# A Sensor-Assisted Self-Authentication Framework for Hardware Trojan Detection

Min Li, Azadeh Davoodi, and Mohammad Tehranipoor*
Department of Electrical and Computer Engineering
University of Wisconsin at Madison, WI 53706 USA
$*$University of Connecticut, CT 06269 USA
Email: {adavoodi}@wisc.edu

***Abstract*—This work offers a framework which does not rely on a Golden IC (GIC) during hardware Trojan (HT) detection. GIC is a Trojan-free IC which is required, in all existing HT frameworks, as a reference point to verify the responses obtained from an IC under authentication. However, identifying a GIC is not a trivial task. A GIC may not even exist, since all the fabricated ICs may be HT-infected. We propose a framework which is based on adding a set of *detection sensors* to a design which are integrated in the free spaces on the layout and fabricated on the same die. After fabrication, a self-authentication procedure is proposed in order to determine if a Trojan is inserted in a set of arbitrarily-selected paths in the design. The detection process uses on-chip measurements on the sensors and the design paths in order to evaluate the correlation between a set of actual and predicted delay ranges. Error in the on-chip measurement infrastructure is considered. If our framework determines that a Trojan is (or is not) inserted on a considered path, then it is accurate. In our computational experiments, conducted for challenging cases of small Trojan circuits in the presence of die-to-die and within-die process variations, we report a high detection rate to show its effectiveness in realizing a self-authentication process which is independent of a GIC.**

## I. Introduction

Hardware Trojan (HT) detection is an increasingly important topic within IC security. It can influence the effectiveness of other security techniques such as IC metering and physically uncolonable functions [1]. Detecting a HT in a fabricated IC is particularly challenging due to a combination of the following factors arising from system complexity and nano-scale non-idealities [1]: (1) lack of observability and controllability in a fabricated chip; (2) complexity due to existence of billions of nano-scale components, and due to the large number of soft, firm, and hard IP cores that may have been integrated in the system; (3) high cost and challenges associated with physical inspection of nanometer feature sizes for reverse engineering which may yet be intrusive and only applicable to a small portion of the chip population that are infected by a Trojan; (4) difficulty to activate a HT since it may only be triggered by rare events; (5) increasing fabrication and environmental variations in nanometer technologies which cause deviation in the expected characteristics of an IC.

One focus of existing research for HT detection is on generating rare events or test patterns which result in increased activity in Trojan-infected ICs [2], [3], [4]. The other set of HT detection techniques are based on measuring the transient signals of power delivery network through the power ports [2], [5], or timing characteristics using on-chip structures or automatic test equipment (ATE) [6], [7]. The works [8], [9] focus on HT detection in the presence of process variability.

A fundamental limitation of the above existing HT detection techniques is relying on a Golden IC (GIC) during the IC authentication. For example, a transient timing or power pattern of an IC should be compared against the one obtained from the GIC since it is determined to be Trojan-free. However, the existence and identification of a GIC can never be guaranteed: First, if the HT is inserted in the GDSII file, or if the foundry alters the mask to insert a HT, then all the ICs will be infected. Second, if an IC passes a testing procedure which is thorough and more aggressive than traditional testing, it still cannot be guaranteed to be a GIC because the HT may be triggered by a rare uncovered event during test. Recently, a self-referring approach was proposed in [10] which is also GIC-free. It is based on comparing a "current signature" of the chip in two different time windows. The idea is to create a variation in these two signatures if a Trojan is present. However, this approach is only applicable to large sequential circuit Trojans.

In this work, we propose a new framework for HT detection which does not require a GIC. It can also handle combinational Trojan circuits of small size. The framework relies on developing on-chip *detection sensors* prior to fabrication and integrating them with the layout. The sensors are design-dependent and are found by an optimization procedure which decomposes the netlist into a set of "similar" sequences of logic gates which are frequently-instantiated. After fabrication, a set of design paths are arbitrarily selected and their delays are measured, for example using an ATE. The on-chip delays of the detection sensors are also measured. An analysis framework is then proposed to detect the presence of a HT. It is based on computing a sensor-assisted delay range for each path, and comparing it against the actual (measured) path delay, while accounting for errors associated with the on-chip delay measurement infrastructure. The self-authentication process is entirely independent of using a Golden IC.

The work of Azadeh Davoodi was supported in part by the National Science Foundation grant CCF-1053496. The work of Mohammad Tehranipoor was supported in part by the National Science Foundation grant CNS-0844995.

978-3-9810801-8-6/DATE12/©2012 EDAA

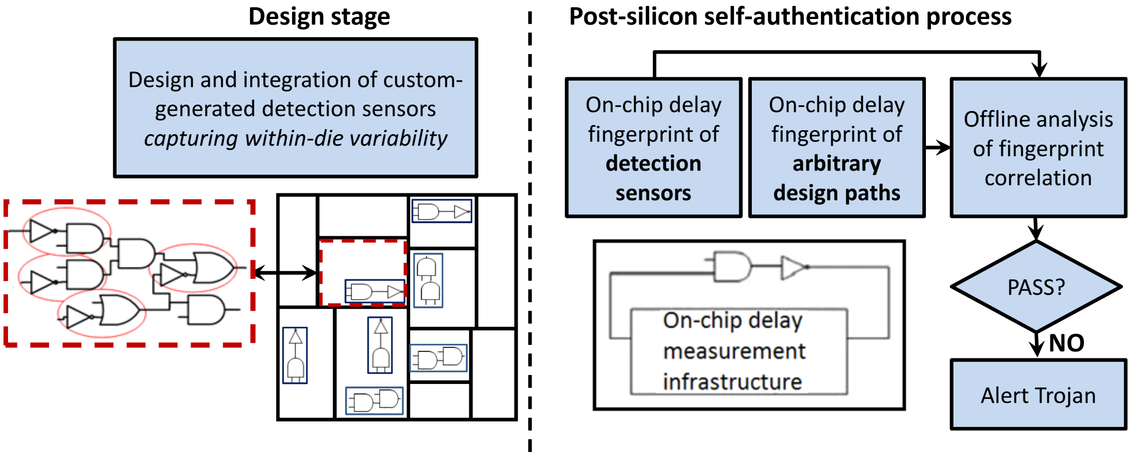
Design stage
Design and integration of custom-generated detection sensors *capturing within-die variability*
Post-silicon self-authentication process
On-chip delay fingerprint of **detection sensors**
On-chip delay fingerprint of **arbitrary design paths**
Offline analysis of fingerprint correlation
On-chip delay measurement infrastructure
PASS?
NO
Alert Trojan

Fig. 1: Overview of our self-authentication Trojan detection framework

The proposed framework is one of the first towards eliminating the GIC; only if the framework fails to determine the existence of the HT, conventional detection techniques should be used which require to identify a GIC. By reporting a high detection ratio in our experiments, we demonstrate the effectiveness of our self-authentication framework. This is for many challenging scenarios of randomly inserting HTs in various regions on the layout, for many chips illustrating die-to-die and within-die process variations in 90nm technology.

In the remainder of this work, we give an overview of the framework and sensor generation procedure in Sections II and III, respectively. Analysis techniques using the on-chip measurements are discussed Section IV. Simulation results are presented in Section V followed by conclusions.

## II. Overview of the Framework

Figure 1 gives an overview of the proposed framework. There are two stages in the framework: 1) identification of detection sensors and their layout integration at the design stage; 2) self-authentication of a chip using a sensor-assisted analysis which incorporates on-chip delay measurements.

First, at the design stage, a set of detection sensors are identified and integrated with the layout. These sensors are custom-generated using an optimization procedure which decomposes the timing graph representing the design's netlist into a set of frequently-instantiated "delay features". Each delay feature is essentially a variation-aware expression with known sensitivities to unknown parameters such as process variations at the die-to-die and within-die levels. Such a variation-aware representation is commonly used in parametric statistical static timing analysis [11]. Each instantiation of a delay feature corresponds to an identified sequence of gates and interconnects in the netlist which have similar sensitivities to parameter variations. Therefore, the sensor generation procedure essentially decomposes the netlist into a set of "similar" sequences in which two sequences are similar if the *changes* in their delays are very close (i.e., less than a specified and small error tolerance). An graphical example of similar sequences is illustrated on the left-hand-side of Figure 1.

Each sensor is then physically implemented and integrated with the layout such that it has the same delay feature as the sequences that it represents. For example it could have an identical layout as one of the sequences that it represents and be placed in the same neighborhood of that sequence to be subject to the same degree of variations. The optimization procedure identifies the sensors subject to an allowable area overhead and the available whitespace on the layout. This work uses a variation of an existing framework for sensor generation which will be briefly explained in Section III.

At the post-silicon stage, a sensor-assisted self-authentication process is applied for each chip. Two "delay fingerprints" are generated. One corresponds to the on-chip delays of the integrated sensors. The other corresponds to the on-chip delays of a set of arbitrary-selected design paths. The delay fingerprint of the sensors are used to *predict* a delay range for each considered path. An *actual* on-chip delay range is also obtained for each path, using an on-chip delay measurement mechanism. For each considered path, a correlation analysis is then conducted between its predicted delay range and its actual delay range. If a hardware Trojan is inserted (in either the sensors or the design path), then its presence can get detected by observing a poor correlation between these two delay ranges. This post-silicon self-authentication process is shown on the right-hand-side of Figure 1 and will be explained in detail in Section IV.

The set of authenticated design paths can be pre-specified at the design stage or can be randomly selected after fabrication. Our framework has the flexibility to consider both cases. In this work, we consider the case when the layout is decomposed into many regions and the design paths are randomly selected from each region to capture within-die delay variations. More details are explained in Section V.

There are many options for on-chip delay measurement of either design paths or detection sensors. For example, the work [12] proposes a circuit for measuring the delays of a set of designated paths on the layout. To measure the on-chip delay of one designated path, the path is excited and the measurement circuitry is calibrated to transform the path into a ring oscillator. A counter then measures the number of oscillations within a time window. In [13], measurement of the delay of a path is with the aid of a debug tester and tunable buffers. The tester is used to generate failures by gradually varying the clock period, each time by decreasing with a small time-step. For each new clock period, a new set of "failing" paths in the design are identified with the aid of tunable buffers. The on-chip delays of these paths are determined from the considered clock period.

Arbitrary path delays can also be measured using an automatic test equipment (ATE), by generating test patterns which excite the desired path(s), assuming they are testable, and then sweeping the clock period in a similar way until failure occurs.

In all the above cases, the on-chip path delay measurement is subject to a measurement error. For example using debug tester or ATE, the measurement error is the time-step by which the clock period is decreased. Our work assumes an infrastructure for on-chip path delay measurement is available. If the paths are pre-specified, then an infrastructure such as [12], or an ATE-based measurement procedure can be used. Otherwise, the technique in [13] can be used to determine the delays of arbitrarily-selected design paths by sweeping the applied clock period. The above procedures also have different associated error in path delay measurement.

*Our framework is independent of the type of measurement infrastructure; for a choice of the measurement infrastructure, an associated error is accounted for in our framework to capture the on-chip measurement inaccuracy. This error will be incorporated in the predicted and actual delay ranges of each design path. Moreover, the possible area overhead of the measurement infrastructure can also be accounted for in our sensor generation procedure.*

## III. Sensor Generation Procedure

Our sensor generation procedure is a variation of an existing optimization framework. In [14], custom test structures are proposed to isolate the paths that fail their delay constraints during post-silicon validation. Here we use this framework to identify detection sensors.

For a given netlist, first a timing graph is extracted which is a directed acyclic graph (DAG) denoted by $G = (\mathcal{V}, \mathcal{E})$. Each node corresponds to the output of a gate. An edge between two nodes represents the delay of *the path* between two gates so it includes factors such as loading capacitance and interconnect delay on this path. We refer to a sequence to be a (directed) path between two arbitrary nodes in this DAG. Therefore a sequence corresponds to a set of consecutively connected gates and interconnects.

We denote $\mathbf{X}$ to be a column vector representing the parameter variations. Each entry in $\mathbf{X}$ lies in the category of die-to-die, within-die and random variations in physical parameters. All the entries in $\mathbf{X}$ are assumed to be independent from each other. This modeling procedure is identical to the previous works in statistical static timing analysis which are summarized in [11]. Similar to [11], we describe the variation-aware delay of edge $e_i$ using the following linear expression:

$$D_{e_i} = \mu_{e_i} + \mathbf{a}_{e_i}^T \mathbf{X} \tag{1}$$

where $\mu_{e_i}$ is the nominal delay of $e_i$ and $\mathbf{a}_{e_i}$ is the sensitivity vector corresponding to $\mathbf{X}$ which is accurately known after design-time characterization. Note, the above expression already accounts for interconnect delay variations.

A sequence $s_i$ is a set of consecutive edges in $G$. The variation-aware delay expression of $s_i$ is found by adding the corresponding delay expressions of the edges that are included in it. It is given by $D_{s_i} = \sum_{\forall j | e_j \in s_i} D_{e_j}$ which can be further expressed in a form similar to the above equation as follows: $D_{s_j} = \mu_{s_j} + \mathbf{a}_{s_j}^T \mathbf{X}$.

The optimization procedure *forms* a set of non-overlapping sequences such that "similar" sequences are assigned to one group and represented by one sensor. Two sequences $s_i$ and $s_j$ are similar if their sensitivities to parameter variations $\mathbf{X}$ is less than a small error tolerance denoted by $\epsilon$. This condition is given by the following inequality:

$$\left\| \mathbf{a}_{s_i}^T - \mathbf{a}_{s_j}^T \right\|_1 \leq \epsilon \tag{2}$$

Each group of similar sequences will be represented by a sensor. The physical implementation of the sensor is also a sequence with a (similar) variation-aware delay expression that satisfies the above condition. In this work, the sensor is implemented to be identical to the smallest-area sequence that it represents in a neighborhood close to it in order to satisfy the above similarity constraint. If whitespace is available, then the addition of the sensor does not result in any additional area overhead. Otherwise, the sequences are formed such that the sensor area overheads do not exceed a specified threshold.

The objective of the optimization is to form groups of similar sequences in order to maximize the coverage of edges $e_i \in \mathcal{E}$ subject to the sensor area constraint which was described above. Maximizing the edge coverage increases the likelihood of Trojan detection while not making any assumptions about the potential locations in the netlist that the Trojan may be inserted. In contrast, the work [14] further associates a statistical-criticality weight with each edge in the graph based on the knowledge of potentially-critical paths in order to better isolate the failing paths after fabrication.

The work [14] presents an exact Integer Linear Programming (ILP) formulation of the above optimization problem. It also presents a set of decomposition techniques to solve the formulation within a reasonable runtime using the solver CPLEX 12.0. For details about extracting the ILP formulation and solving it, please refer to [14].

## IV. Post-Silicon Self-Authentication Process

After the sensors are added to the die, the Trojan detection procedure is based on self-authentication at the post-fabrication stage. For each chip, we consider an arbitrary set of paths and explore the existence of the Trojan. In this section, we explain the procedure for three cases, when the Trojan is inserted in the design paths, in the sensors, and in both.

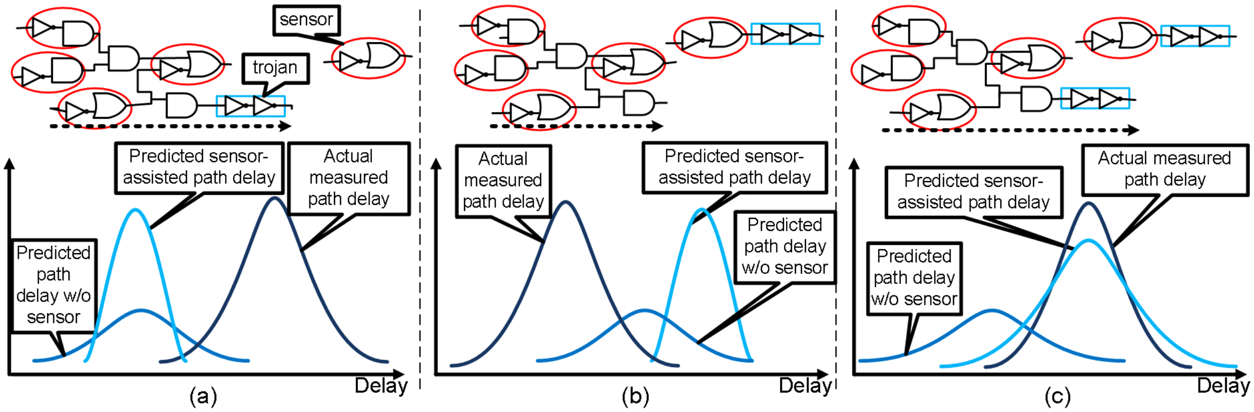
sensor
trojan
Predicted sensor-assisted path delay
Actual measured path delay
Predicted path delay w/o sensor
Delay
(a)
Actual measured path delay
Predicted sensor-assisted path delay
Predicted path delay w/o sensor
Delay
(b)
Predicted sensor-assisted path delay
Actual measured path delay
Predicted path delay w/o sensor
Delay
(c)

Fig. 2: Different scenarios of Trojan insertion: (a) Trojan-infected path; (b) Trojan-infected sensor; (c) Both (a) and (b).

### A. Trojan Inserted in the Design Paths

We assume the Trojan is inserted in an arbitrary location in the design's netlist. As a result, one or more paths are infected by the Trojan. So in our analysis, we arbitrarily select a set of design paths and evaluate if each one is infected by a Trojan, with the assistance of the detection sensors. This case is shown in Figure 2(a). The affected path is shown with a bold arrow and the Trojan is shown as an extra chain of inverters. The sensor is shown on the right corner and its matched sequences in the netlist are circled. Note, the sequences may not be identical and their similarity is defined according to Eq. 2.

To detect the Trojan, we first compute an actual and a predicted delay range for each path and analyze their correlations.

First, the *actual delay range* of path $p$ is computed by measuring its delay using an on-chip measurement infrastructure. The range is due to a measurement error associated with the used infrastructure.

Next, a sensor-assisted *predicted delay range* is found for path $p$. We first compute an actual delay corresponding to one or more sequences on $p$ which are matched by one or more sensors. For each sequence, its delay is determined using on-chip delay measurement on its corresponding sensor. Recall, the sequence and its sensor are formed to have the same on-chip change in their delays. So the change in the delay of the sensor can be computed (by comparing against its pre-silicon delay estimate) in order to find the on-chip delay of the sequence which it represents. We denote the aggregate delay of the matched sequences with $\mu_{match} \pm \gamma$, where $\gamma$ is the error associated with the on-chip measurement infrastructure.

For path $p$, we then consider its remaining portion which is not matched by any sensors. It is denoted by $p_{rmn}$ and identified by a set of edges on $p$ which are not formed into sequences during the sensor generation process. We denote the delay corresponding to $p_{rmn}$ by $D_{rmn}$ which has a variation-aware delay expression obtained by adding the expressions of the edges that are included in it, similar to Eq. 1. We express $D_{rmn} = \sum_{\forall i | e_i \in P_{rmn}} (\mu_{e_i} + a_{e_i}^T \mathbf{X}) = \mu_{rmn} + \mathbf{a}_{rmn}^T \mathbf{X}$. We then compute worst-case and best-case values of $D_{rmn}$, denoted by $D_{rmn}^{(WC)}$ and $D_{rmn}^{(BC)}$. This is by assuming the varying parameters $x \in \mathbf{X}$ are simultaneously in their extreme values in the $D_{rmn}$ expression. The predicted delay range of path $p$ is then given by $\mu_{match} + \mu_{rmn} \pm (\gamma + D_{rmn}^{(WC)} - D_{rmn}^{(BC)})$.

Before discussing the range-based analysis for Trojan detection, we also introduce computing another delay range representing the case when the sensors are not used, to clarify the discussions. In this case the delay of the entire path denoted by $D_p$ is unknown. Therefore $D_p = D_{rmn}$. The predicted path delay range is given by $\mu_p \pm (D_p^{(WC)} - D_p^{(BC)})$. This delay range is larger than the sensor-assisted one because the uncertain part of the path (the major contributor to its range) decreases with matching of its sequences with the sensors.

Figure 2(a) shows the Trojan detection analysis in this case. On the left-hand-side two predicted delay ranges are drawn, corresponding to with and without sensor matching. The actual delay range always falls on the right-hand-side of the predicted delay ranges. This is due to the insertion of the Trojan on the path which shifts its measured delay range to the higher values. We make the following observations:

- If a predicted delay range has no overlap with the actual delay range, then we conclude that a Trojan is inserted. The observation that the actual delay range is higher also prompts that the Trojan is inserted on the path.
- Recall the sensor-assisted prediction results in a smaller delay interval compared to the one without sensor matching. This reduction in interval increases the likelihood that the predicted and the actual delay ranges do not overlap if a Trojan is present.
- The decrease in the Trojan delay results in the predicted and actual delay ranges to be closer to each other and makes the detection more challenging.

### B. Trojan Inserted in the Detection Sensors

Figure 2(b) shows this case. The insertion of the Trojan (shown by an inverter chain) increases the on-chip measured sensor delay. Since the Trojan does not impact the path, the actual delay range of the path and the predicted one *without* sensors will both be accurate and computed similar to the previous case. These two ranges likely overlap with each other. So by comparing them the existence of the Trojan may not be identified. However, the sensor-assisted delay range will be inaccurate, and shifted to the right-hand-side of the actual delay range. Therefore the two intervals are more likely not to overlap, increasing the chances of Trojan detection. Since the predicted interval is on the right-hand-side of the actual, we also conclude that the Trojan is inserted in the sensor.

TABLE I: Comparison of detection rate ($DR$) of different Trojan insertion cases

| Sensor and path generation information | | | | | | $DR$ (Trojan in design) | | $DR$ (Trojan in sensor) | |
|---|---|---|---|---|---|---|---|---|---|
| Bench | $\lvert P\rvert$ | $\lvert R\rvert$ ($\%A_P$) | $\%A_{sensor}$ | T(min) | $MR$ | sensor-assisted | w/o sensor | sensor-assisted | w/o sensor |
| S1423 | 92 | 59 (64%) | 13 | 92 | 1 | 0.68 | 0.58 | 0.67 | 0.03 |
| S1488 | 16 | 33 (21%) | 12 | 20 | 1 | 0.67 | 0.60 | 0.69 | 0.04 |
| S1494 | 16 | 34 (15%) | 12 | 5 | 1 | 0.50 | 0.48 | 0.60 | 0.04 |
| S5378 | 201 | 123 (58%) | 12 | 31 | 0.84 | 0.66 | 0.62 | 0.58 | 0.03 |
| S9234 | 169 | 113 (62%) | 10 | 24 | 1 | 0.71 | 0.69 | 0.75 | 0.03 |
| S13207 | 331 | 94 (69%) | 12 | 35 | 1 | 0.73 | 0.67 | 0.77 | 0.01 |
| S15850 | 136 | 56 (86%) | 15 | 29 | 0.97 | 0.85 | 0.75 | 0.70 | 0.01 |
| S35932 | 1765 | 1014 (78%) | 13 | 170 | 0.96 | 0.73 | 0.70 | 0.93 | 0.01 |
| S38417 | 1587 | 981 (66%) | 12 | 250 | 0.99 | 0.77 | 0.72 | 0.78 | 0.02 |
| S38584 | 1112 | 548 (68%) | 14 | 49 | 1 | 0.80 | 0.64 | 0.97 | 0.00 |

### C. Trojan Inserted in the Paths and Sensors

As shown in Figure 2(c), the sensor-assisted predicted range and the actual one are more likely to overlap with each other in this case, because the insertion of the Trojan shifts both of the intervals to the right-hand-side. The predicted delay without sensors may still overlap due to its larger interval.

## V. Simulation Results

### A. Simulation Configuration

We synthesized ISCAS89 benchmark circuits using Synopsys Design Compiler and a 90nm TSMC technology library. We assume variations in the effective channel length and zero-bias threshold voltage parameters of each gate, with a Gaussian distribution and standard deviations of 10% of their mean values. To capture spatial correlations, we use the multi-level hierarchical model of [11] which divides the chip into a set of rectangular regions. The gates in the same region or in neighboring regions will share all or some of the random variables in their delay expressions and thus will be correlated to each other. This results in 42 variables for smaller benchmarks (S1488 to S9234) for a 3-level hierarchical model, and 682 variables for the larger benchmarks for a 5-level hierarchical model which is consistent with [11].

For each benchmark, we generate sensors as described in Section III. The parameter $\epsilon$ in defining the similarity constraint between two sequences (Eq. 2) was set to be 0.05 of the average gate delay in the library. The allowable area overhead of the sensors was set to 15% of the areas of the logic cells. We also randomly select a set of design paths in our setup which will be used for Trojan insertion in our experiments. To select the paths, for each primary input (PI) or flipflop (FF) in the circuit, we randomly select one path that initiates from it. This is done by randomly selecting fanout gates starting from the PI or FF node, until reaching another FF or a primary output (PO). Therefore the number of paths is proportional with the number of FF and PI nodes in the netlist. This strategy ensures a fair and unbiased coverage of the netlist, as well as the layout. Furthermore, the selected paths may be timing critical or non-critical. These attributes make the set of selected paths a challenging one for inserting Trojan and evaluating our detection mechanism. In Table I, columns 2 to 6 report the path and sensor generation information. The number of selected paths is reported in column 2.

To get an idea about the spatial coverage of the paths, we divide the layout into rectangular regions. If the initiating PI or FF of the path falls within the region, then the area of the region is represented by *at least* one path. We report the number of regions ($|R|$) and the areas of the represented regions as a percentage of the overall layout area ($\%A_P$) in column 3. The layout area coverage ranges from 15% to 86%.

In our experiments, the sensors all fit within the existing whitespaces of the layout. This is because in the physical implementation, the benchmarks had available whitespace which ranged between 10 to 20% of the layout. However, the details of the physical implementation is not discussed in this work due to lack of space. Therefore, for each benchmark, we also report the area usage of the sensors compared to the core area consumed by the logic cells. This overhead is reported as a percentage in column 4 ($\%A_{sensor}$). Please note, in practice there may be an overhead associated with on-chip measurement circuitry for the sensors, which we plan to present as part of our future work regarding the details of the physical implementation of the framework. The runtime of the sensor generation procedure is given in column 5. For each designated path, we also report the fraction of the paths which are matched with at least one sensor as a match ratio ($MR$) in column 6. The $MR$ is high, ranging from 0.84 to 1.

### B. Trojan Insertion in the Design Netlist

In this experiment, we insert a Trojan in each designated path and use the analysis technique described in Section IV-A to evaluate the rate of detecting the presence of the Trojan. First, for each path, we insert the Trojan on a random location on the path. Then, we model the Trojan as a chain of (even-sized) inverters. We experiment with different lengths of the inverter chain such that the Trojan delay varies from 3% to 10% of the *circuit delay* with a uniform step of 0.9%, to obtain an overall number of 30 different Trojan delays on that path. This scale of delay range is categorized as a *small* Trojan circuit. In addition, we consider the same path on many different chips that are subject to die-to-die and within-die process variations. To model the post-silicon path delays on different chips, we use Monte Carlo (MC) simulation with 10K samples for the varying parameters. Each sample, represents one assignment to the random variables in vector $\mathbf{X}$ in Eq. 1. Overall, for a designated path, we consider 300K cases of Trojan insertion and process-induced delay variations.

For each variation sample, we compute the actual delay of the path ($\mu_{actual}$) by plugging in the MC sample value into its variation-aware delay expression (after Trojan insertion). This expression is obtained by adding the variation-aware delay expressions of the individual edges on the path given by Eq. 1. The contribution of the Trojan delay is a fixed value which only depends on the circuit delay and is computed for each chip. Next, we compute an *actual delay range* of the path. We assume a tester is used to measure the on-chip path delay which has a measurement error of 5% of the path delay. This assumption is consistent with [12]. Therefore, the range is given by 95% to 105% of $\mu_{actual}$.

We also compute a predicted delay range for the path as follows: For the sequences on the path which are matched by a sensor, we compute their on-chip delays by plugging in the corresponding MC sample into the variation-aware delay expression of each sequence. For the remainder of the path, we compute a variation-aware delay expression by adding the expressions of the edges which are not matched with a sensor and denote its delay by $D_{rmn}$. Note $D_{rmn}$ is a random variable given in the form of Eq. 1. We then compute the best-case and worst-case delays of $D_{rmn}$ by setting each entry $x$ in vector $\mathbf{X}$ to $\mu_x \pm 3\sigma_x$ where $\mu_x$ and $\sigma_x$ are the mean and standard deviation of $x$, respectively and were explained in our simulation setup. The predicted delay range of the path is then given by $\mu_{rmn} + \mu_{match} \pm (D_{rmn}^{(WC)} - D_{rmn}^{(BC)})$.

We also make comparison with the case when sensors are not used. In this case, the delay of the unknown portion will be the same as of the entire path. A predicted delay range is computed similar to the previous case.

Using this predicted and actual delay ranges, we then use the analysis method in Section IV-A to detect if the path includes the Trojan. We report the fraction of cases that the Trojan was detected in our framework as a *detection rate*, $DR$, in Table I. This is for 300K cases for each of the designated paths. (See column 2 for the number of designated paths.) Columns 7 and 8 report the $DR$s for the sensor-assisted case, and prediction without sensor, respectively. The sensor-assisted case is higher than without sensor, in all the benchmarks consistently. The rate of improvement varies among the benchmarks and is highest in `s1488` and `s38584` which have a balanced structure. In all the cases, we verified that our framework either detects the Trojan or states that a conclusion cannot be made. So it never incorrectly states that path does not include a Trojan.

Figure 3 shows the detection rate in `S13207` in terms of the Trojan delay normalized to the circuit delay. With increase in the Trojan delay, the curves corresponding to sensor-assisted and without sensor cases converge to each other.

### C. Trojan Insertion in the Detection Sensor

Here we assume Trojan is only inserted in the sensors. We consider 10K different chips using MC simulation and for each chip compute the actual path delay (which is now Trojan-free), similar to the previous experiment. The predicted delay range of the path is computed as follows: First, a Trojan-infected on-chip sensor delay is computed. Specifically, we assume the Trojan is only inserted in one of the sensors (randomly selected) that cover each path. We consider many cases when the inserted Trojan delay varies from 3% to 10% of the circuit delay. The Trojan-infected on-chip sensor delay is computed from the corresponding MC sample, similar to the previous experiment. The predicted delay range of the path is also computed from the remaining portion of the path, just like the previous experiment. However, this time, the predicted delay range will be shifted to the higher delay values because of the incorrect calculation of the path delay using the Trojan-infected sensor. Similarly, we compute a detection rate for two cases of sensor-assisted and without sensor in columns 9 and 10 of Table I. Note, the detection rate without sensors is always 0 since sensors are not used in this case and the two ranges of actual and predicted delays overlap almost all the times. The sensor-assisted detection rates are typically higher compared to the previous experiment, since a Trojan-infected sensor may represent multiple sequences on a path.

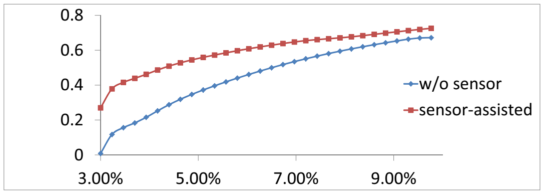

Fig. 3: Detection rate as a function of normalized Trojan delay

## VI. Conclusions

This work offers a hardware Trojan detection framework which is based on self-authentication of each chip. It does not require a golden IC. The process uses on-chip delay measurements on the design paths and custom sensors which are added on the same die and comparing (only) a set of actual and predicted delay *ranges* against each other. Error in the measurement infrastructure is also accounted for. Simulation results show a high detection rate for small Trojan circuits.

## References

[1] M. Tehranipoor, et. al, "A survey of hardware trojan taxonomy and detection," *IEEE DAT*, vol. 27, no. 1, 2010.

[2] M. Banga, et. al, "A novel sustained vector technique for the detection of hardware trojans," in *VLSI Design*, 2009.

[3] S. Jha, et. al, "Randomization based probabilistic approach to detect trojan circuits," in *HASE*, 2008.

[4] H. Salmani, et. al, "New design strategy for improving hardware trojan detection and reducing trojan activation time," in *HOST*, 2009.

[5] R. M. Rad, et. al, "Sensitivity analysis to hardware trojans using power supply transient signals," in *HOST*, 2008.

[6] Y. Jin, et. al, "Hardware trojan detection using path delay fingerprint," in *HOST*, 2008.

[7] J. Li, et. al, "At-speed delay characterization for IC authentication and trojan horse detection," in *HOST*, 2008.

[8] Y. Alkabani, et. al, "Designer's hardware trojan horse," in *HOST*, 2008.

[9] M. Potkonjak, et. al, "Hardware trojan horse detection using gate-level characterization," in *DAC*, 2009.

[10] S. Narasimhan, et. al, "TeSR: A robust temporal self-referencing approach for hardware trojan detection," in *HOST*, 2011.

[11] D. Blaauw, et. al, "Statistical timing analysis: From basic principles to state of the art," *IEEE TCAD*, vol. 27, no. 4, 2008.

[12] X. Wang, et. al, "Path-RO: a novel on-chip critical path delay measurement under process variations," in *ICCAD*, 2008.

[13] K. Killpack, et. al, "Silicon speedpath measurement and feedback into EDA flows," in *DAC*, 2007.

[14] M. Li, et. al, "Custom test structures for post-silicon diagnosis of failing paths," *ECE Technical Report, University of Wisconsin - Madison*, 2011.