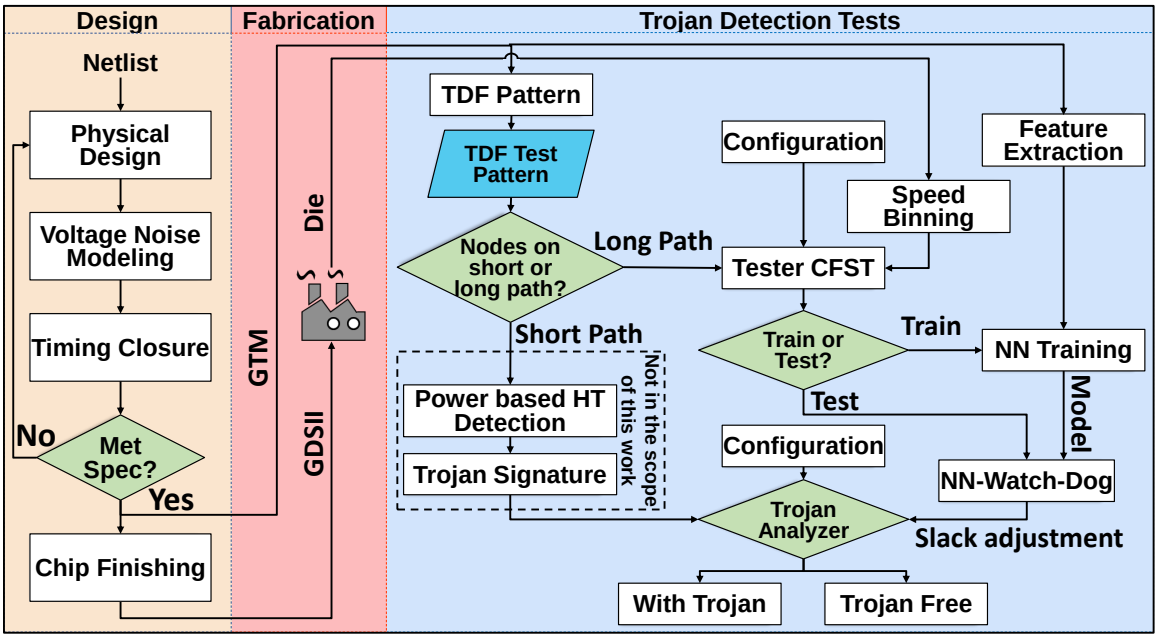


Figure 4.7: Trojan Detection Flow: The model includes changes in the design and test stages. The test stage divides the timing-paths into long and short paths. The short paths are subjected to power side-channel Trojan detection as described in [1] (not covered in this paper), and the long paths are subjected to delay side-channel analysis using GTM as reference timing model, adjusted by a NN that is trained as a process watchdog and by using CFST to find the start-to-fail frequencies for timing-paths under test.

To detect a Trojan, we need to find the TT/TP induced slack change. As Fig. 3.1 shows, a TT adds capacitive load to driving cell of its *observed* net, and the TP appends an additional gate delay to every timing path that passes through its *victim* net. To detect a victimized or a monitored net (by a TP or TT), and for having no prior knowledge on which nets are affected, we need to include all nets in our delay analysis. We define a P2P-wire as a net that connects the output pin of a driver cell (or a primary input) to the input pin of one of its fanout cells (or a primary output). Hence a gate with a fanout of 4 has 4 P2P-wires. Each P2P-wire will be tested for rise and fall transitions. To increase the detection rate and to account for process variation, this process may be repeated for N different timing-paths passing through that net. The second criteria for selecting the timing-paths is the maximum frequency of the tester equipment; The delay of the selected paths should be larger than the limit imposed by the maximum reachable frequency of the tester equipment. If the P2P-wire in no timing-path is long enough for CFST, it is

regarded as a candidate for Trojan detection via power-based detection schemes. Note that timing-paths with a small number of gates (in their data sub-path) have high controllability, making them ideal for the power-based Trojan detection schemes (e.g. [19–21, 44]) that rely on full or partial activation of such paths. For all other timing-path candidates, we generate the Path Delay Fault (PDF) test vectors using an Automatic Test Pattern Generation tool (ATPG). If ATPG cannot generate a test pattern for a path, the path selection changes. If ATPG cannot generate a test vector for any path through that P2P-wire, it is discarded.

Algorithm 2 Trojan Detection Flow

1: N = # paths to be tested through each net in the design
2: $Nets \leftarrow$ all nets in the design.
3: **for all** *net* in $Nets$ **do** ▷ net selection of Path Delay Fault (PDF) test
4: $TimingPaths$ += select N timing-paths passing through *net*
5: Perform speed binning on all dies and assign them to B bins.
6: **for all** *bin* in B **do** ▷ NN training
7: $NN_{bin} \leftarrow$ Train a NN-Watchdog according to the algorithm 3
8: $\sigma_{NN_{bin}} \leftarrow$ the standard deviation of NN_{bin}
9: **for all** *die* in *bin* **do**
10: Slack = 0
11: **for all** *path* in $TimingPaths$ **do**
12: CFST(bin,die,path) $\leftarrow$ *path* slack measured by CFST *die* in the *bin*
13: Slack(bin,path) += CFST(bin,die,path)
14: **for all** *path* in $TimingPaths$ **do**
15: μ_S(bin,path) = Slack(bin,path)/sizeof(bin);
16: T_{Th}= 4×$\sigma_{NN_{bin}}$ ▷ Detection Threshold = 4σ to reduce false positive
17: **for all** *path* in $TimingPaths$ **do**
18: GTM(path) $\leftarrow$ query the slack of path from GTM
19: NNSD(path) $\leftarrow$ slack shift suggested by $NN_{bin}(path)$
20: AS(path) = GTM(path) + NN$_{Watchdog}$(path) ▷ Adjusted Slack
21: $\delta = \mu_S$(bin,path) - AS(path) ▷ Shifted delay after adjustment
22: **if** ($\delta > T_{Th}$) **then** ▷ Trojan Classifier
23: Likely Trojan Set $\leftarrow$ *path*

The Alg. 2 describes our proposed Trojan detection flow. As described in this algorithm, after selecting the set of timing paths for PDF testing, we speed-bin the fabricated dies. In the next step, we collect the NN-Watchdog training data using the flow described in algorithm 3. Then, we train a process tracking NN-Watchdog for each bin and extract the standard deviation of each NN-Watchdog in predicting the shifted delays. For each bin, we perform CFST and measure the start to fail frequencies for the selected timing-paths. The slack difference (δ) between the mean of slacks reported by the CFST and the NN-Watchdog adjusted slack from GTM (in the same bin) represents the likelihood of a timing

path being affected by a Trojan. To make a binary decision, we use a threshold to assess the significance of δ and classify the timing paths into benign or malignant (Trojan) classes.

Algorithm 3 Generating a training set for the NN-Watchdog

1: $NP \leftarrow mR^2$ ▷ R is the registers count, and m is a large number (e.g. 10)
2: $TimingPaths \leftarrow$ Select NP timing-paths (min of m path per endpoint)
3: **for all** *path* in $TimingPaths$ **do**
4: feature(*path*) $\leftarrow$ Extract *path* features from GTM ▷ input feature
5: GTM(*path*) $\leftarrow$ Extract *path* slack from GTM
6: $Slack$(path) = 0
7: **for all** *die* in *Dies* **do**
8: **for all** *path* in $TimingPaths$ **do**
9: CFST(*die*,*path*) $\leftarrow$ Slack of *path* in CFST test of *die*
10: Slack(*path*) += CFST(*die*,*path*)
11: **for all** *path* in $TimingPaths$ **do**
12: Slack(*path*) = Slack(*path*)/NP;
13: Δ_{slack}(*path*) = Slack(*path*) - GTM(*path*) ▷ label
14: data-points(*path*) $\leftarrow$ (features(*path*),Δ_{slack}(*path*))

When choosing a value for Trojan-detection threshold, we face a trade-off between the false positive rate and the accuracy of Trojan detection. The false positive could be the result of 1) inaccuracy in the GTM, 2) inaccuracy of NN, and 3) random process variation for sampling over a small number of ICs. To reduce the false positive rate, the threshold used for detection should be large enough, to account for these. Since we average the delay of each timing-path over many IC samples, the impact of random process variation in the average delay could be reduced to a desirable range. However, we still have to account for the inaccuracy of the NN and systematic variation. Hence, we define the detection threshold to be $T_{Th} = n \times max(\sigma_{NN}, \sigma_{processvariation})$, in which the $\sigma_{processvariation}$ is the expected variance of systematic process variation (excluding random) and σ_{NN} is the standard deviation of the NN. Since σ_{NN} is the aggregated impact of NN inaccuracy (for under-fitting or over-fitting of the trained model) and impact of systematic process variation, the variance of σ_{NN} tends to be larger than $\sigma_{processvariation}$, and we can simply use $T_{Th} = n \times \sigma_{NN}$ (n is selected as 4 in Alg. 2).

To verify the choice of threshold values T_{Th}, we utilized Youden[66] method to extract the threshold value from a Receiver Operating Characteristic (ROC) curve that we generate over our SPICE simulation data (details in chapter 5). *Note that at test time, we do not know*

which timing-paths are affected by HW Trojan. Hence, the optimal threshold of detection cannot be determined using the Youden method.

Change in the temperature affects the speed of transistors and alters the RC characteristics of the connecting wires. But, the temperature change is an extremely slow phenomenon. That's why one can design temperature sensors with sampling frequencies far lower than operational clock frequency [67,68]. At test time, a test vector is loaded into the scan chain using a slow clock, then the circuit operates at-speed for two cycles (launch and capture) using a fast clock. Finally, the scan is offloaded using a slow clock. The heat dissipation when using a slow clock is quite low, and the duration of at-speed test is only two cycles for each test pattern, limiting the extent of temperature changes to a fraction of a degree Celsius. Hence, at test time the die temperature can be tightly controlled to discount the delay impact of temperature variations.

Chapter 5: Results and Discussion

In this chapter, we first look at the improvements obtained by employing proposed voltage modeling, then we look at the accuracy of the NN-Watchdog in tracking the process drift, and then we present the result of applying our proposed test flow, for Trojan detection.

5.1 Proposed Voltage Modeling Accuracy

In this section, the accuracy of our flow in modeling the voltage noise and improvement in the timing closure are quantified.

5.1.1 Verification of Delay Equivalent Voltage

For the verification purpose, we used Ethernet, S38417 and AES128 netlists from IWLS benchmark suit [60]. Using Synopsys Design and IC Compiler, we first hardened these IPs using 32nm cell libraries. Then we run a dynamic vectorless IR simulation using Ansys Redhawk and extracted cell voltages for 100 cycles of dynamic simulation, padded with 10 cycles of pre-simulation using a toggle rate of 10% and 100% for data and clock cells respectively. Subsequently, we collected 4K timing paths from the routed design and calculated the V_{DEV} for each timing path. To accurately model the behavior of a timing path for setup timing check, the voltages applied to the launch and capture path should come from two consecutive cycles. For this reason, for each cycle of simulation, we have computed the V_{DEV} for each of launch and capture segment of each timing path for every cycle. In the SPICE simulation of a timing path, when its LP is annotated with the V_{DEV} of cycle n, the CP is annotated with V_{DEV} of cycle $n+1$. For the base case, we also annotate the LP and CP of each timing path with voltage observed in two consecutive cycles from Redhawk analysis. Hence, when IR simulation for C cycles is available, the setup check could be

constructed $C-1$ times. The hold check, on the other hand, uses the voltage of the launch and capture in the same cycle.

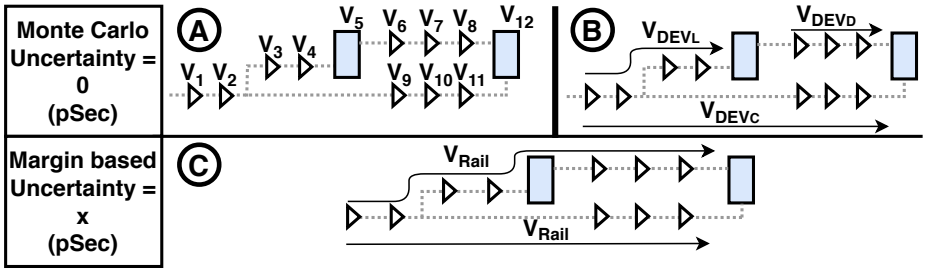


Figure 5.1: setup for a) the SPICE simulation when using actual voltages obtained from Redhawk; b) the SPICE simulation when using computed V_{DEV} voltages for LP and CP; c) the SPICE simulation when using hard margins (using 10% IR drop and 5% uncertainty)

Fig. 5.1 (A and B) illustrate our setup for two sets of SPICE simulation, one when V_{DEV} is used and one when individual cell voltages are applied, and the resulting slack is compared. In order to further illustrate the accuracy of V_{DEV}, a third SPICE simulation is set up where the timing paths slacks are computed using the 10% IR drop rule for the cell voltages and 5% rule for the uncertainty. For verification, we computed the difference in the computed slacks when actual voltages are applied to that of when the modeled voltages (V_{DEV} or voltage obtained from 10% drop) is applied. In the SPICE simulation, the slack is computed using:

$$Slack = t_{cs-cr} + T_{clk} - t_{cs-lr} - t_{clk-q} - t_p - t_{setup} - U \tag{5.1}$$

where t_{cs-lr} and t_{cs-cr} are the delays of clock path from its source to the launch and capture register respectively, t_p is the longest propagation delay of data path, T_{clk} is the clock period, t_{clk-q} and t_{setup} are the inherent clock to Q, and setup delay for the launch and capture registers respectively, and U is the applied uncertainty. Using this equation, the slack differences are obtained from equations:

$$\Delta_{DEV} = |Slack_{Actual} - Slack_{V_{DEV}}| \tag{5.2}$$

$$\Delta_{Conv} = |Slack_{Actual} - Slack_{V_{Conv}}| \tag{5.3}$$

In this equations, $Slack_{Actual}$ is the slack obtained from the application of actual voltages in each cycle, the $Slack_{V_{DEV}}$ is the slack obtained from application of V_{DEV} values where uncertainty constraint is set to zero, and $Slack_{V_{Conv}}$ is the slack obtained by application of a rail voltage and voltage noise related uncertainty (similar to Synopsis PrimeTime). The histogram obtained from SPICE simulations of the timing paths for Δ_{slack} is illustrated in Fig. 5.2. As illustrated, for both benchmarks the Δ_{DEV} is a zero-mean distribution with much smaller standard deviation compared to the Δ_{Conv}. Smaller difference verifies the smaller error. Considering the Ethernet, S38417 and AES128 were designed for 1.4 GHz of frequency, the maximum path delay error for both benchmarks is reduced from $\sim 10\%$ when using the conventional model, to around $\sim 1\%$ when using V_{DEV}.

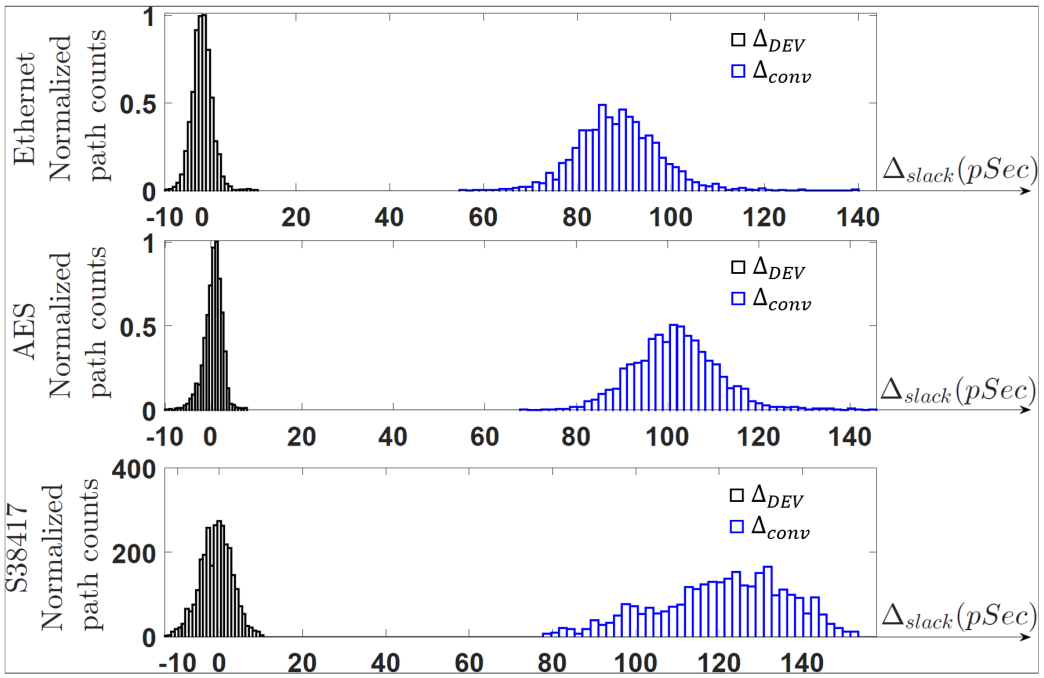


Figure 5.2: The timing slacks in three nearly timing closed design (Ethernet, AES128 and S38417) using conventional margin based and V_{DEV} flow for generation of GTM

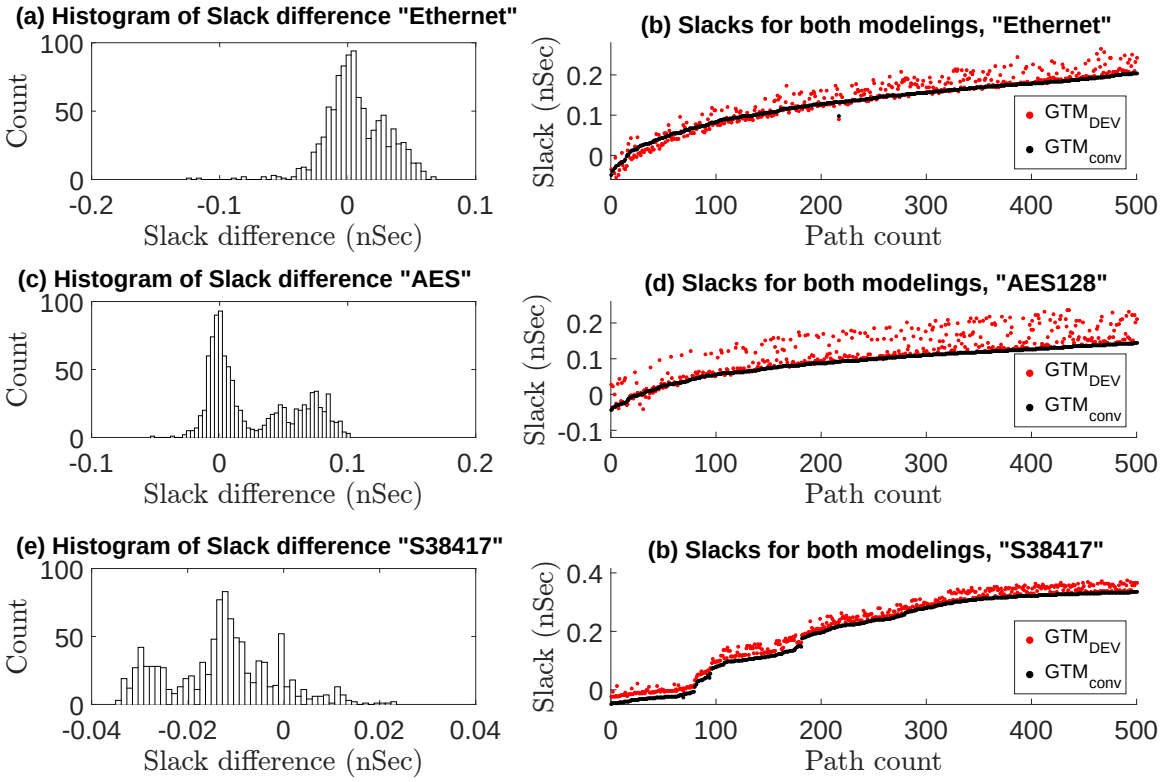


Figure 5.3: V_{DEV} based slacks v.s. margin-based slacks

5.1.2 Improvement in STA accuracy

To illustrate the timing impact of using rail voltages driven from V_{DEV} modeling flow, we performed a case study on Ethernet, S38417 and AES128 benchmarks. The STA was once obtained based on conventional flow (STA_{Conv}) and once using the proposed voltage modeling flow ($STA_{V_{DEV}}$) for IR drop and voltage noise.

Fig. 5.3 illustrates the available slack for the critical timing path in STA_{Conv} and the recalculated slack based on $STA_{V_{DEV}}$. The slacks are sorted in ascending order as reported by STA_{Conv}. Hence, at each X location of this graph, the black dot represents the available timing slack based on the conventional hard-margin-based timing analysis flow, and the red dot represents the new timing slack obtained by using the proposed voltage modeling scheme. As illustrated, most timing paths see an additional timing slack, some as large as 100 pSec. From this graph, we can easily observe that the conventional flow, has penalized many timing paths with unnecessary margins. These margins, if available during physical

Table 5.1: The Accuracy of the NN-Watchdog regression model trained for different benchmarks. The μ and σ are the Mean and Standard deviation of the regression error over the validation set. As discussed in Section 4.1, the Fast, Typical and Slow process are simulated using skewed Spice model with (X,Y) = (5,5), (0,0), (-5,-5), respectively.

Benchmarks	**Gate Count**	**Size**		**Fast**		**Typical**		**Slow**	
		Train	**Test**	μ**(ps)**	σ**(ps)**	μ**(ps)**	σ**(ps)**	μ**(ps)**	σ**(ps)**
AES128	114K	21K	4K	-0.14	7.45	0.04	8.12	-0.02	7.15
Ethernet	40K	20K	4K	0.79	9.65	0.28	9.13	-0.65	8.36
S38417	6K	4K	1K	0.12	6.87	0.08	7.07	0.25	6.38

design flow, could be used for improving the Power, Performance and Area (PPA) of design by means of introducing additional V_T swapping or cell downsizing. In addition, in both benchmarks, there are several timing paths that are timing closed in STA_{Conv}, however, we see violation in $STA_{V_{DEV}}$, indicating that the original margins were not pessimistic enough. Hence, using $STA_{V_{DEV}}$ could discover and fix this types of violations.

5.2 NN-Watchdog Accuracy

Table 5.1 depicts the mean and standard deviation of the NN-Watchdog in predicting the shift in the delay of timing-paths when subjected to process drift. As shown, the standard deviation is reasonably small. To put this in perspective, we can compare the error distribution of NN-Watchdog with the error distribution obtained by finding the difference between delay of timing-paths reported by SPICE (d_{SPICE}) and that obtained from STA (d_{STA}). Fig. 5.4 depicts the distribution of NN-Watchdog error and mean-shifted delay-difference model ($\Delta_{SPICE-STA} = d_{STA} - d_{SPICE}$) over a large selection of timing-paths. As illustrated, the mean shifted SPICE-STA difference, for all benchmarks, has a much larger standard deviation compared to the NN-Watchdog error. This reveals the strength of NN-Watchdog and justifies why an NN-Watchdog could significantly enhance our Trojan detection flow by accurately adjusting the STA reported delay information to account for the impact of process drift.

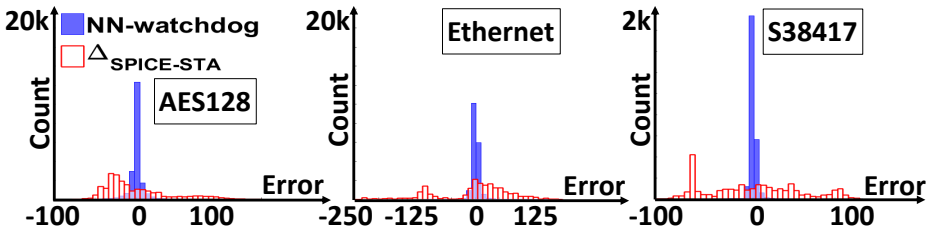


Figure 5.4: Histogram of NN-Watchdog Error trained for different benchmarks.

5.3 HW Trojan Detection Accuracy

Setup: We selected 720 timing-paths from non-critical to critical range, covering a range of 400 ps of slack from 3 largest IWLS benchmarks [60] (Ethernet, S38417 and AES128). Each benchmark is hardened (physical design) and timing closed at 1.4 GHz in 32nm technology. For each benchmark, we divided the selected timing-paths into two groups (360 each) for inserting TTs and TPs. We further divided each subgroup into three smaller groups of 120 paths each to implement small, medium, and large size Trojans. The TP size is controlled by the selection of logic gates with different inherent delays. The TT size is controlled by the distance it is placed from the triggering net. During NN-Watchdog training, we do not know if a timing-path selected for training contains a Trojan. Hence, we also evaluated the impact of including Trojans affected timing paths in the training; We trained 3 NN-Watchdogs with 0, 20 and 40 Trojan paths included in their training set. The rest of the Trojans are used for evaluating the proposed Trojan detection accuracy as a part of its test-set.

To model the voltage variation, we used Redhawk [63] and simulated 50 cycles of vectorless IR simulation when clock and data toggle rates are 100% and 10% respectively. In the SPICE simulation, each timing-path is assigned a random value from a normal distribution for the V_{th} of its transistors (to model the process variation), and each of its gates is annotated with the gate voltage reported by Redhawk in one simulation cycle. Note that each SPICE simulation presents a CFST test performed on a different die at a different time. Furthermore, the slack reported by the SPICE simulation for each timing-path was adjusted to the neighboring larger clock sweeping frequency step, modeling the CFST step

Table 5.2: Threshold values used for TT and TP Trojan detection in Fast-bin in Algorithm 2

Benchmarks	**TP**		**TT**	
	Youden	$4 \times \sigma_{NN}$	**Youden**	$4 \times \sigma_{NN}$
AES128	27.1	29.86	16.3	29.86
Ethernet	35.5	38.67	15.4	38.67
S38417	24.7	27.46	17.2	27.46

size. The step size in the state-of-the-art tester equipment can be as small as 10-15ps. Hence, we selected the step size of the tester as 15ps.

In our simulations, we assessed the effectiveness of Trojan detection using 3 approaches. 1) Shifted STA (SSTA): when STA results are used as Golden Timing Model to detect HW Trojans. The process drift makes the direct usage of STA results quite ineffective. To account for process drift in SSTA, we have computed a static shift value, obtained from averaging the observed shift from many sampled timing-paths, and have shifted all reported slacks by STA using this value. For this approach, we have set the detection threshold to the fixed value of *45ps* which is the delay of a 2-input NAND gate in our standard cell library. 2) Shifted GTM (SGTM): which is similar to SSTA with the exception that the voltage noise and IR-drop are modeled using V_{DEV} voltage modeling, and the Trojan detection threshold is set to *45ps*. 3) Neural shifted Golden Timing Model (NGTM) in which the voltage noise is modeled using V_{DEV} voltage modeling, while the process drift is modeled using NN-Watchdog. The NGTM represents the proposed and utilized detection model. Furthermore, we have investigated the accuracy of NGTM when the training set includes 0, 20 and 40 timing-paths affected by HW Trojans. In all of SSTA, SGTM and NGTM, the effectiveness of the selected threshold is assessed by extracting the optimal threshold from the ROC curve using Youden[66] method.

Fig. 5.5 captures the result of TP detection in Fast (X,Y)=(5,5) speed bin. The top row compares the accuracy of SSTA, SGTM, and NGTM in detecting TPs, and the middle row reports the false positive rate of detection for each model across different benchmarks. The NGTM model is reported 3 times, corresponding to a model having 0, 20 and 40 Trojan paths included in its training set. The bottom row illustrates the ROC curve from which

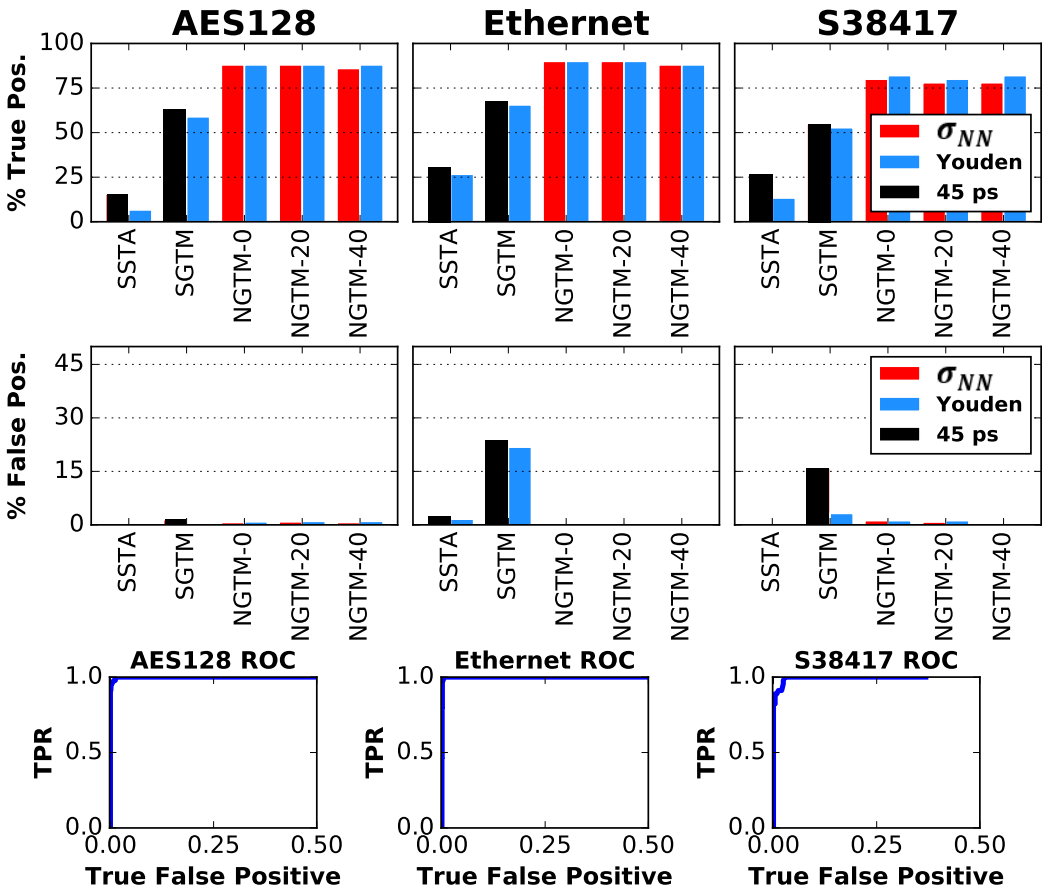


Figure 5.5: Trojan Payload detection results for 3 benchmarks. (top): Detection rate, (middle): False positive rate, (bottom): Associated ROC curve capturing the True Positive Rate (TPR) versus True False Positive Rate. The SSTA bar represents the HW Trojan Payload detection using a (Mean shifted) STA. The SGTM represents Trojan detection when V_{DEV} voltage modeling flow is deployed. The NGTM bars represent the Trojan Payload detection when both V_{DEV} voltage modeling approach and the NN-Watchdog are combined. Each bar shows the NN trained when X Trojans are included in the training set, with $X \in \{0, 20, 40\}$.

the Youden threshold (as described in Section 4.3) is extracted. The threshold values used for detection using each of these methods is reported in Table 5.2. As illustrated, the usage of V_{DEV} voltage modeling in SGTM model improves the TP detection rate compared to the SSTA at the expense of higher false positive. However, the use of V_{DEV} voltage modeling and NN-Watchdog in the NGTM not only results in a significantly higher increase in the TP detection rate (to over 88%), but also significantly depresses the false positive rate. This confirms the ability of NN-Watchdog in modeling the complicated, non-linear, path-specific shift of delays resulting from process drift. Finally, note that the presence of a small number

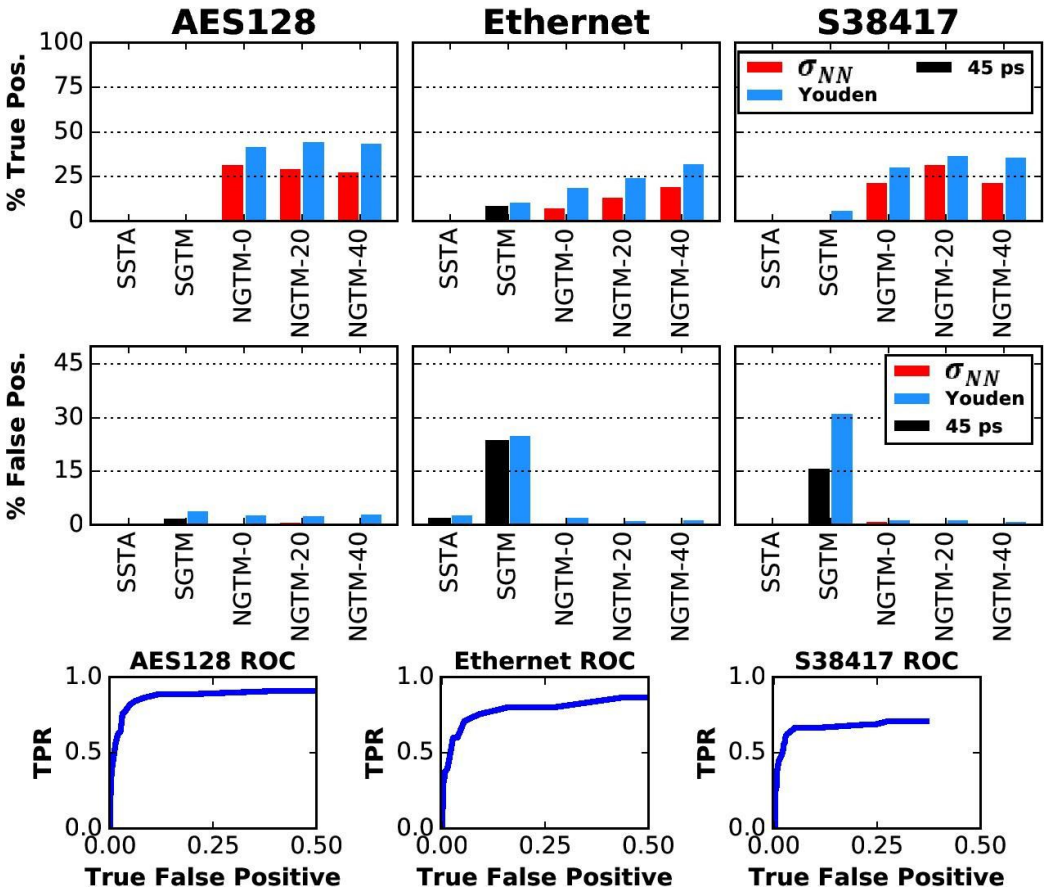


Figure 5.6: Trojan Trigger detection results for 3 benchmarks. (top): Detection rate, (middle): False positive rate, (bottom): Associated ROC curve capturing the True Positive Rate (TPR) versus True False Positive Rate.

of Trojans in the training set does not affect the accuracy of trained NN-Watchdog as the impact of a few samples in a large training set is statistically insignificant.

Figure 5.6 depicts the result of our TT detection in the FAST speed bin with (X,X) = (5,5). As shown, NGTM has a lower rate for detecting TTs compared to TPs due to the smaller impact of TT on the delay of affected observed nets compared to TP (which is at least equal to one gate delay). Similar to the TP case, we observe that contamination of the training set with few HW Trojan data points does not impact the accuracy of trained NN-Watchdog. This is because the number of HW Trojan infested timing paths is statistically insignificant and does not affect the training results. As illustrated, the Trojan trigger detection using our proposed approach closely tracks the Yuden model. Note that

Table 5.3: Percentage of False Positives (FPo) and True Positives (TPo) when the proposed model (as described in Alg. 2) with NGTM-10 is used for detection of TP in Slow, Typical, and Fast speed bins.

Benchmarks	**Slow**		**Typical**		**Fast**		**No-Binning**	
	TPo	**FPo**	**TPo**	**FPo**	**TPo**	**FPo**	**TPo**	**FPo**
AES128	88.6	0.11	87.8	0.17	86.1	0.18	0.78	0.31
Ethernet	87.3	0.17	85.5	0.12	88.6	0.15	0.80	0.48
S38417	83.7	0.19	82.2	0.23	80.3	0.39	0.77	0.45

extracting the Yuden threshold requires a Trojan oracle database that is not available and is only presented to illustrate the effectiveness of our proposed solution. Finally, note that a hardware Trojan can have multiple TT and TP; although we have separately reported the result of TT and TP detection, detection of a single TT or TP is enough to detect the hardware Trojan. Therefore the overall detection rate of a hardware Trojan is larger than the results reported for TT or TP detection.

Table 5.3 captures the results of TP Trojan detection in all speed bins. As reported, the speed binning provides more accurate results for TP detection compared to the No-speed-binning case. This is due to the larger standard deviation of the NN-Watchdog when training over extracted delays from all dies without considering the impact of systematic process variation.

5.4 Conclusion

In this proposal, we presented a novel variation modeling and timing signature, and a promising methodology for Trojan detection based on side-channel delay analysis, that does not require the availability and usage of a Golden IC. For Trojan detection, The proposed scheme relies on 1) improving the timing model at design time to account for voltage noise, and 2) training a Neural Network at test time that is used as a process tracking watchdog to model the process drift (while accounting for process variation). We have reported that our Trojan detection flow could achieve close to 90% Trojan detection in the selected benchmarks.

Bibliography

[1] M. Lecomte et al., "An on-chip technique to detect hardware trojans and assist counterfeit identification," *IEEE Trans. on VLSI Systems*, vol. 25, no. 12, pp. 3317–3330, 2017.

[2] A. Yeh, "Trends in the global ic design service market," *DIGITIMES research*, 2012.

[3] A. Vakil, F. Niknia, A. Mirzaeian, A. Sasan, and N. Karimi, "Learning assisted side channel delay test for detection of recycled ics," in *Asia and South Pacific Design Automation Conf.*, 2021.

[4] K. Zamiri Azar, H. Mardani Kamali, H. Homayoun, and A. Sasan, "Threats on logic locking: A decade later," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 471–476.

[5] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "Smt attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the sat attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 97–122, 2019.

[6] K. Z. Azar, F. Farahmand, H. M. Kamali, S. Roshanisefat, H. Homayoun, W. Diehl, K. Gaj, and A. Sasan, "Coma: Communication and obfuscation management architecture," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 181–195.

[7] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "Nngsat: Neural network guided sat attack on logic locked complex structures," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[8] S. Roshanisefat, H. K. Thirumala, K. Gaj, H. Homayoun, and A. Sasan, "Benchmarking the capabilities and limitations of sat solvers in defeating obfuscation schemes," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*. IEEE, 2018, pp. 275–280.

[9] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2018, pp. 405–410.

[10] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[11] H. M. Kamali et al., "On designing secure and robust scan chain for protecting obfuscated logic," *Great Lakes Symposium on VLSI (GLSVLSI)*, 2020.

[12] H. M. Kamali, K. Z. Azar, S. Roshanisefat, A. Vakil, and A. Sasan, "Extru: A lightweight, fast, and secure expirable trust for the internet of things," *14TH IEEE Dallas Circuits and System Conference (DCAS)*, 2020.

[13] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Interlock: An intercorrelated logic and routing locking," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[14] H. M. Kamali et al., "SCRAMBLE: The state, connectivity and routing augmentation model for building logic encryption," *arXiv preprint arXiv:2005.11789*, 2020.

[15] S. Roshanisefat, H. M. Kamali, K. Z. Azar, S. M. P. Dinakarrao, N. Karimi, H. Homayoun, and A. Sasan, "Dfssd: Deep faults and shallow state duality, a provably strong obfuscation solution for circuits with restricted access to scan chain," in *2020 IEEE 38th VLSI Test Symposium (VTS)*. IEEE, 2020, pp. 1–6.

[16] S. Roshanisefat, H. M. Kamali, H. Homayoun, and A. Sasan, "Sat-hard cyclic logic obfuscation for protecting the ip in the manufacturing supply chain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 954–967, 2020.

[17] S. Roshanisefat, H. Mardani Kamali, and A. Sasan, "Srclock: Sat-resistant cyclic logic locking for protecting the hardware," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, 2018, pp. 153–158.

[18] N. Karimi, J.-L. Danger, and S. Guilley, "On the effect of aging in detecting hardware trojan horses with template analysis," in *International Symposium on On-Line Testing And Robust System Design (IOLTS)*. IEEE, 2018, pp. 281–286.

[19] D. Agrawal et al., "Trojan detection using IC fingerprinting," in *Security and Privacy, 2007. SP'07. IEEE Symp. on.* IEEE, 2007, pp. 296–310.

[20] R. Rad, et al., "A sensitivity analysis of power signal methods for detecting hardware trojans under real process and environmental conditions," *IEEE Trans. on VLSI Systems*, vol. 18, no. 12, pp. 1735–1744, 2010.

[21] H. Salmani et al., "New design strategy for improving hardware trojan detection and reducing trojan activation time," in *IEEE Int. Workshop on Hardware-Oriented Security and Trust*, 2009, pp. 66–73.

[22] Y. Liu et al., "Hardware trojan detection through golden chip-free statistical side-channel fingerprinting," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

[23] C. Lamech et al., "Rebel and tdc: Two embedded test structures for on-chip measurements of within-die path delay variations," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2011, pp. 170–177.

[24] R. Rad et al., "Sensitivity analysis to hardware trojans using power supply transient signals," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 2008, pp. 3–7.

[25] R. M. Rad et al., "Power supply signal calibration techniques for improving detection resolution to hardware trojans," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 632–639.

[26] D. Du et al., "Self-referencing: A scalable side-channel approach for hardware trojan detection," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 173–187.

[27] K. Hu et al., "High-sensitivity hardware trojan detection using multimodal characterization," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1271–1276.

[28] Y. Liu et al., "Hardware trojans in wireless cryptographic ics: silicon demonstration & detection method evaluation," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2013, pp. 399–404.

[29] K. Xiao et al., "A clock sweeping technique for detecting hw trojans impacting circuits delay," *IEEE Design Test*, vol. 30, pp. 26–34, 2013.

[30] Y. et al., "Hw trojan detection using path delay fingerprint," in *IEEE Int. Workshop on HW-Oriented Security & Trust*, 2008, pp. 51–57.

[31] J. Li et al., "At-speed delay characterization for ic authentication and trojan horse detection," in *Int. Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 8–14.

[32] Y. Jin et al., "Hardware trojan detection using path delay fingerprint," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE Int. Workshop on.* IEEE, 2008, pp. 51–57.

[33] X. Cui et al., "Hardware trojan detection using the order of path delay," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 3, p. 33, 2018.

[34] I. Exurville et al., "Resilient hardware trojans detection based on path delay measurements," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST).* IEEE, 2015, pp. 151–156.

[35] D. Ismari et al., "On detecting delay anomalies introduced by hardware trojans," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* IEEE, 2016, pp. 1–7.

[36] M. Heidari and S. Rafatirad, "Using transfer learning approach to implement convolutional neural network model to recommend airline tickets by using online reviews," in *2020 15th International Workshop on Semantic and Social Media Adaptation and Personalization (SMA.* IEEE, 2020, pp. 1–6.

[37] H. Li, Q. Liu, and J. Zhang, "A survey of hardware trojan threat and defense," *Integration*, vol. 55, pp. 426–437, 2016.

[38] N. Jacob, D. Merli, J. Heyszl, and G. Sigl, "Hardware trojans: current challenges and approaches," *IET Computers & Digital Techniques*, vol. 8, no. 6, pp. 264–273, 2014.

[39] K. Xiao, 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)*, vol. 22, no. 1, pp. 1–23, 2016.

[40] M. Tehranipoor et al., "A survey of hw trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, Jan 2010.

[41] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[42] S. R. Hasan, C. A. Kamhoua, K. A. Kwiat, and L. Njilla, "Translating circuit behavior manifestations of hardware trojans using model checkers into run-time trojan detection monitors," in *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*. IEEE, 2016, pp. 1–6.

[43] F. Wolff et al., "Towards trojan-free trusted ics: Problem analysis and detection scheme," in *Design, Automation and Test in Europe*, 2008, pp. 1362–1365.

[44] S. Wei et al., "Scalable hardware Trojan diagnosis," *IEEE Trans. on VLSI Systems*, vol. 20, no. 6, pp. 1049–1057, 2012.

[45] O. Söll, T. Korak, M. Muehlberghuber, and M. Hutter, "Em-based detection of hardware trojans on fpgas," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2014, pp. 84–87.

[46] H. Salmani and M. Tehranipoor, "Layout-aware switching activity localization to enhance hardware trojan detection," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 76–87, 2011.

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

[48] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey, "Hardware trojan horse detection using gate-level characterization," in *2009 46th ACM/IEEE Design Automation Conference*. IEEE, 2009, pp. 688–693.

[49] D. Du, S. Narasimhan, R. S. Chakraborty, and S. Bhunia, "Self-referencing: A scalable side-channel approach for hardware trojan detection," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 173–187.

[50] P. Vuillod, L. Benini, A. Bogliolo, and G. De Micheli, "Clock skew optimization for peak current reduction," in *Proceedings of the 1996 Int. Symp. on Low power electronics and design*. IEEE Press, 1996, pp. 265–270.

[51] S. Pant and D. Blaauw, "Static timing analysis considering power supply variations," in *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.*, Nov 2005, pp. 365–371.

[52] R. Ahmadi and F. N. Najm, "Timing analysis in presence of power supply and ground voltage variations," in *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*, Nov 2003, pp. 176–183.

[53] K. Arabi et al., "Power supply noise in socs: Metrics, management, and measurement," *IEEE Design & Test of Comp.*, vol. 24, no. 3, 2007.

[54] A. Vakil, H. Homayoun, and A. Sasan, "IR-ATA: IR annotated timing analysis, a flow for closing the loop between PDN design, IR analysis & timing closure," in *Asia and South Pacific Design Automation Conf.*, 2019, pp. 152–159.

[55] V. Wang et al., "A design model for random process variability," in *Int. Symp. on Quality Electronic Design*, 2008, pp. 734–737.

[56] A. Vakil, F. Behnia, A. Mirzaeian, H. Homayoun, N. Karimi, and A. Sasan, "Lasca: Learning assisted side channel delay analysis for hardware trojan detection," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 40–45.

[57] F. Chollet *et al.*, "Keras," https://keras.io, 2019.

[58] A. Mirzaeian, H. Homayoun, and A. Sasan, "Tcd-npe: A re-configurable and efficient neural processing engine, powered by novel temporal-carry-deferring macs," in

2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2019, pp. 1–8.

[59] A. Mirzaeian et al., "Nesta: Hamming weight compression-based neural proc. engine," *25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[60] IWLS-org. (2005) Iwls 2005 benchmarks. Accessed July 10, 2019. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[61] T. Sakurai and A. R. Newton, "Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas," *IEEE Journal of solid-state circ.*, vol. 25, pp. 584–594, 1990.

[62] Synopsys. Composite current source delay modeling. Accessed July 10, 2019. [Online]. Available: https://news.synopsys.com/index.php?s=20295&item=122723

[63] ANSYS-Apache. (2020) Redhawk. Accessed Jan 10, 2020. [Online]. Available: https://www.ansys.com/products/semiconductors/ansys-redhawk

[64] E. Bogatin, *Signal and Power Integrity - Simplified*, 3rd ed. Prentice Hall, 2018.

[65] Synopsys. (2019) Primetime. Accessed July 10, 2019. [Online]. Available: http://synopsys.com/implementation-and-signoff/signoff/primetime.html

[66] N. Perkins et al., "The inconsistency of optimal cutpoints obtained using two criteria based on the receiver operating characteristic curve," *American journal of epidemiology*, vol. 163, no. 7, pp. 670–675, 2006.

[67] S. Chen et al., "Fully on-chip temperature, process, and voltage sensors," in *Proceedings of 2010 IEEE Int. Symp. on Circuits and Systems*, May 2010, pp. 897–900.

[68] M. Sasaki et al., "A temperature sensor with an inaccuracy of $-1/+0.8$ ° c using 90-nm 1-v cmos for online thermal monitoring of vlsi circuits," *IEEE Trans. on Semiconductor Manufacturing*, vol. 21, no. 2, pp. 201–208, May 2008.

[69] Synopsys. (2019) Synopsys toolset. Accessed July 10, 2019. [Online]. Available: https://news.synopsys.com/