Security-aware Physical Design against Trojan Insertion, Frontside Probing, and Fault Injection Attacks

Jhih-Wei Hsu
jayhsu@eda.ee.ntu.edu.tw
Graduate Institute of Electrical Engineering, National Taiwan University
Taipei, Taiwan

Kuan-Cheng Chen
ivanchen@eda.ee.ntu.edu.tw
Graduate Institute of Electrical Engineering, National Taiwan University
Taipei, Taiwan

Yan-Syuan Chen
b07901157@ntu.edu.tw
Department of Electrical Engineering, National Taiwan University
Taipei, Taiwan

Yu-Hsiang Lo
b08901047@ntu.edu.tw
Department of Electrical Engineering, National Taiwan University
Taipei, Taiwan

Yao-Wen Chang
ywchang@ntu.edu.tw
Department of Electrical Engineering, National Taiwan University
Taipei, Taiwan

ABSTRACT

The dramatic growth of hardware attacks and the lack of security-concern solutions in design tools lead to severe security problems in modern IC designs. Although many existing countermeasures provide decent protection against security issues, they still lack the global design view with sufficient security consideration in design time. This paper proposes a security-aware framework against Trojan insertion, frontside probing, and fault injection attacks at the design stage. The framework consists of two major techniques: (1) a large-scale shielding method that effectively covers the exposed areas of assets and (2) a cell-movement-based method to eliminate the empty spaces vulnerable to Trojan insertion. Experimental results show that our framework effectively reduces the vulnerability of these attacks and achieves the best overall score compared with the top-3 teams in the 2022 ACM ISPD Security Closure of Physical Layouts Contest.

CCS CONCEPTS

• **Hardware → Physical design (EDA)**.

KEYWORDS

Physical Design, Trojan Insertion, Frontside Probing Attacks, Fault Injection Attacks

ACM Reference Format:
Jhih-Wei Hsu, Kuan-Cheng Chen, Yan-Syuan Chen, Yu-Hsiang Lo, and Yao-Wen Chang. 2023. Security-aware Physical Design against Trojan Insertion, Frontside Probing, and Fault Injection Attacks . In *Proceedings of the 2023 International Symposium on Physical Design (ISPD '23), March 26–29, 2023, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3569052.3571876

ISPD '23, March 26–29, 2023, Virtual Event, USA

ACM ISBN 978-1-4503-9978-4/23/03…$15.00
https://doi.org/10.1145/3569052.3571876

1 INTRODUCTION

The integrated circuit (IC) fabrication process has become more and more complicated as technology advances. As a result, IC companies started outsourcing the design and production of the chips to lower manufacturing costs.

1.1 Background

As the semiconductor fabrication process requires many stages from upstream to downstream, attackers could have plenty of opportunities to modify ICs maliciously [1]. Besides, ICs are applied to many applications, such as mobile phones, communication, transportation, and other critical domains. If the ICs are successfully attacked, it could cost great losses to the semiconductor industry and even jeopardize civilians' lives. Consequently, these issues have raised serious concerns about the authenticity of the fabrication process and the trustworthiness of manufactured ICs. The demand for hardware defense methods is getting more critical than before.

There are two straightforward methods to maintain the reliability of ICs. First, we could thoroughly inspect whether the manufactured chips are damaged before selling them. Nevertheless, this method is not effective enough since ICs are not repairable after fabrication. Second, we could ensure that all the stages in the supply chain are trustworthy. However, this solution is difficult and expensive since the suppliers are all around the globe [2].

For the above reasons, *secure-by-design* has gradually drawn attention. Secure-by-design is a paradigm indicating that confidentiality, integrity, and availability must be considered throughout the computer-aided design (CAD) flow, from the specification stage to the fabrication and assembly stages [3]. The objective of secure-by-design is to proactively protect the ICs and ensure the effects of security methods implemented in the early design stages could remain consistent in the following stages.

Trojan insertion, *frontside probing*, and *fault injection attacks* are three common hardware attacks. Trojan insertion, also known as hardware Trojan attacks, is a malicious modification or inclusion made by suspicious third parties [1]. An IC attacked by Trojan insertion might change its original behavior, leak private information, or even the whole circuitry could be permanently damaged, thereby losing its functionality. Besides, the types of hardware Trojans are of

great variety, and they are becoming harder to be detected since the attacking techniques have been advancing these years [1]. Therefore, the threats brought by Trojan insertion are tremendous. As shown in Figure 1, general layouts without implementing any defense method would have numerous exploitable regions, which are the continuous empty spaces vulnerable to insert Trojans. In previous research, Tehranipoor *et al.* [2] revealed the vulnerability of ICs against malicious Trojan attacks. Given the reasons mentioned above, it is extremely urgent to develop effective Trojan-resisting countermeasures.

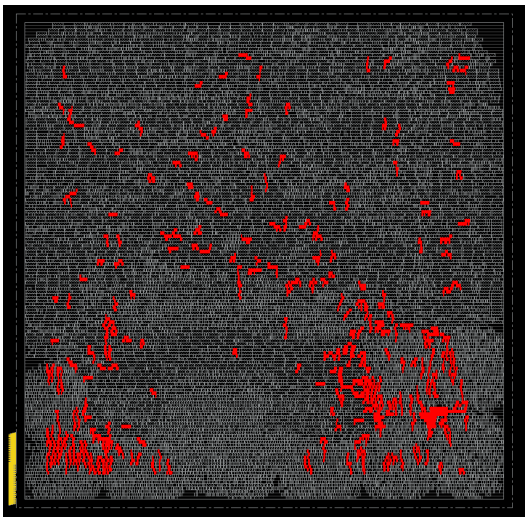

Figure 1. An advanced encryption standard (AES) layout without implementing any defense method against Trojan insertion. The red marks are the exploitable regions.

Probing attacks are invasive attacks meant to leak the private information of the ICs. As shown in Figure 2, probing attacks would bypass the security measures and expose the cells and nets containing sensitive information in the beginning. Afterward, attackers would use hardware tools to physically attach to the cells and nets exposed in the previous step and extract information from them [4]. At the application level, probing attacks could be implemented on security-critical IC devices, such as smart cards, smartphones, financial systems, and even military systems [5], which also indicates the significant risks to confidential information brought by probing attacks.

Fault injection attacks (FIAs) have also drawn attention in the last decade, and it has been proven to be highly effective in private hardware information leaking [6]. An FIA aims to physically interfere with an IC beyond its original functionality, and the errors or side-effects induced could be used to crack cryptographic keys and other secret data [7]. Using FIA and analyzing the abnormal outputs of the ICs, called *differential fault analysis (DFA)*, could significantly reduce the number of experiments originally needed for obtaining the secret keys [6]. Therefore, it is essential to create secure methods against FIAs for information protection.

This work aims to create secure-by-design countermeasures to cope with Trojan insertion, frontside probing, and fault injection attacks.

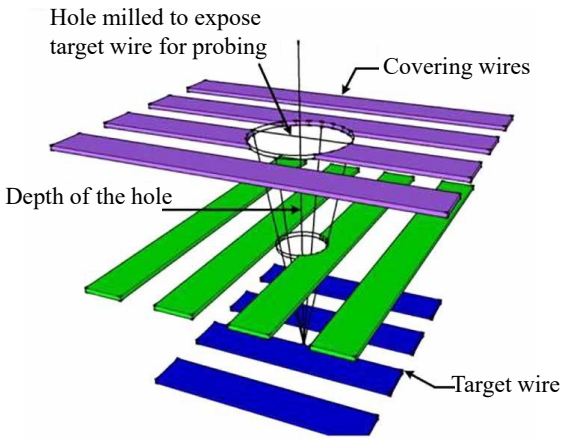

Figure 2. Illustration of the frontside probing attacks with focused ion beams (FIBs) [4].

1.2 Previous Work

1.2.1 Trojan Insertion Countermeasures. Wang *et al.* [8] presented the complexity and strength of Trojans, revealing that it is challenging to detect them after they are inserted into the hardware. Xiao *et al.* [1] proposed three classes of Trojan prevention methods, including (1) logic obfuscation, (2) camouflaging, and (3) functional filler cells. The concept of logic obfuscation is to hide the original implementation or functions of a design by inserting built-in locking mechanisms into it. The camouflaging technique aims at confusing the attackers by adding fake contacts and connections between nodes. Last, functional filler cells aim at filling unused empty spaces with dummy modules so that there is no room for Trojans [1].

Xiao *et al.* [9] present a novel technique called built-in self-authentication (BISA). BISA eliminates empty spaces and replaces them with functional filler cells instead of nonfunctional ones. However, our work is not allowed to insert filler cells. Nabeel *et al.* [10] proposed an active interposer that can integrate untrusted chiplets securely at the system level. It is an active method that monitors malicious behavior at runtime. Guo *et al.* [11] proposed a new language-based framework called QIF-Verilog. The QIF-Verilog framework presents a quantified information flow (QIF) model to evaluate the trustworthiness of a hardware system at the register transfer level (RTL). [10] and [11] focused more on detecting hardware Trojans during runtime or at the RTL level, but our work focused on making the physical layout more robust in the physical design stage.

1.2.2 Probing Attack and Fault Injection Attack Countermeasures. Cioranesco *et al.* [12] proposed an active shield method that detects milling by creating wires carrying dynamic signals on the top metal layer as a protective shield. Wang *et al.* [5] proposed a focused ion beam (FIB)-aware anti-probing physical design flow to protect the crucial wires by internal shield nets. The techniques aim at routing the non-security-related nets above the target nets to create a shield.

1.3 Motivation

Most hardware defense methods, especially Trojan insertion, are detection-based [1]. However, hardening the layouts against attacks at the design stage would be more secure. Current Trojan insertion countermeasures focus on detecting Trojans, and these detection methods could be vulnerable when the Trojan insertion techniques advance. Besides, current frontside probing attack and fault injection attack countermeasures cannot remove all the exposed areas and fully protect the cell and net assets. Furthermore, most previous works mainly focus on security and neglect the rise in design costs brought by their countermeasures.

1.4 Contribution

This paper proposes a framework to proactively harden the layout against Trojan insertion, frontside probing, and fault injection attacks. Unlike previous countermeasures that attempted to identify and block hardware attacks, our flow focuses on building a security-aware layout. The detailed contributions are summarized below:

- We propose an effective defense framework against Trojan insertion, frontside probing, and fault injection attacks. The framework consists of our proposed pre-processing techniques with two primary defense methods, (1) a large-scale shielding method that could effectively remove the exposed areas and (2) an effective cell movement algorithm to eliminate the exploitable regions.
- We propose pre-processing techniques and integrate them into the conventional physical design flow. As a result, we can create a suitable placement and routing result for our large-scale shielding method.
- We propose a large-scale shielding method that can effectively protect all the cell and net assets from frontside probing attacks.
- We propose a cell movement algorithm at the post-route stage, which could effectively eliminate exploitable regions while restraining the power cost.
- Our proposed framework also considers comprehensive design costs, including power, timing, and area, which are not often considered in previous defense methods.
- Experiment results show that our proposed framework can significantly reduce exploitable regions and exposed areas. Even more, we achieve the best scores among the participating teams at the 2022 ACM ISPD Security Closure of Physical Layouts contest [13], which justifies the effectiveness of our work.

The remainder of this paper is organized as follows. Section 2 introduces the addressed attack methods, describes the evaluation models, and formulates the security-aware layout design problem. Section 3 details our proposed algorithm flow and the techniques used at each stage. Section 4 reports the experiment results, and Section 5 concludes this paper.

2 PRELIMINARIES

In this section, we first introduce the Trojan insertion, frontside probing, and fault injection attacks. Then, we give our problem formulation of the security-aware design framework for Trojan insertion, frontside probing, and fault injection attacks.

2.1 Trojan Insertion Attack

Trojan insertion attacks are a malicious hardware modification of an electronic circuit, which leads to incorrect behavior during operation [14]. The concept of Trojans is diverse, including those (1) targeting at the system level, behavior level, gate level, or physical level, (2) intending to leak sensitive information or destroy the functionality of the circuit, and (3) crafted by additive, substitution, or subtractive techniques [3]. In this paper, we focus on post-design time gate-level addictive Trojan attacks.

Most Trojans comprise a trigger and a payload, as shown in Figure 3. The trigger is an optional part that monitors the circuit signals and activates the payload when an expected event occurs. The payload performs the actual attack when receiving the signal from the trigger. Otherwise, the payload remains inert, and the circuit performs as a Trojan-free circuit [1].

Trippel *et al.* [15] mentioned that a successful Trojan attack requires all three conditions: (1) Trojan placement, a spatial space to place the additional Trojan components, (2) Trojan integration, a connection between Trojan payload and security components, and (3) Intra-Trojan routing, a connection among the trigger and payload portions of Trojan. To complete a Trojan insertion attack, the attackers need at least a spatial space and routing resources to wire up the Trojan. Thus, the evaluation method in Section 2.3.1 will be based on this observation.

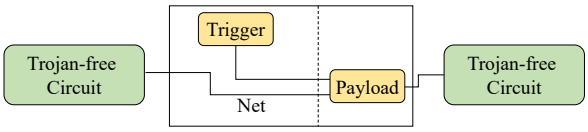

Figure 3. A Trojan consists of a trigger and a payload. To perform a Trojan attack, intra-Trojan routing and Trojan integration are required. That is, the trigger should be connected to the payload as well as a Trojan-free target asset.

2.2 Frontside Probing and Fault Injection Attacks

A probing attack is an invasive physical attack that enables the probe to expose the cells or nets to extract sensitive information through the frontside metal layers or the backside substrate [3]. The concept of probing attacks is diverse and covers those (1) using contact-based micro-probing, electromagnetic field probing, or electro-optical device probing to achieve an attack [13] and (2) aiming to obtain on-chip information, acquire device configuration, or destroy system functionality [3].

This paper focuses on the frontside contact-based micro-probing attack, which requires direct access to the target cells or wires and is often achieved by using techniques like the focused ion beam (FIB) [13]. First, an FIB mills a cavity with an ion beam to access the targeted wires, as shown in Figure 4(a). Then, some metal gas

atoms are injected and can be deposited in the cavity to build a conducting path called an electrical probe contact, as shown in Figure 4(b). Last, the attackers can extract the asset signal via the electrical probe contact [5]. Note that all the steps above should not damage the upper layer circuitry in order not to destroy the functions of the ICs.

Fault injection attacks aim to deduce sensitive information by directly or indirectly injecting faults during a cryptographic operation. Direct fault injection, for example, can be completed by using laser light or electromagnetic waves. On the other hand, indirect fault injection can be performed by repetitively writing to particular memory locations or by deliberately using dynamic voltage and frequency scaling (DVFS) features [13].

This paper concentrates on frontside direct fault injection, like laser fault injection. First, the laser light sneaking through the metal stacks directs to the target assets. Due to the photoelectric effect, electron-hole pairs are created and then induce the transient current. Finally, this may affect the PN junction and the switching of transistors of the related cells [13].

To sum up, the frontside probing and fault injection attacks addressed in this paper share the same attack principle, aiming to get down to the target assets via the metal stack. Thus, the evaluation method in Section 2.3.2 will be based on this observation.

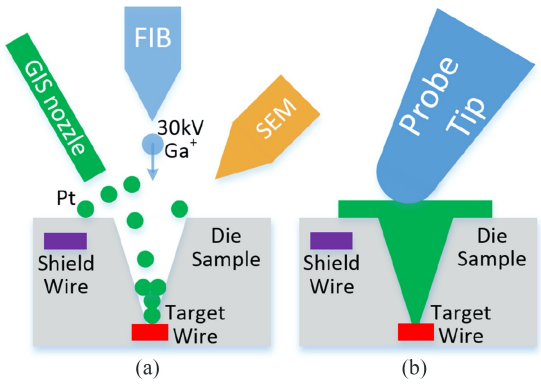

Figure 4. Illustrations of a focused ion beam (FIB). (a) An FIB mills a cavity to reach the target wire. (b) The deposition of the metal gases forms the conducting path, which serves as the electrical probe contact [5].

2.3 Evaluation

The evaluation methods in this paper follow the 2022 ACM ISPD Security Closure of Physical Layouts Contest [13]. This section introduces the exploitable region for Trojan insertion and the exposed area for frontside probing and fault injection attacks.

2.3.1 Exploitable Region for Trojan Insertion Attacks. An *exploitable region* is a region that attackers can use to insert Trojans while meeting the three requirements mentioned in Section 2.1. Specifically, it can be an empty region that is big enough to accommodate the Trojan components and have sufficient routing resources to route the Trojan.

The paper defines exploitable regions as sets of spatially continuous sites larger than the Trojan-placement threshold, which is the minimal exploitable placement sites. Without loss of generality, we set the threshold as 20 in this paper, which is the same as in the ISPD contest. To estimate the severity of the threat, we compute the maximum number, the average number, and the total number of sites across all exploitable regions. The three values can help evaluate the exploitable regions more comprehensively. The routing resource of the exploitable regions is defined by the number of free tracks above the exploitable regions. To simplify the evaluation process, we instead evaluate the number of all tracks above the exploitable regions.

In addition, Trojans should be placed and routed at appropriate positions to satisfy the design's timing constraints. Thus, an exploitable region is only defined within the exploitable distance of cell assets. The exploitable distance is the farthest distance among all legal Trojan placement positions from the target assets. The exploitable distance is computed as follows. First, we extract paths related to the assets with positive timing slacks. Then, we virtually insert an additional Trojan circuit. Without loss of generality, we use NAND gates, the most simple form of Trojan, for evaluation. With the lowest timing overhead among Trojans, NAND gates allow us to estimate exploitable distance conservatively. Next, we estimate the delays resulting from the additional Trojan circuit. For simplification, we do not perform actual routing. Instead, we estimate the slacks with information from the library, like wiring loads and capacitance loads. Finally, the exploitable distance is determined as the maximal distance with at least one path of positive slacks.

To sum up, we determine the exploitable regions as the ample continuous spatial spaces within the exploitable distance and further evaluate the Trojan insertion attack with the exploitable region and the routing resources above the exploitable region.

2.3.2 Exposed Area for Frontside Probing and Fault Injection Attacks. An *exposed area* is the region of the cell and net assets accessible by the probes from the frontside via the metal layers. Same as the ISPD contest, we assume that the probes are infinitely thin and only consider the attack probing from the frontside at zero degrees for simplification. That is, we do not consider the capabilities of adversaries and compute all the regions accessible through a direct line from the right top. This assumption is reasonable because some fault injection techniques do not need a large exposed area to induce fault injection. For example, laser fault injection requires just a small transistor-sized exposed area to induce fault injection. Meanwhile, the increasing exposed area provides an expanding attack surface, so the attacker gains more chances to induce fault injection on the target assets. Hence, we conclude that the exposed area is positively related to the security threat of assets. Here we define the exposed area as any region of the cell and net assets visible from the frontside by a direct line through the metal stacks.

Figure 5 shows an example of the exposed area of the standard cells in a layout. For evaluation, we further compute the maximum percentage, average percentage, and total exposed area across all assets to cover the different aspects of the threat from frontside probing and fault injection attacks.

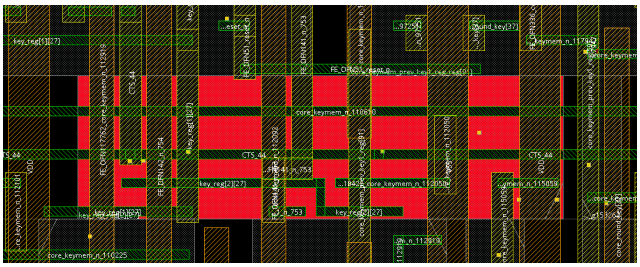

Figure 5. An example of the exposed area of a cell. The exposed area is marked in red.

2.4 Problem formulation

The problem formulated in this paper follows the 2022 ACM ISPD Security Closure of Physical Layouts Contest [13]. The security-aware layout design problem for Trojan insertion, frontside probing, and fault injection attacks is formally defined as follows:

Problem 1 (**The Security-aware Layout Design Problem for Trojan Insertion, Frontside Probing, and Fault Injection Attacks**). *Given a cell assets list, a net assets list, and a netlist of an original design, determine the desired position for each cell and net subject to the following hard constraints so that the security threat posed by Trojan insertion, frontside probing, and fault injection attacks can be minimized.*

- Must maintain functional equivalence.
- Must maintain cell and net assets.
- Must maintain the ratio of area covered by power delivery network(PDN) to die area relatively constant; variations of +/- 10% in area coverage are permitted.
- Cannot add extra metal layers.
- Cannot add customized, specialized circuitry.
- Cannot relocate PDN to a different layer.
- Cannot create custom cells, meaning only cells defined in the given LEF files can be used.
- Cannot insert filler cells, other non-functional cells, and functional but unconnected cells.

3 PROPOSED ALGORITHM

This section provides an overview of our proposed framework, outlining how we tackle the security-aware physical layout problem. Figure 6 illustrates our proposed security-aware design framework, which contains five stages: (1) *Floorplanning*, (2) *Probing-Aware Pre-processing*, (3) *Placement and Routing*, (4) *Probing-Aware Post-processing*, and (5) *Trojan-Insertion-Aware Post-Processing*. In floorplanning, we increase design utilization and maintain the overall structure of the power delivery network. Then we construct the placement regions and routing blockages in probing-aware pre-processing, and then perform region-aware placement and regular routing. In probing-aware post-processing, we create a large-scale shielding net to cover the whole assets. Finally, in Trojan-Insertion-Aware Post-Processing, we perform our greedy algorithm to eliminate exploitable regions.

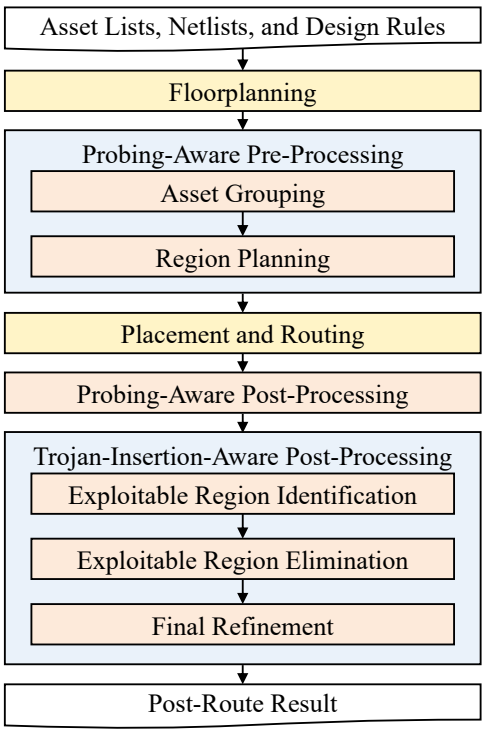

Figure 6. Overview of our proposed framework.

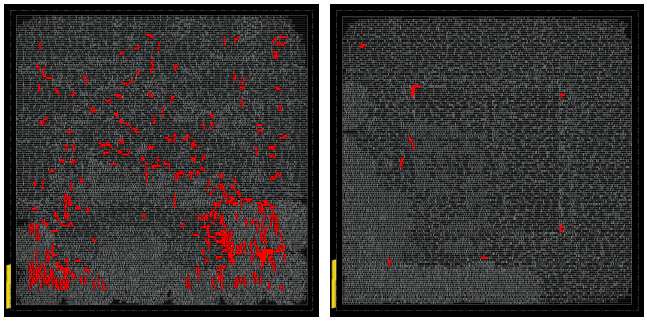

Figure 7. A comparison of layouts with different utilization rates. (a) A layout with a 75% utilization rate. (b) A layout with a 94% utilization rate. The red marks are the exploitable regions. A layout with high utilization rate reduces exploitable regions significantly.

3.1 Floorplanning

The objective of *Floorplanning* is to determine the appropriate core utilization rate and reconstruct the power delivery network (PDN). First, we extract the PDN information from the original netlist. Then apply a high core utilization rate to decrease exploitable regions preliminarily. Finally, we reconstruct the PDN to meet the constraint mentioned in Section 2.4.

Exploitable regions are one of the primary criteria for a Trojan insertion attack. A naive method to reduce exploitable regions is to apply a high core utilization rate, as shown in Figure 7. Increasing

the core utilization rate reduces exploitable regions and area costs significantly. However, a high core utilization rate also leads to the difficulty of timing closure. Based on the experiments, we set the target core utilization rate to 90%–95% for each case to prevent timing closure failures.

3.2 Probing-Aware Pre-Processing

Probing-Aware Pre-Processing is performed after *Floorplanning*. This stage is to make the placer place the cell assets and cells connected to net assets closer. The pre-processing procedure is divided into two steps: (1) *Asset Grouping* and (2) *Region Planning*.

3.2.1 Asset Grouping. Most state-of-the-art placers use routability-driven placement to address the congestion issue during placement. Many previous works proposed various techniques to deal with congestion problems, such as white spacing allocation [16], pin density control [17], and cell inflation [18]. These techniques increase the separation between cells to preserve more routing resources. However, protecting cell and net assets spread out all over the layout is complicated and inefficient. Therefore, we group both cell assets and cells connected to net assets and assign a higher weight to net assets to make assets closer and shorter.

Algorithm 1 Region Planning

Input: an asset group G_a, margin m, α_x, α_y
Output: an asset group region and a routing blockage region
1: $assets\_area \leftarrow 0$
2: **for** each cell c_i in G_a **do**
3: $\quad assets\_area$ += $c_i.area$
4: $ratio \leftarrow sqrt(assets\_area/core.area)$
5: $assets\_region.ll.x \leftarrow core.ll.x + \alpha_x * (1-ratio)/2 * core.width$
6: $assets\_region.ur.x \leftarrow core.ur.x - \alpha_x * (1-ratio)/2 * core.width$
7: $assets\_region.ll.y \leftarrow core.ll.y + \alpha_y * (1-ratio)/2 * core.height$
8: $assets\_region.ur.y \leftarrow core.ur.y - \alpha_y * (1-ratio)/2 * core.height$
9: $blockage\_region.ll.x \leftarrow assets\_region.ll.x - m$
10: $blockage\_region.ur.x \leftarrow assets\_region.ur.x + m$
11: $blockage\_region.ll.y \leftarrow assets\_region.ll.y - m$
12: $blockage\_region.ur.y \leftarrow assets\_region.ur.y + m$

3.2.2 Region Planning. Region Planning aims to construct a placement region and routing blockage. Algorithm 1 summarizes our planning algorithm, and the algorithm is detailed as follows. First, for each cell in the asset group, we calculate the summation of all cell areas in Lines 1–3. Next, we determine the asset region size based on the area ratio of the asset group area and the core area in Lines 4–8. α_x and α_y are two user-defined parameters to adjust the position of the region of the asset group. Finally, Lines 9–12 determine the position and size of a routing blockage by expanding the asset group region to prevent the regular routing wires from getting too close to the shielding net created later. In our implementation, the parameter m is set to the minimum value that satisfies a *parallel run length spacing* constraint.

3.3 Placement and Routing

After performing probing-aware pre-processing, the blockage information is passed to the placer and router, and region-aware placement and routing are performed. The region-aware placement and routing are performed by a leading commercial tool in our implementation. At this stage, any placer and router that honors region constraints can also be used.

3.4 Probing-Aware Post-Processing

This step aims to create a large-scale shielding net to cover all cell and net assets. First, we verify if all the net assets are routed within the routing blockage boundary constructed at the probing-aware pre-processing stage. If there exists any net asset which is routed out of a blockage boundary, these net assets are deleted. Then, we perform rip-up and reroute to ensure that all net assets are routed inside the blockage boundary. Second, we remove the routing blockage and select a high slack net that is not a net asset to create a large-scale shielding net in the top metal layer to cover both cell and net assets. The size of a shielding net is the same as a routing blockage.

3.5 Trojan-Insertion-Aware Post-Processing

Even though using a high core utilization rate can effectively reduce the majority of exploitable regions (Section 3.1), a few exploitable regions may still remain. To eliminate remaining exploitable regions, we apply a cell moving-based greedy algorithm. The algorithm contains three steps: (1) *Exploitable Region Identification*, (2) *Exploitable Region Elimination*, and (3) *Final Refinement*.

Algorithm 2 Exploitable Region Identification

Input: a placed design, a cell assets list L_c, exploitable distance d_e, threshold t
Output: a set of exploitable region list S_e
1: Initialize $boundary$ and S_e variables
2: **for** each cell c_i in L_c **do**
3: $\quad$ **if** $c_i.x < boundary.ll.x$ **then**
4: $\quad\quad boundary.ll.x \leftarrow c_i.x$
5: $\quad$ **else if** $c_i.x > boundary.ur.x$ **then**
6: $\quad\quad boundary.ur.x \leftarrow c_i.x$
7: $\quad$ **if** $c_i.y < boundary.ll.y$ **then**
8: $\quad\quad boundary.ll.y \leftarrow c_i.y$
9: $\quad$ **else if** $c_i.y > boundary.ur.y$ **then**
10: $\quad\quad boundary.ur.y \leftarrow c_i.y$
11: $boundary.ll.x$ -= d_e; $boundary.ur.x$ += d_e
12: $boundary.ll.y$ -= d_e; $boundary.ur.y$ += d_e
13: **for** each row r_i in rows intersects the $boundary$ **do**
14: $\quad$ **for** each site s_i in r_i **do**
15: $\quad\quad$ **if** s_i is empty, unmarked, and inside the $boundary$ **then**
16: $\quad\quad\quad (num\_site, site\_list) \leftarrow$ perform BFS(s_i)
17: $\quad\quad\quad$ **if** $num\_site >= t$ **then**
18: $\quad\quad\quad\quad S_e \leftarrow site\_list$
19: **return** S_e

3.5.1 Exploitable Region Identification. This step aims to identify all exploitable regions to be removed. We first identify the smallest

bounding box covering all cell assets. Next, we find the rows that intersect with the bounding box and then iteratively scan an empty site. Finally, we perform a breadth-first search (BFS) starting at the empty site to check if the site can form a region with more than the Trojan-placement threshold.

Algorithm 2 summarizes our identification algorithm. We first initialize variables *boundary* that store lower-left and upper-right coordinates, where S_e stores a set of exploitable region lists to be returned by Algorithm 2. Next, for each cell in the cell assets list, find the lower-left and upper-right coordinates of all cell assets in Lines 2–10. Then, we expand the bounding box based on a given exploitable distance d_e in Lines 11–12 because exploitable regions are only evaluated within an exploitable distance related to cell assets. Finally, Lines 13–18 for each row intersects the expanded bounding box, iterates each empty site, and performs BFS to check if the site can form an exploitable region. If so, we add a site list returned by the BFS algorithm to S_e and return it.

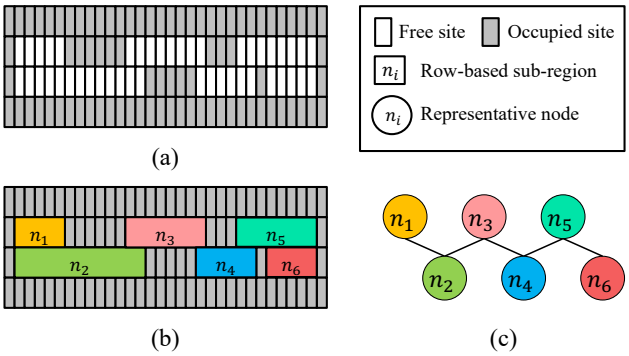

Figure 8. (a) An example of an exploitable region. (b) Row-based sub-regions of the exploitable region. (c) The graph transformed from (b).

3.5.2 Exploitable Region Elimination. This stage is to eliminate exploitable regions identified in Section 3.5.1. Due to the effort in Section 3.1, most designs should only remain sporadic exploitable regions. Thus, we propose a fast cell-movement-based method to eliminate the exploitable regions greedily. The objective is to maximize the gain, defined as the reduction of the total number of sites in exploitable regions caused by cell movement.

First, a row-based graph G is constructed, as shown in Figure 8(c). We partition an exploitable region into sub-regions, as shown in Figure 8(b) and then transform each sub-region into a node with a weight equal to the number of sites in the sub-region. Also, edges are constructed to represent the connection of consecutive sub-regions in adjacent rows.

Next, the exploitable regions are eliminated by separating them into regions with the number of continuous sites less than the Trojan-placement threshold. We initialize the cost with the number of sites in the exploitable region, which is the summation of nodes' weights in G. For each node, we compute the gains obtained by moving the unlocked cell around it to the free sites and select the node with the maximal gain. For example, we calculate the gain for node n_2 by moving the cell c_1, c_2, and c_3, respectively, from the

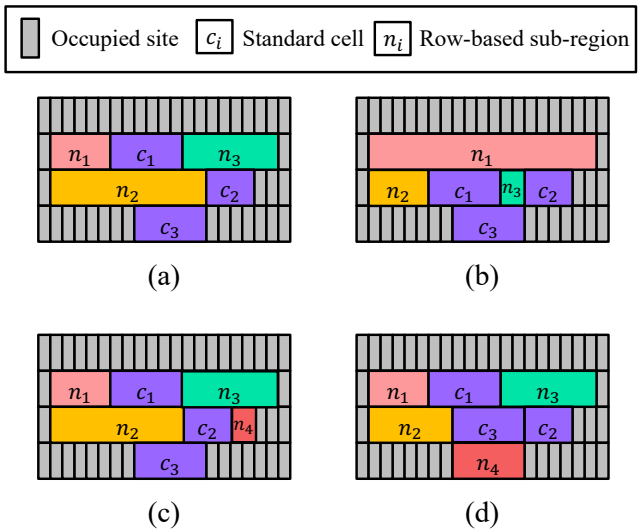

Figure 9. An example of cell movement process to illustrate how to determine which sub-region should be cut. (a) The original placement. (b) c_1 is moved to cut n_2. (c) c_2 is moved to cut n_2. (d) c_3 is moved to cut n_2.

original placement, as shown in Figure 9(a), deriving candidates, as shown in Figures 9(b)–(d), and select n_2 due to the highest gain of Figure 9(d). After that, we determine if the movement operation is legal by checking whether it creates new exploitable regions. The cell will be moved to the designated sites if the movement holds, and G will be updated. Otherwise, we do not perform an actual movement. Then, the cell is locked. The process is repeated until the exploitable regions are removed, or the cells around exploitable regions are all locked.

Our experimental results show that the algorithm can effectively remove most exploitable regions. In rare cases, the remaining ones will be tackled in Section 3.5.3.

3.5.3 Final Refinement. If exploitable regions still exist after executing *Exploitable Region Elimination*. It implies that standard cells around these exploitable regions are too small or too sparse to divide the exploitable regions through cell movement. Thus, upsizing nearby cells and/or inserting dummy buffers are applied to remove all remaining exploitable regions.

4 EXPERIMENTAL RESULTS

	# cells	# cell assets	# nets	# net assets	Utilization	# Layers
AES_1	16509	336 (2.04%)	19541	468 (2.39%)	75.1%	10
AES_2	16509	336 (2.04%)	19541	468 (2.39%)	75.1%	10
AES_3	15836	322 (2.03%)	18868	461 (2.44%)	94.8%	10
Camellia	6710	265 (3.94%)	7094	384 (5.41%)	51.1%	6
CAST	12682	1886 (14.87%)	13050	1919 (14.70%)	51.3%	6
MISTY	9517	256 (2.68%)	9895	14 (0.14%)	51.6%	6
PRESENT	868	164 (18.89%)	1030	145 (14.07%)	50.6%	6
SEED	12682	4810 (37.92%)	13055	4938 (37.83%)	51.37%	6
TDEA	2269	168 (7.40%)	2592	168 (6.48%)	81.12%	6
SPARX	128	8146 (1.57%)	10786	2176 (20.17%)	50.99%	6
openMSP430_1	4690	1541 (32.85%)	5310	1393 (26.23%)	50.46%	6
openMSP430_2	5921	1839 (31.05%)	6307	1717 (27.22%)	80.09%	6

Table 1. Benchmark Statistics.

Benchmark	1st place			2nd place			3rd place			Ours		
	des	ti	fsp_fi	des	ti	fsp_fi	des	ti	fsp_fi	des	ti	fsp_fi
AES_1	0.447645	0.000000	0.000000	0.475469	0.000000	0.000000	0.519014	0.000000	0.000000	**0.427583**	0.000000	0.000000
AES_2	**0.425056**	0.000000	0.000000	0.458233	0.000000	0.000000	0.509517	0.000000	0.000000	0.438185	0.000000	0.000000
AES_3	**0.473199**	0.000000	0.000000	0.498813	0.000000	0.000000	0.541594	0.000000	0.000000	0.485633	0.000000	0.000000
CAST	0.412035	0.000000	0.000000	0.409304	0.000000	0.000000	0.439133	0.000000	0.000000	**0.396717**	0.000000	0.000000
Camellia	**0.398203**	0.000000	0.000000	0.420739	0.000000	0.000000	0.418330	0.000000	0.000000	0.405498	0.000000	0.000000
MISTY	0.418306	0.000000	0.000000	0.396844	0.000000	0.000000	0.417127	0.000000	0.000000	**0.387017**	0.000000	0.000000
PRESENT	**0.359781**	0.000000	0.000000	0.427651	0.000000	0.000000	0.446817	0.000000	0.000000	0.363447	0.000000	0.000000
SEED	0.416061	0.000000	0.000000	0.442646	0.000000	0.000000	0.442510	0.000000	0.000000	**0.414805**	0.000000	0.000000
SPARX	0.397067	0.000000	0.000000	0.420406	0.000000	0.000000	0.404258	0.000000	0.000000	**0.387243**	0.000000	0.000000
TDEA	**0.459273**	0.000000	0.000000	0.526013	0.000000	0.000000	0.524128	0.000000	0.000000	0.484775	0.000000	0.000000
openMSP430_1	0.406426	0.000000	0.000000	0.440711	0.000000	0.000000	0.469361	0.000000	0.000000	**0.401875**	0.000000	0.000000
openMSP430_2	0.464010	0.000000	0.000000	0.543684	0.000000	0.000000	0.570014	0.000000	0.000000	**0.460260**	0.000000	0.000000
average	0.423089	-	-	0.455043	-	-	0.475150	-	-	**0.421087**	-	-
ratio	1.005	-	-	1.081	-	-	1.128	-	-	1.000	-	-

Table 2. Comparison of solution quality. The smallest design costs are marked in bold.

Our framework was implemented in the Tcl script language. We conducted all experiments on an AMD Ryzen 3990X 2.9GHz Linux workstation with 128GB memory. We used *Cadence Innovus 18* for physical design from floorplan to routing. We adopted the benchmark from the 2022 ACM ISPD Security Closure of Physical Layouts Contest, where Table 1 gives the benchmark statistics, including the number of cells ("# cells"), the number of cell assets ("# cell assets"), the number of nets ("# nets"), the number of net assets ("# net assets") , utilization rate ("Utilization"), and the number of metal layers ("# Layers"). The benchmarks contain different crypto core designs and microcontrollers with varying complexity ranges, timing constraints, utilizations, and the number of assets. The diversity of the benchmarks, with the number of cells and nets ranging from hundreds to ten thousand and the proportion of assets ranging from 2.04% to 37.92%, allowed us to examine the robustness of the framework proposed in this paper. All designs were synthesized by the *Synopsys Design Compiler* with the *Nangate 45nm Open Cell Library* [19]. Except for the AES series cases, other cases are limited to six metal layers for routing.

To evaluate the proposed framework, we used the evaluation scripts and the cost metric provided by the 2022 ACM ISPD Security Closure of Physical Layouts Contest. The cost metric is defined as follows:

$$overall\ cost = \frac{ti + fsp\_fi}{2} \times des, \tag{1}$$

where ti, which stands for the cost of Trojan insertion attack, is the average of the placement sites and routing resources of exploitable regions, fsp_fi, which denotes the cost of frontside probing and fault injection attacks, is the average of the exposed area of standard cell and net assets, and des, which represents the design cost, is the mean of power, performance, area, and the number of DRC violations.

Table 2 shows the experimental results. Our framework fully protects the physical design layout and achieves the best scores in terms of security cost. Furthermore, we still obtained the best design score compared with all the participating teams. Specifically, compared with the top-3 teams, we achieved a better average score with 0.5%, 8.1%, and 12.8% smaller design costs, respectively. In particular, we obtained the smallest design costs for seven out of twelve benchmark circuits (marked in bold).

In our observation, the robustness of our framework against the attacks is contributed by our two-stage security-aware processing strategy, which provides a global view in the early stage before placement and the local view in the latter stage after routing. For Trojan insertion attack, we primarily tackle the problem by increasing the utilization rate in the floorplanning stage, which reduces the possible exploitable regions and provides sufficient resources for Trojan-Insertion-Aware Post-Processing. For frontside probing and fault injection attacks, we conduct the probing-aware region planning before placement, which reserves the routing space for a large-scale shielding net and assures that the cell and net assets are placed and routed within the shielding region.

Furthermore, our better design scores result from our proper trade-off between security and conventional design costs. That is, we do not over-emphasize security and sacrifice the design cost. In all security-aware processing, we are also dedicated to minimizing the overhead of design costs. For example, in the region planning stage of probing-aware pre-processing, we determine a suitable region considering power, performance, and area. Also, in Trojan-Insertion-Aware Post-Processing, we conduct the cell-movement method before cell-resizing and/or buffer-insertion because the overhead of the former method is smaller than the latter.

5 CONCLUSION

This paper has proposed a security-aware framework against Trojan insertion, frontside probing attacks, and fault injection attacks while minimizing extra design costs. A large-scale shielding method has been proposed to cover all the exposed cell and net assets. A cell movement method has also been proposed to eliminate the exploitable regions with low overheads. Experimental results have shown that our proposed framework achieves the highest score among all participating teams in the 2022 ACM ISPD Contest.

REFERENCES

[1] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Trans. on TODAES*, vol. 22, no. 1, pp. 1–23, 2016.

[2] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[3] J. Knechtel, J. Gopinath, J. Bhandari, M. Ashraf, H. Amrouch, S. Borkar, S.-K. Lim, O. Sinanoglu, and R. Karri, "Security closure of physical layouts iccad special session paper," in *Proc. of ICCAD*, 2021, pp. 1–9.

[4] H. Wang, D. Forte, M. M. Tehranipoor, and Q. Shi, "Probing attacks on integrated circuits: Challenges and research opportunities," *IEEE Design & Test*, vol. 34, no. 5, pp. 63–71, 2017.

[5] H. Wang, Q. Shi, A. Nahiyan, D. Forte, and M. M. Tehranipoor, "A physical design flow against front-side probing attacks by internal shielding," *IEEE Trans. on CAD*, vol. 39, no. 10, pp. 2152–2165, 2020.

[6] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proc. of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[7] C. Shepherd, K. Markantonakis, N. van Heijningen, D. Aboulkassimi, C. Gaine, T. Heckmann, and D. Naccache, "Physical fault injection and side-channel attacks on mobile devices: A comprehensive analysis," *Computers & Security*, vol. 111, p. 102471, 2021.

[8] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *IEEE International Workshop on HOST*, 2008, pp. 15–19.

[9] K. Xiao, D. Forte, and M. Tehranipoor, "A novel built-in self-authentication technique to prevent inserting hardware trojans," *IEEE Trans. on CAD*, vol. 33, no. 12, 2014.

[10] M. Nabeel, M. Ashraf, S. Patnaik, V. Soteriou, O. Sinanoglu, and J. Knechtel, "2.5D root of trust: Secure system-level integration of untrusted chiplets," *IEEE Trans. on Computers*, vol. 69, no. 11, 2020.

[11] X. Guo, R. G. Dutta, J. He, M. M. Tehranipoor, and Y. Jin, "QIF-Verilog: Quantitative information-flow based hardware description languages for pre-silicon security assessment," in *IEEE International Symposium on HOST*, 2019, pp. 91–100.

[12] J.-M. Cioranesco, J.-L. Danger, T. Graba, S. Guilley, Y. Mathieu, D. Naccache, and X. T. Ngo, "Cryptographically secure shields," in *IEEE International Symposium on HOST*, 2014, pp. 25–31.

[13] Security closure of physical layouts. [Online]. Available: https://wp.nyu.edu/ispd_22_contest/

[14] M. Beaumont, B. Hopkins, and T. Newby, "Hardware trojans-prevention, detection, countermeasures (a literature review)," 2011.

[15] T. Trippel, K. G. Shin, K. B. Bush, and M. Hicks, "Icas: an extensible framework for estimating the susceptibility of ic layouts to additive trojans," in *IEEE Symposium on Security and Privacy*, 2020, pp. 1742–1759.

[16] L. Chen, X. Min, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," in *Proc. of ICCAD*, 2004, pp. 394–401.

[17] M.-K. Hsu, Y.-F. Chen, C.-C. Huang, S. Chou, T.-H. Lin, T.-C. Chen, and Y.-W. Chang, "NTUplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE Trans. on CAD*, vol. 33, no. 12, pp. 1914–1927, 2014.

[18] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, Y.-W. Chang, T.-C. Chen, and I. Bustany, "NTUplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE Trans. on CAD*, vol. 37, no. 3, pp. 669–681, 2018.

[19] Nangate freepdk45 open cell library. [Online]. Available: https://si2.org/open-cell-library/