RareLS: Rarity-Reducing Logic Synthesis for Mitigating Hardware Trojan Threats

Chang Meng, Mingfei Yu
EPFL, Lausanne, Switzerland
{chang.meng,mingfei.yu}@epfl.ch

Hanyu Wang
ETH Zurich, Zurich, Switzerland
hanywang@student.ethz.ch

Wayne Burleson
University of Massachusetts, Amherst, USA
burleson@umass.edu

Giovanni De Micheli
EPFL, Lausanne, Switzerland
giovanni.demicheli@epfl.ch

ABSTRACT

Hardware Trojan (HT) poses a critical security threat to integrated circuits, which can change circuit functionality or leak sensitive data. HTs are typically activated under low-probability conditions by exploiting “rare signals” in logic circuits. In this paper, we propose RareLS, rarity-reducing logic synthesis for mitigating HT threats. Specifically, RareLS reduces the number of rare signals through rarity-oriented technology-independent optimization and technology mapping. Experimental results show that RareLS reduces rare signals by 63.4% on average, with a small overhead of 4.0% in area, 1.9% in delay, and 6.2% in power. Moreover, RareLS complicates HT insertion for attackers by reducing HT trigger logic by 92.94%, and aids defenders in detecting HTs by shortening the test length by more than 80.83%.

KEYWORDS

Rare Signal Reducing, Logic Synthesis, Hardware Trojan Mitigation

1 INTRODUCTION

Hardware Trojan (*HT*) emerges as a crucial security concern in modern integrated circuits [1, 2]. This malicious circuit modification aims to change circuit functionality, leak sensitive data, or cause a denial of service [3]. Fig. 1 depicts a typical HT in a logic circuit, consisting of *trigger logic* and *payload logic* [4, 5]. The trigger logic monitors signals in the original circuit, activating a triggering signal under certain conditions. Upon activation, the payload disrupts normal circuit operations, potentially causing incorrect circuit outputs or even failure.

HTs are usually designed to be stealthy to evade detection, employing a triggering mechanism activated with a low probability [6, 7]. Fig. 1 showcases such a mechanism through a combinational trigger logic using a multiple-input AND gate. Here, each input of the trigger logic is a *rare signal* with signal probability close to 0 or 1, where the signal probability is defined as the probabilities of a signal being logic one. Specifically, two inputs with signal probabilities of 0.01 are directly connected to the AND gate, while another input with a signal probability of 0.99 is first negated to obtain a near-0 signal probability before entering the AND gate. This design ensures that the AND gate’s output, the triggering signal, has a very low signal probability, making the HT hard to detect.

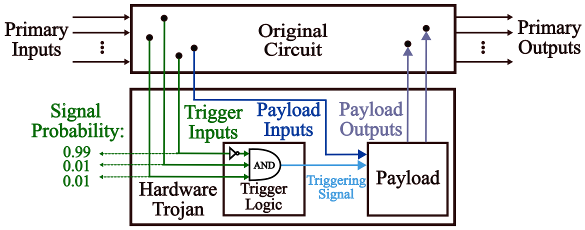


Figure 1: Schematic of a hardware trojan in a logic circuit.

The landscape of HT-related strategies encompasses both attackers’ techniques and defenders’ countermeasures. In what follows, we will review some literature closely related to our work.

From attackers’ perspective, researchers focus on designing stealthy HTs, typically triggered under low-probability conditions. Wang *et al.* [8] exploited low-probability don’t care states to activate HTs. Zhang and Xu [9] aimed to minimize the probability of the trigger condition, thus enhancing the stealthiness of HTs. Haider *et al.* [10] introduced an XOR-LFSR HT, a counter-based HT activated under low-probability conditions. Jain *et al.* [11] presented a HT in an advanced encryption standard core, using sequential trigger logic with a low probability. Cruz *et al.* [12] developed a framework for automated insertion of HTs into gate netlists. Their method identifies rare signals in gate netlists, which are subsequently utilized to construct low-probability trigger conditions. Given that low-probability conditions in logic circuits are often associated with rare signals, reducing rare signals can significantly reduce the number of potential HT triggers, thereby complicating HT insertion.

From defenders’ perspective, extensive research is dedicated to HT detection and prevention. For HT detection, a prominent approach is based on logic testing, which primarily focuses on rare signals in logic circuits. Chakraborty *et al.* [13] introduced the *MERO* method, which generates test patterns through multiple excitations of rare signals. Saha *et al.* [14] combined genetic algorithms with *satisfiability* (*SAT*) solving, and developed an *automatic test pattern generation* (*ATPG*) technique utilizing rare signals. Lyu and Mishra [15] proposed *TARMAC*, an ATPG algorithm based on maximal clique sampling, again leveraging rare signals. Shi *et al.* [16] analyzed the correlation between circuit inputs and rare signals, and applied the genetic algorithm to generate test patterns for HT detection. Pan and Mishra [17] utilized the controllability and observability analysis and signal rarity to improve the efficiency of HT detection, and developed a reinforcement learning-based

This work is supported by the Swiss National Science Foundation Grant “Supercool: Design methods and tools for superconducting electronics” with funding number 200021_1920981 and by Synopsys Inc. Corresponding author: Chang Meng.

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM 979-8-4007-1077-3/24/10…
https://doi.org/10.1145/3676536.3676732

HT detection framework. For HT prevention, many *design-for-trust (*DFTrust*)* techniques are proposed to reduce rare signals in logic circuits. Salmani *et al.* [18] suggested inserting dummy scan flip-flops to eliminate rare signals, albeit with a significant area overhead. Samimi *et al.* [19] used logic encryption to reduce rare signals exploitable by attackers. However, this encryption-based technique induces large hardware overhead in area (31.57%) and delay (56.17%). Recently, Jayasena *et al.* [20] showed a positive correlation between the number of rare signals and the circuit area. They simply applied traditional area-oriented logic synthesis to reduce area, thereby reducing the number of rare signals. However, their method does not directly target the reduction of rare signals, thus not fully exploiting the potential of logic synthesis. In contrast, this paper proposes a novel logic synthesis method tailored for rare signal reduction, which significantly reduces rare signals compared to the approach in [20].

The above discussions illustrate the pivotal role of rare signals in both attacking and defending strategies for HTs. Reducing rare signals limits the possible space for creating HT triggers, thus complicating HT insertion for attackers and simplifying HT detection for defenders. This motivates us to propose RareLS, rarity-reducing logic synthesis for mitigating HT threats. Our main contributions are as follows:

- RareLS explores a new perspective in DFTrust using novel rarity-reducing logic synthesis techniques. Orthogonal to existing DFTrust methods, RareLS can be concurrently applied in conjunction with them.
- We develop a rarity-reducing *And-Inverter Graph* (*AIG*) synthesis technique based on rarity-reducing resubstitution.
- We devise a rarity-reducing technology mapping method that "hides" rare signals within complex gates in standard cell library.
- Experimental results show that RareLS reduces rare signals by an average of 63.4%, with a small overhead of 4.0% in area, 1.9% in delay, and 6.2% in power. Although not eliminating all rare signals, RareLS complicates HT insertion by reducing HT trigger logic by 92.94%, and facilitates HT detection by reducing the test length by more than 80.83%.

The RareLS code is open-source and available at https://github.com/changmg/RareLS.

The rest of the paper is organized as follows. Section 2 introduces preliminaries. Section 3 details the RareLS methodology. Section 4 shows experimental results. Finally, Section 5 concludes the paper and discusses future works.

2 PRELIMINARIES

2.1 Threat Model

The lifecycle of modern supply chains typically includes design, fabrication, testing, and deployment phases [3]. In our threat model, we assume that attackers may be present during the design or fabrication stages, while the testing and deployment stages are trusted. Attackers are assumed to have the ability to insert HTs by changing the gate netlist. This threat model is consistent with those adopted in existing DFTrust methods [18–20]. Possible attacking scenarios under this model are as follows:

- In the design stage, a rogue employee of an IC design company may gain access to the gate netlist format of a design and insert HTs into it.
- In the design stage, a malicious computer-aided design tool could automatically integrate HTs into the gate netlist.
- In the fabrication stage, an untrustworthy foundry may insert HTs by changing the gate netlist.

The inserted HTs are assumed to be stealthy and can remain hidden during traditional functional validation and testing. A typical way of inserting such HTs involves combining several rare signals to serve as the HT trigger.

2.2 Logic Circuit and Rare Signal

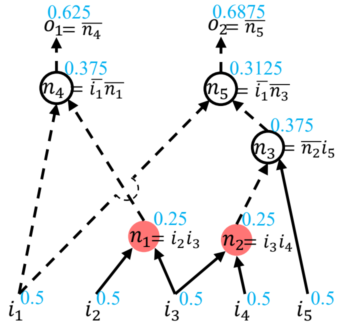


Figure 2: An AIG with each node denoting an AND gate. Solid edges indicate direct connections, and dashed edges denote signal negations. The blue value associated with each node denotes the signal probability of that node. If the rarity threshold is 0.3, then n_1 and n_5 are rare signals.

A logic circuit can be modeled as a directed acyclic graph with nodes and edges. In such circuits, each node produces a signal, and throughout this paper, the terms "signal" and "node" are used interchangeably. A *primary input* (*PI*) is a node without incoming edges, and a *primary output* (*PO*) is a node without outgoing edges. Nodes that are neither PI nor PO are *internal nodes*. A *gate netlist* is a special logic circuit, consisting of nodes representing logic gates interconnected by edges.

An *AND-Inverter Graph* (*AIG*) [21] is an abstract representation of a logic circuit, extensively used in modern logic synthesis methods [22, 23]. Each internal node in an AIG denotes an AND gate, and each edge can be optionally marked to represent signal negations. An example AIG is shown in Fig. 2, where nodes $n_1 \sim n_5$ denote AND gates, solid edges indicate direct connections without negations, and dashed edges indicate signal negations. In an AIG, a *cut* of a node n is defined as a set of nodes, called *leaves*, such that any path from a PI to node n passes at least one leaf [24]. For example, in Fig. 2, n_3's cuts include $\{n_2, i_5\}$ and $\{i_3, i_4, i_5\}$.

Signal probability is the probability of a signal being in a "high" (logic one) state. For example, in Fig. 2, each node is associated with a signal probability, denoted by the blue value. In a gate netlist or an AIG, a *rare signal* is defined as a signal whose signal probability is either below a threshold δ or above $1 - \delta$. We call δ the *rarity threshold*. In Fig. 2, if $\delta = 0.3$, then n_1 and n_5 are rare signals.

3 RARELS METHODOLOGY

This section details the RareLS methodology. We first overview RareLS and then delve into the two key techniques: rarity-reducing technology mapping and rarity-reducing AIG synthesis.

3.1 Overview

RareLS aims to reduce rare signals in a combinational gate netlist, thus effectively mitigating HT threats. The overall flow of RareLS is depicted in Fig. 3. It begins with an initial gate netlist $GNET_{init}$ and a user-defined rarity threshold δ. Then, RareLS proceeds with the three steps:

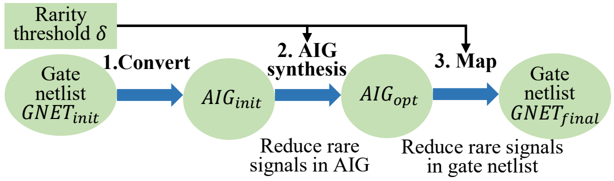

Rarity threshold δ
1.Convert
2. AIG synthesis
3. Map
Gate netlist $GNET_{init}$
AIG_{init}
AIG_{opt}
Gate netlist $GNET_{final}$
Reduce rare signals in AIG
Reduce rare signals in gate netlist

Figure 3: Overall flow of RareLS.

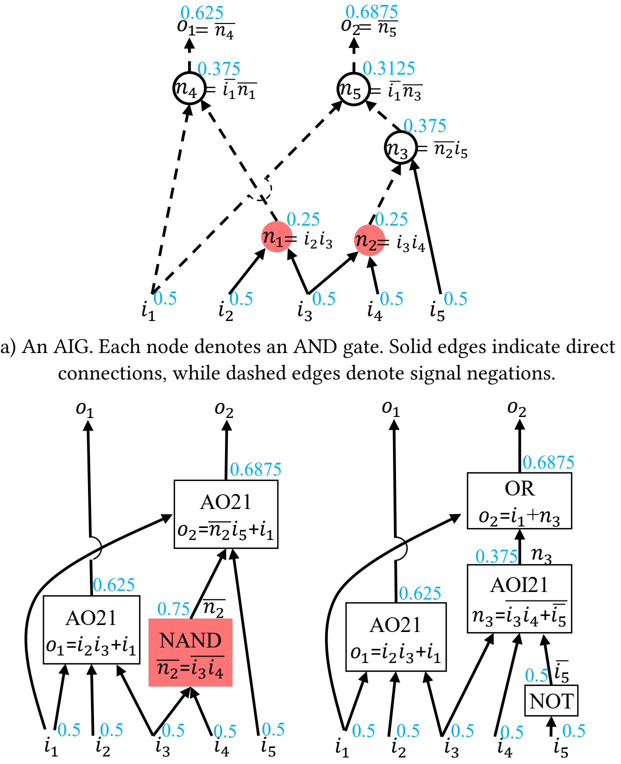
1. Conversion: Convert $GNET_{init}$ into an AIG, denoted as AIG_{init}[1].
2. Rarity-reducing AIG synthesis: Reduce rare signals in AIG_{init} using a novel rarity-oriented resubstitution technique, thereby facilitating the rare signal reduction in the final gate netlist. The optimized AIG is denoted as AIG_{opt}.
3. Rarity-reducing technology mapping: Map AIG_{opt} into the final gate netlist $GNET_{final}$ using a technology mapping method tailored for reducing rare signals.

We should emphasize that RareLS cannot guarantee the complete elimination of all rare signals, but it can significantly reduce the number of rare signals in the final gate netlist. Utilizing RareLS can complicate HT insertion and facilitate testing-based HT detection, which is demonstrated in our experimental part.

The following subsections will introduce the detailed steps. Since Step 2 relies on an important concept proposed in Step 3, we will first introduce Step 3 and then Step 2.

3.2 Step 3: Rarity-Reducing Technology Mapping

This subsection describes how to map AIG_{opt} into the final gate netlist $GNET_{final}$, with a focus on reducing rare signals. In what follows, we will first propose the concept of *"hiding" rare signals* and then present our rarity-reducing technology mapping method based on the concept.



(a) An AIG. Each node denotes an AND gate. Solid edges indicate direct connections, while dashed edges denote signal negations.

(b) Gate netlist produced by area-oriented traditional technology mapping, where $\overline{n_2}$ is a rare signal. Area = $3.99\mu m^2$.

(c) Gate netlist obtained by signal-hiding technology mapping, where n_1 and n_2 are hidden into AO21 and AOI21 gates. Area = $4.24\mu m^2$.

Figure 4: Comparative mapping of an AIG into gate netlists using traditional and rarity-reducing technology mapping methods. Signal probabilities are shown in blue, with rare signals marked in red under a rarity threshold of $\delta = 0.3$.

3.2.1 Hiding Rare Signals. Consider the AIG shown in Fig. 4(a), with a rarity threshold $\delta = 0.3$. Assuming uniform input distribution, the signal probability of each node is calculated and displayed in blue. Nodes n_1 and n_2, each exhibiting a signal probability of 0.25, are identified as rare signals and marked in red. Using a traditional area-oriented technology mapping method with a Nangate 45nm library [25], the resulting gate netlist is shown in Fig. 4(b). This netlist includes two *AND-OR-21* (*AO21*) gates and a NAND gate, with an area of $3.99\mu m^2$. Notably, the rare signal n_1 is hidden within the left AO21 gate after mapping. However, the rare signal $\overline{n_2}$ remains visible as the NAND gate's output.

Using an alternative way of mapping, rare signals n_1 and n_2 can be simultaneously hidden. As depicted in Fig. 4(c), these signals are hidden within the AO21 and *AND-OR-Inverter-21* (*AOI21*) gates, making them undetectable at the gate netlist level. While this netlist has a larger area of $4.24\mu m^2$, it successfully hides the rare signals within complex gates like AO21 and AOI21. This adds an additional layer of complexity for accessing rare signals, increasing the difficulty for attackers to detect these signals in the gate netlist. Hiding rare signals compels attackers to perform more intensive analysis to uncover the rare signals. Moreover, our strategy of hiding rare signals can work in parallel to existing DFTrust methods, such as those in [18, 19], which provides a new perspective of DFTrust.

3.2.2 Rarity-Reducing Technology Mapping Based on Hiding Rare Signals. Our proposed method enhances the generic technology mapping algorithm [24], shown in Algorithm 1, to prioritize rare signal reduction. We will first describe the generic algorithm and then detail our enhancement.

[1] RareLS works on AIGs, since AIG-based modern logic synthesis has shown significant advantages in reducing hardware cost, particularly for CMOS technologies [22, 23].

Algorithm 1: Generic technology mapping.

```
   Input: $AIG$, standard cell library $\mathcal{L}$, and optimization metric $Metr$.
   Output: Gate netlist $GNET$
1  foreach node $n \in AIG$ in topological order do
2      $n.bestcut \leftarrow \emptyset$, $n.bestscore \leftarrow +\infty$;
       // Evaluate the score for each cut of node $n$
3      foreach cut $C$ of node $n$ do
4          $F_C \leftarrow$ logic function represented by $C$;
5          if there exists a gate $g \in \mathcal{L}$ implementing $F_C$ then
6              $C.score \leftarrow ComputeScore(AIG, C, g, Metr)$;
7              if $C.score < n.bestscore$ then
8                  $n.bestcut \leftarrow C$, $n.bestscore \leftarrow C.score$;
9  Build $GNET$ from $AIG$ using the best cuts of nodes;
10 return $GNET$
```

The generic technology mapping algorithm transforms an AIG into a gate netlist *GNET* using a standard cell library $\mathcal{L}$, guided by an optimization metric *Metr*. Traditional metrics prioritize area, delay, or power, but do not consider signal rarity. Algorithm 1 processes each AIG node n in topological order (Line 1). For each node n, it initializes n's best cut and score (Line 2), and then enumerates n's cuts (Line 3). For each cut C of node n, the algorithm determines the logic function F_C represented by C (Line 4). If a gate in $\mathcal{L}$ can implement F_C (Line 5), then the algorithm computes a score for C based on the selected metric *Metr* (Line 6). If the score is smaller (*i.e.*, leading to a lower cost) than the current best score, then n's best cut and score are updated (Lines 7-8). After evaluating all nodes, a gate netlist *GNET* is constructed using the best cuts (Line 9) and then returned.

For traditional area-oriented technology mapping, a common score is the effective area [26], calculated as $C.score_{Area} = \left[Area(g) + \sum_i (Leaf_i.bestscore_{Area})\right] / NumFanouts(n)$. In this formula, $Area(g)$ is gate g's area, $Leaf_i$ is the i-th leaf of the cut C, $Leaf_i.bestscore_{Area}$ is the best effective area at that leaf, and $NumFanouts(n)$ is the count of n's fanouts. The effective area of a cut provides a good estimation of the area contribution of the cut to the resulting gate netlist.

We enhance Algorithm 1 to reduce rare signals in the final gate netlist by hiding rare signals, while ensuring a good quality in area, delay, or power. To hide rare signals during technology mapping, we focus on the *rare cut* in an AIG, defined as a cut containing at least one rare signal. For example, in Fig. 4(a), node o_2 has a rare cut $\{n_2, i_1, i_5\}$. During technology mapping, selecting a rare cut as the best cut for node n will result in the rare signal in the cut exposed in the final gate netlist. For example, in Fig. 4(a), if o_2's best cut is determined to be the rare cut $\{n_2, i_1, i_5\}$, o_2 is then mapped into an AO21 gate with n_2, i_1, and i_5 as inputs, as depicted in Fig. 4(b). Consequently, the rare signal n_2 remains visible after mapping.

Based on this observation, we propose a rarity-oriented scoring mechanism that penalizes rare cuts during the cut evaluation process (Algorithm 1 Lines 3–8). This scoring mechanism is integrated into the computing score step of Algorithm 1 (Line 6). To reduce rare signals while achieving good area, the rarity-oriented score for a cut C of node n is defined as:

$$C.score_{Rarity} = \begin{cases} C.score_{Area} + Penalty, & \text{if } C \text{ is a rare cut} \\ C.score_{Area}, & \text{otherwise,} \end{cases} \tag{1}$$

where *Penalty* is an extremely large constant (set as 10^7 in our implementation). With Eq. (1), if C is a rare cut, its score is extremely high due to the penalty, which deters its selection as n's best cut and prohibits the rare signal within C being exposed in *GNET*. Conversely, if C is a non-rare cut, then C's score equals $C.score_{area}$, promoting the cut for the area optimization. Consequently, if n has some non-rare cuts, then the non-rare cut with the smallest $C.score_{area}$ is determined as n's best cut. For example, in Fig. 4(a), after evaluating all o_2's cuts, the best non-rare cut is $\{i_1, n_3\}$, resulting in a mapping with an OR gate shown in Fig. 4(c). This scoring mechanism reduces rare signals while achieving a good circuit area. The penalty-based approach like Eq. (1) can be similarly applied to enhance delay-oriented mapping and power-oriented mapping.

3.3 Step 2: Rarity-Reducing AIG Synthesis

This subsection presents rarity-reducing AIG synthesis. We first propose a theoretical foundation for the AIG synthesis process, and then detail our rarity-reducing resubstitution technique.

3.3.1 Theoretical Foundation for Rarity-Reducing AIG Synthesis. In the previous subsection, we introduce rarity-reducing technology mapping, which hides rare signals within complex logic gates in the final gate netlist. This raises a pivotal question: *which signals can be hidden and which cannot.* To address this, we propose a sufficient condition that serves as a guideline for rarity-reducing AIG synthesis.

Theorem 1. *Assume that the standard cell library includes a NOT gate, as well as 3-input AOI21, OAI21, NAND3, and NOR3 gates*[2]. *An internal rare signal n in an AIG, which is neither a PI nor a PO, can be hidden using one of the 3-input gates if it follows the following two conditions: 1) All fanins and fanouts of n are non-rare signals. 2) For each fanout of n, denoted as v, n is the only rare signal among v's fanins.*

Proof. Let p and q be the fanins of the non-PI signal n. Signal n can be hidden within a gate during mapping provided that, for any n's fanout signal v, there exists a 3-input gate in the standard cell library that implements v with inputs $\{m, p, q\}$, where m is another fanin of v besides n. Note that the library with the specified gates suffices to implement any 3-input function that may arise in an AIG. Thus, to complete the proof, it suffices to show that the exposed signals v, m, p, q are all non-rare. This holds because criterion 1) ensures v, p, q are non-rare, and criterion 2) ensures m is non-rare. □

We define an *inherently rare signal* as a rare signal that does not satisfy the conditions of Theorem 1. For example, in the left-hand AIG of Fig. 5, if the rarity threshold is 0.3, then n_4 is a rare signal with a signal probability of 0.125. Meanwhile, n_4 is also an inherent rare signal, since it has a rare fanin signal n_1, violating condition 1) of Theorem 1. Note that the number of inherently rare signals is an upper bound for the number of rare signals present in the final gate netlist after applying rarity-reducing technology mapping. Based on this, we propose a rarity-reducing resubstitution

[2] Gate functions: AOI21, $f_{AOI21}(a,b,c) = \overline{ab+c}$; OAI21, $f_{OAI21}(a,b,c) = \overline{(a+b)c}$; NAND3, $f_{NAND3}(a,b,c) = \overline{abc}$; NOR3, $f_{NOR3}(a,b,c) = \overline{a+b+c}$. Modern cell libraries usually have these large cells to achieve better power, performance, and area.

technique, which focuses on reducing inherently rare signals in an AIG.

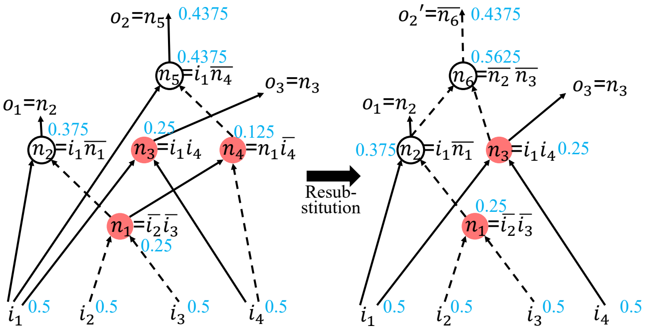

o_2=n_5 0.4375
0.4375
n_5=$i_1\overline{n_4}$
o_3=n_3
o_1=n_2
0.375
n_2=$i_1\overline{n_1}$
0.25
n_3=$i_1 i_4$
0.125
n_4=$n_1\overline{i_4}$
n_1=$\overline{i_2}\,\overline{i_3}$
0.25
i_1 0.5
i_2 0.5
i_3 0.5
i_4 0.5
Resub-
stitution
o_2'=$\overline{n_6}$ 0.4375
0.5625
n_6=$\overline{n_2}\,\overline{n_3}$
o_3=n_3
o_1=n_2
0.375 n_2=$i_1\overline{n_1}$
n_3=$i_1 i_4$ 0.25
0.25
n_1=$\overline{i_2}\,\overline{i_3}$
i_1 0.5
i_2 0.5
i_3 0.5
i_4 0.5

Figure 5: Example of AIG resubstitution [27]. Signal probabilities are shown in blue, with rare signals marked in red under a rarity threshold of $\delta = 0.3$.

3.3.2 Rarity-Reducing Resubstitution. Resubstitution [22, 27] is a powerful technique to simplify a logic circuit in conventional logic synthesis. The AIG resubstitution re-expresses a signal using a set of other signals (also called *divisors*) in the AIG. For example, in the left-hand AIG of Fig. 5, the signal o_2 has the function $o_2 = n_5 = i_1\overline{n_4}$. This function can be re-expressed using signals n_2 and n_3 with a new function $o_2' = \overline{n_6} = \overline{\overline{n_2} \cdot \overline{n_3}} = n_2 + n_3$. After re-expressing o_2 with o_2', the circuit functionality does not change, because o_2 and o_2' are functional equivalent[3]. We can also remove the inherent rare signal n_4 after the resubstitution. The resulting AIG is depicted on the right side of Fig. 5. This example shows that resubstitution can reduce rarity in an AIG. Next, we will introduce a systematic resubstitution approach to reduce the number of inherent rare signals.

We first introduce a generic resubstitution algorithm, as shown in Algorithm 2, and then discuss how to customize it for rarity reduction. Algorithm 2 simplifies an initial AIG based on an optimization metric *Metr*. In traditional logic synthesis, typical metrics can be AIG size for area optimization and AIG depth for delay optimization. The algorithm iteratively traverses each node in the AIG (Line 1). For each node, it collects a candidate divisor set S (Line 2). For each divisor set $s \subseteq S$ (Line 3), it first tries to resubstitute n using s (Line 4). If we can build a new function in terms of signals in s to re-express n's function, then the resubstitution is feasible (Line 5). The term *gain* refers to the difference in metrics of the circuit before and after the resubstitution (Line 6). Algorithms 2 accepts a resubstitution if it results in a positive gain (Lines 7–8). After traversing each node and attempting to resubstitute each node once, the optimized AIG is returned (Line 10).

To customize resubstitution for rarity reduction, in this work, we use rarity as the optimization metric (*Metr* in Algorithm 2). In concrete, for each candidate resubsitution, we define its gain (in Algorithm 2 Line 6) as the number of reduced inherently rare signals after applying the resubstitution. In this way, the resubstitution process directly reduces the inherent rare signals, which further aids the rarity-reducing technology mapping by increasing the opportunities for hiding rare signals according to Theorem 1.

Algorithm 2: Generic AIG resubstitution algorithm.

```
   Input: Initial AIG and optimization metric Metr
   Output: Optimized AIG_opt
1  foreach node n in AIG do
2      S ← collect candidate divisor sets for n;
3      foreach divisor set s⊆S do
4          Try to resubstitute node n using s;
5          if the resubstitution is feasible then
6              gain ← resubstitution gain based on Metr;
7              if gain > 0 then
8                  Apply the resubstitution;
9                  break;
10 return AIG_opt;
```

To efficiently estimate the signal probabilities and evaluate the node rarity during AIG resubstitution, we propose an on-the-fly logic simulation strategy. Initially, we generate random simulation patterns on the PIs. Then, we perform the resubsitution process in a topological order (*i.e.*, Algorithm 2 Line 1 traverses each node n topologically). For each node n, before applying any resubstitution, its simulation pattern can be easily updated based on its fanins' simulation pattern, given that the fanins' simulation patterns have been processed before n's. According to n's simulation patterns, we can estimate its signal probability and determine whether it is a rare signal or not. If a resubstitution is applied to n (*i.e.*, Algorithm 2 Line 8 is executed), then n's simulation pattern will be further updated based on the divisors' simulation patterns, and n's signal probability and rarity will be recalculated accrodingly. In this way, the simulation complexity is linear to the AIG size, making our rarity-reducing resubsitution efficient.

Furthermore, modern logic synthesis flows usually integrate resubstitution with other simplification techniques, such as balancing, rewriting, and refactoring, to reduce more area, delay, and power [22]. Referring to this, our rarity-reducing AIG synthesis flow also synergizes the rarity-reducing resubstitution with these simplification techniques to reduce rare signals while optimizing for area, delay, or power. Moreover, we find that iteratively executing our rarity-reducing AIG synthesis flow for several rounds usually leads to fewer inherent rare signals than a single execution. In our implementation, we apply the rarity-reducing AIG synthesis flow three times, based on the observation that the reduction of inherent rare signals typically plateau after three iterations.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

RareLS, developed in C++, is tested on a single-core AMD 7945HX processor with 64GB RAM. Within RareLS, we use the generic resubstitution framework proposed in [28] as the basis to implement our rarity-reducing AIG synthesis. We modify the area-oriented technology mapping method, *amap*, from a leading logic synthesis system, *abc* [29], to implement our rarity-reducing technology mapping approach. The Nangate 45nm open-cell library [25] is applied for technology mapping. Circuit area, delay, and power are measured by Synopsys Design Compiler [30], with power measurement conducted at 100MHz under a uniform input distribution. The

[3] $o_2 = i_1\overline{n_4} = i_1(\overline{n_1} + i_4)$, and $o_2' = n_2 + n_3 = i_1\overline{n_1} + i_1 i_4 = i_1(\overline{n_1} + i_4)$.

Table 1: Rare signal reduction versus hardware cost overhead caused by RareLS. Rarity threshold $\delta = 0.1$, which is a common choice in [14, 15, 18–20]. A negative hardware cost overhead means that RareLS can further reduce area, delay, or power. Bold entries indicate a reduction in area, delay, power, or rare signals. The input gate netlists are pre-processed using the rare signal reduction technique proposed in [20].

Name	Input gate netlist (pre-processed by [20])					Final rarity-reduced gate netlist after RareLS					Final gate netlist vs initial one			
	#Gate	Area/μm^2	Delay/ns	Power/μW	#Rare	Area/μm^2	Delay/ns	Power/μW	#Rare	Time/s	Area overhead	Delay overhead	Power overhead	Rare reduction
c499	216	253.2	0.7	2.2E+02	56	**239.4**	**0.7**	2.8E+02	**32**	3	**-5.5%**	**-5.6%**	26.2%	**42.9%**
c880	173	182.2	1.1	7.3E+01	17	187.3	1.2	7.5E+01	**9**	1	2.8%	3.5%	3.1%	**47.1%**
c1908	157	173.2	1.1	1.7E+02	21	173.2	**1.1**	1.7E+02	**9**	1	0.0%	**-7.0%**	1.1%	**57.1%**
c2670	300	317.1	1.1	2.1E+02	18	328.2	**1.0**	2.2E+02	**11**	2	3.5%	**-13.3%**	3.3%	**38.9%**
c3540	464	490.0	1.8	3.4E+02	84	511.3	1.8	3.7E+02	**32**	4	4.3%	0.6%	7.8%	**61.9%**
c5315	637	724.6	1.2	4.5E+02	20	726.7	**1.1**	4.7E+02	**10**	4	0.3%	**-5.1%**	4.6%	**50.0%**
c6288	1862	1497.0	4.5	1.3E+03	28	1523.1	**4.5**	1.3E+03	**0**	10	1.7%	**-0.9%**	0.8%	**100.0%**
c7552	778	841.9	2.8	6.8E+02	54	869.3	**2.8**	7.1E+02	**19**	6	3.3%	**-2.8%**	4.0%	**64.8%**
sin	2955	3200.5	8.8	4.3E+03	746	3879.9	11.9	5.5E+03	**260**	389	21.2%	35.1%	29.1%	**65.1%**
max	1682	1662.5	19.2	2.2E+03	15	1672.1	20.3	**2.0E+03**	**2**	8	0.6%	5.7%	**-10.9%**	**86.7%**
square	11508	11139.8	10.7	5.8E+03	1086	11566.7	10.8	6.2E+03	**26**	81	3.8%	1.3%	7.8%	**97.6%**
mult64	13639	14581.6	12.9	2.0E+04	24	14775.8	**12.2**	**2.0E+04**	**1**	172	1.3%	**-5.3%**	**-1.5%**	**95.8%**
log2	15127	17524.3	20.0	2.8E+04	2884	20816.6	21.9	3.2E+04	**2077**	1377	18.8%	9.4%	13.1%	**28.0%**
div	11611	12728.1	222.2	1.4E+04	307	12828.4	249.0	1.5E+04	**236**	79	0.8%	12.1%	2.8%	**23.1%**
hyp	125953	135549.3	1916.8	5.3E+05	7452	138889.2	1920.0	5.4E+05	**557**	823	2.5%	0.2%	1.5%	**92.5%**
Average										197	4.0%	1.9%	6.2%	**63.4%**

rarity of signals is evaluated using Monte Carlo simulation with 2^{20} input patterns under a uniform input distribution.

The input gate netlists of RareLS are shown in Table 1, where the first 5 columns show each gate netlist's name, gate count, area, delay, and power. These input gate netlists are derived from the ISCAS85 [31] and EPFL [32] benchmark suites. For each benchmark, we pre-process it by reducing its rare signals using the area optimization technique proposed in [20], since [20] shows a positive correlation between the number of rare signals and the circuit area. Specifically, we apply *abc*'s *compress2rs* for area-oriented logic synthesis command 3 times, followed by *amap* for area-oriented technology mapping. The pre-processed gate netlists are then fed into RareLS for further rare signal reduction.

4.2 Rare Signal Reduction versus Hardware Cost Overhead

This experiment shows the tradeoff between rare signal reduction and hardware cost overhead achieved by RareLS. Unless otherwise noted, the rarity threshold δ is set to 0.1. Actually, the selection of δ depends on the specific application and the required security level. We choose $\delta = 0.1$, since it is a common choice of rarity threshold in previous works [14, 15, 18–20]. The number of rare signals for the initial gate netlists is shown in the 6th column of Table 1. Note that Table 1 does not report three EPFL benchmarks, *add128*, *barshift*, and *sqrt*, since there are no rare signals in their input gate netlists after pre-processing when $\delta = 0.1$.

We run RareLS on the input gate netlists, whose rare signals have already been reduced by the approach in [20]. Table 1 shows that after performing RareLS, the number of rare signals is further reduced by 63.4% on average, with an average overhead of 4.0% in area, 1.9% in delay, and 6.2% in power. The increase in hardware cost is because, during rarity-reducing AIG synthesis, RareLS needs to add some nodes to reduce rare signals, and during rarity-reducing mapping, using large logic gates to hide rare signals usually increases the hardware cost. Notably, RareLS dramatically reduces rare signals across all benchmarks, completely or nearly eliminating them in circuits like *c6288*, *max*, and *mult64*. Although RareLS cannot eliminate all rare signals, it can increase the difficulty of inserting HTs into the circuits (as shown in Section 4.4), and facilitate the testing-based detection of HTs (as shown in Section 4.5).

It is interesting that on many benchmarks, Table 1 also reports negative hardware cost overhead achieved by RareLS. This means that RareLS can further reduce area, delay, or power. For example, RareLS can further reduce area for *c499*, delay for circuits like *c499*, *c1908*, and *c2670*, and power for *max* and *mult64*. A possible reason is that RareLS adds some nodes into the circuit to reduce rarity, which changes the circuit structure and provides new opportunities for optimizing area, delay, and power. However, relatively large overhead in area and power are observed in circuits like *sin* and *log2*. A possible reason is that these circuits have a large proportion of inherent rare signals in their AIGs, which requires the AIG synthesis process to add more nodes to eliminate them, causing a large overhead. As for runtime, RareLS exhibits an average runtime of 197 seconds. Notably, RareLS can process the *hyp* benchmark with 125953 gates in 823 seconds. The major runtime overhead lies in logic simulation with 2^{20} input patterns for computing the signal rarity.

Moreover, the hardware overhead of RareLS is significantly lower than existing DFTrust techniques. For instance, for the ISCAS benchmark suite, the logic encryption-based approach proposed in [19] reduces 83.17% rare signals on average, with an overhead of 31.57% in area and 56.17% in delay. Notably, RareLS is compatible with existing HT defense techniques like the ones in [13–15, 18–20]. For example, we can first reduce the rare signals with RareLS, which introduces small hardware overhead, and then apply other DFTrust techniques to further eliminate the remaining rare signals.

4.3 Effects of Different Rarity Thresholds

This experiment shows the effects of different rarity thresholds, δ, on various performance metrics with several ISCAS circuits, *i.e.*, *c1908*, *c2670*, *c5315*, and *c6288*. The results are plotted in Fig. 6, which demonstrates the area overhead, delay overhead, power overhead, and rare signal reduction achieved by RareLS at thresholds of δ=0.05, 0.10, 0.15, and 0.20.

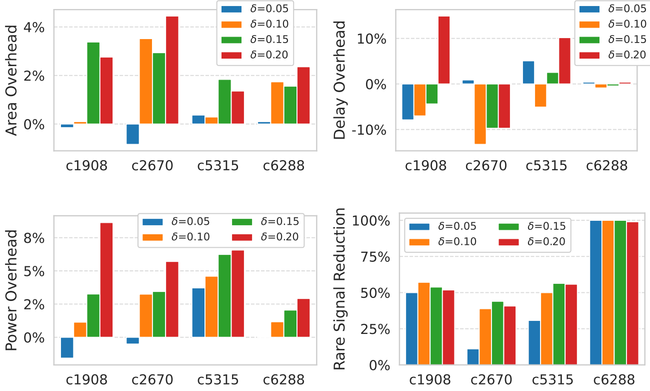


Figure 6: Area overhead, delay overhead, power overhead, and rare signal reduction versus different rarity thresholds δ for various circuits.

The area and power overhead sub-figures reveal an increasing trend in area and power as the δ increases. This is because larger δ corresponds to more rare signals, which require more redundant logic to remove them and a larger number of complex gates to hide them. However, the delay overhead trends vary. As δ increases, the delay increases for *c1908*, while for other circuits, the delay sometimes decreases as δ increases.

4.4 Case Study 1: Complicating HT Insertion

This experiment shows that RareLS can complicate HT insertion. We test an automatic HT insertion framework proposed in [12] and show that RareLS can make this framework less effective.

In [12], it is assumed that an attacker aims to insert a stealthy HT that is hard to detect. To do this, the attacker usually constructs a trigger logic by selecting several rare signals as trigger inputs. Such trigger logic, termed *potential trigger logic*, is not necessarily valid. A trigger logic is *valid* if the corresponding rare conditions at the trigger inputs can happen simultaneously. Cruz *et al.* [12] propose to create a valid trigger logic for HT insertion through the two steps: 1) Randomly selecting rare signals to form a *potential trigger logic*; 2) Checking the validity of the potential trigger logic and determining valid trigger logic.

Utilizing this method, we investigate the number of valid 3-input trigger logic (*i.e.*, trigger logic using 3 rare signals as the trigger inputs) in the initial gate netlists and the ones optimized by RareLS. In this experiment, we assume each trigger logic consists of three rare signals. This aims to precisely figuring out a complete set of valid trigger logic in a gate netlist, to provide a reliable analysis. The following observations and conclusions apply regardless of the targeted number of trigger inputs.

Table 2 shows the numbers of potential trigger logic (#Potential) and valid trigger logic (#Valid) in both the initial gate netlists of several ISCAS benchmarks and the ones optimized by RareLS. We can see that the number of valid trigger logic is significantly reduced by 92.94% on average after applying RareLS. This is because RareLS reduces the number of rare signals (as shown in Table 1), which directly reduces the number of potential trigger logic, leading to a reduction in valid trigger logic. The result on *c499* offers an interesting example: although only part of rare signals is removed by RareLS, the resulting gate netlist has no valid trigger logic. In other words, under our experimental settings, RareLS can prevent all HTs with 3-input trigger logic, which are created by the method in [12], on the benchmark *c499*.

Table 2: Comparison of the amounts of three-input valid trigger logic in gate netlists with and w/o RareLS applied.

Name	Initial (pre-processed by [20])			RareLS-optimized			#Valid
	#Rare	#Potential	#Valid	#Rare	#Potential	#Valid	reduction
c499	56	27 720	56	32	4 960	0	100.00%
c880	17	680	657	9	84	76	88.43%
c1908	21	1 330	1 330	9	84	66	95.04%
c2670	18	816	816	11	165	165	79.78%
c3540	84	95 284	63 420	32	4 960	4 108	93.52%
c5315	20	1 140	866	10	120	77	91.11%
c6288	28	3 276	3 201	0	0	0	100.00%
c7552	54	24 804	21 495	19	969	944	95.61%
Average							92.94%

Although RareLS cannot prevent all HTs, the considerable reduction in rare signals achieved by RareLS means a significant reduction of potential trigger logic. Note that the scalability of existing logic testing-based HT detection techniques is typically bounded by the number of potential trigger logic in circuits [13–15]. Since there is less room for HT insertion, RareLS can improve the efficiency of these HT detection methods, as shown in Section 4.5.

4.5 Case Study 2: Facilitating Testing-Based HT Detection

This experiment demonstrates the efficacy of RareLS in significantly reducing the required test vector length for achieving comprehensive coverage during the logic testing phase, thereby facilitating the detection of HT through testing methodologies.

We utilize the *TRIT-TC* benchmark suite, which is a suite for assessing rare signal-based HT insertion in *TrustHub* [33]. The suite comprises four combinational and four sequential benchmarks sourced from the ISCAS benchmark suite. Additionally, we expand our evaluation to include other combinational circuits from the ISCAS benchmark suite, as examined in prior experiments. We transform sequential circuits into combinational equivalents using the *comb* command in *abc*, wherein storage elements are eliminated by treating their inputs and outputs as additional primary inputs and primary outputs, respectively.

Given a target trigger input size and utilizing gate netlist either optimized with RareLS or without, we generate up to 1 000 000 potential trigger logic by sampling combinations of rare signals. Following a validation process outlined in Section 4.4, we collect the first 1 000 valid trigger logic encountered. Note that the actual number of trigger logic generated for a gate netlist may be fewer than 1 000 due to the limited presence of rare signals or valid trigger logic within the netlist. Subsequently, employing *TARMAC* [15], a state-of-the-art test vector generation tool for HT detection, we

Table 3: Test lengths to achieve full (100%) coverage of the valid trigger logic generated for gate netlists with and w/o RareLS applied.

Name	#Input	Initial (pre-processed by [20])		RareLS		#Test reduction	Name	#Input	Initial (pre-processed by [20])		RareLS		#Test reduction
		#Rare	#Test	#Rare	#Test				#Rare	#Test	#Rare	#Test	
c880	10	17	15	9	1	93.33%	c6288	10	28	7	0	0	100.00%
	11		8		1	87.50%		11		2		0	100.00%
	12		19		1	94.74%		12		3		0	100.00%
c1908	10	21	1	9	0	100.00%	c7552	10	54	9 896	19	64	99.35%
	11		1		0	100.00%		11		26 777		55	99.79%
	12		1		0	100.00%		12		23 800		16	99.93%
c2670	10	18	1	11	1	0.00%	s13207	10	489	>393 216 (99.9%†)	296	93054	>76.34%
	11		1		1	0.00%		11		>1 048 576 (99.7%†)		406 194	>61.26%
	12		1		1	0.00%		12		>1 048 576 (99.9%†)		393 373	>62.39%
c3540	10	84	20 416	32	1 024	94.98%	s1423	10	42	8	8	0	100.00%
	11		26 566		4 482	83.13%		11		11		0	100.00%
	12		60 054		2 326	96.13%		12		38		0	100.00%
c5315	10	20	64	10	1	98.44%	s15850	10	449	>1 048 576 (99.7%†)	186	384 501	>63.33%
	11		58		1	98.28%		11		>1 048 576 (99.7%†)		384 501	>64.70%
	12		64		1	98.44%		12		>1 048 576 (99.1%†)		491 983	>53.08%
Average of all benchmarks													>80.83%

† Coverage achieved at the reported lower bound of test length.

figure out the test length to achieve a 100% coverage of the 1 000 collected valid trigger logic for each benchmark. For consistency, we maintain a rarity threshold of $\delta = 0.1$ in this experiment.

For each benchmark, we examine scenarios where the number of trigger inputs (#Input) ranges from 10 to 12, in alignment with the specifications of *TRIT-TC*. It is noteworthy that the target trigger input size is upper-bounded by the count of rare signals present in the netlist (#Rare). This occurs several times in netlists optimized by RareLS, as the presence of rare signals within the netlist has been substantially diminished. For instance, in the case of *c880*, RareLS effectively decreased the count of rare signals to 9, resulting in a saturated trigger input size of 9, irrespective of the specification.

Table 3 reports a significant reduction in test length (#Test) required for full (100%) coverage of the generated trigger logic, achieved by applying RareLS to optimize the gate netlists. Benchmarks *c499* and *s35932* are omitted from the table as no valid trigger logic exists before applying RareLS, indicating their inherent resistance to HT insertion, thus obviating the need for applying RareLS. In cases where *TARMAC* was unable to determine a minimum test length to cover all generated trigger logic within the allocated time budget (120 hours), lower bounds of the minimums are determined and the coverages achieved at those lower bounds are reported.

Test length serves as a metric for the efforts involved in logic testing-based HT detection [15], thus highlighting RareLS's efficacy in facilitating HT detection. Smaller test length facilitates easier detection of HTs by the defender. Besides, generating shorter test vectors requires less time. On average, the application of RareLS resulted in a reduction in the required test length exceeding 80.83%. Notably, improvements are observed across all evaluated benchmarks except for *c2670*, where despite an almost 40% reduction in the count of rare signals, the test length remained unchanged. Moreover, the reduction in test length significantly accelerates the generation of test vectors. Take the benchmark *7552* for example. Without applying RareLS and with trigger inputs set to 10, 11, and 12, it takes 144.41 seconds, 376.20 seconds, and 337.33 seconds, respectively, to generate the necessary length of test vectors (see Table 3) for fully covering the valid trigger logic. In contrast, after applying RareLS, the runtime decreases dramatically to 1.03 seconds, 0.90 seconds, and 0.38 seconds respectively. This highlights RareLS's substantial impact on enhancing the efficiency of logic testing-based HT detection.

Interestingly, although the gate netlist of *c880* still contains 9 rare signals after optimization by RareLS, the rare conditions of these signals can be activated by the same input pattern, enabling full-coverage testing with just one test vector. In contrast, while also containing some rare signals, the optimized gate netlists of *c1908* and *s1423* yield no valid trigger logic through sampling, resulting in the complete elimination of the requirement of test vectors. These observations underscore that not all rare signals in a circuit are easily exploitable for triggering HT. On the one hand, those rare signals whose rare conditions cannot be activated by the same input pattern cannot contribute to a valid trigger logic, thus they do not increase the circuit's vulnerability to HT insertion. On the other hand, a set of rare signals whose rare conditions can be activated by the same test vector are ineffective as triggers for stealthy HT, as they can be efficiently detected with extremely short test lengths. These findings point to a promising avenue for future research: instead of exclusively targeting the reduction of rare signal counts, logic optimization algorithms could be tailored to eliminate rare signals that could potentially serve as valid and stealthy trigger logic. Such an approach would further enhance the circuit's resilience against HT insertion and streamline the process of testing-based HT detection.

5 CONCLUSION AND FUTURE WORKS

In conclusion, RareLS, a rarity-reducing logic synthesis technique, defends against HTs from a new perspective. It automatically reduces rare signals, making circuits more resistant to HT insertion and more friendly to HT detection. Our experimental results show that RareLS can dramatically reduce rare signals with a minor hardware cost. It also significantly reduces the number of HT triggers and shrinks the test length. In the future, we will improve RareLS by considering sequential circuits and sequential triggers.

REFERENCES

[1] Wei Hu et al. An overview of hardware security and trust: Threats, countermeasures, and design tools. *TCAD*, 40(6):1010–1038, 2020.

[2] Mingfu Xue et al. Ten years of hardware Trojans: A survey from the attacker's perspective. *IET Computers & Digital Techniques*, 14(6):231–246, 2020.

[3] Ayush Jain et al. Survey of recent developments for hardware Trojan detection. In *ISCAS*, pages 1–5, 2021.

[4] Swarup Bhunia et al. Hardware trojan attacks: Threat analysis and countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247, 2014.

[5] Kan Xiao et al. Hardware Trojans: Lessons learned after one decade of research. *TODAES*, 22(1):1–23, 2016.

[6] Nisha Jacob et al. Hardware Trojans: Current challenges and approaches. *IET Computers & Digital Techniques*, 8(6):264–273, 2014.

[7] R Sree Ranjani and M Nirmala Devi. Malicious hardware detection and design for trust: an analysis. *Elektrotehniski Vestnik*, 84(1/2):7, 2017.

[8] Xinmu Wang et al. Sequential hardware Trojan: Side-channel aware design and placement. In *ICCD*, pages 297–300, 2011.

[9] Jie Zhang and Qiang Xu. On hardware Trojan design and implementation at register-transfer level. In *HOST*, pages 107–112, 2013.

[10] Syed Kamran Haider et al. Advancing the state-of-the-art in hardware Trojans design. In *MWSCAS*, pages 823–826, 2017.

[11] Ayush Jain and Ujjwal Guin. A novel tampering attack on AES cores with hardware trojans. In *ITC-Asia*, pages 77–82, 2020.

[12] Jonathan Cruz et al. An automated configurable Trojan insertion framework for dynamic trust benchmarks. In *DATE*, pages 1598–1603, 2018.

[13] Rajat Subhra Chakraborty et al. MERO: A statistical approach for hardware Trojan detection. In *CHES*, pages 396–410, 2009.

[14] Sayandeep Saha et al. Improved test pattern generation for hardware Trojan detection using genetic algorithm and boolean satisfiability. In *CHES*, pages 577–596, 2015.

[15] Yangdi Lyu and Prabhat Mishra. Scalable activation of rare triggers in hardware Trojans by repeated maximal clique sampling. *TCAD*, 40(7):1287–1300, 2020.

[16] Zhendong Shi et al. Test generation for hardware trojan detection using correlation analysis and genetic algorithm. *TECS*, 20(4):1–20, 2021.

[17] Zhixin Pan and Prabhat Mishra. Automated test generation for hardware trojan detection using reinforcement learning. In *ASPDAC*, pages 408–413, 2021.

[18] Hassan Salmani et al. A novel technique for improving hardware Trojan detection and reducing Trojan activation time. *TVLSI*, 20(1):112–125, 2012.

[19] Mohammad Saleh Samimi et al. Hardware enlightening: No where to hide your hardware trojans! In *IOLTS*, pages 251–256, 2016.

[20] Aruna Jayasena and Prabhat Mishra. Design for trust utilizing rareness reduction. In *VLSID*, pages 437–442, 2024.

[21] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In *DAC*, pages 532–535, 2006.

[22] Heinz Riener et al. Scalable generic logic synthesis: One approach to rule them all. In *DAC*, pages 1–6, 2019.

[23] Alessandro Tempia Calvino et al. A versatile mapping approach for technology mapping and graph optimization. In *ASPDAC*, pages 410–416, 2022.

[24] Alan Mishchenko et al. Combinational and sequential mapping with priority cuts. In *ICCAD*, pages 354–361, 2007.

[25] Nangate, Inc. Nangate 45nm open cell library. https://si2.org/open-cell-library/, 2023.

[26] Valavan Manohararajah et al. Heuristics for area minimization in LUT-based FPGA technology mapping. *TCAD*, 25(11):2331–2340, 2006.

[27] Alan Mishchenko and Robert Brayton. Scalable logic synthesis using a simple circuit structure. In *IWLS*, volume 6, pages 15–22, 2006.

[28] Hanyu Wang et al. Cost-generic resubstitution algorithm with customizable cost functions. In *IWLS*, 2022.

[29] Alan Mishchenko et al. ABC: A system for sequential synthesis and verification. http://people.eecs.berkeley.edu/~alanmi/abc/, 2023.

[30] Synopsys, Inc. Synopsys softwares. http://www.synopsys.com, 2023.

[31] Mark C Hansen et al. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design and Test of Computers*, 16(3):72–80, 1999.

[32] EPFL Integrated Systems Laboratory. The EPFL combinational benchmark suite. https://github.com/lsils/benchmarks, 2023.

[33] Bicky Shakya et al. Benchmarking of hardware trojans and maliciously affected circuits. In *HaSS*, 2017.