GDSII-Guard: ECO Anti-Trojan Optimization with Exploratory Timing-Security Trade-Offs

Xinming Wei, Jiaxi Zhang, Guojie Luo✉
School of Computer Science, Peking University
Center for Energy-efficient Computing and Applications, Peking University
{weixinming, zhangjiaxi, gluo}@pku.edu.cn

***Abstract*—With the ever-shrinking feature size of transistors, the exorbitant cost has driven the massive outsourcing of integrated circuits (IC) fabrication. However, this outsourcing poses significant security risks because untrustworthy foundries can conduct insidious fabrication-time attacks without close supervision. Therefore, it is crucial to undertake design-time protection before sending finalized design layouts to the foundry. Foundry-level hardware Trojan has emerged as a major security threat, but existing design-time countermeasures lack sufficient consideration of good trade-offs between design security and performance.**

This work proposes an automatic framework, GDSII-Guard, to strengthen implemented physical layouts against potential fabrication-time Trojan attacks while preserving design performance, power, and quality. We develop an Engineering Change Order (ECO) placement and routing (P&R) flow containing elaborate anti-Trojan operators to prevent Trojan insertion. Moreover, we introduce a multi-objective optimization model with evolutionary strategies that incorporate anti-Trojan flow information to exploit balances between the aforementioned multiple design metrics. Experimental results demonstrate that GDSII-Guard reduces the overall risk of Trojan attacks on given designs by 98.8% with minimized timing, power, and design quality impact, surpassing existing approaches prominently.

I. Introduction

Thanks to the exponential power of Moore's law, transistors in modern Integrated Circuits (ICs) have shrunk to deep nanoscale levels (*e.g.,* 3nm/5nm process). This ever-shrinking trend of transistors has promised higher miniaturization, better performance, and lower power consumption. However, the dramatic increase in chip complexity has driven the cost of IC manufacturing sky-high, which is prohibitive for most IC designers or companies. As a result, mask production, chip fabrication, and final test services are largely outsourced throughout the modern IC industry, while entirely in-house supply chains are rare to be found. Since IC design flows are mostly fabless, untrustworthy foundries can make malicious hardware modifications, known as a ***fabrication-time attack***. Previous works demonstrate several ways that fabrication-time attackers can insert a hardware Trojan into an otherwise trusted, tapeout-ready IC layout [1]–[3], which has emerged as a major hardware security concern [4].

The objectives and measures to insert hardware Trojans can be manifold. Undesirable effects introduced to target hardware or systems are: 1) Functional change, error injection, or system failure. 2) Performance degradation. 3) Sensitive information leakage [4]. A hardware Trojan consists of two key components: trigger and payload. The trigger activates the payload when some attack condition is satisfied, and the payload serves to execute actual attacks, *e.g.,* altering the circuit functionality. While diverse fabrication-time attack schemes have been proposed, they tend to share one similarity: ***Trojan insertion requires extra free space on the finalized physical layout of an IC***. Attackers need to exploit unused placement and routing resources to append Trojan gates and wires, and integrate Trojan logic with victim circuit instances.

Due to the stealthy nature of hardware Trojans, it is quite challenging to protect ICs against fabrication-time Trojans. Conventional post-manufacturing test and post-silicon validation cannot reliably detect Trojans because Trojan instances tend to have varying forms and sizes, and only by rare events will they be triggered. Previous works mainly focus on the Trojan detection in two broad classes: logic testing [5], [6] and side-channel analysis [7]–[9]. Despite being effective in recognizing some kinds of Trojans, detection-based efforts are still prone to elaborate attack models or Trojans with small footprints [4]. Moreover, unlike software, there is no remedy for vulnerabilities detected in ICs after manufacturing. Given the limitations of Trojan detection methods, ***security-by-design***, *i.e., hardening the IC layout to frustrate foundry-level Trojan injection*, should play a more fundamental role. However, Security-by-design approaches are challenging because 1) Design rules and overall functional equivalence of the original design should be followed. 2) Potent security techniques can compromise IC performance and design quality. 3) Mainstream commercial and open-source CAD tools are performance-driven and have no direct support for security at design time.

Recent security-by-design works focus on pre-fabrication, IC layout-level, Trojan prevention [10]–[13]. These works increase the utilization of placement and/or routing resources of the design, to squeeze the available spaces or routing tracks for Trojan insertion. Two major techniques have been proposed and proved useful. ICAS [10] works in an undirected manner by adjusting some P&R parameters (*e.g.,* core density, slew rate) to increase P&R utilization. BISA [11] and Ba *et al.* [12], [13] fill unused spaces on the layout with tamper-evident logic. Nevertheless, though such defenses create barriers for attackers to P&R Trojans, they cannot provide complete confidence against diverse Trojan attacks. Moreover, performance, power, and area (PPA) overheads brought by such approaches are not well demonstrated. For example, increasing global placement density can lower the risk of Trojan insertion by reducing free spaces on the layout, but timing and design quality may inevitably get worsen due to routing congestion.

In this paper, following the notion of security-by-design at physical layout level, we propose GDSII-Guard, an ECO framework to reinforce finalized physical designs against Trojan insertion with timing co-optimization. DRC and power dissipation are also taken into account. Our contributions can be summarized as follows:

- We propose a novel security-driven, timing-aware automatic framework to prevent malicious Trojan insertion while preserving design performance, DRC, and power.
- We develop a suite of parametric, composable, and customized ECO P&R operators: Cell Shift, Local Density Adjustment, and Routing Width Scaling, which together construct an anti-Trojan CAD flow.
- We introduce a multi-objective optimization model to characterize GDSII-Guard flow parameter space and explore Pareto optimality between security and timing.

- We evaluate GDSII-Guard on comprehensive benchmarks, including crypto cores and microprocessors. GDSII-Guard strengthened designs demonstrate an average of 98.8% lowering of Trojan insertion risk with modest timing, power and design quality overheads, outperforming state-of-the-art defenses in both aspects.

II. Background

A. Security Metrics

The prior art has proposed several sets of metrics to evaluate the difficulty of injecting hardware Trojans into a given physical layout [10], [14], [15]. Most recently, J. Knechtel *et al.* [14] develop a framework for the evaluation of P&R resources available for Trojan insertion.

*Definition 2.1 (**Security-Critical Cells [14]**):* Security-critical cells are defined as sensitive cells to be protected in a design. Usually, they belong to key-memory registers or key-control logic. They are highly likely to become the target of Trojan attackers.

*Definition 2.2 (**Exploitable Regions [14]**):* Exploitable regions are generally sets of contiguous placement sites which are free for Trojan insertion, *i.e.,* empty, or occupied by filler cells/non-functional cells/dangling functional cells. A single exploitable region is defined with the following prerequisites: 1) It is a set of spatially (vertical and/or horizontal in a layout) adjacent exploitable sites. 2) Sites in the set are within an *exploitable distance* of security-critical cell assets defined in Definition 2.1 because timing constraints still need to be met after Trojan insertion. 3) The total number of sites in the set is no less than a threshold, `Thresh_ER`.

To be more specific about exploitable regions, setting a threshold to the site number is based on the observation that at least a few contiguous sites are needed for Trojan insertion. The *exploitable distance* is determined as follows: Firstly, paths with positive timing slacks to security-critical cell assets are extracted. Next, an additional NAND gate representing a simplest Trojan is appended into the paths. Finally, the maximal distance (both horizontally and vertically) of Trojan routing after which the consumed slacks still meet timing, is defined as exploitable distance. Fig. 1 shows an example layout with all exploitable regions annotated. It can be estimated that empty spaces close to security-critical assets tend to be exploitable because of the exploitable distance limitations. As for the quantization of the exploitable region, [14] proposes the following two sub-metrics:
Free Placement Sites: #sites composing all exploitable regions.
Free Routing Tracks: #unused routing tracks across all metal layers over all exploitable regions.

B. Threat Model

Our threat model mostly follows that adopted by previous works on foundry-level Trojan attacks and defenses [3], [10]. Specifically, we focus on malicious Trojan attacks at *fabrication-time* and assume all the other phases are trusted. The design process before tapeout, which consists of RTL design, logic synthesis, and P&R, together with the post-fabrication test and packaging, are warranted to be performed by trusted parties. Our focus on fabrication-time is motivated by current IC supply chains, economic forces, and technology reasons that require heavy reliance on outsourced production and otherwise trusted foundries.

Right after tapeout, the attacker in the untrusted foundry starts with the GDSII file. Our threat model assumes the worst case for the defender, that the attacker is fully capable of extracting circuit interconnections and components from the GDSII layout to restore schematic and netlist information, with techniques such as reverse engineering. This stems obfuscation-based defenses because the attacker can select viable victim instances and wires in the physical layout to tap. We assume that the adversary manipulates the trusted layout by inserting additional cells/wires; they cannot remove, resize, or shift existing circuit components. These operations are almost infeasible in practice because they take a high risk of either spoiling the functional equivalence of the underlying design or violating the fragile timing constraints and design rules [10], [14]. Even if such problems may be addressed by arduous fixes, the increased lead time for chip fabrication will break the tight limit agreed in the contract. Moreover, the attacker cannot extend the layers, dimensions or size of the current design, which can be easily discovered after manufacturing. To sum up, in the scope of our work, we assume that the attacker exploits open spaces in the original GDSII for Trojan implementation.

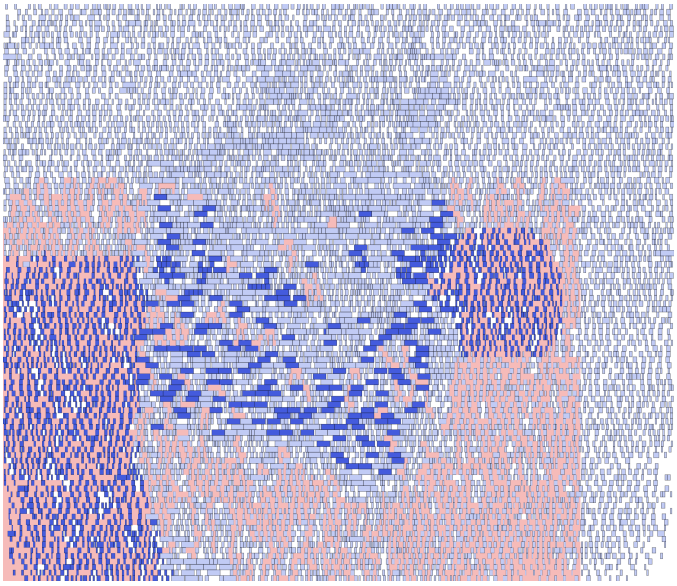


Fig. 1: Exploitable regions (in red), security-critical cells (dark blue rectangles), and all other cells (light blue rectangles) in a placed and routed (routes are hidden) layout.

C. Problem Formulation

Given some baseline physical designs, we summarize the problem as hardening the physical layouts to reduce exploitable regions and thwart Trojan insertion. We consider performance, power, and DRC overheads as well. Throughout the rest of the paper, we denote $\mathbf{L}_{\text{base}}$ and $\mathbf{L}_{\text{opt}}$ as the baseline and optimized layouts, $f(\cdot;\boldsymbol{x})$ as our security flow, $\mathcal{D}$ as the hyper-parameter space of the flow and $\boldsymbol{x} \in \mathcal{D}$ as a parameter feature vector. Let $\text{ERsites}(\cdot)$, $\text{ERtracks}(\cdot)$ denote the sum of free sites and tracks of all the exploitable regions in a layout. Then, the security metric can be formulated as a weighted sum of $\text{ERsites}(\cdot)$ and $\text{ERtracks}(\cdot)$, normalized by the baseline design:

$$\text{Security}(\mathbf{L}_{\text{opt}}) = \alpha \cdot \frac{\text{ERsites}(\mathbf{L}_{\text{opt}})}{\text{ERsites}(\mathbf{L}_{\text{base}})} + (1-\alpha) \cdot \frac{\text{ERtracks}(\mathbf{L}_{\text{opt}})}{\text{ERtracks}(\mathbf{L}_{\text{base}})}$$

We measure timing with Total Negative Slack (TNS). Designs with better timing have negative TNS closer to 0, and the optimal TNS is 0. Power and design cost are characterized by total (leakage, switching, and internal) power and DRC violations, respectively. Now our problem formulation can be summarized as follows:

Inputs: $\mathbf{L}_{\text{base}}$, security-critical assets list, timing specifications.
Output: $\mathbf{L}_{\text{opt}} = f(\mathbf{L}_{\text{base}};\boldsymbol{x})$

Objectives:

$$\begin{aligned} \min \quad & \text{Security}(\mathbf{L}_{\text{opt}}) \\ \min \quad & -\text{TNS}(\mathbf{L}_{\text{opt}}) \\ \text{s.t.} \quad & \text{DRC\_viol}(\mathbf{L}_{\text{opt}}) \leqslant N_{\text{DRC}}, \\ & \text{Power}(\mathbf{L}_{\text{opt}}) \leqslant \beta_{\text{power}} \cdot \text{Power}(\mathbf{L}_{\text{base}}). \end{aligned}$$

The goal is to find an effective layout optimization flow $f(\cdot;\boldsymbol{x})$ and the Pareto-optimal parameter configuration $\boldsymbol{x}$, which bring about improved security metrics and performance while minimizing the impact on power and DRC. Valid results should be subject to the DRC and power upper bounds.

III. GDSII-Guard

A. Overview of GDSII-Guard

Fig. 2 shows an overview of GDSII-Guard framework. The core of GDSII-Guard is a security-driven, timing-aware ECO P&R flow. It takes an implemented baseline layout, LIB/LEF libraries, annotated security-critical cell assets, and a parameter configuration vector of the flow as the input. Initially we preprocess the original design such that the critical cells will not be removed or replaced during the subsequent security optimization. As for the ECO security enhancement, we propose two alternative ECO placement operators and one ECO routing operator. After the design is fully augmented, GDSII-Guard extracts the post-design metrics (*i.e.,* security, timing, DRC, and power), and iteratively explores the optimal flow parameter configurations for the given design. Finally, we derive a set of Pareto-optimal enhanced layouts with security and performance trade-offs.

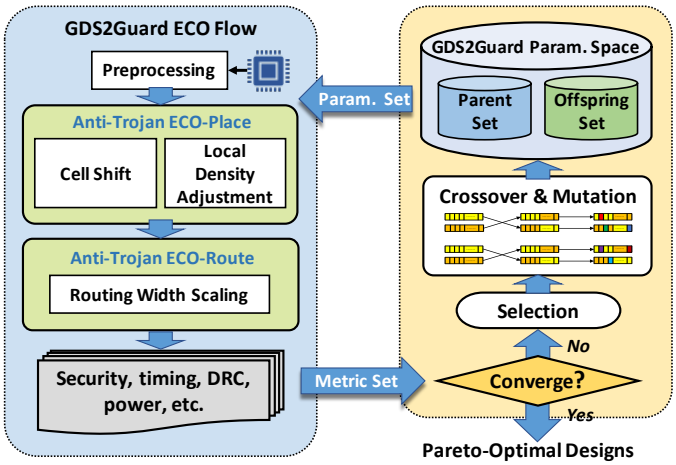


Fig. 2: Overview of GDSII-Guard.

B. Anti-Trojan ECO Placement

Placement is the most important stage for security strengthening measures to eliminate as many exploitable regions in a layout as possible. However, previous placement-centric approaches cannot provide both security confidence and PPA assurance. Running global placement at higher density [10] is essentially security-agnostic and consumes lengthy runtime. Filling empty spaces with functional logic [11]–[13] requires >90% local placement density, which undermines routability and timing closure.

In our work, instead of reducing exploitable regions in a "blind" manner, we manage to perform meticulous Trojan-resistant ECO placement without sacrificing design performance and quality. We conduct security and timing optimizations simultaneously. According to different attributes (*e.g.,* timing constraints, initial placement density) of the original design, we propose the following two alternative anti-Trojan ECO placement operators.

*1) **Cell Shift (CS)**:* Cell shift manages to erase exploitable regions globally in a layout by row-wise shift of cells. This operator applies well for designs with loose timing constraints (*i.e.,* the target clock frequency is easy to achieve after P&R) because in such designs exploitable distances can be so long as to spread across the whole core, and the operator attempts to eliminate large continuous regions everywhere without pursuing a soaring placement density.

We propose an undirected graph model $G = (V, E)$ to represent all the empty sites in each row of a physical layout. Fig. 3 depicts the cell-shift process and the corresponding graph model of a tiny example layout. A **vertex** $v \in V$ (the circle in Fig. 3) refers to a few contiguous sites in the same core row. It adjoins either a cell or the core boundary. Let the weight of v, $\mathtt{w}(v)$ (the number in circles in Fig. 3) denote the number of sites in the vertex. Two vertices v_1, v_2 are defined to be connected (*i.e.,* $\exists e \in E, e = (v_1, v_2)$) iff 1) they are in two adjacent rows respectively, and 2) some of their sites are contiguous, *i.e.,* there exists a site s_1 in v_1, s_2 in v_2 such that s_1 and s_2 are aligned vertically. A **component** C of this graph is a connected subgraph that is not a part of any larger connected subgraph. Obviously, all components partition the vertices in the graph into disjoint sets. The weight of C, $\mathtt{w}(C)$ refers to the summation of sites in the included vertices of the component. So we have $\mathtt{w}(C) = \sum_{(v,e)\in C} \mathtt{w}(v)$. If $\mathtt{w}(C) \geqslant \mathtt{Thresh\_ER}$, the component C is an **exploitable** region. Let $C = \mathtt{compo}(v)$ denote the component containing the vertex v. A vertex is involved in one and only one component according to their definitions. The objective is to shift cells to minimize $|\{C|\mathtt{w}(C) \geqslant \mathtt{Thresh\_ER}\}|$.

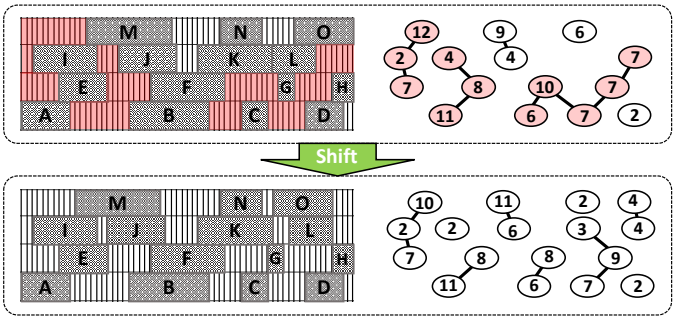


Fig. 3: A toy example layout with the corresponding graph representation. Gray instances with a letter inside are placed cells. Sites/Vertices in red are within exploitable regions/components whose weights reach Thresh_ER (here we take 20). Exploitable regions are erased after row-wise cell shift.

We summarize the procedure of our greedy-based, cell shift algorithm as follows. The cells are shifted horizontally row by row. When processing a row, we construct the graph with the current row and all previously processed rows. Then we visit each vertex v in the row sequentially and obtain $C = \mathtt{compo}(v)$ through *depth-first search* (DFS) traversal. If C is exploitable, the adjacent cell on the right will be moved left to decrease $\mathtt{w}(v)$, until $\mathtt{compo}(v) < \mathtt{Thresh\_ER}$ or the vertex disappears (*i.e.,* $\mathtt{w}(v) = 0$). The principle of movement is to stop moving right after $\mathtt{compo}(v)$ is not exploitable, *e.g.,* $\mathtt{w}(\mathtt{compo}(v)) = \mathtt{Thresh\_ER} - 1$. This makes moving distance as short as possible, thus minimizing the impact on the timing of the original design. Ideally, after processing all the cells in each row, only the rightmost vertex may be in an exploitable region. Algorithm 1 depicts the details of the implementation. The same procedure,

except that the visiting sequence of vertices and the shift direction is reversed, is performed then to remove the potential exploitable regions on the right side of the core.

Algorithm 1: Greedy-Based Cell Shift

Input: A baseline layout $\mathbf{L}_{\text{base}}$ with N rows. $rows[]$ stores the position information of cells in each row.

```
for i ← 0 to N − 1 do
    Build layout graph G_{0,i} with rows[0 : i];
    Get vertex list V_i in rows[i];
    j ← 0;
    while j < length(V_i) do
        v ← V_i[j];
        if w(compo(v)) ⩾ Thresh_ER then
            cell ← the cell on the right of v;
            while w(v) > 0 && w(compo(v)) ⩾ Thresh_ER do
                Shift cell left for one site;
                Update rows[i], G_{0,i} and V_i;
            end
        end
        if is_exist(v) then j++;    ▷ v is deleted from V_i if w(v)==0
    end
end
```

*2) **Dynamic Local Density Adjustment (LDA)**:* In addition to wiping out large contiguous empty spaces globally, an alternative is to adjust local placement density dynamically. In regard to designs with tight timing constraints or low utilization rate, the cell shift operator is no longer adequate because modifying cell positions aggressively can deteriorate the fragile timing. In these designs, the core utilization of about 50%∼60% is benign to timing closure, but the cell shift algorithm cannot work at such a low density due to excessive empty sites. Since the exploitable distance is relatively short in timing-tight designs, empty sites near security-critical assets tend to be more vulnerable. Based on the notion of localized density elevation [16], we partition the layout into tiles and assign relatively higher placement density to the parts with more security-critical cells. Then we perform ECO placement to alter density distribution incrementally. Note that the dynamic density rearrangement via ECO placement is wire-length/timing driven. Therefore, the low-density regions will be "pushed" away from security-critical cells with minimized impact on circuit performance.

Algorithm 2 gives the full implementation of LDA. The core of the layout is partitioned into $N \times N$ grids. We leverage placement blockages to control local placement density. A partial placement blockage can set the density upper bound in a specified region. We create a placement blockage in each grid, and specify the local density upper limit with the following logistics: The number of security-critical cells in each grid is counted and normalized, and then smoothed into a valid density value (0∼100%) with `sigmoid` function. When all the blockages are properly created, ECO placement incrementally and efficiently manages the tile density. The above procedure can be repeated for `n_iter` times to enhance the effect of LDA.

C. Anti-Trojan ECO Routing

To complement placement-centric operators, we develop the anti-Trojan ECO ***Routing Width Scaling (RWS)*** to further reduce free routing tracks. Routing-centric defensive approaches are challenging because 1) imposing extra constraints on the router can easily deteriorate timing, and 2) appending shielding wires over security-critical assets, despite being timing-oblivious, can be easily recognized and removed by attackers due to their dangling nature.

To overcome these challenges, GDSII-Guard modifies the *non-default rule* (NDR) defined in LEF file and increases the routing

Algorithm 2: Dynamic Local Density Adjustment

Input: A baseline layout $\mathbf{L}_{\text{base}}$
Data: Grid partition $N \times N$, iteration times `n_iter`

```
Divide L_base into grids[N][N];
while n_iter- - do
    Delete all existing placement blockages;
    Initialize n_assets[][] with #security-critical cells in each grid;
    μ ← mean(n_assets[][]);
    σ ← stddev(n_assets[][]);
    for i ← 0 to N − 1 do
        for j ← 0 to N − 1 do
            dens_{i,j} ← sigmoid((n_assets[i][j] − μ) / σ);
            Create a placement blockage within grids[i][j] with
              density upper bound dens_{i,j};
        end
    end
    Run ECO placement;
end
```

wire width of different metal layers selectively. In a layout with K metal layers, `scale_M[i]` denotes the routing width scaling factor of layer $i = 1, \ldots, K$. For one thing, wider nets can further decrease the remaining available routing tracks to prevent Trojan routing. For another, reasonably wider nets, *e.g.,* clock nets, have smaller wire resistance and RC delay, thus improving overall timing [17]. Nevertheless, this anti-Trojan routing method works in a heuristic way to minimize free tracks over exploitable regions, so we introduce the black-box parameter tuning method in the next section to improve security and timing simultaneously.

TABLE I: Parameter space of GDSII-Guard operators.

Parameter Name	Description	Candidate Values
`op_select`	The selected ECO-place operator	"CS", "LDA"
`LDA::N`	#Grids in a row/column	2, 4, 8, 16, 32
`LDA::n_iter`	#Density adjustment iterations	1, 2, 3
`RWS::scale_M[i]`	Routing width scale factor of metal i ($i = 1, \ldots, K$)	1.0, 1.2, 1.5

D. Multi-Objective Flow Parameter Tuning

We build a multi-objective optimization model upon the GDSII-Guard ECO flow parameter space to capture the trade-offs between security and performance, and guide the exploration process to find the superior parameter configurations. Table I lists the GDSII-Guard parameter space, whose size is up to 945k given 10 routing layers. We adapt Genetic Algorithm (GA) to our context and map our parameter space to GA *chromosomes* (*i.e.,* solution vectors). Each parameter is encoded as a *gene* in the chromosome. GA evaluates each chromosome with a so-called *fitness function* and selects prior ones from the *population* (*i.e.,* a pool of chromosomes). GDSII-Guard determines fitness according to many factors: valid solutions should first meet hard constraints of power and DRC described in §II-C, and then those with better security and timing metrics will prevail. To generate new solutions called *offspring*, *crossover* and *mutation* are two key genetic operators in GA. Genes of good chromosomes are expected to maintain in the population. For example, in a design with high placement density and tight timing constraints, too wide routes will cause routing congestion everywhere and easily deteriorate timing. Under such conditions, route scale genes, *i.e.,* `scale_M[i]` ($i = 1, \ldots, K$) of 1.0 will appear more frequently in the population. The algorithm terminates if the population has converged (*i.e.,* does not reproduce offsprings with pronounced improvements).

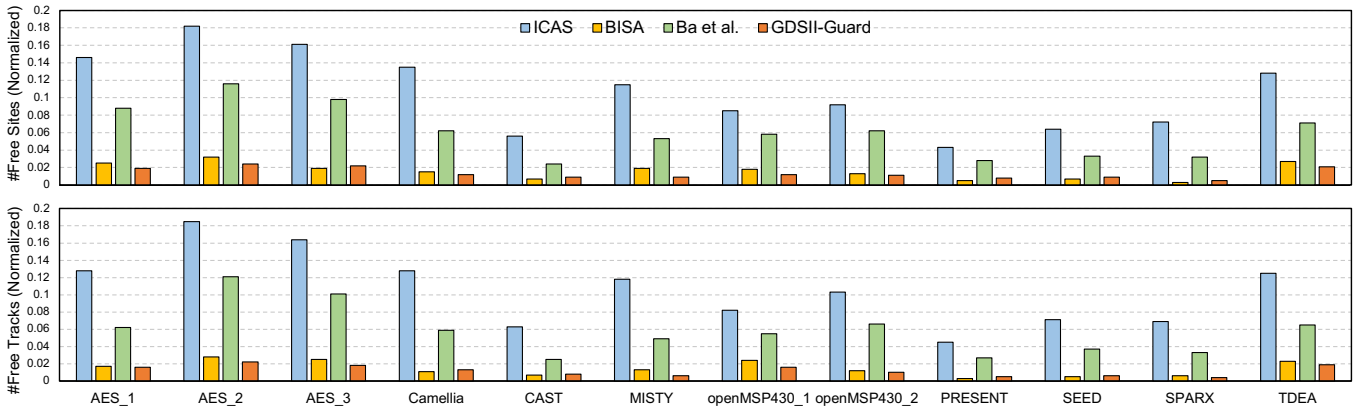


Fig. 4: Comparison of security metrics (normalized total free sites/tracks).

IV. Evaluations

A. Experimental Setup

Implementation. GDSII-Guard integrates the frontend customized `Python/Tcl` scripts with the commercial EDA tool, Cadence Innovus 19.12 as the P&R backend. We introduce a well-known multi-objective GA variant, NSGA-II [18] to improve the convergency and efficiency of GDSII-Guard flow optimizer. We also implement a process-level parallelism to fundamentally speed up the parameter space exploration.

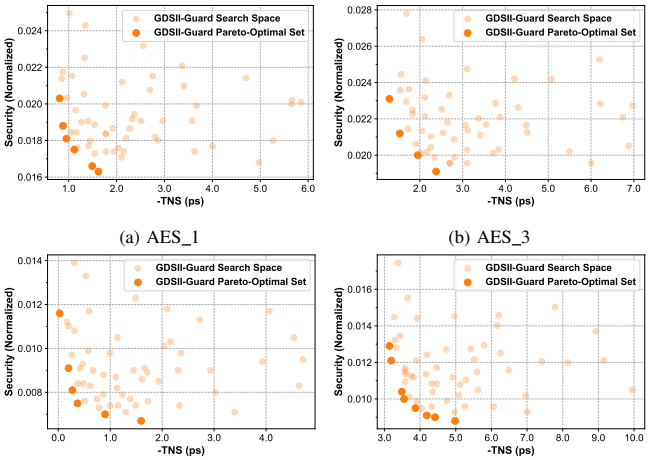
Multi-Objective Model Parameter. We determine the values of parameters in the multi-objective model (§II-C) according to the realistic IC process and actual Trojan conditions. Taken from A2 Trojan [3], [14], we set the value of `Thresh_ER` as 20. We assign $\alpha = 0.5$, $N_{\text{DRC}} = 20$, and $\beta_{\text{power}} = 1.2$ such that: 1) we equally weight free sites and tracks for security evaluation, and 2) we tolerate minor DRC/power degradation, which can further be manually fixed.

Benchmarks and Libraries. We utilize the benchmarks and security metrics in [14] to evaluate our defensive approaches. The benchmark includes physical designs with varied complexity, size, utilization and timing constraints. The designs contain crypto cores and microprocessors. Each design is attached with a list of security-critical cell assets and SDC, MMMC files for timing constraints. The technology library (LIB/LEF) we use is Nangate 45nm Open Cell Library [19], which has 10 available metal layers ($K = 10$).

Baselines. We apply GDSII-Guard defensive optimizations on finalized layout designs, observing the security improvements and extra overheads. Besides, we compare GDSII-Guard with three state-of-the-art, design-time, layout-level defenses: ICAS [10], BISA [11], and Ba *et al.* [12], [13].

B. Explored Pareto Fronts

We record the process of GDSII-Guard parameter space exploration. Fig. 5 illustrates the search space and the explored Pareto fronts of on designs AES_1, AES_3, MISTY, and openMSP430_2. The security value is normalized with the baseline design as described in §II-C. The model has converged within a few iterations, indicated by the growing point density near the explored Pareto front. The results show that we can derive the Pareto-optimal parameter set of GDSII-Guard by sampling and evaluating a modest number of defensive schemes. Moreover, we observe the diversity of Pareto-optimal solutions given that few clusters are formed. GDSII-Guard explores the Pareto front efficiently with short runtime, thanks to both the moderate size of the actual search space and the multi-processing techniques. Eventually, we obtain good trade-offs between security and timing.



(a) AES_1 (b) AES_3

(c) MISTY (d) openMSP430_2

Fig. 5: Explored Pareto fronts with GDSII-Guard on 4 designs.

C. Comparison of Optimized Physical Designs

For each design in the benchmark, we select a Pareto solution from the Pareto-optimal set found by GDSII-Guard. They we compare its security metrics, performance and design cost against other defensive measures and study the security-timing trade-offs of these measures.

Security Analysis. Fig. 4 demonstrates the comparison of the two security metrics normalized with the baseline design, *i.e.,* normalized total free sites/tracks are the division of actual total free sites/tracks of the optimized design by the original. The average remaining free sites over all designs after optimizations by ICAS [10], BISA [11], Ba *et al.* [12], [13], and GDSII-Guard are 10.7%, 1.6%, 6%, **1.3%**, while the average free tracks are 10.6%,1.4%, 5.8%, **1.1%**, respectively. The undirected CAD parameter tuning of ICAS proves inadequate to effectively eliminate vulnerable regions. BISA fills the layout with extra logic ubiquitously and reduces most of Trojan-available resources, though at the expense of performance and design quality (discussed later). Ba *et al.* advanced BISA by appending the logic locally near security-critical cells while the defensive coverage is discounted. GDSII-Guard successfully lowers the risk of Trojan insertion by 98.8% on average via dedicated management of empty spaces on the layout without driving local or global placement density extremely high. Moreover, the normalized free routing tracks are 15%

TABLE II: Comparison of timing (TNS), power, and #DRC violations.

TNS(ps)	AES_1	AES_2	AES_3	Camellia	CAST	MISTY	openMSP430_1	openMSP430_2	PRESENT	SEED	SPARX	TDEA
Original Design	-0.998	-2.577	-1.432	0	-6.693	0	0	-2.946	0	-6.693	0	0
ICAS [10]	-1.657	-2.737	-3.356	0	-7.73	-0.414	0	-4.281	0	-8.025	0	-0.012
BISA [11]	-4.256	-9.731	-8.367	-1.23	-25.324	-4.257	-1.582	-7.768	0	-21.205	-1.432	-2.87
Ba *et al.* [12], [13]	-1.818	-3.47	-2.285	0	-10.589	-0.356	-0.021	-4.875	0	-8.924	0	-0.56
GDSII-Guard	-1.116	-2.893	-1.954	0	-8.035	-0.371	0	-3.548	0	-5.978	0	0

Power(mW)	AES_1	AES_2	AES_3	Camellia	CAST	MISTY	openMSP430_1	openMSP430_2	PRESENT	SEED	SPARX	TDEA
Original Design	66.667	68.906	67.72	1.691	4.596	3.302	0.375	1.161	0.377	4.596	2.252	1.482
ICAS [10]	69.807	70.092	73.863	1.634	6.274	3.141	0.372	1.186	0.41	4.615	2.253	1.458
BISA [11]	81.752	91.424	84.35	2.554	9.124	5.848	0.473	2.069	0.483	6.172	3.065	1.927
Ba *et al.* [12], [13]	75.403	77.38	74.583	2.104	5.973	3.954	0.388	1.536	0.434	4.892	2.266	1.503
GDSII-Guard	71.874	72.782	70.548	1.812	5.168	3.893	0.394	1.214	0.395	4.678	2.249	1.533

#DRC	AES_1	AES_2	AES_3	Camellia	CAST	MISTY	openMSP430_1	openMSP430_2	PRESENT	SEED	SPARX	TDEA
Original Design	0	12	0	0	0	0	0	0	0	0	0	0
ICAS [10]	0	15	0	0	0	0	0	0	0	9	0	0
BISA [11]	11	57	5	3	45	0	0	18	0	24	0	13
Ba *et al.* [12], [13]	5	41	0	0	19	0	0	3	0	11	0	0
GDSII-Guard	0	16	0	0	3	0	0	0	0	0	0	0

less than the site counterpart, which would have been equal without Routing Width Scaling since they are all proportional to the area of exploitable regions. Therefore, anti-Trojan ECO routing further reduces available routing resources on top of ECO-placement.

Performance, Power and DRC Analysis. As shown in Table II, GDSII-Guard demonstrates the minimal overall timing, power, and design quality degradation among all the defenses. The increase in TNS, power consumption, and DRC errors is within an acceptable range. BISA and Ba *et al.* suffer from notable power overheads because they append much additive logic to the original design. As for timing and DRC, although Ba *et al.* improves BISA by prioritizing the empty spaces and decreasing the utilization rate, it still requires 90% local density at least, which inevitably causes routing congestion, timing problems, and design rule violations. GDSII-Guard is the first to optimize security and timing simultaneously while preserving power and design quality. With our PPA-friendly ECO P&R techniques and multi-objective optimization model, GDSII-Guard strikes a good balance between security and timing.

D. Comparison of Runtime

The runtime of GDSII-Guard is closely related to the design size, varying from minutes to hours. Taking the largest design in the benchmark, AES_2 as an example, ICAS, BISA, Ba *et al.*, and GDSII-Guard takes 9.4, 6.5, 7.0, **4.8** hours respectively to complete the optimization. ICAS performs the time-consuming global P&R and exhaustive parameter tuning. BISA and Ba *et al.* spend much time on synthesizing the introduced logic and P&R at a high density. GDSII-Guard is based on efficient ECO incremental operators and utilizes multi-processing and pruning to minimize the GA rounds. Accordingly, the runtime of GDSII-Guard remains relatively short, given complex designs.

V. Conclusions

In this paper, we propose an anti-Trojan, timing-aware ECO framework to harden physical layouts against malicious hardware Trojan while minimizing the side effects brought by security measures, *e.g.,* performance downgrading, DRC violations, and power increase. We develop several anti-Trojan methodologies at ECO P&R stage and design a multi-objective optimization model to explore the security-timing trade-offs in the flow parameter space. Experiments show that GDSII-Guard significantly improves security while preserving timing, power and design quality compared with the prior art. We expect to see booming researches in the hardware security community to further explore the coverage metrics and countermeasures of hardware Trojan or other extensive security threats.

Acknowledgment

This work was partly supported by National Key R&D Program of China (Grant No. 2022YFB4500500), National Natural Science Foundation of China (Grant No. 62090021), and DeePoly Technology Inc.

References

[1] G. T. Becker *et al.*, "Stealthy dopant-level hardware Trojans: extended version," *J. Cryptogr. Eng.*, vol. 4, no. 1, pp. 19–31, 2014.

[2] R. Kumar *et al.*, "Parametric Trojans for fault-injection attacks on cryptographic hardware," in *FDTC*, 2014.

[3] K. Yang *et al.*, "A2: analog malicious hardware," in *IEEE Symposium on Security and Privacy*, 2016.

[4] R. Karri *et al.*, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[5] R. S. Chakraborty *et al.*, "MERO: A statistical approach for hardware trojan detection," in *CHES*, 2009.

[6] M. Banga *et al.*, "Guided test generation for isolation and detection of embedded Trojans in ICs," in *GLSVLSI*, 2008.

[7] Y. Alkabani *et al.*, "Consistency-based characterization for IC trojan detection," in *ICCAD*, 2009.

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

[9] D. Rai *et al.*, "Performance of delay-based Trojan detection techniques under parameter variations," in *HOST*, 2009.

[10] T. Trippel *et al.*, "ICAS: an extensible framework for estimating the susceptibility of IC layouts to additive Trojans," in *IEEE Symposium on Security and Privacy*, 2020.

[11] K. Xiao *et al.*, "BISA: built-in self-authentication for preventing hardware trojan insertion," in *HOST*, 2013.

[12] P. Ba *et al.*, "Hardware Trojan prevention using layout-level design approach," in *ECCTD*, 2015, pp. 1–4.

[13] P. Ba *et al.*, "Hardware trust through layout filling: A hardware Trojan prevention technique," in *ISVLSI*, 2016.

[14] J. Knechtel *et al.*, "Benchmarking security closure of physical layouts: ISPD 2022 contest," in *ISPD*, 2022.

[15] H. Salmani *et al.*, "Vulnerability analysis of a circuit layout to hardware trojan insertion," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 6, pp. 1214–1225, 2016.

[16] J. Knechtel *et al.*, "Security closure of physical layouts ICCAD special session paper," in *ICCAD*, 2021.

[17] A. Kahng *et al.*, *VLSI physical design: from graph partitioning to timing closure*. Springer, 2011, vol. 312.

[18] K. Deb *et al.*, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[19] Nangate Inc 2011, "Nangate FreePDK45 Open Cell Library," http://www.nangate.com.