2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

An Improved Automatic Hardware Trojan Generation Platform

Shichao Yu
CSIT, ECIT
Queen's University Belfast
Belfast, United Kingdom
syu08@qub.ac.uk

Weiqiang Liu
College of EIE
Nanjing Uni. Aero. & Astro.
Nanjing, China
liuweiqiang@nuaa.edu.cn

Maire O'Neill
CSIT, ECIT
Queen's University Belfast
Belfast, United Kingdom
m.oneill@ecit.qub.ac.uk

***Abstract*—Over the past 10 years, various Hardware Trojan (HT) detection techniques have been proposed by the research community. However, the development of HT benchmark suites for testing and evaluating HT detection techniques lags behind. The number of HT-infected circuits available in current public HT benchmarks is somewhat limited and the circuits lack diversity in structure. Therefore, this paper proposes a new method to generate HTs using a highly configurable generation platform based on transition probability. The generation platform is highly configurable in terms of the HT trigger condition, trigger type, payload type and in the number and variety of HT-infected circuits that can be generated. In this research the transition probability of netlists is employed to identify rarely activated internal nodes to target for HT insertion rather than functional simulation as utilised in previous research. The authors believe transition probability provides a more realistic reflection of the netlist activity for use in determining the appropriate position for HT insertion. Finally, the generated HT-infected circuits are tested by a machine learning (ML)-based HT detection technique, which is known as Controllability and Observability for HT Detection (COTD). The resulting false positive and false negative rates illustrate the feasibility of the benchmark suite.**

***Index Terms*—Hardware Trojans, benchmarks, automatic generation, highly configurable, transition probability**

I. Introduction

As a result of the globalization of the semiconductor supply chain, the design and fabrication of Integrated Circuits (ICs) are now distributed worldwide. It benefits IC companies but also raises serious concerns about IC trustworthiness triggered by the use of third-party vendors. For instance, through the use of third party Intellectual Property (IP), offshore foundries and third party test facilities, many different untrusted entities may be involved in the design and assembly phases. Therefore, it is becoming very difficult to ensure the integrity and authenticity of devices. A Hardware Trojan (HT) can be inserted into IC products at any untrusted phase of the IC production chain by third-party vendors or adversaries with an ulterior motive.

To defeat HTs and prevent HT-infected chips from being supplied to the market, researchers have proposed various HT detection techniques over the past decade [1], [2]. Benefiting from the development of training algorithms and computational power, very recent research shows a new trend in adopting machine learning (ML) approaches for HT detection [3], [4].

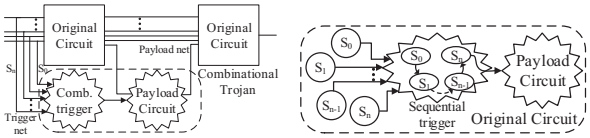
In order to evaluate the efficiency of HT detection techniques, HT-infected circuits created from reference circuits [5] are widely adopted as test samples in experiments by the research community. However, the use of "home-grown" HT-infected circuits in some research makes their detection results difficult to compare with others, which in turn makes the declared advantages and efficiency unconvincing [6]. Accordingly, a public HT benchmark suite was proposed in [6] to provide a fair evaluation platform for different kinds of HT detection techniques and it has been adopted in much of the recent research in this field.

However, current public HT benchmarks have limitations for the development of detection techniques. For example, the limitation in the amount of HT-infected circuits and the similarity of circuit structures results in overfitting of detection results, making it insufficient in supporting ML-based model training. Meanwhile, as these HT benchmarks are pre-designed and static after generation, they can not be updated in a timely manner when new HTs appear. Improving upon these drawbacks, an automatic HT insertion framework has recently been proposed in [7], which can insert Hardware Trojans into gate-level designs based on rarely activated internal nodes identified from functional simulation.

Functional simulation can provide an estimation of the switching activity of internal nodes. However, its accuracy is closely related to the number and quality of test patterns applied to the design inputs, which means it can take a long time to prepare and simulate all of the input test patterns to achieve high accuracy.

Motivated by the limitations mentioned above, in this paper, we propose an new method to generate HTs using a highly configurable generation platform based on transition probability [8] to identify the rarely activated internal nodes to target for HT insertion, rather than functional simulation as used in existing platforms. Transition probability can provide a good estimation of the switching activity for each net in the gate-level netlist without requiring a large number of test patterns. And the high flexibility of this platform is provided through configurations in terms of the HT trigger condition, trigger type, payload type and in the number of HTs generated. Furthermore, user-defined Trojans are permitted and the host circuits for HT insertion can be freely defined by the user.

978-1-7281-3391-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ISVLSI.2019.00062



(a) (b)

Fig. 1. (a) Combinational Trojan in circuitry; (b) Sequential Trojan

The main contributions of this paper are as follows:

1) A new method to generate HTs using a highly configurable generation platform based on transition probability is proposed, rather than functional simulation as used in existing platforms.
2) The structural design of a highly configurable automatic HT generation platform retains strong extensibility and supports user-defined HT types.
3) A ML-based detection technique, COTD [3], is applied to the HT-infected circuits generated by this platform. The detection results show this platform was able to generate failing test conditions for COTD with a high false negative rate (FNR) on nearly all generated HT-infected circuits.

The remainder of this paper is organized as follows: Section II discusses the related HT benchmarks work and transition probability. The architecture of the proposed platform and its work flow are presented in Section III. Section IV presents the experimental results and Section V provides some concluding remarks.

II. Related Work

A. Trojan Structure

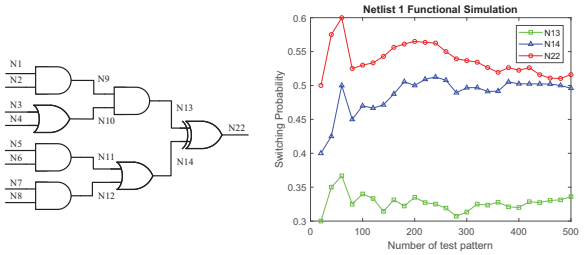
Hardware Trojans (HTs) implemented in ICs can alter a chips' structure and function in many ways. Typically, based on the type of physical characteristic, HTs can be classified into two categories: parametric and functional. This paper focuses on functional HTs which can be inserted at gate-level.

A functional HT comprises trigger circuitry and payload circuitry. The trigger part is a sensing circuit that monitors a set of signals in order to activate the payload after a specific event. Normally, it is kept in an inactive state. Once triggered, the payload will execute a malicious attack on the target circuit.

Furthermore, functional HTs can be further catergorised as either combinational or sequential depending on their trigger circuit. Combinational Trojans, as shown in Fig. 1(a), depend on the simultaneous occurrence of a set of rare signal conditions to trigger the payload. While sequential Trojans are more stealthy, and remain inactive before a series of signal states have occurred in sequence. As shown in Fig. 1(b), the Trojan will experience state S_0 to S_n before triggering.

B. Trojan Benchmarks

Trojan benchmarks provide a fair test environment to evaluate different HT detection techniques and allow for a meaningful comparison between different detection methods.



(a) (b)

Fig. 2. (a) Netlist 1 [8]; (b) Netlist 1 switching probabilities

1) Static Benchmarks: In 2013, a HT benchmark suite was developed in [9] and published on Trust-Hub [10]. This benchmark suite currently contains 94 HT-infected benchmark circuits, which is a great contribution to the standardization of HT testing. However, it still has some limitations. As a static set of benchmarks, the Trojan location and trigger conditions are static, which means detection techniques can be optimized to target HTs in this benchmark suite rather than generic HTs. Also, when new types of HT appear, the benchmarks can not be updated in a timely manner.

2) Dynamic Benchmarks: To address these limitations, a tool for generating HT circuits was recently proposed in [7]. The most significant feature of their automatic HT insertion framework is that it can dynamically insert Hardware Trojans into gate-level designs based on rarely activiated internal nodes identified from functional simulation.

Functional simulation can provide an estimation of the switching activity of internal nodes through statistical analysis. However, the accuracy of functional simulation is closely related to the number and quality of test patterns applied to the design inputs, which means simulation can take a lengthy time to run through a sufficient number of test patterns to achieve accurate results.

As shown in Fig. 2(a), netlist 1 consists of 7 gates in 3 levels. Under functional simulation, the switching probability of net i (SP_i) can be calculated as in (1), where $N_{sa,i}$ represents net i's switching activity, while N_{pse} is the total number of possible switching edges. For combinational circuits, the possible switching edge is the clock edge when the input pattern changed, while for sequential circuits, the switching edge can be either a clock's rising or falling edge based on the Flip-Flop's trigger condition.

$$SP_i = \frac{N_{sa,i}}{N_{pse}} \tag{1}$$

Fig. 2(b) shows the switching probabilities of net N13, N14 and N22 in Netlist 1 obtained from functional simulation. The simulation results show that when applying 500 random test patterns to the circuit sequentially the switching probabilities fluctuate during the simulation. When the number of test patterns is small, the fluctuation of the switching probabilities is large and a large number of test patterns is needed to achieve stable results.

TABLE I
SWITCHING PROBABILITY OF NETLIST 1

Net	Transition Probability	Functional Simulation (Fig. 2(b))			
		40	100	200	500
N12	0.1875	0.3250	0.3600	0.3200	0.3600
N13	0.1523	0.3500	0.3400	0.3350	0.3360
N14	0.2460	0.4250	0.4700	0.5000	0.4960
N22	0.2484	0.5750	0.5300	0.5650	0.5160
Time(ms)	67.2	478.9	485.7	495.2	519.6

C. Transition Probability

Transition probability is modeled using geometric distribution (GD) and is used to estimate the time required to generate a transition on a net [8]. This method can provide a good estimation of the switching activity for each net in a gate-level netlist without requiring lengthy simulation times. Suppose the probability of activating a "0" or "1" at net i is $P0$ or $P1$ respectively, the switching probability from "0" to "1" or "1" to "0" can be defined as in (2):

$$P_i = P0_i \cdot P1_i \tag{2}$$

According to the GD, suppose that X is defined as the number of clock cycles needed to produce a transition, then the probability function of X on net i at the nth clock cycle ($n = 1, 2, 3, 4...$) can be defined as in (3):

$$PT_i(X = n) = P_i \cdot (1 - P_i)^{(n-1)} \tag{3}$$

Then (4) is the expected value of X:

$$\begin{aligned} E_i(X) &= \sum_{k=1}^{\infty} P_i \cdot (1 - P_i)^{k-1} \cdot k \\ &= P_i^{-1} \end{aligned} \tag{4}$$

The expected value $E(X)$ indicates the average number of required clock cycles to generate a transition on net i, which means the smaller P_i is, the longer the time it takes to make a transition. P_i indicates the transition probability.

For example, assuming random inputs ($P1 = 0.5$, $P0 = 0.5$) are applied to the input pins of netlist 1 (Fig. 2(a)), the transition probability of each net can be calculated based on the logic function of each gate. For N12, according to the truth table of an AND gate, $P1_{N12} = P1_{N7} \cdot P1_{N8} = 1/4$, $P0_{N12} = 1 - P1_{N12}$, so the transition probability of net N12 is $P1_{N12} \cdot P0_{N12} = 0.1875$ as shown in Table I. This indicates that an average of 5.3 clock cycles is required to generate a transition at N12. The calculation steps for the whole netlist are introduced in Algorithm 1 in pseudocode.

When comparing the transition probabilities with the functional simulation results in Table I, the ordering of the results from the transition probability is the same as the functional simulation (i.e. $N22 > N14 > N12 > N13$) with 500 patterns. However, with 40 and 200 patterns, the switching probability of N12 is smaller than N13, which indicates that an inadequate number of test patterns leads to a untrustworthy results. Hence, the simulation results can not provide a realistic reflection of net switching activities when testing patterns are insufficient. Moreover, the software simulation needs to compile and run the test patterns within a testbench each time, so the functional simulation takes much more time than the calculation based on transition probability.

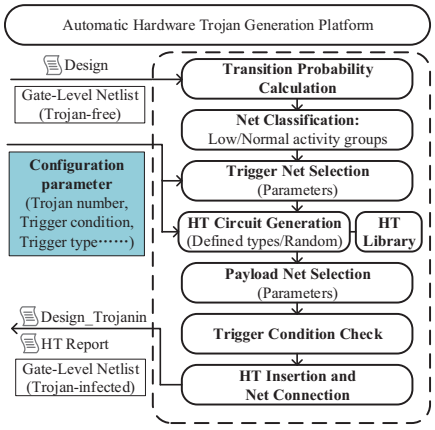


Fig. 3. Flowchart of the proposed automatic HT generation platform

With the above drawbacks of functional simulation, this proposed platform improves upon the previous research by utilizing transition probability [8] to more accurately estimate the switching activities for each net, which helps identify the rarely activated internal nets to target for HT insertion.

III. PROPOSED PLATFORM

The HT generation platform, as detailed in Fig. 3 consists of 8 components, 7 functional modules and 1 HT library. The input to this platform is a gate-level design and configuration parameters. The output is the HT-infected design and a feedback report on the HTs generated. The architecture is built in a modular manner to easily allow for extension and update.

A. Configurable Features

As the platform can generate various HTs and insert single or multiple HTs into a given netlist, configurable parameters are proposed within a configuration dictionary for use with the platform to control the types of HTs and their embedding methods. Overall, they can be divided into two. One set provides global control for the whole generation, while the other provides the local control for the generation of indidviual HTs. Table II provides an overview of all the configurable parameters and their description. Specifically, any parameter that can be set to "None" means it can be automatically configured and "Random" means the value can be randomly selected by the platform. In addition, the "user_def" parameter allows the platform to insert in a user-defined HT design.

B. Trigger Net Allocation

Generally speaking, HTs are stealthy in nature [6]. If designed to activate rarely this can help to evade detection. Therefore, it is preferable to connect the Trojan trigger to nets with low activity to create a rare trigger condition to reduce detectability.

Algorithm 1 shows how we allocate the low activity trigger nets and normal trigger nets for HT insertion. There are 3

TABLE II
HT CONFIGURATION DICTIONARY

Parameter Type		Parameters	Value	Description
Global Parameter		no_low_trig	Number	Total number of low activity triggers
		no_normal_trig	Number	Total number of normal activity triggers
		no_trojan	Number	Total number of generated HTs
Local Parameter	Trojan 1	no_trig	Number/None	Total number of triggers for Trojan 1
		no_low_trig	Number/None	Number of low activity triggers for Trojan 1
		no_normal_trig	Number/None	Number of normal activity triggers for Trojan 1
		trig_type	Type name/Random	Type of trigger for Trojan 1
		pay_type	Type name/Random	Type of the payload for Trojan 1
		no_pay	Number/None	Number of payload pins for Trojan 1
		user_def	True/False	Read in a user-defined Trojan 1
	Trojan ...	...	...	...

Algorithm 1 Trigger Net Allocation Algorithm

Input:
Gate-level design file, $design.v$;
Logic function library of gates, $lib.v$;
Transition probability threshold for classification, θ_{th};
no_low_trig and no_normal_trig in Table II;
Output:
List of low activity trigger nets and normal trigger nets, $list_{low\_trig}$ and $list_{normal\_trig}$;

{**Transition Probability Calculation Module:**}
$netlist\_info \leftarrow NetlistParser(design.v)$;
for each net in $netlist\_info$ **do**
  if net in $nets\_info.DesignInput$ **then**
    $net.prob_0 \leftarrow 0.5$
    $net.prob_1 \leftarrow 0.5$
  end if
end for
for each $gate$ in $netlist\_info$ **do**
  $prenets \leftarrow gate.innets$
  for each net in $gate.outnets$ **do**
    $(net.prob_0, net.prob_1) \leftarrow PC(prenets, gate, lib.v)$
    {PC is probability calculation function};
    $(net.tranprob) \leftarrow net.prob_0 \times net.prob_1$
  end for
end for
{**Nets Classification Module:**}
for each net in $netlist\_info$ **do**
  if $net.tranprob < \theta_{th}$ **then**
    $LowGroup \leftarrow LowGroup \cup net$
  else
    $NormGroup \leftarrow NormGroup \cup net$
  end if
end for
{**Trigger Nets Selection Module:**}
$list_{low\_trig} \leftarrow RSel(no\_low\_trig, LowGroup)$
$list_{normal\_trig} \leftarrow RSel(no\_normal\_trig, NormGroup)$
return $list_{low\_trig}$, $list_{normal\_trig}$;

functional modules involved in this procedure which are Transition Probability Calculation, Net Classification and Trigger Net Selection. First, as the transitions in a netlist are generally induced by transitions in scan cells and primary inputs [11], the $P0$ and $P1$ of each scan flip-flop and primary input are set to 0.5 respectively. Next the switching probability of each net is calculated respectively based on the input probabilities and the logic function of the component. As a trade-off between memory space consumption and precision, reconvergent fanouts are not considered to simplify the calculation. Then, the Net Classification module classifies all of the nets into a low activity list or a normal list based on the transition probability threshold, θ_{th}, as defined by the user.

Lastly, the Trigger Net Selection module will randomly select nets from these two lists based on the number of low activity triggers and normal triggers defined in the parameters. Normal trigger pins are used to obfuscate the trigger condition against smart adversaries as mentioned in [7].

TABLE III
CIRCUIT TYPES IN TROJAN LIBRARY

HT Part	Type Name	Description
Trigger	comb	Combinational Trigger composed of AND gates
	seque_CNT	Counter based sequential trigger
	seque_FSM	Finite-state machine based sequential trigger
	user_def_trig	User-defined trigger circuits
Payload	func	XOR gate based functional error payload
	leak_LFSR	Linear feedback shift register based information leakage payload
	leak_SHIFT	Shift register based information leakage payload
	user_def_pay	User-defined payload circuits

C. Built-in Trojan Types

In order to generate a wide variety of HTs, a extensible Trojan library which contains different types of basic Trojan circuits has been defined in this platform. The Trojan library is composed of trigger circuits and payload circuits. In order to improve the flexibility of the HTs generated, the Trojan library can be extended by adding user-defined trigger and payload circuits. Table III lists the circuit types defined in the Trojan library. As the structures of the "comb" trigger and "func" payload are simple, Fig. 4 shows the structure of the four other types of Trojan circuits given in Table III.

Based on the trigger and payload type defined in the configuration dictionary, different kinds of HTs can be generated. For example Fig. 4(e) is a FSM-based sequential trigger with a shift register (SHIFT) based information leakage payload.

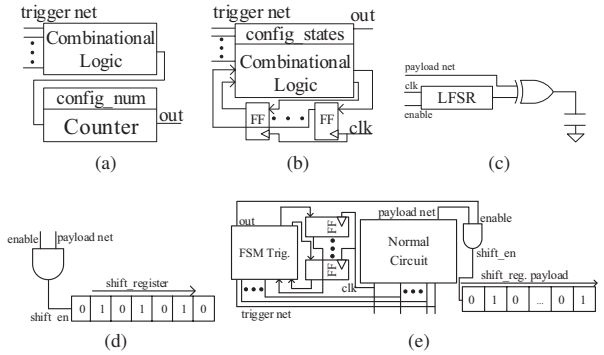


Fig. 4. (a) Seque_CNT; (b) Seque_FSM; (c) Leak_LFSR; (d) Leak_SHIFT; (e) FSM based sequential Trojan with a SHIFT based leakage payload

The FSM-based trigger is activated only when a series of sequential signals have been followed in a particular order. The shift register is pre-loaded with an alternating sequence of zeros and ones. The payload is only activated when both the trigger output and payload net is high. The shifting action can result in additional dynamic power consumption to leak internal signal state [6]. If another trigger or another payload type is selected, the structure could also follow one of the designs in Fig. 4(a)(b)(c)(d) or user-defined ones.

D. Payload Net selection

Payload nets refer to the attacked nets that are leaked by an information leakage payload or affected by a functional error payload. Users can manually specify the payload nets or enable random selection. In order to guarantee an effect on payload nets, the random payload net selection follows several rules in the proposed platform.

- The trigger nets are not allowed to be payload again.
- The topological order of each payload net should be larger than all trigger nets to avoid logical loops [7].
- Nets related to scan-chain and clock signals are avoided in the payload net selection process because of their higher risk of detection.

E. Trigger Condition Check

Before the HT insertion and net connection step, the Trigger Condition Check module ensures that the HTs can be correctly triggered under rare conditions. As all of the input pins of the trigger circuits in the Trojan library have been designed to be activated at a high level to simplify the coding of the Trojan generation process, this module checks the signal probability of each selected trigger net and adds an inverter to the trigger pin when needed.

IV. Experimental Evaluation

A. Experimental Setup

The proposed platform was implemented in Python using open source tools [12]. The calculation of the transition probability is based on the TPC script provided by Trust-Hub [10]. This platform supports reading in RTL or gate-level designs in Verilog and all of the design files are compiled into ungrouped gate-level netlists using the Synopsys SAED90nm standard cell library.

The platform has been tested to generate HT-infected circuits from ISCAS benchmark circuits [13] and opensource cores (UART [14] and AES-128 [15]). In order to evaluate the generated HT-infected circuits, and to allow for a comparison with the evaluation results in [7], the machine learning based HT detection technique, COTD, adopted in [7], is then applied to the generated HT-infected circuits. It utilizes unsupervised k-means clustering to analyze the combinational controllability (CC) and observability (CO) of the gate-level netlists and isolates Trojan signals from normal ones [3]. We utilized TetraMAX to extract the controllability and observability of the netlists and used MATLAB to do the k-means clustering in the COTD approach. The clustering iteration number is set to 5 to get a stable classification result. The trigger condition of all the benchmark circuits is set to 5 low activity nets, and 1 normal net. The resulting false positive (FP) and false negative (FN) rates are obtained and illustrate the feasibility of the benchmark circuits generated.

B. Experimental Results

Table IV present the results of applying COTD to the HT-infected circuits generated by our platform. We inserted HTs into both scan-enabled and non-scan implementations of benchmark circuits(s13207 and s15850). For each implementation, 4 types of HT-infected circuit are generated and tested (Column 3 and 4). The transition probability thresholds are listed in column 2. Column 5 lists the number of normal signals in each netlist while column 6 lists Trojan ones. FN in column 7 is the number of Trojan signals detected as normal ones (FNR=$FN/No.HT\ signals$), while FP in column 8 means normal signals detected as Trojan ones (FPR=$FP/No.Normal\ signals$). Column 9, 10 and 11 present the number of normal signals and HT signals inside each cluster. The signals in the Normal Cluster generally have low CO and CC values, while signals in HT Cluster 1 and HT Cluster 2 have either high CO values or high CC values.

From Table IV we observed that the detection efficiency on non-scan implementations are lower than scan-enabled ones according to both the high FNR and high FPR. In most cases, the Trojan circuits with a combinational trigger and functional error payload result in the highest FPR. Trojans with an information leakage payload result a high FNR in both scan-inserted and non-scan implementations. Compared with HTs with a sequential trigger, HTs with a combinational trigger are easier to detect according to the low FNR.

When compared with the COTD detection results of the HT-infected s13207 and s15850 benchmarks from [7] (see Table V), the false negative rates (FNRs) are all zero, which means all Trojan signals are detected, while in our equivalent benchmark circuits (s13207_scan, s15850_scan) the FNRs are non-zero values, which means some Trojan signals are classified as normal ones so the samples generated by our platform gives more misclassification errors using COTD detection.

TABLE IV
EXPERIMENTAL RESULTS OF COTD-BASED HT DETECTION ON GENERATED HT-INFECTED BENCHMARKS

Benchmark	θ_{th}	Trigger Type	Payload Type	No. Normal Signals	No. HT Signals	FN FNR(%)	FP FPR(%)	Normal Cluster (Norm/HT)	HT Cluster 1 (Norm/HT)	HT Cluster 2 (Norm/HT)
s13207	0.1	comb	func	6622	22	1(4.5)	5210(78)	1412/1	4900/21	310/0
	0.1	comb	leak_LFSR	6622	175	95(54)	1936(29)	4686/95	1626/46	310/34
	0.1	seque_CNT	func	6622	54	50(92)	1936(29)	4686/50	1626/4	310/0
	0.1	seque_fsm	leak_SHIFT	6622	210	48(23)	4996(75)	1626/48	4686/154	310/9
s13207_scan	0.1	comb	func	7946	26	6(23)	31(0.39)	7915/6	11/10	20/10
	0.1	comb	leak_LFSR	7946	173	61(35)	20(0.25)	7926/61	20/33	0/79
	0.1	seque_CNT	func	7946	52	10(19)	33(0.41)	7913/10	12/41	21/1
	0.1	seque_fsm	leak_SHIFT	7946	209	36(17)	31(0.39)	7915/36	20/130	11/43
s15850	0.1	comb	func	7310	22	18(82)	2328(32)	4982/18	2094/4	234/0
	0.1	comb	leak_LFSR	7310	173	148(86)	2320(31)	4990/148	234/2	2086/23
	0.1	seque_CNT	func	7310	56	47(84)	2293(31)	5017/47	285/0	2008/9
	0.1	seque_fsm	leak_SHIFT	7310	180	178(99)	2328(32)	4982/178	234/0	2094/2
s15850_scan	0.1	comb	func	8207	24	6(25)	1015(12)	7192/6	1007/18	8/0
	0.1	comb	leak_LFSR	8207	175	70(40)	8(0.10)	8199/70	8/32	0/73
	0.1	seque_CNT	func	8207	58	10(17)	9(0.11)	8198/10	1/20	8/28
	0.1	seque_fsm	leak_SHIFT	8207	178	34(19)	8(0.10)	8199/34	0/50	8/94
uart	0.15	comb	func	840	22	0(0)	768(91)	72/0	66/4	702/18
	0.15	comb	leak_LFSR	840	175	44(25)	768(91)	72/44	75/89	693/42
	0.15	seque_CNT	func	840	56	4(7)	768(91)	72/4	66/0	702/52
	0.15	seque_fsm	leak_SHIFT	840	178	49(28)	768(91)	72/49	66/46	702/83
aes-128	0.15	comb	func	636704	26	1(4)	598750(94)	37954/1	414544/19	184206/6
	0.15	comb	leak_LFSR	636704	177	108(61)	598722(94)	37982/108	414516/26	184206/43
	0.15	seque_CNT	func	636704	58	43(74)	598750(94)	37954/43	414543/4	184207/11
	0.15	seque_fsm	leak_SHIFT	636704	176	157(89)	596619(94)	40085/157	414544/7	184075/12

TABLE V
COTD-BASED HT DETECTION RESULTS FROM [7]

Benchmarks	Trigger Conditon (Rare/Total)	Type	FPR(%)	FNR(%)
s13207-c5_6	5/6	comb	25	0
s13207-s5_6	5/6	seq	0.11	0
s15850-c5_6	5/6	comb	27	0
s15850-s5_6	5/6	seq	0.09	0

Overall, the detection results show this platform was able to generate failing test conditions for COTD detection with a high FNR on nearly all generated HT-infected circuits.

V. CONCLUSION

In this paper, we propose an new method to generate HTs using a highly configurable generation platform based on transition probability to identify the rarely activated internal nodes to target for HT insertion, rather than functional simulation as used in existing platforms. The platform is highly configurable and can be easily updated and support user-defined HT circuits. The platform has been tested to generate HT-infected circuits from ISCAS benchmark circuits [13] and evaluated by the COTD detection technique.

In future, we will extend the built-in Trojan library and update the platform to support new HT insertion methodologies. The HT-infected benchmarks generated by this platform will be made publicly available to the research community for research and evaluation.

REFERENCES

[1] C. Bao, D. Forte, and A. Srivastava, "On reverse engineering-based hardware trojan detection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 1, pp. 49–57, Jan 2016.

[2] Y. Huang, S. Bhunia, and P. Mishra, "Scalable test generation for trojan detection using side channel analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 11, pp. 2746–2760, Nov 2018.

[3] H. Salmani, "Cotd: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 2, pp. 338–350, Feb 2017.

[4] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Hardware trojans classification for gate-level netlists using multi-layer neural networks," in *Proc. IEEE 23rd Int. Symp. On-Line Testing and Robust System Design*, July 2017, pp. 227–232.

[5] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. on Circuits and Systems*, May 1989, pp. 1929–1934 vol.3.

[6] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, Mar 2017.

[7] J. Cruz, Y. Huang, P. Mishra, and S. Bhunia, "An automated configurable trojan insertion framework for dynamic trust benchmarks," in *Proc. Design, Automation Test in Europe Conf. Exhibition*, March 2018, pp. 1598–1603.

[8] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware trojan detection and reducing trojan activation time," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 20, no. 1, pp. 112–125, Jan 2012.

[9] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *Proc. IEEE 31st Int. Conf. Computer Design*, Oct 2013, pp. 471–474.

[10] H. Salmani and M. Tehranipoor, "Trust-Hub," accessed on 2018-08-02. [Online]. Available: https://www.trust-hub.org/home

[11] R. Sankaralingam, R. R. Oruganti, and N. A. Touba, "Static compaction techniques to control scan vector power dissipation," in *Proc. 18th IEEE VLSI Test Symp.*, April 2000, pp. 35–40.

[12] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for verilog hdl," in *Proc. Int. Symp. Applied Reconfigurable Computing*, 2015, pp. 451–460.

[13] M. Jenihhin, "Iscas89 verilog benchmark," accessed on 2018-06-15. [Online]. Available: http://www.pld.ttu.ee˜maksim/benchmarks/iscas89

[14] Jamieiles, "Verilog uart," accessed on 2019-02-05. [Online]. Available: https://github.com/jamieiles/uart

[15] H. Hsing, "Tiny aes," accessed on 2019-02-05. [Online]. Available: https://opencores.org/projects/tiny_aes