Compatibility Graph Assisted Automatic Hardware Trojan Insertion Framework

Gaurav Kumar, Ashfaq Hussain Shaik, Anjum Riaz, Yamuna Prasad*, and Satyadev Ahlawat
Dept. of EE, Indian Institute of Technology Jammu, India; {gaurav.kumar, 2022pvl0070, anjum.riaz, satyadev.ahlawat}@iitjammu.ac.in
*Dept. of CSE, Indian Institute of Technology Jammu, India; yamuna.prasad@iitjammu.ac.in

***Abstract*—Hardware Trojans (HTs) pose substantial security threats to Integrated Circuits (ICs), compromising their integrity, confidentiality, and functionality. Various HT detection methods have been developed to mitigate these risks. However, the limited availability of comprehensive HT benchmarks necessitates designers to create their own for evaluation purposes. Moreover, the existing benchmarks exhibit several deficiencies, including a restricted range of trigger nodes, susceptibility to detection through random patterns, lengthy HT instance creation and validation process, and a limited number of HT instances per circuit. To address these limitations, we propose a Compatibility Graph assisted automatic Hardware Trojan insertion framework for HT benchmark generation. Given a netlist, this framework generates a design incorporating single or multiple HT instances according to user-defined properties. It allows various configurations of HTs, such as a large number of trigger nodes, low activation probability and large number of unique HT instances. The experimental results demonstrate that the generated HT benchmarks exhibit exceptional resistance to state-of-the-art HT detection schemes. Additionally, the proposed framework achieves an average improvement of 37815.7x and 989.4x over the insertion times of the Random and Reinforcement Learning based HT insertion frameworks, respectively.**

***Index Terms*—Integrated Circuits, Hardware Trojan, Hardware Security, 3PIP, Compatibility Graph**

I. Introduction

The security of electronic systems has become a critical concern for both industries and security agencies. The cyber physical attacks targeting critical infrastructure such as finance, banking sectors, avionics, power grids, and military systems can severely disrupt national infrastructure and pose significant threats to national security. Additionally, the competitive market pressure for rapid product launches has driven the adoption of IP-based design methodologies. This approach involves integrating IP cores outsourced globally from multiple third-party providers (3PIPs). However, these external sources lack complete reliability, posing potential security threats to chip integrity through Hardware Trojans (HTs) [1]. In the subsequent sections of this paper, the term "trojan" will exclusively refer to a Hardware Trojan unless explicitly stated otherwise. Once the trojan is triggered, it may use a covert channel to leak sensitive information or mount a Denial-of-Service (DoS) attack. The consequences of such attacks can be severe, especially in critical applications like medical devices, automotive systems and military equipment, where continuous operation and reliability are crucial [2]. The stealthy nature of these trojans makes them particularly dangerous. They can evade detection by activating under very specific conditions, making them difficult to anticipate and counteract. Therefore, to comprehend the implications of Hardware Trojans and assess the efficacy of trojan detection methods, researchers often use Hardware Trojan benchmark circuits from Trust-Hub [3] or develop their own trojan circuits. However, these trojan benchmarks have limitations, such as a restricted number of available instances per circuit and the number of trigger nodes per instance, making it challenging to thoroughly evaluate current trojan detection methodologies. Furthermore, for generating each trojan instance, the selection of trigger nodes and its validation consume a significant amount of time. Recently, the use of the Reinforcement Learning (RL) [4], [5] model to generate the Hardware Trojan benchmarks has gained momentum. Nevertheless, this approach is also time-intensive, requiring extensive data collection on parameters and prolonged model training time.

In this paper, we aim to address the limitations of existing HT insertion frameworks by introducing a novel approach for trojan instance generation and insertion. In the proposed HT insertion framework, a compatibility graph is employed to identify and create the subsets of rare nodes that can be used as trigger nodes. It has been shown that the proposed framework is able to generate trojan instances with a large number of rare nodes as trigger nodes, ranging from 25 to 125, demonstrating the scalability of the proposed approach. The main contributions of this paper are as follows:

- The proposed framework optimizes HT insertion time by systematically selecting rare nodes as trigger nodes.
- The proposed framework generates trojan instances by leveraging the inherent biases of logic gates towards the rare values of selected rare nodes used as trigger nodes.
- The generated trojan instances have a large number of trigger nodes and negligible area overhead, making them resistant to state-of-the-art HT detection schemes.

The rest of the paper is organized as follows: Section II provides a brief overview of the existing HT insertion framework. The proposed Hardware Trojan insertion framework is explained in Section III. Section IV present experimental results. Finally, Section V concludes the study.

II. Background

Various Hardware Trojan detection methodologies have been proposed based on Side-Channel Analysis (SCA) [6]–[10] and Simulation-based Validation (Logic Testing) [11]–[23]. SCA

978-3-9826741-0-0/DATE25/© 2025 EDAA

based schemes identify trojans by examining discrepancies between expected and actual physical signatures, such as power consumption, timing delays, or area overhead. On the other hand, Logic Testing based methods focus on the functional behavior of the circuit to detect trojans. By analyzing the circuit's response to specific input patterns, these methods aim to reveal any hidden malicious circuit modifications. MERO (Multiple Excitation of Rare Occurrence) [24] was the first statistical algorithm designed to generate the test vectors for trojan detection. The algorithm operates on the statistical principle that increasing the number of times each rare node is activated to its rare value, enhances the probability of the trojan activation. Therefore, this approach focuses on activating rare logic conditions at internal nodes multiple times to enhance the HT detection probability. Recently, Jayasena et al. proposed N-Activation of Rare Events With ATPG (ND-ATPG) [25] scheme. The ND-ATPG scheme generate the test vectors for trojan detection based on the N-detect principle of stuck-at faults. For this, it converts every rare event (which involves a rare value at a rare node) into a stuck-at-fault. Subsequently, it employs ATPG to generate N test vectors corresponding to each rare event. In this way, it ensures that each rare node is triggered N times to its rare value. It should be noted that as the value of N increases, the probability of trojan activation increases. However, it will also increase the test vector generation time. The main challenge in evaluating the effectiveness of these trojan detection schemes is the limited availability of standardized HT benchmarks. Without reliable benchmarks, it's difficult to measure and compare the effectiveness of different trojan detection methodologies. To address this need, Shakya et al. [3] have introduced the Trust-Hub benchmark. This benchmark was generated using a tool designed to evaluate design vulnerabilities at multiple levels, including RTL, gate, and layout. The primary objective of this assessment was to pinpoint hard-to-detect nodes/signals, such as faults that cannot be detected using conventional fault-testing techniques, as well as regions exhibiting minimal changes in critical parameters such as power, area, and delay. Subsequently, these nodes/signals were quantified to determine their significance as potential trigger nodes for trojan activation. The Trust-Hub benchmark includes both gate level and RTL level Hardware Trojans, which have been manually inserted in various benchmarks such as ISCAS-85 [26], ISCAS-89 [27], EthernetMAC10GE, and RS232. Consequently, a large number of trojan instances can not be generated. In contrast to the TrustHub trojan benchmarks, a novel methodology for automated Hardware Trojan insertion was introduced by Cruz et al. [28]. In this scheme, the trojan instances are automatically generated based on user-provided inputs such as the number of trigger nodes (q), the rareness threshold (θ), activation mechanism and the trojan effect. Initially, a specific trojan template is generated according to these input parameters. Subsequently, HT-infected netlists are generated by inserting the trojan template. Although this approach increases the variety of inserted trojans, it does not provide a solution for identifying the optimal set of rare nodes as trigger nodes. The TRIT trojan benchmark generated by this tool is available on the Trust-Hub website [29] with a maximum number of trigger nodes (q) as 7. Later, the same authors introduced a machine learning based HT insertion framework, MIMIC, for the automatic generation of trojan benchmarks [30]. This framework utilizes pre-existing trojans to extract 16 functional and structural features, including signal probability, switching probability, SCOAP parameters, etc. These features are then used to train machine learning models to generate a large number of hypothetical trojans for a given circuit/design. However, the insertion criteria for these trojans are very similar to those in [28], suffering from the same shortcomings. Later, Fyrbiak et al. [31] proposed HAL, a reverse engineering mechanism for trojan insertion. HAL conducts reverse engineering on low-level, unfolded, placed, and routed gate-level netlists. HAL's design is particularly noteworthy for its flexibility, allowing users to navigate and manipulate complex netlists with relative ease. However, this flexibility comes at a cost, the HAL's insertion process requires significant manual effort. This manual intervention limits the tool's scalability, making it challenging to generate a large dataset of trojan instances efficiently. Moreover, the lack of automation in the trojan insertion process undermines the reliability of the benchmarks for evaluation purposes.

Recently, Gohil et al. [32] introduced ATTRITION, a framework leveraging Reinforcement Learning (RL) [5] for trojan insertion. ATTRITION employs an RL model to identify and create subsets of rare nodes that can be utilized as trigger nodes for trojan activation. Despite its innovative approach, ATTRITION faces several challenges. The RL model requires extensive training time, which can be a bottleneck for practical implementation, especially with complex and large-scale designs. Furthermore, the number of trojans that can be generated using this method is limited, which restricts its applicability in scenarios requiring a large variety of trojan instances for robust testing and evaluation of HT detection schemes. Later, Sarihi et al. [4] have proposed a framework that extends the RL-based approach by incorporating rare nodes and SCOAP (Sandia Controllability/Observability Analysis Program) parameters [33]. This enhancement aims to further refine the process of HT insertion by leveraging additional circuit analysis metrics, potentially improving the accuracy and efficiency of the RL model in identifying optimal trigger nodes. Although it presents a novel method, the training time of the RL model is extensive. The $RL\text{-}ISCAS\text{-}85$ trojan benchmark generated by this tool is available on GitHub [34], with a maximum of five trigger nodes. Table I provides a comprehensive overview of the existing frameworks/tools in the trojan insertion domain.

TABLE I: Analysis of Existing HT insertion frameworks/tools

Framework/ Tool	Insertion Criteria	Autom-ation	Open-source
Trust-Hub [3]	Secret leakage, Signal probability	✗	✗
TRIT [28]	Signal probability	✓	✗
MIMIC [30]	Structural & Functional features	✓	✗
HAL [31]	Neighbourhood control value	✗	✓
ATTRITION [32]	Signal probability	✓	✗
RL-ISCAS-85 [4]	SCOAP parameters	✓	✗

III. METHODOLOGY

In this section, we will explain the working of the proposed Hardware Trojan insertion framework. To design a stealthy Hardware Trojan, two fundamental criteria must be satisfied. The first criterion is that the trojan should activate only under rare conditions, ensuring minimal interference with normal circuit operation. The primary challenge in satisfying this criterion is identifying the insertion points/trigger nodes for the trojan activation. If circuit nodes are selected randomly for trojan activation, then they may be activated during the test phase or normal operation. For this, rare nodes are identified using functional simulation (see Subsection III-B), and subsets of these rare nodes are generated using the compatibility graph (see Subsection III-C). The second criterion of the stealthy trojan is that it must be difficult to detect. To satisfy this criterion, specific stealthy trojan instances are created according to the selected subsets of the rare nodes (see Subsection III-D). These generated trojan instances are then inserted into the netlist, resulting in HT-infected circuits. Each step of the proposed framework is explained in the following subsections.

A. Netlist to Data structure

To facilitate HT-free (original) circuit simulation, the netlist is converted into a directed acyclic graph (DAG). Formally, the graph G can be defined as: $G = (V, E)$, where each vertex $v \in V$ represents a logic gate, and each edge $e \in E$ represents a connection between gates. This representation allows for efficient simulation and analysis of the given netlist.

B. Extraction of Rare nodes

To satisfy the first criterion of stealthy HT, i.e., trojan should activate only under rare conditions, rare nodes of circuits are used as trigger nodes for trojan activation. For this, functional simulation is employed to identify the rare nodes of given circuits corresponding to rare values 0 and 1. Initially, vectors from a random vector set (V) are simulated on the circuit netlist. For each input vector v, the logic value at each node is assessed. After all input vectors have been simulated, the frequency with which each node within the circuit reaches to a specific logic value during functional operations is determined. If the number of times a node reaches a specific value is less than a set threshold θ_{RN}, the node is marked as a rare node. The procedure to extract rare nodes is detailed in Algorithm 1. It is important to note that the selection of θ_{RN} and optimal size for the vector set V is crucial for the test generation process. A very small value of θ_{RN} may result in all nodes within the circuit being marked as non-rare. Conversely, a very large θ_{RN} value may result in all nodes being marked as rare. Furthermore, an inappropriate size may increase the time complexity by inaccurately marking the non-rare nodes as rare nodes. The selection of both hyperparameters, namely θ_{RN} and the size of V, is discussed in Subsection IV-A.

Algorithm 1 Extraction of Rare Nodes (Extraction_RN)

Input: Gate-level netlist G, Number of Nodes in netlist K, Random test vector set V, Rare node threshold θ_{RN}
Output: Rare nodes for logic value 1 ($RN1$) and Rare nodes for logic value 0 ($RN0$)

$RN1 \leftarrow \{\}$, $RN0 \leftarrow \{\}$
$count_{C1}[1 \dots K] \leftarrow 0$, $count_{C0}[1 \dots K] \leftarrow 0$
for each vector v in V **do**
    value $\leftarrow$ Apply(v, G) ▷ Simulate vector v
    for each node n_i in G **do**
        if value(n_i) == 1 **then**
            $count_{C1}[n_i] \leftarrow count_{C1}[n_i] + 1$
        else
            $count_{C0}[n_i] \leftarrow count_{C0}[n_i] + 1$
        end if
    end for
end for
for each node n_i in G **do**
    if $count_{C1}[n_i] \leq \theta_{RN}$ **then**
        $RN1 \leftarrow RN1 \cup n_i$
    else if $count_{C0}[n_i] \leq \theta_{RN}$ **then**
        $RN0 \leftarrow RN0 \cup n_i$
    end if
end for
return Rare nodes lists $RN1$, $RN0$

C. Selection of Rare nodes as Trigger nodes

After identifying rare nodes from the circuit netlist, the next step is to create the subsets of rare nodes that will serve as trigger nodes of trojan instances. However, random selection can lead to significant generation time due to the validation process. This is because not all subsets of these rare nodes can be triggered by a single test vector. To address this, we have proposed a compatibility graph based method to create subsets of compatible rare nodes. The compatible rare nodes are those rare nodes which can be triggered to their respective rare value using a single test vector. These subsets will then be used as trigger nodes for the inserted trojan instances. Initially, the PODEM ATPG [35] algorithm is executed to determine test vector which can trigger the rare node n to its respective rare value r. For this, the PODEM's primary objective is set to generate a test vector for node n having a stuck-at-$\bar{r}$ fault. Now, to evaluate the compatibility between two rare nodes, N_1 and N_2, the test vectors TV_1 and TV_2 generated by PODEM are compared. If there are no conflicts between the care bits of TV_1 and TV_2, the test vectors are considered mergeable, and the corresponding nodes N_1 and N_2 are marked as compatible. It should be noted that most of the bits in these test vectors are don't care, i.e. X, allowing flexibility in their values. Conversely, if there are any conflicts between the care bits of TV_1 and TV_2, N_1 and N_2 are considered incompatible. This means that there exist no test vectors that can simultaneously trigger these rare nodes to their specific rare values. This compatibility check is performed for all pairs of rare nodes, resulting in a compatibility graph where nodes represent the rare nodes and edges represent compatibility between them. Within this graph, the complete subgraphs identify the clusters of rare nodes that can be triggered simultaneously with a single test vector. These complete subgraphs are the basis for selecting

Algorithm 2 Generation of Compatibility graph (Gen_compatibility)

Input: Rare nodes for logic value 1 ($RN1$) and Rare nodes for logic value 0 ($RN0$), Number of rare nodes as trigger nodes (q), Number of trojan instances (N)
Output: Complete subgraphs ($complete\_subgraphs$)

$TVS \leftarrow \{\}$
$complete\_subgraphs \leftarrow \{\{\}\}$
$TVS \leftarrow PODEM(RN1, RN0)$
$G \leftarrow nx.Graph()$ ▷ Initialize Graph
$G.addnodes(RN1, RN0)$
for $i \leftarrow 0$ to $len(TVS) - 1$ **do**
  for $j \leftarrow$ i+1 to $len(TVS) - 1$ **do**
    if $check\_compatibility(TVS(i), TVS(j)) == 1$ **then**
      $G.add\_edge(TVS(i), TVS(j))$
    end if
  end for
end for
$complete\_subgraphs \leftarrow find\_cliques(G, q, N)$
return $complete\_subgraphs$

rare nodes as trigger nodes for trojan activation. Each complete subgraph can be used as an insertion point of a trojan instance. In this way, the proposed approach ensures that there exists at least one test vector (test vector after merging TV_1 and TV_2) that can simultaneously trigger these rare nodes to their specific rare values, thereby eliminating the validation time in the trojan generation process. The above procedure for selecting the trigger nodes is detailed in Algorithm 2.

D. HT Instance Generation and Insertion

After identifying the insertion points/trigger nodes, the next step is to create a stealthy trojan instance. To achieve this and to meet the second criterion for a stealthy HT, i.e., ensuring the trojan is difficult to detect, we leverage the inherent output bias of logic gates. For instance, OR and $NAND$ gates exhibit an inherent bias toward generating a logic 1 output. This is because all inputs must be 0 (1) simultaneously for the OR (NAND) gate

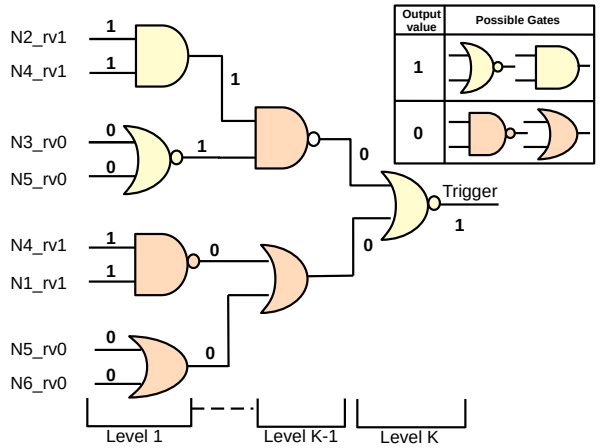

Fig. 1: Generation of Trigger logic of Hardware Trojan

to generate an output 0. Consequently, the probability of these gates generating a logic 0 output is $1/2^k$, assuming the circuit exclusively comprises k-input logic gates. Furthermore, AND and NOR gates exhibit an inherent bias toward generating a logic 0 output. Consequently, the probability of these gates generating a logic 1 output is $1/2^k$. To effectively utilize these probabilities for generating the trigger logic of trojan instances, a backward approach is employed, starting from the output (level K) and moving towards the input (level 1). Suppose the trojan trigger logic needs to generate a logic 1 (0) to activate the trojan instance. Due to output bias, only AND and NOR ($NAND$ and OR) gates can be used at level K for this purpose. This is because AND and NOR ($NAND$ and OR) gates are inherently biased towards generating a logic 0 (1) output, making a logic 1 (0) output rare and thus suitable for stealthy activation. Now, if a NOR/AND (OR/$NAND$) gate is chosen at level K, it requires all its inputs to be 0 (1) in order to produce a logic 1 (0) output. Therefore, OR and $NAND$ (AND and NOR) gates are the only suitable gates for level $K-1$. This process is repeated iteratively, selecting gates for each preceding level until level 1 is reached. In this way, this method ensures that the designed trojan instances are activated under rare conditions and thus remain stealthy and difficult to detect. If an attacker uses q rare nodes, the trigger probability of the generated trojan instance can be expressed as $1/2^{k^q}$. An example of a trojan instance generated using this method is shown in Figure 1. Once the trojan instance has been generated, it needs to be integrated into the netlist to create

Algorithm 3 Generation of HT-infected netlist

Input: Gate-level netlist (G), Complete subgraphs ($complete\_subgraphs$), HT trigger logic ($HT\_instance$), Payload Net ($payload\_net$), Number of HT instances (N)
Output: HT-infected netlists ($\tilde{G}$)

$\tilde{G} \leftarrow \{\}$
for $i \leftarrow 1$ **to** N **do**
  $GN \leftarrow G$
  for each $g \in$ *HT_instance*$[1]$ **do** ▷ Trigger logic level 1
    $g_{\text{type}} \leftarrow$ *HT_instance*$[1][g]$
    $C_i \leftarrow$ *complete_subgraphs*$[i]$
    $R_0 \leftarrow \{v \in C_i \mid \text{node } v \text{ has rare value } 0\}$
    $R_1 \leftarrow \{v \in C_i \mid \text{node } v \text{ has rare value } 1\}$
    if $g_{\text{type}} == AND$ or $g_{\text{type}} == NAND$ **then**
      Connect inputs of g to nodes in R_0
      Update $E \leftarrow E \cup \{payload\_net\}$
      Update $V \leftarrow V \cup \{g\}$
    else
      Connect inputs of g to nodes in R_1
      Update $E \leftarrow E \cup \{\text{new connections}\}$
      Update $V \leftarrow V \cup \{g\}$
    end if
  end for
  $\tilde{G}[i] \leftarrow GN$
end for
return HT-infected netlists $\tilde{G}$

the HT-infected netlist. This integration involves connecting the inputs of the trojan instance to any of the identified rare nodes within the chosen subset. However, if the inputs of the trigger logic are connected to rare nodes arbitrarily, there is a risk that one of these inputs might be connected to a rare node that has a rare value of 1 (or 0). However, the trojan may require a non-rare value (0 or 1) to activate. This misalignment between the required activation value and the actual rare value of the node undermines the trojan's ability to remain undetected, as the trojan may activate under non-rare conditions. To address this, the inputs of trojan instances must be connected to rare nodes based on their rare values. Specifically, rare nodes with a rare value of logic 1 should be connected to AND and $NAND$ gates in the trigger logic of the trojan instance. Conversely, rare nodes with a rare value of logic 0 should be connected to OR and NOR gates in the trigger logic. This careful alignment of rare nodes ensures that the trojan is activated only when all rare nodes are triggered to their corresponding rare values. The process of generation of HT-infected netlist is detailed in Algorithm 3.

IV. Experimental Results

This section initially discusses the hyperparameters required by the algorithm in the proposed HT insertion framework. Subsequently, the effectiveness of the proposed trojan insertion framework is evaluated. The framework was implemented in the C++ programming language. Additionally, the netlists from the ISCAS85 and ISCAS89 benchmarks were used as original designs for HT insertion. The experiments were conducted on a standard computing platform with an Intel Core i7 processor and 32 GB of RAM.

A. Hyperparameters of Extraction of Rare Nodes ((Extraction_RN) Algorithm 1)

Algorithm 1 takes two hyperparameters. These hyperparameters are θ_{RN} and V. If a node is excited to a logic value (0 or 1) less than θ_{RN} times, it is considered a rare node. Figure 2 shows

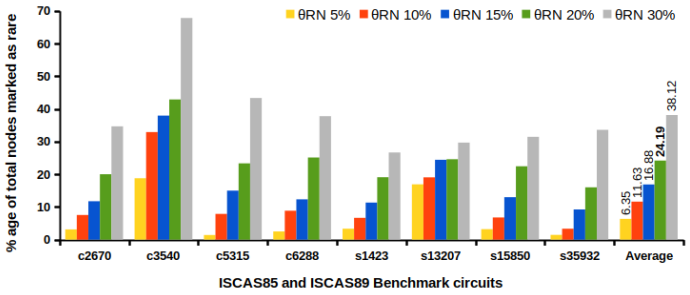

Fig. 2: Number of rare nodes for various rareness thresholds

the number of rare nodes for different rareness threshold values, θ_{RN}, across ISCAS85 and ISCAS89 benchmark netlists. It can be observed from Figure 2 that a value of 20% as θ_{RN} marked 24.19% of total nodes as rare nodes on average. Furthermore, rareness threshold values of 5%, 10% or 15% marked only 6.35%, 11.63%, and 16.88% of the total nodes as rare nodes, respectively. Additionally, a large value of θ_{RN}, i.e., 30%, results in too many nodes (38.12%) marked as rare nodes. Consequently, for the proposed trojan insertion framework, θ_{RN} is selected as 20%. The second hyperparameter is the size of the random test vector set V applied to the netlist to determine the rare nodes. It can be observed from Figure 3 that the number of rare nodes becomes constant after applying $10,000$ test vectors for various benchmark circuits. Consequently, size

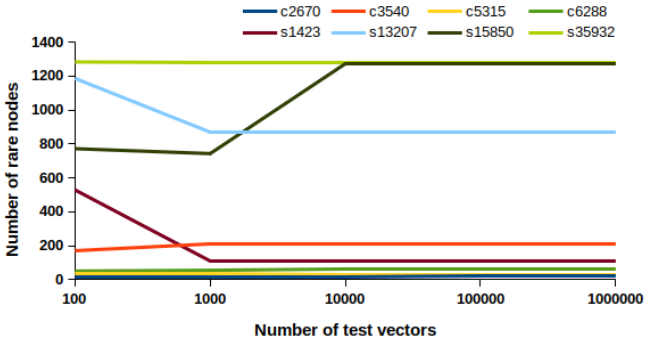

Fig. 3: Number of rare nodes for various numbers of test vectors

TABLE II: Detection Analysis of Various Trojan Benchmarks generated by Various HT Insertion Frameworks

HT insertion framework	Total HT Netlists	Detection Schemes	Measure	Benchmark circuits								%age Coverage
				c2670	c3540	c5315	c6288	s1423	s13207	s15850	s35932	
Random HT Benchmarks	800	Random	TC	12	9	11	8	4	10	12	-	8.25
			DC	7	4	5	4	3	9	8	-	5.00
		MERO	TC	14	12	12	9	7	12	14	-	10.00
			DC	7	5	4	6	3	11	6	-	5.25
		ATPG	TC	17	13	12	11	8	14	14	-	11.12
			DC	9	7	7	6	6	12	9	-	7.00
Reinforcement Learning Benchmarks	400	Random	TC	94	94	93	100	-	-	-	-	95.25
			DC	89	91	90	100	-	-	-	-	92.50
		MERO	TC	96	93	93	100	-	-	-	-	95.50
			DC	93	81	90	100	-	-	-	-	91.00
		ATPG	TC	94	93	92	100	-	-	-	-	94.75
			DC	90	85	85	100	-	-	-	-	90.00
Trust-Hub Benchmarks	560	Random	TC	16	27	15	4	3	1	0	27	21.13
			DC	5	15	7	4	0	0	0	4	7.95
		MERO	TC	21	24	13	3	3	0	0	11	17.04
			DC	6	12	6	3	2	0	0	11	8.86
		ATPG	TC	17	30	16	2	3	1	1	30	22.72
			DC	5	20	9	2	2	1	1	26	15.00
Proposed HT Benchmark	8000	Random	TC	0	0	0	0	0	0	0	0	**0**
			DC	0	0	0	0	0	0	0	0	**0**
		MERO	TC	0	0	0	0	0	0	0	0	**0**
			DC	0	0	0	0	0	0	0	0	**0**
		ATPG	TC	0	1	0	2	0	0	0	0	**0.0004**
			DC	0	1	0	2	0	0	0	0	**0.0004**

of the random test vector set is selected as $10,000$ for the proposed trojan insertion framework.

B. Detection Analysis of Generated HT Benchmarks

To demonstrate the effectiveness of the HT benchmark generated by the proposed trojan insertion framework, we have employed three trojan detection schemes. These trojan detection schemes include random pattern generation, Multiple Excitation Rare Occurrences (MERO) [24] and N-Activation of Rare Events With ATPG (ND-ATPG) [25], explained in Section II. The test patterns generated through these schemes are employed to detect HTs generated by various HT insertion frameworks, including the random, RL based, Trust-Hub, and the proposed framework. The effectiveness of the trojan benchmarks generated by these frameworks against various detection schemes is evaluated using two parameters:

- **Trigger Coverage (TC):** TC refers to the number of HT-infected netlists where the inserted trojan instance is successfully activated. However, its impact remains hidden and is not observable at the output ports.
- **Detection Coverage (DC):** DC refers to the number of HT-infected netlists where the inserted trojan instance is not only activated but also causes an observable effect at the output ports. It should be noted that $TC \subseteq DC$.

It can be observed from Table II that TC and DC vary significantly across trojan benchmarks generated by different HT insertion frameworks for different detection schemes. Notable, the trojan benchmark generated by the proposed framework stands out among all trojan benchmarks by achieving complete evasion of detection across two schemes, specifically random patterns and MERO. Additionally, the detection accuracy using ND-ATPG is negligible, i.e., 0.0004%.

C. HT Insertion Time complexity Analysis

The HT insertion process involves several key steps: levelizing the netlist, identifying rare nodes, identifying trojan insertion points, trojan instance generation, integrating the trojan instance into the netlist, and validating the HT-infected netlist. Table III presents a comparison of trojan insertion times across various circuits from the ISCAS85 and ISCAS89 benchmarks, evaluated using different trojan insertion frameworks. It should be noted that the trojan insertion times (TTs) reported in Table III are based on inserting 100 trojan instances by all HT insertion frameworks. Additionally, the trojan insertion time for the RL based trojan insertion framework is available only for the ISCAS85 benchmark netlists and is directly sourced from [4]. Moreover, no comparison with the Trust-Hub benchmark is provided, as it is not an open-source framework, and therefore, insertion time data for it is unavailable.

TABLE III: Comparison of trojan insertion time (TT) of Various HT Insertion Frameworks

Circuit	Random HT insertion		RL based HT insertion [4]		Proposed HT insertion	
	trigger nodes	TT (min.)	trigger nodes	TT (min.)	trigger nodes	TT (min.)
c2670	10-20	1562	5	69	25-28	0.183
c3540	10-20	13997	5	721	45-56	0.467
c5315	10-20	2293	5	1396	76-78	0.765
c6288	10-20	8240	5	3438	48-68	5.623
s1423	10-20	1588	-	-	39-46	0.139
s13207	10-20	138742	-	-	155-169	0.812
s15850	10-20	25681	-	-	129-148	1.312
s35932	10-20	237786	-	-	100-110	2.067
Average	-	53736.12	-	1406	-	1.421

D. Scalability Analysis of the Proposed Framework

Table IV illustrates the number of complete subgraphs and their generation time for ISCA85 and ISCAS89 benchmark circuits. It is clear from Table IV that an exceptionally large number of complete subgraphs can be retrieved, which facilitates the generation of numerous unique trojan instances. Despite the large number of subgraphs, the generation times are reasonable, demonstrating that the framework can effectively scale with the complexity and size of the circuits.

TABLE IV: Number of complete subgraphs and their generation time for ISCAS85 and ISCAS89 benchmark circuits

Benchmark	ISCAS85				ISCAS89			
Circuits	c2670	c3540	c5315	c6288	s1423	s13207	s15850	s35932
Number of Subgraphs	2000	20042	10000	1000	22093	15000	10000	5000
Generation Time (seconds)	**0.6**	**142.2**	**17.8**	**2.5**	**44**	**135**	**98.8**	**202**

E. Area Overhead Analysis of Generated Trojan Instances

The area overhead introduced by the proposed HT insertion framework was evaluated in terms of the additional area. For synthesis, we have used the GENUS Cadence tool with the 45nm NangateOpenSource Library. It can be observed from Table V that as the size of the original circuit increases, the area overhead reduces significantly. Additionally, the area overhead values reported are for the maximum possible size of the trojan trigger logic, ensuring a worst-case scenario analysis. This analysis provides a comprehensive understanding of the impact of trojan insertion on different circuit sizes, highlighting the scalability and efficiency of the proposed framework.

TABLE V: Area Overhead Analysis for Generated HTs

Benchmark	ISCAS85				ISCAS89			
Circuits	c2670	c3540	c5315	c6288	s1423	s13207	s15850	s35932
Trigger Nodes	28	56	78	68	46	169	148	100
%age Area Overhead	**5.4**	**1.26**	**0.65**	**0.23**	**5.02**	**5.4**	**2.57**	**1.02**

V. Conclusion

In this work, we have presented a novel framework for the insertion of Hardware Trojans (HTs) in digital circuits. The core of our approach involves the generation of complete subgraphs, which is crucial for the creation of unique HT instances. Furthermore, the generated HT-infected netlists exhibit very low detection coverage, making it highly effective for generating stealthy HT benchmarks for the fair evaluation of the new HT detection methodologies.

Acknowledgement

This work has been partially supported by the project numbers IHUB-NTIHAC/2021/01/14, IHUB-NTIHAC/2021/01/15 and Prime Minister's Research Fellowship (PMRF).

REFERENCES

[1] 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.

[2] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[3] B. Shakya, M. T. He, H. Salmani, D. Forte, S. Bhunia, and M. M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, pp. 85–102, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:51996793

[4] A. Sarihi, A. Patooghy, P. Jamieson, and A.-H. A. Badawy, "Trojan playground: a reinforcement learning framework for hardware trojan insertion and detection," *The Journal of Supercomputing*, pp. 1–35, 2024.

[5] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, 2011, pp. 1143–1146.

[6] Y. Gao, J. Su, J. Li, S. Wang, and C. Li, "A neural network framework based on convnext for side-channel hardware trojan detection," *ETRI Journal*.

[7] Y. Lyu and P. Mishra, "Maxsense: Side-channel sensitivity maximization for trojan detection using statistical test patterns," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 3, jan 2021. [Online]. Available: https://doi.org/10.1145/3436820

[8] Y. Huang, S. Bhunia, and P. Mishra, "Mers: Statistical test generation for side-channel analysis based trojan detection," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 130–141. [Online]. Available: https://doi.org/10.1145/2976749.2978396

[9] S. Narasimhan, D. Du, R. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, "Hardware trojan detection by multiple-parameter side-channel analysis," *IEEE Trans. Comput.*, vol. 62, no. 11, p. 2183–2195, nov 2013. [Online]. Available: https://doi.org/10.1109/TC.2012.200

[10] D. Ismari, J. Plusquellic, C. Lamech, S. Bhunia, and F. Saqib, "On detecting delay anomalies introduced by hardware trojans." IEEE Press, 2016, p. 1–7. [Online]. Available: https://doi.org/10.1145/2966986.2967061

[11] S. Chakraborty, M. S. Varenya, A. Mondal, and B. Sen, "Exploring the pso-driven test pattern generation approach for hardware trojan detection," in *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, vol. 2, 2024, pp. 1–6.

[12] A. Mondal, D. Kalita, A. Ghosh, S. Roy, and B. Sen, "Toward the generation of test vectors for the detection of hardware trojan targeting effective switching activity," vol. 19, no. 4, sep 2023. [Online]. Available: https://doi.org/10.1145/3597497

[13] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 408–413.

[14] S. Mulhem, F. Muuss, C. Ewert, R. Buchty, and M. Berekovic, "ML-based trojan classification: Repercussions of toxic boundary nets," *IEEE Embedded Systems Letters*, pp. 1–1, 2023.

[15] C. Dong, Y. Yao, Y. Xu, X. Liu, Y. Wang, H. Zhang, and L. Xu, "A cost-driven method for deep-learning-based hardware trojan detection," *Sensors*, vol. 23, no. 12, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/12/5503

[16] G. M, k. S. Harsha, J. Nikhil, M. S. Eswar, and R. S R, "Hardware trojan detection using supervised machine learning," in *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, 2021, pp. 1451–1456.

[17] S. Kundu, X. Meng, and K. Basu, "Application of machine learning in hardware trojan detection," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 414–419.

[18] Y. Xiang, L. Li, and W. Zhou, "Random forest classifier for hardware trojan detection," in *2019 12th International Symposium on Computational Intelligence and Design (ISCID)*, vol. 1, 2019, pp. 134–137.

[19] C. Dong, J. Chen, W. Guo, and J. Zou, "A machine-learning-based hardware-trojan detection approach for chips in the internet of things," *International Journal of Distributed Sensor Networks*, vol. 15, no. 12, p. 1550147719888098, 2019.

[20] C.-X. Wang, S.-Y. Zhao, X.-S. Wang, M. Luo, and M. Yang, "A neural network trojan detection method based on particle swarm optimization," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2018, pp. 1–3.

[21] S. Chakraborty, A. Ghosh, A. Mondal, and B. Sen, "Test pattern generation for detection of hardware trojans based on improved genetic algorithm," in *2022 IEEE Bombay Section Signature Conference (IBSSC)*, 2022, pp. 1–6.

[22] S. Saha, R. S. Chakraborty, S. S. Nuthakki, Anshul, and D. Mukhopadhyay, "Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 577–596.

[23] X. Mingfu, H. Aiqun, and L. Guyue, "Detecting hardware trojan through heuristic partition and activity driven test pattern generation," 2014.

[24] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A statistical approach for hardware trojan detection," in *Cryptographic Hardware and Embedded Systems - CHES 2009*, C. Clavier and K. Gaj, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 396–410.

[25] A. Jayasena and P. Mishra, "Scalable detection of hardware trojans using ATPG-based activation of rare events," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 4450–4462, 2023.

[26] F. Brglez, "A neutral netlists of 10 combinational circuits and a target translator in fortran," 1985. [Online]. Available: https://api.semanticscholar.org/CorpusID:61388972

[27] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *1989 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1989, pp. 1929–1934 vol.3.

[28] J. Cruz, Y. Huang, P. Mishra, and S. Bhunia, "An automated configurable trojan insertion framework for dynamic trust benchmarks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1598–1603.

[29] Trust-Hub, "Trust-hub: A hardware security benchmarking resource," https://trust-hub.org.

[30] J. Cruz, P. Gaikwad, A. Nair, P. Chakraborty, and S. Bhunia, "Automatic hardware trojan insertion using machine learning," *arXiv preprint arXiv:2204.08580*, 2022.

[31] M. Fyrbiak, S. Wallat, P. Swierczynski, M. Hoffmann, S. Hoppach, M. Wilhelm, T. Weidlich, R. Tessier, and C. Paar, "Hal—the missing piece of the puzzle for hardware reverse engineering, trojan detection and insertion," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 498–510, 2019.

[32] V. Gohil, H. Guo, S. Patnaik, and J. Rajendran, "Attrition: Attacking static hardware trojan detection techniques using reinforcement learning," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1275–1289. [Online]. Available: https://doi.org/10.1145/3548606.3560690

[33] L. Goldstein and E. Thigpen, "SCOAP: Sandia controllability/observability analysis program," in *17th Design Automation Conference*, 1980, pp. 190–196.

[34] RL-ISCAS-85, "Hardware trojan insertion and detection with reinforcement learning: Ht benchmark," https://github.com/NMSU-PEARL/Hardware-Trojan-Insertion-and-Detection-with-Reinforcement-Learning/tree/main/HT_benchmark.

[35] Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Transactions on Computers*, vol. C-30, no. 3, pp. 215–222, 1981.