

HIPR: Hardware IP Protection through Low-Overhead Fine-Grain Redaction

Aritra Dasgupta[✉], Sudipta Paria[✉] and Swarup Bhunia[✉]

University of Florida, Gainesville, FL 32611, USA

aritradasgupta@ufl.edu, sudiptaparia@ufl.edu, swarup@ece.ufl.edu

Abstract.

Hardware IP blocks have been subjected to various forms of confidentiality and integrity attacks in recent years due to the globalization of the semiconductor industry. System-on-chip (SoC) designers are now considering a zero-trust model for security, where an IP can be attacked at any stage of the manufacturing process for piracy, cloning, overproduction, or malicious alterations. Hardware redaction has emerged as a promising countermeasure to thwart confidentiality and integrity attacks by untrusted entities in the globally distributed supply chain. However, existing redaction techniques provide this security at high overhead costs, making them unsuitable for real-world implementation. In this paper, we propose **HIPR**, a fine-grain redaction methodology that is robust, scalable, and incurs significantly lower overhead compared to existing redaction techniques. **HIPR** redacts security-critical Boolean and sequential logic from the hardware design, performs interconnect randomization, and employs multiple overhead optimization steps to reduce overhead costs. We evaluate **HIPR** on open-source benchmarks and reduce area overheads by 1 to 2 orders of magnitude compared to state-of-the-art redaction techniques without compromising security. We also demonstrate that the redaction performed by **HIPR** is resilient against conventional functional and structural attacks on hardware IPs. The redacted test IPs used to evaluate **HIPR** are available at: <https://github.com/UF-NeIms-IoT-Git-Projects/HIPR>.

Keywords: Hardware IP Protection · Programmable Logic · Hardware Redaction · Reverse Engineering · IP Piracy

1 Introduction

The ever-rising demand for integrated circuits (IC) and the globalization of the manufacturing supply chain have led to the outsourcing of critical steps of the semiconductor design process to untrusted facilities abroad. As a result, it has become vital for designers to assume a Zero Trust model for security, where there are no trusted entities involved anywhere during the life cycle of a hardware intellectual property (IP). Under the Zero Trust Model, the IP can be vulnerable to threats like piracy, counterfeiting, over-production, reverse engineering, functionality corruption, or leakage of sensitive information via hardware Trojan insertion [BT18], as shown in Fig. 1. Hence, it is critical to protect the confidentiality and integrity of the IP against any such malicious intent.

In recent years, hardware IP protection has emerged as a critical area of research, with numerous methods being proposed at different abstraction levels. Logic Locking [RKM08, CB09] emerged as one of the prominent IP protection methodologies where designers insert additional key inputs to thwart reverse engineering. However, these techniques are found to be vulnerable against Boolean satisfiability (SAT)-based key extraction attack [SRM15, SLM⁺17]. Provably secure logic locking techniques [XS19, YSN⁺17, AB21]

were developed to address the SAT attack, but have been shown to be susceptible to attacks that leverage structural analysis and machine learning (ML) [CCB18, AFB19, APK⁺21]. IC Camouflaging techniques [LSM⁺19, RSK13] were proposed to protect ICs from reverse engineering and unauthorized access but suffer from high overhead, increased resource requirements, limited flexibility, and vulnerability to advanced attacks [YMSR20].

Hardware IP redaction using embedded FPGA (eFPGA) is a recent technique that aims to implement selected modules with synthesizable eFPGA fabric in the design [MAS⁺21, BTMT⁺21, TC⁺22, KAFT23]. This is a coarse-grain redaction technique in which the security-critical modules are selected for redaction and implemented using eFPGA fabric using open-source frameworks like [TGA⁺19, KDH⁺21]. The programmable bitstream serves as the secret key, requiring the adversary to restore it fully so that the whole system with the eFPGA fabric can function properly. The security analysis of eFPGA redaction shows the resilience to SAT attacks. However, they incur significantly high area overhead, making them less suitable for practical implementation. LUT-based obfuscation [MKZAG⁺18, KPR⁺19] and combined approach with redaction [RAD⁺24] provide an alternative solution that uses MUXes and/or LUTs to selectively replace the critical logic to protect against RE attacks. EvoLUTe [GRK⁺23] was proposed as a fine-grain IP redaction method that efficiently selects sub-circuits and replaces them using LUTs providing SAT resiliency. Both eFPGA redaction and fine-grain LUT-based redaction techniques incur significant overhead and offer limited flexibility. Furthermore, they do not optimize the bitstream to reduce power, performance, and area (PPA) overheads, and may be susceptible to more sophisticated structural and ML-based attacks [AHT⁺20, APK⁺21, DPS⁺25].

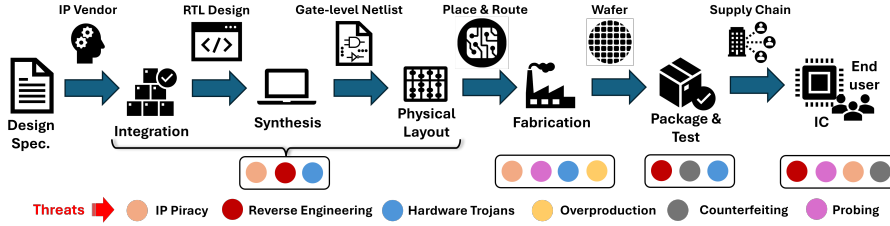


Figure 1: Various threats encountered in the IC design flow.

In this paper, we analyze the scope of fine-grain redaction of logic in order to provide protections to sensitive IPs in hardware. We explore ways to introduce FPGA-like programmability into a design in the ASIC flow. A generalized flow to achieve this goal is to remove critical logic cones from the design and replace them with programmable fabric that can be configured using a functionally equivalent bitstream to the redacted logic cones. We introduce **HIPR**, a novel low-overhead countermeasure for **H**ardware **I**P protection through fine-grain **R**edaction in ASICs. The major contributions in this paper are listed below:

- We propose **HIPR**, a novel and robust countermeasure against confidentiality and integrity attacks on hardware IP in ASICs.
- We introduce a fine-grain redaction methodology for replacing security-critical logic cones identified from the gate-level netlist of a hardware IP.
- We present a custom standard-cell-based configurable fabric for redaction of Boolean and sequential logic functions along with interconnect randomization.
- We implement overhead optimization mechanisms to satisfy design constraints without compromising security and reduce overhead costs.
- We describe how **HIPR** can be easily integrated into the commercial EDA tool flow for IC design.

The remainder of the paper is organized as follows. In Section 2, we briefly discuss the threat model and provide relevant background on existing redaction techniques and attacks. The major steps of **HIPR** methodology are described in Section 3. In Section 4, we present the implementation results of **HIPR** on open-source benchmarks along with the security analysis of **HIPR** against functional and structural attacks and a comparative analysis with other redaction techniques. Section 5 concludes the paper.

2 Background

2.1 Anti-RE solution landscape

The landscape of Anti-RE solutions for IP security includes various countermeasures, which can be categorized into three main types: obfuscation, camouflaging, and redaction, as depicted in Fig. 2a. Obfuscation techniques, which include both combinational and sequential methods, aim to hide the original functionality of the design and employ key-based validation to protect against unauthorized access. Camouflaging utilizes strategies like logic perturbation and split manufacturing to create variations without impacting the original functionality, with distributed manufacturing making it more challenging for adversaries to analyze and replicate the design. Redaction techniques can be divided into fine-grain (LUT-based) and coarse-grain (eFPGA-based) methods as depicted in Fig. 2b, enabling the selective concealment of critical components to prevent unauthorized access. These techniques utilize reconfigurable fabric to implement the intended function configured by a protected bitstream. This allows the design to be manufactured by any untrusted offshore foundry in a zero-trust environment without access to the bitstream, thereby protecting the overall design against reverse engineering.

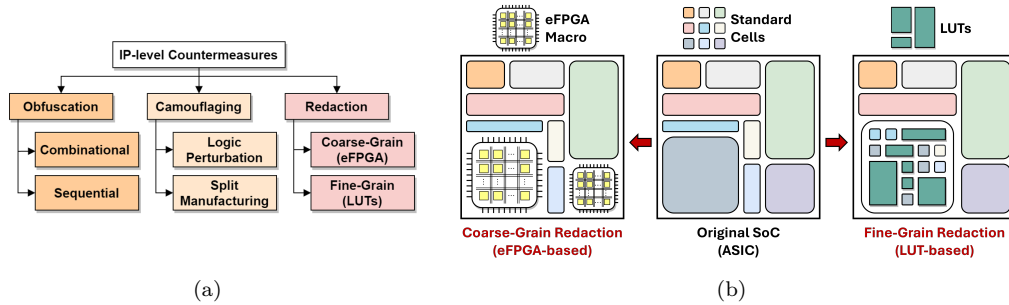


Figure 2: Overview of Anti-RE solution landscape: (a) Taxonomy of various IP-level protection schemes; (b) Comparing coarse-grain (eFPGA-based) vs. fine-grain (LUT-based) redaction techniques.

2.2 Related Works

2.2.1 LUT-based Fine-Grain Redaction

LUT-based IP-level countermeasures protect security-critical designs from unauthorized access and reverse engineering by selectively replacing sensitive logic using programmable LUTs. Several researchers proposed methodologies to implement LUT-based obfuscation, showing that it is resistant to existing attacks on hardware IP. [MKZAG⁺18] proposed LUT-Lock, a LUT-based obfuscation technique that operates at the gate level and employs various key features, such as the fan-in cone of primary outputs, higher skew gates, minimum fan-out gates, etc., and provides resistance to SAT attacks. Custom-LUT [KPR⁺19] utilizes routing obfuscation through MUXes along with logic obfuscation through LUTs to improve

security against SAT attack [SRM15], side-channel analysis (SCA) [Dev24], and removal attack [YMSR20]. The authors in [KAHS19] proposed SAT-hard obfuscation solutions using a set of fully programmable logic and routing blocks (PLR) networks for logic replacement in designs. EvoLUTe [GRK⁺23] offers a fine-grain LUT-based redaction approach that utilizes a graph-based analysis to effectively select candidate subcircuits for redaction and ensures SAT resiliency. Existing solutions offer fine-grain redaction that is secure against SAT attack [SRM15], however, they have not been evaluated against structural attack methodologies [AHT⁺20, DPS⁺25]. Additionally, these solutions incur substantial overhead even for partial design redaction and lack constraint-driven security-aware bitstream optimization to improve PPA overheads.

2.2.2 eFPGA-based Coarse-Grain Redaction

Due to their flexibility and adaptability, embedded field-programmable gate arrays (eFPGAs) have been adopted in heterogeneous computing in recent years. eFPGAs feature Configurable Logic Blocks (CLBs) that include LUTs, flip-flops (FFs), and routing logic, enabling them to be reconfigured for intended functions. [MAS⁺21] proposed a coarse-grain approach that replaces security-critical IP blocks with a synthesizable eFPGA fabric. [BTMT⁺21] explored the feasibility and practical issues of eFPGA-based redaction on open-source IPs and performed security analysis towards SAT attack resilience. [TC⁺22] proposed a design flow that partitions the modules at the register-transfer-level (RTL) between one or more reconfigurable fabrics, automating the generation of redacted designs with comparable area/time/power overheads. [KAFT23] presented a framework for reconstructing and shrinking selected eFPGA fabric customized for specific IP protection with acceptable overheads. Current solutions for eFPGA-based redaction are constrained by the choice of fabric sizes and architectures, and they incur significant PPA overhead even when redacting only a small portion of the overall logic. Although these solutions have been shown to be resistant to SAT attack [SRM15], they have not been evaluated against structural attacks [AHT⁺20, DPS⁺25], or removal attacks [YMSR20].

eFPGA Frameworks OpenFPGA [TGA⁺19] provides an open-source framework that facilitates the design and implementation of eFPGAs. It enables rapid prototyping of custom FPGA architectures through a semi-custom design approach. FABulous [KDH⁺21] introduces an embedded open-source FPGA framework that combines several open-source tools to integrate a full eFPGA framework and provides the eFPGA netlist and constraints generation, eFPGA CAD tools access, and emulation features.

2.2.3 Existing Attacks on Hardware Redaction

SAT attack [SRM15] uses SAT solvers to analyze and potentially exploit vulnerabilities in the design by inferring the correct values of the key inputs using only a small number of input/output observations. It has been demonstrated that SAT attacks could break many primitive locking mechanisms in nearly polynomial time, making them relevant for redaction techniques as well. Satisfiability Modulo Theory (SMT)-based attack [AKHS18] that expands the capabilities of SAT attack by assigning theory solvers to monitor the behavioral and non-functional properties of the obfuscated circuit. Break & Unroll [RAM22] was attack can recover the bitstream of state-of-the-art eFPGA-based redaction schemes with the existence of hard cycles and large size keys in a relatively short time. [BMT⁺23] proposed SAT-based attack for bitstream recovery of eFPGAs using IcySAT [SPJ19] by incrementally executing for different unroll factors. FuncTeller [HSD⁺23] provides a heuristic attack that extracts the approximate functionality of redacted hardware IP implemented on eFPGA with high accuracy within a reasonable time. ML-based attacks [CCB18, APSS22] leverage different machine-learning techniques to predict the

values of the key gates, which can then be used to unlock an obfuscated design; this approach is also applicable to redacted designs. MANTIS [SMS23] follows a combined approach that uses SAT/SMT solvers to obtain training data for generating the synthesizable predictive model, which can replicate and substitute the exact logic mapped onto the eFPGA to circumvent hardware redaction mechanisms. Attacks on existing hardware redaction techniques reveal their vulnerabilities, highlighting the need for a comprehensive redaction approach that can effectively address both structural and functional attacks.

2.2.4 Motivation for HIPR

Contemporary hardware IP redaction techniques face several challenges that make them unsuitable for real-world applications. Coarse-grain redaction techniques [MAS⁺21, TC⁺22, KAFT23] are limited in their ability to analyze and secure critical logic cones due to the RTL abstraction and fixed I/O size options. These techniques also rely on eFPGA fabrics [TGA⁺19, KDH⁺21] for implementing the redacted logic, which can incur extremely high overheads even for very small designs. Moreover, due to static routing outside and around the eFPGA instances, the redacted IPs can be vulnerable to structural analysis and removal attacks [AHT⁺20, DPS⁺25, YMSR20]. LUT-based obfuscation and redaction techniques [MKZAG⁺18, KPR⁺19, GRK⁺23] are promising since they allow for more fine-grain control over the candidate logic to be redacted but can still incur significant overheads for large designs as the bitstream size increases, leaving room for optimizations for overhead reduction. To the best of our knowledge, there are no hardware redaction frameworks that allow true fine-grain control over candidate IP blocks, nor do they offer the option to optimize the configuration bitstream under overhead constraints while satisfying the security requirements, if any. Additionally, attacks on hardware redaction techniques in recent literature and security evaluation mainly focus on *oracle*-guided functional attacks [SRM15], leading to the potential vulnerabilities against other known or unknown attack vectors not being explored. Table 1 provides a summary of existing hardware redaction techniques, highlighting their features and vulnerabilities compared to **HIPR**.

Table 1: Overview of existing hardware redaction techniques in literature.

Method	Type	Component /Framework	Automated Tool-Flow proposed?	Abstraction level	Structural/ Functional Attack Resistant?	Bitstream Optimization?
eFPGA ₁ [MAS ⁺ 21]	Coarse-grain	OpenFPGA	No	RTL	SAT [SRM15]	✗
eFPGA ₂ [BTMT ⁺ 21]	Coarse-grain	OpenFPGA	No	RTL	SAT [SRM15]	✗
ALICE [TC ⁺ 22]	Coarse-grain	OpenFPGA	Yes	RTL	-	✗
SheLL [KAFT23]	Coarse-grain	FABulous	Yes	RTL	SAT [SRM15]	✗
LUT-Lock [MKZAG ⁺ 18]	Fine-grain	LUT	Partial	Gate-level	SAT [SRM15]	✗
Custom-LUT [KPR ⁺ 19]	Fine-grain	MUX, LUT	Partial	Gate-level	SAT [SRM15], Removal Attack [YMSR20]	✗
EvoLUTe [GRK ⁺ 23]	Fine-grain	LUT	Yes	RTL Gate-level	SAT [SRM15]	✗
HIPR* (This work)	Fine-grain	LUT	Yes	RTL Gate-level	SAT [SRM15], DANA [AHT ⁺ 20], Structural Attack	✓

Threat Model Existing solutions for hardware IP redaction [MKZAG⁺18, KPR⁺19, TC⁺22, KAFT23, GRK⁺23] including the state-of-the-art techniques, primarily consider *oracle*-guided attack vectors [SRM15, AKHS18, SPJ19] similar to logic locking [KAFT22]. For a fair and comprehensive security evaluation of **HIPR**, the proposed fine-grain redaction approach for hardware IP protection under the Zero Trust Model, we adopt the *oracle*-guided threat model from previous work and extend it to incorporate *oracle*-less attacks

that leverage structural analysis [AHT⁺20]. Physical attacks, such as microprobing and SCA [TLSB17, RTR⁺20, Dev24] are significant threats against bitstream protection in programmable fabrics like FPGA, and the proposed redaction technique can also be vulnerable to similar attacks in the presence of untrusted foundry. However, in this work, we restrict our security analysis to *oracle*-guided functional attacks and *oracle*-less structural attacks, consistent with the body of prior work on hardware IP redaction [MKZAG⁺18, KPR⁺19, TC⁺22, KAFT23, GRK⁺23].

- **Assets:** RTL or gate-level hardware IPs (with associated design files).
- **Adversary:** The design house responsible for integration, synthesis, and physical layout, untrusted fabrication, untrusted packaging & testing facility, as well as untrusted end-users (post-deployment).
- **Adversarial Access:** The attackers have access to the redacted design and a functional *oracle*, i.e., unlocked design with full scan access. They can observe input/output values by applying test vectors for *oracle*-guided attacks. They can apply structural-analysis-based *oracle*-less attacks of the redacted gate-level netlist.
- **Adversarial Objectives:** To retrieve the correct bitstream to restore the true functionality and extract the design secrets through RE.
- **Trust Model:** The IP vendor/owner/designer is considered trustworthy.

3 Proposed Methodology

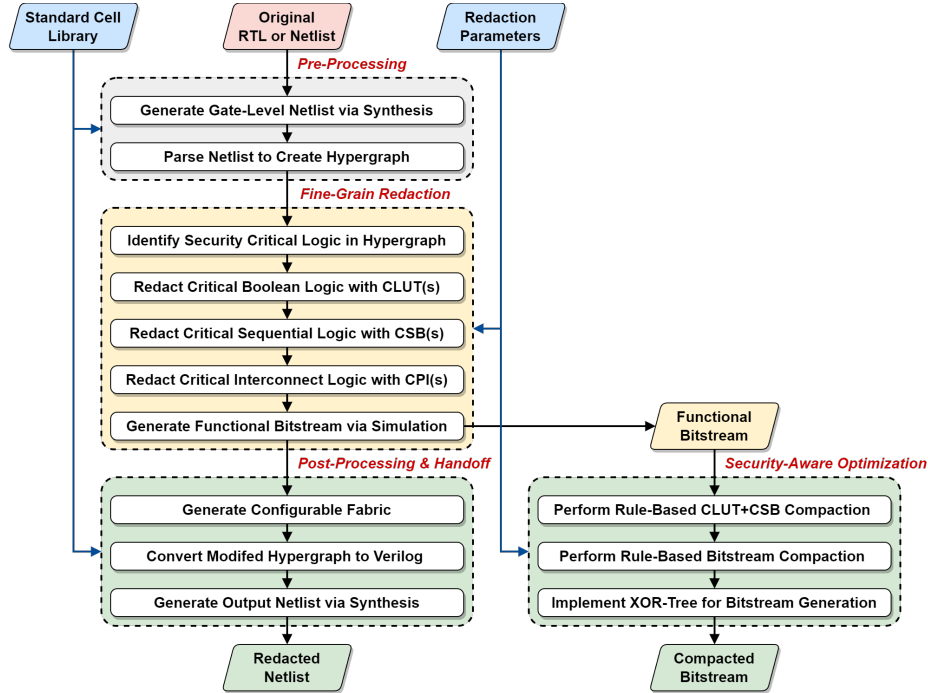


Figure 3: Major steps in HIPR tool flow.

HIPR performs an initial synthesis step to convert the input designs into a gate-level netlist. By default, **HIPR** targets complete Boolean and sequential logic redaction and aims to maximize interconnect randomization for a given design. However, it can also be configured to perform partial redaction with fine-grain precision targeting only the security-critical logic in the netlist. **HIPR** employs a *greedy heuristic* to map the identified critical logic cones onto the configurable fabric, followed by the generation of the programming

bitstream for each component utilized for redaction. The configurable fabric is inserted into the remaining design, and a final synthesis step generates the redacted netlist. The security-aware bitstream optimization stage reduces the functional bitstream size to a more usable size without compromising security. Algorithm 1 outlines the fine-grain redaction methodology in **HIPR** and security-aware overhead optimizations. The major steps in the proposed **HIPR** tool flow are depicted in Fig. 3.

Algorithm 1: HIPR: Fine-Grain Redaction Methodology

Input : Original Netlist (N), Random Seed (θ), Redaction Parameters (γ)
Output : Redacted Netlist (N_{mod}), Compacted Bitstream ($\text{cfg_bitstream}_{compact}$)

```

1  $\mathcal{G}_{org} \leftarrow \text{create\_hypergraph}(N)$ 
2  $\mathcal{G}_{sorted} \leftarrow \text{topological\_sort}(\mathcal{G}_{org})$ 
  /* Identification of Critical Logic */
3  $\mathcal{E}_{crit} \leftarrow \text{identify\_critical\_edges}(\mathcal{G}_{sorted}, \theta, \gamma)$ 
  /* Fine Grain Redaction */
4  $\text{CLUT} \leftarrow \text{redact\_boolean\_logic}(\mathcal{G}_{sorted}, \mathcal{E}_{crit}, \theta, \gamma)$ 
5  $\mathcal{G}_{mod} \leftarrow \mathcal{G}_{sorted} \cup \text{CLUT}$ 
6  $\text{CSB} \leftarrow \text{redact\_sequential\_logic}(\mathcal{G}_{mod}, \theta, \gamma)$ 
7  $\mathcal{G}_{mod} \leftarrow \mathcal{G}_{mod} \cup \text{CSB}$ 
8  $\text{CPI} \leftarrow \text{redact\_interconnect\_logic}(\mathcal{G}_{mod}, \theta, \gamma)$ 
9  $\mathcal{G}_{mod} \leftarrow \mathcal{G}_{mod} \cup \text{CPI}$ 
  /* Generate Configuration Bitstream */
10  $\text{cfg\_bitstream} \leftarrow \text{generate\_bitstream}(\mathcal{G}_{mod}, \theta)$ 
  /* Security Aware Overhead Optimization */
11  $\text{cfg\_bitstream}_{compact} \leftarrow \text{compaction}(\mathcal{G}_{mod}, \text{cfg\_bitstream}, \gamma)$ 
12  $N_{mod} \leftarrow \text{convert\_hypergraph\_to\_netlist}(\mathcal{G}_{mod})$ 
13 return  $N_{mod}, \text{cfg\_bitstream}_{compact}$ 

```

3.1 Fine-Grain Redaction

3.1.1 Configurable Fabric in HIPR

The configurable fabric in **HIPR** utilizes the following components to implement the redacted logic:

- **Configurable Look-Up Tables (CLUTs)** are used to redact Boolean logic cones.
- **Configurable Sequential Blocks (CSBs)** are used to realize sequential logic redacted from the designs and dummy sequential elements. Each CSB also contains a CLUT, which can implement existing Boolean logic or introduce dummy logic.
- **Configurable Programmable Interconnects (CPIs)** are used for interconnect randomization at strategic locations.

Conventional logic gates from a standard cell library (open-source or commercial) are not ideal for implementing the configurable fabric in **HIPR**. Custom-designed standard cells for CLUTs/CSBs/CPIs can help significantly reduce overheads and improve the performance of the redacted logic in an ASIC design flow. Fig. 4 demonstrates the building blocks of the configurable fabric used to realize the redacted logic in **HIPR**.

3.1.2 Identification of Security Critical Logic

The gate-level netlist of a hardware IP can be represented as a Boolean network in the form of a hypergraph [KAV04], where the logic gates/FFs in the design become nodes (vertices), and the wires connecting these nodes become its hyperedges. The original hypergraph ($\mathcal{G}_{org}$) is topologically sorted before the security-critical logic cones to be redacted are identified using a design-independent removal cost function (rcf) and encapsulated as the set of critical hyperedges $\mathcal{E}_{crit}$. The removal cost (κ) is determined for every hyperedge in $\mathcal{G}_{org}$ using stochastic properties, such as, Shannon entropy ($\mathbb{H}$) and structural information such as fan-in ($\mathbb{FI}$) or fan-out cone ($\mathbb{FO}$) sizes:

$$\kappa = rcf(\mathbb{H}, \mathbb{FI}, \mathbb{FO}) \quad (1)$$

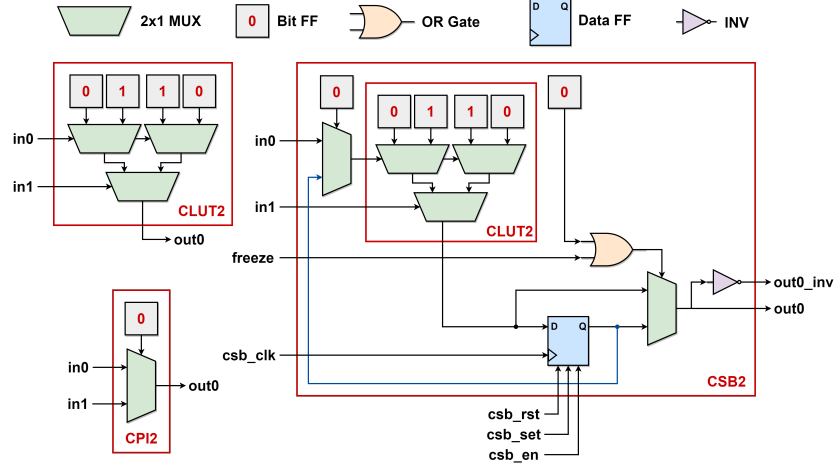


Figure 4: Configurable fabric used for redaction in **HIPR**. The schematics illustrate how to construct CLUT2/CSB2/CPI2 using common gates from a typical standard cell library. For efficient implementation of redacted logic in the ASIC design flow, the components can be designed as custom standard cells to leverage transistor-level optimizations.

The hyperedges in $\mathcal{G}_{org}$ whose normalized removal cost $\hat{k} \geq \mathcal{T}_{crit}$ are identified as the critical hyperedges that constitute $\mathcal{E}_{crit}$, where $\mathcal{T}_{crit}$ is a configurable parameter in γ that provides fine-grain control over redaction in **HIPR**. By default, $\mathcal{T}_{crit}$ is set to 0 and **HIPR** attempts to redact the entire hardware IP.

After identifying the security-critical logic cones, **HIPR** employs a heuristic-based search to determine the optimal CLUT mapping for the Boolean logic to be redacted. Limiting the target library to a set of basic gates, AND2, OR2, INV, during initial synthesis results in an AND-OR-INV hypergraph (AOIG), which increases CLUT coverage as larger logic cones are redacted by each CLUT [CD94, MCB07] thereby helping reduce area overheads incurred during redaction. In addition to increased CLUT coverage, the AOIG permits the use of standard graph algorithms for analyzing design characteristics.

3.1.3 Fine-Grain Redaction Process

The fine-grain logic redaction process in **HIPR** involves Boolean and sequential logic redaction along with interconnect randomization using a configurable fabric consisting of CLUTs/CSBs/CPIs. We provide a brief description of each step below:

Boolean Logic Redaction using CLUTs Algorithm 2 outlines all the steps involved in Boolean logic redaction using CLUTs in **HIPR**.

- **Determine MFFC:** The maximum fan-out free cone (MFFC) [MCB07] is determined for each critical hyperedge in $\mathcal{E}_{crit}$ via graph traversal from the AOIG.
- **Enumerate cuts:** The MFFC obtained for each critical hyperedge is recursively analyzed to obtain the locally optimal cuts (in terms of area and delay) which can be mapped to an Nx1 CLUT. The enumerated cuts for each node are stored in $\mathbf{G}_{cuts}$.
- **Global cut selection:** $\mathbf{G}_{cuts}$ is iteratively traversed using a greedy heuristic to select the global optimal cut from the available cuts for all critical hyperedges in $\mathcal{E}_{crit}$ for AOIG. Thresholds used to select the cuts are determined from the normalized area and delay costs of each cut and are incremented after every iteration until a pre-defined security target specified in γ is met. The selected cuts are then mapped to the corresponding Nx1 CLUTs of the same size N. The CLUTs obtained in CLUT are ranked in the order of a proximity metric, calculated from the relative distance of the logic replaced by these CLUTs in the AOIG. This step guarantees that the CLUTs inserted in the design are chained in a manner that does not violate synthesis constraints.

Algorithm 2: Boolean Logic Redaction using CLUTs in **HIPR**.

Input : $\mathcal{G}_{sorted}, \mathcal{E}_{crit}, \theta, \gamma$
Output : List of CLUTs (CLUT)

```

1 CLUT  $\leftarrow \emptyset$ 
2  $\mathbf{G}_{cuts} \leftarrow \emptyset$   $\triangleleft$  Global Cuts 2D List
3 foreach  $e \in \mathcal{E}_{crit}$  do
4    $e_{MFFC} \leftarrow \text{determine\_MFFC}(\mathcal{G}_{sorted}, e)$   $\triangleleft$  Maximum Fan-out Free Cone
5    $random\_lut\_size \leftarrow \gamma_{min\_size} + \text{random}(\theta) \times (\gamma_{max\_size} - \gamma_{min\_size} + 1)$ 
6    $e_{CUT\_LIST} \leftarrow \text{enumerate\_cuts\_MFFC}(e_{MFFC}, random\_lut\_size)$ 
7    $\mathbf{G}_{cuts}.\text{append}(e_{CUT\_LIST})$ 
/* Global CLUT Mapping */
8 foreach  $\zeta_{list} \in \mathbf{G}_{cuts}$  do
9   foreach  $\zeta \in \zeta_{list}$  do
10    if  $is\_suitable\_clut(\zeta)$  and  $is\_redactable(\mathcal{G}_{org}, \zeta)$  then
11      CLUT. $\text{append}(\text{generate\_clut\_from\_cut}(\zeta))$ 
12 return CLUT

```

CSB Insertion Redacting critical sequential logic in the design can prevent logic equivalence and structural-analysis-based attacks that try to extract the original design functionality and state space. To prevent such attacks, the sequential logic elements can be redacted using CSBs. Each CSB block contains a CLUT, a FF, and I/O muxes, as depicted in Fig. 4. In **HIPR**, CLUTs with a FF at its output are randomly selected to convert into a CSB to hide the FF. In addition, the sequential elements of the design can be directly converted into a CSB by adding a dummy CLUT at the input of the FF. The amount of FFs to be redacted using CSBs can be configured up to 100% of the sequential logic in the design. Algorithm 3 presents all the steps necessary for performing critical sequential logic redaction using CSBs in **HIPR**.

Algorithm 3: Sequential Logic Redaction using CSBs in **HIPR**.

Input : $\mathcal{G}_{org}, \theta, \gamma$
Output : List of CSBs (CSB)

```

1 CSB  $\leftarrow \emptyset$ 
/* Randomized CLUT to CSB Conversion */
2  $n_X \leftarrow (\text{random}(\theta) \% \gamma_{x\_max} \times |\text{CLUT}|)$ 
3 while  $n_X > 0$  do
4    $\text{CLUT}_{inst} \leftarrow \text{select\_clut}(\mathcal{G}_{org}, \text{random}(\theta) \% |\text{CLUT}|)$ 
5   if  $is\_suitable\_csb(\text{CLUT}_{inst})$  and not  $is\_csb(\text{CLUT}_{inst})$  then
6      $\text{CSB}_{inst} \leftarrow \text{convert\_clut\_to\_csb}(\mathcal{G}_{org}, \text{CLUT}_{inst}, \theta, \gamma)$ 
7     CSB. $\text{append}(\text{CSB}_{inst})$ 
8      $n_X \leftarrow n_X - 1$ 
/* Randomized Flop to CSB Conversion */
9  $n_Y \leftarrow (\text{random}(\theta) \% \gamma_{y\_max} \times |\mathcal{G}_{org}.Vert_{FF}|)$ 
10 while  $n_Y > 0$  do
11    $\text{FF}_{inst} \leftarrow \text{select\_flop}(\mathcal{G}_{org}.Vert_{FF}, \text{random}(\theta) \% |\mathcal{G}_{org}.Vert_{FF}|)$ 
12   if  $is\_suitable\_csb(\text{FF}_{inst})$  and not  $is\_csb(\text{FF}_{inst})$  then
13      $\text{CSB}_{inst} \leftarrow \text{convert\_flop\_to\_csb}(\mathcal{G}_{org}, \text{FF}_{inst}, \theta, \gamma)$ 
14     CSB. $\text{append}(\text{CSB}_{inst})$ 
15      $n_Y \leftarrow n_Y - 1$ 
16 return CSB

```

CPI Insertion The SAT attack resilience of the redacted design is improved by injecting CPIs into the design. Each CPI is an SAT-proof block designed using XOR/XNOR gates and MUXes, as these logic gates have a high clause/variable ratio, which makes it difficult for the SAT solver to compute the distinguishing input pattern [KAHS19]. In **HIPR**, each CPI is placed at the output of a randomly selected CLUT, excluding CLUTs converted into CSBs, and we connect the other inputs of the CPI to nets in the design located at the same logic depth as the CLUT to prevent combinational loops from forming. Algorithm 4

details the sequence of steps needed to perform interconnect logic redaction using CPIs.

Algorithm 4: Interconnect Logic Redaction using CPIs in **HIPR**.

```

Input :  $\mathcal{G}_{org}, \theta, \gamma$ 
Output: List of CPIs (CPI)
1  $CPI \leftarrow \emptyset$ 
   /* Parameterized CPI Mapping */
2  $E_{CPI} \leftarrow \text{identify\_cpi\_candidates}(\mathcal{G}_{org})$ 
3  $n_{CPI} \leftarrow (\text{random}(\theta) \% \gamma_{CPI}) \times |E_{CPI}|$ 
4  $n_{cov} \leftarrow 0$ 
5 while  $n_{CPI} > 0$  or  $n_{cov} > |\mathcal{G}_{org}.Edges|$  do
6    $C_{edg} \leftarrow E_{CPI}[\text{random}(\theta) \% |\mathcal{CLUT}|]$ 
7   if  $\text{is\_suitable\_cpi}(C_{edg})$  then
8      $\mathcal{D}_{in} \leftarrow \text{find\_dummy\_inputs}(\mathcal{G}_{org}, C_{edg}, \text{Logic\_Depth})$ 
9      $CPI_{inst} \leftarrow \text{generate\_cpi}(C_{edg}, \mathcal{D}_{in})$ 
10     $CPI.append(CPI_{inst})$ 
11     $n_{CPI} \leftarrow n_{CPI} - 1$ 
12     $n_{cov} \leftarrow n_{cov} + 1$ 
13 return CPI

```

Hardware Redaction **HIPR** achieves fine-grain redaction of security-critical Boolean logic cones, sequential logic like FFs, and interconnect randomization, as demonstrated in Fig. 5. The hardware redacted from the design is removed and replaced with a configurable fabric that can only be programmed correctly with a protected bitstream unknown to adversaries in untrusted facilities.

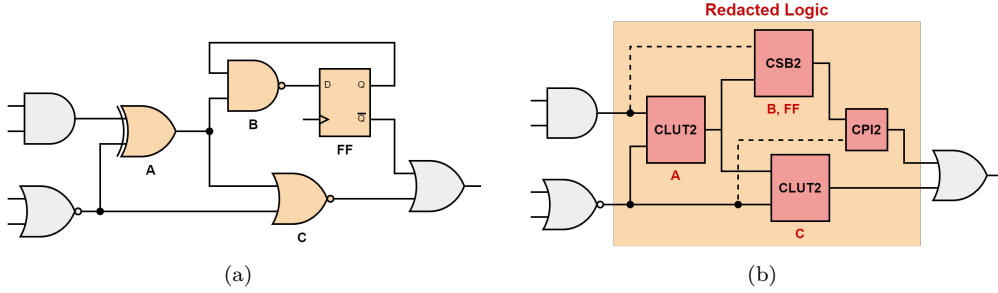


Figure 5: Hardware redaction in **HIPR**: (a) original logic, (b) redacted logic implemented using configurable fabric in **HIPR**.

3.1.4 Functional Bitstream Generation

The configuration bitstream (**cfg_bitstream**) for each CLUT inserted into the design is generated by simulating the Boolean function replaced by the same CLUT, which can be done using any commercial simulation tool. The Boolean functions are simultaneously simulated using a common testbench applying 2^N input patterns, where N is the largest CLUT size in $\mathcal{CLUT}$. The configuration bitstream for the CSB can be derived from the CLUT inside, and the I/O MUXes select bits. Similarly, the CPI bitstream is just the select bits of the MUXes or the XOR/XNOR gates that restore the original interconnect. The generated bitstream can be loaded to the inserted CLUTs during the power-up of the hardware to bring the design into functional mode and encrypted to enhance the security of the bitstream and the redacted design.

3.2 Security-Aware Overhead Optimization

The fine-grain redaction process in **HIPR** can end up generating an extremely large initial bitstream for full design redaction (e.g., redaction of 14.1% of the GPS benchmark¹ generates a bitstream of size > 100 K). Such a large bitstream incurs significant PPA overheads and seems overkill from a security perspective. **HIPR** incorporates optimizations through CLUT/CSB compaction followed by bitstream compaction to reduce the total bitstream size and an XOR-Tree-based bitstream generation technique to generate the bitstream of the desired size after compaction. Algorithm 5 describes the security-aware overhead optimization steps employed in **HIPR**. The goal is to optimize the bitstream to achieve robust security with minimal PPA overheads.

Algorithm 5: Security Aware Overhead Optimization in **HIPR**.

```

Input :  $\mathcal{G}_{mod}, \text{cfg\_bitstream}, \gamma$ 
Output : Compacted Bitstream ( $\text{cfg\_bitstream}_{compact}$ )

/* CLUT Compaction */
1 foreach  $\text{CLUT}_i \in \text{CLUT}$  do
2    $\text{itr} \leftarrow 0.5 * |\text{CLUT}_i|$ 
3   if  $|\text{CLUT}_i| > 2$  then
4     while  $\text{itr} > 0$  do
5        $\text{CLUT}'_i \leftarrow \text{clut\_compaction}(\text{CLUT}_i, \text{itr})$ 
6        $\text{itr} \leftarrow \frac{\text{itr}}{2}$ 
7      $\text{cfg\_bitstream}_{clut} \leftarrow \text{cfg\_bitstream}_{clut} \cup \sum \text{CLUT}'_i.\text{bitstream}$ 
8  $\delta \leftarrow \frac{|\text{cfg\_bitstream}_{clut}|}{\gamma_{compact\_ratio}}$ 
/* Bitstream Compaction */
9 foreach  $\text{CLUT}_i \in \text{CLUT}$  do
10   foreach  $\text{bit}$  in  $\text{CLUT}_i.\text{config}$  do
11     if next available bit  $B_i$  from  $\text{cfg\_bitstream}_{clut}$  exists then
12       if  $B_i$  satisfies  $\mathcal{R}$  then
13         if  $B_i.\text{Fanout} < \delta$  then
14           Assign  $B_i$  to  $\text{CLUT}_i$ 
15            $\text{cfg\_bitstream}_{bs\_compact} \leftarrow \text{cfg\_bitstream}_{bs\_compact} \cup B_i$ 
16         else
17           Goto line 11.
18       else
19         Goto line 11.
20     else
21       break  $\triangleleft$  Goto next CLUT

/* XOR-Tree Bitstream Generation */
22  $\mathcal{L} \leftarrow \frac{|\text{cfg\_bitstream}_{bs\_compact}|}{\gamma_{target\_bs}}$ 
23 while  $\mathcal{L} > 0$  do
24    $\text{xor\_tree\_gen}(\mathcal{L}, \gamma)$ 
25    $\text{cfg\_bitstream}_{compact} \leftarrow \text{cfg\_bitstream}_{compact} \cup \sum \mathcal{L}.\text{bits}$ 
26    $\mathcal{L} \leftarrow \mathcal{L} - 1$ 
27 return  $\text{cfg\_bitstream}_{compact}$ 

```

3.2.1 CLUT/CSB compaction

The CLUT/CSB compaction step takes advantage of the fact that nearby CLUTs can share the same bitstream bit, which allows data reusing stored in a single FF. Fig. 6 illustrates an example CLUT compaction of CLUT3 by **HIPR**. However, naively compacting the bitstream could inadvertently benefit attackers by reducing the number of bits they need to guess. Additionally, sharing bits may lead to a ripple effect, where guessing one storage

¹<https://github.com/mit-11/CEP>

element gives attackers insights into multiple CLUT configurations. Therefore, it is crucial to perform compaction in a way that maintains overall security.

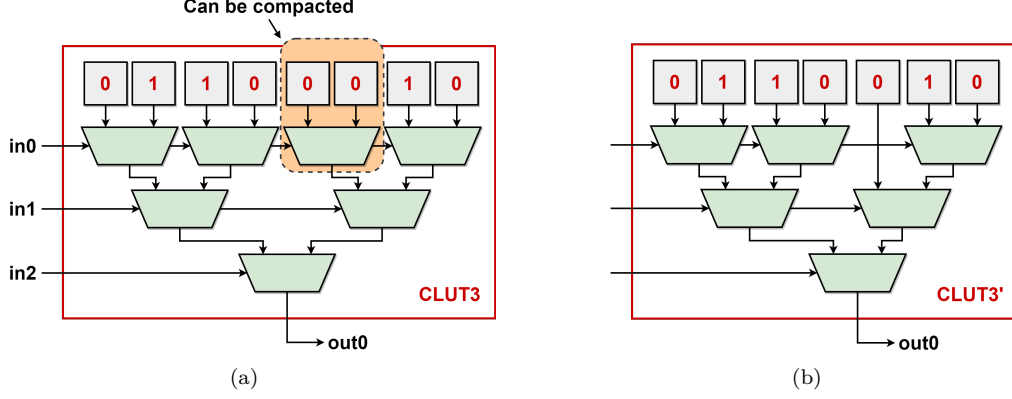


Figure 6: CLUT compaction in **HIPR**: (a) The two programming bits highlighted inside the CLUT3 can be shared without altering its functionality, and (b) the redacted logic after CLUT compaction implemented uses fewer gates, resulting in lower overheads.

3.2.2 Bitstream Compaction

Bitstream compaction refers to sharing configuration bits between CLUTs and reduces the total bitstream required for redaction. Fig. 7 demonstrates how 50% bitstream compaction is implemented for CLUTs located near each other in a redacted design.

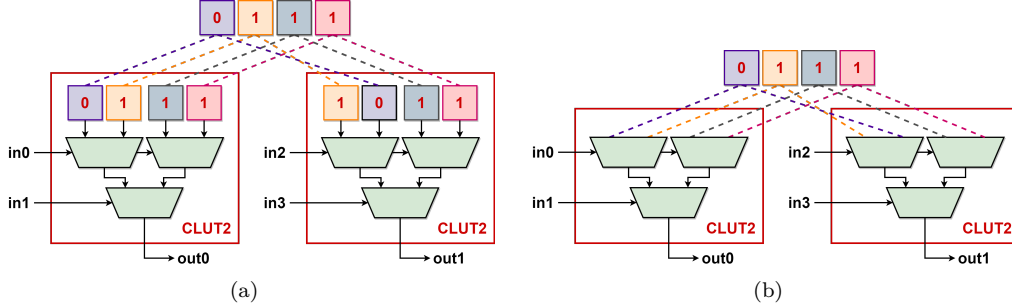


Figure 7: Bitstream compaction in **HIPR**: (a) Two neighboring CLUT2s can share four programming bits under 50% bitstream compaction without modifying their behavior, and (b) the redacted logic after bitstream compaction with 50% fewer bit FFs.

HIPR applies the following security rules ($\mathcal{R}$) while implementing bitstream compaction:

- $\mathcal{R}_1$: The minimum configuration bits required for a CLUT should be ≥ 2 .
Rationale: To prevent the use of CLUT1, which are essentially inverters or buffers and can be easily predicted.
- $\mathcal{R}_2$: Multiple configuration bits from a given CLUT should not be assigned to the same compacted bit.
Rationale: To prevent intra-CLUT configuration bit sharing during bitstream compaction. For a given CLUT, if a compacted bit is shared between k configuration bits, the brute-force attack complexity is reduced by a factor of 2^k .
- $\mathcal{R}_3$: During bitstream compaction, use larger CLUT instances (CLUT3, CLUT4, etc.), wherever possible, to create a larger search space.

Rationale: To expand the CLUT functionality search space for brute-force or oracle-guided attacks. For a 3-input logic cone, using a CLUT3 instead of two cascaded CLUT2s increases the brute-force attack complexity from $2^5 (= 2 \times 2^4)$ to 2^9 .

- $\mathcal{R}_4$: If one of the CLUT inputs consists of a control signal (such as reset or enable pins), do not use bitstream compaction, and the minimum input size of CLUT should be ≥ 2 .

Rationale: Control signals are generally primary inputs that are used to drive multiple CLUTs. Attackers can use these primary inputs to reduce the brute-force attack complexity, which may be further reduced when bitstream compaction is used. To prevent this scenario, larger CLUTs are used without any bitstream compaction.

- $\mathcal{R}_5$: For a set of configuration bits $B^{C_k} = \{b_0, b_1, \dots, b_j\}$ belonging to a CLUT C_k :
 1. If bit b_i is initially chosen for an even (odd) parity index, then b_i in any other CLUT instance appears only in even (odd) parity indices.
 2. There cannot be any partial overlap between two sets of compacted bits belonging to two different CLUTs.

Rationale: To prevent attackers from gaining any advantage due to the overlap of compacted bits between CLUTs. For a set of CLUTs with overlapping configuration bits, attackers can leverage the overlapped bits to minimize their combined functionality search space (e.g., by comparing CLUT truth tables) and reduce the effective brute-force attack complexity.

3.2.3 XOR Tree for Bitstream Generation

HIPR utilizes XOR Tree for bitstream generation, leveraging a hierarchical structure that efficiently combines multiple inputs to produce a single output. A daisy-chained shift register is used to store the N -bit bitstream. **HIPR** calculates the number of hierarchical levels based on the length of the compacted bitstream (`cfg_bitstreambs_compact`) and the target bitstream length specified in the redaction parameter (γ). Fig. 8 demonstrates XOR-Tree for Bitstream Generation in **HIPR**. This approach enhances SAT resilience by hiding the relationships between individual bits. Even if some bits are recovered by the adversary, it remains challenging to derive the original inputs without complete knowledge of the full XOR tree.

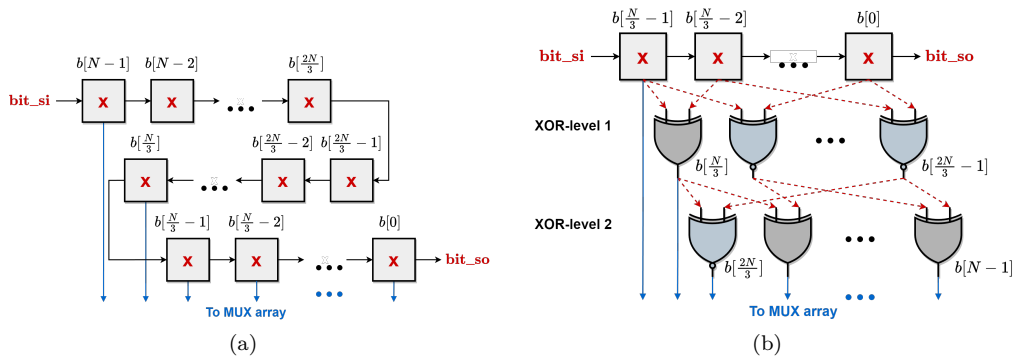


Figure 8: Bitstream generation using XOR-Tree in **HIPR**: (a) daisy-chained shift-register used to store the N -bit bitstream, and (b) generating two-thirds of the bitstream using 2 levels of XOR/XNOR gates. Replacing these bit FFs with XOR/XNOR gates reduces the area overhead for the redacted design, while increasing the SAT-attack resilience.

3.3 Integrating HIPR into RTL-to-GDSII EDA tool flow

HIPR is compatible with standard commercial (Synopsys/Cadence) EDA tool flow and can be easily integrated, enhancing the security of hardware designs by incorporating fine-grained redaction. This integration allows for the redaction of security-critical components during the synthesis phase by **HIPR**. It generates the redacted design that progresses through the subsequent design phases and is finally fabricated to produce ICs. The adversary (either an untrusted foundry or a malicious end-user) can access the redacted netlist without the configuration bitstream. The configuration bitstream is protected and not revealed to the entities in the supply chain, effectively safeguarding the hardware IP against confidentiality and integrity attacks. Only the trusted entities who have access to the configuration bitstream can program the fabricated design with redacted logic. Hence, the integration of **HIPR** streamlines the workflow and enhances design flexibility, making it more resilient against RE attacks. Fig. 9 demonstrates how **HIPR** can be integrated with commercial EDA tools within the IC design flow.

3.3.1 Impact on Post-Synthesis Stages

As depicted in Fig. 9, the **HIPR** tool only interacts with the synthesis stage to replace the original design with the redacted netlist. Since an N -input CLUT typically replaces more than one logic gate on average (due to its flexibility to implement arbitrary N -input logic function), the number of inter-CLUT interconnects is fewer than the original number of nets between logic gates. However, CLUTs themselves will have more local nets internal to the cell compared to standard cells with comparable inputs. In addition, the interconnects between CLUTs and CSBs will remain largely similar to those in the original design. Further, custom cell design for CLUTs/CSBs with fixed height and width, allows for efficient placement and routing and minimizing wire length, similar to their standard cell counterparts. Therefore, one can expect to have no significant penalty in the physical design (e.g., floorplanning, placement, and routing) for the redacted design comprising of CLUTs/CSBs, compared to the original one. Our future work will focus on detailed analysis of the effect of redaction on post-synthesis design stages.

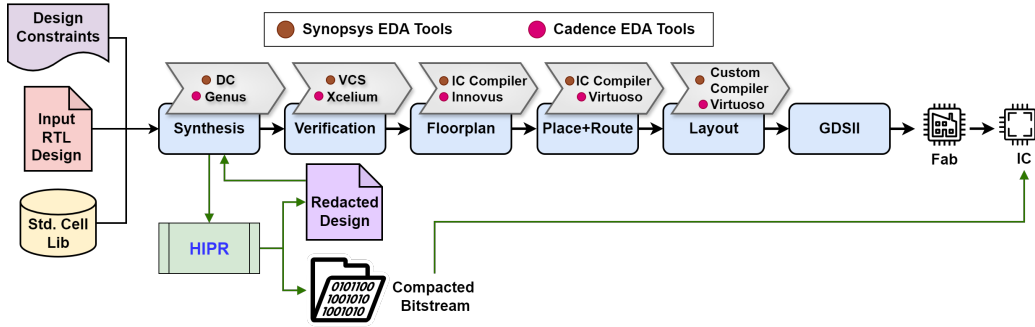


Figure 9: Integrating **HIPR** into commercial EDA tool flow.

3.3.2 Runtime Bitstream Initialization

The compacted bitstream generated by **HIPR** is deployed along with the IC to the end user, similar to an FPGA bitstream, as shown in Fig. 9. The redacted designs need to be programmed with the compacted bitstream by the end-user at run-time to enable intended functionality similar to power-up bitstream initialization in FPGA.

4 Results and Analysis

4.1 Experimental Setup

We evaluated the proposed fine-grain redaction tool flow using the open-source benchmarks from MIT-CEP². The PPA results and benchmark specification were calculated from the gate-level netlist mapped to the NanGate 15nm open standard cell library³ generated via synthesis using Synopsys Design Compiler (V-2023.12-SP5). All synthesis steps performed for PPA overhead calculation and analysis were constrained to a 500MHz operating clock frequency. All cell counts, power, delay and area values reported in section 4.2 (apart from non-**HIPR** results from Table 4) are the exact values or overheads calculated using the exact values obtained from the synthesis logs and reports generated by the synthesis tool. The bitstream required to program the configurable fabric was obtained via functional simulation of redacted logic cones, performed using Synopsys VCS (V-2023.12), and the same tool was used for simulation of the redacted netlist to verify the correctness of the configuration and compacted bitstream. The *sld* executable from a clone repository⁴ was used to run the SAT attack and the dataflow analysis plugin from *hal*⁵ was used to perform the DANA attack. Cadence Conformal (21.10-s300) was used to perform Logic Equivalence Checking (LEC) and obtain design similarity scores. Most of our experiments were carried out on a Red Hat Enterprise Linux Server 7.9 server with AMD® Epyc 7713 64-core processor and 1007.6 GiB Memory, apart from the *hal* runs that were executed on Ubuntu 22.04.3 LTS with Intel(R) Core(TM) i7-8700 CPU.

4.2 Overhead Analysis

Table 2: Redacted IP specifications for MIT-CEP benchmarks under test for **HIPR**.

Benchmark	Modules	Cells *	Full Chip Area* (μm^2)	Test IP [†]	% Redacted by	
					Cells *	Area*
AES192	18	224.5 K	68400.6	RD ₁ : round r1, rf	11.4%	10.6%
DES3	10	2.3 K	821.7	RD ₂ : des3	100.0%	100.0%
DFT	134	148.5 K	82036.3	RD ₃ : perm74590	21.3%	20.9%
FIR	4	1.0 K	1127.5	RD ₄ : FIR_filter	100.0%	100.0%
GPS	21	225.8 K	69101.8	RD ₅ : round r1, r2	15.2%	14.1%
IDFT	134	148.5 K	81763.7	RD ₆ : perm42350	21.4%	21.0%
IIR	4	1.7 K	1738.6	RD ₇ : IIR_filter	100.0%	100.0%
MD5	2	6.5 K	2696.5	RD ₈ : md5	100.0%	100.0%
RSA	10	255.7 K	130004.2	RD ₉ : montprod	12.5%	13.1%
SHA256	3	6.0 K	3194.4	RD ₁₀ : sha256_core	54.2%	56.8%
Average	34	102.1 K	44088.5		53.6%	53.6%

* Reported by Synopsys DC for NanGate 15nm. [†]RD_i : Redacted Design for Test IP.

4.2.1 Evaluation Benchmarks

To demonstrate the robustness and scalability of the proposed tool flow, we targeted test IPs of varying proportions of cell counts and area from the 10 MIT-CEP benchmarks. Table 2 highlights the wide variety of test IPs used to evaluate **HIPR**. It should be noted that although **HIPR** allows fine-grain redaction of critical logic cones at the gate-level, these test IPs are sub-modules or procedural blocks identified from the benchmark at the RTL. This was done to streamline comparative analysis between existing redaction

²<https://github.com/mit-ll/CEP.git>

³<https://si2.org/open-cell-and-free-pdk-libraries/>

⁴<https://github.com/DfX-NYUAD/PSAT>

⁵<https://github.com/emsec/hal/wiki/Dataflow-Analysis>

techniques versus **HIPR** runs under different settings of redaction parameter(γ) and seed (θ). Each test IP is also assigned a **RD_i** that will be used to refer to these IP blocks in subsequent sections. Cell counts ($|Cells|$) reported in Table 2 constitute standard cells (combinational + sequential) from the NanGate 15nm library that are instantiated in the post-synthesis netlist (not NAND2 equivalent). The $|Cells|$ represent the exact value of “*Leaf Cell Count*” obtained from the Quality of Results (QoR) generated during synthesis by Synopsys Design Compiler, however, the tabulated values are reported as multiples of 1000 (= 1 K). Similarly, the area values reported in Table 2 correspond to the exact values (in μm^2) of “*Design Area*” obtained from the synthesis QoR reports.

4.2.2 Implementation Results

Table 3 reports the original PPA numbers reported by the synthesis tool for the 10 test IPs and the IP-level overheads (calculated as x times original metric value) after redaction using **HIPR**. The overhead numbers reported in this section for **HIPR** transformations factor in the bitstream storage as a daisy-chained shift register. Column 2 (titled “Original”) reports the implementation results for the original test IPs, where $\mathcal{A}$ is area (in μm^2), $\mathcal{D}$ is delay (in ps) and $\mathcal{P}$ is power (in mW). Columns 3, 4, and 5 report the corresponding overhead values for transformations **T1**, **T2** and **T3** respectively. For example, the delay overhead due to transformation **T3** over the test IP **RD2 (des3)** is 4.5x or 350% compared to the original value of 165.5 ps.

The original and redacted test IPs for **T1**, **T2** and **T3** (mapped to NanGate 15nm) along with their functional bitstreams, generated by **HIPR**, are publicly available⁶. For each test IP, 100% of Boolean logic was redacted, accompanied by randomized but minimal redaction of sequential and interconnect logic (within 5-10%). The overhead results reported in Table 3 were collected after redaction corresponding to three separate transformation settings (**T1**, **T2**, **T3**) with different security-aware optimization steps implemented in each setting to reduce the overhead costs, described as follows:

- **Transformation T1(α)**: Here α corresponds to a bitstream compaction step with 90% compaction ratio for minimizing overheads. The configuration bitstream size is optimized to about 10% without violating any rules from $\mathcal{R}$.
- **Transformation T2($\alpha+\beta$)**: Here β corresponds to replacing 75% of the bitstream shift register with a 3-level XOR-Tree, in addition to a bitstream compaction step (α) to minimize overheads and potentially increasing SAT attack resistance. The effective configuration bitstream size is reduced to 25% of its compacted size after bitstream compaction.
- **Transformation T3($\alpha+\beta+\delta$)**: where δ corresponds to a rule-based CLUT/CSB compaction step applied to every CLUT/CSB before further overhead optimization due to bitstream compaction (α) and XOR-Tree (β).

From Table 3, we can observe how the overhead values for the test IPs vary with the three separate transformation settings (**T1**, **T2**, **T3**) listed above, compared to the original implementation. The average area overhead decreases from 6.1x in **T1** to 5.4x in **T2** and further down to 3.8x in **T3**, which is expected due to the bitstream size optimizations. The average delay overhead does not vary significantly between **T1** and **T2** as bitstream generation via XOR-Tree does not affect the critical timing path, then decreases marginally to 2.4x in **T3** as gates and FFs are optimized away under CLUT compaction. Notably, the critical timing path slacks for the redacted test IPs remain well within the design constraint of 2000 ps, even with delay overheads >2x original. However, the average power overhead almost triples between **T1** to **T2** from 2.0x to 5.6x, and before decreasing to

⁶<https://github.com/UF-Nelms-IoT-Git-Projects/HIPR>

4.2x in **T3**. The significant increase in power consumption after adding the the XOR-Tree in **T2** and **T3** can be explained by the switching behavior of the XOR/XNOR gates.

Table 3: Overhead results for MIT-CEP benchmarks redacted by **HIPR**.

Test IP	Original	T1 (α)	T2 ($\alpha+\beta$)	T3 ($\alpha+\beta+\delta$)
	$\{\mathcal{A}, \mathcal{D}, \mathcal{P}\}^\dagger$	$*Ovh\{\mathcal{A}, \mathcal{D}, \mathcal{P}\}$	$*Ovh\{\mathcal{A}, \mathcal{D}, \mathcal{P}\}$	$*Ovh\{\mathcal{A}, \mathcal{D}, \mathcal{P}\}$
RD ₁	7270.6, 86.8, 2.0	8.0x, 5.2x, 3.1x	7.0x, 4.8x, 6.8x	4.9x, 3.6x, 4.6x
RD ₂	821.7, 165.5, 0.6	10.1x, 4.5x, 1.7x	8.8x, 4.6x, 7.5x	6.1x, 4.5x, 5.7x
RD ₃	17141.6, 369.2, 8.7	5.1x, 1.8x, 1.8x	4.5x, 1.4x, 5.0x	2.9x, 1.6x, 3.4x
RD ₄	1127.5, 396.7, 0.7	4.1x, 1.6x, 1.5x	3.7x, 1.7x, 2.9x	2.9x, 2.3x, 2.6x
RD ₅	9721.9, 86.8, 3.1	6.1x, 4.4x, 2.0x	5.3x, 5.5x, 5.1x	3.8x, 3.5x, 3.6x
RD ₆	17141.6, 369.2, 8.7	5.1x, 1.8x, 1.8x	4.5x, 1.4x, 5.0x	2.9x, 1.6x, 3.4x
RD ₇	1738.6, 462.6, 1.1	4.0x, 1.5x, 1.3x	3.5x, 1.8x, 2.8x	2.8x, 1.9x, 2.7x
RD ₈	2696.5, 1016.0, 0.7	6.8x, 1.6x, 3.0x	5.9x, 1.7x, 10.4x	4.3x, 1.8x, 8.6x
RD ₉	16985.4, 541.6, 8.7	5.3x, 2.5x, 1.7x	4.6x, 2.4x, 5.0x	3.0x, 1.8x, 2.9x
RD ₁₀	1814.6, 341.5, 1.0	6.8x, 1.6x, 1.7x	5.9x, 1.8x, 5.6x	4.6x, 1.9x, 4.8x
Average	7646.0 383.6, 2.0	6.1x, 2.6x, 2.0x	5.4x, 2.7x, 5.6x	3.8x, 2.4x, 4.2x

$\mathcal{A}/\mathcal{D}/\mathcal{P}$: Area/Delay/Power. † Units: $\mathcal{A}$ (μm^2), $\mathcal{D}$ (ps), $\mathcal{P}$ (mW).

* Overheads (*Ovh*) are reported as x times original value.

4.2.3 Impact of Overhead Optimizations

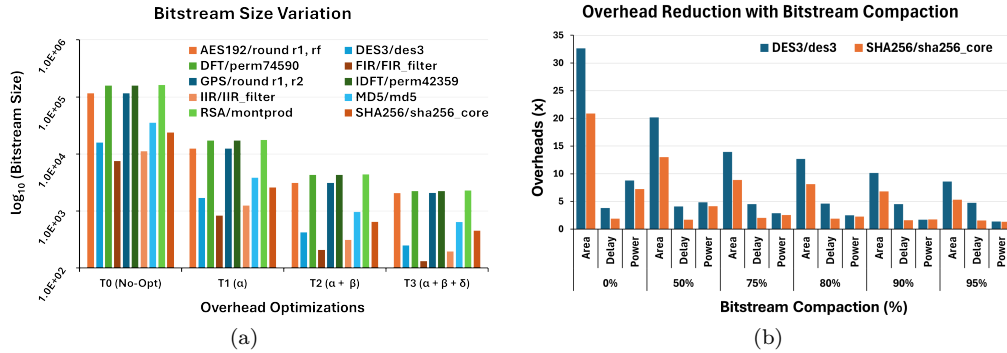


Figure 10: Impact of overhead optimizations on redaction metrics in **HIPR**: (a) Bitstream size for the redacted test IPs from MIT-CEP is reduced by several orders of magnitude under overhead optimizations, (b) PPA overhead costs for **RD₂(des3)** and **RD₁₀(sha256_core)** improve significantly due to bitstream compaction.

Fig. 10 demonstrates how the overhead optimization techniques proposed in **HIPR** affect redacted design metrics like bitstream size and PPA overhead costs. The impact of the overhead optimization steps in **T1**(α), **T2**($\alpha+\beta$), **T3**($\alpha+\beta+\delta$) on the configuration bitstream sizes of the 10 redacted test IPs from MIT-CEP is depicted in Fig. 10b. It can be observed that the effective bitstream size is reduced by several orders of magnitude, compared to transformation **T0** with no overhead optimization. The average bitstream size goes down from 80.9 K bits in **T0** to 8.8 K bits in **T1** (with 90% bitstream compaction), then to 2.2 K bits in **T2** (with 75% XOR/XNOR replacement on top of **T1**) and further drops to 1.3 K bits in **T3** (with CLUT/CSB compaction on top of **T2**).

A similar trend is observed in PPA overheads for **RD₂(des3)** and **RD₁₀(sha256_core)** when the bitstream compaction ratio is increased from in steps 0% to 95%, as shown in Fig. 10b. For both IPs, the area overhead is the most improved metric, with an improvement of about 2 orders of magnitude from 0% compaction to 95% compaction. For example, in

case of **RD₂**, an IP with about 2.3 K cells and a 100% redacted test IP, the area overhead decreases from 20.9x for a bitstream size of 22.4 K bits at 0% compaction to 5.31x area overhead with a bitstream size of 1.3 K bits at 95% bitstream compaction, which is still a significantly large bitstream size for a brute-force guess. It is obvious that the compacted bitstream of size 1.3 K will be less secure than the non-compacted bitstream, but the rules in $\mathcal{R}$ ensure that the redacted **RD₂** design with bitstream compaction is still quite secure. This fact is demonstrated in the security analysis section using both functional and structural attacks on redacted designs.

4.2.4 Comparison with state-of-the-art

Table 4 compares the area overhead from **HIPR** against the state-of-the-art hardware redaction techniques, including recent work on both eFPGA-based [BTMT⁺21, MAS⁺21, KAFT23] and LUT-based redaction [GRK⁺23]. For a fair comparison, the area overhead values are adjusted to account for identical amount of redacted logic as **HIPR** for **RD₅**, which constitutes 14.1% of the **GPS** full-chip by area (see Table 2). The fine-grain redaction by **HIPR** results in an IP-level overhead of 3.8x, which corresponds to a full-chip area overhead of 1.4x. The eFPGA-based redaction techniques [BTMT⁺21, MAS⁺21, KAFT23] are evaluated on the same **GPS** and **AES** benchmarks⁷ as **HIPR**, but the test IPs redacted are extremely small (< 1%). The LUT-based redaction proposed in EvoLUTe [GRK⁺23] is evaluated on the **Multiplier** benchmark⁸ and the size of the redacted logic is determined by the timeout ($\mathcal{TO}$) for SAT attack [SRM15]. Moreover, the overheads reported for EvoLUTe do not take into account the additional area required for bitstream storage and configuration, resulting in lower than actual overheads. Therefore, we consider the equivalent eFPGA overheads of the redacted logic, as reported in [GRK⁺23], to estimate the size the redacted logic and the actual overhead for a fair comparison with **HIPR**.

We can observe from Table 4 that the proposed fine-grain redaction in **HIPR** incurs significantly lower area overhead compared to the state-of-the-art, when adjusted for redacting the same amount of logic. Hence, we can infer that the security-aware overhead optimization employed by **HIPR** helps reduce the overhead by 1 to 2 orders of magnitude compared to existing redaction approaches.

Table 4: Overhead comparison between **HIPR** and state-of-the-art redaction techniques.

Method	Framework	Test IP	Redacted Portion*	Area Overhead*	
				Reported	Adjusted [®]
eFPGA ₁ [MAS ⁺ 21]	OpenFPGA	GPS : pcode	0.6% (by area)	1.4x	34.3x
eFPGA ₂ [BTMT ⁺ 21]	OpenFPGA	GPS : cacode	0.1% (by area)	2.1x	271.3x
SheLL [KAFT23]	FABulous	AES : __addround_xor	0.1% (by area)	1.4x	180.1x
EvoLUTe [GRK ⁺ 23]	LUT	Multiplier	150 LUTs (13 K bits)	0.9x [★]	40.1x
HIPR [†]	LUT	GPS : round r1,r2	14.1% (by area)	1.4x	1.4x

* Compared to the full-chip area. ★ Does not include bitstream storage area.

® Adjusted to match the amount of logic redacted by **HIPR**. † This work.

4.3 Security Analysis

In this section, we evaluate functional and structural attack resistance for the 10 test IPs from the MIT-CEP benchmarks redacted by **HIPR**. The functional attack resistance for the redacted designs is determined by performing the SAT attack [SRM15]. For structural attack resistance, we employ the DANA attack [AHT⁺20] for dataflow analysis on the sequential logic in the redacted IPs and perform LEC between pairs of redacted IPs to observe the structural similarity between them [DPS⁺25].

⁷<https://github.com/mit-ll/CEP.git>

⁸<https://github.com/lsils/benchmarks>

4.3.1 Functional Attack Resistance

Resilience to SAT attack The open-source *sld* executable was used to perform the SAT attack, and the attack timeout ($\mathcal{TO}$) was set to 1 day (24 hours). Since the test IPs are all sequential designs (with FFs) and *sld* can only run on combinational design netlists provided in the BENCH format, each redacted IP was converted into their corresponding combinational profile under the assumption that all FFs in the design have scan-chain access. Under this assumption, each FF input was converted into a functional output (or pseudo primary output), and each FF output was converted into a functional input (or pseudo primary input). The modified netlist hypergraph ($\mathcal{G}_{mod}$) in **HIPR** was used to generate the combinational profiles of each redacted test IP in the BENCH format. The original netlist hypergraph ($\mathcal{G}_{org}$) was similarly utilized to generate its combinational profile, which is required by *sld* as the golden *oracle*.

Table 5 reports the SAT attack runtimes observed for the 10 redacted test IPs under **T1**(α), **T2**($\alpha+\beta$), and **T3**($\alpha+\beta+\delta$). Across the three transformations, the *sld* tool reached $\mathcal{TO}$ for 8 (out of 10) redacted test IPs, but failed to run on the redacted test IPs for **RD₁**(round **r1**, **rf**) and **RD₅**(round **r1**, **r2**), citing I/O mismatch between the redacted IP versus the *oracle*. The I/O mismatch is due to the presence of additional pseudo I/O in the redacted IP corresponding to CSBs inserted by **HIPR** containing dummy FFs that do not exist in the *oracle*, and prevents SAT attack on **RD₁** and **RD₅**. Under the assumed threat model, it is feasible for an adversary to leverage the *oracle* to identify the true I/O in the redacted IP and perform a modified SAT attack with considerable added complexity from isolating the dummy I/O before enumerating all possible dummy input patterns. As a result, the attack runtime for a redacted IP with n true inputs and d dummy inputs increases by a factor of τ , quantified as:

$$\tau = \binom{n+d}{d} \cdot 2^d \quad (2)$$

Hence, we can infer that the redaction performed by **HIPR** is resilient against *oracle*-guided functional attacks such as SAT [SRM15].

4.3.2 Structural Attack Resistance

Resilience to DANA attack The DANA attack was performed on the redacted test IPs using the dataflow analysis plugin in the *hal* tool. The gate-level redacted netlists from the overhead analysis, mapped to the NanGate 15nm, were re-used for this analysis after removing the bitstream shift register FFs. The DANA tool isolates the FFs from the gate-level netlist and attempts to group them together into clusters that correspond to the design registers, which can be used to identify the dummy CSB FFs added by **HIPR**. We also ran DANA on the original test IPs to obtain the baseline for comparison. Table 5 reports the clusters reported by DANA for all 10 redacted test IPs and their runtimes. The average number of clusters reported by DANA increases significantly from 26 to 499 in case of **T1**(α) and **T2**($\alpha+\beta$), and to 434 in case of **T3**($\alpha+\beta+\delta$), which implies that the redaction performed by **HIPR** on sequential designs is resilient against dataflow-based structural attacks such as DANA [AHT⁺20].

Although a higher number of clusters is typically desirable, a cluster count of 1 is also good for security, indicating that DANA was not able to identify any register cluster in the redacted design, as observed for **RD₂**(des3). The number of clusters reported by DANA for redacted test IPs vary between **T1**/**T2** versus **T3** due to randomized structural transformations in **HIPR**. The addition of dummy CSBs and CPIs alters the total FF count as well as their fan-in logic cones, resulting in a reduced number of identified clusters in the case of **RD₃**(perm74590), **RD₆**(perm42350) and **RD₁₀**(sha256_core).

Table 5: Functional (SAT) and Structural (DANA) attack results for MIT-CEP benchmarks redacted by **HIPR**.

Test IP	SAT-Attack: Runtime (h) [†]			DANA-Attack: $\{ Clusters , \text{Runtime (s)}\}$		
	T1(α)	T2($\alpha+\beta$)	T3($\alpha+\beta+\delta$)	Original	T1(α), T2($\alpha+\beta$) [‡]	T3($\alpha+\beta+\delta$)
RD ₁ [*]	-	-	-	5, 0.22s	63, 2.38s	63, 1.54s
RD ₂	$\mathcal{TO}$	$\mathcal{TO}$	$\mathcal{TO}$	8, 0.05s	1, 0.22s	1, 0.15s
RD ₃	$\mathcal{TO}$	$\mathcal{TO}$	$\mathcal{TO}$	26, 2.13s	1767, 40.01s	1503, 25.61s
RD ₄	$\mathcal{TO}$	$\mathcal{TO}$	$\mathcal{TO}$	45, 0.19s	129, 1.09s	154, 1.42s
RD ₅ [*]	-	-	-	6, 0.28s	90, 3.30s	90, 2.56s
RD ₆	$\mathcal{TO}$	$\mathcal{TO}$	$\mathcal{TO}$	26, 2.12s	1767, 37.56s	1503, 25.40s
RD ₇	$\mathcal{TO}$	$\mathcal{TO}$	$\mathcal{TO}$	21, 0.47s	165, 2.68s	246, 3.73s
RD ₈	$\mathcal{TO}$	$\mathcal{TO}$	$\mathcal{TO}$	1, 0.17s	1, 0.86s	1, 0.66s
RD ₉	$\mathcal{TO}$	$\mathcal{TO}$	$\mathcal{TO}$	31, 2.7s	669, 16.49s	771, 16.83s
RD ₁₀	$\mathcal{TO}$	$\mathcal{TO}$	$\mathcal{TO}$	89, 0.60s	335, 2.28s	1, 0.83s
Average				26, 0.89s	499, 10.69s	434, 7.87s

* *sld* run fails from I/O mismatch (due to dummy CSB FFs). [†]Timeout ($\mathcal{TO}$) for *sld* was 24 hours.

[‡]Redacted logic is unchanged between T1 and T2, only bitstream generation is optimized.

Resilience to Structural Analysis using LEC Cadence Conformal was used to perform LEC between pairs of redacted IPs, and the resulting LEC design similarity metric was used to determine the structural and functional similarity between the two IPs after redaction by **HIPR**. We first compared all 10 redacted test IPs from MIT-CEP against themselves, to compare the inter-IP redacted designs. The resulting LEC similarity scores were normalized and plotted as heatmaps for the transformations under **T1(α)**, **T2($\alpha+\beta$)**, **T3($\alpha+\beta+\delta$)** separately, as depicted in Fig. 11a. We observe that all the 10 redacted IPs have a strong structural correlation with itself, depicted by the red diagonal with LEC similarity of 1, and show very weak correlation to the other designs, except when comparing **RD₃(perm74590)** with **RD₆(perm42359)**. These two IPs are actually identical to each other in structure and functionality due to the nature of the DFT and IDFT benchmarks. This can also be verified by comparing the corresponding PPA metrics from Table 3.

Next, we compare the redacted variants generated from the test IP by varying the seed value θ in **HIPR**, and generate 6 redacted variants for the test IPs **RD₁(round r1, rf)**, **RD₂(des3)**, and **RD₁₀(sha256_core)**. For each test IP, LEC similarity scores between the 6 variants are calculated and then we generate heatmaps from the normalized LEC similarity scores, as depicted in Fig. 11b. We observe that each variant only shows a strong correlation with itself and very low or no correlation with the other redacted variants, even though they were generated from the same original test IP.

From the above two experiments, we can infer that the redacted designs generated by **HIPR** have a high degree of structural randomness, even if they were generated by redaction from the same source IP. Hence, the redaction performed by **HIPR** is resilient against more sophisticated structural-analysis-based attacks [DPS⁺25]. Additionally, we can also infer that **HIPR** is resilient to removal attacks since the adversary cannot remove configurable blocks, as removing them can strip the functionality of the circuit.

4.3.3 Physical Attack Resistance

While the redacted design in **HIPR** is provably secure against confidentiality and integrity attacks, physical attacks [TLSB17, RTR⁺20, Dev24] on the bitstream – both invasive, such as, microprobing, as well as non-invasive, such as, SCA – become viable. Such attacks, however, are widely applicable to diverse on-chip secrets – e.g., FPGA bitstream or cryptographic key. As such, existing countermeasures to protect a bitstream against leakage through various forms of physical attacks [BGP⁺12, DWLX21] will be generally applicable to the bitstream generated by **HIPR**. For example, probing-resistant on-chip memory for storing bitstream or approaches to ensuring constant power consumption during bitstream loading can provide strong defense against physical attacks on the **HIPR** bitstream.

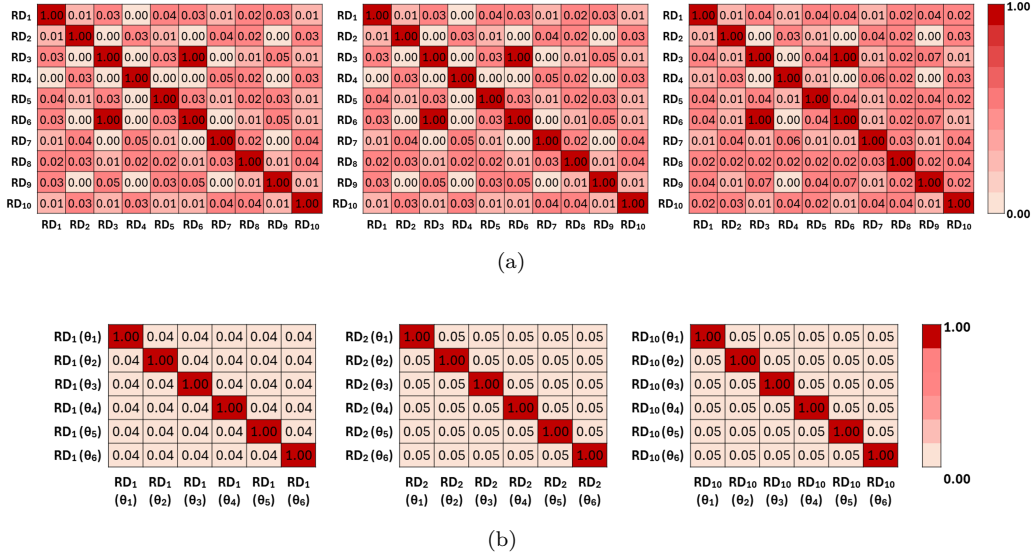


Figure 11: Normalized LEC similarity metrics depicting the structural randomness in the redacted designs generated by **HIPR**: (a) for the 10 MIT-CEP benchmarks redacted under different overhead optimizations (α , $\alpha + \beta$ and $\alpha + \beta + \delta$), and (b) under varying seed (θ) values for **RD₁**(round r1,rf), **RD₂**(des3), and **RD₁₀**(sha256_core). The heatmaps demonstrate that **HIPR** introduces significant structural transformations into the redacted designs under various overhead optimization and different random seeds.

5 Conclusion

In this paper, we have presented **HIPR**, a novel fine-grain hardware redaction methodology for IP protection that can be easily integrated into the commercial EDA tool flow. **HIPR** utilizes custom programmable fabric for implementing the redacted logic followed by security-aware bitstream compaction that effectively reduces area overhead without compromising security. We have demonstrated that **HIPR** is robust and scalable, performing hardware redaction at significantly lower overhead costs than the state-of-the-art. We have performed security analysis of **HIPR**, demonstrating its resilience against conventional structural and functional attacks. Future work will include further reduction in PPA overheads without compromising security, efficient testing and verification of redacted designs, and improving their resilience against physical attacks.

References

- [AB21] Abdulrahman Alaql and Swarup Bhunia. SARO: Scalable Attack-Resistant Logic Locking. *IEEE Transactions on Information Forensics and Security*, 16:3724–3739, 2021.
- [AFB19] Abdulrahman Alaql, Domenic Forte, and Swarup Bhunia. Sweep to the Secret: A Constant Propagation Attack on Logic Locking. In *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6, 2019.
- [AHT⁺20] Nils Albartus, Max Hoffmann, Sebastian Temme, Leonid Azriel, and Christof Paar. DANA Universal Dataflow Analysis for Gate-Level Netlist Reverse Engineering. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):309–336, Aug. 2020.

- [AKHS18] Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan. SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks. *TCHES*, 2019(1):97–122, Nov 2018.
- [APK⁺21] Lilas Alrahis, Satwik Patnaik, Faiq Khalid, Muhammad Abdullah Hanif, Hani Saleh, Muhammad Shafique, and Ozgur Sinanoglu. GNNUnlock: Graph Neural Networks-based Oracle-less Unlocking Scheme for Provably Secure Logic Locking. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 780–785, 2021.
- [APSS22] Lilas Alrahis, Satwik Patnaik, Muhammad Shafique, and Ozgur Sinanoglu. OMLA: An Oracle-Less Machine Learning-Based Attack on Logic Locking. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(3):1602–1606, 2022.
- [BGP⁺12] R. Beat, P. Grabher, D. Page, S. Tillich, and M. Wójcik. On reconfigurable fabrics and generic side-channel countermeasures. *International Conference on Field Programmable Logic and Applications (FPL)*, pages 740–741, 2012.
- [BMT⁺23] Jitendra Bhandari, Abdul Khader Thalakkattu Moosa, Benjamin Tan, Christian Pilato, Ganesh Gore, Xifan Tang, Scott Temple, Pierre-Emmanuel Gaillardon, and Ramesh Karri. Not All Fabrics Are Created Equal: Exploring eFPGA Parameters for IP Redaction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(10):1459–1471, Oct 2023.
- [BT18] Swarup Bhunia and Mark Tehranipoor. *Hardware Security: A Hands-on Learning Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2018.
- [BTMT⁺21] Jitendra Bhandari, Abdul Khader Thalakkattu Moosa, Benjamin Tan, Christian Pilato, Ganesh Gore, Xifan Tang, Scott Temple, Pierre-Emmanuel Gaillardon, and Ramesh Karri. Exploring eFPGA-based Redaction for IP Protection. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.
- [CB09] Rajat Subhra Chakraborty and Swarup Bhunia. HARPOON: an obfuscation-based soc design methodology for hardware protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, 2009.
- [CCB18] Prabuddha Chakraborty, Jonathan Cruz, and Swarup Bhunia. SAIL: Machine learning guided structural analysis attack on hardware obfuscation. In *2018 Asian Hardware Oriented Security and Trust Symposium (Asian-HOST)*, pages 56–61. IEEE, 2018.
- [CD94] J. Cong and Yuzheng Ding. FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(1):1–12, Jan 1994.
- [Dev24] Dev M. Mehta and Mohammad Hashemi and Domenic Forte and Shahin Tajik and Fatemeh Ganji. 1/0 Shades of UC: Photonic Side-Channel Analysis of Universal Circuits. Cryptology ePrint Archive, Paper 2024/072, 2024.

- [DPS⁺25] Aritra Dasgupta, Sudipta Paria, Christopher Sozio, Andrew Lukefahr, and Swarup Bhunia. Library-Attack: Reverse Engineering Approach for Evaluating Hardware IP Protection. arXiv preprint arXiv:2501.12292, 2025.
- [DWLX21] Shijin Duan, Wenhao Wang, Yukui Luo, and Xiaolin Xu. A Survey of Recent Attacks and Mitigation on FPGA Systems. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 284–289, July 2021.
- [GRK⁺23] Rui Guo, M Sazadur Rahman, Hadi M Kamali, Fahim Rahman, Farimah Farahmandi, and Mark Tehranipoor. EvoLUTe: Evaluation of Look-Up-Table-based Fine-Grained IP Redaction. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2023.
- [HSD⁺23] Zhaokun Han, Mohammed Shayan, Aneesh Dixit, Mustafa Shihab, Yiorgos Makris, and Jeyavijayan (JV) Rajendran. FuncTeller: How Well Does eFPGA Hide Functionality? In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5809–5826, Anaheim, CA, August 2023. USENIX Association.
- [KAFT22] Hadi Mardani Kamali, Kimia Zamiri Azar, Farimah Farahmandi, and Mark Tehranipoor. Advances in Logic Locking: Past, Present, and Prospects. Cryptology ePrint Archive, Paper 2022/260, 2022.
- [KAFT23] Hadi M Kamali, Kimia Z Azar, Farimah Farahmandi, and Mark Tehranipoor. SheLL: Shrinking eFPGA Fabrics for Logic Locking. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2023.
- [KAHS19] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. Full-Lock: Hard Distributions of SAT Instances for Obfuscating Circuits Using Fully Configurable Logic and Routing Blocks. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC '19*, New York, NY, USA, 2019. Association for Computing Machinery.
- [KAV04] Dorothy Kucar, Shawki Areibi, and Anthony Vannelli. Hypergraph partitioning techniques. *Dynamics of Continuous Discrete and Impulsive Systems Series A*, 11:339–368, 2004.
- [KDH⁺21] Dirk Koch, Nguyen Dao, Bea Healy, Jing Yu, and Andrew Attwood. FABulous: An Embedded FPGA Framework. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '21*, page 45–56, New York, NY, USA, 2021. Association for Computing Machinery.
- [KPR⁺19] Gaurav Kolhe, Sai Manoj PD, Setareh Rafatirad, Hamid Mahmoodi, Avesta Sasan, and Houman Homayoun. On Custom LUT-based Obfuscation. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI, GLSVLSI '19*, page 477–482, New York, NY, USA, 2019. Association for Computing Machinery.
- [LSM⁺19] Meng Li, Kaveh Shamsi, Travis Meade, Zheng Zhao, Bei Yu, Yier Jin, and David Z. Pan. Provably Secure Camouflaging Strategy for IC Protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(8):1399–1412, 2019.
- [MAS⁺21] Prashanth Mohan, Oguz Atli, Joseph Sweeney, Onur Kibar, Larry Pileggi, and Ken Mai. Hardware Redaction via Designer-Directed Fine-Grained eFPGA Insertion. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1186–1191, 2021.

- [MCB07] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. Improvements to Technology Mapping for LUT-Based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):240–253, 2007.
- [MKZAG⁺18] Hadi Mardani Kamali, Kimia Zamiri Azar, Kris Gaj, Houman Homayoun, and Avesta Sasan. LUT-Lock: A Novel LUT-Based Logic Obfuscation for FPGA-Bitstream and ASIC-Hardware Protection. In *IEEE Computer Society Annual Symposium on VLSI*, pages 405–410, 2018.
- [RAD⁺24] Md Moshir Rahman, Rasheed Almazan, Aritra Dasgupta, Sudipta Paria, and Swarup Bhunia. United We Protect: Protecting IP Confidentiality with Integrated Transformation and Redaction. In *2024 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pages 1–7, 2024.
- [RAM22] Amin Rezaei, Raheel Afsharmazayejani, and Jordan Maynard. Evaluating the Security of eFPGA-Based Redaction Algorithms. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD ’22*, New York, NY, USA, 2022. Association for Computing Machinery.
- [RKM08] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. Epic: Ending piracy of integrated circuits. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1069–1074, 2008.
- [RSK13] Jeyavijayan (Jv) Rajendran, Ozgur Sinanoglu, and Ramesh Karri. Is split manufacturing secure? In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE ’13*, page 1259–1264, San Jose, CA, USA, 2013. EDA Consortium.
- [RTR⁺20] M Tanjidur Rahman, Shahin Tajik, M Sazadur Rahman, Mark Tehranipoor, and Navid Asadizanjani. The Key is Left under the Mat: On the Inappropriate Security Assumption of Logic Locking Schemes. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 262–272, Dec 2020.
- [SLM⁺17] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z. Pan, and Yier Jin. AppSAT: Approximately deobfuscating integrated circuits. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 95–100, 2017.
- [SMS23] Chaitali G. Sathe, Yiorgos Makris, and Benjamin Carrion Schafer. MANTIS: Machine Learning-Based Approximate ModeliNg of RedacTed Integrated CircuitS. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, April 2023.
- [SPJ19] Kaveh Shamsi, David Z. Pan, and Yier Jin. IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7, 2019.
- [SRM15] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143. IEEE, 2015.
- [TC⁺22] Chiara Muscari Tomajoli, Luca Collini, et al. ALICE: An Automatic Design Flow for eFPGA Redaction. In *DAC*, page 781–786. ACM, 2022.

- [TGA⁺19] Xifan Tang, Edouard Giacomin, Aurélien Alacchi, Baudouin Chauviere, and Pierre-Emmanuel Gaillardon. OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 367–374, Sep. 2019.
- [TLSB17] Shahin Tajik, Heiko Lohrke, Jean-Pierre Seifert, and Christian Boit. On the Power of Optical Contactless Probing: Attacking Bitstream Encryption of FPGAs. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, page 1661–1674, New York, NY, USA, 2017. Association for Computing Machinery.
- [XS19] Yang Xie and Ankur Srivastava. Anti-SAT: Mitigating SAT Attack on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(2):199–207, Feb 2019.
- [YMSR20] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Removal Attacks on Logic Locking and Camouflaging Techniques. *IEEE Transactions on Emerging Topics in Computing*, 8(2):517–532, 2020.
- [YSN⁺17] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan (JV) Rajendran, and Ozgur Sinanoglu. *Provably-Secure Logic Locking: From Theory To Practice*, page 1601–1618. Association for Computing Machinery, New York, NY, USA, 2017.